Osborne McGraw-Hill

80386/80286

William H. Murray III.
Chris H. Pappas

A 80386/80286-OS PROCESSZOR ASSEMBLY NYELVŰ PROGRAMOZÁSA

NOVOTRADE

William H. Murray III.
Chris H. Pappas

A 80386/80286-OS PROCESSZOR ASSEMBLY NYELVŰ PROGRAMOZÁSA

McGraw-Hill–Novotrade Rt.

A könyv eredeti címe: 80386/80286 assembly language programming (1986)

Fordította: K. Papp lászlóné
Lektorálta: Fellner László, Nagy Ákos
Szerkesztette: Stankovics János

A kiadásért felel RÉNYI GÁBOR, a Novotrade Rt. vezérigazgatója
Budapest, 1988

Müszaki szerkesztő: Erdősi Zoltán

ISBN 963 02 5558 8

Tartalomjegyzék

BEVEZETÉS

A könyv megírásakor három célunk volt:

1. bevezetni az Olvasót a hatékony gépi kódú programozás világába, felhasználva a 80386/80286-os processzort és a 80387/80287-es társprocesszort;
2. megtanítani az egyszerű, de ugyanakkor hatékony assembly programozást;
3. referencia kézikönyvet készíteni, amely a programozási utasításokon kívül mintaprogramokat is tartalmaz.

A 80386/80286-os assembly nyelv a 8088/8086-os nyelv továbbfejlesztett változata. A könyvben található programok nagy része a 8088/8086-os gépcsaládon futtatható, mivel a 80xxx Intel processzorok egymással felülről kompatibilisek. Az alapvető utasítások mind a korábbi, mind a későbbi változatokban megegyeznek. Az újabban kifejlesztett DOS verziók azonban magasabb szintű programozási technikákat is lehetővé tesznek (mint pl. a védelmi jellegű programozás). A távolság a régebbi és az újabb DOS változatok között egyre növekszik.

Az assembly nyelv a programozó számára lehetővé teszi a számítógép feletti teljes uralmat. A PASCAL, FORTRAN és más magas szintű nyelveknél a programozó kiszolgáltatott a fordítóprogram készítőjének. Ha pl. a játékadapter lekérdezésére van szükségünk, és ez az adott magas szintű nyelven megoldható, akkor minden rendben. De mit tehetünk ellenkező esetben? Az assembly nyelvben lehetőségünk van saját rutinok írására, vagy kihasználhatjuk a számítógép gyártói által szolgáltatott hatékony BIOS (Basic Input Output System) rendszerrutinok előnyeit is. Valójában az assembly programok többsége a magas szintű nyelvek kiegészítését szolgálja olyan esetekben, amelyekről az eredeti szerzők nem gondoskodnak. Ez a könyv az assembly programozást példákon keresztül mutatja be és oktatja. A programok listái minden szükséges fejrészt tartalmaznak, tehát ha a programkódokat pontosan a listáknak megfelelően beírjuk, a programok futtathatók. Nem mutatunk be részlistákat vagy rövidített példákat csak azért, hogy helyet takarítsunk meg. Minden törekvésünk arra irányult, hogy a programokat bemutassuk, elmagyarázzuk, és dokumentáljuk annak érdekében, hogy az Olvasó, akár kezdő, akár gyakorlott, az egyes utasításokat és programozási stílust minél jobban megértse.

1. BEVEZETÉS AZ ASSEMBLY NYELVBE

Az assembly programozási nyelv elsajátítása egyike azoknak a kihívásoknak, amivel egy szoftverfejlesztő szakembernek előbb-utóbb meg kell birkóznia. Az assembly programozókra a perifériák és a memória kezelése, a programkódok hatékonyságának és sebességének növelése és ehhez hasonló feladatok várnak. Az assembly nyelv nagy gyakorlatot és a részletekre való odafigyelést kíván. Hacsak nem dolgozunk a szoftverfejlesztés egy speciális területén, mint pl. egy processzor tervezése, az assembly nyelv a második leghatékonyabb programozási eszközünk lehet. Mivel az assembly nyelv a regiszterekhez közvetlen elérést nyújt, és a bitműveletek különleges világát nyitja meg a számunkra, sok esetben egyedül így tudunk csak megoldani olyan problémákat, amelyek a magas szintű nyelvek hatáskörén már kívül esnek. Az assemblerrel gyorsan futó programok állíthatók elő, mert kimarad az interpreter lépés (mint a BASIC-nél) vagy a fordítási lépés (mint a PASCAL-nál, C-nél vagy PL/1-nél).

Természetesen a gyorsaságnak ára van. Az assembly programozóknak nagyon figyelni kell a részletekre. Az assembly program a fordítóprogram ellenőrzése nélkül, a saját nyelvén irányítja a mikroprocesszort. Az assembler által szolgáltatott alapvető műveletek adatmozgatási feladatokat, egyszerű összeadásokat és logikai műveleteket tesznek csak lehetővé, ezért a legtöbb nagyméretű programot nem assemblerben írják meg. (Kivéve, ha valamilyen szoftverfejlesztés a cél, amelynek eredményeként keletkező gyors és hatékony alkalmazói program megírásához szükséges többletköltség időben és pénzben rendelkezésre áll.)

Ha a számítógéppel a saját nyelvén szeretnénk beszélni, akkor sok olyan jelenséggel kell megismerkednünk, amelyekkel a magas szintű nyelvek használata során még biztosan nem találkoztunk.

- Az assembly programozónak a memória szegmentálását figyelembe kell vennie.
- Amikor a memória elérésének közvetlen irányításáról van szó, gyors döntésekre van szükség.
- Hol kell tárolnunk a memóriában a programot és hol az adatokat?
- Szükség van-e veremre?
- Ha igen, a memória melyik részében helyezzük el?
- A döntéseket minden adattípus esetében a méret (byte, szó, duplaszó, négyesszó, tízbyte) és a típus [karakterlánc, BCD (binárisan kódolt decimális), statikus, dinamikus, előjeles, előjel nélküli, valós, lebegőpontos] figyelembevételével kell meghozni.
- Az előforduló adatelemek dupla indexelésű memóriaváltozók?
- Rendelkezésre áll-e a 80287/80387-es matematikai társpocesszor?
- A memóriaváltozókat át kell-e alakítani a 80287/80387-es adatformátumra?
- Melyik regisztereket kell használni?
- Mely portokon keresztül érhető el a színes monitor, a monokrom monitor és a nyomtató?

A kérdések közül a legtöbb csak az assembly programozással kapcsolatban vetődik fel.

A legnagyobb előnyök: a gyorsaság és a közvetlen vezérlési lehetőség.

Az assembly programok egyik fő előnye a programkódok végre hajtásának gyorsasága. A programkód ebben az esetben egy ún. mnemonik (az aktuális bináris gépi kódú utasítás rövidített szimbólikus emlékeztető jelölése), amelyből létrejön a számítógép által mikroszekundumok alatt

végrehajtott gépi kódú utasítás. Az interpreterek (mint pl. a BASIC-nél használatos interpreter) megvizsgálják a forrásprogramot, és sorról-sorra lefordítják gépi kódra, felhasználva a beépített és előre definiált szubrutinokat.

Valahányszor ezeket a programokat futtatjuk, a fordítási eljárás és a hibaellenőrzés ismételten megtörténik, ezért ezeknek a programoknak a futása meglehetősen lassú. A FORTRAN-nál, a PASCAL-nál vagy a PL/1-nél használatos fordítóprogramok beolvassák a forrásprogramot, és műveletkódok sorozatává alakítják, amelyeket a mikroprocesszor már közvetlenül végre tud hajtani. Az ilyen módon lefordított program gyorsabb, mint az interpretált kód, mert ebben az esetben a fordítás csak egyszer történik meg. Azonban ennek a folyamatnak is van egy hátránya: a fordítóprogramoknak az utasítások széles skáláját értelmezniük kell. Ez a feladat azonban még a legkisebb programoknál is nagyméretű programfejet igényel.

Például egy lefordított BASIC program, ami csak néhány karaktert jelenít meg a képernyőn, akár 33 kbyte-ot is elfoglalhat a számítógép memóriájából. Ha ugyanezt a programot assemblyben írjuk meg, az előbbi hely 1/3-ára, kb. 10 kbyte-ra van csak szükségünk.

Most pedig beszéljünk az assembly programok másik fő előnyéről. A felhasználó az assembly nyelv nélkül a magas szintű nyelvek által kínált lehetőségekre kell hogy hagyatkozzon. Például ha valaki egy olyan felhasználói programot írt, amely kritikus adatokat kezel és a felhasználó egy közepes képzettségű adatrögzítő, akkor hasznos lehet, ha letiltjuk a CTRL-ALT-DEL gombok lenyomását követő adatmegsemmisítő funkciót. Legtöbb esetben ez a feladat csak az assembly nyelven valósítható meg.

Az assembly nyelv a szoftverfejlesztő szakemberek számára lehetővé teszi a kapcsolatteremtést az operációs rendszerrel és közvetlen vezérlést biztosít a monitorra, a nyomtatóra és a merev, ill. hajlékony lemezes tárolóegységre. Az alkalmazói programok készítőinek az operációs rendszerrel gyakran közvetlenül kapcsolatba kell lépniük. Ezek a feladatok szinte kizárólag csak assembly nyelven oldhatók meg.

1.1 A 80286/80386-OS PROCESSZOROK CSALÁDFÁJA

A családfák keletkezése rendszerint több évszázadra vezethető vissza. A mikroprocesszorok családfájának kora azonban csak néhány évtizedben mérhető, és ha a fejlődés továbbra is ilyen sebességű marad, akkor szinte minden évben egy-egy új levelet kell a családfához rajzolnunk. Az első IC-t (integrált áramkört) az 1960-as évek elején készítették. Az IC-k bevezetése jelentősen csökkentette a kondenzátorok, diódák és tranzisztorok méretét, mivel valamennyi ilyen elem egy egyszerű szilícium szeletre került. Egy évtizeddel később az Intel cég kifejlesztette az első 8 bites chip-jét, a 8008-ast. 1974-ben a mikropocesszorok második generációját a 8080-ast hozták létre, ami általános igényeket tudott kielégíteni. Ebben az időben került piacra – a kialakult verseny eredményeként – pl. a Zilog Z80-as processzor is.

A mikroprcesszorok harmadik generációja 4 évvel később, 1978-ban jelent meg, amikor az Intel a 8086-os típust kifejlesztette. Bár ez a chip bizonyos mértékben felülről kompatibilis az elődjével – a 8080-as típussal –, sokkal fejlettebb annál, és számos új tulajdonsága van. Az Intel cég a 8088-as típust is kidolgozta, ami a 8086-os típusnak valamivel egyszerűbb tervezésű változata, és kompatibilis számos I/O (input/output) eszközzel. A 8088-as processzort olyan jól sikerült elkészíteni, hogy az IBM cég valamennyi elsőgenerációs személyi számítógépébe ezt a processzort építette bele.

Szinte ezzel egyidőben jelent meg a piacon a 8088/8086-os processzorok unokatestvére, a 8087-es matematikai társpocesszor is, amely a matematikai számításokat nagy pontossággal és nagy sebességgel képes elvégezni.

Az Intel cég azonban nem elégedett meg az eddigi sikereivel. 1984 közepén ui. megjelent a 80286-os mikroprocesszorral, ami a CPU-k új generációját jelentette. A 80286-os a korábbi típusokhoz képest olyan előnyökkel rendelkezik, mint a védelmi mechanizmusok, taszk-kezelés, memóriagazdálkodás és virtuális memóriatámogatás.

Valamennyi most felsorolt hatékony jellemzőt egyetlen VLSI chip tartalmazza.

A 80286-os processzor a 8088/8086-os típusokkal felülről kompatibilis, mivel a címzési módok és az alapvető utasítások mindegyik processzornál megegyeznek. Az alapfelépítés támogatja a PASCAL-hoz, C-hez hasonló magas szintű nyelveket, mivel a regiszterkészlet tervezése olyan, ami a fordítóprogram által generált kódoknak megfelel.

A 80286-os processzor számos adattípust támogat, mint pl. a karakterláncot, a BCD-t, a lebegőpontos formátumot, valamint lehetővé teszi olyan összetett adatszerkezetek, mint a statikus vagy dinamikus tömbök, rekordok vagy rekordon belüli tömbök hatékony címzését. A 80286-os processzor memóriaszerkezete lehetővé teszi a memóriaszegmenses felosztást, amely által a moduláris programozás megvalósítható. A 80286-os processzor szinte minden igényt kielégítő nagyméretű címzési területet biztosít. A hasznos memóriaterület 16 Mbyte RAM vagy ROM. Ez a memóriaterület lehetővé teszi, hogy a mikroprocesszor a nagyméretű programokat és a hozzájuk tartozó adatokat egyidejűleg a memóriában tartsa, amely által az adatok és a programok igen gyorsan elérhetők. A többfelhasználós rendszerben előforduló dinamikusan változó memóriakövetelmények miatt a 80286-os processzor minden felhasználójának 1 Gbyte virtuális címzési helyet szolgáltat, ami azt jelenti, hogy a programok számára vagy méretére vonatkozó korábbi korlátozások szinte már nem is léteznek. Az Intel cég 80286-os processzorát úgy tervezték meg, hogy támogassa a több felhasználós, újra programozható és valós idejű többfeladatos alkalmazásokat. A mikroszámítógépes rendszerek ezideig általában négy, öt felhasználót tudtak egyidejűleg kiszolgálni. Ez a szám most a háromszorosára emelkedhet. A Intel cég családfájának legújabb tagja a 80386-os típus, ami 32 bites teljes külső adatbusszal rendelkezik (ez kétszerese a 80286-os típusénak) és 4 Gbyte memória megcímzésére képes.

1.2 MIT TANULHATUNK MEG EBBŐL A KÖNYVBŐL?

Az az Olvasó, aki figyelmesen elolvassa a szöveget, és kipróbálja a programokat, megismeri a 80286/80386-os processzorok belső szerkezetét, és az utasításkészletet használni tudja majd. Az assembly nyelv elsajátítása a számítógép működését is érthetővé teszi. Három assemblert mutatunk be részletesen – IBM, Microsoft, Speedware –, így az Olvasó kiválaszthatja az igényeinek megfelelő változatot. Mindenki megtanulhatja, hogy mi a pszeudo művelet és hogyan kell használni, valamint elsajátíthatja az eljárások és makrók megírásának módját is. A makro/szubrutin könyvtár felépítésének, továbbfejlesztésének és karbantartásának alapelvei a könyvben hangsúlyozottan szerepelnek. A grafikus és file-manipulációs műveleteket számos mintaprogram mutatja be. Megérthetjük a 80287/80387-es társprocesszorok használatát, és példákat láthatunk assembly rutinok magas szintű nyelvekhez való csatlakoztatására is. Megtudhatjuk, hogyan kell kapcsolatot teremteni az operációs rendszerrel a DOS és BIOS rutinokon keresztül. Elsajátíthatjuk a programok tesztelésének módját és a hibakeresést is, amelyhez számos technikát alkalmazunk majd (pl. IBM DEBUG és a Microsoft SYMDEB felhasználói programokat).

1.3 MILYEN ELŐZETES TUDÁST FELTÉTELEZ A KÖNYV?

Ez a könyv feltételezi, hogy az Olvasó már jártas valamilyen magas szintű nyelvben (BASIC, PASCAL). Hasznosnak bizonyulhat – de nem szükséges – a tizenhatos számrendszer, valamint a logikai és bitléptető műveletek (AND, NAND, OR, XOR) előzetes ismerete.

1.4 SZÁMRENDSZEREK

Ha meg akarjuk érteni a assembly programozást, először meg kell értenünk, hogyan tárolja a számítógép az információt a memóriában, és hogyan ábrázolja az adatokat. Mivel a számítógép elektromos eszköz, ezért csak a feszültségkülönbséget képes érzékelni. A mikroprocesszor két feszültségi szintet használ, amelyeket 0-val és 1-gyel jelölünk. A (0,1) számpár a számítástechnikában előforduló legkisebb egység, amit bitnek nevezünk.

Az emberek, mivel tíz ujjuk van, a decimális számrendszert használják. A számítógépek, mivel két elkülönülő állapotot tudnak megkülönböztetni, a kettes számrendszerben számolnak. Természetesen emiatt az emberek és a számítógépek között kommunikációs problémák keletkeznek. A nyelvi nehézségek kiküszöbölése a különböző szimbolikus kódok kifejlesztéséhez vezetett. Az ASCII-t (American Standard Code for Information Interchange) azért hozták létre, hogy kifejezzék az írógépek billentyűzetén található valamennyi szimbólumot. Számos numerikus kódot létesítettek annak érdekében, hogy a numerikus értékeket a számítógép részére érthető formában fejezzék ki. A kettes komplemens rendszer előjeles egész számok kifejezését teszi lehetővé. Az assemblyben előforduló egyik legfontosabb numerikus kód a hexadecimális kód, amely lehetővé teszi a bináris információ olvashatóságát, akár utasításról, memóriacímről vagy adatról van szó.

1.5 BINÁRIS SZÁMOK

A tizes (decimális) számrendszerben a számjegyek kifejezésére tíz különböző szimbólumot használunk 0-tól 9-ig. A neve innen származik. A bináris (kettes) rendszerben csak két szimbólum, a 0 és az 1 szerepel. Első látásra úgy tűnhet, hogy ezzel a korlátozott készlettel a nagy számok kifejezése meglehetősen nehézkes lesz. Hamarosan be fogjuk látni, hogy ez nem igaz, sőt a kettes számrendszer alapelvét már ismerjük is. Elsőként vegyük az 1024-es decimális számot. Az általános iskolában úgy tanultuk, hogy ez a szám négy darab 1-est, két darab 10-est, és egy darab 1000-est tartalmaz, 100-ast pedig nem. Ez az ún. súlyozott numerikus ábrázolás. Ahogy a számban képletesen balra mozdulunk el, az egyes értékek súlya nő. Az 1024-es szám vizsgálatára van egy másik módszer is. Mivel 10-es (decimális) rendszerben vagyunk, a számra a következőképpen is gondolhatunk:

```
1 0 2 4
  |   |   └──→ 4 * 10^0 =    4  (Legkisebb helyiértékű)
  |   └──────→ 2 * 10^1 =   20
  └──────────→ 0 * 10^2 =  000
              → 1 * 10^3 = 1000  (Legnagyobb helyiértékű)
```

A konverziót úgy végeztük el, hogy a számban szereplő minden egyes számjegyet az alap megfelelően hatványozott értékével, beszoroztuk. Ha a kisebb helyértékről a nagyobb helyérték felé haladunk, a hatványok növekszenek. Ugyanez a konverziós alapelv érvényesül a kettes számrendszerben is.

A 0101 bináris szám decimálissá alakítása során a következők szerint kell eljárnunk:

1 0 1 0

$0 * 2^0 = 0$ (Legkisebb helyiértékű)
$1 * 2^1 = 2$
$0 * 2^2 = 0$
$1 * 2^3 = 8$ (Legnagyobb helyiértékű)

Ebben az esetben a kettes alapot emeltük a megfelelő hatványra, kezdve a legkisebb helyértékű bittel (LSB), majd folytatva a nagyobb helyértékű bitekkel (MSB). Ha más alapú számrendszerből akarunk áttérni decimálisba, akkor is használhatjuk a most bemutatott módszert, azzal a különbséggel, hogy a kettes vagy a tízes helyett a megfelelő alapot használjuk. Most már tudjuk, hogyan kell a bináris számot decimálissá alakítani. A fordított eljárás során, amikor decimálisból térünk át binárisba, kettő hatványait kell az átalakítandó számból ismételten kivonnunk. Tegyük fel, hogy a 11-es decimális számot akarjuk átalakítani binárissá. A kivonást kezdjük a 2 számmal, ami – mint bizonyára tudjuk – 8-cal egyenlő. A kivonás eredményeképpen maradéknak 3 keletkezik, amiből a 2-t (4) nem tudjuk kivonni, csak a 2-t (2). Ha a kivonást elvégezzük, még mindig keletkezik maradék, az 1. Tehát ebből még 2-t kivonhatunk és ekkor a maradék már nulla lesz. A decimális 11 bináris megfelelője tehát 1011. Bemutatunk egy ennél egyszerűbb módszert is, ami a nagyobb méretű bináris számokra alkalmazható. Az eljárás ismételt osztások sorozatából áll, amelyben az osztó szerepét az a szám tölti be, amelyik az új számrendszer alapja, és az osztandó pedig az a szám, amit át akarunk alakítani. Példaként a decimális 50 bináris átalakítását végezzük el.

Osztandó	Osztó	Eredmény	Maradék
50	2	25	0
25	2	12	1
12	2	6	0
6	2	3	0
3	2	1	1
1	2	0	1

Végeredmény: 110010

Ezzel a módszerrel bármilyen decimális számot bármilyen alapra átalakíhatunk. Ha pl. a decimális 428-at akarjuk átalakítani hexadecimálissá, a 428 szerepeljen osztandóként és a 16 osztóként.

Bináris összeadás és kivonás

A bináris összeadás és kivonás azonos a decimális összeadással és kivonással, azzal a különbséggel, hogy most 2, és nem 10 hatványainál keletkezik átvitel.

Lássunk mindjárt két példát!

```
  25
+ 85
 110
```

Amikor a két 5-öst összeadjuk, átvitel keletkezik a következő helyértékre, vagy másképpen 1*10E1, ezért ezt a számot a következő oszlopban szereplő számokhoz hozzá kell adnunk. Ezután a 8-at és a 2-t valamint az átvitelt adjuk össze, és ekkor egy újabb átvitel keletkezik a 10E2 oszlopban. Ha leírjuk az új átvitelt, 110-et kapunk eredményként.

Most pedig a bináris összeadás következik. (Zárójelben a bináris szám decimális megfelelője szerepel.)

```
 1010 (10)
+0011  (3)
 1101 (12)
```

0+1 eredménye 1 átvitel nélkül. 1+1 eredménye kettő, ami binárisan 10, amelyben az 1 az átvitelt jelenti a következő legnagyobb helyértékű bitre, ami már 2-t jelent. Ha az átvitelt ide leírjuk (hozzáadva a 0+0-t), 1101 lesz az eredmény.

A következő példa a bináris számok összeadásának másik megoldását mutatja:

```
 0011 (3)
+0011 (3)
 0110 (6)
```

Az első két 1-es összeadásakor mindjárt egy átvitel keletkezik a 2-es oszloban, azonban ebben az oszlopban már eleve két 1-es szerepel, és most még az átvitel is, tehát ezt már a harmadik oszlopban kell elhelyezni.

De nézzük meg ezt a lépést részletesebben is!

```
  1
  1
+ 1
 11
```

1+1 bináris eredménye 10, és ha 10-hez még 1-et adunk, 11-et kapunk. (Generáltunk tehát egy átvitelt a következő nagyobb thelyértékű bitre, és 1-et kaptunk eredményként.)

A bináris kivonásnál ugyanazokat az alapelveket kell alkalmaznunk, mint a decimális kivonásnál. Az egyedüli különbség csak az, hogy amikor egy átvitel keletkezik a következő legnagyobb helyértékről, annak értéke a 2 – nem pedig a 10 – hatványozott értéke lesz.

```
 534
-251
 283
```

A decimális 4-ből minden nehézség nélkül ki tudjuk vonni a decimális 1-et, ahhoz azonban, hogy a 3-ból kivonjuk az 5-öt, a következő legnagyobb helyértékről egy áthozatalt kell generálnunk (1*10E2), hogy lehetővé váljon az 50 kivonása a 130-ból. Végül a 2-t kell a 4-ből kivonnunk, és már meg is kaptuk az eredményt.

Byte

Mivel a programjainkat számítógéppel hajtatjuk végre, és az adatokat is itt tároljuk, bizonyos mértékig a számítógép felépítése határozza meg az adatokra és a programkódokra vonatkozó formátumot és tartományt. A számítógépek a változó hosszúságú bináris számokat nem véletlenszerűen tárolják. A 80286-os processzor felépítésében a hardver memóriahely 8 egymás utáni bit sorozatából áll. A 8 bitből álló csoportot nevezzük byte-nak. A bitpozíciókat a legkisebb helyértékű bittől (0-s) kezdjük számozni a legnagyobb helyértékű bitig (7-es). Az 1–1. ábrán néhány egymás után következő memóriahelyet mutatunk be, amelyek mindegyike 1–1 byte értékű információt tárol. Ez az információ lehet a gép számára szóló utasítás, egy másik memóriahely címe, de numerikus vagy ASCII karakter is. (L. az ASCII karakterekre vonatkozó mellékletet.) A nyolc bit 256 önálló állapot kifejezésére képes, amelynek következtében egyetlen memóriahely a 0-tól 255-ös tartományba eső valamennyi pozitív egész szám bináris ábrázolását tudja elvégezni. A szó kifejezéssel egyetlen memóriahelyen tárolt bizonyos számú bitre tudunk utalni. A korábbi számítógép felépítésekben egy szó 4 bitet jelentett. Néhány számítógép egy memóriahelyen 64 bitet képes tárolni, tehát egy szó ebben az esetben 64 bitet jelent. A 8 bitből álló byte két 4 bites csoportra osztható. Egy 4 bites csoport egy hexadecimális szám kifejezésére képes.

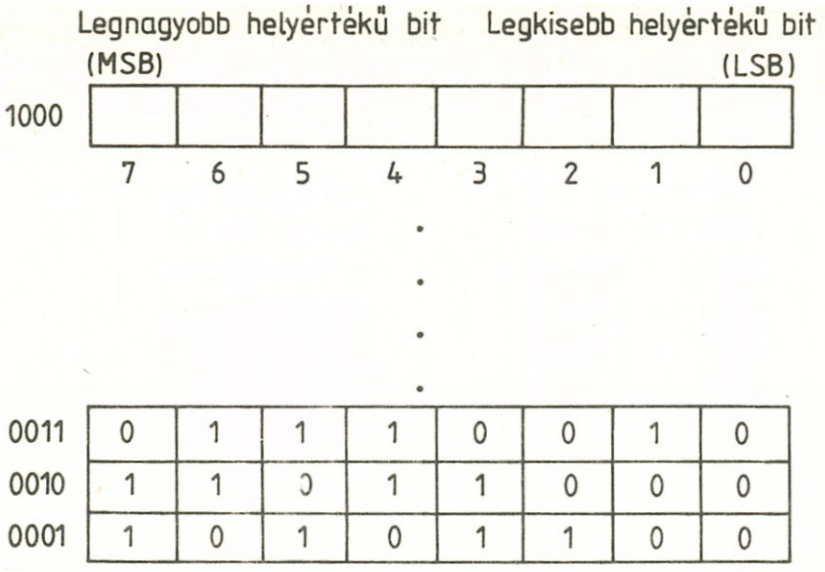
Karakterek

A byte-ban elhelyezkedő 8 bit nem mindig fejez ki numerikus értéket. Az ASCII egy 7-bites kódrendszer, amelynek segíségével az alfabetikus és numerikus karakterek kifejezése lehetséges. Ez a kódrendszer minden betűhöz, számjegyhez, az írógépeken általában szereplő speciális karaterhez, valamint bizonyos speciális vezérlőkódokhoz tetszőlegesen kiválasztott bináris számokat rendel.

A 7 bit-es kód 128 önálló szimbólum és kód kifejezésére képes. A 8 bitet a rendszer néha az adatátvitel és a hibafelderítés kódjaként használja. Vannak olyan cégek is, amelyek a 8. bit segítségével kibővített karakterkészletet hoznak létre. Egy további bit hozzáadása ui. az önállóan kifejezhető szimbólumok számát 256-ra növeli. Ezért lehetővé válik az angol nyelvben nem szereplő speciális nyelvi, matematikai és grafikus szimbólumok kifejezése.

Előjeles számok

Vizsgáljuk meg közelebbről az egész számok ábrázolását. Ha mind a 8 bitet felhasználjuk a pozitív egész szám kifejezésére, akkor 0-tól (00000000) 255-ig (11111111) tudjuk a számokat kifejezni. Ha a 255-höz 1-et adunk, akkor a teljes ciklus végéhez érkezünk, vagyis vissza a 0-hoz. Ha pozitív és negatív egész számokat akarunk kifejezni, akkor bizonyos kompromisszumot kell kötnünk, mert a 8 bit terjedelemben ki kell fejeznünk a szám előjelét is. Ennek egyedüli lehetséges módja az, hogy egy bitpoziciót az előjel kifejezésére tartunk fenn, így tehát csak 7 bit marad a numerikus érték ábrázolására, vagyis 0-tól (0000000) 127-ig (1111111) tudjuk a számokat kifejezni.



1–1. ábra Egymás utáni memóriahelyek

(Mivel bináris számokról van szó, a bitek számának 1-gyel való csökkentése a lehetséges állapotok tartományának felezését jelenti.) Az előjeles adatábrázoláskor a legnagyobb helyértékű bit tartalmazza az előjelet. Ha ez a bit 0, akkor pozitív, ha 1, akkor negatív számról van szó. pl.:

0000 0101 +5-öt jelent,
1000 0101 −5-öt jelent.

Mi a helyzet a 0-val?
Elméletileg a következőket írhatjuk:

0000 0000 +0-át jelent,
1000 0000 −0-át jelent.

Ehhez az adatábrázolási formához új szabályok és aritmetika szükségesek, amelyek közül az egyik azt határozza meg, hogy a 0 numerikus értékét mindig pozitívnak kell ábrázolni: 0000 0000. Mivel a számok értékének kifejezésére csak 7 bit áll rendelkezésünkre, csak a 0-ról, ill. a plusz és mínusz 127-es tartományba eső számokról beszélhetünk.

```
  0000 0000    +0
− 0000 0001    +1
  1111 1111    −127
```

A példa eredménye helytelen.

Ha előjeles számokkal akarunk aritmetikai műveleteket elvégezni, akkor az 1111 1111 számnak a −1-et kell jelentenie. A következő példa a kivonásnak egy másik módját mutatja be:

```
  1111 1110    −2
− 0000 0010    +2
  1111 1100    −4
```

Ha az 1111 1100 bináris szám decimális átalakítását kellene elvégeznünk, feltételezve azt, hogy a legnagyobb helyértékű bit az előjelet jelenti, 124-et kapnánk. A legnagyobb helyértékű bit 1, ami azt jelenti, hogy negatív számról van szó, tehát a szám −124. De miért lesz az eredmény −4? Erre a kérdésre a következő részben kapjuk meg a választ.

Számok ábrázolása kettes komplemens kódban

Ahhoz, hogy előjeles számokkal az összeadást és kivonást megfelelően el lehessen végezni, a számokat az ún. kettes komplemenses formában kell kifejeznünk:

0000 0100	+4
0000 0011	+3
0000 0010	+2
0000 0001	+1
0000 0000	0
1111 1111	−1
1111 1110	−2
1111 1101	−3
1111 1100	−4

Vegyük észre, hogy a legnagyobb helyértékű bit az előjel kifejezését szolgálja. Amikor a kettes komplemenses számokat összeadjuk vagy kivonjuk, soha ne feledkezzünk meg arról, hogy az eredmény is kettes komplemensben értendő. Pl.:

0000 0100	+4	
+1111 1101	−3	
1)0000 0001	+1	(A keletkezett átvitelt figyelmen kívül hagyjuk.)

Ha egy negatív érték kettes komplemens megfelelőjét akarjuk előállítani, minden bitet az ellenkezőjére kell beállítanunk, és ezután 1-et hozzáadnunk. A következő példa azt mutatja, hogyan kell a +5 előjelét megváltoztatnunk.

0000 0101	+5
1111 1010	(A bitek komplemensének előállítása.)
+0000 0001	(A +1 hozzáadása.)
1111 1011	−5

Az ellenkező irányú átalakítás is ugyanilyen könnyű. Állítsuk elő pl. a −4 pozitív megfelelőjét!

1111 1100	−4
0000 0011	(A bitek komplemensének előállítása.)
+0000 0001	(A +1 hozzáadása.)
0000 0100	+4

Végül egy utolsó példa:

1111 1001	−7	(Kettes komplemens jelölés.)
+1111 1000	−8	(Kettes komplemens jelölés.)
1111 0001	−15	(Kettes komplemens jelölés.)

1.6 ELŐJEL-KITERJESZTÉS

Mivel a 80286-os processzor belső adatregiszterei 16 bit szervezésűek, egy memóriahely pedig 1 byte, valószínűleg előfordul olyan eset, hogy két különböző méretű számot kell összeadnunk. Tegyük fel, hogy egy 8 bit terjedelmű előjelezett kettes komplemensű számot akarunk hozzáadni az ugyanilyen formátumú 16 bites számhoz.

```
          0000 0101      +5
+1111 1111 1111 1101      -3
```

Az előbbi példánál a +5 legnagyobb helyértékű bitjeihez 0000 0000-t kell hozzáfűznünk. De mit tegyünk akkor, ha egy 8 bites negatív számról van szó?

```
          1111 1111      -1
+0000 0000 0000 0100      +4
```

Mivel a 8 bites érték negatív volt, ezért most nyolc 1-est kell a számhoz hozzáfűznünk. Az előjel-kiterjesztés általános szabálya tehát a következő: A legnagyobb helyértékű bitekhez nyolc 0-t vagy nyolc 1-est kell hozzáfűzni a szám előjelétől függően. A korábbi két 8 bites szám kibővített értékei tehát a következők:

```
0000 0000 0000 0101    (A +5 16 bites ábrázolása.)
1111 1111 1111 1111    (A -1 16 bites ábrázolása.)
```

1.7 HEXADECIMÁLIS SZÁMOK

A hexadecimális számredszer elnevezése onnan származik, hogy a számkészlete 16 különböző szimbólumból áll. A hexadecimális számok nagyon hasznosak assembly kódok és adatok olvasásakor és írásakor, valamint a memória tárkiírásakor (dump), mert egy hexadecimális szimbólummal egy 4 bites bináris csoportot lehet kifejezni.

A hexadecimális számjegyek a következők: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Az 1–1. táblázat a decimális, bináris és hexadecimális kódok közötti összefüggéseket mutat be. A következő két rövid példa a hexadecimális számok használatát illusztrálja. Ha a programozónak néhány egymást követő memóriahely tartalmát kell kiíratnia, a megjelenő lista bináris számokkal a következő lenne:

```
1001 0110
1110 0011
1010 1011
0010 1100
```

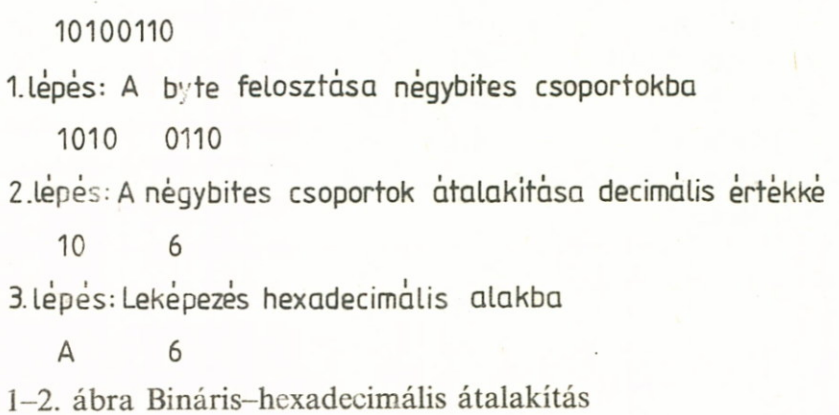
Ugyanez az információ hexadecimális rendszerben a következő:

96
E3
AB
2C

A második listát sokkal könyebb nyomon követni és ellenőrizni. A bináris számok átalakítása hexadecimálissá nagyon egyszerű művelet. A legkisebb helyértékű bittel kezdve 4 bites csoportokat kell kialakítanunk, majd ezután minden négyes csoporthoz hozzá kell rendelnünk a hexadecimális megfelelőjét. Az 1–2. ábrán ez az eljárás látható. A hexadecimálisból binárisba való átalakításkor is ugyanezt a módszert kell alkalmaznunk.

Decimális szám	Bináris szám	Hexadecimális szám
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

1–1. táblázat Ugyanazon számok decimális, bináris és hexadecimális alakja



1–2. ábra Bináris–hexadecimális átalakítás

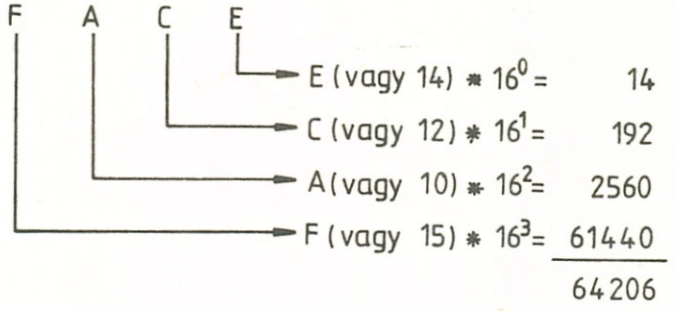
Az 1–3 ábrán az FACE szám hexadecimális-decimális átalakítását láthatjuk. A FACE (64206) bináris megfelelője a következő:

1111 1010 1100 1110

Ha két memóriahelyen tároljuk, akkor pedig:

11111010 11001110

Ha az előbbi bináris számot hexadecimálissá alakítjuk, akkor a következő értéket kapjuk: FA CE.



1–3. ábra Hexadecimális–decimális átalakítás

1.8 TOVÁBBI BITCSOPORTOSÍTÁSOK

A 8 bitből álló byte nem minden esetben alkalmas a különböző assembly feladatok megoldására. A 80286/80386-os processzorral folytatott munka során a bitek más típusú szervezésére is szükség van.

Szavak

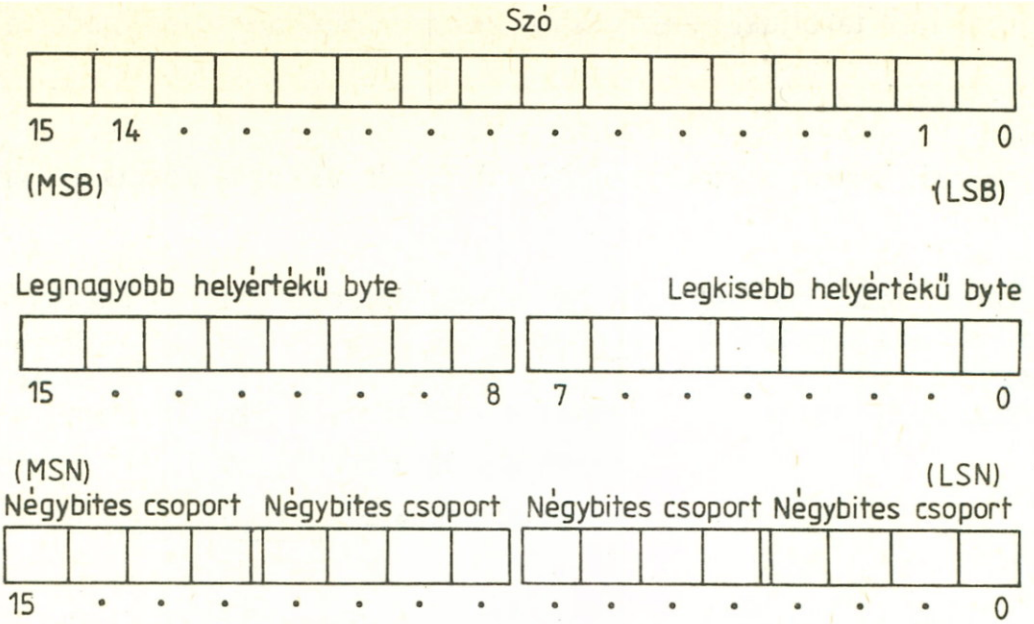
A 80286-os 16 bites mikroprocesszor, a leghatékonyabban 16 bites értékekkel tud dolgozni.

Az 1–4. ábrán látható, hogy egy 16 bites szó két byte-ból építhető fel. Ez egyben azt is jelenti, hogy a szóban kifejezhető maximális pozitív egész érték 255-ről 65 535-re nő. A karakterlánc és egész változók memóriahelyeinek nyomon követése 16 bites címmutatók segítségével történik 0000H-tól FFFFH-ig. (A H betű arra utal, hogy hexadecimális számról van szó.) (Fordítói megjegyzés: Az FFFFH szám 64 kbyte-ot jelent, ami 64 * 1024, azaz 64 * 2 bit.) Az egész típusu szó tárolásakor a rendszer a két byte-ot megcseréli (swap), vagyis az alsó byte kerül előre, a felső byte pedig a következő memóriacímre.

Azt kell tehát megjegyeznünk, hogy az alacsonyabb sorrendű byte kerül a kisebb memóriacímre és a magasabb sorrendü byte a magasabb memóriacímre. Például ha a 3456H értéket tároljuk a memóriában, a következőket fogjuk tapasztalni:

Memóriacím	Érték
0000 0001	34H
0000 0000	56H

Legtöbb esetben a assemblyben programozó szakembernek nem kell törődnie ezzel a címzési/tárolási rendszerrel, ui. valamennyi memóriára vonatkozó utasítás megérti ezt a tárolási formátumot, és elvégzi a szükséges átalakításokat. A programozónak azonban nem szabad



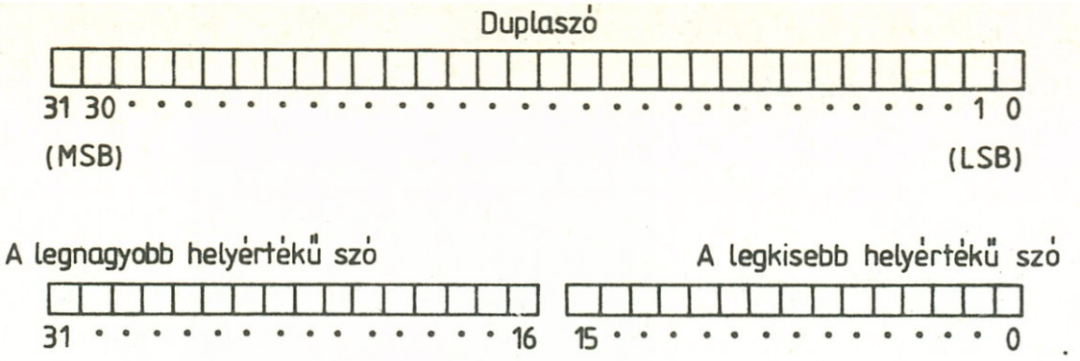
1–4. ábra A szó bitjeinek négybites csoportjai és byte-jai

megfeledkeznie erről a jelenségről akkor, amikor tárkiírás után a memória tartalmát akarja nyomon követni. (Tárkiíratő programok például a CodeSmith-86, a Trace-86 vagy a Disk-Look.) A mikroprocesszor mindig 16 bites szavakat, nem pedig két 8 bites byte-ot vizsgál.

Duplaszavak

A duplaszó pontosan azt jelenti, mint amire az olvasásakor gondolunk: két szó. A duplaszavak 32 bit szélesek, mint ahogy ezt az 1–5. ábra is mutatja, és két egymást követő szóból állnak. Ez egy nagyon fontos adatformátum, mert ezáltal a 80286-os processzor a memória több millió byte-ját tudja elérni. Ez egyben azt is jelenti, hogy a memória adatait 32 bites mutatókkal fogjuk kijelölni. A 32 bites duplaszó használata által az aritmetikai műveletek végrehajtása is pontosabb. A 32 bites adatábrázolás mind az egész, mind a lebegőpontos számok körében igen kicsi és igen nagy számok kifejezésére alkalmas. A duplaszavakat a szavakhoz hasonló elrendezésben tárolja a rendszer. Ebben az esetben a kisebb helyértékű szó kerül az alacsonyabb memóriacímre, a nagyobb helyértékű pedig a magasabb memóriahelyre. A 32 bites duplaszót négy byte sorozataként tárolja a rendszer a legkisebb helyértékű byte-tal kezdve. Az 12345678H számot a következőképpen képzelhetjük el:

Memóriacím	Érték		
0000 0011	12 (MSB)	16 bites szó	Duplaszó
0000 0010	34		
0000 0001	56	16 bites szó	
0000 0000	78 (LSB)		



1–5. ábra A duplaszó szavai és byte-jai

Négyesszavak

Ha a duplaszó mem adja azt a numerikus pontosságot, mint amire szükségünk van, akkor a négyesszó bizonyára megoldja ezt a problémát is. A négyesszó négy szóból áll, így igen nagy számok vagy hosszú karakterláncok tárolására használhatjuk. A tárolási séma, ami az 1–6. ábrán látható, teljesen összhangban van az eddig megismert valamennyi esettel. A legkisebb helyértékű szó a legalacsonyabb című memóriahelyen van, míg a legnagyobb helyértékű szó a legmagasabb című memóriahelyre kerül. Az 1234567890ABCDEF szám tárolását az 1–6. ábra mutatja.

Memóriacím	Érték	
0000 0111	12H (MSB)	32 bites duplaszó
0000 0110	34H	
0000 0101	56H	
0000 0100	78H	
0000 0011	90H	32 bites duplaszó
0000 0010	ABH	
0000 0001	CDH	
0000 0000	EFH (LSB)	

1–6. ábra A négyesszó a memóriában

Tízbyte-ok

A tízbyte a duplaszóhoz hasonlóan pontosan azt jelenti, mint amire az elolvasásakor gondolunk. Ez egy 80 bites érték, amit rendkívül nagy szám, vagy igen sok karakter tárolásra használhatunk. Ez a 80286-os processzor esetében a legnagyobb definiálható adattípus, amelynek tárolási sémája azonos a dupla- és négyesszónál megismerttel. A legkisebb helyértékű byte a legalacsonyabb című, a legnagyobb helyértékű byte a legmagasabb című következő memóriahelyen kerül tárolásra.

1.9 A 80386-OS MIKROPROCESSZOR ADATTÍPUSAI

A 8088-as, 8086-os és 80286-os mikroprocesszor által támogatott adattípusokon kívül a 80386-os felépítés támogatja a 32 bites előjeles és előjel nélküli egész számokat, valamint a 32 bites mezőket. Ebben az esetben 32 bites eltolási értékekkel és 48 bites (teljes) mutatókkal dolgozhatunk. Két új változótípust, a DBIT (egy bit)-et és a DP (48 bit)-et is bevezették, hogy ezáltal segítsék a 80386-os adattípusok támogatását. Mindezekről részletesen a 6. fejezetben lesz szó.

1.10 A SZABVÁNYTÓL ELTÉRŐ BITMEZŐK

Az assemblyben kezelhető legtöbb adat az előbbiekben definiált valamelyik adattípus kategóriájába tartozik, vannak azonban kivételek. Például a PC-DOS tervezői – annak érdekében, hogy ne veszítsenek helyet a memória hasznosításából – elhatározták, hogy a dátum tárolását egy teljesen önálló konfigurációban végzik el. Ha megvizsgáljuk az év, hónap és nap mezők által valójában elfoglalt helyet, azt figyelhetjük meg, hogy a dátum kifejezése egyetlen 16 bites szóban történik, amelyből 7 bit az évre vonatkozó mező. Elméletileg ez az elrendezés 0-tól 127-ig képes a számok ábrázolására. Valójában az előbbi tartományt 0-tól 119-ig korlátozták, és ezt az értéket adják az 1980-as kiinduló alaphoz. A 12 hónap kifejezésére 4 bit elegendő. A napok maximális száma 31, melynek ábrázolásához 5 bit szükséges. Egy másik szabványtól eltérő bitfelosztás a számítógép grafikus alkalmazásakor kerül elő. Ebben az esetben ui. egy önálló bit a monitor képernyőjén egy pontnak felel meg. A zérus bitérték a hozzá tartozó képernyőpont kikapcsolt, az 1 pedig a pont bekapcsolt állapotát jelenti. A monitoron jelentkező pontmező rendszerről rendszerre változhat. Ha egy bizonyos képernyőpont színét akarjuk kifejezni, ehhez 2 bitet kell felhasználnunk. (A három alapszín ki- és bekapcsolása.) A most megismert szabványtól eltérő bitkonfigurációkról ne feledkezzünk meg akkor, amikor bináris adatokat tárolunk, alkalmazunk, vagy műveleteket végzünk el velük, ill. amikor a memóriatartalmat listázzuk és értelmezzük!

1.11 BINÁRIS MŰVELETEK

Ha a mikroprocesszor működését a legegyszerűbb műveletek – összeadás, kivonás – elvégzésénél megvizsgáljuk, azt tapasztaljuk, hogy semmi mást nem végez, mint bitösszehasonlítások ismételt sorozatát. Az igaz, hogy a CPU számos összeadó áramkört tartalmaz, azonban ezek az áramkörök is a bitek egymás utáni összehasonlításán alapulnak. Ez az összehasonlítási művelet igen egyszerű eljárást jelent, mivel két bitnek csak három lehetséges állapota létezik. (Mindkét bit lehet zérus, a két bit közül valamelyik lehet egy, vagy mindkét bit lehet egy.) Akár összeadásról, kivonásról, szorzásról vagy osztásról van szó, a számítógép mindig a bitek összehasonlításának sorozatát végzi el.

Az összeadás esetében a következő szabályok érvényesek:

– Két zérus értékű bit összeadása a szummajelző bitet 0-ra állítja be.
– Ha a két bit közül valamelyik értéke 1, a szummajelző bit is 1.
– Ha mindkét bit 1, akkor a szummajelző bit 0, és az átviteljelző bit 1.

Ez az eljárás minden egymás után következő bit esetében megismétlődik, figyelembe véve a megelőző átviteleket is. Bár a CPU-ba az összeadás, kivonás, szorzás és osztás műveletét beleépítették, az assembler programozónak lehetősége van más bitösszehasonlító műveletek elvégzésére is. Ezek a műveletek a logikai AND (és), OR (vagy), XOR (kizáró vagy), SHIFT LEFT (bitléptetés balra), SHIFT RIGHT (bitléptetés jobbra), ROTATE RIGHT (forgatás jobbra), ROTATE LEFT (forgatás balra) és a komplemensképzés. A logikai AND, OR operátorokkal és a komplemensképzéssel a jelzőbitek állapota jobban érzékelhető, mintha numerikus értékeket vizsgálnánk. A logikai AND művelet két bitet hasonlít össze. Ha mindkét bit értéke 1, akkor az eredmény is 1. A művelet a bináris összeadástól annyiban különbözik, hogy két olyan bit összeadásakor, amelyek mindegyikének értéke 1, a szummajelző bit nullára, míg az átviteljelző bit 1-re áll be.

Logikai AND (ÉS) művelet

0. bit	1. bit	Eredmény
0	0	0
0	1	0
1	0	0
1	1	1

Az AND műveletet gyakran bizonyos bitpozíciók maszkolására használják. Példaként egy bináris formában kifejezett tömörítetlen decimális számból eltávolítjuk a legnagyobb helyértékű négy bitet, mielőtt végrehajtanánk rajta egy decimális szorzást vagy osztást. Magát a műveletet az 0000 1111 maszkon és az adott számon végrehajtott logikai AND jelenti.

```
     1010 0011
AND  0000 1111 (MASZK)
-------------------
     0000 0011
```

A kétoperandusos logikai OR művelet eredménye akkor 1, ha mindkét bit, vagy a bitek valamelyikének értéke 1. Az OR művelet bizonyos bitpoziciók beállítására használható.

Logikai OR (VAGY) művelet

0. bit	1. bit	Eredmény
0	0	0
0	1	1
1	0	1
1	1	1

Például egy 8 bites szám legnagyobb helyértékű bitjét az 1000 0000 számon és az adott számon végrehajtott OR művelettel lehet beállítani.

```
     0001 1010
OR   1000 0000
-------------
     1001 1010
```

Az EXCLUSIVE OR (kizáró VAGY) kétoperandusos művelet eredménye csak akkor 1, ha a két bit értéke eltérő. Ez a logikai művelet bizonyos bitpozíciók komplemensének képzésekor nagyon hasznos.

EXCLUSIVE OR (kizáró VAGY) művelet

0. bit	1. bit	Eredmény
0	0	0
0	1	1
1	0	1
1	1	0

A következő példában egy 8 bites szám középső négy bitjének komplemensét készítjük el a 0011 1100 szám és az EX. OR művelet felhasználásával:

```
        1010 0110
EX. OR  0011 1100
        ---------
        1001 1010
```

A bitléptető utasítások egy szám felezésére vagy megduplázásra használhatók. Ez a módszer kevesebb byte-ot és gépi ciklust igényel, mint egy valóságos osztás vagy szorzás. Előjel nélküli számok esetében a bitléptetés balra művelet és a legkisebb helyértékű bit nullázása a szám értékének megduplázását jelenti.

```
SHL   0100 0001   (Decimálisan  65)
      ----------------------------
      1000 0010   (Decimálisan 130)
```

Az előjel nélküli számok értékének felezését nagyon egyszerűen elvégezhetjük a bitléptetés jobbra művelettel és a legnagyobb helyértékű bitpozició nullázásával.

```
SHR   0000 1010   (Decimálisan  10)
      ----------------------------
      0000 0101   (Decimálisan   5)
```

A forgató műveletek a bitek átrendezését teszik lehetővé. A bitléptető műveletekhez hasonlóan ez a művelet is jobbra, ill. balra végezhető, azonban a bitléptető műveletktől eltérően – ahol a legszélső bitpozíciókon álló bitértékek elvesztek, attól függően, hogy jobbra, ill. balra léptettünk – a forgató műveleteknél az a bit, ami a szám egyik végén kilép, átkerül a szám másik szélén felszabaduló helyre:

```
ROR   0000 1111   (Forgatás jobbra)
      ----------------------------
      1000 0111

ROL   0000 1111   (Forgatás balra)
      ----------------------------
      0001 1110
```

1.12 CÍMZÉSI MÓDOK

A 80286/80386-os utasítások nemcsak egy bizonyos végrehajtandó műveletről adnak információkat, hanem a műveletben előforduló operandusok típusára, valamint ezek helyére vonatkozó előírásokat is tartalmaznak. Nyolc fő címzési módot különböztetünk meg:

1. Azonnali címzés
2. Regisztercímzés
3. Közvetlen címzés
4. Közvetett regisztercímzés
5. Relatív báziscímzés
6. Közvetlen indexelt címzés
7. Indexelt báziscímzés eltolással vagy anélkül
8. 80386-os kiegészítések

Azonnali címzés

A mikroprocesszor a művelet szintaxisából állapítja meg, hogy melyik címzési módot kell alkalmaznia. Például a következő utasítások beírásakor:

```
MOV     AH,00
MOV     AL,04
```

az operandus értéke az utasításon belül megtalálható. Jelen esetben az AH regiszter nullázásáról és az AL regiszter 4-gyel (binárisan 0000 0100) való feltöltésről van szó. Meg kell említenünk, hogy a 80386-os mikroprocesszor 32 bites operandusokat is elfogad. A következő példában egy 16 bites forrásoperandust töltünk az AX regiszterbe (valamennyi betűvel jelzett hexadecimális számjegy előtt egy 0-nak kell állnia).

```
MOV     AX,0FFFFH
```

Az azonnali címzési mód hasznosítása során valamennyi operandusnál szerepelhet előjel-kibővítés is, ha szükséges. Ez azt jelenti, hogy az operandus legnagyobb helyértékű bitjének ismétlésével a szám kitölti a céloperandus bitszélességét. Például:

```
MOV     AX,302
```

Ez az utasítás veszi a 302 10 biten kifejezhető bináris megfelelőjét, ami 0100101110, és kibővíti a céloperandus 16 bites szélességére úgy, hogy a 0 előjelbitet az AX regiszter legnagyobb helyértékű bitjeibe ismételten átmásolja. Az AX regiszterbe tehát a 0000 0001 0010 1110 szám kerül.

```
MOV     AL,-40
```

Az előjel-kibővítés a 8 bites forrás- és céloperandusokra is vonatkozik. A −40 7 bites megfelelője 1011000, amit ha 8 bitre bővítünk ki, a 11011000 számot kapjuk.

Regisztercímzés

A regisztercímzés esetében a forrásoperandus értékét a 80286/80386 belső regisztereinek egyikébe előzetesen elhelyeztük. Ez az érték 8 bites, 16 bites, vagy a 80386-os processzor esetében 32 bites lehet. A mikroprocesszor a regiszter nevéből következtet az operandus szélességére. Például:

```
MOV     DS,AX
```

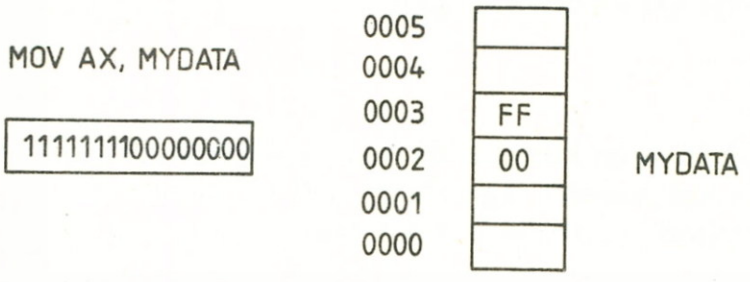
A előbbi regisztercímzési mód arra utasítja a mikroprocesszort, hogy a forrásoperandus (AX regiszter) 16 bites tartalmát tegye át a 16 bites DS regiszterbe. Ez a művelet egy 8 bites forrás- és célregiszter között is végrehajtható:

```
MOV     DL,AL
```

A nyolc fő címzési mód közül az azonnali és a regisztercímzésnél van szükség a legkevesebb gépi ciklusra a műveletek végrehajtásához. Mivel az operandus értéke magában az utasításban, ill. egy belső regiszterben már előzetesen tárolva van, külső memória vagy egyéb eszköz időigényes elérésére nem kerül sor. A további 6 címzési mód hosszabb végrehajtási időt kíván, mert a mikroprocesszornak ki kell számítania az operandus címét a szegmenscím, a szegmenshelyzet, ill. a bázisregiszter vagy az indexregiszter tartalmának alapján. A továbbiakban erre a származtatott címre mint az operandus effektív címére (EA) utalunk.

Közvetlen címzés

Közvetlen címzéskor az operandus szegmenshelyzete az utasításban mint 16 bites mennyiség szerepel. Ez az eltolási érték hozzáadódik az adatszegmens (DS) tartalmához és 20 bites származtatott operanduscímet (EA), azaz valóságos fizikai címet alkot. A közvetlen címzésű operandus általában egy címke. Például az 1–7. ábrán látható utasítás hatására a mikroprocesszor betölti az AX regiszterbe annak a memóriahelynek a tartalmát, amelyre a MYDATA címkével azonosított memóriacím mutat. Vegyük észre, hogy a mikroprocesszor az alacsony helyértékű byte-ot az alacsonyabb memóriacímre, míg a magas helyértékű byte-ot a magasabb memóriacímre helyezi.



1–7. ábra Közvetlen címzés

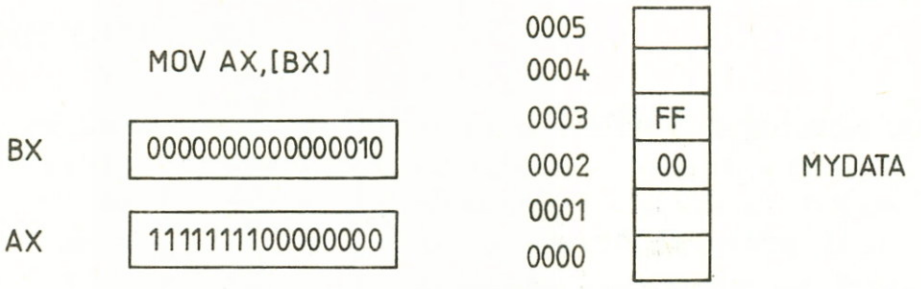
Közvetett regisztercímzés

Közvetett regisztercímzéskor a forrásoperandus megcímzése egy helyzetcímmel történik, amely a következő regiszterek egyikében található:

SI (forrásindex),
DI (célindex),
BX (bázisregiszter),
néhány esetben pedig a BP (bázismutató).

A mikroprocesszor az utasítás szintaxisa alapján ismeri fel a közvetett regisztercímzést. A forrásoperandus azonosítóját szögletes zárójelek közé foglaljuk. Az 1–8. ábrán látható művelet csak akkor működik, ha a BX regiszterbe előzetesen betöltjük a MYDATA-hoz tartozó helyzetcímet. Az OFFSET operátor használatával ez a következőképpen valósítható meg:

```
MOV     BX,OFFSET MYDATA
```



1–8. ábra A regiszter közvetett címzése

A LEA (load effective address = effektív cím betöltése) utasítás használatával az előző művelettel azonos eredményre jutunk:

```
LEA     BX,MYDATA
```

A regiszter közvetett címzése jól alkalmazható olyan esetekben, amikor az adatokat egy táblázatban tároltuk. Az egyes elemek elérése hatékonyabb, ha a bázisregiszter értékét egy ciklusban növeljük, mintha minden alkalommal kivesszük a szükséges címet a memóriából, majd ezzel az értékkel címezzük meg a forrásoperandust.

Relatív báziscímzés

Az operandus effektív címét, a relatív báziscímzés segítségével úgy kaphatjuk meg, hogy összegezzük a bázisregiszter (BX vagy BP) tartalmát és az adatokra mutató eltolási értéket. A relatív báziscímzési módot leggyakrabban összetett adatszerkezetek – például rekordok – eléréséhez használjuk. A bázisregiszterbe az adatszerkezet kezdőcímét töltjük, és az adott mező a relatív címzéssel választható ki. A relatív cím megváltoztatása a rekordon belüli különböző mezők elérését teszi lehetővé. Különböző rekordok azonos mezőinek eléréséhez pedig csak a bázisregiszter tartalmát kell megváltoztatnunk.

Az 1–9. ábrán látható kódrészletben a MESGE1 változó egy karakterláncot tartalmaz, amelynek a helyzetcímét a LEA utasítás tölti be a BX regiszterbe. Ha a MESGE1-hez tartozó karakterlánc negyedik elemére akarunk utalni, akkor a MESGE1 báziscímét (BX) hozzá kell

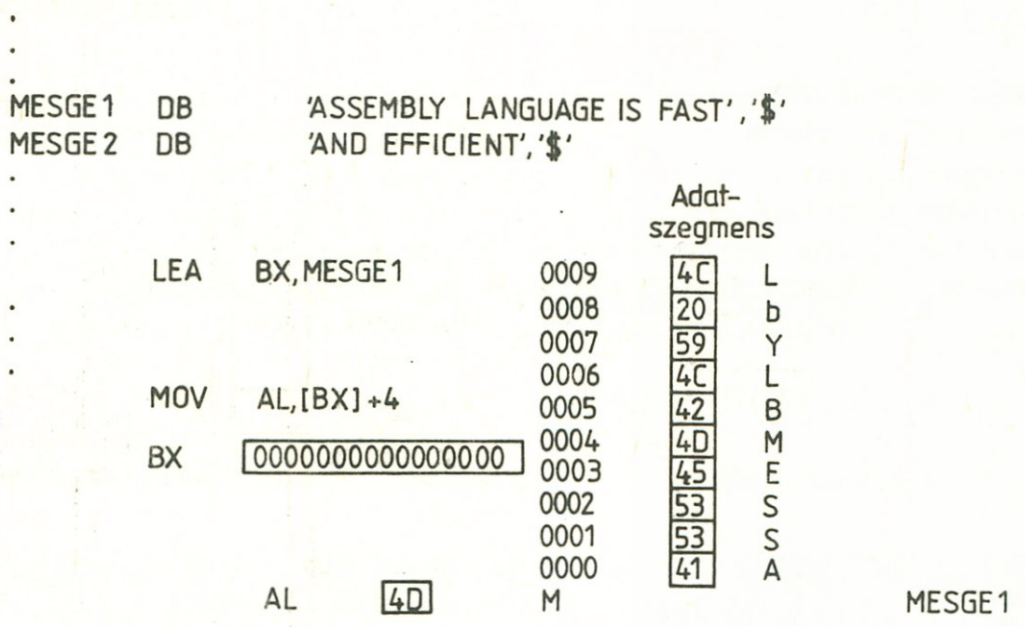
adnunk a plusz 4 relatív címhez. Az assembly nyelv a következő három módszert relatív báziscímzésnek tekinti:

```
LEA     [BX] +4
LEA     4[BX]
LEA     [BX +4]
```

Az első módszert használják a leggyakrabban, de a relatív cím megelőzheti a bázisregisztert, és azzal közös szögletes zárójelbe is kerülhet.

Ha a MESGE1 adatait sorban el akarjuk érni, a relatív címet kell növelnünk, az üzenet módosításához pedig a báziscímet kell megváltoztatnunk a következő utasítással:

```
LEA     BX,MESGE2
```



1–9. ábra Relatív báziscímzés

Közvetlen indexelt címzés

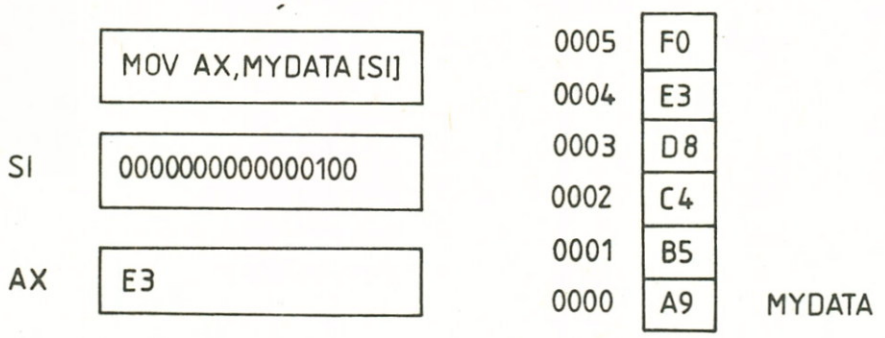
A közvetlen indexelt címzés esetében az operandus helyzetcímét úgy számíthatjuk ki, hogy a kiválasztott szegmensen belül az eltolási értéket egy indexregiszterhez (SI vagy DI) adjuk. A közvetlen indexelt címzést leggyakrabban statikus tömbök elemeinek eléréséhez használják. Az eltolási érték határozza meg a tömb kezdetét, és az indexregiszterben tárolt értékkel utalhatunk a szerkezeten belüli önálló elemre. A rekordoktól eltérően – ahol az önálló mezők szélessége adattípus és méret szerint változhat –, a tömb elemei homogének. Mivel az elemek típusa és mérete megegyezik, a tömbön belüli mozgás az eltolási érték növelésével vagy csökkentésével érhető el.

```
MOV     SI,4
MOV     AL,ARAY1[SI]
```

Az eltolási érték megválasztásakor gondosan kell eljárnunk, mivel figyelembe kell vennünk a tömb adattípusát is. Az előbbi példában az AL regisztert az ARAY1 tömb ötödik elemével töltjük fel, mivel a tömb adattípusa byte.

```
MOV     AX,ARAY1[SI]
```

Mint az 1–10. ábrán látható, ugyanezzel az utasítással az AX regiszter egy 16 bites értékkel is feltölthető attól függően, hogy milyen a tömb adattípusa.



1–10. ábra Közvetlen indexelt címzés

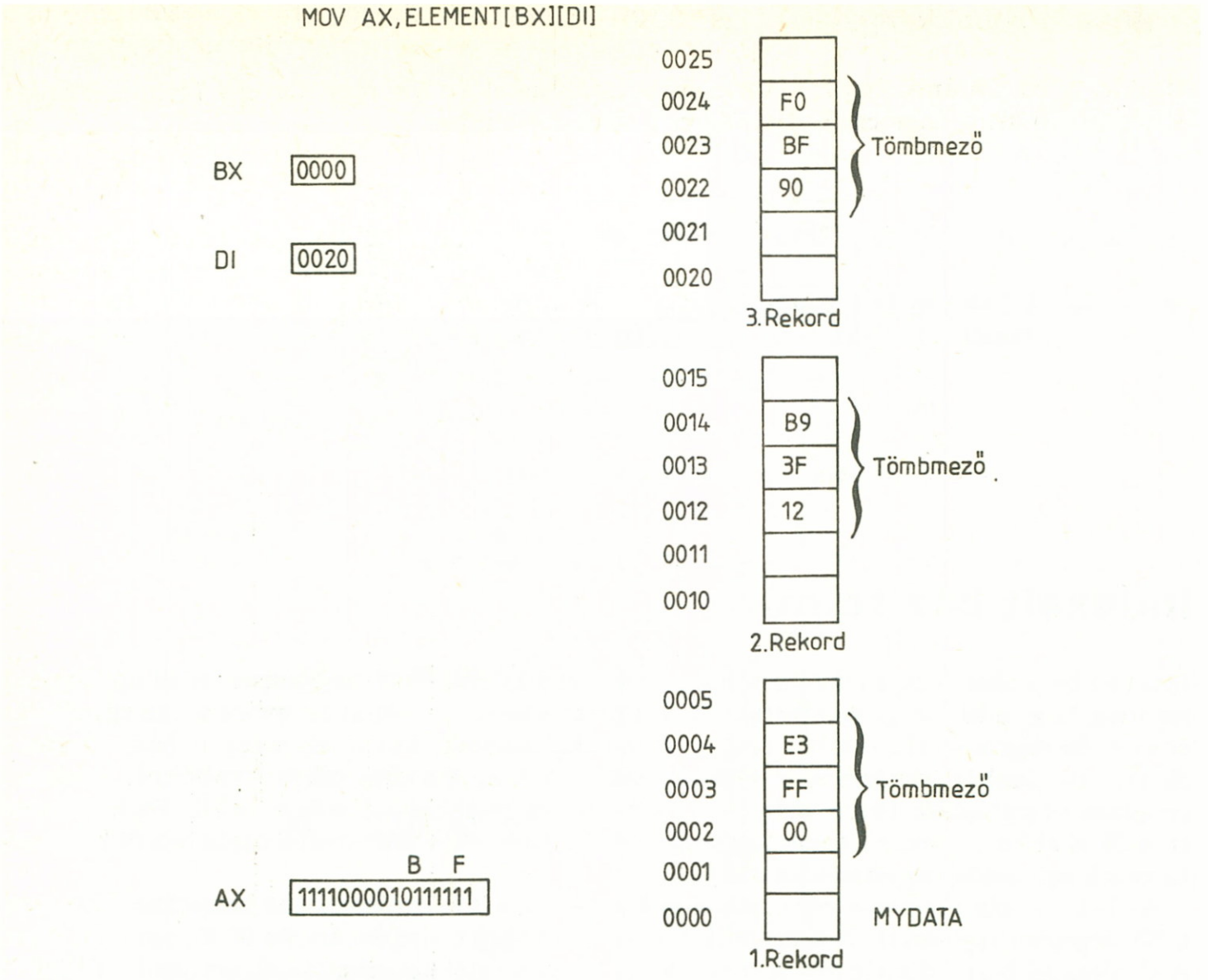
Indexelt báziscímzés

Indexelt báziscímzés esetén az operandus helyzetét a kiválasztott szegmensen belül úgy kaphatjuk meg, hogy a bázisregiszter tartalmát, az indexregiszter tartalmát és opcionálisan egy eltolási értéket összegzünk. Ha eltolási értéket nem alkalmazunk, akkor az indexelt báziscímzést a dinamikus tömbök elemeinek eléréséhez használhatjuk. (A dinamikus tömböknél a bázis a program végrehajtása során változhat.) Az eltolási értékkel a tömbön belüli önálló elemek érhetők el akkor, amikor a tömb egy adatszerkezeten – rekordon – belüli egyik mezőt képviseli. Lássunk egy példát az elmondottakra!

Az 1–11. ábrán egy olyan esetet mutatunk be, amikor a rekordszerkezet báziscíme 0000, amit a BX regiszter tartalmaz. A harmadik rekord kezdetének eltolási értéke 0020, amit DI tárol. A tömbmező harmadik elemét ugyancsak egy eltolási értékkel érhetjük el, ami az ELEMENT változó inicializálási értéke.

80386-os kiegészítések

A 32 bites címzési mód esetén lehetővé válik hogy bármelyik regisztert használjuk index- vagy bázisregiszterként, azonban mindkettőben érvényes 32 bites értékeknek kell lenniük. Bármilyen 16 bites utasítás megcsonkítja a 32 bites regiszter tartalmát, tehát a felső 16 bitet figyelmen kívül hagyja.



1–11. ábra Indexelt báziscímzés

1.13 PROGRAMOZÁSI STÍLUS

Az assembly program olyan végrehajtható utasítások sorozatából áll, amelyek arra késztetik az assemblert, hogy különböző utasításokat végezzen el. Ezen utasítások sorozatára gyakran mint forráskódra utalunk. Mint bármely más nyelvnek, az assembly forráskódnak is előre definiált szintaxisa van. Minden assembly utasítás a következő négy mezőből áll:

Névmező **Műveleti mező** **Operandusmező** **Megjegyzésmező**

bár bizonyos assembly utasítások nem használják mindegyik mezőt. A megjegyzésmező a programozás belső dokumentálását szolgálja. Alkalmazása nem kötelező.

Névmező

A névmező, amelyet néha címkemezőnek is neveznek, egy assembler utasítás aktuális kezdő memóriacíméhez egy szimbolikus nevet rendel hozzá, amelynek hatására a programozó egy utasításra név szerint utalhat, és így szükségtelenné válik az utasítás címének nyomon követése. Ez különösen az áthelyezhető kód készítésénél hasznos. A szimbolikus hivatkozással a programozó a szerkesztőprogram számára lehetővé teszi, hogy az maga válassza ki a memóriában azt a helyet, ahová a programot betölti majd. A kód áthelyezésekor valamennyi utasítás, hivatkozás automatikusan megváltozik.

Nevet bármelyik utasításhoz hozzárendelhetünk. Ez a mező azonban rendszerint olyan utasítások részére van fenntartva, amelyekre adatdefiníciókban, konstansokban, szegmensekben, ciklusokban, programelágaztatásokban vagy szubrutin hívásokban hivatkozunk majd. A névnek alfabetikus karakterrel kell kezdődnie, és maximálisan 31 karaktert tartalmazhat a következők szerint:

- Valamennyi betű: A-tól Z-ig (Angol ABC!)
- Numerikus számjegyek: 0-tól 9-ig
- Speciális szimbólumok: _ $. ? @ % (6 darab)

A név kiválasztásakor gondosan kell eljárnunk. Olyan nevet nem használhatunk, ami az assembler részére fenntartott szó vagy deklaráció. Ha a névnek pontot (.) kell tartalmaznia, akkor ennek kell szerepelnie első karakterként.

Változók

A változónév egy olyan memóriahelyet jelent, aminek a tartalma a program futása során változhat. A változók definiálása a memóriahely címére, az adat típusára és méretére vonatkozó információt tartalmazza. A változók egyszerű, indexelt vagy strukturált operandusként alkalmazhatók.

Címkék

Az alkalmazói programban a végrehajtható utasításokhoz rendelt nevekre történő hivatkozások kódrelatívak. A névnek vagy címkének ebben az esetben három jellemzője van: a szegmenscím, a szegmenseltolás és az elérés módját leíró elem (NEAR/FAR). A CPU kétféle módon címezhet meg egy bizonyos címkét. Ha a címke, amelyre a hivatkozás történik, az aktuális kódszegmensen belül van, akkor az utasítás eléréséhez csak a szegmenseltolásra van szükség. Ebben az esetben azt mondhatjuk, hogy a címke NEAR (közeli) típusú. A NEAR típusú címke egy kettősponttal definiálható, amelyet közvetlenül a címke után kell elhelyezni, de a NEAR pszeudo művelet is használható:

```
LOOP1:
```

A kettőspont jelzi a fordítóprogram számára, hogy ez egy olyan utasítás, amelyre az aktuális kódszegmensen belül található az utalás.

```
COUNT    LABEL    NEAR
```

A előbbi példában a címkét a LABEL pszeudo művelettel egyértelműen NEAR-nek definiáltuk. A címke címezhetőségének másik módjánál egyaránt szükség van a szegmens címére és az eltolásra is. Ez az eset akkor fordul elő, amikor az assembly utasítás, amelyre utalás történt, nincs az aktuális kódszegmensen belül. Ilyenkor a címkét FAR-nak kell definiálnunk.

```
MYCODE    LABEL    FAR
```

Az előbbi példában a LABEL pszeudo műveletet használtuk a FAR attributummal a címke jellemzőjének kifejezésére. A FAR attributumot az EQUate (egyenlőség), és PROCedure (eljárás), valamint az EXTRNal (külső) utasítások címkézésénél is használhatjuk, mint ezt a következő példák is mutatják:

```
TEN       EQU      FAR      10
PRNTIT    PROC     FAR
EXTRN     RANDM:FAR
```

Konstansok

Lehetőségünk van olyan memóriahelyek definiálására is, amelyeknek az inicializált értékük a program futása során változatlan marad. Ezeket az értékeket konstansoknak nevezzük. A konstansoknak nyolc típusát különböztetjük meg.

1. Bináris

A bináris konstans 0-k és 1-esek sorozatából áll, amelyet egy B betűnek kell lezárnia. Például:

```
EIGHT     EQU      00001000B
```

2. Decimális

A decimális konstans a 0–9 tartományba eső számok sorozatából áll, amelyet – nem kötelezően – egy D betű követhet. A rendszer a számok sorozatát egy decimális számnak fogja tekinteni, hacsak az alapszám (RADIX) meg nem változott. Lássunk tehát egy példát a decimális konstansra is:

```
FORTY     EQU      40D
```

3. Hexadecimális

A hexadecimális konstans a 0–9 és az A–F tartományokba eső számjegyek, ill. betűk sorozatából áll, amelyet a H betűnek kell követnie. Az első karakternek számjegynek kell lennie, ui. a fordítóprogram innen tudja eldönteni, hogy ebben az esetben egy számról, nem pedig címkére való utalásról vagy változónévről van szó.

Ha a hexadecimális szám az A, B, C, D, E, F betűk valamelyikével kezdődik, akkor a betű elé egy nullát kell helyeznünk. A hexadecimális konstans deklarálására is mutatunk két példát:

```
FIFTY     EQU      32H
HEXNM     EQU      0FFH
```

Az FFH számnál a 0 jelzi a fordítóprogramnak, hogy hexadecimális számról nem pedig címkéről vagy változónévről van szó.

4. Oktális

Az oktális konstans a 0–7 tartományba eső számjegyekből állhat, amelyet az O vagy a Q betűnek kell követnie. Például:

```
SIX     EQU     6O      vagy    6Q
```

5. Karakter

A karakterkonstans bármelyik ASCII karakter lehet, amelyet szimpla vagy dupla idézőjelek közé foglalunk. Ha a konstans kettőnél több karakterből áll, akkor a DB (Define Byte = byte definiálás) pszeudo műveletet kell használnunk.

Ha a karakterlánc-állandó csak egy, vagy két karakterből áll, akkor a DD, DQ, DT vagy DW pszeudo műveletet lehet használni. Például:

```
INITL   DD      'B'
NAME    DB      "J WILLIAMS"
```

6. Lebegőpontos

Ennél az adattípusnál, amelyet nem minden assembler támogat, a decimális számokat exponenciális alakban adjuk meg. Például:

```
SINE    DD      0.332E-1
```

7. Hexadecimális valós

Ez egy olyan konstans, amely a 0–9, ill. a az A–F tartományokba eső számok, ill. betűk sorozatából áll, amelyet az R betű követ. A hexadecimális konstanshoz hasonlóan itt is az első karakternek a 0–9 számok valamelyikének kell lennie. A konstans összesen 8, 16 vagy 20 számjegyből állhat, kivéve akkor, ha az első számjegy 0. Ebben az esetben ui. a számjegyek számának összesen 9-nek, 17-nek vagy 21-nek kell lennie. (Ezt az adattípust sem támogatja minden assembler.) Például:

```
HRNUM   DD      0FAB12345R
```

8. Egyenlőségek

A névmezőben szereplő címkét az EQU pszeudo művelettel vagy az (=) egyenlőségjellel hozzárendelhetjük egy operandusmező értékéhez. Az EQU pszeudo művelet a változóhoz egy konstanst rendel hozzá, amelynek értéke a program futása során nem változik. Ha az egyenlőségjelet használjuk, a konstans értéke a program futása során megváltoztatható. Például:

```
SCADD   EQU   [BP+16]
BASNUM  =     1980
```

Az első példában, az SCADD elnevezés a [BP+16] index-kifejezés helyére behelyettesíthető. Hasonlóan, a BASNUM is behelyettesíthető az 1980-as érték helyébe. Az utóbbi esetben a BASNUM-hoz a program futása során új érték is hozzárendelhető.

9. Szegmensnevek

A szegmenscímkét a szegmensutasítás mezőjében adhatjuk meg. Például:

```
MYCODE      SEGMENT     PARA     'CODE'
```

Műveleti mező

A műveleti mező az aktuális mikroprocesszor utasítás mnemonikját tartalmazza. A mnemonik egy olyan 2–6 karakterből álló emlékeztető szimbólum, amellyel a gépi kódú utasítások bináris vagy hexadecimális értékei helyett angol szavak rövidítéseit használhatjuk. A mnemonik a program olvasását és értelmezését megkönnyítő nemzetközi átalakítási táblázat, amelynek segítségével nem kell, hogy kapcsolatban legyünk a valóságos bináris gépi kódú értékekkel. A mnemonik jelenthet gépi utasítást, makro utasítást vagy pszeudo műveletet.

Lássunk mindjárt egy példát!

```
INITIAL     MOV     AX,0H
```

Ebben az esetben az INITIAL a címke, a MOV pedig a művelet. A műveleti mezőt az operandusmező követi. A művelet nemcsak azt mondja meg az assemblernek, hogy melyik utasítást kell végrehajtania, hanem azt is, hogy a műveletben mennyi és milyen típusú operandusra van szükség. A művelet makro utalást is tartalmazhat, ami az assemblert előzetesen definiált utasításokból álló rutin végrehajtására utasítja. Ennek hatására az assembler egy olyan forráskódot generál, mintha ezek a makro rutinok is az eredeti program részét képezték volna.

Például:

```
DOS_INT     MACRO     SERVICE_ID
```

Ez a művelet jelzi az assemblernek, hogy egy olyan kód következik, ami a MACRO definíció része. A pszeudo műveletek hatására gépi kód keletkezése helyett az assembler bizonyos műveleteket elvégez az adatokon, a kódlistákon, a programelágazásokon vagy a makrókon.

Operandusmező

Az operandusmező olyan adatok helyét vagy helyeit tartalmazza, amelyekkel a műveletet el kell végezni. Ha az utasítás egy- vagy kétoperandusos, az operandusokat az utasítástól legalább egy szóközzel el kell választani. Ha két operandus szerepel, ezeket vesszővel kell elválasztani egymástól. Vannak azonban olyan műveletek is, amelyeknek nincs szükségük operandusra.

Ha a művelet kétoperandusos, az első operandust cél-, a másodikat forrásoperandusnak nevezzük. Az adatátviteli, a regiszteres vagy a memóriatárolási műveletek kétoperandusos utasításokkal történnek. Például:

```
MOV     AX,8
```

Az előbbi esetben az adat, amellyel dolgozunk, forrásoperandusként szerepel, és ezt az értéket helyezzük el a céloperandusba, vagyis az AX regiszterbe.

Megjegyzésmező

A megjegyzésmező a négy mező közül az utolsó, de az egyik leghasznosabb. A célja az, hogy a forrásprogramon belül az utasításokat dokumentálja. A megjegyzéseket az assembler figyelmen kívül hagyja, tehát csak a forrásprogram listázásakor lényeges a programozó számára. A megjegyzést a legutolsó mezőtől legalább egy szóközzel el kell választani és pontosvesszővel kell kezdeni. Ajánlatos minden olyan esetben megjegyzésekkel ellátni a program sorait, amikor a művelet célja nem érthető azonnal. Például:

```
MOV     AH,45H      ;PARAMETER EGY KARAKTER BEOLVASASAHOZ
```

A megjegyzésben elmagyaráztuk, hogy miért van szükség az AH regiszter 45H-val való feltöltésére. Ebben az esetben a 45H a megfelelő művelet indítását teszi lehetővé egy megszakítás hívásakor.

1.14 AZ ASSEMBLY NYELV ELŐNYEI

A programozás kezdeti időszakában valamennyi program gépi kódokból állt. Az utasításokat bináris vagy hexadecimális alakban kellett megírni, amelyet a hardver értelmezett és végrehajtotta a kívánt műveletet. Ahogy a programok egyre nagyobbakká és bonyolultabbakká váltak, erősen megnőtt az igény olyan szoftverek és nyomkövetési eljárások iránt, amelyekkel produktív szoftverfejlesztés valósítható meg. Az assembly programozás volt a megoldás, ez a nyelv ui. azonnal és könnyen érthető, és automatikusan nyomon követi a gépi kódra jellemző aprólékos részleteket, nem tartalmazza a magas szintű nyelvekre jellemző előzetes adminisztrációkat, és igen gyorsan fut.

1.15 ASSEMBLY NYELVŰ MINTAPROGRAM

A következőkben egy olyan assembly programot mutatunk be, amelyet már előzetesen lefordítottunk. A lista jobb oldalán a könnyen olvasható és jól dokumentált assembly forrásprogram látható. A bal oldalon a lefordított gépi kódú megfelelők szerepelnek. (A gyakorlatlan programozó számára a gépi kód először rejtélyesnek tűnhet.)

Az első oszlopban 1-től 31-ig egy sorszámozást figyelhetünk meg, amit az assembler készít el. A második oszlop azokat a valóságos memóriacímeket tartalmazza, ahonnan kezdve a lefordított mnemonik műveletet a rendszer tárolta. (A címek hexadecimális számrendszerben jelennek meg, és 16 bitesek.) A harmadik és negyedik oszlopban az egyes mnemonik műveletek gépi kódú megfelelői láthatók. Az oszlopok száma az utasítások típusától és az operandusok számától függően változik. A további oszlopok az assembly forrásprogram szintaxisának megfelelően a név-, a műveleti, az operandus- és a megjegyzésmezőket jelenti. Vegyük észre, hogy a 2–4. 21. és 25. sor kizárólag megjegyzéseket tartalmaz. (Az Olvasó az 5. fejezetben sajátíthatja el az assembly programozás fortélyait.)

```
                                ; 8088/80386-OS GEPEKRE KESZULT PROGRAM
                                ; EGYSZERU HEXADECIMALIS OSSZEADAS BEMUTATASA AZONNALI
                                ; CIMZES FELHASZNALASAVAL

0000                            STACK   SEGMENT PARA STACK
0000     40 [                           DB      64 DUP ('MYSTACK ')
            4D 59 53 54
            41 43 4B 20
                        ]

0200                            STACK   ENDS

0000                            MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
0000                            MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
                                        ASSUME  CS:MYCODE,SS:STACK
0000  1E                                PUSH    DS              ; A DS REGISZTER ELMENTESE
0001  2B C0                             SUB     AX,AX           ; AX TORLESE
0003  50                                PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE

                                ; HAROM SZAM OSSZEADASA
0004  B0 23                             MOV     AL,23H          ; 23H ELHELYEZESE AZ AL REGISZTERBEN
0006  04 0A                             ADD     AL,0AH          ; 0AH HOZZAADASA AZ AL REGISZTERHEZ
0008  04 10                             ADD     AL,10H          ; 10H HOZZAADASA AZ AL REGISZTERHEZ

                                ; VEGE AZ OSSZEADASI PELDANAK. AZ EREDMENY AL-BEN

000A  CB                                RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
000B                            MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
000B                            MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                        END     MYPROC          ; A PROGRAM VEGE
```

2. BEVEZETÉS AZ ASSEMBLEREKBE

Ebben a fejezetben megtudhatjuk, hogy mi az assembler, miért van rá szükség, és mit végez el. Rámutatunk az assemblerek és a magas szintű nyelvek fordítóprogramjai közötti hasonlóságokra és eltérésekre. Egyszerű mintaprogram felhasználásával bemutatjuk, hogyan kell létrehozni, lefordítani és futtatni az assembly programot. A fejezetben ismertetésre kerülő SAMPLE.ASM programot használjuk az assembly program minden szüséges összetevőjének bemutatására. A 80286/80386-os és 80287/80387-es utasításkészletének részletes ismertetésére a 3., ill. a 4. fejezetben kerül sor.

Az assembler egy olyan program, ami a szintaktikailag stilizált szövegfile-t, a forrásprogramot el tudja olvasni. A forrásprogramot angol szavak rövidítéseiből képzett mnemonikokkal kell megírnunk. Az assembler ezeket a mnemonikokat, bináris nullákká és egyekké alakítja, ami már a mikroprocesszor anyanyelve. Ezt a lefordított változatot nevezzük gépi kódnak. Minden egyes mnemonik egy valóságos gépi kódú utasítás számunkra értelmet jelentő rövidítése.

A mnemonik és az aktuális gépi kódú utasítás kölcsönösen megfeleltethető egymásnak. A mnemonikok segítségével az assemlby forrásprogram könnyen megírható, olvasható és nyomon követhető. A magas szintű nyelvek fordítóprogramjai hasonló feladatot látnak el, vagyis beolvassák a szövegfile-t (a forrásprogramot), és átalakítják gépi kóddá. A magas szintű nyelvek a programozó számára lehetővé teszik, hogy a programozási feladatra, ne pedig a részletekre (hogy melyik regisztert, memóriahelyet vagy portot használja) koncentráljon. A hardver-függetlenség eredményeként a magas szintű program nem kötődik egy adott típusú mikroprocesszorhoz, mint az assembly nyelvű társa. A magas szintű nyelvek esetében a forrásprogram sorai és azok gépi kódú megfelelői között nincs kölcsönös megfeleltetés. A program lefordított változata általában sokkal hosszabb, mint egy ennek megfelelő, jól megírt assembly nyelvű program. Ez azonban nem jelenti azt, hogy a fordítóprogramok nem képesek hatékony kódot generálni. Valójában, ha az első assembly nyelvű próbálkozásainkat vetnénk össze vele, elképzelhető, hogy a simábban járható utat, vagyis egy magas szintű nyelven készített és fordítóprogrammal lefordított változatot választanánk.

2.1 A GÉPI KÓD ÉS AZ ASSEMBLY NYELV

A következő két programrészlet az assembler hasznosságát, valamint az aktuális gépi kódok és az assembly nyelvű programsorok kölcsönös megfeleltetését mutatja be. Mind a gépi kódú, mind az assembly nyelvű programrészlet funkcionálisan azonos, vagyis elvégeznek bizonyos szükséges kezdeti inicializálásokat, betöltik az akkumulátorba azt az értéket, amit a NUMONE változóban tároltuk és 3H-t adnak az akkumulátorban tárolt értékhez.

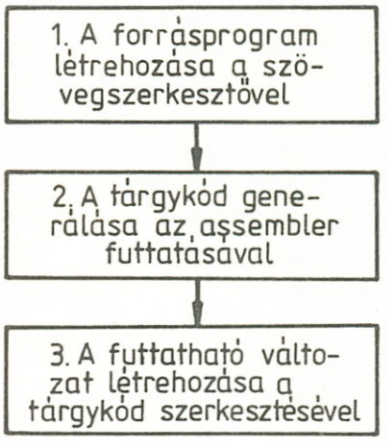
Gépi kód	Assembly megfelelő	
00011110	PUSH	DS
00101011	SUB	AX,AX
11000000		
01010000	PUSH	AX
10111000	MOV	AX,MYDATA
00000001		
11101100		
10001110	MOV	DS,AX
11011000		
10100001	MOV	AX,NUMONE
0000		
00000101	ADD	AX,3H
0011		

A program gépi kódú változatát tárgykódnak is szokás nevezni. Ha a program tárgykód alakban van, akkor közvetlenül elhelyezhető a memóriában és a megfelelő mutatók beállításával azonnal futtatható. A gépi kódú programozás meglehetősen időigényes, a szemet megerőltető és könnyű hibát véteni benne. Miként az előbbi rövid gépi kódú részletnél is láthattuk, a bináris értékek értelmét meglehetősen nehéz lenne megfejteni. Honnan tudjuk, hogy utasításokról, adatokról vagy címekről van szó? A bináris kódban szereplő programoknak nemcsak a megírása, hanem a hibák megkeresése és kijavítása is rendkívül nehéz feladat. Annak ellenére, hogy a mikroprocesszorok csak a nullákat és az egyeseket tudják értelmezni, szerencsére vannak assemblerek, amelyek segítenek a gépi kódú programok megírásában. Az előbbi program assembly változata sokkal jobban érthető a számunkra, a numerikus utasításokat ui. alfabetikus rövidítésekkel cseréltük fel. Ezáltal nagymértékben növeltük a program értelmezhetőségét és megkönyítettük a hibakeresést. Érdemes megfigyelnünk az assembly nyelvű utasítások és azok gépi kódú megfelelői közötti kapcsolatot is. A könyvben a három legnépszerűbb assemblert tárgyaljuk: az IBM Makroassemblert, a Microsoft Makroassemblert és a Turbo Editasm assemblert.

Az Olvasó tapasztalni fogja, hogy a három assemblernek sok hasonló jellemzője van. A különbözőségek főleg a szövegszerkesztési funkcióban és a fordítási opciókban vannak. Amikor az aktuális kód megírása és végrehajtása következik, valamennyi assembler ugyanazt az utasításkészletet, ill. alkészletet használja. A könyv végén található mellékletekben mindhárom assemblerről részletes leírást adunk. Ha már az utasításkészletről szó esett, érdemes megjegyezni azt az elvet, amit az Intel cég a mikroprocesszorai kifejlesztése során szem előtt tartott. Az Intel cég mikroprocesszorai felülről kompatibilisek egymással, ami azt jelenti, hogy a mikroprocesszorok új generációi az előzőeknél gyorsabbak, nagyobb utasításkészlettel és fejlettebb hardverrel rendelkeznek, azonban az előző generációk utasításait is tartalmazzák. Tehát a régebbi programok módosítás nélkül vagy csak kis módosítással futtathatók az újabban készült mikroprocesszorokon. Ezáltal a felhasználó mentesül a a szoftverek költséges újrafejlesztésétől.

2.2 A TIPIKUS ASSEMBLY ELJÁRÁS LÉPÉSEI

Az assembly program létrehozása több lépésből áll. Ebben a részben azokat a jellemzőket emeljük ki, amelyek mindhárom assemblerben közösek. (Az alapvető programozási technikák részletes leírását az 5. fejezetben tartalmazza.) A futtatható program létrehozása három alapvető lépésre bontható (l. 2–1. ábrát). Az első lépésben a szövegszerkesztő felhasználásával létrehozzuk a forráskódot. A második lépésben az assembler segítségével a forráskódot tárgykóddá alakítjuk, ami hasonló a gépi kódhoz. A harmadik lépés a szerkesztés, ami átalakítja a tárgykódot .EXE file-lá, ami már futtatható. Egy másik fordítási folyamattal .COM file is létrehozható, azonban ezt a folyamatot az egyes assemblereknél külön tárgyaljuk. Mivel az elég valószínűtlen, hogy mindenki elsőre hibátlan programot írjon, egy további hibakeresési lépésnek is szerepelnie kell a szoftverfejlesztés folyamatában.



2–1. ábra Az assembly eljárás menete

1. Lépés: A forráskód létrehozása

Ehhez a művelethez bármelyik szövegszerkesztő felhasználható, csak az a fontos, hogy az elkészített file ASCII formátumban legyen. Az ASCII file-formátum mentes mindenféle vezérlőkódoktól, amelyekkel az aláhúzást, nyomdai kizárást, indexelést, vastag betűt stb. lehet készíteni. Az ilyen információk feleslegesek az assembler számára. Majdnem minden számítógépben megtalálhatók az IBM DOS EDLIN sorszerkesztőjéhez hasonló egyszerű szerkesztőprogramok. Az EDLIN-t sorszerkesztőnek nevezzük, mert a szöveg minden sorát külön kezeli és emiatt teljesen különbözik a teljes képernyős szövegszerkesztőtől. A szövegmódosítások, törlések, beszúrások a kurzor mozgatása a képernyőn a teljes képernyős szövegszerkesztő esetében egyszerűen és gyorsan elvégezhetők, míg az EDLIN-nel a szövegfile-nak egyszerre csak egy sora kezelhető. Ha komoly assembly programot akarunk írni, akkor egy teljes képernyős szövegszerkesztőt kell használnunk. A könyvben található valamennyi mintaprogramot az IBM Professional Editor-ral, ill. a Norton Editor-ral készítettük.

Az ASCII formátumú file-t több szabványos szövegszerkesztő is elő tudja állítani. (A Wordperfect, a Wordstar és az EasyWriter egyaránt képesek ASCII file-ok előállítására.) A szövegszerkesztő segítségével gépeljük be a 2–2. ábrán látható assembly forráskódot. A beírásnál igyekezzünk minden sort úgy elhelyezni, ahogy az ábrán látjuk. Minden sort az első oszlopban kezdjünk el írni. Mivel mindhárom assembler, amelyről majd szót ejtünk, az .ASM

kiterjesztéssel várja a forráskódot tartalmazó file nevét, az elkészült file-t a SAMPLE.ASM névvel mentsük ki. (A fordító megjegyzése: A SAMPLE helyett természetesen más szó is szerepelhet – pl. MINTA –, de az ASM kiterjesztésen nem módosíthatunk.) A sorszámozás csak a jobb eligazodást szolgálja. A saját file-unk elején nem szabad sorszámokat elhelyeznünk. Amint a forráskód létrehozásával elkészültünk, nézzük végig, hogy miért szükségesek bizonyos kódok és mi történik az egyes kódokon belül.

```
        ;IBM 8088/80286-OS GEPEKRE KESZITETT PROGRAM
2.      ;A PROGRAM KARAKTEREKET JELENIT MEG A KEPERNYON

3.      STACK   SEGMENT PARA STACK
4.              DB      64 DUP ('STACK')
5.      STACK   ENDS

6.      DATA    SEGMENT PARA    'DATA'
7.      MESAGE  DB  'MOST MINDENKINEK KITARTASRA, HITRE, REMENYRE ES'
                DB  ' SZERETETRE VAN SZUKSEGE','$'
8.      DATA    ENDS

9.      CODE    SEGMENT PARA  'CODE'    ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
10.     MAIN    PROC    FAR             ;AZ ELJARAS NEVE MAIN
11.             ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
12.             PUSH    DS              ;A DS REGISZTER TAROLASA
13.             SUB     AX,AX           ;AX TORLESE
14.             PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
15.             MOV     AX,DATA         ;AZ ADATOK HELYENEK BETOLTESE AX-BE
16.             MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
17.             MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE

18.     ;AZ UZENET MEGJELENITESET SZOLGALO RUTIN
19.             LEA     DX,MESAGE       ;MESAGE HELYENEK BEKERESE
20.             MOV     AH,09           ;DOS PARAMETER BEALLITAS
21.             INT     21H             ;DOS MEGSZAKITAS

22.             RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
23.     MAIN    ENDP                    ;A MAIN ELNEVEZESU ELJARAS VEGE
24.     CODE    ENDS                    ;A KOD ELNEVEZESU KODSZEGMENS VEGE
25.             END     MAIN            ;A PROGRAM VEGE
```

2–2. ábra SAMPLE.ASM program

A program első két sora a következő:

```
1.      ;IBM 8088/80286-OS GEPEKRE KESZITETT PROGRAM
2.      ;A PROGRAM KARAKTEREKET JELENIT MEG A KEPERNYON
```

Láthatjuk, hogy megjegyzésekről van szó. Minden olyan karakter- láncot, amely előtt pontosvessző áll, az assembler figyelmen kívül hagy. Jelen esetben a megjegyzést arra használtuk, hogy a programot elnevezzük, és tömören ismertessük a program célját.

A következő programrészlet két utasítást tartalmaz (SEGMENT (3. sor) és ENDS (5. sor)), amelyek az átalakítandó utasítások elejét és végét definiálják.

```
3.      STACK   SEGMENT PARA STACK
4.              DB      64 DUP ('STACK')
5.      STACK   ENDS
```

Az a három makroassembler, amelyet részletesen ismertetünk, a következő négy egyidejűleg aktív memóriaszegmenst engedélyezi:

CODE	(kód)
DATA	(adat)
STACK	(verem)
EXTRA	(extra)

A szegmentálás lehetővé teszi a jól-strukturált moduláris forráskód megírását. A harmadik sor a címkemezőben a STACK szót tartalmazza, ami az ebben a sorban szereplő szegmens elnevezését jelenti. Az ugyancsak a harmadik sorban szereplő PARA operandus előírja, hogy a szegmensnek a memória egy szabványos paragrafusánál, vagyis 16 byte-os határánál kell majd kezdődnie. A sor végén szereplő STACK azt a nevet adja meg, amivel a szegmens a keresztreferenciában kapcsolatba hozható. A negyedik sor a verem méretét 64 Definiált Bytenak (DB) adja meg és közli az assemblerrel, hogy ezeket a helyeket a STACK szóval inicializálja. Az ENDS utasítás (5. sor) előtt egy címke áll, ami a STACK szegmens definiálásának befejezését írja elő.

A következő három programsor az adatszegmens (DATA) megnevezését és definiálását tartalmazza.

```
6.      DATA    SEGMENT PARA    'DATA'
7.      MESAGE  DB  'MOST MINDENKINEK KITARTASRA, HITRE, REMENYRE ES'
                DB  ' SZERETETRE VAN SZUKSEGE','$'
8.      DATA    ENDS
```

Ez a szegmens szintén egy paragrafus határánál kezdődik, és a DATA szolgáltatja a keresztreferenciát (6. sor). Ebben a tömör adatszegmensben csak egyetlen változót definiáltunk. A MESAGE (DB) karakterlánc ui. változónak tekintendő, amelynek a kezdőértéke a következő: 'MOST MINDENKINEK KITARTASRA, HITRE, REMENYRE ES SZERETETRE VAN SZUKSEGE', '$'.

Meg kell jegyeznünk, hogy néhány assembler igen érzékeny a DATA szegmensben definiált és felhasznált változókkal kapcsolatban. A könyvben szereplő valamennyi példában az adatszegmens a program elején szerepel. (Az adatszegmens a kódszegmens után is szerepelhet.)

Ha az adatszegmenst a forráskód elején elhelyezve erre vonatkozó hibaüzenetet kapunk, helyezzük át a program végére.

A 9. sor a kódszegmens kezdetét definiálja, ami ugyancsak egy paragrafus határán kezdődik, és a CODE szó adja a keresztreferenciát.

```
9.      CODE    SEGMENT PARA  'CODE'    ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
```

Vegyük észre hogy ettől a sortól kezdve néhány soron keresztül egy- egy tömör megjegyzéssel magyarázzuk el az utasítás hatását.

A kód következő sora a MAIN címkét és a PROC(edure eljárás) deklarációs utasítást tartalmazza.

```
10.     MAIN    PROC    FAR             ;AZ ELJARAS NEVE MAIN
```

Az eljárások (procedure) olyan programrészletek, amelyek a főprogramból való meghívással futtathatók. Valahányszor egy eljárást meghívunk, azok az utasítások, amelyek az eljárásba

tartoznak, végrehajtódnak, majd ezután az irányítást visszakapja a hívó főprogram. Az eljárás deklarálása a PROC kifejezéssel kezdődik és az ENDP kifejezéssel zárul. NEAR és FAR opciókat egyaránt tartalmazhat. A NEAR vagy a FAR arról informálja az assemblert, hogy milyen típusú ugró vagy hívó utasítást kell véghezvinnie, amikor az adott programrészhez jut. Legtöbb esetben a NEAR vagy a FAR az adott eljárás részére generált visszatérési utasítás típusát is meghatározza. A NEAR típusú eljárás hívása csak az IP értéket hagyja a veremben, míg a FAR eljárás esetén mind a CS, mind az IP értékek tárolásra kerülnek. Az 1–10 sorszámú kódsorok programról programra változhatnak. Minden programnak saját önálló megjegyzései és különböző módon dimenzionált és inicializált verem- és adatszegmensei lehetnek. A következő hét programsor (11. sortól a 17. sorig) azonban változatlan marad. Ahhoz, hogy a program megfelelően fusson, majd visszatérjen az operáiós rendszerbe anélkül, hogy azt megbénítaná, bizonyos utasításokat valamennyi assembly forráskódba el kell helyeznünk. Ezt a továbbiakban adminisztrációs programrésznek vagy programfejnek nevezzük.

Az adminisztrációs programrész

Az adatok szegmenseken belüli címezhetőségét a következő utasítás teszi lehetővé:

```
11.          ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
```

Az ASSUME utasítás azt közli az assemblerrel, hogy a C(ode) S(egment) regisztert a CODE-dal megjelölt hellyel, a D(ata) S(egment) és E(xtra) S(egment) regisztert a DATA-val megjelölt hellyel, és a S(tack) S(egment) regisztert a STACK hellyel hozza kapcsolatba.

A következő három utasítás az operációs rendszerrel kapcsolatos adatok tárolását szolgálja azért, hogy az assembly program befejeződésekor valamennyi szükséges mutatót vissza lehessen állítani.

```
12.          PUSH    DS              ;A DS REGISZTER TAROLASA
13.          SUB     AX,AX           ;AX TORLESE
14.          PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

A 12. sor az adatszegmens éppen érvényes tartalmát ráhelyezi a veremre. A 13. sor törli AX-et, amit a 14. sor utasítása ráhelyez a veremre. Az adatszegmens (DS) regiszter régi tartalmának tárolása legalább olyan bonyolult, mint feltöltése egy új értékkel. Az adatszegmens regisztert egy változó értékével nem lehet közvetlenül feltölteni.

A 15. és a 16. sor a következő:

```
15.          MOV     AX,DATA         ;AZ ADATOK HELYENEK BETOLTESE AX-BE
16.          MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
```

Legelőször az adatszegmens helyét az AX regiszterbe, majd az AX-ből a DS regiszterbe töltjük. Ugyanez az alapelv az extraszegmens (ES) feltöltésekor is.

```
17.          MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE
```

Ebben a példában mindkét szegmens ugyanarra a helyre mutat.

A 17. sor az adminisztrációs programrész utolsó sora. A 18–21. sor tartalmazza az alkalmazói programot.

```
18.     ;AZ UZENET MEGJELENITESET SZOLGALO RUTIN
19.          LEA     DX,MESAGE       ;MESAGE HELYENEK BEKERESE
20.          MOV     AH,09           ;DOS PARAMETER BEALLITAS
21.          INT     21H             ;DOS MEGSZAKITAS
```

Ebben az esetben a négy kódsor beállítja, inicializálja és hívja azt a szubrutint, amely megjeleníti a képernyőn a karakterláncot. Egy tömör négysoros programrészlettel kell megjelölnünk a program végét.

```
22.           RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
23.     MAIN  ENDP                    ;A MAIN ELNEVEZESU ELJARAS VEGE
24.     CODE  ENDS                    ;A KOD ELNEVEZESU KODSZEGMENS VEGE
25.           END     MAIN            ;A PROGRAM VEGE
```

A 22. sor RET utasítása kiveszi a veremből a visszatérés címét, ami ebben az esetben az operációs rendszerbe való visszatérést jelenti.

A 23. sor lezárja az eljárás definicióját. A kódszegmens végét a 24. sor ENDS utasítása jelzi és a 25. sor END utasítása jelenti az assembler számára azt, hogy elérte a forráskód végét. Miután a SAMPLE.ASM névvel a forráskódot tároltuk, hozzáláthatunk az assembly eljárás második lépésének elvégzéséhez.

2. Lépés: A tárgykód előállítása

A tárgykódot az assembler futtatásával tudjuk előállítani, amelyhez mindhárom assembler esetében még számos opció is a rendelkezésünkre áll. (Az opciókról a megfelelő mellékletben részletes leírást találunk.) Jelenleg csak azt ismertetjük, hogyan lehet az .OBJ file-t előállítani.

Ha a három makroassembler közül az egyiket elhelyeztük az A meghajtóban és a SAMPLE.-ASM forráskódot tartalmazó lemez a B meghajtóban van – ami az alapértelmezés szerinti meghajtó –, írjuk be a következőket:

```
B>A:MASM SAMPLE;      ;IBM MAKROASSEMBLER ESETEN
B>A:MASM SAMPLE;      ;MICROSOFT MAKROASSEMBLER ESETEN
```

Ha a Turbo Editasm programot használjuk, akkor ezt kell beírnunk:

```
B>A:TASMB SAMPLE;
```

A .OBJ file előállításához válasszuk az F7-es opciót, majd az assembler elindításához nyomjuk le az A billentyűt. A TASMB megkérdezi tőlünk az .OBJ file nevét. Az alapértelmezés szerint a név SAMPLE.OBJ lesz. A név elfogadásához az Y betű beírására van csak szükség.

Felhívjuk az Olvasó figyelmét arra, hogy mindhárom assembler .ASM kiterjesztésű forrás-file-t tételez fel. Ha a fordítási folyamat sikeresen lefutott, akkor létrejön a SAMPLE.OBJ tárgyfile, ami a B meghajtó katalógusában megtalálható.

Annak ellenére, hogy ez a file valamennyi utasítást gépi kódban tartalmaz, még nincs olyan formában, hogy az operációs rendszer a memóriába betölthetné. Ehhez előbb a harmadik assembly műveletet is el kell végeznünk az .OBJ file-on.

3. Lépés: Szerkesztés

A LINK (szerkesztő) program készíti el az .OBJ file-ból a futtatható file-t. Munkánkat ebben a fázisban is számos opció segíti, amelyeket az A, B, C mellékletben részletesen felsorolunk. Jelenleg csak azt magyarázzuk el, hogyan kell az .EXE file-t elkészíteni. A következő utasítások azt feltételezik, hogy a LINK program, amellyel mindegyik assemblert vagy DOS-t ellátták,

az A meghajtóban, míg a SAMPLE.OBJ file a B (alapértelmezés szerinti) meghajtóban van.
A SAMPLE.EXE file létrehozásához írjuk be a következőket:

```
B>A:LINK SAMPLE;        ;IBM MAKROASSEMBLER ESETEN
B>A:LINK SAMPLE;        ;MICROSOFT MAKROASSEMBLER ESETEN
B>A:LINK SAMPLE;        ;TURBO EDITASM ESETEN
```

Feltételezve, hogy a szerkesztés sikeresen lefut, a B meghajtóban létrejön a SAMPLE.EXE file. A program futtatásához – feltételezve, hogy a B az alapértelmezés szerinti meghajtó – a következőket kell beírnunk:

```
B>SAMPLE
```

Amint a program az üzenetet megjeleníti a képernyőn, automatikusan visszatér a DOS-ba, amit a B rendszerüzenettel jelez számunkra.

3. REGISZTEREK, JELZŐBITEK ÉS UTASÍTÁSOK

3.1 A 80286-OS MIKROPROCESSZOR

A 80286-os mikroprocesszornak sok olyan tulajdonsága van, ami kielégíti a többfelhasználós és több feladatot ellátó (multitasking) rendszerek igényeit. A 80286-os az operációs rendszerekre nézve beépített memóriavédelemmel, valamint program és adat titkosítással rendelkezik.

A 10 MHz-cel működő 80286-os képes akár hatszor gyorsabban elvégezni egy-egy feladatot, mint a 4,7...8 MHz-cel működő 8086-os processzor.

A 80286-os a 8086/8088-as szoftverekkel felülről kompatibilis. Valós címzési módban a 80286-os kód kompatibilis a már létrehozott 8086/8088-as tárgykóddal. Virtuális címzési mód esetén a 80286-os forráskód kompatibilis a 8086/8088-as tárgykóddal, de bizonyos módosítások szükségesek ahhoz, hogy a 80286-os által támogatott virtuális címzések előnyeit kihasználhassuk, mint ez a 3–1. ábrán is látható.

3.2 ALAPVETŐ FELÉPÍTÉS

A 80286-os processzornak nyolc 16 bites általános célú regisztere van, amelyek a következők (3–2.ábra):

Az AX, BX, CX, DX regiszterek teljes 16 bites regiszterként használhatók, ill. mindegyik regiszter két 8 bites regiszterre is feloszthető, és így nyolc darab 8 bites regiszterhez juthatunk. Az X betűvel végződő regiszternevek (pl. BX) a teljes 16 bites értéket használják.

A felezéssel képzett 8 bites regiszterekre L, ill. H betűvel lehet hivatkozni, ahol az L az alacsony, a H a magas helyértékű byte-ra vonatkozik (pl. AH és AL).

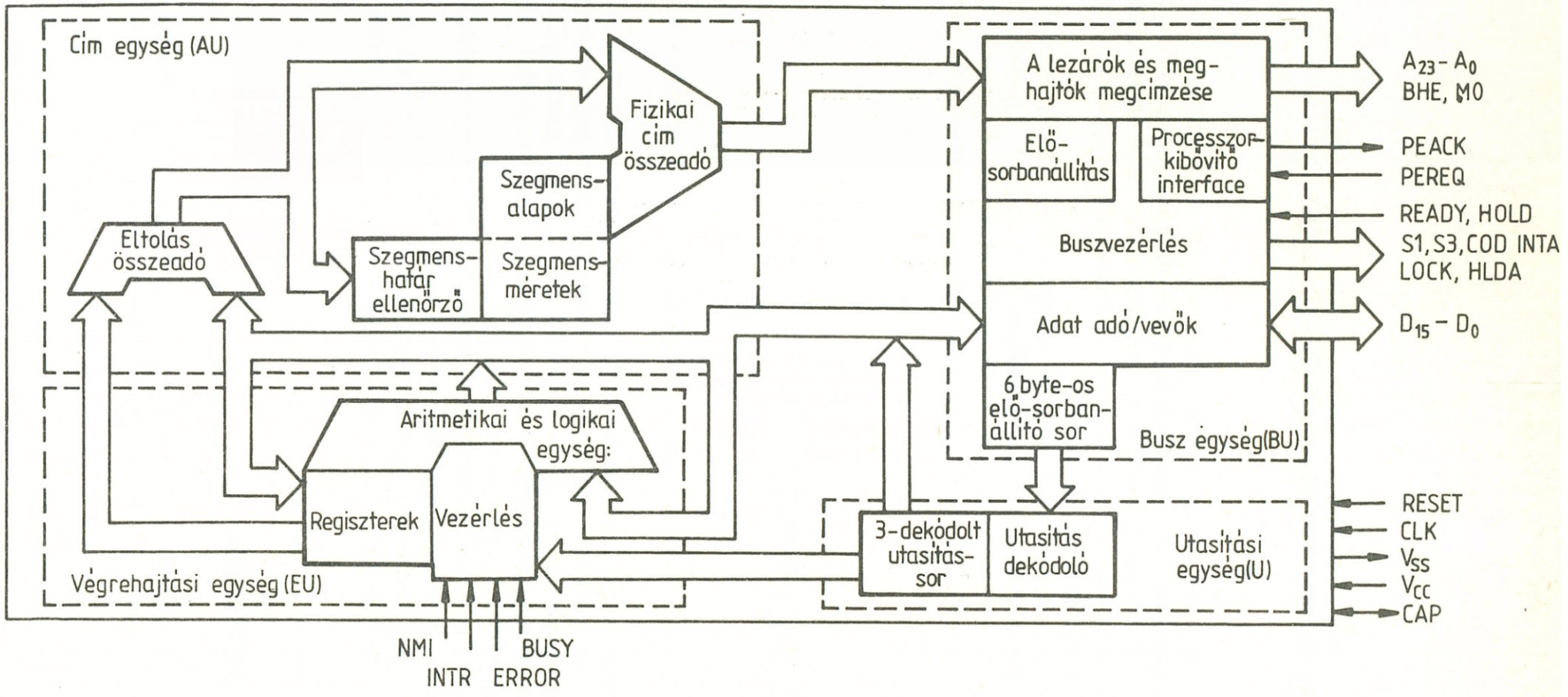
A 16 bites veremmutató (SP) és bázismutató (BP) regiszterpárt a veremmanipulációk során használják, amelyek az éppen érvényes veremhelyzeteket tartalmazzák.

Az SI (forrásindex) és a DI (célindex) regiszterpárt index- regisztereknek is nevezik, és inkrementálható, ill. dekrementálható indexértékekként használják az összetettebb adatszerkezeteken való léptetéskor.

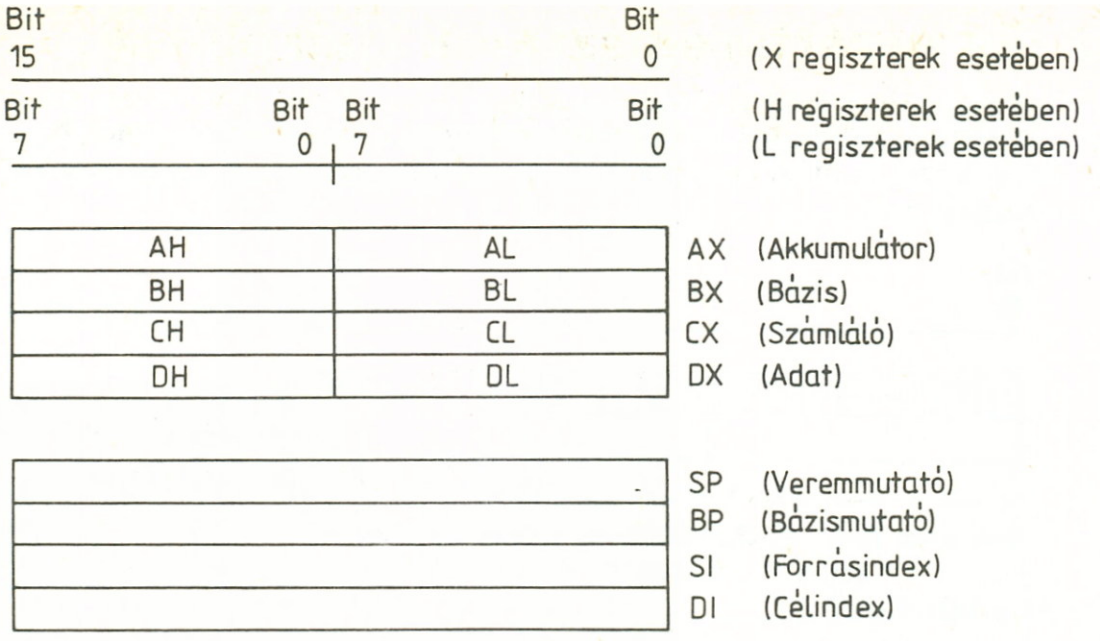
Szegmensregiszterek

A 80286-os négy egyidőben elérhető kódmodult támogat, amelyeket szegmenseknek nevezünk. Ezek a szegmensek a CS, DS, SS, ES 16 bites regiszterekkel címezhetők, mint a 3–3. ábra is mutatja.

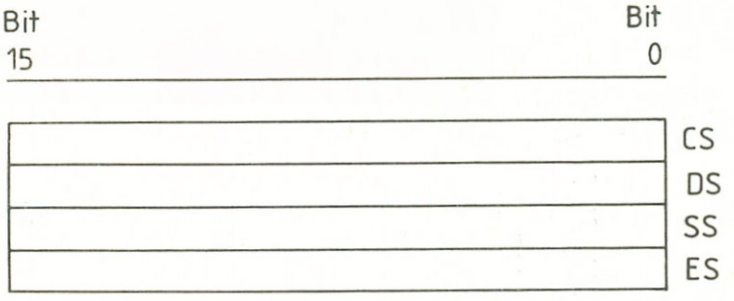
A memóriában elhelyezett éppen futó programot a CS (kódszegmens), az éppen aktív adatszegmens bázisát a DS (adatszegmens) regiszter címezi meg. Azok a vermek, amelyek a közbenső eredményeket és a szubrutin hívásokat tárolják, szintén saját memóriaszegmenssel



3–1. ábra A 80286-os processzor belső blokkdiagramja



3–2. ábra A 80286-os általános célú regiszterei



3–3. ábra A 80286-os szegmensregiszterei

rendelkeznek. Az éppen aktív veremszegmens báziscímét az SS (veremszegmens) regiszter tartalmazza.

A programozónak ezen kívül egy második egyidejűleg aktív adatszegmens – amelyet extraszegmensnek is neveznek – elérésére is lehetősége van, amelyet az ES regiszter címez meg. A szegmensen belüli elem megcímzéséhez először egy aktív szegmenst kell kiválasztanunk, majd a báziscímtől számítva egy 16 bites eltolási értéket kell biztosítanunk. A 16 bites szegmenscím és a 16 bites eltolási érték a 32 bites virtuális címmutató alacsony és magas helyértékű részét alkotják. Miután egy szegmenst kiválasztottunk, az utasításon belül csak a 16 bites eltolási értéket kell megadnunk.

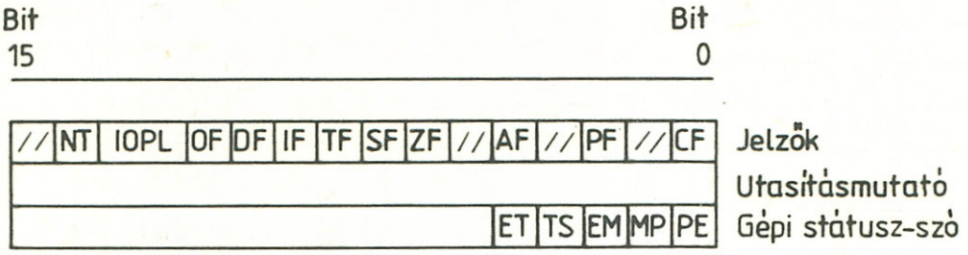
A szegmenscímeket – attól függően, hogy a 80286-os milyen módban működik – különbözően kell értelmezni. A valós címzési módnál a szegmensregiszterek aktuális fizikai címeket tartalmaznak. A védett címzési módnál a szegmensregiszterek virtuális memóriacímeket tartalmaznak, amelyeket át kell alakítanunk ahhoz, hogy fizikai memóriacímeket kapjunk.

Index-, mutató- és bázisregiszterek

Mint korábban már említettük, a kiválasztott szegmensen belül bármelyik adott elem fizikai címét a szegmenscím és az eltolási érték összegeként kapjuk. Az eltolást bármelyik mutató-, bázis- vagy indexregiszterben tárolhatjuk. A veremműveleteket a veremszegmens szelektor (SS) és a veremmutató (SP) vagy a bázismutató (BP) regiszterpár segíti. Az adatszegmensen belüli eltolási érték a BX regiszterből származik. sszetettebb adatmanipulációkat a forrásindex (SI) és a célindex (DI) használatával tudunk megvalósítani.

Státusz- és vezérlőregiszterek (jelzőbitek)

A 80286-os rendelkezik jelzőbiteket tartalmazó regiszterrel is (3–4.ábra), amelyben 11 jelzőmező található. Ezek közül 6-ot állapotjelzőnek nevezünk, amelyeket az arimetikai és logikai döntések változtatnak meg.



3–4. ábra A 80286-os státusz- és kontrollregiszterei

Ez a hat jelzőbit a következő:

CF (CARRY FLAG = átviteljelző bit)
PF (PARITY FLAG = paritásjelző bit)
AF (AUXILIARY CARRY FLAG = segédátviteljelző bit)
ZF (ZERO FLAG = zérusjelző bit)
SF (SIGN FLAG = előjeljelző bit)
OF (OVERFLOW FLAG = túlcsordulásjelző bit)

- A CF 1-re áll be akkor, ha egy 8 vagy 16 bites operanduson elvégzett aritmetikai művelet során átvitel vagy áthozat keletkezik. Egyébként az értéke zérus. A CF a bitléptető és -forgató utasítások során is használatba kerül.
- PF-re elsősorban az adatkommunikációs alkalmazások során van szükség. Értéke páros paritás esetén zérus, páratlan paritás esetén 1.
- Az AF a BCD (bináisan kódolt decimális) aritmetikában használatos, amikor is azt jelzi, hogy a BCD szám legkisebb helyértékű 4 bitjén történt-e áthozat vagy átvitel.
- A ZF 1-re áll be akkor, ha valamely eredmény értéke zérus.
- Az SF negatív eredmény esetén 1-re, pozitív eredmény esetén 0-ra áll be.
- Az OF azt jelzi, hogy egy matematikai művelet eredménye meghaladja a kiszabott tartományt.

A 11 jelzőbit közül három – TF, IF és DF – bizonyos processzorműveletek irányítására használható.

Ha a TF (TRAP FLAG = megszakításjelző bit) be van állítva, akkor a processzor egylépéses üzemmódba kerül, ami lehetővé teszi a hibakeresést és a nyomkövetést.

Az IF (INTERRUPT ENABLE FLAG = megszakítás-engedélyezési jelzőbit) külső megszakításokat tesz lehetővé ha értéke 1, egyébként pedig letiltja a külső megszakításokat.

A karakterlánc-műveleteket a DF (DIRECTION FLAG = irányjelző bit) irányítja. Ha DF 0, akkor SI és/vagy DI értéke automatikusan nő. Ha DF értéke 1, akkor SI és/vagy DI értéke automatikusan csökken.

Az IOPL és NT jelzőbitekről eddig nem tettünk említést, ezek ui. csak a processzor védett üzemmódjában használatosak.

A kétbites IOPL (INPUT OUTPUT PRIVILEGE LEVEL FLAG = input/output kiváltságos szintjelző) azt garantálja, hogy egy utasítás csak olyan műveleteket hajtson végre, amelyek engedélyezve vannak.

Az NT (NESTED TAG FLAG = egymásba ágyazott toldalékjelző bit) azt jelzi, ha az éppen végrehajtott feladat egy másikba ágyazódik.

Ha NT értéke 1, akkor az éppen végrehajtott feladat kapcsolatban van a megelőző feladattal.

Utasításmutató (IP)

Az IP azt az eltolási értéket tartalmazza, ami az éppen aktív kódszegmensen belül a következő utasítás megcímzéséhez szükséges (l. a 3–4. ábrát). Az IP a szekvenciálisan következő utasításhoz egy teljes 32 bites mutatót generál.

A gépi státusz-szó

A 16 bites gépi státusz-szó első 5 bitje (l. a 3–4. ábrát):

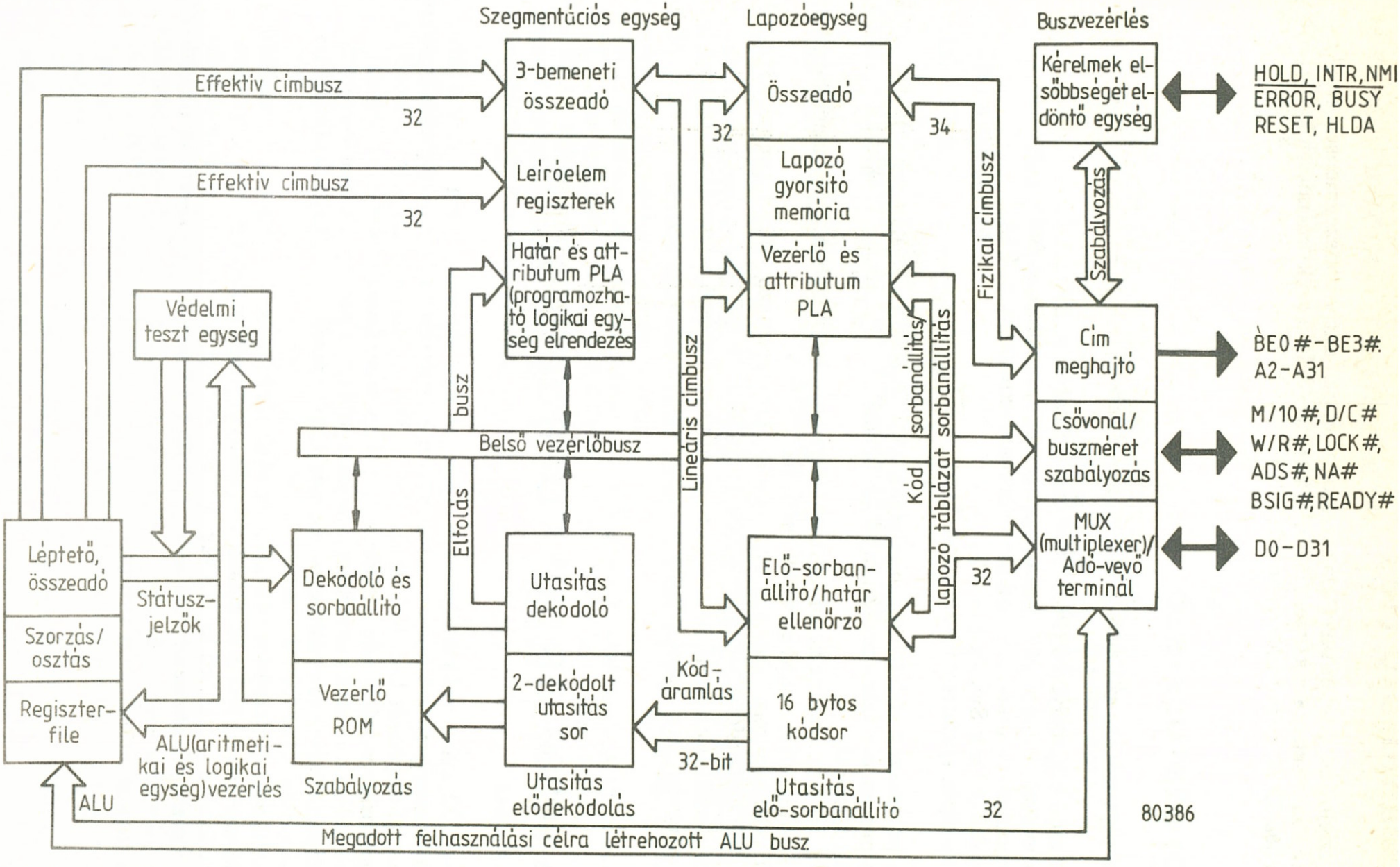
0 – védett mód engedélyezés (Protected mode Enable = PE)
1 – társprocesszor felügyelet (Monitor processor Extension = MP)
2 – társprocesszor emuláció (Emulate processor extension = EM)
3 – taszkváltás (Task Swiched = TS)
4 – bővítési típus (processor Extension Type = ET)

- A PE a processzor védett üzemmódjának aktivizálására használható. Ha PE értéke zérus, akkor a processzor valós címzési módban üzemel. Ha PE értéke 1, akkor a védett üzemmód aktivizált állapotú.
- Az MP a TS bittel együtt kerül felhasználásra annak érdekében, hogy megállapítható legyen a WAIT műveletkód által generált hiba, ha TS = 1.
- Az EM beállításának hatására valamennyi társprocesszor hibát generál. Ha EM zérus, valamennyi társprocesszor műveletkód végrehajtásra kerül.
- A TS automatikusan beáll, ha taszkváltás művelet hajtódik végre. Ha TS értéke 1, a társprocesszor műveleti kód hibát generál.

3.3 A 80386-OS MIKROPROCESSZOR

A 80386-os 32 bites mikroprocesszor, amelyet úgy terveztek meg, hogy a több feladatot párhuzamosan végrehajtó operációs rendszereket, valamint a 32 bites címeket és adattípusokat támogassa. (Felépítését l. a 3–5. ábrán). A 80386-os mikroprocesszor maximálisan 4 Gbyte fizikai memória és 64 Tbyte virtuális memória megcímzését képes elvégezni. Az integrált memóriakezelés és a védelmi szerkezet magába foglalja a parancsértelmező regisztereket, és egy védelmi mechanizmus támogatja az operációs rendszereket és a fejlett többfelhasználós hardvert.

Az utasítások végrehajtásának átlagos idejét jelentősen csökkenti a buszsávok megnövelt szélessége, a chipen belüli memóriacím-kezelés (memory managament), és az utasítás csővonal (pipeline). Ezek a szerkezeti sajátosságok a 80386-os processzort másodpercenként 3...4 millió utasítás végrehajtására teszik képessé. További jellemzői az önteszt, az oldalfordító cashe-memória közvetlen elérése és négy új töréspont regiszter. A 80386-os a 8086/8088/80286-os processzorokkal felülről kompatibilis.



3–5. ábra A 80386-os 32 bites szerkezete

Adattípusok

A 80386-os processzor a 8086/80286-os processzorok által támogatott adattípusokon kívül még számos adattípust támogat. Például: 32 bites előjeles és előjel nélküli egészek, 1–32 bit hosszúságú bitmezők. A 80386-os a 8086/80286-os családban szereplő szabványos mutatótípusokat, valamint egy 32 bites kizárólagos eltolás- mutatót és egy 48 bites teljes mutatót is támogat.

Az operandusok címzése

A 16 bites azonnali operandusméreteken kívül (amelyeket a 8086/80286/80386-os támogat) a 80386-os egy 32 bites operandus- méretet is támogat, amit a 16 bites azonnali címzésű operandus 32 bitre történő kiegészítésével képez.

Ha 32 bites operandusméretet alkalmazunk, az ENTER és RET utasítások 16 bites azonnali címzésű operandusokat igényelnek, amelyekben a 32 bitre való kiegészítés nullákkal történik.

Az effektív cím számítása

64 K-nál nagyobb szegmensek elérésekor az effektív címek 16 és 32 bitesek egyaránt lehetnek. Az effektív címet az opcionális bázis, ill. az indexregiszterek és az opcionális eltolás összeadásával kaphatjuk meg. Ezen kívül a 32 bites címzési módnál bármelyik általános célú regiszter betöltheti a bázis- vagy az indexregiszter szerepét.

Kódvégrehajtás a 80386-os processzor esetében

A 80386-os proceszor két olyan működési móddal rendelkezik, amelyek a 8086/80286-os utasításkészletével bit szinten kompatibilisek. A 8086-os tárgykód végrehajtása is biztosított. A valós üzemmódot a mikroprocesszor a RESET után használja. A virtuális 8086-os mód a 80386-os védett üzemmódjának egy részhalmazát képezi, amely által lehetővé válik, hogy a 8086-os kód a 80386-os védett üzemmódjában és a lapozós operációs környezetében működőképes legyen.

3.4 ALAPVETŐ FELÉPÍTÉS

A 80386-os processzor esetében a programozó 32 regisztert használhat, amelyek hat fő kategóriába sorolhatók:

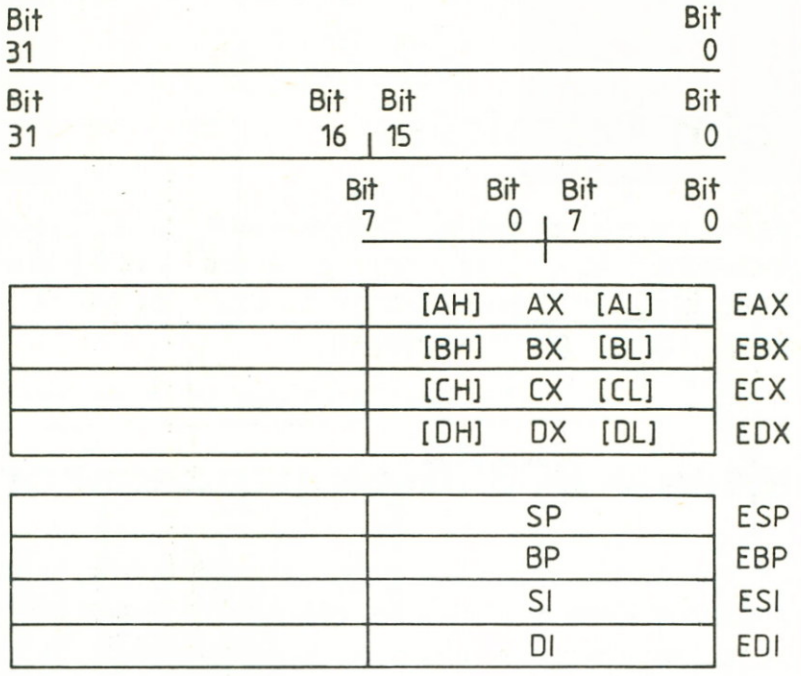
Általános célú regiszterek
Szegmensregiszterek
Utasításmutató és az EFLAGS regiszter
Kontrollregiszterek

Rendszercím-regiszterek
Tesztregiszterek

A 32 bites 80386-os mikroprocesszor valamennyi 16-bites 8086-os és 80286-os regisztert tartalmazza.

Általános célú regiszterek

A nyolc általános célú regiszter (3–6. ábra) teljesen hasonló módon használható, mint a 80286-os általános célú regiszterei, azonban most a regiszterek 32 bitesek. (A 80286-os mikroprocesszor regisztereinek ismertetése ennek a fejezetnek az elején található.) Az általános célú regiszterek az 1, 8, 16 és 32 bites adatoperandusokat, az 1–32 bites mezőket és 16/32 bites címoperandusokat támogatják.



3–6. ábra A 80386-os általános célú regiszterei

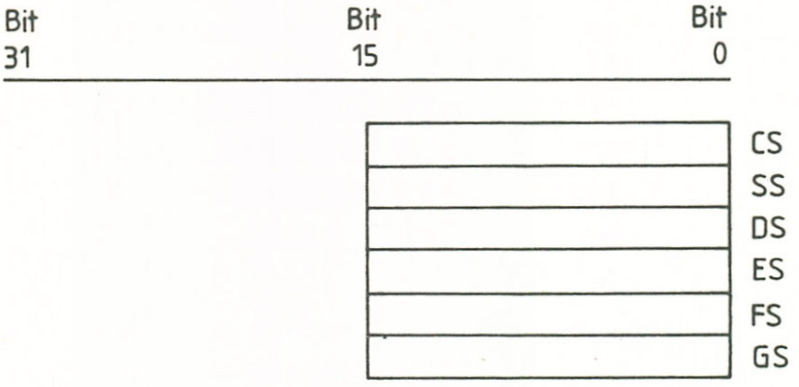
A nyolc regiszter a következő:

EAX – Akkumulátor
EBX – Bázis
ECX – Számláló
EDX – Adat
ESP – Veremmutató
EBP – Bázismutató
ESI – Forrásindex
EDI – Célindex

A regiszter valamennyi bitjének eléréséhez a regiszter-hivatkozásoknak az E betűvel kell kezdődniük. Ha a regiszterekre az E betű nélkül hivatkozunk, a 8086/80286-os 16 bites megfelelőit kapjuk.

Szegmensregiszterek

Mint a 3–7. ábrán látható, a 80386-os mikroprocesszor hat darab 16 bites szegmensregisztert tartalmaz. A szegmensregiszterek az éppen megcímezhető memóriahelyek szelektor értékeit tartalmazzák.



3–7. ábra A 80386-os szegmensregiszterei

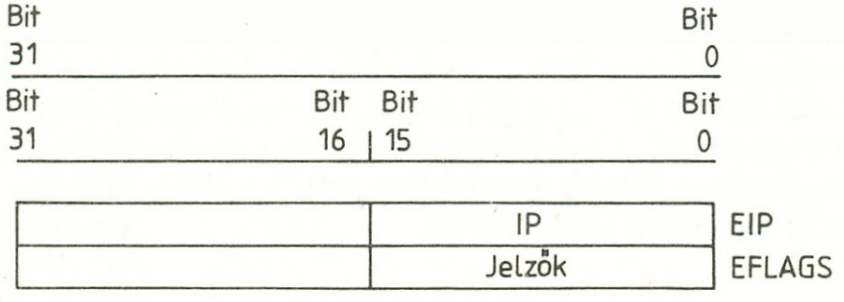
A valós címzési módban a szegmens mérete 1 byte-tól a maximális szegmensméretig, 64 kbyte-ig változhat. A védett címzési módban a szegmens mérete 1 byte-tól maximálisan 4 Gbyte-ig változhat. A hat szegmensregiszter a következő:

CS – Kódszegmens
DS – Adatszegmens
SS – Veremszegmens
ES – Extraszegmens
FS, GS – A megfelelő rövidítés beazonosítása a programozó feladata.

Az FS és GS regiszterek célja az ES regiszter zsúfoltságának csökkentése, valamint az általános regiszterkészletben szereplő bázis- és indexregiszterek minél jobb összeillesztése.

Utasításmutató és az EFLAGS regiszter

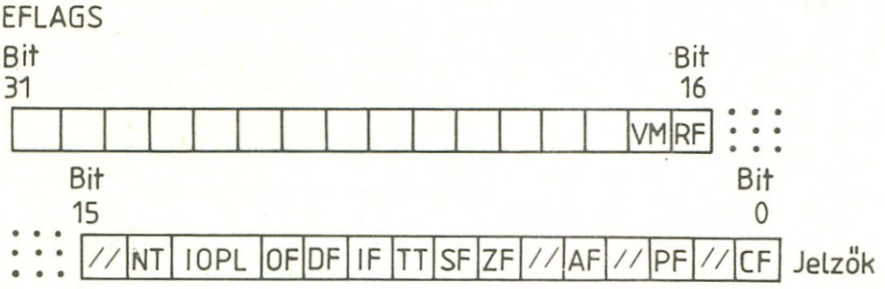
A 80386-os mikroprocesszor egy 32 bites utasítás-mutató regisztert (EIP) tartalmaz (3–8. ábra). Az EIP regiszter a következő végrehajtandó utasítás eltolási értékét tárolja, ami mindig az éppen aktív CS (kódszegmens) bázisától számítandó. Az EIP alacsony helyértékű 16 bitje külön is elérhető. Ezt nevezzük IP regiszternek, és a 16 bites címzésben használjuk.



3–8. ábra A 80386-os utasításmutatói és EFLAGS regisztere

A 80386-os processzor 32 bitre kibővített EFLAGS regisztere a 3–9. ábrán látható. Az EFLAGS regiszter bizonyos műveletek irányítására használható, valamint magának a 80386-osnak az állapotát is jelzi. Az EFLAGS regiszter a korábbiakhoz képest két új jelzőt tartalmaz:

VM (Virtual 8086 Mode flag) – virtuális 8086 mód jelző. RF (Resume Flag) – újrakezdés jelző.



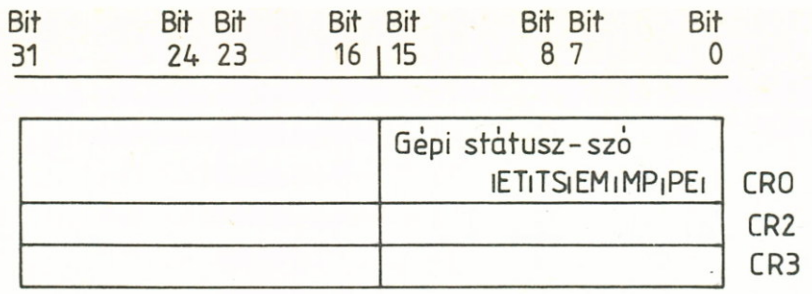
3–9. ábra A 80386-os EFLAGS regiszterei

Az EFLAGS regiszter alacsony helyértékű 16 bitjét FLAGS regiszternek nevezzük, és ugyanazokat a műveleti vezérlő- és státuszjelzőket tartalmazza, mint amiket a 8086/80286-os mikroprocesszoroknál már megismerhettünk. További információkat ennek a fejezetnek az elején találhatunk a FLAGS regiszterről. Az EFLAGS VM jelzője a virtuális 8086-os üzemmód beállítását teszi lehetővé. Ha VM értéke 1 és a 80386-os védett módban van, a mikroprocesszor virtuális 8086-os üzemmódba kapcsol át, amelynek következtében valamennyi szegmensműveletet úgy hajt végre, mintha 8086-os futásról lenne szó. A 8086-os emulációban a 80386-os a 13-as kizáráshibát (exception-13) állítja elő a privilégizált műveletkódokra. Az RF jelző szintén az EFLAGS regiszterben található, és a nyomkövető regiszter töréspontjaival kapcsolatban, vagy az egyesével léptetéskor kerül felhasználásra. Ha RF értéke 1, a következő utasításban valamennyi nyomkövetési hiba figyelmen kívül marad. Az RF minden egyes utasítás sikeres végrehajtásakor automatikusan zérussá válik.

Kontrollregiszterek

A 80386-os mikroprocesszor három 32 bites kontrollregisztert tartalmaz – CR0, CR2, CR3 – amelyek a 3–10. ábrán láthatók. Ezek a regiszterek a számítógép feladattól független állapotáról tartalmaznak információkat. A kontrollregiszterek elérése betöltő és tároló utasításokon keresztül valósítható meg. A CR0 hat darab előre definiált jelzőt tartalmaz, amelyek a mikroprocesszor kontroll és státusz céljaira kerülnek felhasználásra. A CR0 regiszter 0–15 bitjei MSW-ként (Machine Status Word = gépi státusz- szó) is ismertek, amely által a védett üzemmódban a 80386-os kompatibilissá válik a 80286-ossal. Az LMSW és SMSW utasítások egyformán működnek mind a 80286-os, mind a 80386-os esetében. A CR0 eléréséhez a programozónak a MOV CR0 utasítást kell használnia. Az ET, a TS, az EM, az MP és a PE bitek ugyanúgy működnek, mint a 80286-os esetében (részletesen l. a fejezet elején).

A CR1, amelyet külön nem ábrázoltunk, a későbbiekben kifejlesztendő Intel mikroprocesszorok használatára van fenntartva. A CR2 azt a 32 bites lineáris címet tartalmazza, ami az utolsó felderített laphibát okozta. A CR3 a lapkatalógus fizikai báziscímét tárolja. Mivel a 80386-os lapkatalógus táblázat mindig lapkiegyenlített, a CR3 legalacsonyabb helyértékű 12 bitjét a rendszer mindig figyelmen kívül hagyja, ha ezekbe beírás történik.



3–10. ábra A 80386-os kontrollregiszterei

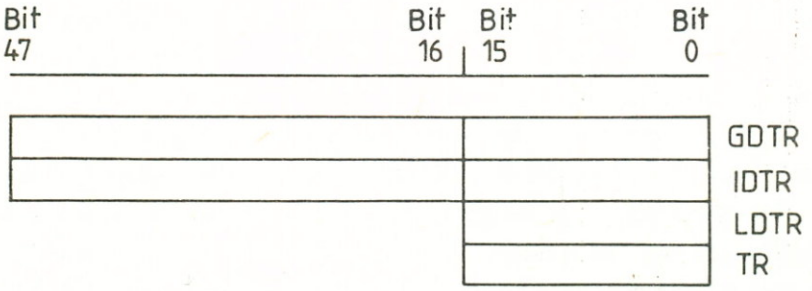
Rendszercím-regiszterek

A 80386-os mikroprocesszor 4 speciális célú regisztert tartalmaz (l. a 3–11. ábrát), amelyek a 80286/80386-os védett üzemmód által támogatott szegmensekhez vagy táblázatokhoz használhatók. A GDT (Global Descriptor Table = globális leírótáblázat), az IDT (Interrupt Descriptor Table = megszakításleíró táblázat), az LDT (Local Descriptor Table = lokális leírótáblázat) és a TSS (Task State Segment table = feladat állapot szegmens táblázat) címeit ezek a speciális regiszterek tárolják. A regiszterek elnevezése a következő:

GDTR, IDTR, LDTR, TR.

A 32 bites GDTR és IDTR regiszterek a 32 bites lineáris báziscímet és a GDT és IDT 16 bites szegmenshatárát tartalmazzák.

Az LDTR és TR 16 bites regiszterek tárolják az LDT és TSS szegmensek 16 bites szelektorát.



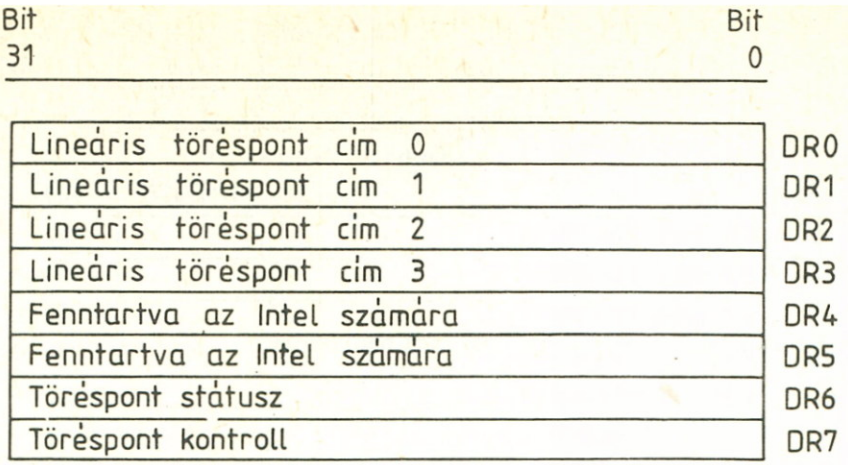
3–11. ábra A 80386-os rendszercím regiszterei

Nyomkövető és tesztregiszterek

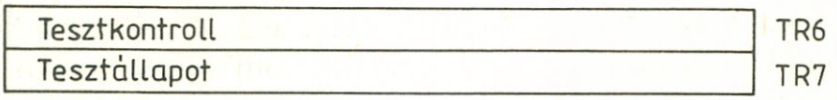
A 80386-os processzor hat, egyenként 32 bites nyomkövető regiszterrel – DR0, DR1, DR2, DR3, DR6, DR7 – (l. a 3–12. ábrát) rendelkezik. A DR4 és a DR5 regisztert az Intel a saját céljaira tarja fenn. A DR6 nyomkövető vezérlőregiszter a töréspontok beállítására használható. A DR7 a töréspontok éppen érvényes állapotának jelzését szolgálja.

A 80386-os mikroprocesszor két 32 bites tesztregisztert is tartalmaz (l. a 3–13. ábrát), amelyek a RAM és a CAM (asszociatív memória = tartalma alapján címezhető memória) ellenőrzését irányítják a fordításfigyelő (translation lookaside) segédtárolóban.

A TR6 egy parancsellenőrző regiszter, a TR7 pedig mint adatregiszter funkcionál a fordításfigyelő segédtároló ellenőrzésével kapcsolatban.



3–12. ábra A 80386-os hibakereső regiszterei



3–13. ábra A 80386-os tesztregiszterei

3.5 A 80286/80386-OS UTASÍTÁSKÉSZLETE

AAA (80286/80386)

Az AL regiszter ASCII kiigazítása összeadás után (ASCII Adjust AL after Addition)
Utasítás: AAA
Tipikus órajel: (80286) 3, (80386) 4
Az utasítás leírása: Az AL regiszter összeadás utáni kiigazítása
Az utasítás működése: Az AAA utasítás kizárólag az ADD utasítás után következhet, ami az AL regiszterben byte eredményt állít elő. Az AAA utasítás az AL tartalmát pakolatlan decimális számmá alakítja át, és megvizsgálja AL alacsony helyértékű négybites szavát, hogy az a 0–9 tartományba eső érvényes BCD számot tartalmazza-e. Az AAA utasítás az AL magas helyértékű négybites szavát beállítja zérusra. Decimális átvitel esetén az AF és CF jelzők ugyancsak zérusra állnak be. Ha az alacsony helyértékű négybites szóban tárolt érték nagyobb mint 9, vagy ha az AF értéke 1, akkor AAA a következő műveleteket végzi el: Az AL regiszter értékét 6-tal, AH értékét 1-gyel növeli, az AF és CF jelzőket 1-re állítja be, AL felső négy bitjét pedig 0-ra.
Szintaxis: AAA (operandus nélkül)
Befolyásolt jelzők: AF, CF
Definiálatlan jelzők: OF, ZF, SF, PF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:

```
ADD     AL,BL       ;Az AL-ben és BL-ben tárol BCD számok
                    összeadása
AAA                 ;Pakolatlan eredmény képzése
```

AAD (80286/80386)

Az AX regiszter ASCII kiigazítása osztáshoz. (ASCII Adjust AX before Division)
Utasítás: AAD
Tipikus órajel: (80286) 14, (80386) 19
Az utasítás leírása: Az AX regiszter ASCII beállítása osztáshoz
Az utasítás működése: Az AAD utasítás két pakolatlan BCD számot (a alacsony helyértékű számjegyet az AL regiszterben, a magas helyértékű számjegyet az AH regiszterben) felkészíti az osztási műveletre, amelynek eredménye ugyancsak pakolatlan formátumban lesz. AL értékét AL+(10*AH)-ra, AH értékét zérusra állítja be. Az AX regiszterben képződő eredmény az eredeti pakolatlan kétszámjegyes szám bináris megfelelője lesz.
Szintaxis: AAD (operandus nélkül)
Befolyásolt jelzők: SF, ZF, PF
Definiálatlan jelzők: OF, AF, CF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: AAD ;Az osztási művelet előtt használatos

AAM (80286/80386)

Az AX regiszter ASCII kiigazítása szorzás után (ASCII Adjust AX after Multiply)
Utasítás: AAM
Tipikus órajel: (80286) 16, (80386) 17
Az utasítás leírása: Az AX regiszter ASCII kiigazítása szorzás után
Az utasítás működése: Az AAM utasítás két pakolatlan decimális operandus összeszorzásának eredményét, ami az AX regiszterben képződik, kiigazítja. Ezt az utasítást csak a MUL utasítás után lehet kiadni. Mivel az eredmény 100-nál kisebb, az AL regiszterben elfér. Az AAM az AL-ben található eredményt visszaalakítja úgy, hogy AL-t, elosztja 10-zel. A hányadost AL-ben, a maradékot pedig AH-ban helyezi el.
Szintaxis: AAM (operandus nélkül)
Befolyásolt jelzők: SF, ZF, PF
Definiálatlan jelzők: OF, AF, CF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: AAM ;A szorzás után használatos

AAS (80286/80386)

Az AL regiszter ASCII kiigazítása kivonás után. (ASCII Adjust AL after Subtraction)
Utasítás: AAS
Tipikus órajel: (80286) 3, (80386) 4
Az utasítás leírása: Az AL regiszter ASCII kiigazítása kivonás után
Az utasítás működése: Az AAS utasítás célja az, hogy két pakolatlan decimális operandus kivonását követően az AL regiszterben található eredményt korrigálja. Az eredmény pakolatlan decimális formában lesz, ami a következő szabályok használatával valósul meg: ha az AL regiszter alacsony helyértékű négybites szava 9-nél nagyobb, vagy ha az AF jelző értéke 1, az AL regiszter értékét 6-tal, az AH regiszter értékét pedig 1-gyel csökkenti. A CF és AF jelzőket 1-re állítja be, egyébként pedig mind AF-et, mind CF-et zérusra. Ennélfogva AL eredeti értéke egy olyan byte-tal cserélődik ki, amelynek a felső négybites szava zérus, az alsó négybites szava pedig egy olyan szám, ami a 0-9 tartományba esik.
Szintaxis: AAS (operandus nélkül)

Befolyásolt jelzők: AF, CF
Definiálatlan jelzők: OF, SF, ZF, PF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: AAS ;Kivonás után használatos

ADC (80286/80386)

Összeadás átvitellel (ADd with Carry)
Utasítás: ADC
Tipikus órajel: (80286) 2-7, (80386) 2-7
Az utasítás leírása: Két operandus összeadása átvitellel
Az utasítás működése: Az ADC két operandus egész összeadását valósítja meg. Ha CF értéke 1, akkor a két operandus összegéhez az 1 hozzáadódik és az eredmény a céloperandusba kerül vissza.
Szintaxis: ADC *cél,forrás*
Befolyásolt jelzők: OF, SF, ZF, AF, PF, CF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: Azonnali címzésű operandus hozzáadása az akkumlátorhoz átvitellel:

```
        ADC     AL,4
        ADC     AX,298
        ADC     EBX,22334455H       (csak 80386-os esetében)
```

Azonnali címzésű operandus hozzáadása regiszterhez vagy memóriahelyhez átvitellel:

```
        ADC     CX,341
        ADC     BL,10

        ADC     TABLE[SI],2
        ADC     MEMORY,6293
        ADC     NUMBER,12345678     (csak 80386-os esetében)
```

Összeadás átvitellel:

```
                Regisztert          regiszterhez
                Regisztert          memóriához
                Memóriát            regiszterhez
        ADC     DL,BL
        ADC     MEM_WRD,AX
        ADC     SI,MEM_WRD
```

ADD (80286/80386)

Összeadás (ADDition)
Utasítás: ADD
Tipikus órajel: (80286) 2-7, (80386) 2-7
Az utasítás leírása: Két operandus összeadása

Az utasítás működése: Az ADD művelet két operadus összeadását végzi el. Az eredményt a céloperandus helyére írja.
Szintaxis: ADD *cél,forrás*
Befolyásolt jelzők: AF, CF, OF, PF, SF, ZF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: Regisztert regiszterhez:

```
        ADD     AX,BX
        ADD     EBX,ECX             (csak 80386-os esetében)
```

Regisztert memóriához:

```
        ADD     EXTMEM,AX
```

Memóriát regiszterhez:

```
        ADD     DX,BUFF
```

Azonnali címzésű operandust akkumulátorhoz:

```
        ADD     AL,4
        ADD     EAX,98765432H       (csak 80386-os esetében)
```

Azonnali címzésű operandust regiszterhez:

```
        ADD     CX,1985
```

Azonnali címzésű operandust memóriához:

```
        ADD     EXTMEM,23
```

AND (80286/80386)

Logikai ÉS művelet (AND)
Utasítás: AND
Tipikus órajel: (80286) 2-7, (80386) 2-7
Az utasítás leírása: Két operandus logikai AND műveletének elvégzése bitről bitre.
Az utasítás működése: Az eredmény azon bitpozíciókban lesz 1, ahol mindkét operandusban 1 szerepelt, egyéb bitkombinációkban pedig zérus. Az eredményt a céloperandus helyére írja. Az OF és CF flag-ek zérusra állnak be.
Szintaxis: AND *cél,forrás*
Befolyásolt jelzők: OF = 0, CF = 0, PF, SF, ZF
Definiálatlan jelzők: AF
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: Azonnali címzésű operandust regiszterhez:

```
        AND     BL,11001110B
        AND     EAX,11110000111100001111000011110000B
                                    (csak 80386-os esetében)
```

Azonnali címzésű operandust memóriához:

```
        AND     EXTMEM,10011100B
        AND     NUMBER,0F0F0F0FH
```

(csak 80386-os esetében)
Azonnali címzésű operandust akkumulátorhoz:

```
        AND     AX,1001111101010110B
        AND     AL,MASK_BYTE
```

Regisztert regiszterhez:

```
        AND     AX,BX
        AND     EAX,ECX                 (csak 80386-os esetében)
```

Regisztert memóriához:

```
        AND     EXTMEM,SI
```

Memóriát regiszterhez:

```
        AND     DH,EXTBYTE
```

ARPL (80286/80386)

A szelektor RPL mezőjének beállítása (Adjust RPL field of selector)
Utasítás: ARPL
Tipikus órajel: (80286) 10-11, (80386) 20-21
Az utasítás leírása: Az EA szó RPL mezőjének beállítása
Az utasítás működése: Az ARPL utasítás feladata az, hogy meghiúsítsa az operációs rendszer szoftverének hozzáférését olyan szubrutinokhoz, amelyek magasabb prioritási szinten vannak. Az ARPL két operandusa közül az első egy 16 bites memóriaváltozó vagy regiszter, ami a szelektor értékét tartalmazza. A második operandus egy 16 bites regiszter. Ha az első operandus RPL mezője (alsó két bit) kisebb mint a második operandus RPL mezője, akkor a zérusjelző bit 1-re állítódik be. Az első operandus értéke nő, hogy megegyezzen a második RPL-lel. Egyébként a zérusjelző bit értéke zérusra áll be, és az első operandusban semmilyen változás nem történik.
Szintaxis: ARPL *(szelektor,CS_szelektor)*
Befolyásolt jelzők: ZF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: INT 6 áll elő. Az ARPL utasítást nem ismeri fel a rendszer.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: ARPL MEM_WRD,BX

BOUND (80286/80386)

Tömbök indexértékeinek ellenőrzése
Utasítás: BOUND
Tipikus órajel: (80286) 13, (80386) 10
Az utasítás leírása: INT 5 következik be, ha a szóregiszter nincs a határokon belül
Az utasítás működése: A BOUND művelettel az biztosítható, hogy egy tömbindex a memória kétszavas blokkja által definiált határok közé essen. Az első operandusnak (regiszter) nagyobbnak vagy egyenlőnek kell lennie, mint a memória első, és kisebbnek, vagy egyenlőnek, mint a

memória második szava. Ha ezek a feltételek nem teljesülnek, az 5-ös megszakítás következik be.
Szintaxis: BOUND *cél, forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a határérték-teszt hamis eredményt ad, INT 5 áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a határérték-vizsgálat hamis, INT 5 áll elő. Ha a második operandus a 0FFFDH-nál vagy ennél nagyobb eltolásnál van, INT 13 áll elő. INT 6 áll elő, ha a második operandus egy regiszter.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

```
Példa:  BUOND   AX, MEM_WORD
        BOUND   EBX,NEW_WORD      (csak 80386-os esetében)
```

CALL (80286/80386)

Szubrutin hívás (CALL)
Utasítás: CALL
Tipikus órajel: (80286) 7-185, (80386) 3-275
Az utasítás leírása: Tárolja a következő utasítás címét és átadja a vezérlést.
Az utasítás működése: A CALL hatására a következő utasítás címe a veremre kerül, majd a program vezérlését a paraméter- operandus kapja meg. Amikor a meghívott eljárás befejeződik, a hívó program végrehajtása a CALL után folytatódik.
Szintaxis: CALL *paraméter_operandus*
Befolyásolt jelzők: Nincs (kivéve ha feladatkapcsolás történik)
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: FAR (távoli) szubrutinhívás: általános védelmi kizárás, leíróelem nincs jelen kizárás, veremhiba kizárás és érvénytelen taszk állapotszegmens áll elő. NEAR (közeli) közvetlen szubrutinhívás: általános védelmi kizárás keletkezik, ha az eljárás helye a kódszegmens határain túl van. NEAR (közeli) közvetett szubrutinhívás: Ha a CS, DS vagy ES regiszter illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő. Ha az indirekt eltolás a kódszegmens határain kívül van, általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szóopereandusok 32, a mutatóoperandusok 48 bitesek.

```
Példa:  CALL    ADDER               ;Szegmensek közötti vagy szegmensen belüli
                                    szubrutin hívás
80386:
        CALL    IMMPTR              ;A mutatóoperandusok 48 bitesek
        CALL    DISP16              ;32 bites kibővített utasításmutató
        CALL    regiszter vagy memória
                                    ;Az eltolás 32 bites. A szóoperandusok 32
                                    bitesek
```

CBW (80286/80386)

Byte konvertálása szóvá (Converts Byte into Word)

Utasítás: CBW
Tipikus órajel: (80286) 2, (80386) 3
Az utasítás leírása: Byte konvertálása szóvá (AH = AL felső bitje)
Az utasítás működése: A CBW művelet az AL-ben található előjeles byte-ot előjeles szóvá alakítja az AX regiszterben. A művelet úgy valósul meg, hogy AL legnagyobb helyértékű bitje AH valamennyi bitjére kiterjed. L.a 80386-nál levő megjegyzést.
Szintaxis: CBW (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A szóméret 32 bites. Az előjellel bővített érték AX-ben van, az eredmény pedig EAX-ben.
Példa: CBW

CLC (80286/80386)

Az átviteljelző bit (CF) törlése (CLear Carry flag)
Utasítás: CLC
Tipikus órajel: 2
Az utasítás leírása: Az átviteljelző bit törlése
Az utasítás működése: A CLC művelet az átviteljelző bit értékét zérusra állítja be. Egyéb regiszterekre vagy jelzőkre nincs hatással.
Szintaxis: CLC (operandus nélkül)
Befolyásolt jelzők: CF = 0
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: CLC

CLD (80286/80386)

Az irányjelző bit törlése (CLear Direction flag)
Utasítás: CLD
Tipikus órajel: 2
Az utasítás leírása: Az irányjelző bit törlése
Az utasítás működése: A CLD törli az irányjelző bitet. Egyéb regiszterekre vagy jelzőkre nincs hatással. A CLD utasítás végrehajtása után a karakterlánc műveletek automatikusan növelik az indexregiszterek (SI és/vagy DI) értékét.
Szintaxis: CLD (operandus nélkül)
Befolyásolt jelzők: DF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: CLD

CLI (80286/80386)

A megszakítási jelzőbit (IF) törlése (CLear Interrupt)

Utasítás: CLI
Tipikus órajel: 3
Az utasítás leírása: Törli a megszakítási jelzőbitet és letiltja a megszakításokat.
Az utasítás működése: A CLI utasítás törli a megszakítást engedélyező jelzőbitet, amelynek hatásaként valamennyi megszakítást letiltja, kivéve a nem maszkolható megszakításokat, amelyek az NMI vonalon következnek be. Az utasítás egyéb jelzőkre nincs hatással.
Szintaxis: CLI (operandus nélkül)
Befolyásolt jelzők: IF = 0
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az éppen érvényes privilégizált szint magasabb, mint az IOPL a jelzőket tartalmazó regiszterben, általános védelmi kizárás áll elő. Az IOPL azt a legkevésbé privilégizált szintet jelzi, amelynél az I/O végrehajtható.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: CLI

CLTS (80286/80386)

A taszkváltásjelző bit törlése (CLears Task Switched flag)
Utasítás: CLTS
Tipikus órajel: (80286) 2, (80386) 5
Az utasítás leírása: Törli a taszkváltásjelző bitet.
Az utasítás működése: A CLTS törli a taszkváltás- jelző bitet a gépi státusz-szóban. Ez egy privilégizált utasítás, ami csak a 0-s szinten hajtható végre, és az operációs rendszer szoftverének van fenntartva! A CLTS minden WAIT vagy ESC végrehajtását nyomon követi, és az MSW MP jelzőjének és a taszkváltás jelzőjének beállásakor megszakad.
Szintaxis: CLTS (operandus nélkül)
Befolyásolt jelzők: TS = 0
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az éppen érvényes privilégizált szint 0-tól eltérő értéket tartalmaz, általános védelmi kizárás áll elő a CLTS utasítás végrahajtásakor.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: CLTS

CMC (80286/80386)

Az átviteljelző bit komplemensének képzése (CoMplement Carry flag)
Utasítás: CMC
Tipikus órajel: 2
Az utasítás leírása: Az átviteljelző bit komplemensét képezi
Az utasítás működése: Az utasítás átkapcsoja az átviteljelző bitet, tehát ha annak eredeti értéke 1 volt, az utasítás hatására zérus lesz és fordítva.
Szintaxis: CMC
Befolyásolt jelzők: CF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: CMC

CMP (80286/80386)

Két operandus összehasonlítása (CoMPare)
Utasítás: CMP
Tipikus órajel: (80286) 2-7, (80386) 2-6
Az utasítás leírása: Kivonja az operandusokat és befolyásolja a jelzőket
Az utasítás működése: A CMP utasítás kivonja a forrásoperandust a céloperandusból. Az operandusok tartalma változatlanul marad, azonban a jelzők megváltozhatnak. A forrás- és céloperandusnak azonos típusúnak kell lennie. Ez alól csak az azonnali címzési mód kizárás, ebben az esetben ui. egy előjelkiterjesztett azonnali címzésű adatbyte és egy memóriaszó összehasonlítása történik meg.
Szintaxis: CMP *cél, forrás*
Befolyásolt jelzők: OF, SF, ZF, AF, PF, CF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: Azonnali címzésű operandus összehasonlítása a memóriával:

```
        CMP     EXTMEM,5CAFH
```

Azonnali címzésű operandus összehasonlítása a regiszterrel:

```
        CMP     BL,5
        CMP     EAX,0FFFF0000H      (csak 80386-os esetében)
```

Azonnali címzésű operandus összehasonlítása az akkumulátorral:

```
        CMP     AL,7
```

Regiszter összehasonlítása regiszterrel:

```
        CMP     AX,BX
        CMP     EDX,EAX             (csak 80386-os esetében)
```

Regiszter összehasonlítása memóriával:

```
        CMP     EXTMEM,DX
```

Memória összehasonlítása regiszterrel:

```
        CMP     CH,EXTMEM
        CMP     EBX,VALUE           (csak 80386-os esetében)
```

CMPS/CMPSB/CMPSW (80286/80386)

Karaterlánc-operandusok összehasonlítása (CoMPare String operands)
Utasítás: CMPS/SB/SW
Tipikus órajel: (80286) 8, (80386) 10
Az utasítás leírása: DS:[SI] által megcímzett byte-okat/szavakat összehasonlítja az ES:[DI] által megcímzettel, vagy a DS:[ESI] által megcímzettet az ES:[EDI] által megcímzettel
Az utasítás működése: A CMPS/SB/SW utasítás összehasonlítja az SI (ESI) regiszter által megcímzett memóriahely tartalmát a DI (EDI) regiszter által megcímzett memóriahely tartalmával. A CMPS utasítás kivonja a DI (EDI) regiszterrel kijelölt memóriahely tartalmát az SI (ESI) regiszter által kijelölt memóriahely tartalmából. Az eredmény beállítja a jelzőket, de egyik memóriahely tartalmát sem változtatja meg. Az SI (ESI) és DI (EDI) regiszterek értéke a DF flag értékének megfelelően nő vagy csökken. Ha DF zérus, SI (ESI) és DI (EDI) értéke nő. Ha DF értéke 1, SI (ESI) és DI (EDI) értéke csökken. Mind byte, mind szó összehasonlítása

előírható. SI (ESI) és DI (EDI) értéke byte-karakterlánc esetén 1-gyel, szókarakterlánc esetén 2-vel nő.
Megjegyzés: A CMPS jobb oldali argumentuma egy olyan operandus, amit a DI regiszter indexel és az ES regiszter címez meg. Ez az alapértelmezés nem írható felül.
Szintaxis: CMPS *forrás_karakterlánc, cél_karakterlánc*
CMPSB (operandus nélkül)
CMPSW (operandus nélkül)
Befolyásolt jelzők: CF, AF, PF, OF, SF, ZF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. Használjuk a CDQ utasítást.

```
Példa:  MOV     SI,OFFSET STRING1
        MOV     DI,OFFSET STRING2
        CMPS    STRING1,STRING2
        CMPS    DS:BYTE PTR[SI],ES:[DI]
        LEA     ESI,STRING1          (csak 80386-os esetében)
        LEA     EDI,STRING2
        CMPS    STRING1,STRING2
```

CWD (80286/80386)

Szó átalakítása duplaszóvá (Convert Word to Doubleword)
Utasítás: CWD
Tipikus órajel: 2
Az utasítás leírása: A szót átalakítja duplaszóvá
Az utasítás működése: A CWD utasítás az AX regiszterben szereplő előjeles szót duplaszóvá alakítja a DX:AX regiszterekben. A művelet végrehajtásakor AX legnagyobb helyértékű bitje DX valamennyi bitpozíciójába kiterjed. L. a 80386-os megjegyzést.
Szintaxis: CWD (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A szóméret 32 bites. Az előjellel kiterjesztett érték EAX-ben van. Az eredményeket az EDX: EAX regiszterpárban tároljuk.
Példa: CWD

DAA (80286/80386)

Decimális kiigazítás összeadás után (Decimal Adjust after Addition)
Utasítás: DAA
Tipikus órajel: (80286) 3, (80386) 4
Az utasítás leírása: AL decimális kiigazítása összeadás után
Az utasítás működése: A DAA utasítást csak két pakolt BCD operandus összeadása után használhatjuk. A DAA utasítás az AL-ben szereplő eredményt pakolt decimális formátummá alakítja a következő szabályok figyelembevételével: Ha AL alacsony helyértékű négybites

csoportja 9-nél nagyobb, vagy ha CF értéke 1, AL értékét 6-tal növeli és AF-et 1-re állítja be, egyébként AF-nek a zérus értéket adja. Ha egy megelőző művelet eredményeképpen AL nagyobb mint 9FH, vagy ha CF értéke 1, AL értékét 60H-val növeli és CF-et beállítja 1-re, egyébként CF-et nullázza.
Szintaxis: DAA (operandus nélkül)
Befolyásolt jelzők: AF, CF, SF, ZF, PF
Definiálatlan jelzők: OF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:

```
MOV     AL,08
ADD     AL,03          ;Az eredmény 0BH
DAA                    ;Az eredmény 11H
```

DAS (80286/80386)

Decimális kiigazítás kivonás után (Decimal Adjust AL after Subtraction)
Utasítás: DAS
Tipikus órajel: (80286) 3, (80386) 4
Az utasítás leírása: A DAS AL-t kivonás után decimális formára hozza
Az utasítás működése: A DAS utasítás csak két pakolt BCD szám kivonása után alkalmazható. A DAS a BCD eredményt AL-be állítja be a következő szabályok szerint: Ha AL alacsony helyértékű négybites csoportjának értéke 9-nél nagyobb, vagy ha AF értéke 1, AL értékét 6-tal csökkenti, és beállítja az AF jelzőbitet, ellenkező esetben AF-et nullázza. Ha egy megelőző művelet eredményeképpen AL nagyobb mint 9FH, vagy ha CF értéke 1, AL-t 60H-val csökkenti és beállítja CF-et 1-re, egyébként CF-et nullázza.
Szintaxis: DAS (operandus nélkül)
Befolyásolt jelzők: AF, CF, SF, PF, ZF
Definiálatlan jelzők: OF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:

```
MOV     AL,12
SUB     AL,03          ;Az AL-ben képződő eredmény 09H
DAS
```

DEC (80286/80386)

Csökkentés (DECrement)
Utasítás: DEC
Tipikus órajel: (80286) 2-7, (80386) 2-6
Az utasítás leírása: Az EA byte/szó regiszter értékét 1-gyel csökkenti
Az utasítás működése: A DEC utasítás az előírt memóriahely vagy regiszter (byte vagy szó) tartalmából 1-et kivon.
Szintaxis: DEC *cél*
Befolyásolt jelzők: SF, OF, ZF, AF, PF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.

Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bit
Példa:
```
DEC     AX
DEC     EXTNEM
DEC     TABLE[BX][SI]
DEC     ECX                 (csak 80386-os esetében)
```

DIV (80286/80386)

Előjel nélküli osztás (unsigned DIVision)
Utasítás: DIV
Tipikus órajel: (80286) 14-25, (80386) 14-41
Az utasítás leírása: AX (byte) előjel nélküli osztását, ill. DX:AX (szó) előjel nélküli osztása
Az utasítás működése: A DIV utasítás előjel nélküli osztást hajt végre. Ha byte méretű forrásoperandust írunk elő, akkor az AX regisztert az operandussal elosztja és a hányadost az AL, a maradékot az AH regiszterben tárolja. Szó forrásoperandus esetén DX:AX-et a szóval osztja. Az osztandó legnagyobb helyértékű 16 bitjét DX, a hányadost AX és a maradékot DX tárolja.
Szintaxis: DIV *forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: SF, AF, OF, ZF, PF, CF
Kizárások védett üzemmód esetén: Ha az osztó zérus, vagy ha a hányados túl nagy ahhoz, hogy beférjen a kijelölt regiszterbe (AX vagy AL), INT 0 áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha az osztó zérus, vagy a hányados túl nagy ahhoz, hogy beférjen a kijelölt regiszterbe (AX vagy AL), INT 0 áll elő. Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő. Duplapontos osztás az EDX:EAX regiszterben elhelyezett 64 bites osztandóval, a regiszterben vagy a memóriában tárolt 32 bites osztóval és az EDX-ben keletkező 32 bites maradékkal valósul meg.
Példa: Byte osztása egy másik byte-tal:

```
        MOV     AL,NUMER_BTE
        DIV     DIVSR_BTE       ;A hányados AL-ben, a maradék AH-ban.
```

Szó osztása byte-tal:

```
        MOV     AL,NUMER_BTE
        DIV     DIVSR_BTE       ;A hányados AL-ben, a maradék AH-ban.
```

Szó osztása duplaszóval:

```
        MOV     AX,NUMER_MSW
        MOV     AX,NUMER_LSW
        DIV     DIVSR_WRO       ;A hányados AX-ben, a maradék DX-ben.
80386:
        MOV     EDX,MSB_NUMBER  ;Csak 80386-os esetében.
        MOV     EAX,LSB_NUMBER
        DIV     ECX             ;A hányados EAX-ben, a maradék EDX-ben.
```

ENTER (80286/80386)

A veremkeret elkészítése az eljárás paraméterei számára
Utasítás: ENTER

Tipikus órajel: (80286) 11-16, (80386) 10-19
Az utasítás leírása: Elkészíti a veremkeretet az eljárás paraméterei számára
Az utasítás működése: Az ENTER utasítást a veremkeret elkészítéséhez kell használnunk, amelyre a legtöbb blokk szerkezetű magas szintű nyelvnek szüksége van. Az első operandus azt írja elő, hogy mennyi dinamikus tárolású byte-ot kell lefoglalni a veremben annak a rutinnak a számára, amelybe belépünk. A második operandus a magas szintű nyelv forráskódján belül a rutinok egymásba ágyazási szintjét adja meg. Az ENTER utasítás meghatározza, hogy mennyi veremkeret mutatót kell bemásolni az új veremkeretbe az előzőből. A BP (EBP) az éppen érvényes veremkeret mutatójaként használható. Ha a második operandus zérus, az ENTER a BP-t (EBP-t) ráhelyezi a veremre, BP-t (EBP-t) beállítja SP-re (ESP-re), és kivonja az első operandust SP-ből (ESP-ből).
Szintaxis: ENTER *azonnali címzésű_szó, azonnali címzésű_byte*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Veremhiba kizárás keletkezik, ha SP az utasítás végrehajtásának bármelyik részében a veremhatáron kívülre esik.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A 32 bites EBD és ESP regiszterek kerülnek felhasználásra. A keret címzése 16 bitről 32 bitre bővül ki.
Példa: ENTER 12,0

HLT (80286/80386)

Leállítás (HaLT)
Utasítás: HLT
Tipikus órajel: (80286) 2, (80386) 5
Az utasítás leírása: Az utasítások végrehajtásának leállítása
Az utasítás működése: A HLT utasítás a program végrehajtását állítja le. Csak külső megszakítással vagy törléssel (RESET) lehetséges az újraindítás. Az utasítás nem befolyásolja a regisztereket vagy a státuszjelzőket. Ha megszakítással indítjuk újra a programot, a tárolt CS:IP értéke arra az utasításra mutat, ami a HLT-t követi.
Szintaxis: HLT (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: A HLT egy privilégizált helyzetű utasítás, ami általános védelmi kizárást generál minden olyan esetben, amikor az éppen érvényes privilégizált szint nullától eltérő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: HLT

IDIV (80286/80386)

Előjeles egész osztás (Integer DIVision)
Utasítás: IDIV
Tipikus órajel: (80286) 17-28, (80386) 19-43
Az utasítás leírása: AX előjeles (byte-os) osztása ill. DX:AX előjeles (szavas) osztása
Az utasítás működése: Az IDIV utasítás előjeles osztási műveletet hajt végre. Ha byte méretű forrásoperandust írtunk elő, akkor az AX regisztert oszthatjuk a byte-operandussal. A hányados az AL, a maradék az AH regiszterbe kerül. Szó forrásoperandus esetén DX:AX-et egy szóval oszthatjuk. Az osztandó legnagyobb helyértékű 16 bitjét DX, a hányadost AX és a

maradékot DX tárolja. A maradéknak ugyanaz az előjele, mint az osztandónak és mindig kisebb, mint az osztandó. (L. a 80386-os megjegyzést.)
Szintaxis: IDIV *forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: SF, AF, OF, ZF, PF, CF
Kizárások védett üzemmód esetén: Ha az osztó zérus, vagy a hányados túl nagy ahhoz, hogy a kijelölt regiszterben (AX vagy AL) elférjen, INT 0 áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha az osztó zérus, vagy a hányados túl nagy ahhoz, hogy a kijelölt regiszterben (AX vagy AL) elférjen, INT 0 áll elő. Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. Duplapontos osztás az EDX:EAX regiszterpárban elhelyezett osztandóval valósítható meg. Az osztó a regiszterben/memóriában elhelyezett 32 bites érték. A 32 bites hányadost EAX-ben, a 32 bites maradékot pedig EDX-ben találhatjuk.
Példa: Szó osztása byte-tal:

```
        MOV     AX,NUMER_WRD
        IDIV    DIVSR_BTE
```

Szó osztása szóval:

```
        MOV     AX,NUMER_WORD
        CWD                             ;Szó átalakítása duplaszóvá
        IDIV    DIVSR_WRD
```

Duplaszó osztása szóval:

```
        MOV     DX,NUMER_MSW
        MOV     AX,NUMER_LSV
        IDIV    DIVSR_WRD
```

80386:

```
        MOV     EDX,MSB_NUMBER
        MOV     EAX,LSB_NUMBER
        IDIV    DIVISOR
```

IMUL (80286/80386)

Egész szorzás (Integer MULtiply)
Utasítás: IMUL
Tipikus órajel: (80286) 13-24, (80386) 9-41
Az utasítás leírása: Előjeles szorzás végrehajtása
Az utasítás működése: Az IMUL utasítás előjeles szorzási műveletet valósít meg. Ha a forrásoperandus egyetlen byte, akkor a forrásoperandust megszorozza az AL regiszter tartalmával és a 16 bites előjeles eredmény AX-ben marad. CF és OF törlődik, ha az AH regiszter AL előjeles kiterjesztése. Egyébként CF és OF 1-re áll be. (L. a 80386-os megjegyzést.) Ha az IMUL utasításban előírt forrásoperandus egy szó, akkor a forrásoperandus az AX regiszter tartalmával összeszorzódik, és a 32 bites előjeles eredmény DX:AX-ben marad. A DX regiszter tartalmazza a legnagyobb helyértékű 16 bitet. CF és OF 0 zérusra áll be, ha DX az AX előjeles kiterjesztése (L. a 80386-os megjegyzést), egyébként pedig 1-re áll be. Ha az IMUL utasítás három operandust tartalmaz, a második operandus – ami egy effektív címzésű szó – a harmadik azonnali címzésű szóoperandussal szorzódik össze. A 16 bites eredmény az első szóregiszter operandusba kerül. CF és OF értéke zérus, ha az eredmény egy olyan előjeles szó, aminek értéke a − 32768 és + 32768 tartományba esik. Egyébként CF és OF 1-re áll be.
Szintaxis: IMUL *forrás*

Befolyásolt jelzők: OF, CF
Definiálatlan jelzők: ZF, SF, AF, PF
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. Duplapontos szorzás végrehajtható, és a 64 bites eredmény az EDX:EAX regiszterpárba kerül. Ez lesz az eredménye a 32 bites regiszter/memória érték EAX-szeres szorzásának.

Példa:
```
MOV     AL,NUMBER
IMUL    NUMBER              ;Eredmény AX-ben
MOV     AX,VALUE1
IMUL    VALUE2              ;Eredmény DX:AX-ben
MOV     EAX,0FCABA1234H     ;Csak 80386-os esetén
IMUL    NUMBER              ;Eredmény EDX:EAX-ben
```

IN (80286/80386)

Byte vagy szó inputja (INput byte or word)
Utasítás: IN
Tipikus órajel: (80286) 5, (80386) 5-6
Az utasítás leírása: Byte vagy szó inputja
Az utasítás működése: Az IN utasítás az AL vagy AX regisztereket feltölti, a kijelölt port (adathozzáférési pont) tartalmával. Az átvitt adat byte vagy szó lehet. A programozó 0-tól 65535-ig bármelyik portot elérheti, ha a port számát a DX regiszterbe elhelyezi. Ha a portcím előírása az utasításon belüli adatbyte-tal történik, akkor csak a 0-tól 255-ös című portok elérése lehetséges. Az Intel a 00F8H-tól 00FFH-ig terjedő I/O portcímeket fenntartja magának, így ezeket nem szabad használnunk.
Szintaxis: IN *akkumulátor, port*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az IOPL (a jelzőbiteket tartalmazó regiszterben található) nagyobb értéket tartalmaz, mint az éppen érvényes privilégizált szint, általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

Példa:
```
IN      AL, B_P_ADR         ;Egy byte inputja AL-be
IN      AX, W_P_ADR         ;Egy szó inputja AX-be
IN      AL,DX               ;Egy byte inputja AL-be
IN      AX,DX               ;Egy szó inputja AX-be
IN      EAX,PORT8           ;Csak 80386-os esetében
```

INC (80286/80386)

Értéknövelés 1-gyel (INCrement)
Utasítás: INC
Tipikus órajel: 2-7
Az utasítás leírása: Az EA byte, szó, szóregiszter értékét 1-gyel növeli
Az utasítás működése: Az INC utasítás az operandushoz 1-et hozzáad CF változtatása nélkül.

Szintaxis: INC *céloperandus*
Befolyásolt jelzők: SF, OF, ZF, AF, PF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az operandus írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
INC     AX
INC     EBX                 ;Csak 80386-os esetében
INC     NEM_LOC
```

INS/INSB/INSW (80286/80386)

Input portból karakterláncba (INput from port to String)
Utasítás: INS,INSB,INSW
Tipikus órajel: (80286) 5, (80386) 5-8
Az utasítás leírása: Byte, szó inputját végzi a DX portról ES:[DI]- be.
Az utasítás működése: Amikor a DX (EDX) regiszterrel rámutatunk egy input portra, az INS utasítás átviszi a byte- vagy szókarakterlánc elemet az ES:DI-vel kijelölt memóriahelyre. A rendszer az INS utasítás első operandusából határozza meg, hogy byte-ok vagy szavak mozgatásáról van-e szó. A memóriaoperandusnak az ES regiszterből címezhetőnek kell lennie, mert az utasítás a szegmens törléséről nem gondoskodik. Az INS utasítás nem teszi lehetővé, hogy a port számát azonnali címzésű értéknek adjuk meg. A port csak a DX regiszter felhasználásával címezhető. Az utasítás végrehajtása után a DI (EDI) regiszter automatikusan növekszik. Ha DF értéke zérus (CLD utasítás végrehajtása után), DI (EDI) értéke nő. Ha DF értéke 1 (STD végrehajtása után), DI (EDI) értéke csökken. Az inkrementálás, ill. dekrementálás értéke 1, ha byte, 2 ha szó átviteléről van szó.
Szintaxis: INS *cél_karakterlánc, port*
INSB (operandus nélkül)
INSW (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha CPL értéke nagyobb mint IOPL-é, általános védelmi kizárás áll elő. Ha a céloperandus írásvédett szegmensben van, ugyancsak általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A memóriacímek 32 bites effektív címekből kerülnek meghatározásra. A port a DX regiszterből kapja a címét. Az effektív cím az I/O port címét nem befolyásolja.
Példa:

```
INS     BTE_STRG,DX     ;Egy byte inputja
INS     WRD_STRG,DX     ;Egy szó inputja
INSB                    ;Egy byte inputja
INSW                    ;Egy szó inputja
```

A karakterlánc hosszát az ECS regiszterből veszi az utasítás.

INT/INTO (80286/80386)

Megszakítás (INTerrupt)
Utasítás: INT
Tipikus órajel: (80286) 23-167, (80386) 33-287
Az utasítás leírása: Megszakítási eljárás hívása
Az utasítás működése: Az INT utasítás egy szoftverkapcsoló segítségével meghívja a megszakítási eljárást. Az azonnali címzésű operandus, ami a 0 - 255-ös tartományba esik, a megszakítási eljárások megszakításleíró táblázatában a meghívott eljárás indexét adja meg. Védett üzemmódban a megszakításleíró táblázat 8 byte-os leíróelemekből áll. Az aktivizált megszakítás leíróelemének meg kell határoznia egy megszakításkaput vagy egy taszk-kaput. Valós címzési módban a megszakításleíró táblázat egy olyan tömb, ami 4 byte-os mutatókat tartalmaz és a 0000H fix memóriahelyhez kötött. Az INTO utasítás megegyezik az INT utasítással, azonban két különbség van: A megszakítás számát az utasítás 4-nek veszi. Megszakítás csak akkor hozható létre, ha a túlcsordulásjelző bit (OF) értéke 1. Valós címzési mód esetén az INT utasítás a jelzőket, a CS és a visszatérési IP (EIP) regisztert a most felsorolt sorrendben ráhelyezi a veremre, majd a megszakítási szám által indexelt mutatót használja.
Szintaxis: INT *megszakítás_típus*
INTO (operandus nélkül)
Befolyásolt jelzők: Az utasítás valamenyi jelzőbitre befolyással van, ha feladatkapcsoló szerepel, egyébként a jelzőbitekre nincs hatással.
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Az általános védelmi kizárás, leíróelem nincs jelen kizárás, veremhiba kizárás vagy érvénytelen taszk állapotszegmens áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Az operandus méretének nincs hatása. Az EIP, ESP és a FLAGS regiszterek mérete a kapuból származik.
Példa:
```
INT     21H
INTO
```

IRET (80286/80386)

Visszatérés megszakításból (Interrupt RETurn)
Utasítás: IRET
Tipikus órajel: (80286) 17-169, (80386) 22-275
Az utasítás leírása: Megszakításból való visszatérés lehetővé tétele
Az utasítás működése: Valós címzéi mód esetén az IRET utasítás kiveszi az IP (EIP), CS és FLAGS regisztereket a veremből, és újraindítja a megszakított kódot. Védett módban az IRET utasítás a beágyazott taszkjelző (NT) beállításától függ. Ha NT = 1, az utasítás megfordítja a taszkváltást okozó CALL vagy INT működését. Az IRET-et végrehajtó kód aktualizált állapotát a taszk állapotszegmense tárolja, ami azt jelenti, hogy a taszk ismételt elérésével az IRET-et követő utasításokat hajtja végre a rendszer. Ha NT = 0, az IRET utasítás nélkül tér vissza a megszakító szubrutinból. A forráskód privilégiumszintjének a megszakítási rutin szintjével egyenlőnek vagy kisebbnek kell lennie.
Szintaxis: IRET (operandus nélkül)
Befolyásolt jelzők: A jelzőket tartalmazó teljes regisztert leemeli a veremről
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Az általános védelmi kizárás, leíróelem nincs jelen kizárás vagy veremhiba kizárás állhat elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.

Megjegyzés a 80386-os processzorral kapcsolatban: 32 bites kibővített utasításmutató kerül felhasználásra. Az utasítás leemeli a veremről a 48 bitesEIP és a 32 bites FLAG regisztert.
Példa: IRET

J(feltétel) (80286/80386)

Feltételes rövid ugrások (Jump short)

Utasítás:	Tipikus órajel:	Leírás:
JA (Jump if Above)	7	Ugrik, ha előjel nélkül nagyobb
JAE (Jump if Above or Equal)	7	Ugrik, ha előjel nélkül nagyobb vagy egyenlő
JB (Jump if Below)	7	Ugrik, ha előjel nélkül kisebb
JBE (Jump if Below or Equal)	7-9	Ugrik, ha előjel nélkül kisebb vagy egyenlő
JC (Jump if Carry (set))	7-9	Ugrik, ha CF = 1
JCXZ (Jump if CX is Zero)	8	Ugrik, ha CX = 0
JE (Jump if Equal)	7	Ugrik, ha egyenlő
JG (Jump if Greater)	7	Ugrik, ha előjelesen nagyobb
JGE (Jump if Greater than or Equal)	7	Ugrik, ha előjelesen nagyobb vagy egyenlő
JL (Jump if Less than)	7	Ugrik, ha előjelesen kisebb
JLE (Jump if Less than or Equal)	7	Ugrik, ha előjelesen kisebb vagy egyenlő
JNA (Jump if Not Above)	7	Ugrik, ha előjel nélkül nem nagyobb
JNAE (Jump if Not Above or Equal)	7	Ugrik, ha előjel nélkül nem nagyobb vagy egyenlő
JNB (Jump if Not Below)	7	Ugrik, ha előjel nélkül nem kisebb
JNBE (Jump if Not Below or Equal)	7	Ugrik, ha előjel nélkül nem kisebb vagy egyenlő
JNC (Jump if Not Carry)	7	Ugrik, ha CF = 0
JNE (Jump if Not Equal)	7	Ugrik, ha nem egyenlő
JNG (Jump if Not Greater)	7	Ugrik, ha előjelesen nem nagyobb
JNGE (Jump if Not Greater or Equal)	7	Ugrik, ha előjelesen nem nagyobb vagy egyenlő
JNL (Jump if Not Less)	7	Ugrik, ha előjelesen nem kisebb
JNLE (Jump if Not Less or Equal)	7	Ugrik, ha előjelesen nem kisebb vagy egyenlő
JNO (Jump if Not Overflow)	7	Ugrik, ha nincs túlcsordulás (OF = 0)
JNP (Jump if Not Parity)	7	Ugrik, ha nincs paritás (PF = 0)
JNS (Jump if Not Sign)	7	Ugrik, ha nincs előjel (SF = 0)
JNZ (Jump if Not Zero)	7	Ugrik, ha nem zérus (ZF = 0)
JO (Jump if Overflow)	7	Ugrik, ha túlcsordulás van (OF = 1)
JP (Jump if Parity)	7	Ugrik, ha van paritás (PF = 1)
JPE (Jump if Parity Even)	7	Ugrik, ha a paritás páros
JPO (Jump if Parity Odd)	7	Ugrik, ha a paritás páratlan
JS (Jump if Sign)	7	Ugrik, ha a van előjel (SF = 1) (Az eredmény negatív)
JZ (Jump if Zero)	7	Ugrik, ha zérus (ZF = 1)

Az utasítás működése: A J utasítás, amit az előbbiekben felsorolt valamelyik feltétel követ, az utasításban előírt operandusnak adja át a vezérlést. Ezek a feltételes rövid ugrások a jelzőbitek állapotát vizsgálják. A tesztben megadott operandus a következő utasítástól maxi-

málisan − 128– + 128 byte-ra lehet. Ez a korlátozás azért szükséges, hogy az assembler az éppen érvényes utasítás végétől 1 byte-os eltolást tudjon létrehozni.
Szintaxis: J*(teszt_feltétel)*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az a helyzet, amelyre az ugrás történt, túl van a kódszegmens határain, általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: 32 bites kibővített utasításmutatót használ. Az eltolás 8 bites, előjelesen kibővítve 32 bites, hacsak nem jelöljük másként.
Példa: JA INST_LABEL

JMP (80286/80386)

Ugrás (JuMP)
Utasítás: JMP
Tipikus órajel: (80286) 7-183, (80386) 7-268
Az utasítás leírása: Feltétel nélküli ugrás végrehajtása
Az utasítás működése: A JMP utasítás hatására a program vezérlését egy másik utasítás kapja meg anélkül, hogy a visszatérésről bármilyen információt tárolna. A JMP utasítás szegmensen belüli közvetlen és közvetett, valamint szegmensek közötti ugrást valósít meg. A szegmensen belüli közvetlen ugrások az utasítást követő eltolási byte-ot használják. A szegmensen belüli közvetett ugrások annak a helynek a tartalmát használják, amelyet az utasítás byte-ját követő byte-ok címeznek meg. Ha az utasítás egy szegmensen belüli közvetlen ugrás, az utasítás vége és a célként megjelölt címke közötti távolság IP (EIP)-hez hozzáadódik. A szegmensek közötti ugrások kicserélik a CS regiszter tartalmát, ami közvetlen ugrás esetében az utasítást követő második szó, míg közvetett ugrás esetében az adatcímet követő második szó felhasználásával valósul meg.
Szintaxis: JMP*(cél)*
Befolyásolt jelzők: Az utasítás valamennyi jelzőbitre befolyással van, ha taszkváltás történik, egyébként a jelzőbitekre nincs hatással.
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az ugrás közeli (NEAR), általános védelmi kizárás állhat elő, ha az eltolással kijelölt hely az éppen érvényes kódszegmens határain kívül esik. Ha az ugrás távoli (FAR), általános védelmi kizárás áll elő. További lehetséges kizárások a leíróelem nincs jelen kizárás, a veremhiba kizárás és az érvénytelen taszk állapotszegmens kizárás. Ha a szegmensek közötti közvetett ugrás operandusa egy regiszter, definiálatlan műveletkód kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szegmensek közötti közvetett ugrás operandusa egy regiszter, definiálatlan műveletkód kizárás áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A kibővített utasításmutató 32 bites. A mutatóoperandusok 48 bitesek, az eltolási értékek pedig 32 bitesek.
Példa: Szegmensek közötti közvetlen ugrás:

```
                JMP         FAR_LABEL
Szegmensek közötti közvetlen ugrás:
                JMP         TABLE[BP][DI]
                JMP         TABLE[EBP][EDI]
(Csak 80386-os esetében)
Szegmensen belüli közvetlen ugrás:
                JMP         NEAR_LABEL
Szegmensen belüli közvetlen ugrás:
                JMP         WORDPTR[BX][DI]
```

LAHF (80286/80386)

A jelzők betöltése AH-ba (Load AH from Flag)
Utasítás: LAHF
Tipikus órajel: 2
Az utasítás leírása: A kiválasztott AH bitpozícióba betölti SF-t, ZF-t, AF-t, PF-t és CF-t.
Az utasítás működése: Az LAHF utasítás az SF, ZF, AF, PF és CF jelzőbiteket betölti az AH regiszterbe a következők szerint:

7.	6.	5.	4.	3.	2.	1.	0.
SF	ZF		AF		PF		CF
7.bit							0.bit

Az 1-es, 3-as, 5-ös bitpozíciók definiálatlanok maradnak.
Szintaxis: LAHF (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: LAHF

LAR (80286/80386)

Hozzáférési jogot tartalmazó byte betöltése, ha elérhető (Load Access Rights byte)
Utasítás: LAR
Tipikus órajel: (80286) 14-16, (80386) 15-16
Az utasítás leírása: A hozzáférési jog byte betöltése
Az utasítás működése: A LAR utasítás második operandusa (memória- vagy regiszterszó) egy szelektort tartalmaz. Ha a hozzárendelt leíróelem az éppen érvényes privilégiumszinten látható, akkor a leíróelem hozzáférési jog byte-ját az utasítás betölti az első (regiszter) operandus magas byte-jába és az alacsony byte-ot zérusra állítja be. Ha a betöltés megtörtént, ZF beáll 1-re, egyébként értéke zérus.
Szintaxis: LAR*(hozzáférési_jog_byte,szelektor)*
Befolyásolt jelzők: ZF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: INT 6 áll elő. A rendszer a LAR utasítást valós címzési módban nem ismeri fel.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: LAR ARB,SELECTR

LDS/LES (80286/80386)

LSS/LFS/LGS (80386)

Duplaszavas mutató betöltése. (Load using DS/ES)
Utasítás: LDS/LES

Tipikus órajel: 7
Az utasítás leírása: Betölti EA-t, ami duplaszó, a DS/ES és a szóregiszterbe
Az utasítás működése: Az LDS/LES betölti az utasítás második operandusa által megjelölt memóriahelyen található 4 byte-os mutatót egy szegmens- és egy szóregiszterbe. A mutató első szavát (eltolás) az utasítás első operandusa által megjelölt regiszterbe tölti, míg a második szó a DS (LDS esetén) vagy az ES (LES esetén) regiszterbe kerül.
Szintaxis: LDS*cél,forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Lehetséges kizárások az általános védelmi kizárás és a leíróelem nincs jelen kizárás. Ha az operandus a szegmens határán kívül van, általános védelmi kizárás vagy veremhiba kizárás áll elő. Ha a forrásoperandus egy regiszter, definiálatlan műveletkód kizárás áll elő.
Kizárások valós címzési mód esetén: Ha az operandus a 0FFFFH vagy a 0FFFDH eltolási értéknél van, INT 13 áll elő. Ha a forrásoperandus egy regiszter, definiálatlan műveletkód kizárás áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A mutatóoperandusok 48 bitesek. A művelet kiterjeszthető LSS/LFS/LGS-re.

```
Példa:  LDS     SI,DOUBLEWRD1
        LES     BX,DOUBLEWRD2
```

LEA (80286/80386)

Az effektív cím betöltése (Load Effectiv Address offset)
Utasítás: LEA
Tipikus órajel: (80286) 3, (80386) 2
Az utasítás leírása: A forrásoperandus eltolási értékét átadja a céloperandusnak
Az utasítás működése: A LEA utasítás első operandusa a célregiszter operandus, ami a második operandus eltolási címét kapja meg.
Szintaxis: LEA *cél, forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a második operandus egy regiszter, definiálatlan műveletkód kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a második operandus egy regiszter, definiálatlan műveletkód kizárás áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A memóriacímek a 32 bites effektív címekből alakulnak ki.

```
Példa:  LEA     AX,[BP][DI]
        LEA     EBX,MYDATA          (Csak 80386-os esetében)
```

LEAVE (80286/80386)

Kilépés magas szintű eljárásból (LEAVE)
Utasítás: LEAVE
Tipikus órajel: (80286) 5, (80386) 4
Az utasítás leírása: Egy magas szintű nyelv eljárásából való kilépést teszi lehetővé.
Az utasítás működése: A LEAVE az ENTER utasítás ellentétes művelete. A LEAVE valamennyi lokális változót felszabadítja, BP-t beállítja SP-re. A regiszterek közvetlenül az eljárás hívása utáni értrékeiket veszik fel. Amikor BP (EBP) bemásolja SP (ESP)-be, az eljárás által lefoglalt veremterület felszabadul. A veremből a régi keretmutató BP (EBP) kapja meg,

visszaállítva a hívó keretét, és az ezt követő RET nn utasítás eltávolít minden argumentumot, ami az eljárás részére a verembe került.
Szintaxis: LEAVE (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha BP (EBP) nem mutat az éppen érvényes veremszegmensre, veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. Az ESP és EBP regiszterek kerülnek felhasználásra.
Példa: LEAVE

LGDT/LIDT (80286/80386)

A globális/megszakítás leíróelemtáblázat-regiszter betöltése (Load Global/Interrupt Descriptor Table register)
Utasítás: LGDT/LIDT
Tipikus órajel: 11-12
Az utasítás leírása: A memória bizonyos részének betöltése a GDT vagy az IDT regiszterbe
Az utasítás működése: Az LGDT vagy LIDT utasítás betölti a globális vagy megszakítás leíróelemtáblázat-regiszterbe a memóriának azt a 6 byte-ját, amire az operandus effektív címe rámutat. A leíróelem-táblázat-regiszter LIMIT (határ) mezője az első szóból kapja az értékét. A következő három byte a regiszter BASE (bázis) mezőjébe kerül. Az utolsó byte figyelmen kívül marad. Ezek az operációs regiszter szoftver utasításai közé tartoznak és alkalmazói programokban nem találhatók meg.
Szintaxis: LGDT *memória_operandus*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az éppen érvényes privilégiumszint nem zérus, általános védelmi kizárás áll elő. Ha a forrásoperandus egy regiszter, definiálatlan műveletkód kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:
```
LGDT    NEM_WRD
LIDT    NEM_WRD
```

LLDT (80286/80386)

A lokális leíróelemtáblázat-regiszter betöltése (Load Local Descriptor Table register)
Utasítás: LLDT
Tipikus órajel: (80286) 17-19, (80386) 20
Az utasítás leírása: A szelektort a lokális leíróelemtáblázat- regiszterbe tölti be
Az utasítás működése: Az LLDT szóoperandusának (memória vagy regiszter) a globális leíróelem táblázathoz mutató szelektort kell tartalmaznia. Ennek a belépési pontnak lokális leíróelem táblázatnak kell lennie. Ekkor a lokális leíróelem táblázat regiszter betöltődik a belépési pontból. Ez az utasítás az operációs rendszer szoftver utasításai közé tartozik és alkalmazói szoftverekben nem található meg.

Szintaxis: LLDT *szó_operandus*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az éppen érvényes privilégium- szint nem zérus, általános védelmi kizárás áll elő. Ha a szelektor operandusa nem mutat a globális leíróelem táblázatra, vagy ha a GDT belépési pontja nem egy lokális leíróelem táblázat, általános védelmi kizárás áll elő. LDT leíróelem hiányában, leíróelem nincs jelen kivétel áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: INT 6 áll elő. A valós címzési módban az LLDT utasítást nem ismeri fel a processzor.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: LLDT BP

LMSW (80286/80386)

A gépi státusz-szó betöltése (Load Machine Status Word)
Utasítás: LMSW
Tipikus órajel: (80286) 3-6, (80386) 10-13
Az utasítás leírása: Az EA szó betöltése a gépi státusz-szóba
Az utasítás működése: Az LMSW az operációs rendszer egyik szoftver utasítása, ami alkalmazói programokban nem jelenik meg. Az LMSW a forrásoperandusból betölti a gépi státuszszót, amely által védett módba kapcsolhatunk át. Ha védett módot használunk, a következő utasításnak szegmensen belüli ugrásnak kell lennie.
Szintaxis: LMSW *forrás_operandus*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az éppen érvényes privilégium-szint nem zérus, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH eltolásnál van, INT 13 áll elő.
Példa: LMSW SP

LOCK (80286/80386)

Buszlezáró jel aktivizálása. (Activate the BUS LOCK signal)
Utasítás: LOCK
Tipikus órajel: 0
Az utasítás leírása: A buszlezáró jelet aktivizálja
Az utasítás működése: A LOCK előkészítő utasítás lezárja azt a memóriahelyet, amelyet az utasítás céloperandusa előír. Ez a művelet, amely a 80386-os buszlezáró jelének aktivizálásával valósul meg, hatásban marad az utasítás végrehajtása alatt. A LOCK prefix utasítás a következő 80386-os utasításokat előzheti meg:

BT, BTS, BTR, BTC	memória, regiszter/azonnali
XCG	regiszter, memória
XCG	memória, regiszter
ADC, SUB, ADC, SBB, OR, XOR, AND	memória, regiszter/azonnali
NOT, NEG, INC, DEC	memória

Szintaxis: LOCK *utasítás, operandus_típus*

Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az Input/Output privilégium szint alacsonyabb, mint az éppen érvényes privilégiumszint, általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A 8086-os és a 80286-os a memóriazáró funkciók bővebb készletével rendelkezik mint a 80386-os. Előfordulhat, hogy az előbb említett processzorokra írt alkalmazások a 80386-on nem működnek megfelelően.
Példa: LOCKXCGNEM_WORD,AX

LODS/LODSB/LODSW (80286/80386)

Byte- vagy szókarakterlánc betöltése (LOaD String Byte/Word)
Utasítás: LODS/LODSB/LODSW
Tipikus órajel: 5
Az utasítás leírása: Az [SI], DS:[SI] által megcímzett byte betöltése AL-be, ill. az [ESI], DS:[ESI] által megcímzett byte betöltése AL-be
Az utasítás működése: Az LODS utasítás azt a byte- vagy szóoperandust, amire az SI (ESI) regiszter rámutat, áttölti a forrásoperandusból az AL vagy AX regiszterbe. SI (ESI) értéke automatikusan nő. Ha az irányjelző (DF) zérus (egy CLD utasítás végrehajtása után), SI (ESI) értéke nő. Ha DF 1 (STD-t hajtottunk végre), SI (ESI) értéke csökken. A léptetés byte-operandusok esetén 1, szóoperandusok esetén 2.
Szintaxis: LODS *forrás_karakterlánc*
LODSB (nincs operandus)
LODSW (nincs operandus)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites.
Példa:

```
LEA     SI,DATA          ;Ha az adat 16 bites, az eredmény AX-ben áll elő.
LODS
```

LOOP (80286/80386)

Ciklusszervezés a CX számláló felhasználásával (LOOP control with CX counter)
Utasítás **Magyarázat**

LOOP	CX értékének csökkentése 1-gyel és ugrás, ha CX $\neq$ 0
LOOPE	Ugrás, ha CX <> 0 és ZF = 1
LOOPZ	Ugrás, ha CX <> 0 és ZF = 1
LOOPNE	Ugrás, ha CX <> 0 és ZF = 0
LOOPNZ	Ugrás, ha CX <> 0 és ZF = 0

Tipikus órajel: (80286) 8, (80386) 11
Az utasítás működése: A LOOP utasítás a CX (ECX) regiszter értékét 1-gyel csökkenti. A jelzőket nem befolyásolja, csak megvizsgálja a LOOPn utasítással előírt feltételt. Ha a vizsgálat eredménye a feltételnek megfelelő, akkor szegmensen belüli vezérlésátadás történik.

A céloperandus által előírt memóriahelynek az utasítástól számított +128 −127-es tartományba kell esnie.

Szintaxis: LOOP *rövid_cél*
LOOPE *rövid_cél*
LOOPZ *rövid_cél*
LOOPNZ *rövid_cél*
LOOPNE *rövid_cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Az ugrás az éppen érvényes kódszegmens határain kivül esik, általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:

```
        MOV     CL,04H
AGAIN:  ADD     BX,01H
        ADD     AL,INFO[BX]
        DAA
        LOOP    AGAIN
        MOV     CX,12H
NEXT:   INC     BX
        CMP     TABLE[BX],0
        LOOPE   NEXT
        MOV     ECX,0ABCDEH   (Csak 80386-os esetében)
REPEAT: INC     BX
        MOV     AL,TABLE_A[BX]
        ADD     AL,TABLE_B[BX]
        MOV     TOTAL[BX],AL
        LOOPNZ  REPEAT
```

LSL (80286/80386)

A szegmenshatár betöltése (Load Segment Limit)
Utasítás: LSL
Tipikus órajel: (80286) 14-16, (80386) 20-26
Az utasítás leírása: A szegmenshatár betöltése egy álregiszterbe
Az utasítás működése: Ha a szelektor a második (memória vagy regiszter) operandusban CPL-nél látható, az a szó, ami a leíróelem határmezőjéből áll, a bal oldali operandusba töltődik be, aminek regiszternek kell lennie. Az érték a szegmensre vonatkozó határmező. A zérus flag (ZF) értéke 1, ha a betöltés megtörtént, egyébként ZF törlődik.
Szintaxis: LSL *szegmens_határ, szelektor*
Befolyásolt jelzők: ZF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: INT 6 áll elő. A valós címzési módban a LSL utasítást nem ismeri fel a rendszer.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:

```
LSL     AX,SELECTR
```

LTR (80286/80386)

Taszkregiszter betöltés (Load Task Register)
Utasítás: LTR
Tipikus órajel: (80286) 17-19, (80386) 23-27
Az utasítás leírása: Az EA szót betölti a taszkregiszterbe
Az utasítás működése: Az LTR utasítás hatására a taszkregiszterbe a forrásregiszter vagy a memóriahely forrásoperandus kerül. A betöltött TSS foglalt állapotot jelez. Ez az operációs rendszer egyik szoftver utasítása, ami alkalmazói programokban nem jelenik meg.
Szintaxis: LTR *forrás_operandus*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: INT 6 áll elő. A valós címzési módban a LTR utasítást nem ismeri fel a rendszer.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: LTR MEM_WRD

MOV (80286/80386)

Adatmozgatás (MOVe)
Utasítás: MOV
Tipikus órajel: (80286) 2-19, (80386) 2-22
Az utasítás leírása: A forrásoperandus tartalmát átmásolja a céloperandusba
Az utasítás működése: Számos MOV utasítás létezik, azonban mindegyiknek azonos a funkciója: a forrásoperandus tartalmát átmásolja a céloperandusba anélkül, hogy a forrást felülírná.
Szintaxis: MOV *cél, forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Szegmensregiszter betöltése esetén általános védelmi kizárás, veremhiba kizárás vagy leíróelem nincs jelen kizárás áll elő. Ha a céloperandus írásvédett szegmensben van, akkor általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A memóriacímek 32 bites effektív címek vagy 32 bites eltolások felhasználásával alakulnak ki.
Példa:

```
Memóriából akkumulátorba
MOV     AX,MEM_WRD
MOV     ECX,MYWORD      (Csak 80386-os esetében)
Akkumulátorból memóriába
MOV     MEM_BYTE,AL
Szegmensregiszterből memóriába/regiszterbe
MOV     BX,ES
MOV     TABLE[BX],SS
Memóriából/regiszterből szegmensregiszterbe
MOV     ES,NEXT_WRD[SI]
```

```
        MOV     DS,AX
Regiszterből regiszterbe
        MOV     CX,DI
        MOV     EAX,ECX             (Csak 80386-os esetében)
Memóriából/regiszterből regiszterbe
        MOV     AX,MEM_VAL
        MOV     CX,[BP][SI]
Azonnali címzésű adatból regiszterbe
        MOV     DI,513
        MOV     EAX,12345678H       (Csak 80386-os esetében)
Azonnali címzésű adatból memóriába/regiszterbe
        MOV     TABLE[BP][SI],25
        MOV     BX,77
```

MOVS/MOVSB/MOVSW (80286/80386)

Byte- vagy szókarakterlánc másolása (MOV String Byte/Word)
Utasítás: MOVS/MOVSB/MOVSW
Tipikus órajel: (80286) 5, (80386) 7
Az utasítás leírása: A DS:[SI] által megcímzett forrás- karakterlánc átmásolása az ES:[DI] által megcímzett helyre, ill. a DS:[ESI] által megcímzett forráskarakterlánc átmásolása az ES:[EDI] által megcímzett helyre.
Az utasítás működése: A MOV utasítás az [SI] ([ESI])-nél található byte-tot vagy szót átmásolja az ES:[DI]-nél (ES:[EDI]-nél) található byte- vagy szó céloperandusba. A céloperandusnak címezhetőnek kell lennie az ES regiszterből. A forrásoperandus szegmensregisztere átdefiniálható, míg a céloperandusé nem. Az utasítás végrehajtása után SI (ESI) és DI (EDI) automatikus léptetése a következő szabályok szerint történik: Ha DF zérus (CLD-t hajtottunk végre), a regiszterek értéke nő. Ha DF 1 (CLD-t hajtottunk végre), a regiszterek értéke csökken. A regiszterek léptetése byte operandusok esetén 1, szóoperandusok esetében pedig 2.
Szintaxis: MOVS *cél_karakterlánc, forrás_karakterlánc*
MOVSB (operandus nélkül)
MOVSW (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a céloperandus írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A memóriacímek 32 bites effektív címek felhasználásával alakulnak ki. A karakterlánc hosszát (számláló) az ECX regiszterben kell beállítani.
Példa:

```
        LEA     SI,MYSOURCE
        LEA     DI,ES:HERDESTINATION
        MOV     CX,50
REP     MOVS    HERDESTINATION,MYSOURCE
```

MUL (80286/80386)

Előjel nélküli szorzás (MULtiply)
Utasítás: MUL
Tipikus órajel: (80286) 13-21, (80386) 9-41
Az utasítás leírása: Előjel nélküli szorzást valósít meg (AX = AL * előírt byte) Előjel nélküli szorzást valósít meg (DX:AX = AX * előírt szó)
Az utasítás működése: A MUL utasítás a operandust beszorozza az AL regiszterrel és az eredményt az AX regiszterben helyezi el. Ha AH értéke zérus, CF és OF zérusra, egyébként pedig 1-re áll be. Szóoperandus előírása esetén a MUL utasítás az AX regiszter és egy szó összeszorzását valósítja meg, és az eredményt DX:AX-ben helyezi el. DX tartalmazza az eredmény nagyobb helyértékű 16 bitjét. CF és OF zérust vesz fel, ha DX zérus, egyébként pedig 1-et. (L. a 80386-os megjegyzést.)
Szintaxis: MUL *forrás*
Befolyásolt jelzők: OF, CF
Definiálatlan jelzők: SF, ZF, AF, PF
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. Duplapontos szorzás az EAX és egy 32 bites regiszter/memóriahely összeszorzásával valósítható meg. A 64 bites eredményt az EDX:EAX regiszterpár tárolja.
Példa:

```
Byte szorzása szóval
        MOV     AL,MULTP_BTE
        CBW                             ;AL kiterjesztése szóvá
        MUL     VAL_BTE
Byte szorzása byte-tal
        MOV     AL,MULTP_BTE
        MUL     VAL_BTE
Szó szorzása szóval
        MOV     AX,MULTP_WRD
        MUL     VAL_WRD                 ;Magas bitek DX-ben, alacsony bitek AX-ben
80386:
        MOV     EAX,0FCAB1234H          (Csak 80386-os esetében)
        MUL     EBX
```

NEG (80286/80386)

Kettes komplemenses negáció (NEGation)
Utasítás: NEG
Tipikus órajel: (80286) 2-7, (80386) 2-6
Az utasítás leírása: Az előírt byte-tal vagy szóval kettes komplemenses negáció elvégzése
Az utasítás működése: Az operandust 0-ból kivonja, hozzáad 1-et, és az eredményt – kettes komplemenses alakot – visszateszi az operandusba. Az átviteljelző bitet (CF) 1-re állítja be, kivéve, ha az operandus zérus. Ebben az esetben CF zérus lesz.
Szintaxis: NEG *cél*
Befolyásolt jelzők: OF, SF, ZF, AF, CF, PF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános

védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: NEG AL
NEG EBX (Csak 80386-os esetében)

NOP (80286/80386)

Művelet nélküli utasítás (No OPeration)
Utasítás: NOP
Tipikus órajel: 3
Az utasítás leírása: Tevékenység nem történik
Az utasítás működése: Az NOP 1 byte-os utasítás, csak az IP (EIP) regiszterre van hatással.
Szintaxis: NOP (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: NOP

NOT (80286/80386)

Logikai nem művelet (NOT)
Utasítás: NOT
Tipikus órajel: (80286) 2-7, (80386) 2-6
Az utasítás leírása: EA byte/szó minden egyes bitjét megfordítja
Az utasítás működése: A NOT utasítás az operandus minden bitjét az ellenkezőjére billenti, és az eredményt az operandusba teszi vissza. A jelzőket nem befolyásolja.
Szintaxis: NOT *cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: NOT AX
NOT EBX (Csak 80386-os esetében)

OR (80286/80386)

Logikai megengedő VAGY művelet (inclusive OR)
Utasítás: OR
Tipikus órajel: (80286) 2-7, (80386) 2-7
Az utasítás leírása: A logikai megengedő VAGY műveletet végrehajtása

Az utasítás működése: A megengedő VAGY logikai művelet két operandus összetartozó bitpárjait vizsgálja; ha azok mindegyike zérus, akkor a művelet eredménye zérus, minden egyéb esetben pedig 1. Az OR az eredményt a céloperandusban tárolja.
Szintaxis: OR *cél, forrás*
Befolyásolt jelzők: CF = 0, OF = 0, SF, ZF, PF
Definiálatlan jelzők: AF
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
OR      CX,DI
OR      EAX,0FFFF0000H      (Csak 80386-os esetében)
OR      EBX,ECX             (Csak 80386-os esetében)
```

OUT (80286/80386)

Byte vagy szó kiírása (OUTput)
Utasítás: OUT
Tipikus órajel: (80286) 3, (80386) 3-4
Az utasítás leírása: Szó vagy byte kiírása egy azonnali címzésű portra [DX]
Az utasítás működése: Az akkumulátor (AX vagy AL) tartalmát kiküldi a portra. 0-tól 65535-ig bármelyik port használható.
Szintaxis: OUT *port, akkumulátor*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az IOPL (a jelzőbit regiszterben található) az éppen érvényes kiváltságos szintnél magasabbat tartalmaz, általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
OUT     WRD_PORT,AX
OUT     BYT_PORT,AL
OUT     EDX,EAX             (Csak 80386-os esetében)
```

OUTS/OUTSB/OUTSW (80286/80386)

Karakterlánc kiküldése portra (OUTput String to port)
Utasítás: OUTS/OUTSB/OUTSW
Tipikus órajel: (80286) 5, (80386) 7
Az utasítás leírása: Az [SI], DS:[SI] által megcímzett byte-ot (szót) kiküldi a DX portra
Az utasítás működése: Az OUT utasítás, a memóriában elhelyezkedő byte- vagy szókarakterláncot, amire DS:SI-vel rámutatunk, kiküldi a DX-szel megcímzett portra. Az OUT második operandusa az áthelyezendő adat típusát (szó vagy byte) határozza meg. SI értéke automatikusan növekszik, ha az irányjelző bit (DF) zérus. Ha DF értéke 1, SI csökken. A regiszter léptetése byte-operandus esetén 1, szóoperandus esetén 2.
Szintaxis: OUTS *port, forrás_karakterlánc*
OUTSB (operandus nélkül)
OUTSW (operandus nélkül)
Befolyásolt jelzők: Nincs

Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha CPL értéke nagyobb vagy egyenlő IOPL értékével, általános védelmi kizárás áll elő. Ez történik akkor is, ha a CS DS vagy ES szegmens egy illegális memóriaoperandus effektív címét tartalmazza. Ha az SS szegmens egy illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A memóriacímek a 32 bites effektív címekből állnak elő. A 16 bites rész a DX regiszterből származik. A 32 bites specifikáció az I/O portcímet nem befolyásolja. A karakterlánc hosszúságát (számláló) az ECX regiszterben kell beállítani.

Példa:
```
OUTS    DX,BYT_STRNG
OUTS    DX,WRD_STRNG
OUTSB
OUTSW
```

POP (80286/80386)

Szó leemelése a veremről (POP)
Utasítás: POP
Tipikus órajel: (80286) 5, (80386) 5-7
Az utasítás leírása: A verem tetején levő adatot a következő helyek valamelyikébe helyezi át: DS, ES, SS, memóriaszó, regiszter
Az utasítás működése: A POP művelet azt a szót, amire a veremmutató (SP) rámutat, áthelyezi a céloperandusba és SP értékét 2-vel növeli.
Szintaxis: POP *cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha szegmensregisztert töltünk be, általános védelmi kizárás, veremhiba kizárás vagy leíróelem nincs jelen kizárás áll elő. Veremhiba kizárás áll elő, ha a verem éppen érvényes legfelső eleme nincs a veremszegmensen belül. Ha a céloperandus címe írásvédett szegmensben van, ugyancsak általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A memóriacímek 32 bites effektív címekből képződnek.

Példa: Regiszteroperandus:
```
POP     CX
POP     EAX             (Csak 80386-os esetében)
```
Szegmensregiszter:
```
POP     SS
```

POPA (80286/80386)

Valamennyi általános célú regiszter leemelése a veremről (POP ALL)
Utasítás: POPA
Tipikus órajel: (80286) 19, (80386) 24
Az utasítás leírása: A DI, SI, BP, SP, BX, DX, CX, AX értékeket leemeli a veremről (80286-os

esetében). A EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX értékeket leemeli a veremről (80386-os esetében)
Az utasítás működése: A POPA utasítás mind a nyolc általános célú regisztert – kivéve SP (ESP)-t, amit a rendszer eldob – kiveszi a veremből, és behelyezi a megfelelő regiszterbe. A POPA utasítás, ami a PUSHA fordítottja, visszaállítja az általános célú regiszterekbe a PUSHA végrehajtása előtti értékeket. A DI (EDI) az első regiszter, amit leemel a veremről.
Szintaxis: POPA (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a verem kezdő- vagy zárócíme nincs a veremszegmensen belül, veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: POPA

POPF (80286/80386)

Jelzőbitek leemelése a veremről (POP Flags)
Utasítás: POPF
Tipikus órajel: 5
Az utasítás leírása: A verem tetején tárolt szóval felülírja a jelzőbit regisztert.
Az utasítás működése: A POPF utasítás SS:SP-vel mutat rá a verem tetejére és az itt található információt bemásolja a jelzőbiteket tároló regiszterbe. SP (ESP) értéke automatikusan 2-vel nő. A jelzők a 15. bittől kezdve a következők:

Definiálatlan
Egymásba ágyazott taszkbit (Nested Task = NT)
I/O privilégiumszint (I/O Privileged Level = IOPL), kétbites
Túlcsordulás (Overflow = OF)
Irány (Direction = DF)
Megszakítás-engedélyezés (Interrupt Enabled = IF)
Nyomkövetés (Trap = TF)
Előjel (Sign = SF)
Zérus (Zero = ZF)
Definiálatlan
Segédátvitel (Auxiliary Carry = AF)
Definiálatlan
Paritás (Parity = PF)
Definiálatlan
Átviteli (Carry = CF)

Szintaxis: POPF (operandus nélkül)
Befolyásolt jelzők: A jelzőbiteket tartalmazó teljes regisztert leemeli a veremről.
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a verem teteje nincs a veremszegmensen belül, veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: POPF

PUSH (80286/80386)

Szó ráhelyezése a veremre (PUSH word onto stack)
Utasítás: PUSH
Tipikus órajel: (80286) 3-5, (80386) 2-5
Az utasítás leírása: ES, CS, SS, DS, regiszter, memória vagy azonnali címzésű operandus ráhelyezése a veremre
Az utasítás működése: A PUSH utasítás SP (ESP) értékét kettővel csökkenti, és az operandust elhelyezi a verem tetején. A 80286/80386-os PUSH utasítás az SP (ESP) utasítás előtti értékét helyezi rá a veremre. Megjegyezzük, hogy ez eltérő a 8086-os PUSH-tól, ahol SP új (2-vel csökkentett) értéke kerül rá a veremre.
Szintaxis: PUSH *forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha SP új értéke a veremszegmens határán kívülre esik, veremhiba kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites. A memóriacímek 32 bites effektív címekből állnak elő.
Példa:

```
PUSH    AX
PUSH    EAX                 (Csak 80386-os esetében)
PUSH    CS
```

PUSHA (80286/80386)

Valamenyi általános célú regiszter ráhelyezése a veremre (PUSH All general registers)
Utasítás: PUSHA
Tipikus órajel: (80286) 17, (80386) 18
Az utasítás leírása: Az AX, CX, DX, BX, valamint az eredeti SP, BP, SI, DI tartalmát helyezi rá a veremre (80286-os esetében). Az EAX, ECX, EDX, EBX, valamint az eredeti ESP, EBP, ESI, EDI tartalmat helyezi rá a veremre (80386-os esetében)
Az utasítás működése: A PUSHA utasítás a nyolc általános célú regiszter tartalmát ráhelyezi a veremre. Az veremmutató (SP) értékét 16-tal csökkenti, így nyolc szó tárolása lehetséges. A regiszterek tartalma a következő sorrend szerint kerül a verembe: AX, CX, DX, BX, az eredeti SP, BP, SI, DI (80286) EAX, ECX, EDX, EBX, az eredeti ESP, EBP, ESI, EDI (80386), amelynek következtében a 16 új verembyte fordított sorrendben szerepel a veremben.
Szintaxis: PUSHA (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a kezdő- vagy zárócím a veremszegmens határain kívülre esik, veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: PUSHA

PUSHF (80286/80386)

A jelzőbiteket tartalmazó regiszter ráhelyezése a veremre (PUSH Flags register)
Utasítás: PUSHF
Tipikus órajel: (80286) 3, (80386) 4
Az utasítás leírása: A jelzőbiteket tartalmazó regisztert ráhelyezi a veremre
Az utasítás működése: A PUSHF utasítás az SP (ESP) regisztert 2-vel csökkenti, és a jelzőbiteket tartalmazó teljes regisztert bemásolja az SP (ESP)-vel megcímzett szóoperandus bizonyos bitjeire. A jelzők a 15. bittől kezdve a következők:

Definiálatlan
Egymásba ágyazott taszkbit (Nested Task = NT)
I/O privilégiumszint (I/O Privileged Level = IOPL), kétbites
Túlcsordulás (Overflow = OF)
Irány (Direction = DF)
Megszakítás-engedélyezés (Interrupt Enabled = IF)
Nyomkövetés (Trap = TF)
Előjel (Sign = SF)
Zérus (Zero = ZF)
Definiálatlan
Segédátvitel (Auxiliary Carry = AF)
Definiálatlan
Paritás (Parity = PF)
Definiálatlan
Átviteli (Carry = CF)

Szintaxis: PUSHF (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha SP új értéke a veremszegmens határán kívülre esik, veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A 32 bites kibővített, jelzőbiteket tartalmazó regisztert helyezi rá a veremre.
Példa: PUSHF

RCL/RCR/ROL/ROR (80286/80386)

Bitforgató utasítások (Rotate)
Utasítás: RCL/RCR/ROL/ROR
Tipikus órajel: (80286) 2-5, (80386) 3-10
Az utasítás leírása: Bitforgatási utasítások
Az utasítás működése: A bitforgató utasítások mindegyike az előírt regiszter vagy memória-operandus bitjeit forgatja. A ROL (ROtate Left = bitforgatás balra) valamennyi bitet 1 pozícióval balra forgatja. A legnagyobb helyértékü bit a legkisebb helyértékü bit helyére kerül. Az ROR (ROtate Right = bitforgatás jobbra) utasías az előzővel ellentétes. Valamennyi bitet 1 pozícióval jobbra forgatja és a legkisebb helyértékü bitet a legnagyobb helyértékű bitbe helyezi. (A kicsorgó bitek mindkét utasításnál bekerülnek CF-be is.) Az RCL utasítás (Rotate through Carry Left) CF-et a legkisebb helyértékü bitbe helyezi és a legnagyobb helyértékü bitet pedig CF-be. Az RCR (Rotate through Carry Right) utasítás az ellenkezőjét végzi. CF-et helyezi a legnagyobb helyértékü bitpozícióba és a legkisebb helyértékű bitet pedig CF-be. A bitléptető utasítások második operandusával a forgatások számát adhatjuk meg. A túlcsor-

dulás jelző bit (OF) csak akkor állítódik be 1-re, ha a bitforgató utasítások második operandusa 1. RCR esetén a túlcsordulás vizsgálata a bitforgatás előtt történik, RCL, ROL és ROR esetén pedig a bitforgatás után. Az OF beállítása 1-re a következő feltételektől függ: Ha CF az operandus magas bitjével egyenlő, OF értéke zérusra áll be. Ha CF az operandus magas bitjével nem egyenlő, OF értéke 1 lesz.

Megjegyzés: A 80286/80386-os bitforgató utasítások maximálisan 31 bitforgatást tesznek lehetővé. Ha 31-nél több a forgatások száma, akkor a számláló (CL) legalacsonyabb helyértékü öt bitjét veszi figyelembe.

Szintaxis: RCL *cél*, 1
RCL *cél*, CL ;CL a bitléptetések számát határozza meg
RCL *cél, számláló*
RCR *cél*, 1
RCR *cél*, CL
RCR *cél, számláló*
ROL *cél*, 1
ROL *cél*, CL
ROL *cél, számláló*
ROR *cél*, 1
ROR *cél*, CL
ROR *cél, számláló*

Befolyásolt jelzők: OF (csak egyetlen bitforgatásnál), CF

Definiálatlan jelzők: OF egynél több bitforgatás esetén

Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás áll elő. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás áll elő. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás áll elő.

Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.

Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

Példa:
```
RCL     AH, 1
RCL     MEMRY_BTE, REP_VAL
RCL     DH, CL
RCR     BL, 1
RCR     MEMRY_BTE, REP_VAL
RCR     TABLE[BX][DI], CL
ROL     CX, 1
ROL     MEMRY_BTE,
        REP_VAL
ROL     AX, CL
ROR     BL, 1
ROR     MEMRY_BTE,
        REP_VAL
ROR     TABLE[BX][DI], CL
```

REP/REPZ/REPE/REPNE/REPNZ (80286/80386)

Karakterlánc műveletek ismételtetése (REPeat)

Utasítás: REP/REPZ/REPE/REPNE/REPNZ

Tipikus órajel: 5 +9 * N (N = iterációk száma)

Az utasítás leírása: Karakterlánc műveletek ismételtetése

Az utasítás működése: A REP, REPE és REPNE prefix műveleteket követő egyszerű karakterlánc műveletek mindaddig ismétlődnek, amíg a CX (ECX) regiszter értéke nem lesz zérus.

A CMPS és SCAS műveleteknél az ismétlés befejeződik, ha a REP-et követő egyszerű művelet ismétlése után ZF különbözik az ismétlő prefix Z bitjétől.
Szintaxis: REP
REPE
REPNE
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

```
Példa:  LEA     DI, VALUE1      ;A célkarakterlánc betöltése
        LEA     SI, VALUE2      ;A forráskarakterlánc betöltése
        REPE    CMPSB           ;Ismétlés, ha a byte-onkénti összehasonlítás
                                eredménye egyenlő
```

RET (80286/80386)

Visszatérés az eljárásból (RETurn)
Utasítás: RET
Tipikus órajel: (80286) 15-55, (80386) 10-68
Az utasítás leírása: Visszaadja a vezérlést a NEAR (közeli) vagy FAR (távoli) hívónak
Az utasítás működése: A RET utasítás használatakor a hívó program végrehajtása folytatódhat, mivel ez az utasítás annak a visszatérési címnek adja át a program vezérlését, ami az eljárás hívásakor a verembe kerül. Ez a címet rendszerint a CALL utasítás helyezi be a verembe. A szegmensen belüli RET azt a két byte-ot, amit a verem tetején talál, az utasításmutatónak (IP) adja át. Ez a két byte a következő végrehajtandó utasítás eltolási címét képezi. A szegmensek közötti RET utasítás abban hasonlít a szegmensen belüli RET-hez, hogy a verem legfelső két byte-ját áthelyezi az utasításmutatóba. (Ez lesz a következő végrehajtandó utasítás eltolási címe.) A verem következő két byte-ja, amelyek a következő utasítás kódszegmens címét jelentik, a CS regiszterbe kerülnek. A szegmensen belüli RET utasítás végrehajtásához és a veremmutató megváltoztatásához tehát a következő műveletekre van szükség:

A verem legfelső két byte-jának áthelyezése az utasításmutatóba.
A verem következő két byte-jának áthelyezése a veremmutatóba.

Ez a két utolsó byte a veremmutató utolsó pozíciója, amely a CALL utasítást megelőzően kerülhetett a verembe. A szegmensek közötti RET utasítás végrehajtásához és a veremmutató megváltoztatásához a következő műveletekre van szükség:

A verem legfelső két byte-jának áthelyezése az utasításmutatóba.
(Ez a következő végrehajtandó utasítás eltolási címe.)
A verem következő két byte-jának áthelyezése a CS regiszterbe, ami a következő végrehajtandó utasítás kódszegmens címét jelenti.

A verem következő két byte-ja, amelyek a CALL utasítás hívását megelőzően a veremre helyezett paramétereket jelentik, a veremmutatóba kerül.
Szintaxis: RET (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Bizonyos esetekben általános védelmi kizárás, leíróelem nincs jelen kizárás és veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Ha a veremműveletek során a verem 0FFFFH-ról 0-ra fordul át, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: 48 bites ESP és 32 bites EIP szerepel a veremről való leemeléskor.
Példa: RET

SAHF (80286/80386)

AH rátöltése a STÁTUSZ alsó 8 bitjére (Store AH in Flags)
Utasítás: SAHF
Tipikus órajel: (80286) 2, (80386) 3
Az utasítás leírása: AH-t rátölti az SF, ZF, xx, AF, xx, PF, xx, CF jelzőbitekre
Az utasítás működése: Az AH regiszterben tárolt értéket rátölti az előbb felsorolt a 7., 6., 4., 2. és 0. sorszámú bitekre.
Szintaxis: SAHF (operandus nélkül)
Befolyásolt jelzők: ZF, SF, AF, PF, CF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: SAHF

SAL/SAR/SHL/SHR (80286/80386)

Bitléptető utasítások (Shift Arimetical Left/Right, SHift Right/Left)
Utasítás: SAL/SAR/SHL/SHR
Tipikus órajel: (80286) 2-5, (80386) 3-7
Az utasítás leírása: Bitek léptetése
Az utasítás működése: Az SAL és SHL utasítások az előírt operandus bitjeit balra léptetik. A legnagyobb helyértékű bit CF-be (átvitelijelző bit) kerül és a legkisebb helyértékű bit törlődik. Az SAR és SHR utasítások az előírt operandus bitjeit jobbra léptetik. A legkisebb helyértékű bit CF-be kerül. Az SAR utasítás, ami előjeles osztást valósít meg, a legnagyobb helyértékű bitet változatlanul hagyja (ismétli). Az SHR, előjel nélküli osztást végez el úgy, hogy a legnagyobb helyértékű bitet nullázza. A bitléptetések ismétlésének számát az utasítás második operandusával lehet megadni. Ez a szám egyaránt lehet azonnali címzésű operandus, vagy a CL regiszter. A 80286/80386-os bitléptető utasításoknál a léptetések maximális száma 31. Ha ennél nagyobb értéket használunk, akkor a számlálónak csak a legalacsonyabb öt bitjét értelmezi az utasítás. Az OF (túlcsordulásjelző bit) beállítása 1-re csak akkor történik meg, ha egyszeri bitléptetési műveletet hajtunk végre. Bitléptetés balra műveleteknél OF törlődik, ha az eredmény legnagyobb helyértékű bitjének értéke és CF értéke megegyezik, egyébként OF értéke 1 lesz. Az SAR utasítás esetében OF értéke zérus lesz, ha egyszeri bitléptetési műveletről van szó. Az SHR utasítás az eredeti operandus legnagyobb helyértékű bitjének értékére állítja be OF-et.
Szintaxis: SAL *cél, számláló*
SHL *cél, számláló*
SAL *cél,* 1
SAL *cél,* CL
SAR *cél, számláló*
SHL *cél,* 1
SHL *cél,* CL
SHR *cél, számláló*
Befolyásolt jelzők: OF (csak egyszeri bitléptetés esetén), CF, ZF, OF, PF, SF
Definiálatlan jelzők: AF
Kizárások védett üzemmód esetén: Ha az operandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.

Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
SHL     MEMRY_BTE,1
SHL     AX,CL
SHL     TABLE[BX][DI],CL
SAR     BL,1
SAR     MEMRY_BTE,REP_VAL
SAR     DH,CL
SAR     CX,1
SAR     MEMRY_BTE,1
SAR     AX,CL
```

SBB (80286/80386)

Kivonás átvitellel (SuBtraction with Borrow)
Utasítás: SBB
Tipikus órajel: (80286) 2-3, (80386) 2-7
Az utasítás leírása: Egész számok kivonása átvitellel
Az utasítás működése: Az SBB utasítás a második operandust hozzáadja CF-hez (átviteljelző bit) és ezt az eredményt kivonja az első operandusból. Az eredmény az első operandusban kerül elhelyezésre.
Szintaxis: SBB *cél,forrás*
Befolyásolt jelzők: OF, SF, ZF, PF, CF, AF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
SBB     AX,BX
SBB     DX,MEMRY_WRD
SBB     TABLE[BX][DI],SI
SBB     AL,3
SBB     EAX,EBX          (Csak 80386-os esetében)
SBB     ECX,0ABCD1234H   (Csak 80386-os esetében)
```

SCAS/SCASB/SCASW (80286/80386)

Karakterlánc adatok összehasonlítása (SCAn String Byte/String Word)
Utasítás: SCAS/SCASB/SCASW
Tipikus órajel: 7
Az utasítás leírása: A byte (AL), ill. a szó (AX) összehasonlítása az ES:[DI]-vel vagy ES:[EDI]-vel megcímzett byte (szó) értékével.
Az utasítás működése: Az SCAS utasítás kivonja azt a byte= vagy szóoperandust, amelyre ES:DI ([ES:EDI]) mutat, vagy az AL, vagy az AX regiszterből és az eredmény figyelmen kívül hagyva a megfelelő jelzőket beállítja. A szegmensek felülírása nincs engedélyezve. A memóriaoperandusnak az ES szegmensből címezhetőnek kell lennie. Az összehasonlítás elvégzése után DI (EDI) értéke nő, ha DF = 0, és csökken, ha DF = 1. Byte=operandus előírása esetén DI (EDI) léptetése 1, szóoperandus esetén 2.

Szintaxis: SCAS *cél_karakterlánc*
SCASB (operandus nélkül)
SCASW (operandus nélkül)
Befolyásolt jelzők: OF, SF, ZF, PF, CF, AF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

```
Példa:  MOV     CL,10H          ;16 byte vizsgálata
        LEA     DI,VALUE1       ;Karakterlánc helyének bekérése
        MOV     AL,'S'          ;A karakter, amit vizsgálni kell
        REPNE   SCASB           ;Ismétlés, ha nem egyenlő, byte-ok vizsgálata
```

SGDT/SIDT (80286/80386)

A globális/megszakító leíróelemtáblázat-regiszter tárolása (Store Global/Interrupt Descriptor Table)
Utasítás: SGDT/SIDT
Tipikus órajel: (80286) 11, 12, (80386) 9
Az utasítás leírása: A SGDT vagy SIDT regiszter tárolása a memóriában
Az utasítás működése: Mind az SGDT, mind az SIDT utasítás betölti a leíróelemtáblázat-regiszter tartalmát a memóriának abba a 6 byte-jába, amelyre a céloperandus rámutat. A regiszter LIMIT (határ) mezője az effektív cím első szavára, a következő három byte-ra pedig a regiszter BASE mezője kerül. Az utolsó byte definiálatlan.
Szintaxis: SGDT *cél*
SIDT *cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a céloperandus egy regiszter, definiálatlan műveletkód kizárás áll elő. Ha a céloperadndus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a céloperandus egy regiszter, definiálatlan műveletkód kizárás áll elő. Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.

```
Példa:  SGDT    MEM_WRD
        SIDT    MEM_WRD
```

SLDT (80286/80386)

A lokális leíróelemtáblázat-regiszter tárolása (Store Local Descriptor Table register)
Utasítás: SLDT
Tipikus órajel: 2
Az utasítás leírása: Az LDT regiszter betöltése egy effektív címzésű operandus által megjelölt helyre
Az utasítás működése: Az SLDT utasítás abba a 2 byte-os regiszterbe vagy memóriahelyre tárolja a lokális leíróelemtáblázat-regiszter tartalmát, amelyre az effektív címzésű operandus rámutat.

Szintaxis: SLDT *cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a céloperandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: INT 6 áll elő. A valós címzési módban SLDT-t nem ismeri fel a processzor.
Példa: SLDT BP

SMSW (80286/80386)

A gépi státusz-szó tárolása (Store Machine Status Word)
Utasítás: SMSW
Tipikus órajel: 2
Az utasítás leírása: Az effektív címzésű szóval megjelölt helyre tárolja a gépi státusz-szót
Az utasítás működése: A céloperandus (effektív címzés, 2 byte-os regiszter vagy memória) megkapja a gépi státusz-szó másolatát.
Szintaxis: SMSW *cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a céloperandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Példa: SMSW BP

STC (80286/80386)

Az átviteljelző bit (CF) beállítása (SeT Carry flag)
Utasítás: STC
Tipikus órajel: 2
Az utasítás leírása: Beállítja az átviteljelző bitet
Az utasítás működése: Az STC utasítás az átviteljelző bitet 1-re állítja be
Szintaxis: STC (operandus nélkül)
Befolyásolt jelzők: CF = 1
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: STC

STD (80286/80386)

A irányjelző bit beállítása (SeT Direction flag)
Utasítás: STD
Tipikus órajel: 2

Az utasítás leírása: A irányjelző (DF) bit beállítása
Az utasítás működése: Az STD utasítás DF-et 1-re állítja be, amelynek hatására az ezt követő karakterlánc utasítások csökkentik a megfelelő SI és/vagy DI regiszterek értékét.
Szintaxis: STD (operandus nélkül)
Befolyásolt jelzők: DF = 1
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa: STD

STI (80286/80386)

A megszakítás-engedélyező bit beállítása (SeT Interrupt enable flag)
Utasítás: STI
Tipikus órajel: (80286) 2, (80386) 3
Az utasítás leírása: Beállítja az megszakítás-engedélyező bitet.
Az utasítás működése: Az STI utasítás az megszakítás-engedélyező bitet 1-re állítja be, így lehetővé válik a következő utasítás végrehajtása után maszkolható külső megszakítások kibocsátása.
Szintaxis: STI (operandus nélkül)
Befolyásolt jelzők: IF = 1
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az éppen érvényes privilégium-szint az I/O privilégiumszintnél alacsonyabb, általános veremhiba kizárás áll elő.
Kizárások valós címzési mód esetén: Nincs
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: STI

STOS/STOSB/STOSW (80286/80386)

String adatok tárolása (STOre String Byte/String Word)
Utasítás: STOS/STOSB/STOSW
Tipikus órajel: (80286) 3, (80386) 4
Az utasítás leírása: AL byte-ot vagy AX szót az ES:DI, vagy az ES:[EDI] által megjelölt helyre tárolja
Az utasítás működése: Az STOS utasítás átadja az AL vagy AX regiszterek valamelyikének tartalmát annak a memóriabyte-nak vagy -szónak, amelyre ES:DI ([ES:EDI]) rámutat. A céloperandusnak az ES regiszterből címezhetőnek kell lennie, mivel a szegmenselnyomás nincs engedélyezve. Ha DF a zérus értéket veszi fel, akkor DI (EDI) automatikusan növekszik. Ha DF 1, DI (EDI) csökken. A növekmény/csökkenés mértéke byte esetében 1, szó esetében 2.
Szintaxis: STOS *cél_karakterlánc*
STOSB (operandus nélkül)
STOSW (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a céloperandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.

Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:
```
        MOV     ECX,0FFCCAAH    (Csak 80386-os esetében)
        LEA     EDI,VARIABLE
        MOV     AX,'-'
REP     STOSB
```

STR (80286/80386)

A **taszkregiszter tárolása** (Store Task Register)
Utasítás: STR
Tipikus órajel: (80286) 2, (80386) 23-27
Az utasítás leírása: A taszkregisztert az EA szóval megjelölt helyre tárolja
Az utasítás működése: Az STR utasítás a taszkregiszter tartalmát átmásolja abba a 2 byte-os regiszterbe vagy arra memóriahelyre, amelyre az effektív címoperandus mutat.
Szintaxis: STR *cél*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a céloperandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: INT 6 áll elő. Valós címzési módban az STR utasítást nem ismeri fel a processzor.
Példa:
```
        STR     BP
```

SUB (80286/80386)

Kivonás (SUBtraction)
Utasítás: SUB
Tipikus órajel: 2-7
Az utasítás leírása: Egész számok kivonását hajtja végre
Az utasítás működése: A SUB utasítás a forrásoperandust kivonja a céloperandusból, és az eredményt a céloperandusba teszi.
Szintaxis: SUB *cél, forrás*
Befolyásolt jelzők: OF, SF, ZF, PF, AF, CF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:
```
        SUB     AX,BX
        SUB     EAX,EDX             (Csak 80386-os esetében)
        SUB     DX,MEMRY_WRD
        SUB     MEMRY_WRD,AX
        SUB     MEM_BTE,7
        SUB             NUMBER,0FC981576H   (Csak 80386-os esetében)
```

TEST (80286/80386)

Bitenkénti logikai ÉS művelet
Utasítás: TEST
Tipikus órajel: (80286) 2-6, (80386) 2-5
Az utasítás leírása: Logikai összehasonlítást végez
Az utasítás működése: A TEST utasítás két operandus bitenkénti AND (ÉS) műveletét hajtja végre, amelynek eredménye 1, ha az egymásnak megfelelő mindkét bit 1, egyébként az eredmény zérus. Az operandusok nem módosulnak, csak a jelzők.
Szintaxis: TEST *cél, forrás*
Befolyásolt jelzők: OF = 0, CF = 0, SF, ZF, PF
Definiálatlan jelzők: AF
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
TEST    AX,BX
TEST    EAX,EBX              (Csak 80386-os esetében)
TEST    MEM_BTE,6
TEST    TABLE[BX][DI],CX
TEST    TABLE[BX][DI],ECX    (Csak 80386-os esetében)
```

VERR/VERW (80286/80386)

Szegmens ellenőrzése íráshoz vagy olvasáshoz (VErify a segment for Reading or Writing)
Utasítás: VERR/VERW
Tipikus órajel: (80286) 14-16, (80386) 10-16
Az utasítás leírása: Ha a szegmens írható/olvasható, ZF-et 1-re állítja be
Az utasítás működése: A VERR és VERW utasítások megállapítják, hogy az a szegmens, amelyre a 2 byte-os regiszter- vagy memória- operandus szelektor rámutat, az éppen érvényes privilégiumszintről elérhető-e. Az utasítás azt is megállapítja, hogy a szegmens írható, ill. olvasható-e. Ha a szegmens elérhető, ZF 1-re, egyébként pedig zérusra áll be.
Szintaxis: VERR *cél_olvasható szelektor*
VERW *cél_írható szelektor*
Befolyásolt jelzők: ZF
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az operandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: INT 6 áll elő. A valós címzési módban a VERR/VERW utasításokat nem ismeri fel a processzor.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa:

```
VERR    BP
VERW    SELECTR
```

WAIT (80286/80386)

Várakozás (WAIT)
Utasítás: WAIT
Tipikus órajel: (80286) 3, (80386) 6 (minimum)
Az utasítás leírása: A processzor várakozó állapotba lép mindaddig, amíg egy külső megszakítás nem következik be
Az utasítás működése: A WAIT utasítás hatására a 80286/80386-os mindaddig várakozik, amíg egy külső megszakítás bekövetkezik. Az utasítás célja a matematikai társprocesszoral való együttműködés szinkronizálása.
Szintaxis: WAIT (operandus nélkül)
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az MSW státusz-szóban a taszkcsere bit 1, akkor az utasítás hatására a nincs matematikai társprocesszor kizárás jön létre. Ha a matematikai társprocesszor hibás utasítást kap, akkor a matematikai társprocesszor hiba jön létre. A kizárások feldolgozása a védett és a valós címzési módban megeggyezik.
Megjegyzés a 80386-os processzorral kapcsolatban: Nincs kivétel
Példa: WAIT

XCHG (80286/80386)

Operandusok felcserélése (eXCHanGe)
Utasítás: XCHG
Tipikus órajel: 3-5
Az utasítás leírása: Szavak vagy byte-ok cserélése
Az utasítás működése: Az XCHG utasítás a byte- vagy szó forrás- operandust az ezzel egyező adattípusú céloperandussal felcseréli.
Szintaxis: XCHG *cél, forrás*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha az operandus írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites
Példa:

```
XCHG    AX,BX
XCHG    DH,DATA_WRD
XCHG    AL,DL
XCHG    EAX,EBX          (Csak 80386-os esetében)
```

XLAT (80286/80386)

Áthelyezés (table look-up trans(X)LATe)
Utasítás: XLAT
Tipikus órajel: 5
Az utasítás leírása: Hozzáférés a táblához
Az utasítás működése: Az XLAT utasítás a DS:BX (DS:EBX) által megcímzett adattáblázat AL-lel indexelt elemét kiolvassa és beírja AL-be.

Szintaxis: XLAT *táblázat*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

```
Példa:  LEA     BX,MYTABLE      ;A táblázat helyzete
        MOV     AL,INDEX        ;A táblázaton belüli eltolási érték
        XLAT    MYTABLE         ;A visszakapott értéket AL-ben találjuk
```

XOR (80286/80386)

Logikai kizáró VAGY (logical eXclusive OR)
Utasítás: XOR
Tipikus órajel: (80286) 2-3, (80386) 2-7
Az utasítás leírása: A logikai kizáró VAGY műveletének végrehajtása
Az utasítás működése: Az XOR utasítás a forrás- és céloperandus egymásnak megfelelő bitjeit összehasonlítja és bitenkénti kizáró VAGY műveletet hajt végre. Az eredmény minden olyan pozícióban zérus, ahol az összehasonlított operandusok mindegyike 1 vagy zérus. Az XOR művelet eredménye azokban a pozícióban 1, ahol az operandusok ellenkezőek.
Szintaxis: XOR *cél, forrás*
Befolyásolt jelzők: OF = 0, CF = 0, SF, ZF, PF
Definiálatlan jelzők: AF
Kizárások védett üzemmód esetén: Ha az eredmény írásvédett szegmensben van, általános védelmi kizárás keletkezik. Ha a CS, DS vagy ES szegmens illegális memóriaoperandus effektív címét tartalmazza, általános védelmi kizárás keletkezik. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Megjegyzés a 80386-os processzorral kapcsolatban: A szó mérete 32 bites

```
Példa:  XOR     AX,MEMRY_WRD
        XOR     AH,BL
        XOR     TABLE[BX][DI],AX
        XOR     EAX,ECX             (Csak 80386-os esetében)
        XOR     EDX,0FCAD1579H      (Csak 80386-os esetében)
```

3.6 A 80386-OS UTASÍTÁSKÉSZLETE

BSF/BSR (80386)

Bitvizsgálat előre/hátra (Bit Scan Forward/Reverse)
Utasítás: BSF/BSR
Tipikus órajel: 10+3n
Az utasítás leírása: Bitvizsgálatot végez előre/hátra
Az utasítás működése: A BSF és BSR utasítások a forrásoperandusban (16 vagy 32 bit) megkeresik az első 1-re beállított bit helyét, és ennek indexét a célregiszterben elhelyezik. A BSF utasítás zérus indexű pozíciótól kezdve, balról jobbra haladva végzi a keresést. A BSR utasítás jobbról balra haladva végzi a keresést 16 bites forrásoperandus esetén a 15. pozíciótól, 32 bites forrásoperandus esetén a 31. pozíciótól kezdve. Ha a teljes szó zérusokból áll, ZF értéke 1 lesz. Ha létezik 1 értékű bit, ZF zérust vesz fel. Ha az utasítás nem talál 1 értékű bitet, a céloperandus definiálatlan marad.
Szintaxis: BSF *cél_regiszter, forrás_operandus*
BSR *cél_regiszter, forrás_operandus*
Befolyásolt jelzők: ZF
Definiálatlan jelzők: OF, SF, AF, PF, CF
Kizárások védett üzemmód esetén: Ha az operandus a szegmenshatár megsértése vagy az elérési jog megszegése miatt nem használható, 13-as általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: 13-as általános védelmi kizárás áll elő, ha olyan operandusra hivatkozunk, ami a szegmenshatáron (0FFFFH) kívülre esik.
Példa:
```
BSF     AX,MEM_WORD
BSR     EAX,ECX
```

BT/BTS/BTR/BTC (80386)

Bitvizsgálat, bit beállítása zérusra/1-re, komplemens képzés (BiT test/Set/Reset/Complement)
Utasítás: BT/BTS/BTR/BTC
Tipikus órajel: 3,13
Az utasítás leírása: nálló biteken műveleteket hajtanak végre.
Az utasítás működése: A BT, BTS, BTR, BTC utasítások memórián vagy regiszteren belül az egyes önálló biteken hajtanak végre műveleteket. Ha regisztert írunk elő, az lehet akár 16 vagy 32 bites lánc is. A kiválasztott bit az utasításon belül azonnali címzésű konstanssal vagy egy általános regiszterben tárolt értékkel határozható meg. (Ezt az operandust módként kell venni az operandus méretéhez.) Az azonnali címzésű bit helyzetének tartománya 16 bites lánc esetén 0–15, míg 32 bites lánc esetében 0–31 lehet. Mind a négy utasítás a kiválasztott bit értékét először CF-nek adja át. A kiválasztott utasításnak megfelelően a bit új értéket kap (kivéve a bitvizsgáló utasítást).
Szintaxis: BT *regiszter/memória, szelektor*
BTS *regiszter/memória, szelektor*
BTR *regiszter/memória, szelektor*
BTC *regiszter/memória, szelektor*
Befolyásolt jelzők: CF a kiválasztott bit értékét veszi fel
Definiálatlan jelzők: OF, SF, ZF, AF, PF
Kizárások védett üzemmód esetén: Ha az operandus a szegmeshatár megsértése vagy az elérési jog megszegése miatt nem használható, 13-as általános védelmi kizárás áll elő.

Kizárások valós címzési mód esetén: 13-as általános védelmi kizárás áll elő, ha olyan operandusra történik hivatkozás, ami a szegmenshatáron (0FFFFH) kívülre esik.

```
Példa:  BT      AL,BL           ;BL = helyzet
        BT      MEM_WRD,0BH     ;0BH = azonnali címzésű helyzet
        BTS     AX,CL           ;CL = helyzet
        BTS     INFO[BX],7H     ;7H = azonnali címzésű helyzet
        BTR     AL,AL           ;AL = helyzet
        BTR     TABLE[SI],0EH   ;0EH = azonnali címzésű helyzet
        BTC     BL,AL           ;AL = helyzet
        BTC     BUFF[BX],2H     ;2H = azonnali címzésű helyzet
```

IBTS (80386)

Bitlánc beszúrása (Insert BiT String)
Utasítás: IBTS
Tipikus órajel: 12/19
Az utasítás leírása: A bitek beszúrása a regiszterbe vagy a memóriába
Az utasítás működése: Az IBTS utasítás egy register alacsony helyértékű bitjeit beilleszti egy másik regiszterbe vagy memória- helyre anélkül, annak bitjeit akármelyik oldalon megváltoztatná. Az IBTS utasításnak négy operandusa van: a bitlánc báziscíme, a beillesztendő alstring kezdőcímének bithelyzete, az alstring hossza és az a regiszter, ami a beillesztett értéket megkapja. A báziscím regiszter- vagy memóriacím lehet. Az eltolást az EAX regiszter (AX 16 bites operandus esetén) tárolja. A mezőhosszt a CL regiszterben kell elhelyezni. Az utolsó mező írja elő azt az általános regisztert, amibe az érték beillesztése történik.
Szintaxis: IBTS *bázis, helyzet, hossz, forrás*
IBTS *reg/memória, (E)AX, CL, regiszter*
Befolyásolt jelzők: OF, SF, ZF, AF, PF
Definiálatlan jelzők: CF
Kizárások védett üzemmód esetén: Ha az operandus a szegmenshatár megsértése vagy az elérési.jog megszegése miatt nem használható, 13-as általános védelmi kizárás áll elő.
Kizárások valós címzési mód esetén: 13-as általános védelmi kizárás áll elő, ha olyan operandusra történik hivatkozás, ami a szegmenshatáron (0FFFFH) kívülre esik.

```
Példa:  IBTS    BX,AX,CL,DX
```

MOV CRn (80386)

A kontrollregiszterek betöltése és tárolása (Load and store control registers)
Utasítás: MOV CRn,regiszter
MOV regiszter,CRn**Tipikus órajel:** 2-4
Az utasítás leírása: CRn regiszter betöltése
CRn tárolása regiszterbe
Az utasítás működése: Az utasítás a vezérlőregiszterek (CRn) betöltését vagy azok tárolását végzi el. Ezek az utasítások mindig 32 bites operandusokat használnak és csak a CR0, CR2 és CR3 definiált a 80386-os számára.
Szintaxis: MOV *CRn, forrás_operandus*
MOV *cél, CRn*
n = 0,2,3
Befolyásolt jelzők: OF, ZF, SF, PF, AF
Definiálatlan jelzők: CF
Kizárások védett üzemmód esetén: A fenti utasítások privilegizáltak. Ha zérustól eltérő privilégiumszinten kerülnek végrehajtásra, védelmi hiba áll elő.

Kizárások valós címzési mód esetén: Nincs
Példa: MOV CR2,EAX
MOV EBX,CR3

MOV DRn (80386)

A nyomkövető regiszterek betöltése és tárolása (Load and store debug registers)
Utasítás: MOV Dreg,reg
MOV reg,Dreg
Tipikus órajel: 2-4
Az utasítás leírása: Betölti a nyomkövető regisztert
Tárolja a nyomkövető regiszter tartalmát
Az utasítás működése: A MOV DRn utasítás 32 bites értéket tölt be a nyomkövető regiszterbe, vagy annak tartalmát tárolja.
Szintaxis: MOV D*reg, regiszter*
MOV *regiszter, Dreg*
Befolyásolt jelzők: OF, ZF, SF, AF, PF
Definiálatlan jelzők: CF
Kizárások védett üzemmód esetén: A fenti utasítások privilegizáltak. Ha zérustól eltérő privilégiumszinten kerülnek végrehajtásra, védelmi hiba áll elő.
Kizárások valós címzési mód esetén: Nincs
Példa: MOV DR4,EAX
MOV EBX,DR5

MOV TRn (80386)

Tesztregiszterek betöltése és tárolása (load and store test registers)
Utasítás: MOV Treg,regiszter
MOV regiszter,Treg
Tipikus órajel: 2-4
Az utasítás leírása: Betölti a tesztregisztert
Tárolja a tesztregisztert tartalmát
Az utasítás működése: A MOV TRn utasítás segítségével teszt regiszterek betöltését és tartalmának tárolását végezhetjük el. Csak a TR6 és TR7 utasítás definiált a 80386-os számára.
Szintaxis: MOV *Treg, regiszter*
MOV *regiszter, Treg*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: A fenti utasítások privilegizáltak. Ha zérustól eltérő privilégiumszinten kerülnek végrehajtásra, védelmi hiba áll elő.
Kizárások valós címzési mód esetén: Nincs
Példa: MOV TR6,EAX

MOVZX/MOVSX (80386)

Regiszter/memória átmásolása előjel/zérus kiterjesztésű 16/32 bites regiszterbe (MOVe reg/mem to Sign/Zero extension 16/32-bit register)
Utasítás: MOVZX/MOVSX
Tipikus órajel: 3-6
Az utasítás leírása: A forrás átmásolása az előjel/zérus kiterjesztésű 16 vagy 32 bites regiszterbe

Az utasítás működése: A MOVZX és MOVSX utasítások felhasználására akkor van szükség, ha 8 vagy 16 bites regiszter- vagy memória- értékeket előjel vagy zérus kiterjesztéssel akarunk átmásolni 16 vagy 32 bites regiszterekbe.
Szintaxis: MOVZX *regiszter, regiszter/memória*
MOVSX *regiszter, regiszter/memória*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: Az általános védelmi kizárás áll elő, ha a memóriaoperandus megsérti a szegmens határát vagy a a védett üzemmódot. Ha az SS szegmens illegális címet tartalmaz, akkor veremhiba kizárás keletkezik.
Kizárások valós címzési mód esetén: Ha a szóoperandus a 0FFFFH helyzetnél van, INT 13 áll elő.
Példa:

```
MOVZX   EAX,BX
MOVZX   CX,AL
MOVSX   AX,BX                ;Normál áthelyezés
MOVSX   ECX,CL
MOVSX   EAX,MYMEMORY
```

SETfeltétel (80386)

Byte-beállítás feltétellel (Byte SET on condition)

Utasítás	Leírás
SET feltétel	A feltétel vizsgálatától függően beállítja a byte-ot Byte-beállítás ha:
SETO (SET if Overflow)	van túlcsordulás
SETNO (SET if Not Overflow)	nincs túlcsordulás
SETB (SET if Below)	előjel nélkül kisebb
SETNAE (SET if Not Above or Equal)	előjel nélkül nem nagyobb vagy egyenlő
SETNB (SET if Not Below)	előjel nélkül nem kisebb
SETAE (SET if Above or Equal)	előjel nélkül nagyobb vagy egyenlő
SETE (SET if Equal)	egyenlő
SETZ (SET if Zero)	zérus
SETNE (SET if Not Equal)	nem egyenlő
SETNZ (SET if Not Zero)	nem zérus
SETBE (SET if Below or Equal)	előjel nélkül kisebb vagy egyenlő
SETNA (SET if Not Above)	előjel nélkül nem nagyobb
SETNBE (SET if Not Below or Equal)	előjel nélkül nem kisebb vagy egyenlő
SETA (SET if Above)	előjel nélkül nagyobb
SETS (SET if Sign)	van előjel
SETNS (SET if Not Sign)	nincs előjel
SETP (SET if Parity)	van paritás
SETPE (SET if Parity Even)	paritás páros
SETNP (SET if Not Parity)	nincs paritás
SETPO (SET if Parity Odd)	paritás páratlan
SETL (SET if Less)	előjelesen kisebb
SETNGE (SET if Not Greater or Equal) előjelesen nem nagyobb vagy egyenlő	
SETNL (SET if Not Less)	előjelesen nem kisebb
SETGE (SET if Greater or Equal)	előjelesen nagyobb vagy egyenlő
SETLE (SET if Less or Equal)	előjelesen kisebb vagy egyenlő
SETNG (SET if Not Greater)	előjelesen nem nagyobb
SETNLE (SET if Not Less or Equal)	előjelesen nem kisebb vagy egyenlő
SETG (SET if Greater)	előjelesen nagyobb

Tipikus órajel: Valamennyi utasítás órajele N/A
Az utasítás működése: A SET utasítás az előbb felsorolt 16 feltétel valamelyikét alapul véve beállítja a kiválasztott byte-ot 0-ra vagy 1-re. Az utasítás egyedüli operandusa egy 1 byte-os regiszter vagy memória. A következő értékadó utasítások érvényesek: if SET *feltétel* igaz, akkor *reg/mem* = 1, egyébként: *reg/mem* = 0.
Szintaxis: SET *feltétel reg/mem*
Befolyásolt jelzők: Nincs
Definiálatlan jelzők: Nincs
Kizárások védett üzemmód esetén: A fenti utasítások privilegizáltak. Ha zérustól eltérő privilégiumszinten kerülnek végrehajtásra, védelmi hiba áll elő.
Kizárások valós címzési mód esetén: Nincs
Példa: SETNO ARG

SHLD/SHRD (80386)

Duplapontos érték bitléptető utasítása (SHift Left/Right Double)
Utasítás: SHLD/SHRD
Tipikus órajel: 3-7
Az utasítás leírása: Duplapontos érték bitléptetési művelete
Az utasítás működése: Az SHLD/SHRD utasítások dupla pontosságú mennyiségek bitléptetését végzik balra vagy jobbra annak érdekében, hogy egyszeres pontosságú érték álljon elő. A regiszterben vagy memóriahely operandusban annyi bitléptetést tudunk elvégezni, amennyit a számlálóoperandusban megadunk. Ez a bitléptetés abban különbözik az egyszeres pontosságú bitléptetéstől, hogy ez az utasítás a beléptett bitek kibocsátához a regiszteroperandust használja. A bemeneti operandust a regiszter vagy memóriahely operandusa írja elő. A regisztermezőben kell megadni azokat a biteket, amelyeken a léptetési műveletet el akarjuk végezni. Az azonnali címzésű operandus, egy CL tartalmazza a léptetések számát. A SHLD utasítás esetén a regiszter- vagy memóriaoperandus tartalmazza a dupla pontosságú érték magas helyértékű bitjeit, a regisztermező pedig az alacsony helyértékű biteket. A bitléptetés balra műveletnél a regiszteroperandus magas helyértékű bitjei a jobb oldalon lépnek be a regiszteroperandusból (az alacsony helyértékű pozícióban). Az eredmény a regiszter- vagy memóriaoperandusban kerül tárolásra. Az SHRD utasítás esetén a regiszter- vagy memóriaoperandus a dupla pontosságú érték alacsony helyértékű bitjeit, míg a regiszter- operandus a magas helyértékű bitjeit tartalmazza. A bitléptetés jobbra műveletnél a regiszteroperandus alacsony helyértékű bitjei a bal oldalon lépnek be a regiszteroperandusból (a magas helyértékű pozícióban). Az eredményt a regiszter- vagy memóriaoperandus tárolja.
Szintaxis: SHLD *reg/mem, reg, azonnali címzésű érték*
SHRD *reg/mem, reg,* CL
Befolyásolt jelzők: CF a legutoljára kiléptetett bit értékét veszi fel. OF 1-re áll be, ha a legutolsó bitléptetés túlcsordulást okozott. SF, ZF és PF az eredmény értékétől függően áll be.
Definiálatlan jelzők: AF
Kizárások védett üzemmód esetén: Nincs
Kizárások valós címzési mód esetén: Nincs
Példa:

```
SHLD    AX,BL,0AH       ;0AH = Azonnali címzésű bitléptetés
                        számláló
SHRD    AL,BL,CL        ;CL = Bitléptetés számláló
```

XBTS (80386)

Bitláncok kiemelése (eXtract BiT String)
Utasítás: XBTS
Tipikus órajel: 6-13
Az utasítás leírása: Bitláncok kiemelése regiszterbe jobbra igazítva
Az utasítás működése: Az XBTS utasítás bitláncok kiemelését szolgálja A képződött állánc az előírt regiszterben kibővítve és jobbra igazítva kerül tárolásra úgy, hogy a magas helyértékű bitek zérust vesznek fel. Az XBTS utasítás a következő négy operandussal rendelkezik:

a bitlánc báziscíme,
az állánc – amit kiemelünk – kezdetének bithelyzete,
az állánc hossza,
a regiszter, ami a beillesztett értéket tárolja.

A báziscím regiszter- vagy memóriacím lehet. A helyzetet az EAX (16 bites operandusnál AX) regiszterben kell tárolni. A mező szélességét a CL regiszterben kell megadni. Az utolsó mező azt az általános regisztert adja meg, ami a kiemelt értéket tartalmazza.
Szintaxis: XBTS *cél, bázis, helyzet, hossz*
XBTS *regiszter, reg/mem,* (E)AX, CL
Befolyásolt jelzők: OF, SF, ZF, AF, PF
Definiálatlan jelzők: CF
Kizárások védett üzemmód esetén: A fenti utasítások privilegizáltak. Ha zérustól eltérő privilégiumszinten kerülnek végrehajtásra, védelmi hiba áll elő.
Kizárások valós címzési mód esetén: 13-as általános védelmi kizárás áll elő, ha olyan operandusra történik hivatkozás, ami a szegmens határon (0FFFFH) kívülre esik.
Példa: XBTS AX,BX,AX,CL

4. A 80287/80387-ES MATEMATIKAI TÁRSPROCESSZOR

Ez a fejezet a 80287/80387-es matematikai társprocesszor jellemzőit ismerteti (lebegőpontos verem, státusz-szó, vezérlőszavak és a tagok, kizárási mutatók, adattípusok). Áttekintést adunk a 80287/80387-es processzor működéséről, numerikus feldolgozó képességéről és a kizárások kezeléséről. Referenciaként a 80287/80387-es utasításkészletének listáját is megadjuk.

4.1 A 80287/80387-ES MŰKÖDÉSE

A 80287/80387-es matematikai társprocesszort úgy tervezték meg, hogy a 80286/80386-os processzorral párhuzamosan tudjon működni. A 80287/80387-es utasításkészletében számos hatékony lebegőpontos művelet van. Amikor a 80286/80386-os egy lebegőpontos utasítással találkozik, a szükséges műveletkódot és memóriaoperandus címeket elküldi a 80287/80387-es részére. Ez a lépés felszabadítja a 80286/83386-os processzort a következő utasítás végrehajtása alól, amíg a 80287/80387-es ezzel párhuzamosan elvégzi a numerikus számítást. A 80287/80387-es processzor a 80286/80386-ba beépített külön adatcsatornán éri el a memóriát. A memóriahozzáférés során a védelmi szabályok megsértése esetén a megfelelő kizárás jön létre. Bizonyos 80286/80386-os műveletek esetében arra kell kényszeríteni a 80286/80386-os processzort, hogy várja meg a társprocesszor által előállított eredményt. A WAIT vagy FWAIT 80286/80386-os utasításokkal lehet a két processzor közötti szinkronizációt megvalósítani.

4.2 LEBEGŐPONTOS VEREM

A 80287/80387-es processzor esetében a verem nyolc darab 80 bites elemből áll (l. a 4–1. ábrát), amelyek mezőkre vannak felosztva. Ezen mezők azokra az ideiglenes valós adatformátumokra vonatkoznak, amelyeket a társprocesszor valamennyi veremműveletnél használ. A lebegőpontos verem önálló elemeit explicit vagy implicit módon címezhetjük. Néhány lebegőpontos utasítás alapértelmezése azonban a verem bizonyos elemeihez kötődik.

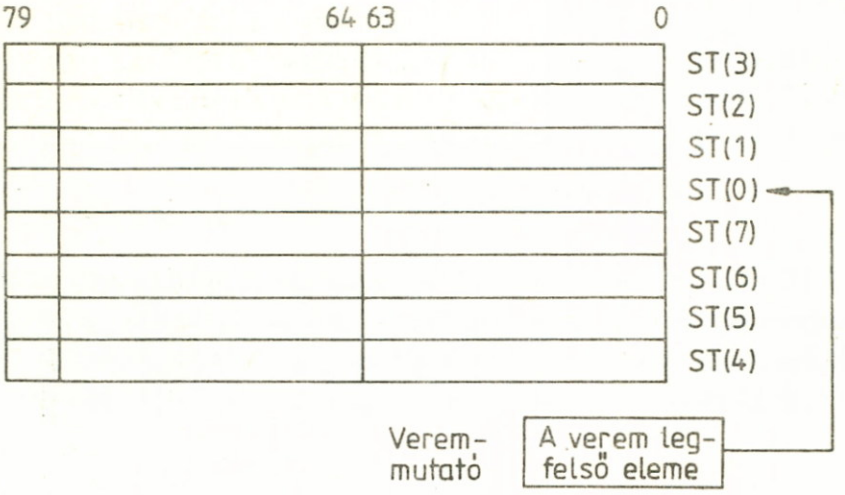
Például: FSQRT

Az FSQRT utasítás veszi az éppen érvényes 80287/80387-es legfelső veremregiszter (ST(0)) tartalmának négyzetgyökét, és ezt az értéket ráírja a veremre. Vannak azonban olyan utasítá-

sok is, amelyek lehetővé teszik, hogy a programozó válassza ki a veremelemet, mint a következő példánál:

FST(7)

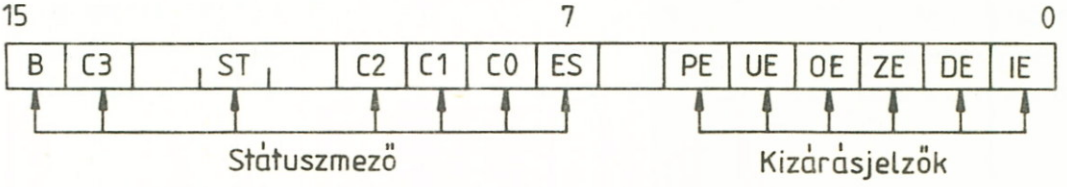
Az FST(7) utasítás veszi az éppen érvényes legfelső veremregiszter (ST(0)) értékét és a lebegőpontos verem hetedik elemébe tárolja.



4–1. ábra A 80287/80387-es verme

4.3 STÁTUSZ-SZÓ

A 80287/80387-es státusz-szóból (4–2. ábra) a társprocesszor általános állapotát tudhatjuk meg. A státusz-szó két mezőre osztható fel: a kizárásjelző bitmezőre és a státusz bitmezőre. A státusz-szó vizsgálatához előbb egy 80287/80387-es utasítással tárolnunk kell azt egy memóriahelyre, majd az egyes biteket a 80286/80386-os kód felhasználásával elemezhetjük.



4–2. ábra A 80287/80387-es státusz-szava

[B] Ez az 1 bites mező a 80287/80387-es processzor inaktív vagy aktív állapotát jelzi.

[ST] Ez a három bitből álló mező jelzi, hogy a verem nyolc eleme közül éppen melyik a verem teteje.

[ST] értékei:
000 – A 0. elem a veremtető
001 – Az 1. elem a veremtető
:
111 – A7. elem a veremtető.

[C3, C2, Cl, C0] Ez a négy 1 bites mező a verem tetején elhelyezkedő elemről további információkat ad. A mezők állapotától függően feltételes elágaztatások végeztethetők

el. (L. a 80287/80387-es utasításkészletének FCOM, FCOMP FCOMP, FTST, FXAM és FPREM utasításait, ahol a C3, C2, C1, C0 bitekről további információkat kaphatunk.)

[IR] (Interrupt Request): Ez az 1 bites megszakítási igényt jelző bitmező fejezi ki azt, hogy a 80287/80387-es megszakítási igénnyel lép fel a 80286/80386-os felé.

[PE] (Precision Exception): A pontossági kizárást jelző bitmező beállítása akkor szükséges, ha az eredményt kerekíteni kell a normál alak kialakítása miatt.

[UE] (Underflow Exception): Az alulcsordulási kizárást jelző 1 bites mező beállítása akkor történik meg, ha az eredmény túl kicsi ahhoz, hogy a kijelölt lebegőpontos formátumban normalizálás nélkül tárolásra kerülhessen.

[OE] (Overflow Exception): Ez az 1 bites túlcsordulási kizárást jelző mező minden esetben 1-re áll be, ha a kiszámított eredmény túl nagy ahhoz, hogy a kijelölt lebegőpontos formátumban elférjen.

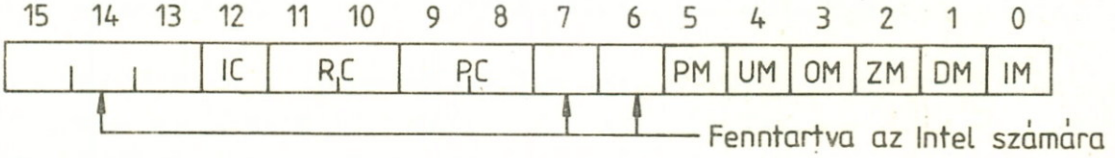
[ZE] (Zero divide Exception): Ez az 1 bites mező jelzi, ha zérussal osztunk vagy ha az osztandó zérustól különböző szám.

[DE] (Denormalized operand Exception): A denormalizált operandus kizárás jelző 1 bites mező, ami azt érzékeli, hogy az utasítást normalizálás nélküli operanduson kell-e végrehajtani.

[IE] (Invalid operation Exception): Ez az 1 bites mező illegális műveletek végrehajtását jelzi. (Pl.: negatív szám négyzetgyökének képzése vagy művelet végrehajtása olyan adattal, ami nem szám.)

4.4 VEZÉRLŐSZÓ (CONTROL WORD)

A 80287/80387-es vezérlőszava (4–3. ábra) a kizárási maszkokat, az megszakítás-engedélyezés maszkokat és néhány vezérlőbitet tartalmaz.



4–3. ábra A 80287/80387-es vezérlőszava

[IC] (Infinity Control): Ez az 1 bites mező jelzi, hogy a rendszer melyik típusú végtelen aritmetikát alkalmazza. Ha IC zérus, akkor projektívről, ha 1, affinról van szó.

[RC] (Rounding Control): Ezzel a 2 bites mezővel a következő négy kerekítési irány közül választhatunk:
- Kerekítés a legközelebbi vagy páros értékre
- Kerekítés pozitív felé
- Kerekítés negatív felé
- Kerekítés zérus felé

[PC] (Precision Control): Ez a 2 bites mező jelzi, hogy melyiket választottuk a következő háromféle pontosság közül:
- Ideiglenes valós (64 bites eredmény)
- Hosszú valós (53 bites eredmény)
- Rövid valós (24 bites eredmény)

Kizárási maszkok (Exception masks)

A vezérlőszóban található kizárási maszkok jelzik, hogy melyik kizárást kell a szabványos 80287/80387-es korrekciós tevékenységének elvégeznie (ez a maszkolható kizárásra példa), és melyik kizárásnak kell a 80287/80387-est megszakítójel előállítására késztetnie. (Ez a nem maszkolt kizárás.)§b A kizárási maszkok a következők:

[PM] (Precision Mask): Pontossági maszk

[UM] (Underflow Mask): Alulcsordulási maszk

[OM] (Overflow Mask): Túlcsordulási maszk

[ZM] (Zero-divide Mask): Zérus-osztás maszk

[DM] (Denormalized-operand Mask): Normalizálatlan operandusmaszk

[IM] (Invalid-operation Mask): Érvénytelen műveletmaszk

4.5 TOLDALÉKSZÓ (TAG WORD)

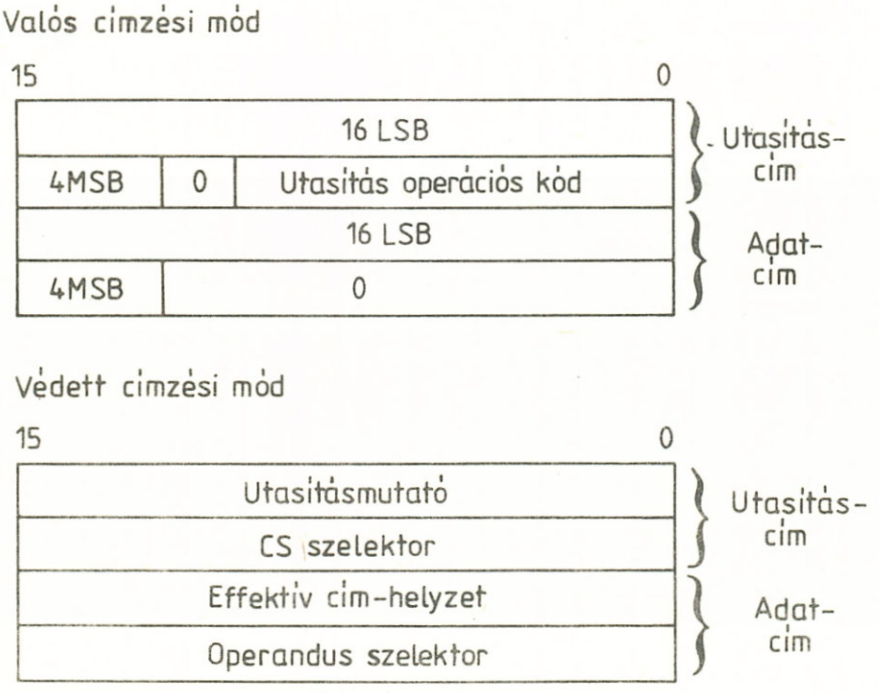
A toldalékszó olyan toldalékokat tartalmaz, amelyek a hozzájuk rendelt veremelem tartalmát írják le. A toldalékértékek a következők:
TAG(7) TAG(6) TAG(5) TAG(4) TAG(3) TAG(2) TAG(1) TAG(0)
A toldalék értékei:

00 = Érvényes (szabályos vagy nem szabályos)
01 = Zérus (igaz)
10 = Speciális (nem szám, végtelen vagy normalizálatlan)
11 = Üres

4.6 KIZÁRÁSI MUTATÓK

A kizárási mutatók lehetővé teszik, hogy a programozó saját kizáráskezelő programot írjon. Amikor a 80287/80387-es egy utasítást végrehajt, az utasítás címe a kizárási mutatókban kerül tárolásra. Ha a végrehajtott utasítás memóriaoperandusra is vonatkozik, akkor az operandus címét is tárolja a rendszer. A programozó olyan kizáráskezelő programot is írhat, ami a

mutatókat a memóriában tárolja, és így megtudhatja, hogy mi okozta a hibát. Ez az információ két formátumban kerül tárolásra (l. a 4–4. ábrát). Az egyik a valós címzési mód, a másik a védett címzési mód számára.
A 80387-es rendszerbusszal támogatja az adat-interface-t.



4–4. ábra A 80287/80387-es utasítás- és adatmutató formátumai

4.7 ADATTÍPUSOK

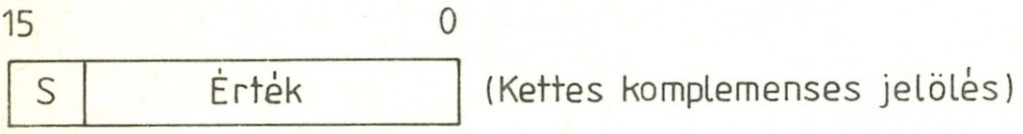
A 80287/80387-es hét külöböző adattípust támogat, amelyek a 80286/80386-os által támogatott valamennyi szabványos címzési móddal elérhetők. Mind a hét adatformátum esetén a szám előjelét mindig a legnagyobb helyértékű számjegy, vagy a mező bal szélén álló számjegy tárolja.§b A következő szimbólumok további információkat szolgáltatnak:

S	Előjelbit (0 = pozitív, 1 = negatív)
d17-d0	Decimális számjegyek byte-onként kettesével tárolva
UD	Definiálatlan, nincs jelentése
^	Implicit bináris pont helyzete
[]	A mantissza egész bitje

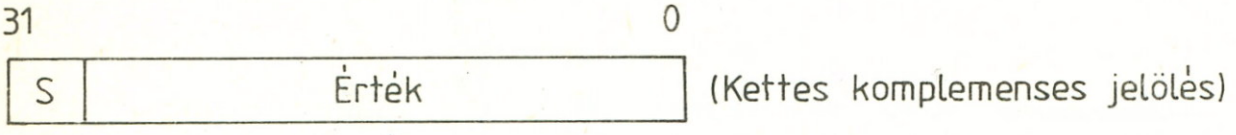
Bináris egész számok

A három bináris egész adatformátum csak mezőhosszban tér el egymástól, ami a szám tartományát határozza meg. A bal szélső bitet a szám előjeleként kell értelmezni. A negatív számok kettes komplemens formátumban kerülnek ábrázolásra. A zérusnak pozitív az előjele. A 80287/80387-es egész szó azonos a 80286/80386-os 16 bites előjeles egész adatformátummal.

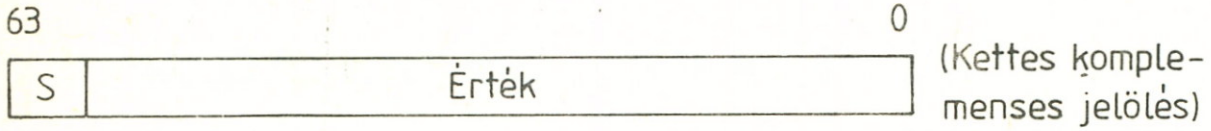
Egész szó (tartomány: $-32768 <= X <= +32767$)



Rövid egész (tartomány: $-2 \times 10^9 <= X <= +2 \times 10^9$)



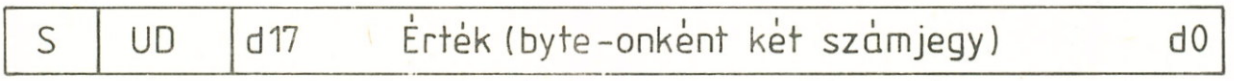
Hosszú egész (tartomány: $-9 \times 10^{18} <= X <= +9 \times 10^{18}$)



Pakolt decimális jelölés

A pakolt decimális jelölés decimális egész számok tárolását szolgálja. Minden egyes byte két decimális számjegyet tárol (pakolás). Az előjelbitből tudhatjuk meg, hogy a szám pozitív, vagy negatív. A számjegyeknek a 0H–9H tartományba kell esniük.

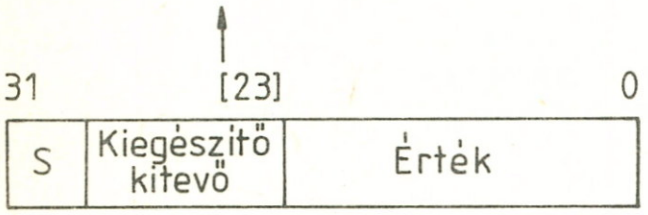
Pakolt decimális (tartomány: -99...99<=X<=+99...99 (18 digits)



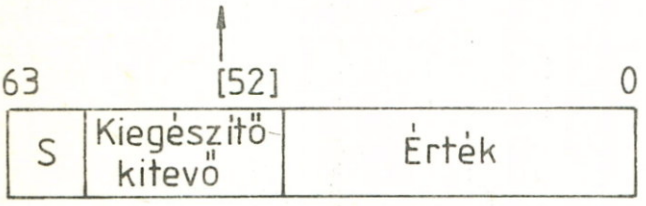
Rövid valós, hosszú valós és ideiglenes valós számformátumok

A rövid valós és hosszú valós adatformátumok csak a memóriában léteznek. Ha az ilyen módon tárolt számokat a lebegőpontos verembe töltjük, automatikusan ideiglenes valós formátummá alakulnak át.

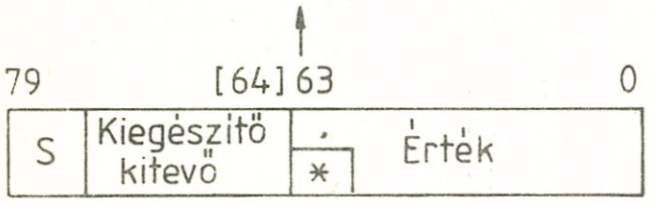
Rövid valós (tartomány: $0, 1.2 \times 10^{-38} <= X <= 3.4 \times 10^{38}$)



Hosszú valós (tartomány: $0, 2.3 \times 10^{-308} <= X <= 1.7 \times 10^{308}$)



Ideiglenes valós (tartomány: $0, 3.4 \times 10^{-4932} <= X <= 1.1 \times 10^{4932}$)



Kiegészítő kitevő (normalizált):

Rövid valós:	127 (7FH)
Hosszú valós:	1023 (3FFH)
Ideiglenes valós:	16383 (3FFFH)

Speciális értékek

A 80287/80387-es matematikai segédprocessszor néhány olyan speciális értékkel is rendelkezik, amivel a numerikus számítások rugalmassága megnövelhető.§b Ezek az értékek a következők:

- Nem-szabályos (unnormals)
- Normalizálatlan (denormal)
- Végtelen (indefinit)
- Nem számok (NAN = not a number)
- +és −végtelen ábrázolások (+and −infinity)
- Előjeles zérus (signed zero)

4.8 A 80287/80387-ES UTASÍTÁSKÉSZLETE

F2XM1(80287/80387)

A kettő az x-ediken mínusz 1 függvény kiszámítása (2 raised to the X power Minus 1)
Utasítás: F2XM1
Tipikus órajel: 500 (80287)
Az utasítás leírása: A 2 az x-ediken mínusz 1 függvény értékének kiszámítása
Az utasítás működése: Az F2XM1 utasítás a 2 az x-ediken mínusz 1 függvény értékét számítja ki. X értéke a 80287/80387 TOS (Top Of Stack = veremtető) regiszteréből származik. Az eredmény az eredeti TOS helyére kerül.
Szintaxis: F2XM1 (operandus nélkül)
Kizárásjelzők: U, P
Példa: F2XM1

FABS (80287/80387)

Abszolút érték képzése (ABSolute)
Utasítás: FABS
Tipikus órajel: 14 (80287)
Az utasítás leírása: A 80287/80387-es verem legfelső eleme abszolút értékének képzése
Az utasítás működése: Az FABS utasítás a 80287/80387-es verem legfelső elemét kicseréli annak abszolút értékével.
Szintaxis: FABS (operandus nékül)
Kizárásjelzők: I
Példa: FABS

FADD (80287/80387)

Valós számok összeadása (ADD real)
Utasítás: FADD
Tipikus órajel: 85, 105, 110 (80287)
Az utasítás leírása: A forrás- és céloperandusok összeadása
Az utasítás működése: Az FADD utasítás összeadja a forrás- és a céloperandust, és az eredményt a céloperandus helyére teszi. Ha az FADD utasítás előírt operandus nélkül kerül végrehajtásra, a 80287/80387-es verem tetején álló elem hozzáadódik a veremben második helyen álló elemhez. Az eredmény a verem tetején marad. A FADD utasítás lehetővé teszi, hogy a forrásoperandus a memóriában tárolt valós szám legyen. A céloperandus a 80287/80387-es verem legfelső eleme lesz. Az utasításnak ennél a formájánál a memória változó értéke a 80287/80387-es verem tetején álló elemhez hozzá adódik, és az eredmény a verem tetejére kerül. A FADD utasítás harmadik változatánál operandusként az összeadási művelet egyik operandusaként a 80287/80387-es verem bármely más regiszterét is választhatjuk. A második operandus továbbra is a 80287/80387-es verem tetején álló elem marad. Az utasítás végrehajtja a két elem összeadását és az eredményt a céloperandus helyére írja.
Szintaxis: FADD (operandus nélkül)
FADD *forrás_operandus*
FADD *cél_operandus,forrás_operandus*
Kizárásjelzők: I, D, U, P, O
Példa:
```
FADD
FADD    REAL_NUM
FADD    ST,ST(1)
FADD    ST(7),ST
```

FADDP (80287/80387)

Valós számok összeadása és leemelés a veremről (ADD real and Pop)
Utasítás: FADDP
Tipikus órajel: 90 (80287)
Az utasítás leírása: Operandusok összeadása, az eredmény tárolása a céloperandus helyén, és egy elem leemelése a veremről.
Az utasítás működése: A FADDP utasítás elvégzi a forrás- és céloperandusok összeadását és az összeget a céloperandus helyére írja, valamint egy elemet leemel a veremről. A forrásoperandus a 80287/80387-es verem éppen érvényes legfelső eleme lesz. Céloperandusként a 80287/80387-es verem bármelyik más regisztere szerepelhet.
Szintaxis: FADDP *cél_operandus,forrás_operandus*
Kizárásjelzők: I, O, U, P, D
Példa:
```
FADDP    ST(1),ST
```

FBLD (80287/80387)

Pakolt decimális BCD érték betöltése (packed decimal Bcd LoaD)
Utasítás: FBLD
Tipikus órajel: 300 (80287)
Az utasítás leírása: A BCD szám átalakítása ideiglenes valós számmá, és az eredmény ráhelyezése a veremre
Az utasítás működése: Az FBLD utasítás a pakolt BCD forrásoperandust átalakítja ideiglenes valós számmá, az eredményt ráhelyezi a 80287/80387-es veremre. Az FBLD utasítás feltételezi, hogy a forrásoperandus a 0–9H érvényes BCD tartományon belül van. Hibaellenőrzést nem végez.
Szintaxis: FBLD *forrás_operandus*
Kizárásjelzők: I
Példa: FBLD BCD_VALUE

FBSTP(80287/80387)

Pakolt decimális BCD érték tárolása és leemelés a veremről (packed decimal Bcd STore and Pop)
Utasítás: FBSTP
Tipikus órajel: 530 (80287)
Az utasítás leírása: A verem legfelső értékének átalakítása BCD-re, tárolása a céloperandusban és TOS leemelése a veremről.
Az utasítás működése: Az FBSTP utasítás azt az értéket, amit a 80287/80387-es verem tetején tárolt, pakolt decimális egész formátumra hozza, majd az eredményt céloperandusban tárolja és a 80287/80387-es verem TOS elemét leemeli.
Szintaxis: FBSTP *cél_operandus*
Kizárásjelzők: I
Példa: FBSTP BCD_VALUE

FCHS (80287/80387)

Előjelváltás (CHange Sign)
Utasítás: FCHS
Tipikus órajel: 15 (80287)
Az utasítás leírása: A 80287/80387-es verem TOS elemének előjelváltása
Az utasítás működése: Az FCHS utasítás megváltoztatja a 80287/80387-es verem legfelső elemének előjelét.
Szintaxis: FCHS (operandus nélkül)
Kizárásjelzők: I
Példa: FCHS

FCLEX (80287/80387)

Kizárások törlése (CLear EXceptions)
Utasítás: FCLEX
Tipikus órajel: 5 (80287)
Az utasítás leírása: A kizárásjelző bitek, az IR és a foglaltsági B jelző törlése
Az utasítás működése: A FCLEX utasítás törli valamennyi kizárás- jelzőt, a megszakítási igény jelzőbitet és a foglaltsági jelzőbitet a státusz-szóban.
Szintaxis: FCLEX (operandus nélkül)

Kizárásjelzők: Nincs
Példa: FCLEX

FCOM (80287/80387)

Valós mennyiségek összehasonlítása (COMpare)
Utasítás: FCOM
Tipikus órajel: 45, 65, 70 (80287)
Az utasítás leírása: TOS összehasonlítása a forrásoperandussal
Az utasítás működése: Az FCOM utasítás a 80287/80387-es verem éppen érvényes legfelső elemét összehasonlítja a forrásoperandussal. (Operandus előírása nélkül ST(1)-et tételezi fel.) Forrásoperandusként szerepelhet a 80287/80387-es verem regisztere vagy rövid, ill. hosszú valós memóriaoperandus. A 80287/80387-es státusz-szóban a feltételes kódok a következő feltételek alapján változnak meg:

ha ST > *forrás_operandus*
akkor C0 = 0 és C3 = 0
ha ST < *forrás_operandus*
akkor C0 = 1 és C3 = 0
ha ST = *forrás_operandus*
akkor C0 = 0 és C3 = 1
ha ST < > *forrás_operandus*
akkor C0 = 1 és C3 = 1

Szintaxis: FCOM (operandus nélkül)
FCOM *forrás_operandus*
Kizárásjelzők: I,D
Példa: FCOM
FCOM LONG_REAL_VAL
FCOM SHORT_REAL_VAL

FCOMP (80287/80387)

Valós mennyiségek összehasonlítása és leemelés a veremről (COMpare real and Pop)
Utasítás: FCOMP
Tipikus órajel: 47, 68, 72 (80287)
Az utasítás leírása: sszehasonlítja TOS-t a forrásoperandussal és leemeli TOS-t a veremről
Az utasítás működése: Az FCOMP utasítás összehasonlítja a 80287/80387-es verem éppen aktuális legfelső elemét a forrásoperandussal és leemeli a legfelső elemet. Az FCOMP utasítás forrásoperandus előírása nélkül ST(1)-et tételezi fel. A forrásoperandus rövid, vagy hosszú valós memória- operandus vagy a 80287/80387-es verem egyik regisztere lehet. A 80287/80387-es státusz-szóban a feltételes kódok a következő feltételek alapján változnak meg:

ha ST > *forrás_operandus*
akkor C0 = 0 és C3 = 0
ha ST < *forrás_operandus*
akkor C0 = 1 és C3 = 0
ha ST = *forrás_operandus*
akkor C0 = 0 és C3 = 1
ha ST < > *forrás_operandus*
akkor C0 = 1 és C3 = 1

Szintaxis: FCOMP (operandus nélkül)
FCOMP *forrás_operandus*
Kizárásjelzők: I, D
Példa: FCOMP
FCOMP LONG_REAL_VAL
FCOMP SHORT_REAL
_VAL

FCOMPP (80287/80387)

Valós mennyiségek összehasonlítása és leemelés a veremről kétszer (COMpare real and Pop twice)
Utasítás: FCOMPP
Tipikus órajel: 50 (80287)
Az utasítás leírása: sszehasonlítja TOS-t ST(1)-gyel és leemeli a verem két legfelső elemét
Az utasítás működése: Az FCOMPP utasítás a 80287/80387-es verem legfelső elemét összehasonlítja az ST(1) regiszterrel és leemeli a 80287/80387-es verem két legfelső elemét. A 80287/80387-es státusz-szóban a feltételes kódok a következő összehasonlítások szerint változnak meg:
ha ST > *forrás_operandus*
akkor C0 = 0 és C3 = 0
ha ST < *forrás_operandus*
akkor C0 = 1 és C3 = 0
ha ST = *forrás_operandus*
akkor C0 = 0 és C3 = 1
ha ST < > *forrás_operandus*
akkor C0 = 1 és C3 = 1
Szintaxis: FCOMPP (operandus nélkül)
Kizárásjelzők: I, D
Példa: FCOMPP

FCOS (80387)

Koszinusz-számítás (COSine)
Utasítás: FCOS
Tipikus órajel: N/A
Az utasítás leírása: Kiszámítja egy szög koszinuszát
Az utasítás mőködése: Az FCOS utasítás forgásszögek koszinuszát számítja ki. A szöget a verem legfelső elemének kell tartalmaznia. A művelet elvégzése után a koszinuszérték a verem TOS elemének helyére kerül.
Szintaxis: FCOS (operandus nélkül)
Kizárásjelzők: D, IS, I, P
Példa: FCOS

FDECSTP (80287/80387)

A 80287/80387-es veremmutató (ST) csökkentése (DECrement 80287/80387 STack Pointer)
Utasítás: FDECSTP
Tipikus órajel: 9 (80287)
Az utasítás leírása: A 80287/80387-es veremmutató csökkentése 1-gyel

Az utasítás működése: Az FDECSTP utasítás csökkenti 1-gyel a státusz-szóban elhelyezkedő 80287/80387-es veremmutatót (ami TOS-hez mutat).
Szintaxis: FDECSTP (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FDECSTP

FDISI (80287/80387)

Megszakítások letiltása (DISable Interrupts)
Utasítás: FDISI
Tipikus órajel: 5 (80287)
Az utasítás leírása: Megakadályozza a 80287/80387-es megszakítási igény kiadását
Az utasítás működése: Az FDISI utasítás a vezérlőszó megszakítását engedélyező maszkjának beállításával megakadályozza, hogy a 80287/80387-es megszakítási igényt adjon ki.
Szintaxis: FDISI (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FDISI

FDIV (80287/80387)

Valós mennyiségek osztása (DIVide real)
Utasítás: FDIV
Tipikus órajel: 198, 220, 225 (80287)
Az utasítás leírása: A céloperandust elosztja a forrásoperandussal és az eredményt a céloperandusban helyezi el
Az utasítás működése: Az FDIV utasítás a céloperandust elosztja a forrásoperandussal és a hányadost a céloperandusba teszi. Az FDIV utasítás forrás- vagy céloperandus előírása nélkül ST(1)-et és ST-t tételezi fel. Ebben az esetben az utasítás ST-t használja forrás- és ST(1)-et céloperandusként, végrehajt egy POP műveletet és az eredményt ST-be teszi.
Szintaxis: FDIV (operandus nélkül)
FDIV *forrás_operandus*
FDIV *cél_operandus, forrás_operandus*
Kizárásjelzők: I, D, O, U, Z, P
Példa:

```
FDIV
FDIV    ST,ST(7)
FDIV    ST(7),ST
FDIV    LONG_REAL_VAL
FDIV    SHORT_REAL_VAL
```

FDIVP (80287/80387)

Valós osztás és leemelés a veremről (DIVide real and Pop)
Utasítás: FDIVP
Tipikus órajel: 202 (80287)
Az utasítás leírása: A céloperandust elosztja a forrásoperandussal és leemeli a verem legfelső elemét
Az utasítás működése: Az FDIVP utasítás ST-t veszi forrás-, ST(n)-t pedig céloperandusnak. Leemeli a verem legfelső elemét, elvégzi az osztást és az eredményt ST(n)-ben helyezi el.
Szintaxis: FDIVP *cél_operandus, forrás_operandus*
Kizárásjelzők: I, D, O, U, Z, P
Példa:

```
FDIVP   ST(7),ST
```

FDIVR (80287/80387)

Megfordított valós osztás (DIVide real Reversed)
Utasítás: FDIVR
Tipikus órajel: 199, 221, 226 (80287)
Az utasítás leírása: A forrásoperandust elosztja a céloperandussal és az eredményt a céloperandusban helyezi el
Az utasítás működése: Az FDIVR utasítás egy megfordított valós osztást hajt végre, ui. a forrásoperandust osztja el a céloperandussal és a hányadost a céloperandusba írja. Az FDIVR utasítás, ha előírt operandusok nélkül szerepel, ST(1)-et és ST-t használja operandusként.
Szintaxis: FDIVR (operandus nélkül)
FDIVR *forrás_operandus*
FDIVR *cél_operandus, forrás_operandus*
Kizárásjelzők: I, D, O, U, Z, P
Példa:
```
FDIVR
FDIVR    ST,ST(7)
FDIVR    ST(7),ST
FDIVR    LONG_REAL_VAL
FDIVR    SHORT_REAL_VAL
```

FDIVRP (80287/80387)

Megfordított valós osztás és leemelés a veremről (DIVide real Reversed and Pop)
Utasítás: FDIVRP
Tipikus órajel: 203 (80287)
Az utasítás leírása: A forrásoperandust elosztja a céloperandussal, az eredményt a céloperandusba teszi és leemeli a verem legfelső elemét
Az utasítás működése: Az FDIVRP utasítás a forrásoperandust elosztja a céloperandussal (megfordított osztási művelet), a hányadost a céloperandusba helyezi, és leemeli a 80287/80387-es verem legfelső elemét. A forrásoperandus mindig az ST regiszter, míg a céloperandus ST(n).
Szintaxis: FDIVRP *cél_operandus, forrás_operandus*
Kizárásjelzők: I, D, O, U, P, Z
Példa:
```
FDIVRP   ST(7),ST
FDIVRP   ST(1),ST
```

FENI (80287/80387)

Megszakítások engedélyezése (ENable Interrupts)
Utasítás: FENI
Tipikus órajel: 5 (80287)
Az utasítás leírása: A megszakítási kérések engedélyezése
Az utasítás működése: A FENI utasítás a vezérlőszóban törli a megszakítási engedély maszkját, amely által lehetővé válik a megszakítási folyamatok kezdeményezése.
Szintaxis: FENI (operandus nélkül)
Kizárásjelzők: Nincs
Példa:
```
FENI
```

FFREE (80287/80387)

Regiszter felszabadítása (FREE register)
Utasítás: FFREE

Tipikus órajel: 11 (80287)
Az utasítás leírása: A célregiszter felszabadítása
Az utasítás működése: Az FFREE utasítás felszabadítja a célregiszter toldalékát (TAG) úgy, hogy a regiszter tartalmának változtatása nélkül törli azt.
Szintaxis: FFREE *cél_operandus*
Kizárásjelzők: Nincs
Példa: FFREE ST(1)

FIADD (80287/80387)

Egész összeadás (Integer ADD)
Utasítás: FIADD
Tipikus órajel: 120, 125 (80287)
Az utasítás leírása: sszeadja a forrás- és a céloperandusokat és az eredményt a céloperandusba tárolja
Az utasítás működése: Az FIADD utasítás a forrásoperandust hozzáadja a céloperandushoz és az összeget a céloperandusba teszi. A feltételezett céloperandus mindig a 80287/80387-es TOS regiszter.
Szintaxis: FIADD *forrás_operandus*
Kizárásjelzők: I, O, P, D
Példa: FIADD WORD_INT
FIADD SHORT_INT

FICOM (80287/80387)

Egész mennyiségek összehasonlítása (Integer COMpare)
Utasítás: FICOM
Tipikus órajel: 80, 85 (80287)
Az utasítás leírása: TOS-t összehasonlítja a forrásoperandussal
Az utasítás működése: A FICOM utasítás a forrásoperandust ideiglenes valós mennyiséggé alakítja át, és ezzel hasonlítja össze a 80287/80387-es TOS-t. A státusz-szó feltételes kódjai a következő feltételek alapján változnak meg:

ha ST > *forrás_operandus*
akkor C0 = 0 és C3 = 0
ha ST < *forrás_operandus*
akkor C0 = 1 és C3 = 0
ha ST = *forrás_operandus*
akkor C0 = 0 és C3 = 1
ha ST < > *forrás_operandus*
akkor C0 = 1 és C3 = 1

Szintaxis: FICOM *forrás_operandus*
Kizárásjelzők: I, D
Példa: FICOM WORD_INT
FICOM SHORT_INT

FICOMP (80287/80387)

Egész mennyiségek összehasonlítása és leemelés a veremről (Integer COMpare and Pop)
Utasítás: FICOMP
Az utasítás leírása: TOS-t összehasonlítja a forrásoperandussal és leemeli TOS-t a veremről
Az utasítás működése: Az FICOMP utasítás ugyanazt végzi el, mint az FICOM utasítás,

azzal a különbséggel, hogy a 80287/80387-es verem TOS elemét leemeli. A státusz szó feltételes kódjai a következő feltételek alapján változnak meg:

ha ST > *forrás_operandus*
akkor C0 = 0 és C3 = 0
ha ST < *forrás_operandus*
akkor C0 = 1 és C3 = 0
ha ST = *forrás_operandus*
akkor C0 = 0 és C3 = 1
ha ST < > *forrás_operandus*
akkor CO = 1 és C3 = 1

Szintaxis: FICOMP *forrás_operandus*
Kizárásjelzők: I, D
Példa: FICOMP WORD_INT
FICOMP SHORT_INT

FIDIV (80287/80387)

Egész mennyiségek osztása (Integer DIVide)
Utasítás: FIDIV
Tipikus órajel: 230, 236 (80287)
Az utasítás leírása: A céloperandust elosztja a forrásoperandussal és az eredményt a céloperandusba írja
Az utasítás működése: Az FIDIV utasítás a céloperandust elosztja a forrásoperandussal és a hányadost a céloperandusba teszi. A kijelölt céloperandus mindig a 80287/80387-es TOS regisztere.
Szintaxis: FIDIV *forrás_operandus*
Kizárásjelzők: I, D, O, U, P, Z
Példa: FIDIV WORD_INT
FIDIV SHORT_INT

FIDIVR (80287/80387)

Megfordított egész osztás (Integer DIVide Reversed)
Utasítás: FIDIVR
Tipikus órajel: 230, 237 (80287)
Az utasítás leírása: A forrásoperandust elosztja a céloperandussal és az eredményt a céloperandusba írja
Az utasítás működése: Az FIDIVR utasítás megfordított osztási műveletet hajt végre azáltal, hogy a forrásoperandust osztja az utasításba beleértett ST céloperandussal. A hányadost a 80287/80387-es verem TOS regiszterébe helyezi el.
Szintaxis: FIDIVR *forrás_operandus*
Kizárásjelzők: I, D, O, U, P, Z
Példa: FIDIVR WORD_INT
FIDIVR SHORT_INT

FILD (80287/80387)

Egész mennyiség betöltése (Integer LoaD)
Utasítás: FILD
Tipikus órajel: 50, 56, 64 (80287)

Az utasítás leírása: A forrásoperandust ideiglenes valós mennyiséggé alakítja, és az eredményt a beleértett céloperandusba, ST-be helyezi.
Az utasítás működése: Az FILD utasítás a bináris egész forrásoperandust ideiglenes valós számmá alakítja és az eredményt az utasításba beleértett ST céloperandusba helyezi el.
Szintaxis: FILD *forrás_operandus*
Kizárásjelzők: I
Példa:
```
FILD    WORD_INT
FILD    LONG_INT
FILD    SHORT_INT
```

FIMUL (80287/80387)

Egész mennyiségek szorzása (Integer MULtiply)
Utasítás: FIMUL
Tipikus órajel: 130, 136 (80287)
Az utasítás leírása: sszeszorozza a forrás- és a céloperandusokat és az eredményt a céloperandusban helyezi el
Az utasítás működése: Az FIMUL utasítás a céloperandust összeszorozza a forrásoperandussal és az eredményt az utasításba beleértett céloperandusba, ST-be helyezi el.
Szintaxis: FIMUL *forrás_operandus*
Kizárásjelzők: I, D, P, O
Példa:
```
FIMUL   WORD_INT
FIMUL   SHORT_INT
```

FINCSTP (80287/80387)

A 80287/80387-es veremmutató növelése (INCrement 80287/80387 STack Pointer)
Utasítás: FINCSTP
Tipikus órajel: 9 (80287)
Az utasítás leírása: A 80287/80387-es TOP-hoz hozzáad 1-et
Az utasítás működése: Az utasítás a 80287/80387-es verem státusz-szavában a veremtető mutatót 1-gyel növeli.
Szintaxis: FINCSTP (operandus nélkül)
Kizárásjelzők: Nincs
Példa:
```
FINCSTP
```

FINIT (80287/80387)

A processzor inicializálása (INITialize processor)
Utasítás: FINIT
Tipikus órajel: 5 (80287)
Az utasítás leírása: A 80287/80387-es processzor funkcionális alaphelyzetbe állítását hajtja végre
Az utasítás működése: Az FINIT utasítás a vezérlőszót a 03FFH-ra állítja be, törli a kizárásjelzőket, a foglaltsági megszakításokat, valamint felszabadítja valamennyi 80287/80387-es lebegőpontos veremelemet. Ez az utasítás a hardver alaphelyzetbe állítás (reset) funkcionális megfelelője azzal a különbséggel, hogy az utasítás a 80287/80387-es utasításfelhozatal szinkronizációjára nincs hatással.
Szintaxis: FINIT (operandus nélkül)
Kizárásjelzők: Nincs
Példa:
```
FINIT
```

FIST (80287/80387)

Egész mennyiség tárolása (Integer STore)
Utasítás: FIST
Tipikus órajel: 86, 88 (80287)
Az utasítás leírása: Kerekíti TOS-t, és az eredményt a céloperandusba helyezi el
Az utasítás működése: Az FIST utasítás a 80287/80387-es TOS tartalmát a vezérlőszó RC mezőjének megfelelően kerekíti és az eredményt a céloperandusba helyezi el.
Szintaxis: FIST *cél_operandus*
Kizárásjelzők: I, P
Példa:
```
FIST    WORD_INT
FIST    SHORT_INT
```

FISTP (80287/80387)

Egész mennyiség tárolása és leemelése a veremről (Integer STore and Pop)
Utasítás: FISTP
Tipikus órajel: 88, 90, 100 (80287)
Az utasítás leírása: Kerekíti TOS-t, az eredményt a céloperandusba helyezi és leemeli a verem legfelső elemét
Az utasítás működése: Az FISTP utasítás TOS tartalmát úgy kerekíti, mint ahogy az FIST utasításnál láttuk, az eredményt a céloperandusba helyezi és leemeli a verem legfelső elemét.
Szintaxis: FISTP *cél_operandus*
Kizárásjelzők: I, P
Példa:
```
FISTP   WORD_INT
FISTP   LONG_INT
FISTP   SHORT_INT
```

FISUB (80287/80387)

Egész mennyiség kivonása (Integer SUBTract)
Utasítás: FISUB
Tipikus órajel: 120, 125 (80287)
Az utasítás leírása: A céloperandusból kivonja a forrásoperandust és az eredményt a céloperandusba helyezi
Az utasítás működése: Az FISUB utasítás a forrásoperandust kivonja az utasításba beleértett céloperandusból, ST-ből és a különbséget 80287/80387-es TOS regiszterbe teszi.
Szintaxis: FISUB *forrás_operandus*
Kizárásjelzők: I, D, P, O
Példa:
```
FISUB   WORD_INT
FISUB   SHORT_INT
```

FISUBR (80287/80387)

Megfordított egész kivonás (Integer SUBtraction Reversed)
Utasítás: FISUBR
Tipikus órajel: 120, 125 (80287)
Az utasítás leírása: A céloperandust vonja ki a forrásoperan- dusból és az eredményt a céloperandusba helyezi
Az utasítás működése: Az FISUBR utasítás megfordított kivonási műveletet valósít meg

azáltal, hogy a forrásoperandusból vonja ki az utasításba beleértett céloperandust, ST-t. Az eredményt a 80287/80387-es TOS-ra helyezi vissza.
Szintaxis: FISUBR *forrás_operandus*
Kizárásjelzők: I, D, P, O
Példa: FISUBR WORD_INT
FISUBR SHORT_INT

FLD (80287/80387)

Valós mennyiség betöltése (LoaD real)
Utasítás: FLD
Tipikus órajel: 20, 43, 46, 57 (80287)
Az utasítás leírása: A forrásoperandust TOS-ra helyezi
Az utasítás működése: Az FLD utasítás a 80287/80387-es TOS-ra helyezi a forrásoperandust úgy, hogy a 80287/80387-es veremmutatót 1-gyel csökkenti, és a forrásoperandus tartalmát bemásolja TOS-ba.
Szintaxis: FLD *forrás_operandus*
Kizárásjelzők: I, D
Példa: FLD ST(7)
FLD LONG_REAL
FLD SHORT_REAL

FLD1 (80287/80387)

+1.0 ráhelyezése a 80287/80387-es TOS-ra (LoaD +1.0 onto 80287/80387 TOS)
Utasítás: FLD1
Tipikus órajel: 18 (80287)
Az utasítás leírása: +1.0 ráhelyezése a 80287/80387-es TOS-ra
Az utasítás működése: Az FLD1 utasítás az ideiglenes valós 64 bites pontosságú és 19 decimális helyet elfoglaló +1.0 konstans értéket ráhelyezi a 80287/80387-es veremre.
Szintaxis: FLD1 (operandus nélkül)
Kizárásjelzők: I
Példa: FLD1

FLDCW (80287/80387)

A vezérlőszó betöltése (LoaD Control Word)
Utasítás: FLDCW
Tipikus órajel: 10 (80287)
Az utasítás leírása: A vezérlőszó felcserélése a forrásoperandussal
Az utasítás működése: Az FLDCW utasítás a mikroprocesszor vezérlőszavának éppen érvényes tartalmát felcseréli a forrásoperandus által definiált szóval.
Szintaxis: FLDCW *forrás_operandus*
Kizárásjelzők: Nincs
Példa: FLDCW MEM_WORD

FLDENV (80287/80387)

A környezet betöltése (LoaD ENVironment)
Utasítás: FLDENV
Tipikus órajel: 40 (80287)

Az utasítás leírása: Betölti a környezetbe a forrásoperandust
Az utasítás működése: Az FLDENV utasítás betölti a 80287/80387-es környezetbe a forrás-operandus által definiált memóriahely tartalmát.
Szintaxis: FLDENV *forrás_operandus*
Kizárásjelzők: Nincs
Példa: FLDENV FOURTEEN_BYTES

FLDL2E (80287/80387)

Log e betöltése (LoaD Log_2e)
Utasítás: FLDL2E
Tipikus órajel: 18 (80287)
Az utasítás leírása: A log_2e érték ráhelyezése a 80287/80387-es veremre
Az utasítás működése: Az FLDL2E utasítás az ideiglenes valós 64 bites és 19 decimális helyet elfoglaló log_2e konstans értéket rátölti a 80287/80387-es veremre.
Szintaxis: FLDL2E (operandus nélkül)
Kizárásjelzők: I
Példa: FLDL2E

FLDL2T (80287/80387)

A log 10 érték betöltése (LoaD Log_210)
Utasítás: FLDL2T
Tipikus órajel: 19 (80287)
Az utasítás leírása: A log_210 érték ráhelyezése a 80287/80387-es veremre
Az utasítás működése: Az FLDL2T utasítás az ideiglenes valós 64 bites és 19 decimális hely pontosságú log_210 konstans értéket ráhelyezi a 80287/80387-es veremre.
Szintaxis: FLDL2T (operandus nélkül)
Kizárásjelzők: I
Példa: FLDL2T

FLDLG2 (80287/80387)

A log 2 érték betöltése (LoaD $LoG_{10}2$)
Utasítás: FLDLG2
Tipikus órajel: 21 (80287)
Az utasítás leírása: A $log_{10}2$ érték ráhelyezése a 80287/80387-es veremre
Az utasítás működése: Az FLDLG2 utasítás az ideiglenes valós 64 bites és 19 decimális hely pontosságú $log_{10}2$ konstans értéket ráhelyezi a lebegőpontos veremre.
Szintaxis: FLDLG2 (operandus nélkül)
Kizárásjelzők: I
Példa: FLDLG2

FLDLN2 (80287/80387)

A log 2 érték betöltése (LoaD Log_e2)
Utasítás: FLDLN2
Tipikus órajel: 20 (80287)
Az utasítás leírása: A log_e2 érték ráhelyezése a 80287/80387-es veremre
Az utasítás működése: Az FLDLN2 utasítás az ideiglenes valós 64 bites és 19 decimális hely pontosságú log_e2 konstans értéket ráhelyezi a lebegőpontos veremre.

Szintaxis: FLDLN2 (operandus nélkül)
Kizárásjelzők: I
Példa: FLDLN2

FLDPI (80287/80387)

PI betöltése (LoaD PI)
Utasítás: FLDPI
Tipikus órajel: 19 (80287)
Az utasítás leírása: PI érték ráhelyezése a 80287/80387-es veremre
Az utasítás működése: Az FLDPI utasítás az ideiglenes valós 64 bites és 19 decimális hely pontosságú PI konstans értéket ráhelyezi a lebegőpontos veremre.
Szintaxis: FLDPI (operandus nélkül)
Kizárásjelzők: I
Példa: FLDPI

FLDZ (80287/80387)

Zérus betöltése (LoaD Zero)
Utasítás: FLDZ
Tipikus órajel: 14 (80287)
Az utasítás leírása: A +0.0 érték ráhelyezése a 80287/80387-es veremre
Az utasítás működése: Az FLDZ utasítás az ideiglenes valós 64 bites és 19 decimális hely pontosságú +0.0 konstans értéket ráhelyezi a lebegőpontos veremre.
Szintaxis: FLDZ (operandus nélkül)
Kizárásjelzők: I
Példa: FLDZ

FMUL (80287/80387)

Valós mennyiségek szorzása (MULtiply real)
Utasítás: FMUL
Tipikus órajel: 97, 118, 120, 138, 161 (80287)
Az utasítás leírása: A céloperandus összeszorzása a forrásoperandussal és az eredmény elhelyezése a céloperandusba
Az utasítás működése: Az FMUL utasítás a céloperandust összeszorozza a forrásoperandussal és az eredményt a céloperandusba helyezi el. Ha az FMUL-nál nem írunk elő operandusokat, akkor az ST(1) és ST regisztereket használja fel az utasítás.
Szintaxis: FMUL (operandus nélkül)
FMUL *forrás_operandus*
FMUL *cél_operandus,forrás_operandus*
Kizárásjelzők: I, D, U, P, O
Példa:

```
FMUL
FMUL    LONG_REAL
FMUL    SHORT_REAL
FMUL    ST,ST(7)
FMUL    ST(7),ST
```

FMULP (80287/80387)

Valós mennyiségek szorzása és leemelés a veremről (MULtiply real and Pop)
Utasítás: FMULP
Tipikus órajel: 100, 142 (80287)
Az utasítás leírása: sszeszorozza a forrás- és céloperandust, az eredményt tárolja és leemeli a verem legfelső elemét
Az utasítás működése: Az FMULP utasítás kiveszi a forrásoperandust a 80287/80387-es TOS regiszterből. A céloperandus az ST(n) regiszterből származik. Ezek után következik a verem legfelső elemének leemelése és a szorzás eredményének tárolása az ST(n) regiszterben.
Szintaxis: FMULP *cél_operandus, forrás_operandus*
Kizárásjelzők: I, D, U, O, P
Példa: FMULP ST(7), ST

FNCLEX (80287/80387)

Kizárások törlése (CLear EXeptions)
Utasítás: FNCLEX
Tipikus órajel: 5 (80287)
Az utasítás leírása: Törli a kizárási, a megszakítási igény és a foglaltsági jelzőket.
Az utasítás működése: Az FNCLEX utasítás törli valamennyi kizárás jelzőt, a B foglaltsági jelzőt és az IR megszakítási igény jelzőt a státusz-szóban.
Szintaxis: FNCLEX (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FNCLEX

FNDISI (80287/80387)

Megszakítások használatának letiltása (DISable Interrupts)
Utasítás: FNDISI
Tipikus órajel: N/A
Az utasítás leírása: A megszakítást engedélyező maszk beállítása a vezérlőszóban
Az utasítás működése: Az FNDISI utasítás a vezérlőszóban beállítja a megszakítást engedélyező maszkot, emelynek következtében a mikroprocesszor nem tud kiadni megszakítási igényeket.
Szintaxis: FNDISI (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FNDISI

FNENI (80287/80387)

Megszakítások használatának engedélyezése (ENable Interrupts)
Utasítás: FNENI
Tipikus órajel: N/A
Az utasítás leírása: Az utasítás törli a megszakítást engedélyező maszkot
Az utasítás működése: Az FNENI utasítás a vezérlőszóban törli a megszakítást engedélyező maszkot, és ezáltal lehetővé válik, hogy a mikroprocesszor megszakítási igényeket generáljon.
Szintaxis: FNENI (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FNENI

FNINIT (80287/80387)

A processzor inicializálása (INITialize processor)
Utasítás: FNINIT
Tipikus órajel: 5 (80287)
Az utasítás leírása: Alaphelyzetbe állítja (RESET) a mikroprocesszort
Az utasítás működése: Az FNINIT utasítás a vezérlőszót a 03FFH-ra állítja be, törli a kizárásjelzőket, a foglaltsági megszakításokat és felszabadítja valamennyi 80287/80387-es lebegőpontos veremelemet. Ez az utasítás a hardver reset funkcionális megfelelője, azzal a különbséggel, hogy a 80287/80387-es utasításfelhozatal szinkronizációjára nincs hatással.
Szintaxis: FNINIT (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FNINIT

FNOP (80287/80387)

Nincs művelet (No Operation)
Utasítás: FNOP
Tipikus órajel: 13 (80287)
Az utasítás leírása: Nem hajt végre semmilyen műveletet
Az utasítás működése: Az FNOP utasítás semmilyen műveletet nem hajt végre.
Szintaxis: FNOP (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FNOP

FNSAVE (80287/80387)

Állapot elmentése (SAVE state)
Utasítás: FNSAVE
Tipikus órajel: N/A
Az utasítás leírása: A teljes állapotot eltárolja a céloperandusként megjelölt memóriahelyre.
Az utasítás működése: Az FNSAVE utasítás tárolja a teljes 80287/80387-es környezetet, beleértve a veremregisztereket is, arra a helyre, amit a céloperandussal előírunk, majd alaphelyzetbe állítja a 80287/80387-es processzort.
Szintaxis: FNSAVE *cél_operandus*
Kizárásjelzők: Nincs
Példa: FNSAVE MEM_LOC

FNSTCW (80287/80387)

A vezérlőszó tárolása (STore Control Word)
Utasítás: FNSTCW
Tipikus órajel: 15 (80287)
Az utasítás leírása: A vezérlőszót a céloperandusba helyezi
Az utasítás működése: Az FNSTCW utasítás a 80287/80387-es vezérlő- szó éppen érvényes állapotát elhelyezi arra a helyre, amit a céloperandussal előírunk.
Szintaxis: FNSTCW (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FNSTCW MEM_LOC

FNSTENV (80287/80387)

A környezet tárolása (STore ENVironment)
Utasítás: FNSTENV
Tipikus órajel: 45 (80287)
Az utasítás leírása: A környezetet tárolja a céloperandusba
Az utasítás működése: Az FNSTENV utasítás az éppen érvényes 80287/80387-es környezetet – kizárásmutatók, toldalékok, vezérlőszó, státusz-szó – tárolja arra a memóriahelyre, amelyet a céloperandussal előírunk.
Szintaxis: FNSTENV *cél_operandus*
Kizárásjelzők: Nincs
Példa: FNSTENV MEM_LOC

FNSTSW (80287/80387)

A státusz-szó tárolása (STore Status Word)
Utasítás: FNSTSW
Tipikus órajel: 15 (80287)
Az utasítás leírása: A státusz-szót tárolja a céloperandusba
Az utasítás működése: Az FNSTSW utasítás a 80287/80387-es státusz-szó éppen érvényes állapotát tárolja arra a memóriahelyre, amit a céloperandussal előírunk.
Szintaxis: FNSTSW *cél_operandus*
Kizárásjelzők: Nincs
Példa: FNSTSW MEM_LOC

FPATAN (80287/80387)

Parciális árkusz tangens képzés (Partial Arc Tangens)
Utasítás: FPATAN
Tipikus órajel: 650 (80287)
Az utasítás leírása: Az ARCTAN függvényérték kiszámítása
Az utasítás működése: Az FPATAN utasítás X értékét leemeli az éppen aktuális TOS-ról, majd leemeli a verem következő elemét is, és az aktuális TOS-t Y-nak tekinti. Ezt követően kiszámítja ARCTAN (Y/X) szöget és ráhelyezi a 80287/80387-es TOS regiszterre. X-nek és Y-nak a megfelelő értelmezési tartományba kell esnie, vagyis:
$0 < Y < X < \infty$
Szintaxis: FPATAN (operandus nélkül)
Kizárásjelzők: U, P
Példa: FPATAN

FPREM (80287/80387)

Parciális maradékképzés (Partial REMainder)
Utasítás: FPREM
Tipikus órajel: 125 (80287)
Az utasítás leírása: ST-t osztja (modulo) ST(1)-gyel és a maradékot ST-ben helyezi el
Az utasítás működése: Az FPREM utasítás a 80287/80387-es TOS regiszter éppen aktuális tartalmát elosztja a következő elemmel, ST(1)-gyel és a maradékot TOS-ba helyezi el.
Szintaxis: FPREM (operandus nélkül)
Kizárásjelzők: I, U, D
Példa: FPREM

FPTAN (80287/80387)

Parciális tangesképzés (Partial TANgent)
Utasítás: FPTAN
Tipikus órajel: 450 (80287)
Az utasítás leírása: Szögek tangensének kiszámítása
Az utasítás működése: Az FPTAN utasítás kiszámítja egy szög (θ) tangensét (Y/X = TAN (θ)). A szögnek, amelynek a verem legfelső elemének kell lennie, a $0 <= \theta < \pi/4$ szögnegyedbe kell esnie. Az utasítás befejezésekor Y felváltja θ-t a veremben és X lesz az új verem tetején. A megadott tartományon belüli szögeket nem értelmezi az utasítás. 80387-es megjegyzés: forgásszög lehet. Ez a processzor FCOS és FSIN utasításokkal is rendelkezik.
Szintaxis: FPTAN (operandus nélkül)
Kizárásjelzők: I, P
Példa: FPTAN

FRNDINT (80287/80387)

Kerekítés egészre (RouND INTeger)
Utasítás: FRNDINT
Tipikus órajel: 45 (80287)
Az utasítás leírása: A 80287/80387-es verem TOS elemét egészre kerekíti
Az utasítás működése: Az FRNDINT utasítás a 80287/80387-es TOS regiszter tartalmát egészre kerekíti. A vezérlőszó RC mezőjétől függően négy kerekítési módot különböztetünk meg.

RC = 00	Kerekítés a legközelebbi egészre, ha az érték pontosan a tartomány fele, a páros felé történik a kerekítés
RC = 01	Lefelé kerekítés
RC = 10	Felfelé kerekítés
RC = 11	Kerekítés zérus irányába

Szintaxis: FRNDINT (operandus nélkül)
Kizárásjelzők: I, P
Példa: FRNDINT

FRSTOR (80287/80387)

A 80287/80387-es állapotának visszaállítása (ReSTORe state)
Utasítás: FRSTOR
Tipikus órajel: 205 (80287)
Az utasítás leírása: A 80287/80387-es állapotának visszaállítása
Az utasítás működése: Az FRSTOR utasítás a forrásoperandussal megadott 94 byte-os memóriaterületről visszaállítja a 80287/80387-es állapotát. A kompatibilitás miatt az ehhez szükséges adatokat az FSAVE vagy FNSAVE utasítások valamelyikével kell tárolni.
Szintaxis: FRSTOR *forrás_operandus*
Kizárásjelzők: Nincs
Példa: FRSTOR MEM_LOC

FSAVE (80287/80387)

A 80287/80387-es állapotának tárolása (SAVE state)
Utasítás: FSAVE
Tipikus órajel: 205 (80287)
Az utasítás leírása: A 80287/80387-es állapotának tárolása
Az utasítás működése: Az FSAVE utasítás a 80287/80387-es processzor teljes állapotát tárolja, beleértve a veremregisztereket és a környezetet. Az információ arra a helyre kerül, amit céloperandussal megadunk. Ezt követően az utasítás elvégzi a 80287/80387-es inicializálását.
Szintaxis: FSAVE *cél_operandus*
Kizárásjelzők: Nincs
Példa: FSAVE MEM_LOC

FSCALE (80287/80387)

Aránybeállítás (SCALE)
Utasítás: FSCALE
Tipikus órajel: 35 (80287)
Az utasítás leírása: Szorzás vagy osztás kettő hatványozott értékével
Az utasítás működése: Az FSCALE utasítás az ST(1)-ben található értéket egészként értelmezi, és hozzáadja az ST-ben található szám kitevőjének értékéhez. Az eredmény tehát $ST*2_{ST(1)}$ lesz.
Szintaxis: FSCALE (operandus nélkül)
Kizárásjelzők: I, U, O
Példa: FSCALE

FSETPM (80287/80387)

Védett mód beállítása (SET Protected Mode)
Utasítás: FSETPM
Tipikus órajel: 5 (80287)
Az utasítás leírása: A védett mód beállítása
Az utasítás működése: Az FSETPM utasítás a 80287/80387-est védett módba helyezi. Ha az FSETPM-t végrehajtottuk, a processzor a következő hardver törlésig (reset) védett módban marad, még az FRSTOR, FSAVE vagy FINIT utasítások kiadása után is.
Szintaxis: FSETPM (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FSETPM

FSIN (80387)

Szinusz-számítás (SINe)
Utasítás: FSIN
Tipikus órajel: N/A
Az utasítás leírása: Kiszámítja egy szög szinuszát
Az utasítás működése: Az FSIN utasítás kiszámítja egy szög szinuszát. A szöget, ami forgásszög lehet, a verem tetejéről veszi. A végrehajtás során a szinuszérték a szögértéket felváltja a verem tetején.
Szintaxis: FISN (operandus nélkül)
Kizárásjelzők: D, IS, I, P
Példa: FSIN

FSINCOS (80387)

Szinusz-koszinusz-számítás (SINe/COSine)
Utasítás: FSINCOS
Tipikus órajel: N/A
Az utasítás leírása: Egyidejű szinusz- és koszinusz-számítás.
Az utasítás működése: Az FSINCOS utasítás kiszámítja egy szög szinuszát és koszinuszát. A szöget, ami forgásszög lehet, TOS-ből veszi. Az utasítás végrehajtása után a koszinuszérték ST-be, a szinuszérték ST(1)-be kerül.
Szintaxis: FSINCOS (operandus nélkül)
Kizárásjelzők: D, IS, I, P
Példa: FSINCOS

FSQRT (80287/80387)

Négyzetgyökvonás (SQuare RooT)
Utasítás: FSQRT
Tipikus órajel: 183 (80287)
Az utasítás leírása: ST négyzetgyökének számítása
Az utasítás működése: Az FSQRT utasítás a 80287/80387-es TOS regiszter éppen aktuális tartalmának a négyzetgyökét képezi, majd ezt az értéket ráhelyezi a veremre.
Szintaxis: FSQRT (operandus nélkül)
Kizárásjelzők: I, P, D
Példa: FSQRT

FST (80287/80387)

Valós mennyiségek tárolása (STore real)
Utasítás: FST
Tipikus órajel: 18, 87, 100 (80287)
Az utasítás leírása: ST-t a céloperandus által előírt helyre tárolja
Az utasítás működése: Az FST utasítás a 80287/80387-es TOS regiszter értékét arra a memóriahelyre tárolja, amit a céloperandussal előírtunk.
Szintaxis: FST *cél_operandus*
Kizárásjelzők: I, U, O, P
Példa:
```
FST     ST(7)
FST     LONG_REAL_VAL
FST     SHORT_REAL_VAL
```

FSTCW (80287/80387)

A vezérlőszó tárolása (STore Control Word)
Utasítás: FSTCW
Tipikus órajel: 15 (80287)
Az utasítás leírása: A vezérlőszó tárolása a céloperandus által megjelölt helyre
Az utasítás működése: Az FSTCW utasítás a 80287/80387-es kontroll- szó éppen érvényes állapotát tárolja arra a memóriahelyre, amit a céloperandussal előírunk.
Szintaxis: FSTCW *cél_operandus*
Kizárásjelzők: Nincs
Példa: FSTCW MEM_LOC

FSTENV (80287/80387)

A környezet tárolása (STore ENVironment)
Utasítás: FSTENV
Tipikus órajel: 45 (80287)
Az utasítás leírása: A 80287/80387-es státuszt tárolja a céloperandus által megjelölt helyre
Az utasítás működése: Az FSTENV utasítás tárolja a környezetet – beleértve a státusz-, vezérlő- és toldalékszavakat és a kizárás- mutatókat – arra a memóriahelyre, amelyet a céloperandussal előírunk.
Szintaxis: FSTENV *cél_operandus*
Kizárásjelzők: Nincs
Példa: FSTENV MEM_LOC

FSTP (80287/80387)

Valós mennyiség tárolása és leemelés a veremről (STore real and Pop)
Utasítás: FSTP
Tipikus órajel: 20, 55, 102 (80289)
Az utasítás leírása: ST tárolása a memóriába és leemelés a veremről
Az utasítás működése: Az FSTP utasítás a 80287/80387-es TOS regisztert a memóriának arra a helyére tárolja, amit a céloperandussal előírunk, és leemeli a verem legfelső elemét.
Szintaxis: FSTP *cél_operandus*
Kizárásjelzők: I, U, O, P
Példa:
```
FSTP    ST(n)
FSTP    LONG_REAL_VAL
FSTP    SHORT_REAL_VAL
```

FSTSW (80287/80387)

A státusz-szó tárolása (STore Status Word)
Utasítás: FSTSW
Tipikus órajel: 15 (80287)
Az utasítás leírása: A státusz-szó tárolása a memóriába
Az utasítás működése: Az FSTSW utasítás a 80287/80387-es státusz-szó éppen érvényes értékét a céloperandus által előírt memóriahelyre tárolja
Szintaxis: FSTSW *cél_operandus*
Kizárásjelzők: Nincs
Példa: FSTSW

FSUB (80287/80387)

Valós mennyiségek kivonása (SUBtract real)
Utasítás: FSUB
Tipikus órajel: 85, 105, 110 (80287)
Az utasítás leírása: A forrásoperanust kivonja a céloperandusból és a különbséget a céloperandusba teszi
Az utasítás működése: Az FSUB utasítás a forrásoperandust kivonja a céloperandusból és az eredményt a céloperandusba írja. Az FSUB operandusok nélküli használata esetén az ST(1) és ST regisztereket használja a rendszer. Ha csak a forrásoperandust írjuk elő, akkor ST lesz a céloperandus.

Szintaxis: FSUB (operandus nélkül)
FSUB *forrás_operandus*
FSUB *cél_operandus, forrás_operandus*
Kizárásjelzők: I, D, P, O, U
Példa:
```
FSUB
FSUB    LONG_REAL_VAL
FSUB    SHORT_REAL_VAL
FSUB    ST,ST(7)
FSUB    ST(7),ST
```

FSUBP (80287/80387)

Valós mennyiségek kivonása és leemelés a veremről (SUBtract real and Pop)
Utasítás: FSUBP
Tipikus órajel: 90 (80287)
Az utasítás leírása: Kivonja a forrásoperandust a céloperandusból, a különbséget a céloperandusban helyezi el és leemeli a verem legfelső elemét
Az utasítás működése: Az FSUBP utasítás a forrásoperandust kivonja a céloperandusból és a különbséget a céloperandusba helyezi el, majd leemeli a lebegőpontos verem legfelső elemét.
Szintaxis: FSUBP *cél_operanus, forrás_operandus*
Kizárásjelzők: I, D, U, O, P
Példa:
```
FSUBP    ST(7),ST
```

FSUBR (80287/80387)

Megfordított valós kivonás (SUBtract real Reversed)
Utasítás: FSUBR
Tipikus órajel: 87,105,110 (80287)
Az utasítás leírása: A céloperandust vonja ki a forrásoperandusból és az eredményt a céloperandusba helyezi
Az utasítás működése: Az FSUBR utasítás megfordított kivonási műveletet valósít meg azzal, hogy a forrásoperandusból vonja ki a céloperandust. A különbséget a céloperandusba helyezi. Az FSUBR operandus nélküli használata esetén ST(1) és ST lesznek az operandusok. Ha csak a forrásoperandust írjuk elő, ST-t értelmezi az utasítás céloperandusként.
Szintaxis: FSUBR (operandus nélkül)
FSUBR *forrás_operandus*
FSUBR *cél_operandus, forrás_operandus*
Kizárásjelzők: I, O, D, U, P
Példa:
```
FSUBR
FSUBR    LONG_REAL_VAL
FSUBR    SHORT_REAL_VAL
FSUBR    ST,ST(7)
FSUBR    ST(7),ST
```

FSUBRP (80287/80387)

Megfordított valós kivonás és leemelés a veremről (SUBtract real Revesed and Pop)
Utasítás: FSUBRP
Tipikus órajel: 90 (80287)
Az utasítás leírása: A céloperandust kivonja a forrásoperandusból, az eredményt tárolja és leemeli a verem legfelső elemét

Az utasítás működése: Az FSUBRP utasítás megfordított kivonási műveletet valósít meg azzal, hogy a céloperandust (ST(n)) kivonja a forrásoperandusból (ST). Ezt követően leemeli a verem legfelső elemét, majd az eredményt visszateszi a céloperandusba, ST(n)-be.
Szintaxis: FSUBRP *cél_operandus, forrás_operandus*
Kizárásjelzők: I, O, U, D, P
Példa: FSUBRP

FTST (80287/80387)

Vizsgálat (TeST)
Utasítás: FTST
Tipikus órajel: 42 (80387)
Az utasítás leírása: A 80287/80387-es TOS-t összehasonlítja +0.0-val és beállítja a feltételes kódokat
Az utasítás működése: Az FTST utasítás a 80287/80387-es TOS regiszter, TS tartalmát összehasonlítja +0.0-val. Az eredmény a státusz-szóban szereplő feltételes kódokat a következők szerint befolyásolja:

Ha ST pozitív és nem zérus	C0 = 0	C3 = 0
Ha ST negatív és nem zérus	C0 = 1	C3 = 0
Ha ST zérus	C0 = 0	C3 = 1
Ha ST nem hasonlítható össze (NAN)	C0 = 1	C3 = 1

Szintaxis: FTST (operandus nélkül)
Kizárásjelzők: I, D
Példa: FTST

FWAIT (80287/80387)

Várakozás (WAIT) (CPU utasítás)
Utasítás: FWAIT
Tipikus órajel: 3+5n (80287)
Az utasítás leírása: A 80286-os processzor az utasítás hatására megvárja, amíg a 80287/80387-es processzor befejezi a rábízott feladatot
Az utasítás működése: A FWAIT utasítás hatására a 80286/80386-os mindaddig várakozik, amíg a 80287/80387-es befejezi azt az utasítást, amin éppen dolgozik. Ezután a 80286-os végrehajtja a következő utasítást. Az n a legutolsó utasítás elvégzése előtti foglaltsági (BUSY) vizsgálatok számát jelenti.
Szintaxis: FWAIT (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FWAIT

FXAM (80287/80387)

Vizsgálat (eXAMine)
Utasítás: FXAM
Tipikus órajel: 17 (80287)
Az utasítás leírása: A vizsgálat alapján beállítja a státusz-szó feltételes kódjait
Az utasítás működése: Az FXAM utasítás megvizsgálja a 80287/80387-es TOS regiszter tartalmát és a következők szerint állítja be a státusz-szó feltételes kódjait:

C0	C1	C2	C3	EREDMÉNY
0	0	0	0	+ Nem szabályos
0	0	0	1	+ NAN
0	0	1	0	− Nem szabályos
0	0	1	1	− NAN
0	1	0	0	+ Normalizált
0	1	0	1	+ 00
0	1	1	0	− Normalizált
0	1	1	1	− 00
1	0	0	0	+ 0
1	0	0	1	Üres
1	0	1	0	− 0
1	0	1	1	Üres
1	1	0	0	+ Normalizálás nélküli
1	1	0	1	Üres
1	1	1	0	− Normalizálás nélküli
1	1	1	1	Üres

Szintaxis: FXAM (operandus nélkül)
Kizárásjelzők: Nincs
Példa: FXAM

FXCH (80287/80387)

Regiszterek megcserélése (eXCHange registers)
Utasítás: FXCH
Tipikus órajel: 12 (80287)
Az utasítás leírása: A céloperandust TOS-sal cseréli fel
Az utasítás működése: Az FXCH utasítás a céloperandust felcseréli a 80287/80387-es TOS regiszterrel. Ha nem írunk elő operandust, akkor az utasítás ST(1)-et értelmezi céloperandusként.
Szintaxis: FXCH (operandus nélkül)
FXCH *cél_operandus*
Kizárásjelzők: I
Példa: FXCH
FXCH ST(7)

FXTRACT (80287/80387)

Kitevő és mantissza kiemelése (eXTRACT exponent and significand)
Utasítás: FXTRACT
Tipikus órajel: 50 (80287)
Az utasítás leírása: Kiemeli a kitevőt és a mantisszát
Az utasítás működése: Az FXTRACT utasítás a 80287/80387 TOS regiszterében található számot kitevőre és mantisszára bontja és valós számként fejezi ki. A kitevő az eredeti TOS tartalmat kicseréli, majd a mantissza rákerül a 80287/80387-es TOS regiszterre.
Szintaxis: FXTRACT
Kizárásjelzők: I
Példa: FXTARCT

FYL2X (80287/80387)

$Y * \log_2 X$
Utasítás: FYL2X
Tipikus órajel: 950 (80287)
Az utasítás leírása: Az $Y * \log_2 X$ érték kiszámítása
Az utasítás működése: Az FYL2X utasítás X értékét a 80287/80387-es TOS regiszterből, Y értékét pedig ST(1)-ből veszi ki. Ezt követően leemeli a verem legfelső elemét, és a kiszámított eredményt TOS-ba helyezi el.
Szintaxis: FYL2X (operandus nélkül)
Kizárásjelzők: P
Példa: FYL2X

FYL2XP1 (80287/80387) $Y * \log_2 (X+1)$

Utasítás: FYL2XP1
Tipikus órajel: 850 (80287)
Az utasítás leírása: $Y * \log_2 (X+1)$ értékének kiszámítása
Az utasítás működése: Az FYL2XP1 utasítás X értékét a 80287/80387-es TOS-ből, Y-t pedig ST(1)-ből veszi, leemeli a verem legfelső elemét és a számítás eredményét TOS-ba helyezi el. Ezt az utasítást akkor kell használni, amikor 1-hez közeli szám logaritmusát akarjuk kiszámítani.
Szintaxis: FYL2XP1 (operandus nélkül)
Kizárásjelzők: P
Példa: FYL2XP1

5. EGYSZERŰ PROGRAMOZÁSI TECHNIKÁK

Az assembly nyelv alapjait, valamint a 80286/80386-os és 80287/80387-es mikroprocesszorok utasításkészletét a könyv első négy fejezetéből megismerhettük. A 3. és 4. fejezetben egyszerű példákon keresztül muttattuk be az egyes utasítások formáját és szintaxisát. Ezen programok egyike sem volt futtatható.

Az 5. fejezet segítségével a programok szintaxisát és az egyszerűbb programok megírásának módját sajátíthatjuk el. Az általános programozás, az önálló utasítások vagy speciális programozási technikák szempontjából minden egyes példa valami újat nyújt. Mindenkinek, még a haladó assembly programozóknak is javasoljuk, hogy a fejezet elejétől kezdve minden programot figyelmesen tanulmányozzon át. A bemutatott programok fokozatosan egyre nehezebbek lesznek. A szükséges bevezető részekkel minden program rendelkezik, ami azt jelenti, hogy fordítás és szerkesztés után futtatható állapotban lesz.

(A 2. fejezetben bemutatott programfej a legtöbb általános assembly program sturkturálásához szükséges.) A 2. fejezetben már megismerhettük a COM és EXE file-ok közötti különbséget. A könyvben szereplő szinte valamennyi programot egyetlen szegmensben (64 K) is megírhattuk volna, ebben az esetben ui. elegendő a COM szerkezet használata. Azonban szinte valamennyi népszerű assembler támogatja az EXE file-ok kialakítását, amelyek a sokkal hatékonyabb assembler szerkezeteknek viselik gondját. Ezért a könyvben szereplő összes program – kivéve azokat, amelyeket külön megjelöltünk – rendelkezik EXE programfejjel. A fejezetben előforduló valamennyi program a következőhöz hasonló programfejet és lezáró részt használja:

```
;(itt mindig a program celjat kell megadni)

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
(Itt kell elhelyezni az aktualis adatokat)
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

(Itt kell elhelyezni az aktualis programkodot)

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A fejezet első tizenöt programja olyan általános jellegű program, amely majdnem minden 8088/80386-os gépen futtatható. A többi program gépfüggő, amelyek közül néhány az IBM AT számítógépre készült, és olyan speciális DOS és BIOS rutinokat használ, amelyek kizárólagosan IBM termékek.

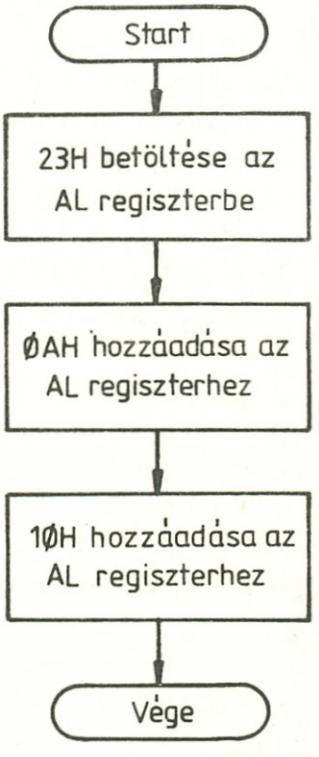
Ha az Olvasó a 80286-ossal kompatibilis gépet használ, akkor elképzelhető, hogy – a kompatibilitás fokától függően – bizonyos modosításokra lesz szüksége. Van néhány olyan program is, ami kizárólag a 80386-os mikroprocesszorra készült. A fejezetben szereplő programok mindegyike rövid és egyszerű, hiszen a programozási alapelvek elsajátításához könnyen áttekinthető, jól strukturált programokra van szükség.

5.1 ARITMETIKAI PROGRAMOK

Ebben az alfejezetben olyan assembly programok bemutatása és magyarázata található, amelyek az alapvető aritmetikai műveleteket – összeadás, kivonás, szorzás, osztás – szemléltetik. Az utolsó példa olyan egyszerű algoritmust használ, ami egy hexadecimális egész szám négyzetgyökét határozza meg. A feladatokban mindenhol hexadecimális számokkal dolgozunk. (Az eltéréseket külön jelöljük.)

Hexadecimális összeadás azonnali címzéssel

A legegyszerűbb példa, amit assembly nyelven be lehet mutatni, a hexadecimális számok összeadása. A feladat elvégzéséhez az előbbiekben bemutatott programfejhez csak három kódsort kell hozzáírnunk. Az 5–1. ábrán ennek a három lépésnek a diagramja látható. Ez a példa – bár maga a programozás egyszerű – számos olyan jellemzőt tartalmaz, ami a programot érdekessé teszi.



5–1. ábra A hexadecimális összeadás programozási lépései

Az aktuális assembly programkód listája a következő:

```
; 8088/80386-OS GEPEKRE KESZULT PROGRAM
; EGYSZERU HEXADECIMALIS OSSZEADAS BEMUTATASA AZONNALI
; CIMZES FELHASZNALASAVAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE

; HAROM SZAM OSSZEADASA
        MOV     AL,23H          ; 23H ELHELYEZESE AZ AL REGISZTERBEN
        ADD     AL,0AH          ; 0AH HOZZAADASA AZ AL REGISZTERHEZ
        ADD     AL,10H          ; 10H HOZZAADASA AZ AL REGISZTERHEZ
; VEGE AZ OSSZEADASI PELDANAK. AZ EREDMENY AL-BEN

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

Először a program szerkezetét vizsgáljuk meg.

Az első három sorban szereplő megjegyzésből megtudhatjuk, hogy a program milyen gépeken futtatható, és milyen feladatot lát el. A megjegyzéseknek – amelyek a négy mező bármelyikében elhelyezhetők – pontosvesszővel kell kezdődniük.

A következő kódcsoport a STACK-nek elnevezett veremszegmens deklarációjával kezdődik. A veremszegmens elnevezése tetszőleges. A veremszegmens létrehozásával és a MYSTACK szó 64-szeri betöltésével lehetővé válik, hogy olyan szoftver eszközöket használhassunk, mint a nyomkövetés (debug). A MYSTACK szó jelzőként szolgál, így a gép memóriájának ebben a részében könnyen rátalálhatunk a végeredményeinkre.

Ebben a példában nem használtunk különállóan tárolt adatokat, így adatszegmens létrehozására nem volt szükség. A kódszegmensnek a MYCODE elnevezést adtuk. A kódszegmens elnevezése az engedélyezett assembly karakterek felhasználásával tetszőlegesen választható meg. Minden kódszegmensnek tartalmaznia kell egy eljárást, ami a mi esetünkben a MYPROC elnevezést viseli. A programfej további részét a 2. fejezetben már részletesen tárgyaltuk.

A hexadecimális összeadást a következő kódsorok valósítják meg:

```
MOV     AL,23H          ; 23H ELHELYEZESE AZ AL REGISZTERBEN
ADD     AL,0AH          ; 0AH HOZZAADASA AZ AL REGISZTERHEZ
ADD     AL,10H          ; 10H HOZZAADASA AZ AL REGISZTERHEZ
```

A példánkban a 23H hexadecimális számot az AL regiszterben helyezzük el. Emlékeztetésül: Az AX regiszter két 8 bites részre (AH és AL) bontható. Mivel AL 8 bites regiszter, ezért 00H-tól FFH-ig képes a számok tárolására. A MOV utasítás a 8088/80386-os programozás egyik leggyakrabban használt utasítása, mivel ez teszi lehetővé, hogy a programozó információkat tárolhasson vagy tölthessen be. A következő művelet az AL regiszter éppen érvényes tartalmához 0AH-t ad, és így AL a művelet után 2DH-t fog tartalmazni. Az utolsó ADD

utasítás 10H-t ad AL-hez, így az összeadás végeredménye 3DH lesz. A program általános használatát ebben a példában számos tényező korlátozza.

Először: program nem veszi figyelembe a túlcsordulást (vagyis ha AL-ben a végösszeg értéke 0FFH-t meghaladja).

Másodszor: A program nem teszi lehetővé a 0FFH-nál nagyobb számok összeadását.

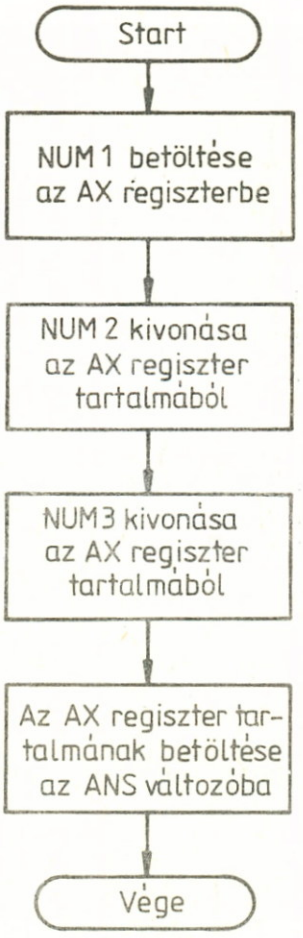
Harmadszor: A program méretét rendkívül megnövelné, ha pl. 1000 számot szeretnénk ilyen módon összeadni.

Negyedszer: Az összeadandó számok megváltoztatásához a programozónak a kódszegmens sorait kellene módosítania.

A most bemutatott program azonban még ezekkel a korlátokkal sem mondható haszontalannak, ui. néhány konstans érték összeadásához teljesen megfelelő. (Egy program hatékonysága mindig attól függ, hogy mennyire ismeri a programozó a program korlátait.)

Hexadecimális kivonás közvetlen címzéssel

A kivonási művelet ugyanolyan egyszerű, mint az összeadás. A 5–2. ábra diagramján azokat a lépéseket láthatjuk, amelyek három hexadecimális szám kivonásához szükségesek.



5–2. ábra A hexadecimális kivonás programozási lépései

Az összeadási műveletet megvalósító példánál az összeadandó számokat azonnali címzéssel helyeztük el a regiszterekbe. A jelenlegi példánál a számokhoz először változóneveket rendelünk az adatszegmensben, majd a közvetlen címzési technika felhasználásával vonjuk ki azokat egymásból. 8 bites regiszterek helyett 16 bites regisztereket használunk.

A 8088/80286-os számítógép családban a 16 bites regisztertérmet a maximális. A 80386-os processzor már 32 bites általános regisztereket tartalmaz. A 16 bites regiszterek használata

0000H-tól FFFFH-ig (decimálisan 0-tól 65532-ig) terjeszti ki a felhasználható számok tartományát. (A 8 bites regisztereknél ez a tartomány 00H-tól 0FFH-ig terjed csak.) A következő lista a kivonást tartalmazó programot mutatja.

```
;A 8088/80386-OS GEPEKRE KESZITETT PROGRAM
;A PROGRAM A HEXADECIMALIS KIVONAST MUTATJA BE KOZVETLEN
;CIMZES FELHASZNALASAVAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM1    DW      1234H           ; AZ ELSO 16 BITES SZAM
NUM2    DW      24H             ; A MASODIK 16 BITES SZAM
NUM3    DW      0ABCH           ; A HARMADIK 16 BITES SZAM
ANS     DW      ?               ; A VALASZ RESZERE FENNTARTOTT HELY
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

KIVONASI MUVELET HAROM 16 BITES SZAM FELHASZNALASAVAL
        MOV     AX,NUM1         ; NUM1 BETOLTESE AZ AX-BE
        SUB     AX,NUM2         ; NUM2 KIVONASA NUM1-BOL
        SUB     AX,NUM3         ; NUM3 KIVONASA AZ ELOBBI EREDMENYBOL
        MOV     ANS,AX          ; AZ EREDMENY BETOLTESE ANS-BA
;A KIVONASI FELADAT VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

Ez az első olyan mintaprogram, ami önálló adatszegmenst használ.

```
MYDATA  SEGMENT PARA 'DATA'
NUM1    DW      1234H           ; AZ ELSO 16 BITES SZAM
NUM2    DW      24H             ; A MASODIK 16 BITES SZAM
NUM3    DW      0ABCH           ; A HARMADIK 16 BITES SZAM
ANS     DW      ?               ; A VALASZ RESZERE FENNTARTOTT HELY
MYDATA  ENDS
```

Ismételten felhívjuk az Olvasó figyelmét arra, hogy az adatszegmensnek tetszőleges elnevezés adható, tehát a MYDATA (jelentése: az én adataim) helyett, akár a TEGNAPIADAT elnevezés is szerepelhetne, bár ez a név egy kicsit hosszú, és bizonyára senki sem szeret sokat írni. Az első sor további része azt írja elő, hogy ennek a szegmensnek a memória egyik paragrafushatáránál kell kezdődnie, és adatszegmensről (DATA) van szó. Ebben a szegmensben négy változó deklarálásáról gondoskodunk. A NUM1, NUM2 és NUM3 a kivonási feladatban szereplő számokat tartalmazzák. Minden változót szónak definiálunk (Defined Word = DW).

A 1234H, 24H és 0ABCH számok legtöbb 8088/80386-os assembler esetében 16 bitre egészülnek ki. A programozónak tehát úgy kell ezekre a számokra gondolnia, mintha 1234H, 0024H és 0ABCH-ként szerepelnének az adatszegmensben. Az ANS (Answer = válasz) elnevezésű negyedik változót deklaráltuk, de nem állítottunk be hozzá semmilyen kezdőértéket, csak egy kérdőjelet (?) írtunk a kezdőértékek oszlopába. A kédőjel 16 bites tárolási egység lefoglalását biztosítja az ANS változó részére. A szegmens utolsó sora a MYDATA elnevezésű adatszegmens lezárásáról gondoskodik (ENDSegment = szegmens vége).
A kivonást végrehajtó programkód ezek után már szinte magától értetődik:

```
MOV     AX,NUM1         ; NUM1 BETOLTESE AZ AX-BE
SUB     AX,NUM2         ; NUM2 KIVONASA NUM1-BOL
SUB     AX,NUM3         ; NUM3 KIVONASA AZ ELOBBI EREDMENYBOL
MOV     ANS,AX          ; AZ EREDMENY BETOLTESE ANS-BA
```

A NUM1 elnevezésű számot az AX regiszterbe helyezzük el. A következő két kódsor NUM2-öt és NUM3-at kivonja az AX tartalmából. A negyedik kódsor az AX regiszter tartalmát (754H) az ANS változóba helyezi el. Ahhoz, hogy az eredményt megtudjuk, jelenleg nincs semmilyen más eszközünk, mint az adatszegmens tárkiíratása (dump). A következőkben egy olyan memóriakiíratást mutatunk be, amit a program futása után az IBM DEBUG programjának felhasználásával készítettünk. Figyeljük meg, hogy a 16 bites számok 8 bites csoportokban kerülnek tárolásra.

```
-D DS : 0000
1C9D : 0000  34 12 24 00 BC 0A 54 07-00 00 00 00 00 00 00 00  4.$...T.........
1C9D : 0010  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1C9D : 0020  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1C9D : 0030  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1C9D : 0040  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1C9D : 0050  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1C9D : 0060  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1C9D : 0070  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
```

Ebből a programból nemcsak a hexadecimális kivonás módját tudhattuk meg, hanem azt is, hogyan lehet az előző feladat korlátain felülkerekedni.

Először: Az adatszegmens használatával a kivonandó számokat a valóságos programkód módosítása nélkül megváltoztathatjuk.

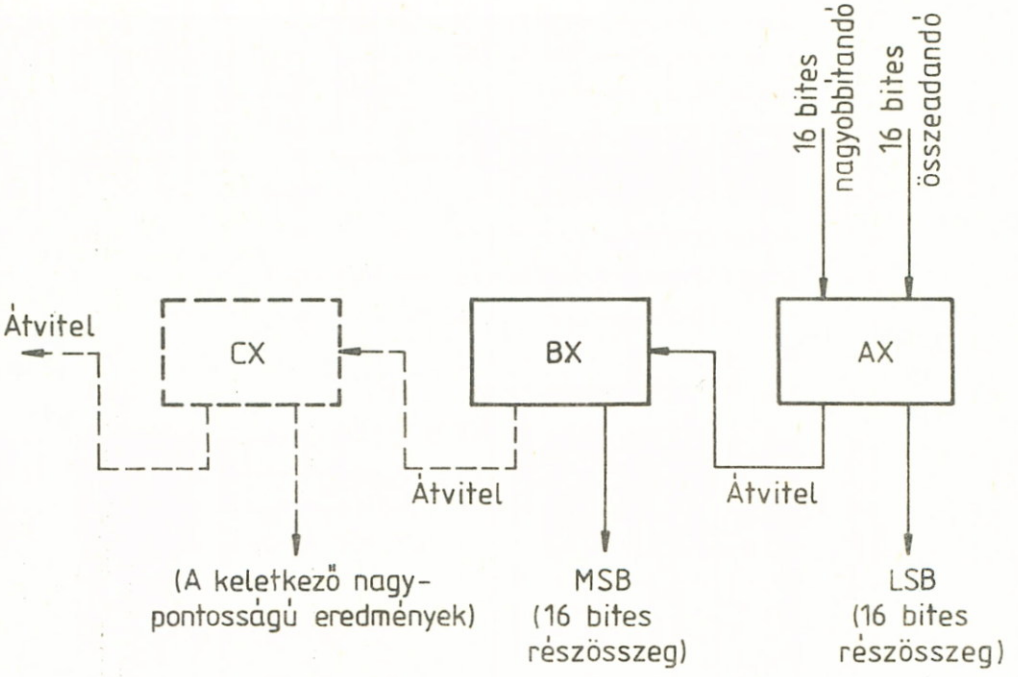
Másodszor: 16 bites számok használata egyrészt nagyobb pontosságot biztosít, másrészt a programozónak lehetősége nyílik nagyobb számokkal való műveletek végrehajtására is.

Többszörös pontosságú összeadás közvetlen címzéssel

A 8088/80386-os processzorcsalád általános célokat szolgáló AX, BX, CX, DX regiszterei 16 bitesek. A programozók tehát 0000H-tól 0FFFF-ig (decimálisan 0-tól 65535-ig) terjedő egész számok használatára lennének kényszerítve, ha nem állna rendelkezésre olyan programozási technika, amivel ezt a korlátozást el lehet kerülni. A következő példában általános eljárással 32 bites aritmetikát valósítottunk meg, ami maximálisan a 0FFFFFFFFH szám (decimálisan 4294967295) ábrázolását teszi lehetővé a 8088/80286-os gépeken.

Bár a 80386-os processzor lehetővé teszi a 32 bites aritmetikát, elképzelhető hogy szükség lehet 0FFFFFFFFH-nál nagyobb számok kezelésére. Az átviteljelző bitet vagy túlcsordulási

jelzőbitet valamennyi processzor 1-re állítja be, ha az aritmetikai művelet úgy kívánja. Az első két programban ezt a tulajdonságot nem használtuk fel. Ha átviteli művelet következik be, és erről a program nem szerez tudomást, helytelen eredményhez juthatunk. Az ADC (összeadás átvitellel) művelet lehetővé teszi, hogy a program az átvitelről információt szerezzen, és ennek előnyeit kihasználva lehetővé tegye a többszörös pontosságú aritmetikát. Az 5–3. ábra a többszörös pontosságú összeadási művelet megvalósításának módját, míg az 5–4. ábra a program általános folyamatábráját mutatja. Emlékeztetőül: Az átviteljelző bit minden ADD vagy ADC művelet után beállítódik vagy törlődik, de ezt az információt csak az ADC utasítás használja fel.



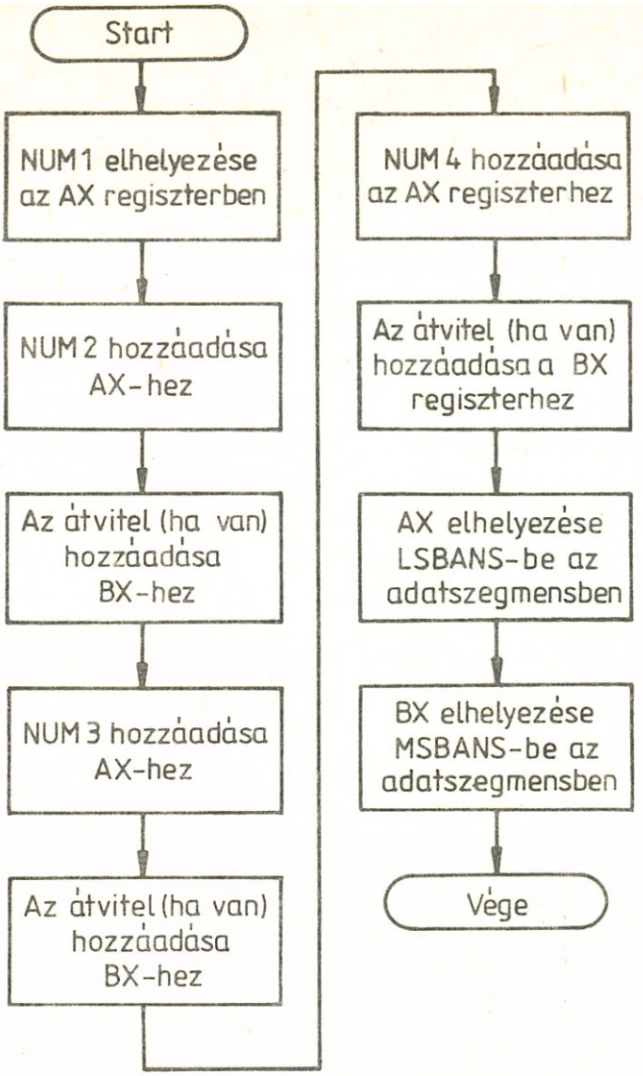
5–3. ábra Nagy pontosságú összeadás megvalósítása

A program az LSB-vel (alsó byte-tal) kezdi a művelet végrehajtását és MSB (legnagyobb helyiértékű bit) felé halad. Lehetséges, hogy két 16 bites szám összeadása nem eredményez átvitelt, míg más esetben átvitel keletkezik. Például az 1234H és 1532H számok összeadása során nem keletkezik átvitel, míg a 0ABCDH és 5600H esetében igen. Ha LSB számjegyekkel dolgozunk, a megelőző összeadásokból nem származik átvitel. Ezért az LSB összeadásokat rendszerint az ADD művelettel végezzük el. Ha átvitel keletkezik, az átviteljelző bit beáll 1-re, amelyet aztán az ADC utasítás a felső byte-ra vonatkozó 16 bites összeadás elvégzésekor felhasznál.

Csak az átviteli információ kerülhet a magasabb 16 bitbe, mivel a megadott összeadandó számok az alsó byte-ra vannak korlátozva. Emiatt az ADC műveletet 0000H-val végezzük el. Az 5–4. ábra segítségével a teljes programot áttekinthetjük. A többszörös pontosságú 32 bites eredményt két 16 bites változóban (LSBANS és MSBANS) tároljuk.

A feladatot megvalósító program a következő:

```
MOV     AX,NUM1         ; NUM1 BETOLTESE AZ AX-BE
ADD     AX,NUM2         ; A MASODIK SZAM HOZZAADASA AX-HEZ
ADC     BX,00H          ; ATVITEL-HA VAN-HOZZAADASA BX-HEZ
ADD     AX,NUM3         ; A HARMADIK SZAM HOZZAADASA AX-HEZ
ADC     BX,00H          ; ATVITEL-HA VAN-HOZZAADASA BX-HEZ
ADD     AX,NUM4         ; A NEGYEDIK SZAM HOZZAADASA AX-HEZ
ADC     BX,00H          ; ATVITEL-HA VAN-HOZZAADASA BX-HEZ
MOV     LSBANS,AX       ; AX TARTALMANAK ELMENTESE LSBANS-BE
MOV     MSBANS,BX       ; BX TARTALMANAK ELMENTESE MSBANS-BE
```



5–4. ábra A nagy pontosságú összeadás programozási lépései

```
;A 8088/8038. JS GEPEKRE KESZITETT PROGRAM
;A PROGRAM A TOBBSZOROS PONTOSSAGU OSSZEADAST MUTATJA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM1    DW      0FB00H          ; AZ ELSO 16 BITES SZAM
NUM2    DW      0F01BH          ; A MASODIK 16 BITES SZAM
NUM3    DW       2000H          ; A HARMADIK 16 BITES SZAM
NUM4    DW      0E000H          ; A NEGYEDIK 16 BITES SZAM
LSBANS  DW       ?              ; AZ EREDMENY LSB RESZE
MSBANS  DW       ?              ; AZ EREDMENY MSB RESZE
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
```

```
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;NEGY INTEGER SZAM OSSZEADASA
        MOV     AX,NUM1         ; NUM1 BETOLTESE AZ AX-BE
        ADD     AX,NUM2         ; A MASODIK SZAM HOZZAADASA AX-HEZ
        ADC     BX,00H          ; ATVITEL-HA VAN-HOZZAADASA BX-HEZ
        ADD     AX,NUM3         ; A HARMADIK SZAM HOZZAADASA AX-HEZ
        ADC     BX,00H          ; ATVITEL-HA VAN-HOZZAADASA BX-HEZ
        ADD     AX,NUM4         ; A NEGYEDIK SZAM HOZZAADASA AX-HEZ
        ADC     BX,00H          ; ATVITEL-HA VAN-HOZZAADASA BX-HEZ
        MOV     LSBANS,AX       ; AX TARTALMANAK ELMENTESE LSBANS-BE
        MOV     MSBANS,BX       ; BX TARTALMANAK ELMENTESE MSBANS-BE
;AZ OSSZEADASI MUVELET VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

5–5. ábra

A NUM1 és NUM2 számok összeadása az első feladatban már megismert MOV és ADD ismételt elvégzésével történik.

Ha az összeadás következtében az átviteljelző bit beáll, akkor a következő kódsor (ADC BC,00H) az átvitelt hozzáadja a BX regiszter tartalmához.

Az ADD/ADC sorrend minden egyes összeadandó szám esetében megismétlődik. A végösszeget két regiszterben képezzük. A BX tartalmazza az eredmény felső bitjeit, míg AX a alsó bitjeit. A regiszterek tartalmát az adatszegmens két változójába (LSBANS, HSBANS) helyezzük el. A két szám összeláncolásával, az eredmény 0002EB1BH-nak olvasható.

A most megismert módszer minden olyan esetben alkalmazható, amikor az összeadandó számok mérete meghaladja egy adott számítógép regisztereinek méretét.

Az eddig bemutatott programozási technikáknak van egy hátrányuk. Minden egyes összeadáshoz vagy kivonáshoz egy-egy (néha kettő) programsor tartozik. Néhány szám összeadásánál ez nem is jelent problémát. Képzeljünk el azonban egy olyan esetet, amikor 20 szám összeadását szeretnénk ilyen módon megoldani. A program hosszú és nehézkes lenne.

Többszörös pontosságú összeadás indexelt címzéssel

Az "egy összeadás egy sor" programozási technika hátrányainak megszüntetése a következő módon lehetséges:

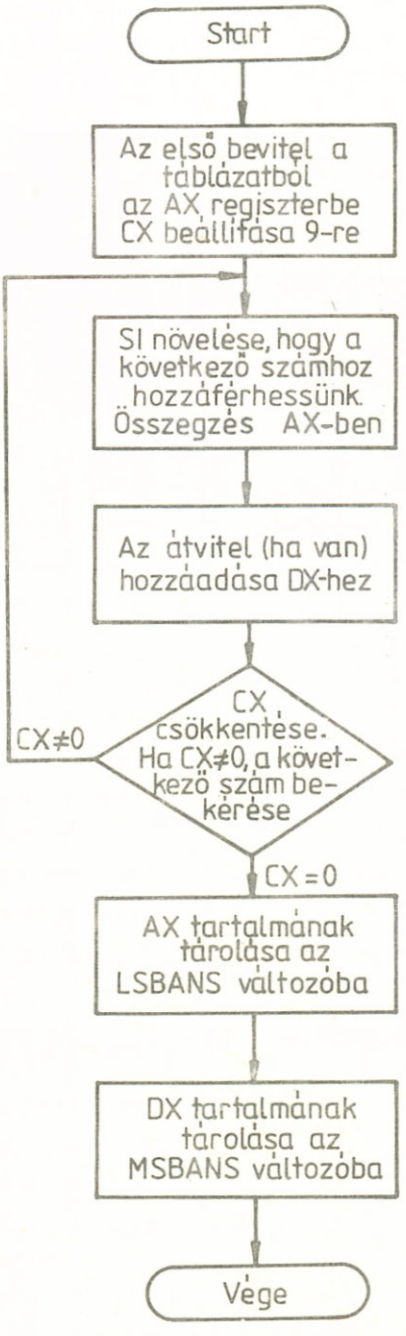
1. A számok tárolási táblázatának (table) kialakítása.
2. Programciklus szervezése, amellyel ugyanazt a kódot többször aktivizálhatjuk.
3. Indexelt címzés, amellyel a tárolási táblázatban az egyik számról a másikra léphetünk.

A számok táblázatos tárolásával egyetlen változónévhez több számot is hozzárendelhetünk.

A következő adatszegmens ezt illusztrálja:

```
MYDATA  SEGMENT PARA 'DATA'
TABLE   DW      1234H,5678H,9ABCH,0DEF0H,1111H,2222H,3333H,
        DW      4444H,5555H,6666H,7777H,8888H,9999H,0AAAAH,
        DW      0BBBBH,0CCCCH,0DDDDH,0EEEEH,0FFFFH
LSBANS  DW      ?
MSBANS  DW      ?
MYDATA  ENDS
```

A 19 bejegyzést tartalmazó TABLE elnevezésű táblázatban minden egyes adatot szónak definiáltunk. Annak érdekében, hogy a program ezekkel a számokkal műveleteket tudjon végezni, szükség van egy olyan technikára, amivel a táblázat tárolási helyére be lehet lépni, és onnan a szükséges számokat ki lehet emelni. Az 5–6. ábrán egy olyan többszörös pontosságú összeadási művelet lépéseit követhetjük végig, amelynél ciklus és indexelt címzés szerepel. A ciklus segítségével a program annyiszor halad végig a ciklusba foglalt utasításokon, ahányszor azt előírjuk.



5–6. ábra Nagy pontosságú összeadás lépései ciklus és indexelt címzés felhasználásával

Az összeadási művelet cikluson belüli használata tehát kiküszöböli a művelet ismételt leírását. A következő lista egy olyan programot mutat, amellyel a táblázatban szereplő első tíz szám összeadását valósítottuk meg.

```
;8088/8386-OS GEPEKRE KESZITETT PROGRAM
;TOBBSZOROS PONTOSSAGU OSSZEADAS TABLAZATBAN TAROLT ADATOK
;INDEXELT  CIMZESEVEL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
TABLE   DW      1234H,5678H,9ABCH,0DEF0H,1111H,2222H,3333H,
        DW      4444H,5555H,6666H,7777H,8888H,9999H,0AAAAH,
        DW      0BBBBH,0CCCCH,0DDDDH,0EEEEH,0FFFFH
LSBANS  DW      ?
MSBANS  DW      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;A TABLAZATBAN TAROLT ELSO TIZ SZAM OSSZEADASA
;INDEXELT CIMZESSEL
        LEA     BX,TABLE        ; A TABLAZAT HELYE BX-BE KERUL
        MOV     SI,00H          ; AZ INDEX BEALLITASA ZERUSRA
        MOV     AX,TABLE[SI]    ; AZ ELSO SZAM BEKERESE
        MOV     CX,09H          ; AZ OSSZEADASOK SZAMANAK BEALLITASA
AGAIN:  ADD     SI,02H          ; A SZO INDEXELESE A TABLAZATBAN
        ADD     AX,TABLE[SI]    ; RESZOSSZEG KEPZESE
        ADC     DX,00H          ; ATVITEL-HA VAN- HOZZAADASA DX-HEZ
        LOOP    AGAIN           ; HA CX <> 0, A KOV. SZAM BEKERESE
        MOV     LSBANS,AX       ; AX ELHELYEZESE LSBANS-BE
        MOV     MSBANS,DX       ; DX ELHELYEZESE MSBANS-BE
;VEGE AZ OSSZEADASI FELADATNAK

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A program leglényegesebb részét külön is kiemeljük:

```
        LEA     DX,TABLE        ; A TABLAZAT HELYE BX-BE KERUL
        MOV     SI,00H          ; AZ INDEX BEALLITASA ZERUSRA
        MOV     AX,TABLE[SI]    ; AZ ELSO SZAM BEKERESE
        MOV     CX,09H          ; AZ OSSZEADASOK SZAMANAK BEALLITASA
AGAIN:  ADD     SI,02H          ; A SZO INDEXELESE A TABLAZATBAN
        ADD     AX,TABLE[SI]    ; RESZOSSZEG KEPZESE
        ADC     DX,00H          ; ATVITEL-HA VAN- HOZZAADASA DX-HEZ
        LOOP    AGAIN           ; HA CX <> 0, A KOV. SZAM BEKERESE
```

Az SI indexregiszter, ami a táblázaton belüli eltolást adja, a zérus kezdőértéket veszi fel. A táblázat első eleme (1234H), még mielőtt a program a ciklusba belépne, az AX regiszterbe kerül. A ciklusszámlálót (CX) ezért 9-re állítjuk be, amivel tíz szám összeadását érjük el. (Az elsőt már a cikluson kívül betöltöttük!) A ciklus az AGAIN cimke és a LOOP AGAIN programsor között helyezkedik el. A ciklusban legelőször 02H-t adunk az indexregiszter értékéhez, ami azt jelenti, hogy a táblázatban a mutatót a következő számra állítjuk. (A mikroprocesszorok byte-orientáltak, tehát a következő szóra történő rámutatáshoz kétbyte-os növekményt kell előírnunk.) A kód következő sora kikeresi a táblázatnak azt az elemét, amelyre az indexregiszterrel (SI) rámutattunk, kiveszi az 5678H számot és hozzáadja az AX regiszter éppen érvényes tartalmához. Az összeadás során esetlegesen előforduló túlcsordulást a DX regiszter gyűjti össze. Ha a tíz szám összeadása még nem történt meg, a program visszaugrik az AGAIN cimkére és megismételi a műveleteket. A ciklusszámláló értéke minden alkalommal 1-gyel csökken, és amikor értéke zérussá válik, a program kilép a ciklusból. A DX/AX-ben keletkezett eredményeket két 16 bites változó tárolja. A végeredményt az adatszegmens kiíratásával tudhatjuk meg:

```
-D DS : 0000
1CC1 : 0000  34 12 78 56 BC 9A F0 DE-11 11 22 22 33 33 44 44  4.xV ..... ""33DD
1CC1 : 0010  55 55 66 66 77 77 88 88-99 99 AA AA BB BB CC CC  UUffww ..... .....
1CC1 : 0020  DD DD EE EE FF FF BD 48-03 00 00 00 00 00 00 00  ..... H .....
1CC1 : 0030  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CC1 : 0040  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CC1 : 0050  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CC1 : 0060  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CC1 : 0070  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
```

Decimális számok összeadása közvetett regisztercímzéssel

Az assemblereknek a hexadecimális aritmetika a természetes, hiszen a mikroprocesszor bináris rendszerben dolgozik. (Fordítói megj.: a bináris tömörített kifejezése a hexadecimális.) Előfordulhatnak azonban olyan esetek, amikor a programozó decimális számokat szeretne összeadni vagy kivonni. Legtöbb assembler lehetővé teszi, hogy a numerikus adatokat bináris, decimális, hexadecimális, vagy oktális formában beírhassuk. Ez azonban nem jelenti azt, hogy a számítógép a műveleteket ezekben a különböző számrendszerekben hajtja végre. Arról van szó, hogy az assembler binárisba váltja át ezeket a számokat, így mentesít bennünket ez alól a feladat alól. A hexadecimális (bináris) rendszerektől eltérő számrendszerekben végzendő műveletek azonban speciális utasításokat igényelnek.

A 8088/80386-os mikroprocesszor család rendelkezik két utasítással (DAA,DAS), amelyek segítségével decimális aritmetikai műveletek korlátozott mértékben elvégezhetők.

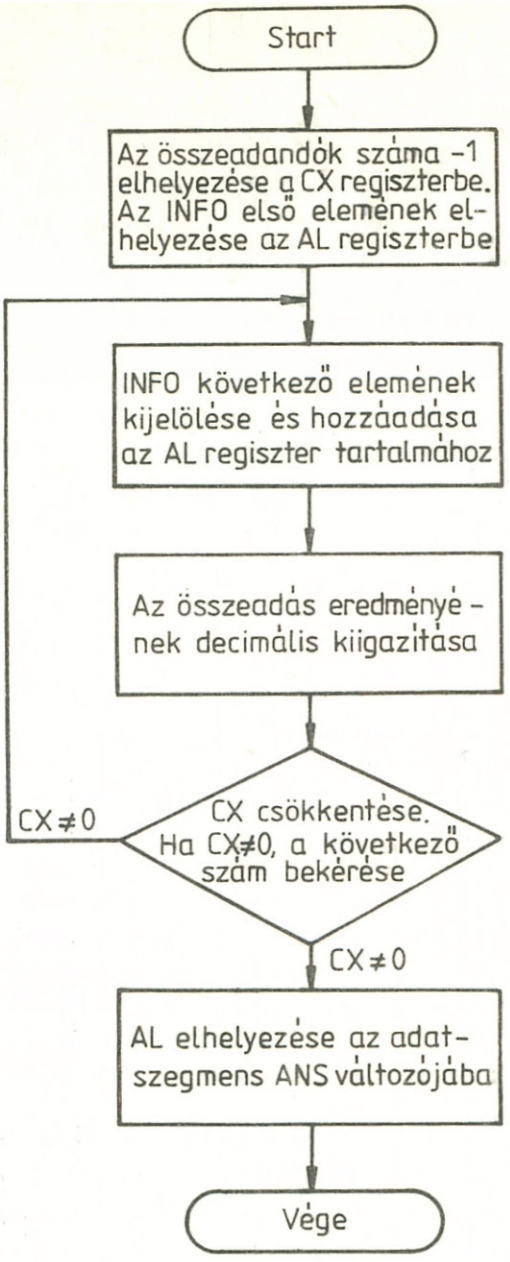
Az 5–7. ábrán a közvetett regisztercímzést alkalmazó decimális összeadás lépései figyelhetők meg.

Ha a DAA utasítással több számot adunk össze, számos tényezőre kell tekintettel lennünk.

Először: az összeadandó számoknak decimális számként kell szerepelniük. Például: 3, 4, 8, 11, 12, 30, 89. A következő számokkal nem kapnánk helyes eredményt: 1AH, 0AH, 1BH.

Másodszor: A DAA utasítás csak az AL regiszterrel működik.

Harmadszor: A DAA utasítást közvetlenül az ADD vagy ADC utasítás után kell alkalmaznunk. Olvassuk el még egyszer a DAA utasításról leírtakat. Vegyük észre, hogy a DAA nem vált át mindent a szó szorosan vett értelmében, hanem a regiszter tartalmát kiigazítja úgy, hogy



5–7. ábra Decimális összeadás közvetett regisztercímzés felhasználásával

a szám decimális alakban jelenjen meg. Ettől még a számítógép gondokozása nem változik meg! A gép csak hexadecimális számokkal tud dolgozni. Tehát ha pl. a 6 és az 5 összegét képezi, 0BH hexadecimális eredmény keletkezik. A DAA utasítás ehhez még 06H-t ad, és így áll elő a 11 szám, mint az összeadás decimális eredménye. (Természetesen a 11 mint hexadecimális 11H kerül tárolásra.) Most pedig lássuk a decimális összeadást megvalósító program listáját!

Figyeljük meg, hogy az adatszegmensben tárolt számok formája olyan, amilyet a decimális összeadás megkövetel:

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;A PROGRAM A DAA UTASITAS HASZNALATAT MUTATJA BE
;A KOZVETLEN REGISZTERCIMZES FELHASZNALASAVAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
INFO    DB      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
ANS     DB      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;AZ INFO TABLAZATBAN ELHELYEZETT ELSO 11 DECIMALIS SZAM
;OSSZEADASA

        LEA     BX,INFO         ; INFO HELYENEK BETOLTESE BX-BE
        MOV     CX,10           ; AZ OSSZEADASOK SZAMA
        MOV     AL,INFO         ; AZ ELSO SZAM BEKERESE AZ INFO-BOL
AGAIN:  ADD     BX,01H          ; A KOVETKEZO SZAM KIJELOLESE
        ADD     AL,INFO[BX]     ; RESZOSSZEG KEPZESE
        DAA                     ; AZ AL DECIMALIS KIIGAZITASA
        LOOP    AGAIN           ; MEGTORTENT A 11 OSSZEADAS?
        MOV     ANS,AL          ; A VEGEREDMENY ANS-BA KERUL
;A DECIMALIS OSSZEADASI MUVELETNEK VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A kódszegmens legfontosabb részét külön is kiemeljük:

```
        LEA     BX,INFO         ; INFO HELYENEK BETOLTESE BX-BE
        MOV     CX,10           ; AZ OSSZEADASOK SZAMA
        MOV     AL,INFO         ; AZ ELSO SZAM BEKERESE AZ INFO-BOL
AGAIN:  ADD     BX,01H          ; A KOVETKEZO SZAM KIJELOLESE
        ADD     AL,INFO[BX]     ; RESZOSSZEG KEPZESE
        DAA                     ; AZ AL DECIMALIS KIIGAZITASA
        LOOP    AGAIN           ; MEGTORTENT A 11 OSSZEADAS?
```

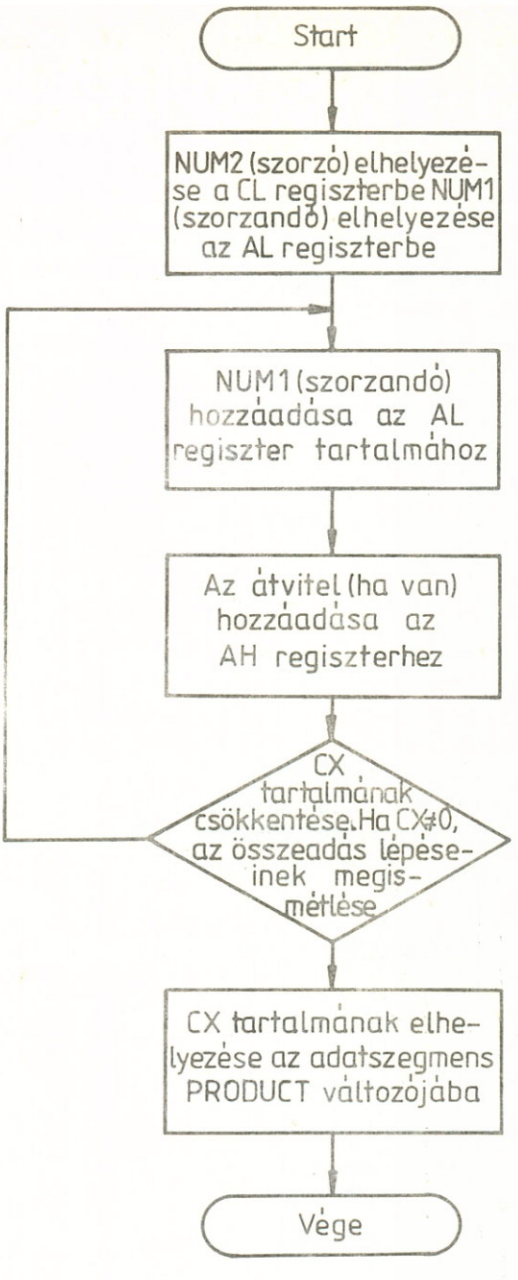
A CX regiszter tartalmazza a ciklusszámlálót. A jelenlegi programszegmens annyiban különbözik az előzőtől, hogy az index-regisztert nem használtuk fel a táblázat elemeire való rámutatáshoz, ui. BX növelése elegendő volt ahhoz, hogy a táblázat elemeihez hozzáférjünk.

Ez alkalommal BX-et 1-gyel kell növelni, mert az INFO táblázat csupa byte méretű számból áll. A ciklus szervezése ugyanaz, mint az előző példánál, kivéve, hogy az ADD utasítás után közvetlenül a DAA utasítás áll. Amikor a táblázatban szereplő első 11 szám összeadása megtörtént, AL tartalma az ANS változóba kerül, és a program befejeződik. A következő listából a program futása után képződött eredményeket figyelhetjük meg. (Csak emlékeztetőül: az adatszegmens minden adata hexadecimális alakban van.) Próbáljuk megtalálni a kiinduló mennyiségeket és az eredményt!

```
-D DS : 0000
1CE5 : 0000  01 02 03 04 05 06 07 08-09 0A 0B 0C 0D 0E 0F 66  ..............  F
1CE5 : 0010  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CE5 : 0020  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CE5 : 0030  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CE5 : 0040  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CE5 : 0050  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CE5 : 0060  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
1CE5 : 0070  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
```

Szorzás ismételt összeadással

Az assembly programozás kezdetén – még a 6800-as és 6502-es mikroprocesszorok idején – csak az összeadás és a kivonás állt a programozók rendelkezésére, így magának a programozónak kellett megalkotnia azt az algoritmust, amivel a szorzást és az osztást el tudta végezni. A 8088/80386-os processzorcsalád mind a szorzási, mind az osztási műveletekkel rendelkezik. Még mielőtt ezekkel megismerkednénk, tanulmányozzuk át a következő egyszerű algoritmust, ami a szorzást ismételt összeadási műveletekkel valósítja meg. Általánosan ismert, hogy pl. az 5 * 3 eredményét úgy is megkaphatjuk, hogy az 5-öt háromszor önmagához hozzáadjuk. Ez az alapja annak az algoritmusnak, amit most bemutatunk. Az eljárás fő hátránya a ciklusok ismételt végrehajtásának időigényessége. Az 5–8. ábrán azokat a programozási lépéseket figyelhetjük meg, amelyek két 8 bites szám összeszorzásához szükségesek. Ebben a példában a szorzó a ciklusszámláló szerepét tölti be. A szorzandót mindaddig önmagához adjuk, amíg az összeadások száma a szorzó értékével egyezik meg.



5–8. ábra Szorzás ismételt összeadással

Most pedig lássuk a szorzás programjának listáját!

```
; 8088/80386-OS GEPEKRE KESZITETT PROGRAM
; SZORZAS MEGVALOSITASA ISMETELT OSSZEADASOKKAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM1    DB      2AH
NUM2    DB      78H
PRODUCT DW      ?
MYDATA  ENDS
```

```
MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;A SZORZAST MEGVALOSITO KOD
        SUB     AX,AX           ; AZ AX REGISZTER NULLAZASA
        SUB     CX,CX           ; A CX REGISZTER NULLAZASA
        MOV     CL,NUM2         ; A SZORZO ELHELYEZESE CL-BEN
        SUB     CL,01H          ; AZ INDEX BEALLITASA
        MOV     AL,NUM1         ; A SZORZANDO ELHELYEZESE AL-BEN
AGAIN:  ADD     AL,NUM1         ; RESZOSSZEG KEPZESE
        ADC     AH,00H          ; AZ ATVITEL-HA VAN- ELMENTESE
        LOOP    AGAIN           ; ISMETLES, HA CL<>0
        MOV     PRODUCT,AX      ; A VEGEREDMENY A PRODUCT-BA KERUL
;A SZORZASI MUVELET VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

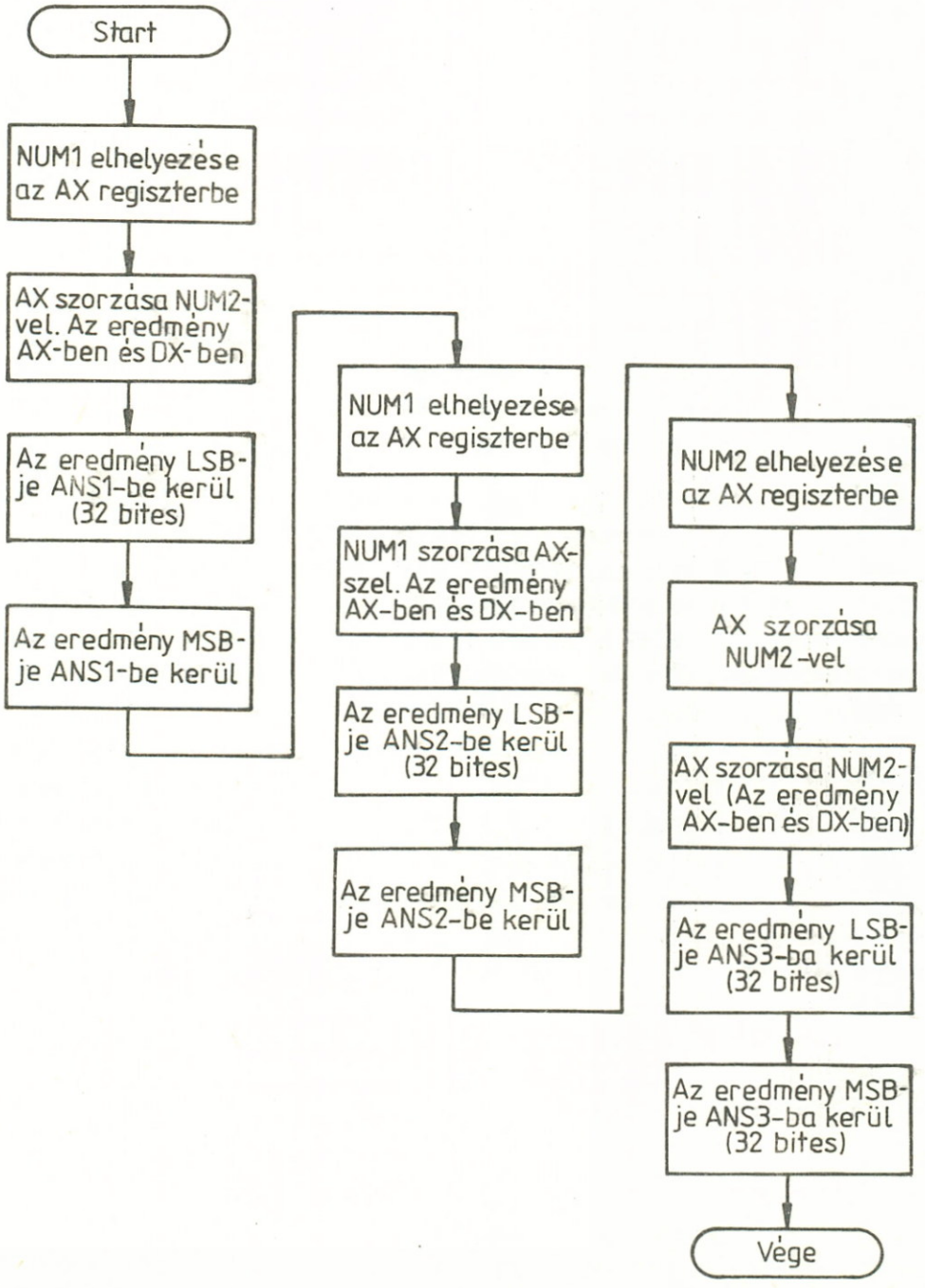
A következőkben a program leglényegesebb részét emeljük ki:

```
        SUB     AX,AX           ; AZ AX REGISZTER NULLAZASA
        SUB     CX,CX           ; A CX REGISZTER NULLAZASA
        MOV     CL,NUM2         ; A SZORZO ELHELYEZESE CL-BEN
        SUB     CL,01H          ; AZ INDEX BEALLITASA
        MOV     AL,NUM1         ; A SZORZANDO ELHELYEZESE AL-BEN
AGAIN:  ADD     AL,NUM1         ; RESZOSSZEG KEPZESE
        ADC     AH,00H          ; AZ ATVITEL-HA VAN- ELMENTESE
        LOOP    AGAIN           ; ISMETLES, HA CL<>0
```

A CX és AX regisztereket előzetesen nulláztuk, hogy biztosan a helyes eredményt kapjuk. Az AX általános célú regiszterben történik az összeadás és CL tartalmazza a szorzó értékét, amitől a ciklus végrehajtásának száma függ. A szorzót, miután elhelyeztük a CL regiszterben, 1-gyel csökkentjük, ui. a szorzandót még a ciklusba lépés előtt elhelyeztük AL-ben. A ciklus első menetében a szorzandót önmagához adjuk. Az ADC utasítás az átviteljelző bitek összegyűjtését szolgálja a 16 bites eredmény kialakítása miatt. A ciklus mindaddig ismétlődik, amíg a szorzó zérussá válik. Az AH regiszter tartalmazza a felső byte-ot AL pedig az alsó byte-ot, így alakul ki AX-ben a 16 bites eredmény.

Számok szorzása, négyzetre és köbre emelése a szorzás utasítással

A következő példa a szorzásra, a négyzetre- és köbre emelésre vonatkozó programcsomagokat tartalmazza. Az 5–9. ábrán azokat a programozási lépéseket követhetjük végig, amelyek a kívánt eredmények eléréséhez szükségesek. Amennyiben számok magasabb hatványát is elő akarjuk állítani, az előbbi példában megismert algoritmus szerint kell eljárnunk. Ebben az esetben a hatványozást mint szorzások ismételt műveletét kell felfognunk.



5–9. ábra Számok összeszorzásának, négyzetre és köbre emelésének programozási lépései

```
; 8088/80386-OS GEPEKRE KESZITETT PROGRAM
; A MUL UTASITAS FELHASZNALASA SZORZASHOZ
; SZAMOK NEGYZETRE ES KOBRE EMELESEHEZ
; A LEGNAGYOBB SZAM, AMINEK A KOBE ELOALLITHATO 509H

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM1    DW      1234H
NUM2    DW      00CDH
ANS1    DD      ?
ANS2    DD      ?
ANS3    DD      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;KET SZAM OSSZESZORZASA ES AZ EREDMENY TAROLASA AZ ANS1 VALTOZOBAN
        SUB     DX,DX           ; A TULCSORDULASI REGISZTER TORLESE
        MOV     AX,NUM1         ; AZ ELSO SZAM ELHELYEZESE AX-BEN
        MUL     NUM2            ; A KET SZAM OSSZESZORZASA
        MOV     WORD PTR ANS1,AX ; AX ELHELYEZESE ANS1 LSB-JEBEN
        MOV     WORD PTR ANS1+2,DX ; DX ELHELYEZESE ANS1 MSB-JEBEN
;A SZORZASI MUVELET VEGE

;EGY SZAM NEGYZETRE EMELESE SZORZASSAL.
;AZ EREDMENY AZ ANS2 VALTOZOBA KERUL.
        SUB     DX,DX           ; A TULCSORDULASI REGISZTER TORLESE
        MOV     AX,NUM1         ; AZ ELSO SZAM AX-BE KERUL
        MUL     NUM1            ; A SZAM SZORZASA ONMAGAVAL
        MOV     WORD PTR ANS2,AX ; AX ELHELYEZESE ANS2 LSB-JEBEN
        MOV     WORD PTR ANS2+2,DX ; DX ELHELYEZESE ANS2 MSB-JEBEN
;A NEGYZETRE EMELES VEGE

;EGY SZAM KOBRE EMELESE SZORZASSAL.
;AZ EREDMENY AZ ANS3 VALTOZOBA KERUL.
        SUB     DX,DX           ; A TULCSORDULASI REGISZTER TORLESE
        MOV     AX,NUM2         ; A MASODIK SZAM ELHELYEZESE AX-BEN
        MUL     NUM2            ; A SZAM SZORZASA ONMAGAVAL
        MUL     NUM2            ; A SZAM SZORZASA ONMAGAVAL
        MOV     WORD PTR ANS3,AX ; AX ELHELYEZESE ANS3 LSB-JEBEN
        MOV     WORD PTR ANS3+2,DX ; DX ELHELYEZESE ANS3 MSB-JEBEN
;A KOBRE EMELES VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A következő adatszegmensben érdekes adatbeviteleket figyelhetünk meg:

```
MYDATA  SEGMENT PARA 'DATA'
NUM1    DW      1234H
NUM2    DW      00CDH
ANS1    DD      ?
ANS2    DD      ?
ANS3    DD      ?
MYDATA  ENDS
```

Azt már tudjuk, hogy a NUM1 és NUM2 változók mindegyike 16 bites, de mi a méretük az eredményeket tároló ANS1, NS2 és ANS3 változóknak?

A DD jelentése definiált duplaszó (Defined Doubleword), ami 32 bites méretet jelent. A 8088/80286-os processzoroknál azonban óvatosan kell bánni a 32 bites tárolóegységgel, mivel az általános célú regiszterek maximális mérete 16 bites. Ahhoz, hogy 32 bites tárolási egységet létesítsünk, speciális programozási technikát kell alkalmaznunk. A szorzást megvalósító kód a következő:

```
SUB     DX,DX           ; A TULCSORDULASI REGISZTER TORLESE
MOV     AX,NUM1         ; AZ ELSO SZAM ELHELYEZESE AX-BEN
MUL     NUM2            ; A KET SZAM OSSZESZORZASA
MOV     WORD PTR ANS1,AX ; AX ELHELYEZESE ANS1 LSB-JEBEN
MOV     WORD PTR ANS1+2,DX ; DX ELHELYEZESE ANS1 MSB-JEBEN
```

A MUL utasítás az AX és DX regiszterekben gyűjti össze a 32 bites eredményt. Ennél a példánál NUM1-et az AX regiszterben helyeztük el és NUM2-vel szoroztuk be. Vegyük észre, hogy NUM2-t nem kellett előzetesen regiszterben tárolnunk. Számos aritmetikai művelet – összeadás, kivonás, egy szám növelése és csökkentése – közvetlenül elvégezhető az adatszegmens változóival anélkül, hogy azokat előzetesen általános célú regiszterekbe kellene elhelyeznünk. A szorzás eredménye a DX(000EH) és AX(93A4H) regiszterekben található. Ahhoz, hogy a DX és AX tartalmát egyetlen 32 bites változóban elhelyezhessük, a PTR utasítást kell alkalmaznunk. A MOV WORD PTR ANS1,AX programkód használatával az AX regiszter az ANS1 tárolási hely első (alacsony) 16 bitjébe kerül, MOV WORD PTR ANS1+2,DX programkód pedig azt biztosítja, hogy a DX regiszter tartalma az ANS1 változó felső 16 bitjébe jusson. A kétbyte-os növekmény használatára külön felhívjuk a figyelmet. A PTR utasítás megszüntet minden olyan kétértelműséget, ami egy 16 bites eredmény 32 bitben való elhelyezésekor előállhat. A következő programrészlet azt az egyszerű négyzetre emelési eljárást mutatja, amikor a számot önmagával szorozzuk:

```
SUB     DX,DX           ; A TULCSORDULASI REGISZTER TORLESE
MOV     AX,NUM1         ; AZ ELSO SZAM AX-BE KERUL
MUL     NUM1            ; A SZAM SZORZASA ONMAGAVAL
MOV     WORD PTR ANS2,AX ; AX ELHELYEZESE ANS2 LSB-JEBEN
MOV     WORD PTR ANS2+2,DX ; DX ELHELYEZESE ANS2 MSB-JEBEN
```

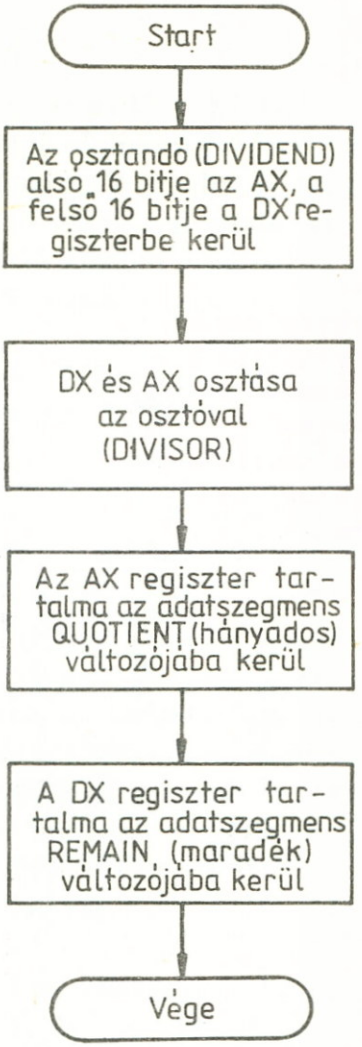
Ebben az esetben a szorzó és a szorzandó ugyanaz a szám. A DX-ben képződő 014BH és az AX-ben képződő 5A90H eredményt az ANS2 elnevezésű 32 bites változó kapja meg.

A példa harmadik programrészlete a köbre emelés technikáját mutatja be, amelynél egyszerűen csak arról van szó, hogy a számot háromszor kell önmagával szoroznunk.

Belefér az eredmény a számára kijelölt helyre? A 0CDH szám köbre emelése 8374D5H-t ad eredményül, ami egy 32 bites regiszterben elfér. Mindaddig nem következik be túlcsordulási probléma, amíg az a szám, amit köbre emelünk nem nagyobb 50AH-nál.

A DIV (osztás) művelet használata definiált duplaszóval

Az 5–10. ábrán az osztási művelethez szükséges programozási lépéseket kísérhetjük figyelemmel.



5–10. ábra A hexadecimális osztás programozási lépései

A folyamatábrából azonban nem derül ki az osztandó és az osztó mérete. Amikor az osztás műveletét a 3. fejezetben tárgyaltuk, felhívtuk a figyelmet arra, hogy az osztandó két 16 bites regisztert elfoglalhat. (DX-et és AX-et). Ebben a példában az osztandót az adatszegmensben tároltuk, és DD-nek (definiált duplaszó) definiáltuk. A következő listán a teljes program látható.

```
;A 8088/80386-OS GEPEKRE KESZITETT PROGRAM
;AZ OSZTAS UTASITAS BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
DIVIDEND DD     02A8B7654H      ; OSZTANDO
DIVISOR  DW     5ABCH           ; OSZTO
QUOTIENT DW     ?               ; HANYADAOS
REMAIN   DW     ?               ; MARADEK
MYDATA  ENDS
```

```
MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;32 BITES OSZTANDO ELOSZTASA 16 BITES OSZTOVAL
        MOV     AX,WORD PTR DIVIDEND  ; AZ OSZTANDO LSB-JE AX-BE KERUL
        MOV     DX,WORD PTR DIVIDEND+2  ;AZ OSZTANDO MSB-JE DX-BE KERUL
        DIV     DIVISOR         ;(DX AX) OSZTASA DIVISOR-RAL
        MOV     QUOTIENT,AX             ; AX ELHELYEZESE A HANYADOSBA
        MOV     REMAIN,DX               ; DX ELHELYEZESE A MARADEKBA
;AZ OSZTASI MUVELET VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A programból elsőként az adatszegmenst emeljük ki, amiben négy változót (DIVIDEND, DIVISOR, QUOTIENT, REMAIN) definiáltunk:

```
MYDATA  SEGMENT PARA 'DATA'
DIVIDEND DD     02A8B7654H      ; OSZTANDO
DIVISOR  DW     5ABCH           ; OSZTO
QUOTIENT DW     ?               ; HANYADAOS
REMAIN   DW     ?               ; MARADEK
MYDATA  ENDS
```

A változók méretét a program működéséhez igazítva előzetesen beállítottuk. A DIVIDEND (osztandó) 02A8B7654H-t (decimálisan 713782868-at), a DIVISOR (osztó) 5ABCH-t (decimálisan 23228-at) tartalmaz. A QUOTIENT (hányados) és a REMAIN (maradék) mérete nem lehet nagyobb DW-nél (definiált szó). Ezeket a memória- helyeket a ? szimbólummal foglaltuk le. A program legérdekesebb része a 32 bites osztandó felhasználása.

Az erre vonatkozó programrészlet a következő:

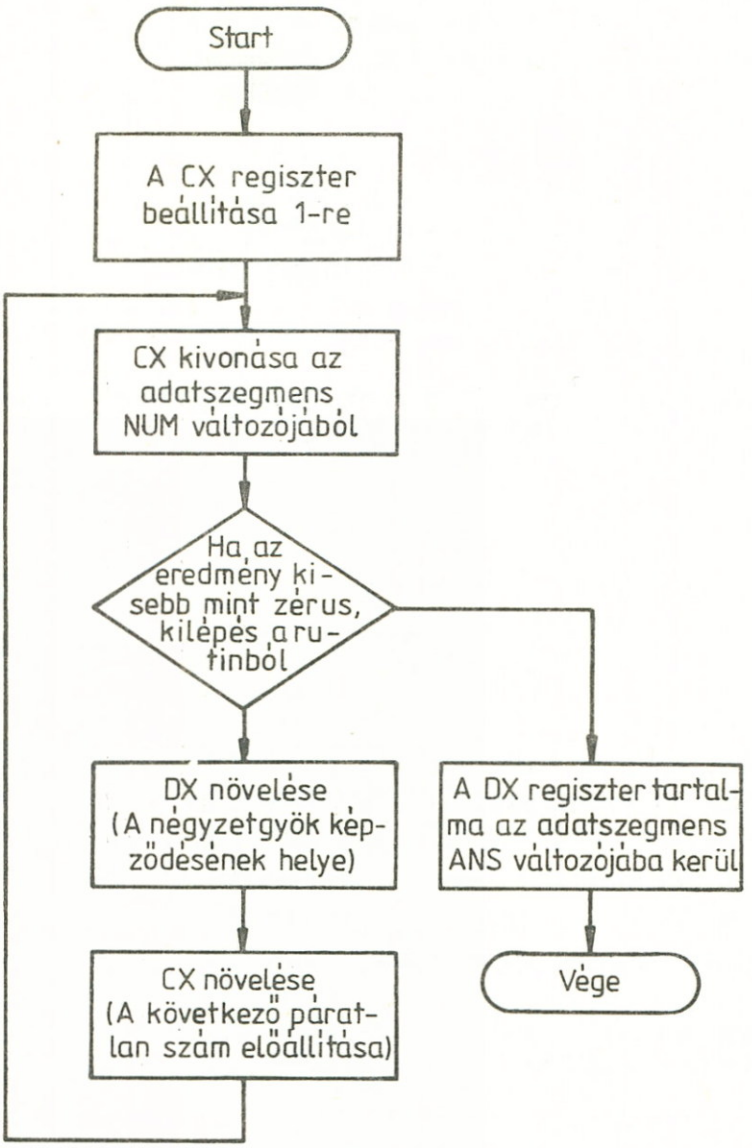
```
MOV     AX,WORD PTR DIVIDEND  ; AZ OSZTANDO LSB-JE AX-BE KERUL
MOV     DX,WORD PTR DIVIDEND+2  ;AZ OSZTANDO MSB-JE DX-BE KERUL
DIV     DIVISOR               ;(DX AX) OSZTASA DIVISOR-RAL
MOV     QUOTIENT,AX           ; AX ELHELYEZESE A HANYADOSBA
MOV     REMAIN,DX             ; DX ELHELYEZESE A MARADEKBA
```

Emlékeztetjük az Olvasót arra, hogy a DIV művelet használatakor a 32 bites szám LSB-jét AX, MSB-jét DX tartalmazza. Ezeknek a kívánalmaknak a WORD PTR használatával tudunk eleget tenni, így ui. az osztandó első 16 bitje AX-be kerül. Mivel a processzor byte-orientált, a WORD PTR DIVIDEND + 2 kódra van szükségünk ahhoz, hogy az osztandó felső 16 bitje a DX regiszterbe kerüljön. Ebben a programban az osztó (DIVISOR) közvetlenül az adatszegmensből kerül meghívásra. A hányados, 7809H (decimálisan 30729) az AX rengiszterbe, míg a maradék, 25B8H (decimálisan 9656) a DX regiszterben található.

A négyzetgyökvonás algoritmusa

Az első algoritmus, amit ebben a fejezetben bemutattunk, az ismételt összeadásokkal előállított szorzás volt. Erre az algoritmusra valójában csak az oktatási szempontok miatt volt szükség, mivel a 8088/80386-os processzorcsalád a szorzás művelettel rendelkezik. A jelenlegi példánkban valójában is szükség van az algoritmusra, mert négyzetgyökvonás utasítás nem áll készen a rendelkezésünkre.

Az 5–11. ábrán egy egyszerű algoritmust mutatunk be, amivel megkaphatjuk egy adott hexadecimális szám négyzetgyökét. Az algoritmus a következőképpen fogalmazható meg:



5–11. ábra Hexadecimális szám négyzetgyökének előállításához szükséges programozási lépések

Egy szám négyzetgyöke megközelítően úgy határozható meg, hogy a számból kivonjuk az egymás után következő páratlan számokat mindaddig, amíg az eredeti szám zérussá (ill. negatívvá) válik. A kivonások száma az eredeti szám közelítő négyzetgyökét adja. A most megismert módszerrel határozzuk meg a decimális 82 négyzetgyökét:

82 − 1 = 81
81 − 3 = 78

```
 78 -  5 = 73
 73 -  7 = 66
 66 -  9 = 57
 57 - 11 = 46
 46 - 13 = 33
 33 - 15 = 18
 18 - 17 =  1        Ez a kilencedik kivonás, ami még
  1 - 19 = -18       pozitív eredményt ad!
```

Tehát a 82 négyzetgyöke közelítőleg 9. Hexadecimális rendszerben is ugyanez az eljárás. A következő listán a négyzegyökvonás assembly algoritmusa látható.

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;NEGYZETGYOK VONAS BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM     DW      1CE4H
ANS     DW      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;A NEGYZETGYOK VONAS RUTINJA
        MOV     DX,0H           ; DX TORLESE
        MOV     CX,01H          ; A KIVONANDO SZAM BEALLITASA
AGAIN:  SUB     NUM,CX          ; A PARATLAN INT.SZAM KIVONASA NUM-BOL
        JL      OUT             ; KILEPES A RUTINBOL
        INC     DX              ; A NEGYZETGYOKERTEK NOVELESE 1-GYEL
        ADD     CX,02H          ; A KOVETKEZO PARATLAN SZAM ELOALLITASA
        JMP     AGAIN           ; A CIKLUS ISMETLESE
OUT:    MOV     ANS,DX          ; AZ EREDMENY AZ ANS VALTOZOBA KERUL
;A RUTIN VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

Az aktuális kód szinte magától értetődő:

```
        MOV     DX,0H           ; DX TORLESE
        MOV     CX,01H          ; A KIVONANDO SZAM BEALLITASA
AGAIN:  SUB     NUM,CX          ; A PARATLAN INT.SZAM KIVONASA NUM-BOL
        JL      OUT             ; KILEPES A RUTINBOL
        INC     DX              ; A NEGYZETGYOKERTEK NOVELESE 1-GYEL
        ADD     CX,02H          ; A KOVETKEZO PARATLAN SZAM ELOALLITASA
        JMP     AGAIN           ; A CIKLUS ISMETLESE
OUT:    MOV     ANS,DX          ; AZ EREDMENY AZ ANS VALTOZOBA KERUL
```

DX kezdőértékét zérusra állítjuk be, mivel ebben a regiszterben képződik a négyzetgyökvonás eredménye (a négyzetgyök). CX, amelynek 1 a kezdőértéke, az egymás után következő páratlan számokat tárolja. A ciklus legelső utasításával CX-et kivonjuk a NUM változóban tárolt értékből. Ha a kivonás eredménye negatív, a program befejeződik, míg ellenkező esetben CX a következő páratlan számra áll be, és a ciklus megismétlődik. Amikor a program a ciklusból kilép, a négyzetgyök értékét az ANS változó kapja meg.

5.2 LOGIKAI MŰVELETEK

Az assembly programozás során számos olyan eset adódik, amikor az OR, AND és XOR logikai függvényeket mint maszkokat használjuk a programjainkban. A grafikus programokban XOR-t gyakran használjuk egy előzetesen betöltött képernyőpont törlésére. Az AND egy adott byte bizonyos bitjeinek kiszűrésére használható. Mindkét alkalmazást a könyvben később még tárgyaljuk.

Logikai kapuk és műveletek szimulálása assembly nyelven

A logikai függvények egyik alkalmazási területe egyszerű hardver logikai áramkörök szimulálása. Az 5–12. ábrán négy logikai kapu diagramját láthatjuk. Az IN1, IN2, IN3 és IN4 elnevezésű négy bemenet valós hardver kapuk elektromos csatlakozó pontjait jelentik. A keletkező kimenetet F-fel jelöltük. Az áramkör az igazságtáblázat felhasználásával papíron is analilzálható. A jelenlegi példában a négy bemenet mint bináris adat szerepel az adatszegmensben. Az áramkör (program) a bináris bemenetet bitenként analizálja és előállítja a választ. Milyen lesz a kimenet a következő bemenetek esetén:

```
IN1  DB  10111010B
IN2  DB  11111100B
IN3  DB  00010100B
IN4  DB  00010011B
```

Legelőször az IN1 bitjeit egy logikai negációval megfordítjuk. Ekkor a 01000101B érték keletkezik. Ez a bemenet, valamint az IN2 OR művelete a következő eredményt állítja elő:

```
01000101B
11111100B
--------
11111101B   (Az OR kapu kimenete)
```

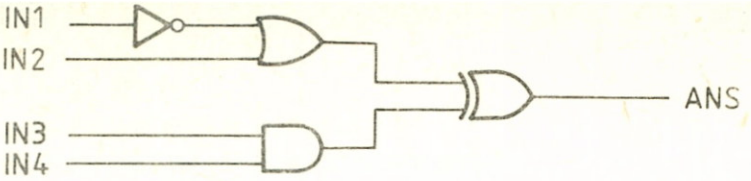
Az IN3 és IN4 értékeket egy AND műveletnek vetjük alá. Ekkor a következő eredmény keletkezik:

```
00010100B
00010011B
--------
00010000B   (Az AND kapu kimenete)
```

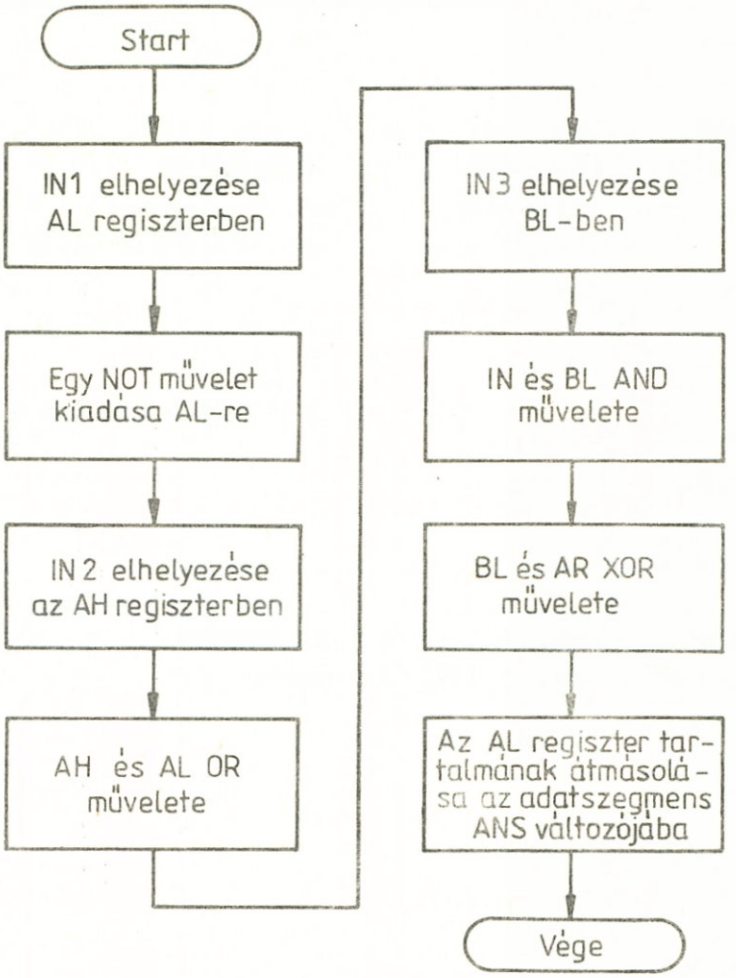
Végül az OR kapu és AND kapu kimenetét egy kizáró vagy (XOR) műveletnek vetjük alá. A következő eredményt kapjuk:

```
11111101B   (Az OR kapu kimenete)
00010000B   (Az AND kapu kimenete)
--------
11101101B   (Az XOR kapu kimenete)
            (0EDH)
```

Az 5–13. ábrán azokat a programozási lépéseket figyelhetjük meg, amelyek a hardver áramkör megvalósításához szükségesek. Annak érdekében, hogy a programot érthetőbbé tegyük, az adatokat bináris alakban szerepeltetjük. Bár a számok első látásra meglehetősen hosszúnak



5–12. ábra Szoftver modellezéséhez felhasznált egyszerű hardver logikai áramkör



5–13. ábra Az 5–12. ábra hardver áramkörének modellezéséhez szükséges programozási lépések

tűnnek, mégis elférnek egy byte-ban (DB). Ha azonban a programot egy hibakeresővel (debugger) megvizsgálnánk, az adatszegmensben szereplő számokat hexadecimális alakban kellene keresnünk. A következő listán az áramkört modellező program látható.

```
;8088/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;AZ AND/OR/XOR/NOT LOGIKAI OPERATOROK HASZNALATANAK
;BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS
```

```
MYDATA  SEGMENT PARA 'DATA'
IN1     DB      10111010B
IN2     DB      11111100B
IN3     DB      00010100B
IN4     DB      00010011B
ANS     DB      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;PELDA A LOGIKAI MUVELETEK HASZNALATARA
        MOV     AL,IN1          ; AZ ELSO BEMENET ELHELYEZESE AL-BEN
        NOT     AL              ; AL BITJEINEK INVERZE (LOGIKAI NOT)
        MOV     AH,IN2          ; A MASODIK BEMENET ELHELYEZESE AH-BAN
        OR      AL,AH           ; AL ES AH LOGIKAI OR MUVELETE
        MOV     BL,IN3          ; A HARMADIK BEMENET ELHELYEZESE BL-BEN
        AND     BL,IN4          ; BL ES IN4 LOGIKAI AND MUVELETE
        XOR     AL,BL           ; AL ES BL KIZARO VAGY MUVELETE
        MOV     ANS,AL          ; AZ EREDMENY ANS-BA KERUL
;A LOGIKAI MUVELETEKET MEGVALOSITO PELDA VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A kódszegmensben szereplő program igen egyszerű. A bemeneteket 8 bites regiszterekben olvashatjuk be, majd végrehajtjuk a kívánt logikai műveleteket. A végeredmény az ANS változóba kerül. (A program futása után az adatszegmensben található végeredmény hexadecimális alakban szerepel.)

```
MOV     AL,IN1          ; AZ ELSO BEMENET ELHELYEZESE AL-BEN
NOT     AL              ; AL BITJEINEK INVERZE (LOGIKAI NOT)
MOV     AH,IN2          ; A MASODIK BEMENET ELHELYEZESE AH-BAN
OR      AL,AH           ; AL ES AH LOGIKAI OR MUVELETE
MOV     BL,IN3          ; A HARMADIK BEMENET ELHELYEZESE BL-BEN
AND     BL,IN4          ; BL ES IN4 LOGIKAI AND MUVELETE
XOR     AL,BL           ; AL ES BL KIZARO VAGY MUVELETE
MOV     ANS,AL          ; AZ EREDMENY ANS-BA KERUL
```

5.3 KIKERESÉSI TÁBLÁZATOK SPECIÁLIS ALKALMAZÁSOKHOZ

A kikeresési táblázat – mint a neve is jelzi – a programozó számára lehetővé teszi, hogy az előzetesen már definiált táblázaból kikeressen (kiemeljen) adatokat. (Ilyen adattáblázatok képezik pl. a nagysebességű grafikus vagy helyesírást ellenőrző programok gerincét.) Akár egy egész szótár is elhelyezhető az adatszegmensben táblázatos formában. A bemeneti és a tárolt szavakat könnyen össze tudjuk hasonlítani, ha a táblázat elemeit megfelelően indexeljük. Ha a program megtalálja a szót, ezt egy üzenettel jelezheti a számunkra. Pl.:

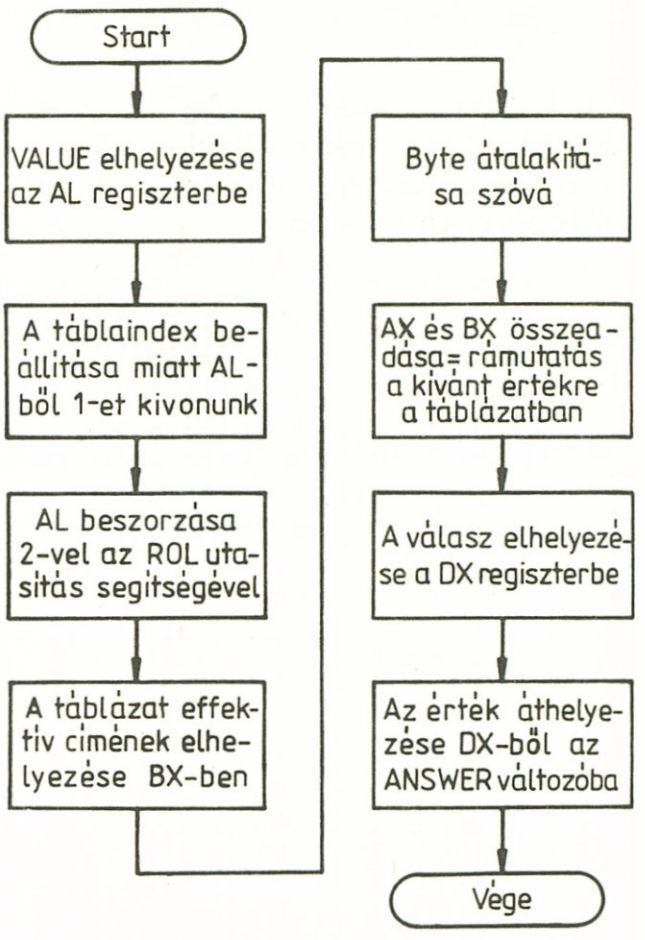
A szó helyesírása rendben.

Ellenkező esetben pl. a Probálja ismétüzenettel ajánlhatjuk fel az új próbálkozás lehetőségét.

Ebben az alfejezetben két egyszerű példát mutatunk be, a 8. fejezetben azonban a kikeresési tábázatot hasznosító helyesírást ellenőrző programmal is megismerkedhet az Olvasó.

A kikeresési táblázat használata logaritmus kiszámításához

Bizonyos speciális matematikai függvények algoritmusának nemcsak a meghatározása, hanem az assembly program elkészítése is nehéz feladat. (Talán azt is mondhatnánk, hogy minden olyan függvény, aminek nincs közvetlen assembly utasítása, speciálisnak tekinthető). A valós számokkal dolgozó társprocesszor használata adja a probléma legkönnyebb és leghatékonyabb megoldását. A nehezen kiszámítható matematikai értékek táblázatos elhelyezésével nagyon jól bemutatható a kikeresési táblázat használata.



5–14. ábra Számok logaritmusának kikereséséhez szükséges programozási lépések

Az 5–14. ábrán megfigyelhetjük azokat a lépéseket, amelyek a táblázatban már előzetesen tárolt logaritmusértékek kikereséséhez szükségesek. A program adatszegmensében egy olyan táblázatot láthatunk, amiben 1-től 10-ig terjedő decimális egészekhez tartozó 10-es alapú logaritmusértékeket tároltuk.

```
TABLE   DW      0,3010,4771,6021,6990,7782,8451,9031,9542,10000
VALUE   DW      7
ANSWER  DW      ?
```

A táblázat minden értékébe bele kell értenünk a tizedespontot. (Például a 4771-es szám a 0.4771-es logaritmus értéket jelenti.) Azt is érdemes megfigyelnünk, hogy ezeket a számokat decimális alakban tároltuk, tehát pl. a 10000-es érték egy szó méretű tárolási egységben fér csak el. A következő listán a teljes program látható.

```
;8088/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;A KIKERESESI TABLAZAT HASZNALATANAK BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
TABLE   DW      0,3010,4771,6021,6990,7782,8451,9031,9542,10000
VALUE   DB      7
ANSWER  DW      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;A KIKERESESI TABLAZAT HASZNALATA ELORE MEGADOTT INDEX
;FELHASZNALASAVAL
        MOV     AL,VALUE        ; VALUE AZ ELORE MEGADOTT INDEX
        SUB     AL,1            ; AZ INDEX KORREKCIOJA
        ROL     AL,1            ; AL SZORZASA 2-VEL
        LEA     BX,TABLE        ; A TABLAZAT KEZDOCIME BX-BE KERUL
        CBW                     ; AL ATALAKITASA BYTE-BOL SZOBA
        ADD     BX,AX           ; INDEX+TABLAHELYZET
        MOV     DX,TABLE[BX]    ; A TABLAZATBOL AZ ERTEK DX-BE KERUL
        MOV     ANSWER,DX       ; AZ EREDMENY ELHELYEZESE ANSWER-BEN
;A KIKERESESI MUVELET VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
        ;p2362.aaa
        MOV     AL,VALUE        ; VALUE AZ ELORE MEGADOTT INDEX
        SUB     AL,1            ; AZ INDEX KORREKCIOJA
        ROL     AL,1            ; AL SZORZASA 2-VEL
        LEA     BX,TABLE        ; A TABLAZAT KEZDOCIME BX-BE KERUL
```

```
CBW                 ; AL ATALAKITASA BYTE-BOL SZOBA
ADD     BX,AX       ; INDEX+TABLAHELYZET
MOV     DX,TABLE[BX]; A TABLAZATBOL AZ ERTEK DX-BE KERUL
MOV     ANSWER,DX   ; AZ EREDMENY ELHELYEZESE ANSWER-BEN
```

A program egyszerűsítésének érdekében a VALUE változóban helyeztük el azt a számot, aminek a logaritmusát keressük. A kódszegmenst külön is kiemeltük, mivel számos érdekes jellemzője van:

```
MOV     AL,VALUE    ; VALUE AZ ELORE MEGADOTT INDEX
SUB     AL,1        ; AZ INDEX KORREKCIOJA
ROL     AL,1        ; AL SZORZASA 2-VEL
LEA     BX,TABLE    ; A TABLAZAT KEZDOCIME BX-BE KERUL
CBW                 ; AL ATALAKITASA BYTE-BOL SZOBA
ADD     BX,AX       ; INDEX+TABLAHELYZET
MOV     DX,TABLE[BX]; A TABLAZATBOL AZ ERTEK DX-BE KERUL
MOV     ANSWER,DX   ; AZ EREDMENY ELHELYEZESE ANSWER-BEN
```

Az AL-ben tárolt érték, bizonyos szoftver feltételek után a kikeresési táblázat indexét jelenti. Mivel a zérusnak nincs logaritmusa, AL értékét csökkenteni kell 1-gyel. A programban egy olyan táblázat szerepel, ami szó méretű értékeket (DW) tartalmaz. Ezért az index értékét kettővel szoroznunk kell. Szorzási utasítás helyett az egyszerűbb és gyorsabb bitforgató utasítást (ROL) alkalmazzuk. Az index még mindig byte alakban van, a táblázat indexét azonban a BX regiszterbe kell majd elhelyeznünk, ami szó méretű. A CWB utasítással tehát át kell alakítanunk a byte-ot szóvá (AL-ből AX!). Ezek után AX értékét hozzá kell adnunk a táblázat kezdőcíméhez – amit már BX-be elhelyeztünk – és így kapjuk meg a táblázat adott elemének helyzetcímét. A két egymás utáni MOV utasítással a 8451 számot először DX-be, majd az ANSWER változóba tesszük. A táblázatos adattárolásnak két előnye van:

– Áttekinthető és egyszerű programszerkezet.

– Nagy pontosságú decimális vagy hexadecimális értékek alkalmazásának lehetősége.

Azonban a hátrányokról is szólnunk kell:

– Az adatokat az adatszegmensben előzetesen tárolnunk kell, ami a program rugalmasságát korlátozza. (Ha pl. a 23 logaritmusát szeretnénk megkapni, a programot módosítanunk kellene.)
– A válasz pontosságát a programozó szabja meg, nem pedig a matematikai függvény.
– Az adatbevitel során – különösen sok adatot tartalmazó táblázat esetében – nagy a hibázási lehetőség.

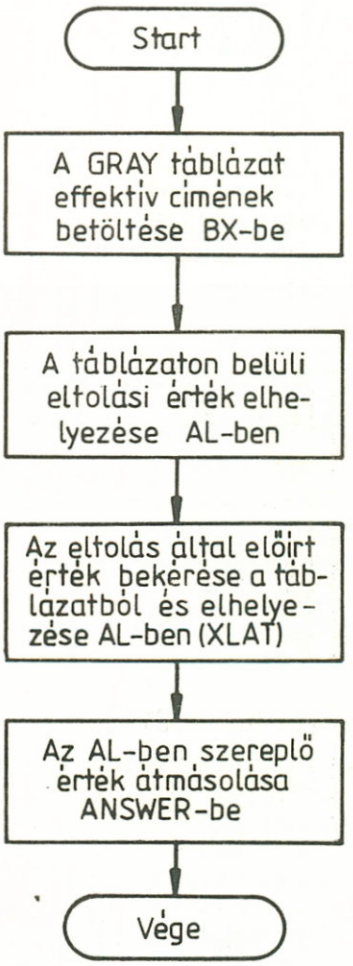
Kódrendszerek közötti átváltás táblázatban tárolt adatok használatával

Az áruk azonosításának vonalkódja vagy a telefonszám, néhány olyan kódrendszer, amivel minden nap találkozunk. A számítógépes világ ugyancsak speciális kódokkal rendelkezik. Az assambly programozásban a különböző számrendszerek alkotják a kódok egyik csoportját. További speciális kódok az ASCII, a Gray, az Excess 3 és a BCD. Az assembly nyelvben gyakran előfordul olyan feladat, amikor az egyik kódrendszerből a másikba kell áttérnünk. A súlyozott kódok konverziója viszonylag egyszerű, míg a súlyozatlan kódoknál ez a művelet

meglehetősen bonyolult is lehet. A következő példa, ami a fejezet utolsó táblázatos példája, hexadecimális számok Gray kódba való átváltását oldja meg. Mivel a hibázás lehetősége a bináris szám minden egyes bitjének beírásakor fennáll, a Gray kód egymást követő elemei mindig csak egy bitben térnek el egymástól.

```
GRAY    DB      0010B,0110B,0111B,0101B,0100B,1100B,1101B
        DB      1111B,1110B,1010B
VALUE   DB      8H
ANSWER  DB      ?
```

A programrészletben a 0–9 számok Gray kódjai szerepelnek. A programban egy új utasítást is alkalmazunk, az XLAT-ot, ami lehetővé teszi, hogy a byte-okat tartalmazó táblázatban az AL-ben előírt értékkel indexeljünk. A táblázatból beolvasott szám is az AL-be kerül. A következő listán a hexa - Gray átváltás programja látható:



5–15. ábra Hexadecimális–Gray átalakítás programozási lépései

```
;8088/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM,
;AMI A KIKERESESI TABLAZAT ES AZ XLAT UTASITAS ALKALMAZASAT
;MUTATJA BE HEXA SZAMOK GRAY KODJANAK KIKERESESEHEZ

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
GRAY    DB      0010B,0110B,0111B.0101B,0100B,1100B,1101B
        DB      1111B,1110B,1010B
VALUE   DB      8H
```

```
ANSWER  DB      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;BYTE-OKAT TARTALMAZO TABLAZAT ELEMEINEK ELERESE AZ
;XLAT UTASITAS FELHASZNALASAVAL
        LEA     BX,GRAY         ; A TABLAZAT KEZDOCIME BX-BE KERUL
        MOV     AL,VALUE        ; AZ INDEX ELHELYEZESE AL-BE
        XLAT    GRAY            ; A MEGFELELO ERTEK KIKERESESE
        MOV     ANSWER,AL       ; A TABLAZATOS ERTEK ANSWER-BE KERUL
;AZ XLAT ALKALMAZASAT BEMUTATO PELDA VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A program érdemi része csak négy sorból áll. Az XLAT utasítás alkalmazásához a táblázat helyzetének BX-ben, az indexnek AL-ben kell lennie. Az erre vonatkozó négy kódsort megismételjük:

```
LEA     BX,GRAY         ; A TABLAZAT KEZDOCIME BX-BE KERUL
MOV     AL,VALUE        ; AZ INDEX ELHELYEZESE AL-BE
XLAT    GRAY            ; A MEGFELELO ERTEK KIKERESESE
MOV     ANSWER,AL       ; A TABLAZATOS ERTEK ANSWER-BE KERUL
```

Ebben a példában a VALUE változónak a 08H értéket adtuk. (Ez az index értéke.) A program futtatása után az ANSWER az 1110B eredményt fogja tartalmazni.

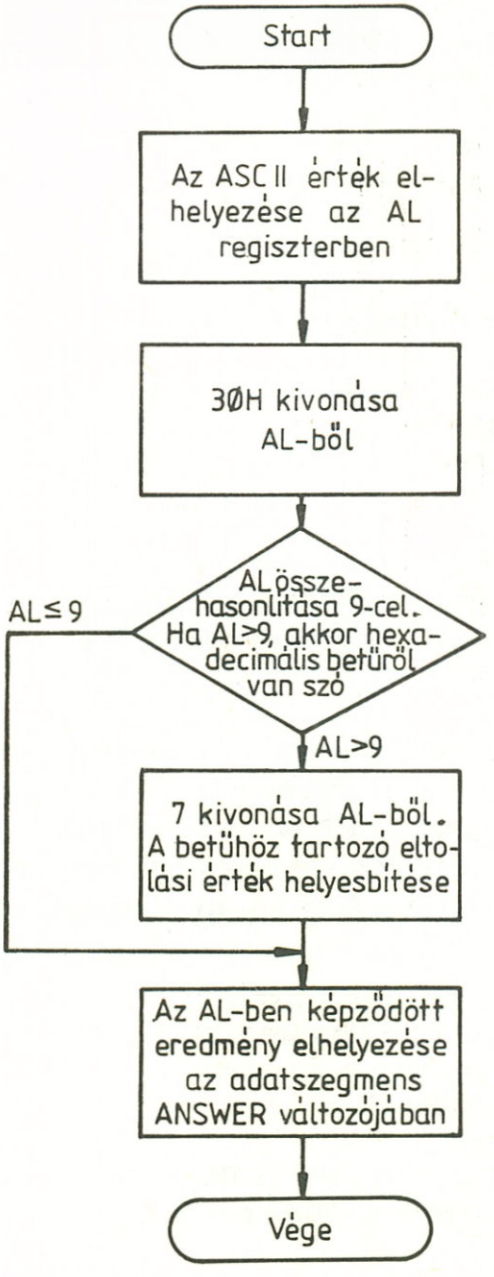
5.4 ASCII – HEXADECIMÁLIS ÁTVÁLTÁS

Az ASCII – hexadecimális átváltáshoz kikeresési táblázat helyett matematikai konverziót használunk. Majdnem minden információ, amit a billentyűzetről beolvastatunk vagy monitorra, ill. nyomtatóra küldünk, ASCII kódok formájában történik. Például ha az 5-ös számot nyomjuk le a billentyűzeten, ennek ASCII kódja, vagyis 35H kerül az AL regiszterbe. Ha számokat akarunk a képernyőn megjeleníteni, akkor az ASCII kódot, nem pedig a hexadecimális értéket kell megadnunk. Az 5–1. táblázat ezt az egyszerű konverziót mutatja be.

Az ASCII számok hexadecimális átváltásához minden számból 30H-t kell kivonni, a 41H és 46H közötti számok esetében pedig 37H-t. Ne feledkezzünk meg arról, hogy kivonásnál mindig a hexadecimális aritmetikát használjuk. A 5–16. ábrán az átváltáshoz szükséges programozási lépések láthatók. Felhívjuk a figyelmet arra, hogy többféle ASCII kód létezik, ez a példa azonban csak a numerikus átváltást tartalmazza hibaellenőrzés nélkül. Más szavakkal fogal-

ASCII	Hexadecimális
30H	0H
31H	1H
32H	2H
33H	3H
34H	4H
35H	5H
36H	6H
37H	7H
38H	8H
39H	9H
40H	0AH
41H	0BH
42H	0CH
43H	0DH
44H	0EH
45H	0FH
46H	

5–1. táblázat ASCII-ben kifejezett számok hexadecimális megfelelői



5–16. ábra ASCII számok átváltása hexadecimális formátumba

mazva ez azt jelenti, hogy a programozónak a listában felsorolt ASCII kódokat kell szerepeltetnie.

Ennyi bevezető után lássuk magát a programot!

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;ASCII-BEN ABRAZOLT SZAMOK ATVALTASA HEXABA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
ASCII   DB      42H             ; A B HEXA SZAM ASCII KODJA
ANSWER  DB      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;ASCII-HEXA ATVALTAS HIBAELLENORZES NELKUL
;A PROGRAM A 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F BEMENETEK
;VALAMELYIKET TETELEZI FEL
        MOV     AL,ASCII        ; ASCII SZAM ELHELYEZESE AL-BEN
        SUB     AL,30H          ; A SZAM A 0-9 TARTOMANYBA ESIK?
        CMP     AL,9            ; HA NEM AKKOR, 9-NEL NAGYOBB,
        JG      LETTER          ; TEHAT BETU
        JMP     END             ; KILEPES
LETTER: SUB     AL,07H          ; A BETUKNEL MEG 7-ET LE KELL VONNI
END:    MOV     ANSWER,AL       ; AZ EREDMENY AZ ANSWER-BE KERUL
;A KONVERZIOS FELADAT VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

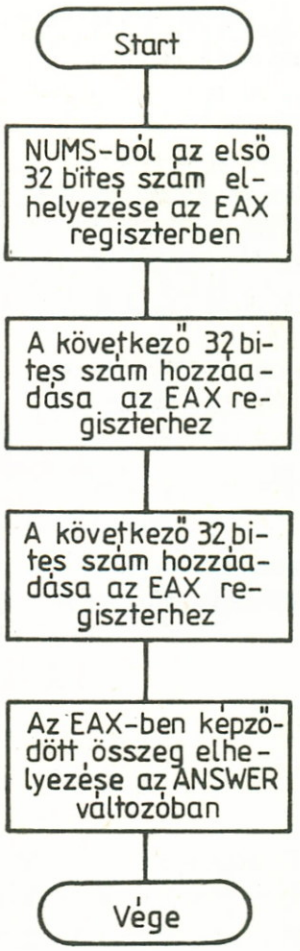
A programszegmens elején az ASCII számból azonnal kivonunk 30H-t, mivel azt feltételezzük, hogy az input a 30H–46H tartományba esik. Ha a kivonás eredménye a 0H–9H tartományba eső szám, akkor a program az END cimkére ugrik és befejeződik. Ha a CMP utasítás 09H-nál nagyobb értéket talál, akkor a számjegy egy betű, tehát ebből még 07H-t kivonunk (összesen tehát 37H-t), mivel a betűt a A–F hexadecimális számjegyek tartományába kívánjuk elhelyezni.

```
        MOV     AL,ASCII        ; ASCII SZAM ELHELYEZESE AL-BEN
        SUB     AL,30H          ; A SZAM A 0-9 TARTOMANYBA ESIK?
        CMP     AL,9            ; HA NEM, AKKOR 9-NEL NAGYOBB,
        JG      LETTER          ; TEHAT BETU
        JMP     END             ; KILEPES
LETTER: SUB     AL,07H          ; A BETUKNEL MEG 7-ET LE KELL VONNI
END:    MOV     ANSWER,AL       ; AZ EREDMENY AZ ANSWER-BE KERUL
```

5.5 EGYSZERŰ 32 BITES ARITMETIKA A 80386-OS MIKROPROCESSZORRAL

A megelőző programok egyikénél a duplapontos aritmetika felhasználásával 32 bites numerikus pontosságot állítottunk elő. A 8088/80286-os számítógépcsalád regiszterei 16 bitesek, ami azt jelenti, hogy a közvetlenül tárolható egész szám nagysága maximálisan 0FFFFH (decimálisan 65535) lehet. A 80386-os processzor általános regiszterei – EAX, EBC, ECX, EDX – már 32 bitesek, így maximálisan FFFFFFFFH (decimálisan 4 294 967 295) szám tárolására képesek. A 80386-os rendkívül gyors és hatékony mikroprocesszor, részben a megnövelt regisztерméret miatt. A most következő két példa azt mutatja, miként lehet nagyméretű számokkal műveleteket végrehajtani a többszörös pontosságú aritmetika eszközei nélkül.

Az 5–17. ábrán a 32 bites számok összeadásához szükséges programozási lépéseket követhetjük nyomon.



5–17. ábra A 80386-os összeadás programozási lépései

Ezt a programot az 5–4. ábrán látható programmal összehasonlítva – még egy ilyen egyszerű példánál is – látható, hogy a programozás sokkal tömörebbé vált. A 32 bites összeadás programja a következő:

```
; 80386-OS SZAMITOGEPRE KESZITETT PROGRAM
; 32 BITES OSSZEADASA 80386-OS MIKROPROCESSZORRAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUMS    DD  12345678H,9ABCDEF0H,23H,10000000H,0CADH
ANSWER  DD      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;A NUMS VALTOZOBAN TAROLT ELSO HAROM SZAM OSSZEADASA
        LEA     BX,NUMS         ; NUMS HELYZETENEK BETOLTESE
        MOV     EAX,[BX]        ; AZ ELSO SZAM ELHELYEZESE EAX-BEN
        ADD     EAX,[BX]+4      ; A KOVETKEZO SZAM HOZZADASA EAX-HEZ
        ADD     EAX,[BX]+8      ; A FENTI MUVELET MEGEGYSZER
        MOV     ANSWER,EAX      ; AZ EREDMENY ELHELYEZESE ANSWER-BEN

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

Ez a program is tartogat néhány újdonságot.
Először az adatszegmenst vesszük közelebbről szemügyre:

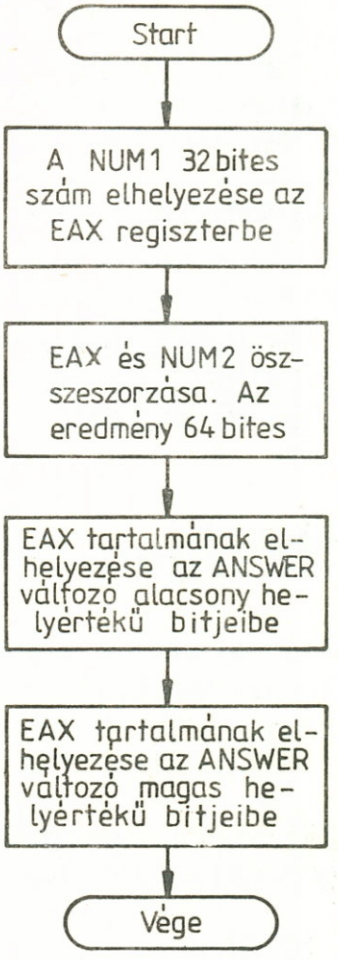
```
MYDATA  SEGMENT PARA 'DATA'
NUMS    DD  12345678H,9ABCDEF0H,23H,10000000H,0CADH
ANSWER  DD      ?
MYDATA  ENDS
```

Az összeadandó 32 bites számok mindegyikét duplaszónak definiáltuk. Az ANSWER elnevezésü változót ebben a példában 32 bites pontosságra korlátoztuk, és ugyancsak duplaszónak definiáltuk. (Az előbbi példáknál ezeket a számokat a méretük miatt nem tudtuk volna egyetlen regiszterbe közvetelnül betölteni.) De lássuk mit tehetünk most!

```
MOV     EAX,[BX]        ; AZ ELSO SZAM ELHELYEZESE EAX-BEN
ADD     EAX,[BX]+4      ; A KOVETKEZO SZAM HOZZADASA EAX-HEZ
ADD     EAX,[BX]+8      ; A FENTI MUVELET MEGEGYSZER
MOV     ANSWER,EAX      ; AZ EREDMENY ELHELYEZESE ANSWER-BEN
```

Az első szám (12345678H) a MOV utasítás segítségével közvetlenül az EAX regiszterbe kerül. A második számot az EAX tartalmához adjuk hozzá. Mivel duplaszavakról van szó, 4 byte-os eltolást kell használnunk. A harmadik számot az előzőhöz hasonlóan adjuk hozzá EAX-hez. Vegyük észre, hogy EAX 32 bites teljes tartalmát (0BCF14238H vagy decimálisan

3 169 927 736) egysoros kóddal helyezi el az ANSWER változóban. Az 5–18. ábrán két 32 bites szám összeszorzásához szükséges programozási lépéseket láthatunk.



5–18. ábra Szorzás a 80386-os processzor felhasználásával

A 80386-os esetében a 64 bites eredmény az EDX:EAX regiszterekben összegyűjthető. A maximálisan kifejezhető szám ebben az esetben 0FFFFFFFFFFFFFFFH (decimálisan közelítőleg 1.8446744073709553E19). A 64 bites mennyiség egy négyesszónak (DQ) definiált regiszterben tárolható a PTR operátor felhasználásával. A következő listán két nagyméretű egész szám összeszorzását figyelhetjük meg.

```
;80386-OS GEPEKRE KESZITETT PROGRAM
;32 BITES SZAMOK OSSZESZORZASA
;AZ EREDMENY 32 BITES

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM1    DD      12345678H
NUM2    DD      9ABCDEF0H
ANSWER  DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
```

```
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;KET 32 BITES SZAM OSSZESZORZASA
        MOV     EAX,NUM1        ; AZ ELSO SZAM BETOLTESE EAX-BE
        MUL     EAX,NUM2        ; EAX OSSZESZORZASA A KOV. SZAMMAL
        MOV     DWORD PTR ANSWER,EAX ; AZ EREDMENY ALSO 32 BITJE
                                ;EAX-BE KERUL
        MOV     DWORD PTR ANSWER+4,EDX ; AZ EREDMENY FELSO 32 BITJE
                                ;EDX-BE KERUL
;A SZORZASI FELADAT VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

Elsőként az adatszegmenssel foglalkozunk.

```
MYDATA  SEGMENT PARA 'DATA'
NUM1    DD      12345678H
NUM2    DD      9ABCDEF0H
ANDWER  DQ      ?
MYDATA  ENDS
```

Ebben a példában NUM1-et és NUM2-öt duplaszónak definiáltuk, mivel mindegyik 32 bites. Az eredményt egy 64 bites négyesszóban tároljuk. A programkód meglehetősen egyszerű:

```
MOV     EAX,NUM1        ; AZ ELSO SZAM BETOLTESE EAX-BE
MUL     EAX,NUM2        ; EAX OSSZESZORZASA A KOV. SZAMMAL
MOV     DWORD PTR ANSWER,EAX ; AZ EREDMENY ALSO 32 BITJE
                        ;EAX-BE KERUL
MOV     DWORD PTR ANSWER,EDX+4 ; AZ EREDMENY FELSO 32 BITJE
                        ;EDX-BE KERUL
```

Az első szám (NUM1) a 32 bites EAX regiszterbe kerül, amelyet a NUM2 változó tartalmával szorzunk össze. Az eredményt az EDX és EAX regiszterekben tároljuk. (A felső 32 bit EDX-ben, az alsó 32 bit EAX-ban van.) A DWORD PTR utasítás használatával gondoskodunk az egyes regiszterek 32 bites tartalmának tárolásáról a négyesszónak definiált ANSWER változóban.

5.6 BIOS ÉS DOS MEGSZAKÍTÁSOK HASZNÁLATA

Ha a mikroprocesszort a számítógép szívének, akkor a BIOS és DOS rutinokat a számítógép eszének nevezhetjük. Minden számítógépben már a gyártás során elhelyeznek bizonyos mennyiségű információt, amit a hardver ROM-ban (csak olvasható memória) tárolnak. A legtöbb BIOS (Basic Input/Output System = alap be/kiviteli rendszer) rutin is itt található. A ROM-ban tárolt információ állandó.

A DOS (Disk Operating System = lemez-operációs rendszer) rutinokra úgy kell gondolnunk, mint egy memóriára, amit csak később kapcsolunk a géphez. Valójában ez egy olyan memória, ami a számítógép bekapcsolásakor vagy a DOS-szal való behúzáskor kapcsolódik a géphez. Lényeges tulajdonsága, hogy bármikor aktualizálható és a DOS legújabb változataival is feltölthető. A DOS műveletek a RAM-ban (véletlen elérésű memória), vagy lemezen találhatók mindaddig, amíg nincs szükségünk rájuk. A ROM és a RAM együttesen vezérlik a számítógép működését. Az assembly nyelv segítségével lehetővé válik a memóriában tárolt néhány rutin felhasználása, amelyekkel a programjaink futási ideje jelentősen csökkenthető, de maga a program is hatékonyabbá válik. Minden gépnek (8088-as, 80286-os és 80386-os) saját BIOS és DOS utasításkészlete van. Az utasítások használatának leírását az IBM Műszaki Kézikönyve tartalmazza. A továbbiakban bemutatásra kerülő programok csak azokon az IBM számítógépeken futnak majd, amelyek a megfelelő tartozékokkal is rendelkeznek. A legtöbb program az IBM kompatibilis gépeken változtatás nélkül fut. (Bizonyos változtatások a kompatibilitás mértékétől függően szükségesek lehetnek.)

Képernyőtörlés BIOS megszakítással

A programozás során gyakran szükséges a bevitel és a kivitel megjelenítése a képernyőn. Lehet, hogy csak egy egyszerű képernyőtörlési műveletre van szükségünk, de az is előfordulhat, hogy a képernyő színét vagy egyéb jellemzőit akarjuk megváltoztatni. A képernyő kezelése a 10H BIOS megszakítással történik. Az 5–2. táblázatban a 10H BIOS megszakítással elérhető programozási variációkat soroltuk fel.

Figyelmeztetés: a megszakító rutin egy hexadecimális számmal hívható és INT 10H alakban kell szerepelnie a programban. A megszakításon belül a művelet a regiszterekben előzetesen beállított paraméterekkel határozható meg.

Például: MOV AH,13. A megszakítás segítségével az IBM kódokat a saját programunkban használhatjuk.

A monitor képernyőjét elég gyakran le kell törölni. A feladat megoldására két lehetőséget mutatunk be. Az egyiknél a 10H BIOS megszakítást alkalmazzuk (ami a képernyőt görgetéssel törli), a másiknál pedig közvetlenül a monitor portjára írunk. Először lássuk a 10H megszakítással történő képernyőtörlés programját.

```
;8088/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;KEPERNYOTORLES BIOS MEGSZAKITAS FELHASZNALASAVAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE

;A SZINES KEPERNYO TORLESENEK AKTUALIS KODJA
        MOV     CX,0000         ; A BAL FELSO SAROK (SOR,OSZL.) POZ.
        MOV     DX,2479H        ; A JOBB FELSO SAROK (SOR,OSZL.) POZ.
        MOV     BH,07           ; NORMAL ATTRIBUTUM
        MOV     AH,06           ; AZ AKTIV OLDAL FELFELE GORGETESE
```

```
        MOV     AL,00           ; A TELJES ABLAK GORGERESE
        INT     10H             ; A BIOS MEGSZAKITAS MEGHIVASA
; A RUTIN VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A programozó ebben az esetben nem látja az IBM kódot. (A BIOS rutinok listája az IBM Műszaki Kézikönyvében megtalálható.) A megszakítás használatához azonban nem szükséges a rutin kódját ismernünk. A következő programszegmensben a paramétereket az 5–2. táblázat szerint állítottuk be, és csak ezt követően hívtuk a megszakítást.

```
MOV     CX,0000         ; A BAL FELSO SAROK (SOR,OSZL.) POZ.
MOV     DX,2479H        ; A JOBB FELSO SAROK (SOR,OSZL.) POZ.
MOV     BH,07           ; NORMAL ATTRIBUTUM
MOV     AH,06           ; AZ AKTIV OLDAL FELFELE GORGETESE
MOV     AL,00           ; A TELJES ABLAK GORGERESE
INT     10H             ; A BIOS MEGSZAKITAS MEGHIVASA
```

Az „Aktív oldal felfelé görgetése" műveletet írjuk elő, ha az AH regiszterbe 6-ot töltünk. AL zérus tartalma miatt a művelet a teljes képernyőre vonatkozik. Amennyiben BH értéke 7, az attribútum normál állapotú lesz. (A későbbiekben az attribútumokat részletesen tárgyaljuk majd.) A CX és DX regiszterekkel az ablakméretet tudjuk definiálni. CH és CL az ablak bal felső, míg DH és DL az ablak jobb alsó sarkának pozícióját adja meg. Jelen esetben CX-et 0-ra állítottuk be, amelynek következtében CH és CL értéke egyaránt zérus. Ezzel a képernyő bal felső sarkát írtuk elő. DX-nek a 2479H értéket adtuk, tehát DH 24H-t, míg DL 79H-t tartalmaz. Ez a két koordináta a képernyő jobb alsó sarkát írja elő. A beállítások után már csak a megszakítást kell meghívnunk, aminek hatására a képernyőt letöröljük. Amint a programból kilépünk a DOS-ba, a képernyőn megjelenik a kurzor is. A másik módszer a képernyőmemória közvetlen törlése.

5–2. táblázat Képernyővezérlés BIOS 10H megszakítással

Szintaxis: INT 10H (A paramétereket a kívánt műveleteknek megfelelően kell beállítani.)

AH értéke	Funkció	Bemenet	Kimenet
	A CTR interface vezérlése		
AH = 0	Kijelzési üzemmód beállítása	AL = 0	40 * 25 fekete/fehér
		AL = 1	40 * 25 színes
		AL = 2	80 * 25 fekete/fehér
		AL = 3	80 * 25 színes
		AL = 4	320 * 200 grafkus színes
		AL = 5	320 * 200 grafikus fekete/fehér
		AL = 6	640 * 200 grafikus fekete/fehér
		AL = 10	640 * 350 grafikus E.G.A
AH = 1	Kurzortípus beállítása	CH =	A kurzor kezdősora (bitek 0-tól 4-ig)
		CL =	A kurzor zárósora (bitek 0-tól 4-ig)
AH = 2	Kurzorpozíció beállítása	DH =	Sor
		DL =	Oszlop
		BH =	A kijelzés lapszáma
AH = 3	Kurzorpozíció beolvasása (Az értékek a programfutástól függnek)	DH =	Sor
		DL =	Oszlop
		CH =	Kurzormód
		CL =	Kurzormód
		BH =	A kijelzés lapszáma
AH = 4	A fényceruza pozíciójának bekérése (Az értékek a programfutástól függnek)	AL = 0	Bekapcsolva/kikapcsolva
		AH = 1	Érvényes válaszok
		DH =	Sor
		DL =	Oszlop
		CH =	Grafikus sor (0–199)
		BX =	Grafikus oszlop (0/319/639)
AH = 5	Az aktív kijelzett oldal beállítása	AL =	Az új oldal értéke Üzemmód: 0 és 1 (0–7) Üzemmód: 2 és 3 (0–3)
AH = 6	Az aktív oldal felfelé görgetése	AL =	A sorok száma: 0 teljes képernyő esetén
		CH =	Sor (bal felső sarok)
		CL =	Oszlop (bal felső sarok)
		DH =	Sor (jobb felső sarok)
		DL =	Oszlop (jobb alsó sarok)
		BH =	A felhasználandó attributum
AH = 7	Az aktív oldal lefelé görgetése	AL =	A sorok száma; 0 teljes képernyő esetén
		CH =	Sor (bal felső sarok)
		CL =	Oszlop (bal felső sarok)
		DH =	Sor (jobb felső sarok)
		DL =	Oszlop (jobb alsó sarok)
		BH =	A felhasználandó attributum

AH értéke	Funkció	Bemenet	Kimenet
	Kezelőkarakterek		
AH = 8	Az attributum/karakter beolvasása az aktuális kurzorpozícióba	BH = AL = AH =	A képernyőoldal A beolvasott karakter A karakterek attributuma
AH = 9	Az attributum/karakter kiírása az aktuális kurzorpozícióba	BH = CX = AL = BL =	A képernyőoldal A kiírandó karakterek száma A kiírandó karakter A karakter attributuma
AH = 10	Karakter kiírása az aktuális kurzorpozícióba	BH = CX = AL =	A képernyőoldal A kiírandó karakterek száma a kiírandó karakter
	Grafikus interface		
AH = 11	Színpaletta kiválasztása	BH = BL =	Palettaazonosító (0/127) A palettaazonosítójára vonatkozó szín 0 – Háttér (0–15) 1 – Paletta 0 – zöld (1), piros (2), sárga (3) 1 – kék (1), bíbor (2), fehér (3)
AH = 12	Pontrajzolás a képernyőre	DX = CX = AL =	Sor (0–199) Oszlop (0–319/639) A pont színe
AH = 13	A pontra vonatkozó információk beolvasása	DX = CX = AL =	Sor (0–199) Oszlop (0–319/639) A pont értéke
	ASCII teletype-kivitel		
AH = 14	Írás aktív oldalra	AL = BL =	A kiírandó karakter Az előtér színe
AH = 15	A visszakapott vidóállapot	AL = AH = BH =	Az éppen érvényes üzemmód A képernyő oszlopainak száma Az éppen kijelzett oldal
AH = 16	Foglalt		
AH = 17	Foglalt		
AH = 18	Foglalt		
AH = 19	Karakterlánc kiírása	ES:BP = CX = DX = BH = AL = 0 AL = 1 AL = 2 AL = 3	Rámutatás a karakterláncra A karakterlánc hossza A kezdő kurzorpozíció Oldalszám BL = attributum **(char,char,char....char)** A kurzor mozdulatlan BL = attributum **(char,char, char....char)** A kurzor mozog **(char,attr,char,attr....)** A kurzor mozdulatlan **(char,attr,char,attr....)** A kurzor mozog

Megjegyzés: További részletek az IBM AT számítógép Műszaki Kézikönyvében találhatók.

A következő program szóközöket ír a képernyőkezelő kártya memóriájába.

```
;RGB MONITORRAL ELLATOTT 8088/80386-OS SZAMITOGEPEKRE KESZITETT
;PROGRAM
;SZINES KEPERNYO TORLESE BIOS MEGSZAKITAS NELKUL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE

;A KEPERNYO TORLESET VEGZO AKTUALIS KOD
        MOV     AX,0B800H       ; BELEPESI PONT A SZINES RAM-BA
        MOV     ES,AX           ; A BELEPESI PONT ELHELYEZESE ES-BEN
        MOV     DI,00H          ; KEZDOCIM
        MOV     AL,00H          ; A KIIRANDO KARAKTER
        MOV     AH,07H          ; NORMAL ATTRIBUTUM BEALLITASA
        MOV     CX,7D0H         ; IRAS 2000-SZER
REP     STOSW                   ; A MUVELET KONKRET VEGREHAJTASA
;A RUTIN VEGE

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE


        MOV     AX,0B800H       ; BELEPESI PONT A SZINES RAM-BA
        MOV     ES,AX           ; A BELEPESI PONT ELHELYEZESE ES-BEN
        MOV     DI,00H          ; KEZDOCIM
        MOV     AL,00H          ; A KIIRANDO KARAKTER
        MOV     AH,07H          ; NORMAL ATTRIBUTUM BEALLITASA
        MOV     CX,7D0H         ; IRAS 2000-SZER
REP     STOSW                   ; A MUVELET KONKRET VEGREHAJTASA
```

Ez a program csak a szabványos színes kártyával használható. A programszegmens számos jellegzetességére érdemes felfigyelnünk. A színes monitorkártyának saját memóriája van, aminek eléréséhez ES-be a 0B800H értéket kell elhelyezni. (Monokrom monitor esetében ez az érték 0B000H.) Az ES regisztert közvetlenül nem lehet betölteni, ezért a fenti értéket AX-en keresztül közvetetten helyezzük el ES-be. A DI regiszterben az adott memória eltolási értékét kell szerepeltetni.

Az AH regiszterrel a képernyő attribútumát írjuk elő. A 7-es érték a normál helyzetet jelenti (fehér, nem villogó, normál intenzitású). AL a kiírandó karaktert tartalmazza, ami jelen esetben egy szóköz. A képernyő törlését úgy valósítjuk meg, hogy a 80 * 25 karakteres monitor minden egyes képernyőpozíciójába egy szóközt írunk. Az írási művelet a REP STOSW (karakterlánc ismételt tárolása) utasítással, ami az AX regiszter tartalmát minden egyes memóriahelyre elhelyezi, gyorsan megvalósítható.

Fejléc készítése BIOS megszakítással

A 10H megszakítás használatával a képernyő irányítását a kezünkbe vesszük. Különösebb bevezetés nélkül lássuk mindjárt magát a programot, ami a képernyőn egy fejlécet helyez el.

```
;RGB MONITORRAL ELLATOTT 8088/80386-OS SZAMITOGEPEKRE
;KESZITETT PROGRRAM
;INT 10H FELHASZNALASA KARAKTERLANCOK KIIRATASAHOZ

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
BACK    DB      2000 DUP(' ')
LOGO    DB      13 DUP (' '),'                                              ',14 DUP (' ')
        DB      13 DUP (' '),'                                              ',14 DUP (' ')
        DB      13 DUP (' '),'        Programozas Assembler nyelven         ',14 DUP (' ')
        DB      13 DUP (' '),'                                              ',14 DUP (' ')
        DB      13 DUP (' '),'        a 80286/80386-os processzorral        ',14 DUP (' ')
        DB      13 DUP (' '),'                                              ',14 DUP (' ')
        DB      13 DUP (' '),'                                              ',14 DUP (' ')
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,ES:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     ES,AX           ; AX TARTALMANAK ATADASA ES-BE

;A PROGRAMSZEGMENS 80*25 SZOKOZ KEPERNYORE IRATASAVAL TORLI
;A KEPERNYOT. A BL-BEN ELHELYEZETT 40H HATASARA A KEPERNYO
;PIROSRA VALT.
        LEA     BP,BACK         ;BACK HELYENEK BEKERESE
        MOV     DX,0000         ;KURZORPOZICIONALAS A BAL FELSO SAROKBA
        MOV     AH,19           ;A KARAKTERLANC ATTRIBUTUMANAK BEALLITASA
        MOV     AL,1            ;A KARAKTER MEGJELENITESE ES KURZORMOZGATAS
        MOV     BL,01000000B    ;PIROS HATTER
        MOV     CX,07D0H        ;SZOKOZIRAS 2000-SZER
        INT     10H             ;A MEGSZAKITAS HIVASA
;A PROGRAMSZEGMENS 9 SORRAL LEFELE MOZGATJA A KURZORT,
;ES KINYOMTATJA A LOGO CIMKEJU FELIRATOT. BL-BEN 4EH-T
;HELYEZTUNK EL, AMIVEL PIROS HATTERET ES SARGA KIEMELEST
;BIZTOSITUNK.
        LEA     BP,LOGO         ;LOGO HELYENEK BEKERESE
        MOV     DH,09           ;AZ UJ KURZORPOZICIO BEALLITASA
        MOV     AH,19           ;A KARAKTERLANC ATTRIBUTUMANK BEALLITASA
        MOV     AL,1            ;A KARKATER MEGJELENITESE ES KURZORMOZGATAS
        MOV     BL,01001110B    ;PIROS HATTER ES SARGA KIEMELES
        MOV     CX,196H         ;A LOGO KARAKTEREINEK SZAMA
        INT     10H             ;A MEGSZAKITAS HIVASA

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A fejléc piros háttérben sárga felirattal jelenik meg.

Ha az üzenetet megváltoztatjuk, nagyon körültekintően járjunk el a szöveg elhelyezésekor. Az assembly nyelv engesztelhetetlen a hibákkal szemben. Az előző listából már bizonyára felfigyeltünk arra, hogy a program két részre tagolódik. Az első rész a képernyő törléséről gondoskodik, míg a második rész a fejléc megjelenítését biztosítja.

A képernyő törlését a következő programrészlet végzi el:

```
LEA     BP,BACK         ;BACK HELYENEK BEKERESE
MOV     DX,0000         ;KURZORPOZICIONALAS A BAL FELSO SAROKBA
MOV     AH,19           ;A KARAKTERLANC ATTRIBUTUMANAK BEALLITASA
MOV     AL,1            ;A KARAKTER MEGJELENITESE ES KURZORMOZGATAS
MOV     BL,01000000B    ;PIROS HATTER
MOV     CX,07D0H        ;SZOKOZIRAS 2000-SZER
INT     10H             ;A MEGSZAKITAS HIVASA
```

A 10H megszakítás – az AT (80286) számítógép BIOS rutinjainál – AH 19-re történő beállításával a karakterláncok kiíratását tudjuk megvalósítani.

A program ezt az opciót kétszer használja: a képernyő letörléséhez és a fejléc kiírásához. A fejlécet az ES regiszterben tároltuk, a BACK változó effektív címét pedig a BP regiszterbe töltöttük, ahogy a megszakítás ezt megkívánja. DX-ben írtuk elő a kurzor kezdőpozícióját (bal felső sarok). Ha AL-t 1-re állítjuk be, a megszakítás kinyomtat egy karaktert (jelen esetben egy szóközt) a képernyőre, és a kurzort a következő pozícióba állítja. A BL regiszterrel a képernyő attribútumát tudjuk befolyásolni. A BL 8 bites regiszter a következők szerint osztható fel:

BL

7 6 5 4	3 2 1 0
(Magasbitek)	(Alacsony bitek)
(Háttér)	(Előtér)

A háttér és az előtér (felirat) színei az 5–3. táblázatban felsorolt palettából választhatók ki. Jelen esetben a háttér színe piros (4), az előtér színe pedig 0 (nincs színmegjelölés). A programban ezek az értékek bináris alakban szerepelnek. (Tehát a 40H-ból 01000000B lesz.) Ha sárga hátteret szeretnénk, akkor 0E0H-t, azaz 1110000B-t kellene beírnunk. Ez a kódrészlet 2000 szóközt (piros háttérrel) ír a képernyőre, a második kódrészlet pedig a fejlécet jeleníti meg.

A BL regiszter felső négy bitjének tehát a háttérre, míg alsó négy bitnek az előtérre vonatkozó információt kell tartalmaznia. Jelen esetben az előtér színe sárga, a 09-es kurzorpozíciónál kezdődik és 560 karaktert tartalmaz.

```
LEA     BP,LOGO         ;LOGO HELYENEK BEKERESE
MOV     DH,05           ;AZ UJ KURZORPOZICIO BEALLITASA
MOV     AH,19           ;A KARAKTERLANC ATTRIBUTUMANAK BEALLITASA
MOV     AL,1            ;A KARKATER MEGJELENITESE ES KURZORMOZGATAS
MOV     BL,01001110B    ;PIROS HATTER ES SARGA KIEMELES
MOV     CX,196H         ;A LOGO KARAKTEREINEK SZAMA
INT     10H             ;A MEGSZAKITAS HIVASA
```

0 – nincs színmegjelölés	1 – kék	2 – zöld
3 – ciklámen	4 – piros	5 – bíbor
6 – barna	7 – fehér	8 – szürke
9 – világoskék	10 – világoszöld	11 – világosciklámen
12 – halványpiros	13 – halványbíbor	14 – sárga
15 – világosszürke		

5–3. táblázat Az IBM PC egy AT színpalettája

Program adatainak megjelenítése a képernyőn DOS megszakítással

A 21H DOS megszakítás függvényhívó megszakítás, amelynek specifikációját és a használati előírásait az 5–4. táblázatban foglaltuk össze. A következő programban ezt a megszakítást a hatékonyabb programozási technika bemutatására használjuk ki. Ez a program lehetővé teszi, hogy a kiválasztott változó tartalmát a nyomkövető program alkalmazása nélkül megtudjuk. Sok időt takaríthatunk meg magunknak, ha a különböző utasítások kipróbálása során az eredményt rögtön megpillanthatjuk a képernyőn.

AH értéke	Funkció	Bemenet	Kimenet
AH = 1	Várakozik és kijelzi a billentyűzet karakterét CTRL–BREAK-ellenőrzéssel	=	AL – a beléptetett karakter
AH = 2	Kijelzi a karaktert CTRL–BREAK-ellenőrzéssel	DL =	A megjelenítendő karakter
AH = 3	Aszinkron karakterbevitel (soros vonal)		AL – beléptetett karakter
AH = 4	Aszinkron karakterkivitel (soros vonal)	DL =	A megjelenítendő karakter
AH = 5	Karakterkiírás nyomtatóra	DL =	A kiírandó karakter
AH = 6	Karakterbeolvasás a billentyűzetről	DL = 0FFH	A beléptetett karakter; 0 ha nincs
AH = 7	Várakozás a billentyűzetkarakterre (nincs kijelzés)		AL – a beléptetett karakter
AH = 8	Várakozás a billentyűzetkarakterre (nincs kijelzés – CTRL–BREAK-ellenőrzés)		AL – a beléptetett karakter
AH = 9	Karakterlánc-kijelzés	DX:DX =	A karakterlánc címe. A $ elválasztó karakterrel kell végződnie
AH = A	Pufferelt bevitel billentyűzetről	DS:DX =	A puffer címe. Első byte = méret, második byte = a beolvasott karakterek száma

AH értéke	Funkció	Bemenet	Kimenet
AH = B	A billentyűzet státuszának ellenőrzése		AL – nincs karakter = 0FFH, karakter = 0
AH = C	A billentyűzet pufferének törlése és funkcióhívás	AL =	1, 6, 7, 8, 0A – a funkció száma
AH = D	Alapértelmezés szerinti lemezmeghajtó alaphelyzetbe állítás	Nincs	Nincs
AH = E	Alapértelmezés szerinti lemezmeghajtó kiválasztás	DL =	AL – a meghajtók száma 0 = A meghajtó 1 = B meghajtó stb.
AH = 19	Meghajtó kód (alapértelmezés szerinti)		AL – 0 = A meghajtó 1 = B meghajtó stb.
AH = 25	Megszakítási vektor (beállítás)	DS:DX = AL =	A megszakítási vektor címe A megszakítás száma
AH = 2A	Dátum (beolvasás)		CX – Év (80-tól 99-ig) DH – Hónap (1-től 12-ig) DL – Nap (1-től 31-ig)
AH = 2A	Dátum (beállítás)	CX:DX	Ugyanaz, mint fenn AL – 0 ha érvényes. Kikapcsolva, ha nem érvényes
AH = 2C	Idő (beolvasás)		CH – Óra (0-tól 23-ig) CL – Perc (0-tól 59-ig)
AH = 2D	Idő (beállítás)	CX:DX	Ugyanaz, mint fenn AL – 0, ha érvényes. Kikapcsolva, ha nem érvényes
AH = 2E	Ellenőrzési állapot (beállítás)	DL = AL =	0 0 = Ellenőrzés kikapcsolva 1 = Ellenőrzés bekapcsolva
AH = 35	Megszakítási cím (beolvasás)	AL = ES:BX	A megszakítás száma A vektorcímhez mutat
AH = 36	A rendelkezésre álló lemezterület	DL =	Meghajtó (0 – alapértelmezés, 1 – A, 2 – B stb.) AX – szektor/cluster (lemezblokk) (FFFF, ha érvénytelen) BX – a szabad clusterek száma CX – a szektoronkénti byte-ok száma DX – az összes cluster száma
AH = 39	Katalógus készítése	DS:DX =	A karakterlánc címe a katalógus részére
AH = 3A	A katalógus megszüntetése	DS:DX =	A katalógushoz tartozó karakterlánc címe
AH = 3B	A lemezkatalógus megváltoztatása	DS:DX =	Az új katalógus címe

AH értéke	Funkció	Bemenet	Kimenet
AH = 3C	File (létrehozás)	DS:DX =	A file-név címe
			AX – file-kezelés
		CX =	File-attributum
AH = 3D	File (nyitás)	DS:DX =	A file nevének címe
		AL =	0 – Nyitás olvasáshoz
			1 – Nyitás íráshoz
			2 – Nyitás íráshoz-olvasáshoz
		AX =	A file-kezelőt tartalmazza
AH = 3E	File (lezárás)	BX =	File-kezelő
AH = 3F	File vagy eszköz (olvasás)	BX =	File-kezelő
		CX =	A beolvasandó byte-ok száma
		DS:DX =	A puffer címe
			AX – a beolvasandó byte-ok száma
AH = 40	File vagy eszköz (írás)	BX =	File-kezelő
		CX =	A kiírásra kerülő byte-ok száma
		DS:DX =	A kiírásra kerülő adatok címe
			AX – A kiírásra kerülő byte-ok száma
AH = 41	File (törlés)	DS:DX	A file nevének címe
AH = 42	Írási/olvasási mutató átállítás	AL = 0	A mutató eltolási mértéke a file kezdetétől
			(CX:DX) byte-tal eltolt mutató
		AL = 1	Eltolás a jelenleg érvényes helyzethez képest
		AL = 2	Eltolás a file végétől visszafelé
AH = 43	File (attributum beállítása)	DS:DX =	A file nevének címe
			Ha AL = 0, az attributumot CX-ben kapjuk vissza.
			Ha AL = 1, a file az AX attributum szerint állítódik be
AH = 47	Az éppen érvényes lemezkatalógus	DL =	A meghajtó száma (0 – alapértelmezés)
			(1 – A meghajtó,
			2 – B meghajtó)
		DS:SI	A puffer címe
			DS:SI – A karakterlánc címét adja vissza
AH = 54	Ellenőrzési állapot	Nincs	AL – 0, az ellenőrzés bekapcsolva
			AL – 1, az ellenőrzés kikapcsolva

AH értéke	Funkció	Bemenet	Kimenet
AH = 56	File (átnevezés)	DS:DX =	A régi információt hordozó karakterlánc címe
		ES:DI =	Az új információt hordozó karakterlánc címe

5–4. táblázat A DOS 21H megszakítás specifikációja és használati előírásai

A programozási technika hasznosságának bemutatására egy aritmetikai példát oldunk meg:

```
MOV     AX,01234H       ;AZ ELSO OSSZEADANDO SZAM
ADD     AX,02299H       ;A MASODIK OSSZEADANDO SZAM
MOV     TMPNUM,AX       ;A7 EREDMENY EGY IDEIGLENES HELYRE
                        ;KERUL
```

Az összeadás eredményét (34CDH) a TMPNUM elnevezésű változóban tároljuk. Ha TMPNUM tartalmát a képernyőn akarjuk megjeleníteni, ASCII karakterláncba kell átalakítanunk. A következő programrészlet ezt valósítja meg:

```
        MOV     CX,04H          ;AZ ATALAKITANDO SZAMJAGYEK SZAMA
AGAIN:  MOV     AX,TMPNUM       ;A 16 BITES ADAT AX-BE KERUL
        AND     AX,0FH          ;CSAK AZ ALSO NEGY BITET TARTJUK MEG
        ADD     AL,30H          ;AZ ASCII KOD ELKESZITESE
        CMP     AL,39H          ;BETUROL VAN SZO?
        JL      ASCII           ;HA A KARAKTER SZAM, UGRAS ASCII-RE
        ADD     AL,07H          ;HA A KARAKTER BETU, KIIGAZITAS
ASCII:  MOV     SI,CX           ;AZ INDEX-BE AZ ATALAKITANDO
                                ;KARAKTEREK SZAMA KERUL
        MOV     TMPCHAR[SI],AL  ;A KONVERTALT SZAM ELMENTESE
        ROR     TMPNUM,1        ;A KOVETKEZO SZAMJEGY ROTALASA LSB-BE
        ROR     TMPNUM,1
        ROR     TMPNUM,1
        ROR     TMPNUM,1
        LOOP    AGAIN           ;HA CX>0, A KOV. SZAMJEGY ATALAKITASA
```

A TMPNUM egy 16 bites regiszter, ami négy hexadecimális számjegy tárolására képes. Az ASCII megfelelők kialakítása egyesével történik. Az első menetben a TMPNUM számot az AX regiszterbe helyezzük el, és a 0FH számmal végrehajtunk rajta egy AND műveletet. Ezáltal a legkisebb helyértékű számjegyet, a 0DH-t kapjuk meg. Ezután a számjegyet 30H-val egy OR műveletnek vetjük alá, hogy az ASCII formátumot előállítsuk, majd megvizsgáljuk, hogy számról vagy betűről van-e szó. Jelen esetben, mivel betűt alakítunk át, egy ugrást kell végrehajtanunk, hogy az AL regiszter tartalmához még 07H-t adhassunk hozzá. Az AL tehát 44H-t tartalmaz most, ami az ASCII D karakterének felel meg. Ezt a karaktert a TMPCHAR elnevezésű karakterlánc-változóban helyezzük el olyan eltolással, ami a pozíció értékével egyenlő (balról jobbra számolva a 4. hely) az eredeti számban. Ezt követően TMPNUM-ot 4 bittel rotáljuk – vagyis 1 számjeggyel –, így a következő számjegy kerül az alsó négy bitbe. Az eljárást addig ismételjük, amíg mind a négy számjegy konvertálása megtörténik.

A következő kódszegmens TMPCHAR megjelenítéséről gondoskodik:

```
LEA     DX,TMPCHAR      ;KARAKTERLANCOKAT MEGJELENITO RUTIN
MOV     AH,9            ;DOS PARAMETER
INT     21H             ;DOS MEGSZAKITAS
```

A DOS 21H típusú megszakítása az adatszegmens adatait mindaddig nyomtatja a képernyőre, amíg egy dollárjellel találkozik. Ha erről a jelzésről elfeledkezünk, az INT 21H folytatja a karakterek kinyomtatását mindaddig, amíg egy dollárjellel találkozik.

A billentyűzet karaktereinek beolvasása DOS megszakítással

Ha egy programmal párbeszédet kívánunk folytatni (interaktív rendszer), a feltett kérdésre valamilyen módon válaszolnunk kell (pl. az Y (igen) vagy N (nem) billentyűk lenyomásával). A következőkben egy olyan program listáját mutatjuk be, ami DOS megszakítás segítségével olvassa be a billentyűzetről származó információt.

```
;8088/80286-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;KARAKTER BEOLVASASA A BILLENTYUZETROL, ES KIJELZESE
;A TEST-BE MAS ERTEK KERUL AZ 'Y', ES MAS AZ 'N'
;LENYOMASARA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
TEST    DW      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;A BILLENTYUZET EGY KARAKTERENEK BEVITELE ES MEGJELENITESE
        MOV     AH,01H          ;PARAMETER A KARAKTER BEOLVASASAHOZ
        INT     21H             ;A KARAKTER MEGJELENITESE
        CMP     AL,'Y'          ;'Y'-ROL VAN SZO ?
        JNE     HERE            ;HA NEM, AKKOR UGRAS A HERE CIMKERE
        MOV     BX,9999H        ;HA IGEN, 9999H ELHELYEZESE BX-BE
        JMP     ENDO            ;A PROGRAM BEFEJEZESE
HERE:   CMP     AL,'N'          ;'N'-ROL VAN SZO ?
        JNE     ENDO            ;HA NEM, AKKOR UGRAS AZ ENDO CIMKERE
        MOV     BX,5555H        ;HA IGEN, 5555H ELHELYEZESE BX-BE

ENDO:   MOV     TEST,BX         ;BX TARTALMA TEST-BE KERUL
```

```
        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

Ez a program Y vagy N karakterek lenyomásától függően döntésre késztethető. (Vigyázat! Az y és n kisbetűkkel a program nem működik.) A programkódot külön is kiemeljük:

```
        MOV     AH,01H          ;PARAMETER A KARAKTER BEOLVASASAHOZ
        INT     21H             ;A KARAKTER MEGJELENITESE
        CMP     AL,'Y'          ;'Y'-ROL VAN SZO ?
        JNE     HERE            ;HA NEM, AKKOR UGRAS A HERE CIMKERE
        MOV     BX,9999H        ;HA IGEN, 9999H ELHELYEZESE BX-BE
        JMP     ENDO            ;A PROGRAM BEFEJEZESE
HERE:   CMP     AL,'N'          ;'N'-ROL VAN SZO ?
        JNE     ENDO            ;HA NEM, AKKOR UGRAS AZ ENDO CIMKERE
        MOV     BX,5555H        ;HA IGEN, 5555H ELHELYEZESE BX-BE

ENDO:   MOV     TEST,BX         ;BX TARTALMA TEST-BE KERUL
```

Az 5–4. táblázat a 21H DOS megszakítás használatához minden szükséges információt megad. Ha AH-t 1-re állítjuk be, és meghívjuk a megszakítást, a program egy billentyű lenyomására várakozik, majd pedig megjeleníti a karaktert a képernyőn, és a karakter ASCII értékét elhelyezi az AL regiszterben. Ha az Y-t nyomjuk le, a program a 9999H értéket a BX regiszterbe és a TEST elnevezésű változóba elhelyezi. Ha az N gombot nyomjuk le, a program az 5555H értéket tárolja. Bármely más gomb lenyomásakor a TEST-ben azt az értéket találjuk, ami a program elején BX értéke volt.

Karakterláncok beolvasása DOS megszakítással

Billentyűzetről a memóriába való karakterlánc-beolvasás esetén előzetesen bizonyos meggondolásokra van szükség, mivel most is ugyanazt a 21H DOS megszakítást fogjuk használni. Vizsgáljuk meg a következő adatszegmenst:

```
BUFF    DB      80 DUP (' '),'$';PUFFER A KARAKTERLANC SZAMARA
LINE    DB      0AH,0DH,'$'     ;KOCSIVISSZA ES SOREMELES KARAKTEREK
```

A BUFF elnevezésű változó 80 karakterből álló lánc elfogadásáról és tárolásáról gondoskodik. Vegyük észre, hogy a karakterlánchoz egy vezérlőkaraktert is hozzákapcsoltunk. A LINE változó két vezérlőkaraktert tartalmaz, amelyek közül az egyik a kocsivissza, a másik a soremelés. Ezek célja a kurzor megfelelő helyzetbe állítása.

A program listája a következő:

```
;8088/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;A BILLENTYUZETEN BEIRT KARAKTERLANC BEOLVASASA ES
;MEGJELNITESE A KEPERNYON

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
BUFF    DB      80 DUP (' '),'$';PUFFER A KARAKTERLANC SZAMARA
LINE    DB      0AH,0DH,'$'     ;KOCSIVISSZA ES SOREMELES KARAKTEREK
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE
        LEA     BX,BUFF         ;A PUFFER HELYZETE

;A BILLENTYUZET EGY KARAKTERENEK BEOLVASASA ES MEGJELENITESE
        MOV     AH,01H          ;A BEOLVASASHOZ SZUKSEGES PARAMETER
        MOV     CX,00H          ;A SZAMLALO BEALLITASA ZERUSRA
HERE:   INT     21H             ;EGY KARAKTER MEGJELENITESE
        CMP     AL,0DH          ;A KARAKTER A KOCSIVISSZA KARAKTER-E ?
        JE      NOMORE          ;HA IGEN, KILEPES A RUTINBOL
        MOV     BUFF[BX],AL     ;A KARAKTER ELHELYEZESE A PUFFERBA
        CMP     CX,79           ;BETOLTOTTUNK MAR 80 KARAKTERT ?
        JE      NOMORE          ;HA IGEN, KILEPES A RUTINBOL
        INC     CX              ;A BETUSZAMLALO NOVELESE
        INC     BX              ;A KOVETKEZO TAROLASI HELY BEALLITASA
        JMP     HERE            ;A KOVETKEZO KARAKTER BEKERESE
;A PUFFERBAN TAROLT KARAKTERLANC MEGJELENITESENEK RUTINJA
NOMORE:
        LEA     DX,LINE         ;KOCSIVISSZA ES SOREMELES MEGVALOSITASA
        MOV     AH,09           ;A KURZOR BEALLITASA A KOVETKEZO
                                ;SOR ELEJERE
        INT     21H             ;DOS MEGSZAKITAS
        LEA     DX,BUFF         ;A KARAKTERLANC MEGJELENITESE
        MOV     AH,09
        INT     21H             ;DOS MEGSZAKITAS

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VÉGE
```

A programban alkalmazott számláló 80 karakter beírását engedélyezi. A inputból a kocsivissza gomb (0DH) lenyomásával lehet kilépni. A karakterlánc 1-től 80-ig tetszőleges számú karaktert tartalmazhat. A kódszegmens egy részét külön kiemeljük:

```
        MOV     AH,01H          ;A BEOLVASASHOZ SZUKSEGES PARAMETER
        MOV     CX,00H          ;A SZAMLALO BEALLITASA ZERUSRA
HERE:   INT     21H             ;EGY KARAKTER MEGJELENITESE
        CMP     AL,0DH          ;A KARAKTER A KOCSIVISSZA KARAKTER-E ?
        JE      NOMORE          ;HA IGEN, KILEPES A RUTINBOL
        MOV     BUFF[BX],AL     ;A KARAKTER ELHELYEZESE A PUFFERBA
        CMP     CX,79           ;BETOLTOTTUNK MAR 80 KARAKTERT ?
        JE      NOMORE          ;HA IGEN, KILEPES A RUTINBOL
        INC     CX              ;A BETUSZAMLALO NOVELESE
        INC     BX              ;A KOVETKEZO TAROLASI HELY BEALLITASA
        JMP     HERE            ;A KOVETKEZO KARAKTER BEKERESE
```

Ebben a programszegmensben a billentyűzet karaktereinek befogadása hasonló, mint ahogy azt az előző példánál láttuk. AL tartalmát (az ASCII karakter) először a kocsivissza karakterrel (0DH) hasonlítjuk össze. Ha a karakter egy kocsivissza, akkor a JE utasítás kiléptet a ciklusból. Ellenkező esetben a karakter a BUFF változóban a BX által meghatározott helyre kerül. A következő sorban ellenőrizzük, hogy 80 karakter beírása megtörtént-e. Ha nem, akkor CX és BX értékét növeljük. CX-ben számláljuk a betűket és BX tartalmazza BUFF átmeneti tárolóban az eltolási értéket. A ciklus a 80 karakter beírásáig vagy a kocsivissza gomb lenyomásig ismétlődik. Most pedig azt a programszegmenst tanulmányozzuk, ami a karakterlán kiírásáról gondoskodik:

```
LEA     DX,LINE         ;KOCSIVISSZA ES SOREMELES MEGVALOSITASA
MOV     AH,09           ;A KURZOR BEALLITASA A KOVETKEZO
                        ;SOR ELEJERE
INT     21H             ;DOS MEGSZAKITAS
LEA     DX,BUFF         ;A KARAKTERLANC MEGJELENITESE
MOV     AH,09
INT     21H             ;DOS MEGSZAKITAS
```

A LINE változóval a soremelés és a kocsivissza karakterekre, míg a BUFF változóval a billentyűzetről beírt karakterláncra tudunk rámutatni. Ez a program a karakterláncot az éppen érvényes kurzorpozíciótól kezdve nyomtatja ki.

Idő- és dátumbeolvasás BIOS megszakítással

Az 1AH BIOS megszakítást az IBM AT számítógépeknél a 8088-hoz képest kibővítették. Lehetővé vált ui., hogy az idő és a dátum éppen érvényes értékeit a rendszer órájába beírjuk, ill. onnan kiolvassuk. Az 5–5. táblázat az 1AH BIOS megszakítással elérhető opciók listáját tartalmazza. Ezen kívül még a 80286-os riasztás funkcióját is beállíthatjuk vagy törölhetjük. A következő listán egy olyan 80286-os program látható, amivel a rendszer órájából kiolvashatjuk az időt és a dátumot.

```
; A 80286-OS IBM GEPRE KESZITETT PROGRAM
;IDO ES DATUMADATOK BEOLVASASA BIOS MEGSZAKITASSAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
TIME    DD      0H              ;IDOERTEKEK DUPLASZAVAS HELYFOGLALASA
```

```
DATE    DD      0H              ;DATUMERTEKEK DUPLASZAVAS HELYFOGLALASA
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ; KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ; AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ; A DS REGISZTER ELMENTESE
        SUB     AX,AX           ; AX TORLESE
        PUSH    AX              ; ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ; AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ; AX TARTALMANAK ATADASA DS-BE

;AZ IDOERTEK BEKERESE  BIOS MEGSZAKITASSAL
        MOV     AH,02H          ;AZ IDORE VONATKOZO PARAMETER
        INT     1AH             ;IDO/DATUM MEGSZAKITAS
        MOV     BYTE PTR TIME+3,DH  ;MP ERTEKEK ELHELYEZESE TIME-BAN
        MOV     BYTE PTR TIME+2,CL  ;PERC ERTEKEK ELHELYEZESE TIME-BAN
        MOV     BYTE PTR TIME+1,CH  ;ORA ERTEKEK ELHELYEZESE TIME-BAN

;A DATUM BEKERESE BIOS MEGSZAKITASSAL
        MOV     AH,04           ;A DATUMRA VONATKOZO PARAMETER
        INT     1AH             ;IDO/DATUM MEGSZAKITAS
        MOV     BYTE PTR DATE+3,DL  ;A NAP ELHELYEZESE DATE-BEN
        MOV     BYTE PTR DATE+2,DH  ;A HONAP ELHELYEZESE DATE-BEN
        MOV     BYTE PTR DATE+1,CL  ;AZ EV ELHELYEZESE DATE-BEN
        MOV     BYTE PTR DATE,CH    ;EVSZAZAD HOZZARENDELESE AZ EVHEZ

        RET                     ; A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ; A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ; A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ; A PROGRAM VEGE
```

A program két szegmensre tagolódik. Az idő beolvasását szolgáló szegmenst külön kiemeljük:

```
MOV     AH,02H              ;AZ IDORE VONATKOZO PARAMETER
INT     1AH                 ;IDO/DATUM MEGSZAKITAS
MOV     BYTE PTR TIME+3,DH  ;MP ERTEKEK ELHELYEZESE TIME-BAN
MOV     BYTE PTR TIME+2,CL  ;PERC ERTEKEK ELHELYEZESE TIME-BAN
MOV     BYTE PTR TIME+1,CH  ;ORA ERTEKEK ELHELYEZESE TIME-BAN
```

Mivel AH-t 2-re állítottuk be, az 1AH megszakítás az éppen érvényes idő – amit három 8 bites regiszter tárol – értékét adja meg. Ez a program a TIME változóba, amit duplaszónak (DD) definiáltunk, menti el az idő értékét. A programszegmensben látható, hogy ezt a BYTE PTR utasítással valósítjuk meg.

A következő programszegmens beolvassa és dekódolja a dátumot:

```
MOV     AH,04           ;A DATUMRA VONATKOZO PARAMETER
INT     1AH             ;IDO/DATUM MEGSZAKITAS
MOV     BYTE PTR DATE+3,DL  ;A NAP ELHELYEZESE DATE-BEN
MOV     BYTE PTR DATE+2,DH  ;A HONAP ELHELYEZESE DATE-BEN
MOV     BYTE PTR DATE+1,CL  ;AZ EV ELHELYEZESE DATE-BEN
MOV     BYTE PTR DATE,CH    ;EVSZAZAD HOZZARENDELESE AZ EVHEZ
```

A dátum beolvasásához AH-t 4-re kell beállítanunk.

Ha az 1AH megszakítást meghívjuk, a kódolt információt négy, egyenként 8 bites regiszter

adja vissza. A BYTE PTR utasítás segítségével a dekódolt dátumot a DATE elnevezésű duplaszónak definiált változóban helyezzük el. Mind a TIME, mind a DATE változó értéke a program futása után a nyomkövető programmal készített memóriakiíratásban megfigyelhető.

Szintaxis: INT 1AH (ahol a különböző paramétereket a kívánt értékre kell beállítani).

Funkció: Ez a megszakítás a rendszeróra beállítását vagy beolvasását teszi lehetővé.

AH értéke	Funkció	Bemenet	Kimenet
AH = 0	Az éppen érvényes óra beolvasása		CX = Az óraérték magas byte-jai DX = Az óraérték alacsony byte-jai AL = 0, ha a timer nem haladt még át a 24 órán
AH = 1	Az éppen érvényes óra beállítása	CX =	Az óraérték magas byte-jai
		DX =	Az óraérték alacsony byte-jai
AH = 2	Időbeolvasás – valós idő óra		CH = az órák BCD-ben CL = a percek BCD-ben DH = a másodpercek BCD-ben
AH = 3	Időbeállítás – valós idő óra	CH = CL = DH = DL =	Az órák BCD-ben A percek BCD-ben A másodpercek BCD-ben 1 – Aznap eltelt idő 0 – Szabványidő
AH = 4	Dátumbeolvasás – valós idő óra		CH = Az évszázad BCD-ben CL = Az év BCD-ben DH = A hónap BCD-ben DL = A nap BCD-ben
AH = 5	Dátumbeállítás – valós idő óra	CH = CL = DH = DL =	Az évszázad BCD-ben Az év BCD-ben A hónap BCD-ben A nap BCD-ben
AH = 6	Riasztásbeállítás (23:59:59-ig)	CH = CL = DH =	Az óra BCD-ben A perc BCD-ben A másodperc BCD-ben
AH = 7	Riasztás törlése		

5–5. táblázat 1AH típusú BIOS megszakítás opciói az IBM AT (80286) számítógép esetén

Az IBM AT memóriaméretének megállapítása BIOS megszakítással

Az IBM AT gépek beépített memóriájának méretét két BIOS megszakítás felhasználásával olvashatjuk be. A következő listán maga a program látható:

```
;IBM 80286-OS GEPRE KESZITETT PROGRAM
;AZ IBM AT MEMORIAMERETENEK MEGHATAROZASA
;BIOS MEGSZAKITAS FELHASZNALASAVAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
SYSMEM  DW      ?               ;KOZPONTI MEMORIA (640 K-IG)
EXTMEM  DW      ?               ;KIBOVITETT MEMORIA (640 K FELETT)
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

;A KOZPONTI MEMORIA MERETET MEGHATAROZO RUTIN
        INT     12H             ;A MEMORIARA VON. MEGSZAKITAS HIVASA
        MOV     SYSMEM,AX       ;AX ERTEKENEK ELHELYEZESE SYSMEM-BE
;A KIBOVITETT MEMORIATERULET MEGHATAROZASA
        MOV     AH,88H          ;MEGSZAKITASI PARAMETER
        INT     15H             ;A KIBOVITETT MEMORIARA VONATKOZO
                                ;MEGSZAKITAS HIVASA
        MOV     EXTMEM,AX


        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A 12H megszakítást mind a PC (8088), mind az AT (80286) számítógépek támogatják. A 12H megszakítás a rendszerbe beépített összefüggő központi memória méretét állapítja meg.

```
INT     12H             ;A MEMORIARA VON. MEGSZAKITAS HIVASA
MOV     SYSMEM,AX       ;AX ERTEKENEK ELHELYEZESE SYSMEM-BE
```

Az AX-ben képződő érték a beépített memória 1 kbyte-os blokkjainak számát jelenti.

Az ezt követő programszegmens a "bújtasd ahova tudod" programozási technika érdekes esete.

```
MOV     AH,88H          ;MEGSZAKITASI PARAMETER
INT     15H             ;A KIBOVITETT MEMORIARA VONATKOZO
                        ;MEGSZAKITAS HIVASA
MOV     EXTMEM,AX
```

A PC (8088) számítógépben az 15H megszakítás a kazettás adathordozó I/O funkcióit szolgálta. Az AT-nak (80286) nincs ilyen portja, így ez a megszakítás érdekes feladatokat lát el. Ezek közül az egyik a kibővített memóriaterület méretét határozza meg. Ha AX-et 88H-ra állítjuk be, és meghívjuk a megszakítást, a 640 K feletti összefüggő memória méretét az AX regiszterben keletkező értékből megtudhatjuk. (Az érték az 1 kbyte-os blokkok számát jelenti.)

Opcionális egységek meghatározása BIOS megszakítással

A következőkben egy olyan programot mutatunk be, ami a PC és AT számítógépek bizonyos egységeinek meghatározását szolgálja.

```
;8088/80286-OS IBM SZAMITOGEPEKRE KESZITETT PROGRAM
;A SZAMITOGEP OPCIONALIS EGYSEGEINEK MEGHATAROZASA
;BIOS MEGSZAKITASSAL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
EQUIP   DW      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

;AZ EGYSEGEK MEGHATAROZASANAK RUTINJA
        INT     11H             ;AZ EGYSEGEKET MEGHATAROZO MEGSZKAKITAS
        MOV     EQUIP,AX        ;AZ INFORMACIO EQUIP-BA KERUL

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A program mind a PC, mind az AT számítógépen működőképes. Az EQUIP változóban képződő eredményt azonban mindkét gépnél másként kell értelmezni. A programszegmens a következő:

```
INT     11H             ;AZ EGYSEGEKET MEGHATAROZO MEGSZKAKITAS
MOV     EQUIP,AX        ;AZ INFORMACIO EQUIP-BA KERUL
```

Az EQUIP változó értékét a program futtatása után az adatszegmens kiíratásával, vagy az AX regiszter tartalmának a képernyőn való megjelenítésével tudhatjuk meg. A hexadecimális mennyiséget binárissá kell átalakítanunk, és a 15 bit mindegyikét külön kell vizsgálnunk. Az egységek az 5–6. táblázatban leírtak szerint határozhatók meg. Például ha az AX regiszterbe

Bitek	Egységek
PC és XT számítógépek	
15, 14	A nyomtatók száma
13	Használaton kívül
12	Játékadapter
11, 10, 9	RS–232 kártyák száma
8	Használaton kívül
7, 6	A hajlékonylemez-meghajtók száma: 00 bitérték 1 meghajtó 01 bitérték 2 meghajtó 10 bitérték 3 meghajtó 11 bitérték 4 meghajtó
5, 4	Kiindulási videóüzemmód 01 = 40 * 25 fekete/fehér üzemmód színes kártya használatával 10 = 80 * 25 fekete/fehér üzemmód színes kártya használatával 11 = 80 * 25 fekete/fehér üzemmód monokrom kártya használatával
3, 2	RAM-méret 00 = 16 K 01 = 32 K 10 = 48 K 11 = 64 K
1	Használaton kívül
0	További lemezmeghajtó
AT számítógép (csak az XT-től eltérő paramétereket jelezzük)	
12	Használaton kívül
3, 2	Használaton kívül
1	80287-es társprocesszor

5–6. táblázat Az AX regiszter bitpozíciójának jelentése 11H megszakítás esetén

4463H kerül, ennek bináris megfelelője a következő:

```
 0  1  0  0  0  1  0  0  0  1  1  0  0  0  1  1 (Bináris szám)
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0 (Bitpozíció)
```

Ez a kódolás balról jobbra olvasva a következő egységeket jelenti:

– Lemezmeghajtók üzembe helyezve

- 80287-es matematikai társprocesszor
- 80 * 25-ös fekete/fehér üzemmód színes kártya használatával
- 2 lemezmeghajtó
- 2 RS-232 port
- 1 nyomtató

Karakterlánc kiküldése a nyomtatóra BIOS megszakítással

A következő program segítségével egy előzetesen definiált karakterlánc az LPT1-hez csatlakoztatott nyomtatóra küldhető ki.

```
;8088/80286-OS IBM GEPEKRE KESZITETT PROGRAM
;KARAKTERLANC KIKULDESE AZ LPT1-RE CSATLAKOZTATOTT
;NYOMTATORA, BIOS RUTIN FELHASZNALASAVAL
;A FUTTATAS ELOTT A CTRL-PRTSC LENYOMASA SZUKSEGES

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
MESSGE  DB      'AZ ASSEMBLY PROGRAMOK GYORSAK','$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        LEA     BX,MESSGE       ;A MESSGE VALTOZO HELYENEK BEKERESE

;KARAKTER KIKULDESE A NYOMTATORA
        MOV     DX,00H          ;PARAMETERBEALLITAS A MEGSZAKITASHOZ
        MOV     AH,01H          ;NYOMTATO-INICIALIZALAS
        INT     17H             ;A NYOMTATORA VONATKOZO MEGSZAKITAS
AGAIN:  MOV     AL,MESSGE[BX]   ;EGY KARAKTER BEKERESE A KAR.LANCBOL
        CMP     AL,'$'          ;VEGE VAN A SZOVEGNEK ?
        JE      ENDO            ;HA IGEN, A RUTIN BEFEJEZODIK
        MOV     AH,00H          ;A KAR. KINYOMTATASAHOZ SZUKSEGES
                                ;PARAMETER
        INT     17H             ;A NYOMTATORA VONATKOZO MEGSZAKITAS
        INC     BX              ;A KOVETKEZO KARAKTER BEKERESE
        JMP     AGAIN           ;ISMETLES
ENDO:

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A CTRL-PRTSC gombok lenyomása a program futtatása előtt szükséges. Ha a billentyűzetről egyszerre csak egy karakter beolvasása történik, a nyomtatón is egyszerre csak egy karakter kerül kinyomtatásra. A feladatot a következő programozási technikával tudjuk megvalósítani:

```
        MOV     DX,00H          ;PARAMETERBEALLITAS A MEGSZAKITASHOZ
        MOV     AH,01H          ;NYOMTATO-INICIALIZALAS
        INT     17H             ;A NYOMTATORA VONATKOZO MEGSZAKITAS
AGAIN:  MOV     AL,MESSGE[BX]   ;EGY KARAKTER BEKERESE A KAR.LANCBOL
        CMP     AL,'$'          ;VEGE VAN A SZOVEGNEK ?
        JE      ENDO            ;HA IGEN, A RUTIN BEFEJEZODIK
        MOV     AH,00H          ;A KAR. KINYOMTATASAHOZ SZUKSEGES
                                ;PARAMETER
        INT     17H             ;A NYOMTATORA VONATKOZO MEGSZAKITAS
        INC     BX              ;A KOVETKEZO KARAKTER BEKERESE
        JMP     AGAIN           ;ISMETLES
ENDO:
```

Ahhoz, hogy a 17H megszakítást használhassuk, a nyomtatót először inicializálni kell. DX értékét 0-ra, 1-re vagy 2-re állítjuk be a nyomtató bázisterületének (a nyomtatócímek) aktuális értékétől függően. A 0 az LPT1-re vonatkozik. A 17H megszakításnál a nyomtató inicializálásáról AH 1-re való beállításával gondoskodunk. (Mint a programból is látható, ezt csak egyszer kell megtennünk.) A program ezután egy ciklusba lép be, ami egyszerre egy karakter kiküldését teszi lehetővé. A MESSGE változóból egy karakter átkerül az AL regiszterbe, amelyet rögtön megvizsgálunk, hogy vezérlőkarakter-e. A nyomtató portja ui. nem rendelkezik beépített vezérlőkarakterrel. Ha a karakter nem elhatároló jel, AH-ba zérust léptetünk, és a 17H megszakítás kinyomtatja a karaktert a nyomtatóra. Ezt követően BX értékét növeljük, amivel a karakterlánc következő karakterére mutatunk rá és a ciklust megismételjük. A ciklus addig ismétlődik, amíg a $ elhatároló jellel találkozik. (A karakterlánc tehát meglehetősen hosszú is lehet.)

Pontok felrajzolása a közepes felbontású színes képernyőn BIOS megszakítással

Az IBM számítógépcsalád a szabványos színes monitoron kétféle grafikus üzemmódot támogat. Az egyik a közepes felbontású, a másik a nagyfelbontású üzemmód. A közepes felbontású üzemmódban vízszintesen 320 pontot, függőlegesen 200 pontot tudunk elhelyezni a képernyőn négy különböző színben. A nagyfelbontású üzemmód vízszintesen 640 pont, függőlegesen 200 pont elhelyezését támogatja két színben (általában a fekete-fehér színekben). A fejezetben már korábban szerepelt az 5–2. táblázat, amelyben látható, hogy a 11-es, 12-es és 13-as AH értékek írják elő a grafikus üzemmódot. A bemutatásra kerülő programmal három apró pontot helyezünk el a képernyő középpontja közelében.

```
;RGB MONITORRAL ELLATOTT 8088/80286-OS SZAMITOGEPEKRE
;KESZITETT PROGRAM
;PONTOK ELHELYEZESE KOZEPES FELBONTASU SZINES KEPERNYON
;BIOS MEGSZAKITAS FELHASZNALASAVAL
;SZINES GRAFIKUS KARTYA SZUKSEGES!

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
```

```
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

;HAROM PONT ELHELYEZESE A KEPERNYO KOZEPPONTJANAK KOZELEBEN
        MOV     AH,00           ;A KEPERNYO UZEMMODJANAK BEALLITASA
        MOV     AL,04           ;320*200-AS SZINES UZEMMOD
        INT     10H             ;A MEGSZAKITO RUTIN HIVASA

        MOV     AH,11           ;A SZINPALETTA BEALLITASA
        MOV     BH,00           ;A HATTER SZINE KEK LESZ
        MOV     BL,01
        INT     10H             ;A MEGSZAKITO RUTIN HIVASA

        MOV     AH,11           ;A SZINPALETTA BEALLITASA
        MOV     BH,01           ;A SZOVEGKIEMELES SZINENEK KIVALASZTASA
        MOV     BL,00           ;ZOLD/PIROS/SARGA
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

        MOV     AL,02           ;A PONT SZINE PIROS LESZ
        MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
        MOV     DX,64H          ;FUGGOLEGES 100 BEALLITASA
        MOV     CX,9EH          ;VIZSZINTES 158 BEALLITASA
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

        MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
        MOV     CX,0A0H         ;EGY MASIK PONT KOVETKEZIK
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

        MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
        MOV     CX,0A2H         ;EGY MASIK PONT KOVETKEZIK
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA


        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Mielőtt a pontokat elhelyeznénk a képernyőn, gondoskodnunk kell a közepes felbontású képernyő inicializálásáról. Az erre vonatkozó programszegmens a következő:

```
MOV     AH,00           ;A KEPERNYO UZEMMODJANAK BEALLITASA
MOV     AL,04           ;320*200-AS SZINES UZEMMOD
INT     10H             ;A MEGSZAKITO RUTIN HIVASA
```

(Ha visszalapozunk az 5–2. táblázathoz, láthatjuk, hogyan kell AH és AL értékét beállítani.) Következő lépés az előtér és a háttér színének beállítása:

```
MOV     AH,11           ;A SZINPALETTA BEALLITASA
MOV     BH,00           ;A HATTER SZINE KEK LESZ
MOV     BL,01
INT     10H             ;A MEGSZAKITO RUTIN HIVASA
```

Azzal, hogy BH-ba 0-t töltünk be, a kiválasztott szín a háttér színét jelenti. A színtartomány

0-tól 15-ig terjed (l. az 5–3. táblázatot). A háttér színe ebben a példában kék. Az előtér színét másképpen választjuk ki.

```
MOV     AH,11           ;A SZINPALETTA BEALLITASA
MOV     BH,01           ;A SZOVEGKIEMELES SZINENEK KIVALASZTASA
MOV     BL,00           ;ZOLD/PIROS/SARGA
INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA
```

A BH regisztert 11-re állítjuk be, ami azt jelenti, hogy az előtér színének kiválasztásáról van szó. A BL regiszterben szereplő érték a zöld/piros/sárga paletta kijelölését jelenti. Az inicializálásnak ezt a három lépését csak egyszer kell elvégeznünk, hacsak a teljes képernyő színét nem akarjuk megváltoztatni.

A program hátralevő részében a pont megjelenítését biztosító kódot használjuk.

```
MOV     AL,02           ;A PONT SZINE PIROS LESZ
MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
MOV     DX,64H          ;FUGGOLEGES 100 BEALLITASA
MOV     CX,9EH          ;VIZSZINTES 158 BEALLITASA
INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
MOV     CX,0A0H         ;EGY MASIK PONT KOVETKEZIK
INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
MOV     CX,0A2H         ;EGY MASIK PONT KOVETKEZIK
INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA
```

A feladat elvégzéséhez DX-nek a függőleges, CX-nek a vízszintes eltolási értéket kell tartalmaznia. Mivel egy sorban három pontot akarunk elhelyezni, ezért CX értékét a következő pontok megjelenítése előtt kettővel mindig növeljük.

Mivel az AL regisztert 2-re állítottuk be, a pontok színe piros lesz. A BASIC és Pascal nyelv számos grafikus utasítást támogat, azonban a BIOS csak a pontok megjelenítését és helyzetük beolvasását teszi lehetővé. Emiatt pl. egy vonalat pontok sorozatának megjelenítéséből kell összeállítanunk. Ha ez a vonal vízszintes vagy függőleges, nincs is különösebb baj. De képzeljünk el az ettől eltérő helyzetek bonyolultságát. A körről nem is beszélve!

Vonalrajzolás a nagyfelbontású képernyőre BIOS megszakítással

Az előző fejezetrészben megismert feladatot most kibővítjük úgy, hogy vonalat rajzolunk a nagyfelbontású képernyőre. A következő programlista azt mutatja, hogyan lehet ezt a műveletet egy ciklusban megvalósítani.

```
;RGB MONITORRAL ELLATOTT 8088/80286-OS IBM SZAMITOGEPEKRE
;KESZITETT PROGRAM
;NAGYFLEBONTASU GRAFIKUS KEPERNYON EGY VONAL PONTOKBOL
;VALO OSSZEALLITASA BIOS MEGSZAKITAS FELHASZNALASAVAL
;SZINES GRAFIKUS KARTYA SZUKSEGES!

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS
```

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

;VONAL OSSZEALLITASA PONTOKBOL
        MOV     AH,00           ;A KEPERNYO UZEMMODJANAK BEALLITASA
        MOV     AL,06           ;640$200-ASRA
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

        MOV     AL,01           ;A PONT SZINENEK BEALLITASA
        MOV     DX,00           ;FUGGOLEGES KEZDOHELYZET A BAL OLDALON
        MOV     CX,00           ;VIZSZINTES KEZDOHELYZET FELUL
AGAIN:  MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA
        INC     DX              ;A PONT FUGG. HELYZETENEK NOVELESE
        INC     CX              ;A PONT VIZSZ. HELYZETENEK NOVELESE
        INC     CX
        INC     CX
        CMP     DX,0C8H         ;A VONAL ELERTE A KEPERNYO ALJAT (200)?
        JE      ENDO            ;HA IGEN, A PROGRAM BEFEJEZODIK
        JMP     AGAIN           ;HA NEM, UJ PONT ELHELYEZESE KOVETKEZIK
ENDO:

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Mivel csak a két alapértelmezés szerint megadott színt használjuk (fekete/fehér), a háttér és az előtér színének beállításával nem kell törődnünk. A programszegmensben érdemes megfigyelnünk, hogy a 640 * 200-as grafikus üzemmód beállításához AL-be 6-ot kell elhelyeznünk.

```
        MOV     AH,00           ;A KEPERNYO UZEMMODJANAK BEALLITASA
        MOV     AL,06           ;640$200-ASRA
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA

        MOV     AL,01           ;A PONT SZINENEK BEALLITASA
        MOV     DX,00           ;FUGGOLEGES KEZDOHELYZET A BAL OLDALON
        MOV     CX,00           ;VIZSZINTES KEZDOHELYZET FELUL
AGAIN:  MOV     AH,12           ;A PONT PARAMETERENEK BEIRASA
        INT     10H             ;A MEGSZAKITO RUTIN MEGHIVASA
        INC     DX              ;A PONT FUGG. HELYZETENEK NOVELESE
        INC     CX              ;A PONT VIZSZ. HELYZETENEK NOVELESE
        INC     CX
        INC     CX
        CMP     DX,0C8H         ;A VONAL ELERTE A KEPERNYO ALJAT (200)?
        JE      ENDO            ;HA IGEN, A PROGRAM BEFEJEZODIK
        JMP     AGAIN           ;HA NEM, UJ PONT ELHELYEZESE KOVETKEZIK
ENDO:
```

Ez a program a képernyő bal felső sarkától kezdve (DX = 0 és CX = 0) egy átlós vonalat húz. A pontok színe fehér (AL = 1), és az első pont akkor kerül a képernyőre, amikor a program belép a ciklusba. A függőleges eltolás (DX) növekménye 1, a vízszintes eltolásé (CX)

pedig 3. (Mivel a képernyő mérete pontokban kifejezve 640 vízszintesen, és 200 függőlegesen, a vízszintes növekményt a függőleges érték háromszorosára választottuk meg, hogy jobb átlós vonalat kapjunk.) Amikor függőleges értelemben a 200. pont elhelyezése is megtörtént, a program befejeződik.

5.7 KARAKTER KERESÉSE KARAKTERLÁNCBA FEJLETT KARAKTERLÁNC-KEZELŐ UTASÍTÁS HASZNÁLATÁVAL

Ennek a fejezetrésznek a programjai igen gyors programfutást tesznek lehetővé azáltal, hogy nem tartalmaznak ciklusokat. Az első programban egy adott karaktert keresünk meg egy karakterláncban. Ha a programot ezek nélkül a speciális utasítások nélkül írnánk meg, a karakterlánc minden egyes karakterének indexeléséről gondoskodnunk kellene.

A második programban egy teljes karakterlánc átmásolását valósítjuk meg az egyik változóból a másikba, ciklus használata nélkül.

```
;8088/80286-OS IBM GEPEKRE KESZITETT PROGRAM
;HATASOS KARAKTERKERESO ELJARAS BEMUTATASA
;

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
GROUP1  DB      'AZ ELLENSEGETOL IS TANUL AZ OKOS'
MESS1   DB      'A BETUT A KARAKTERLANCBAN MEGTALALTAM$'
MESS2   DB      'ILYEN BETU NEM SZEREPEL A KARAKTERLANCBAN$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,ES:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     ES,AX           ;AX TARTLAMANK ELHELYEZESE ES-BE

;KARAKTERKERESES A GROUP1 KARAKTERLANCBAN
        CLD                     ;A KERESES IRANYA BALROL JOBBRA
        LEA     DI,GROUP1       ;GROUP1 HELYENEK BEKERESE
        MOV     CX,32           ;A BETUK SZAMANAK ELHELYEZESE CX-BEN
        MOV     AL,'X'          ;AZ 'X' BETUT KERESSUK A KAR.LANCBAN
REPNE   SCASB                   ;KERESESI MUVELET
        JCXZ    NONE            ;UGRAS NONE-RA, HA NINCS ILYEN
                                ;KARAKTER
```

```
        LEA     DX,MESS1        ;HA VAN 'X', A MESS1 UZENET BEKERESE
        JMP     PRINTIT         ;ES MEGJELENITESE
NONE:   LEA     DX,MESS2        ;HA NINCS 'X', A MESS2 UZENET BEKERESE

;A MEGFELELO UZENET  MEGJELENITESE A KEPERNYON
PRINTIT:
        MOV     AH,09           ;PARAMETERBEALLITAS
        INT     21H             ;DOS MEGSZAKITAS

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Az adatszegmens három karakterláncot tartalmaz:

```
GROUP1  DB      'AZ ELLENSEGETOL IS TANUL AZ OKOS'
MESS1   DB      'A BETUT A KARAKTERLANCBAN MEGTALALTAM$'
MESS2   DB      'ILYEN BETU NEM SZEREPEL A KARAKTERLANCBAN$'
```

A megadott karaktert a GROUP1 elnevezésű karakterláncban fogjuk kerestetni. A MESS1 és MESS2 karakterláncok a képernyőn megjelenítendő üzeneteket tartalmazzák.

A karakterlánc-utasítások használatakor a karakterlánc információfor- rását és célját azonosítani kell. Ilyen kis programoknál valamennyi adatot egyetlen szegmens tartalmaz. A DS és ES regiszterek mindegyike a megfelelő szegmens helyének értékét veszi fel. A keresési műveletet megvalósító kódszegmens a következő:

```
        CLD                     ;A KERESES IRANYA BALROL JOBBRA
        LEA     DI,GROUP1       ;GROUP1 HELYENEK BEKERESE
        MOV     CX,32           ;A BETUK SZAMANAK ELHELYEZESE CX-BEN
        MOV     AL,'X'          ;AZ 'X' BETUT KERESSUK A KAR.LANCBAN
REPNE   SCASB                   ;KERESESI MUVELET
        JCXZ    NONE            ;UGRAS NONE-RA, HA NINCS ILYEN
                                ;KARAKTER
        LEA     DX,MESS1        ;HA VAN 'X', A MESS1 UZENET BEKERESE
        JMP     PRINTIT         ;ES MEGJELENITESE
NONE:   LEA     DX,MESS2        ;HA NINCS 'X', A MESS2 UZENET BEKERESE
```

A CLD utasítás balról jobbra állítja be a keresés irányát. Annak a karakterláncnak a helyét, amelyben a keresést folytatjuk, a DI regiszterben kell megadni. A rutin gyorsaságát és hatékonyságát a REPNE SCASB utasítás biztosítja, ami arra készteti a processzort, hogy a karakterláncban az előírt karaktert megkeresse. (A keresés ismétlési száma 34, mivel a karakterlánc ennyi karakterből áll.) Ha az utasítás nem találja a megadott karaktert, CX zérusra csökken és a JCXZ ugrás hatására a MESS2 üzenet jelenik meg a képernyőn. Ha a karaktert a program megtalálja, ezt a MESS1 üzenettel jelzi.

(Ezt a karakterkeresési módszert a 8. fejezet helyesírást ellenőrző programjában fogjuk hasznosítani.)

5.8 SZEGMENSEN BELÜLI KARATERLÁNC-ÁTHELYEZÉS

A következő lista egy másik karakterlánc-műveletet mutat be:

```
;8088/80286-OS IBM GEPEKRE KESZITETT PROGRAM
;HATEKONY KARAKTERLANC-KEZELO UTASITAS BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
GROUP1  DB      'AZ ELLENSEGETOL IS TANUL AZ OKOS'
GROUP2  DB      32 DUP (?),'$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,ES:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE

;A GROUP1 KARAKTERLANC TARTALMANK ATHELYEZESE
;A GROUP2 KARAKTERLANCBA
        LEA     SI,GROUP1       ;AZ INFORMACIOT TARTALMAZO KAR.LANC
                                ;HELYE
        LEA     DI,GROUP2       ;A CELKENT MEGJELOLT KAR.LANC HELYE
        MOV     CX,32           ;32 BYTE-OT HELYEZUNK AT
REP     MOVSB                   ;AZ ATHELYEZESI MUVELET

;A GROUP2 KARAKTERLANC MEGJELENITESE A KEPERNYON
        LEA     DX,GROUP2       ;PARAMETEREK BEALLITASA
        MOV     AH,09
        INT     21H             ;DOS MEGSZAKITAS

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Ez a program az egyik változóban tárolt karakterláncot átmásolja a másik változóba. A konkrét műveletet a REP MOVSB utasítás hajtja végre igen gyorsan. Az erre vonatkozó programszegmenset külön kiemeljük:

```
        LEA     SI,GROUP1       ;AZ INFORMACIOT TARTALMAZO KAR.LANC
                                ;HELYE
        LEA     DI,GROUP2       ;A CELKENT MEGJELOLT KAR.LANC HELYE
        MOV     CX,32           ;32 BYTE-OT HELYEZUNK AT
REP     MOVSB                   ;AZ ATHELYEZESI MUVELET
```

Az információt hordozó forráskarakterlánc címét az SI regiszterben, míg a célkarakterlánc címét a DI regiszterben helyezzük el. CX-et 32-re kell beállítani, mivel ennyi a forrás karaktereinek száma (a másolandó karakterek száma).

A program nagyon gyorsan lefut. A karakterek áthelyezéséhez szükséges adatokat az előbbi programszegmens első három sorában állítjuk be, míg magát az áthelyezési műveletet a REP MOVSB utasítás végzi el.

6. AZ ASSEMBLER PSZEUDO MŰVELETEINEK HASZNÁLATA

A pszeudo műveletek az assembly programozás lényeges részei, mivel ezekkel az assemlert bizonyos feladatok elvégzésére késztethetjük. A pszeudo művelet inkább assembly parancsnak vagy operandusnak tekintendő, mint assembly utasításnak. A pszeudo műveletekből – amelyeket az assembler a fordítás időszakában használ fel – nem készül gépi kód. Emiatt pszeudo műveletek inkább assembler-, mint processzorfüggőek. Az előbbi fejezetekben már szó volt arról, hogy az assembly nyelv igen erősen processzorfüggő. Nem történik azonban semmi különös, ha 8088-as írt programot 80286-oson futtatunk, ui. az Intel cég a processzorait felülről kompatibilisnek tervezte. A 80286-as utasításkészlete nagyobb és hatékonyabb, mint a 8088-as processzoré. Hasonló kompatibilitás figyelhető meg az assembler nyelvek területén is. Igy az Intel cég ASM286-os és ASM386-os assemblerjei képesek lefordítani a 8088/80386-osra írt kódot is. A többi 8088/80386-os assembler az Intel ASM assembler nyelv részhalmazát használja, kiegészítve igen hasznos program- csomagokkal. A Microsoft Assembler és az IBM Assembler majdnem azonos termékek, már csak azért is, mert mindkettőt a Microsoft cég készítette. Az elmúlt években főleg ezeket a programokat használták a 8088/8086-os számítógépeken. A Speedware cég TURBO EDITASM elnevezésű assemblerét érdemes még megemlíteni, ami a 80286-os számítógépcsaládhoz készült. A TURBO EDITASM igen sok Microsoft által készített pszeudo műveletet támogat. Az előbb említett szoftvereken kívül más cégek is gyártottak assemblereket, amelyek között bizonyos szoftverkompatibilitás tapasztalható. Az 5. fejezetben szereplő programokat úgy írtuk meg, hogy csak minimális mennyiségű pszeudo műveletre legyen szükség, ott ui. a mikroprocesszor műveleti kódjainak használatára fektettük a hangsúlyt.

Ebben a fejezetben az IBM, Microsoft és és Speedware assemblereivel alkalmazható népszerű pszeudo műveletek ismertetésére törekszünk. Igyekszünk olyan példákat bemutatni, amelyekből kiderül, hogy hol használhatók a leghatásosabban ezek a pszeudo műveletek. A könyv további fejezeteiben igen gyakran alkalmazzuk a pszeudo műveleteket, mivel ezekkel az assembly programozás még hatékonyabb lesz.

6.1 PSZEUDO MŰVELETEK

A pszeudo műveletek – az assembler mnemonikokhoz hasonlóan – számos kategóriába sorolhatók. Ebben a könyvben a következő öt pszeudo műveleti kategóriát különböztetjük meg:

feltételes,
adat,
listázó,
makro,
üzemmód.

A pszeudo műveleteket numerikus, ill. alfabetikus sorrendben tárgyaljuk.

.186 (A 80186-os üzemmódot írja elő)

Assemblerek: Microsoft, Speedware
Leírás: A 80186-os utasítások engedélyezése
Formátum: .186 (operandus nélkül)
Megjegyzés: Ez az utasítás a .8086-os pszeudo művelettel érvényteleníthető. A .186 pszeudo művelet lehetővé teszi valamennyi 8086-os és 80186-os utasítás fordítását, ha azok nincsenek védett üzemmódban.
Példa:

```
;ASSEMBLY PROGRAM FORDITASA ES FUTTATASA 80186-OS
;SZAMITOGEPEN

.186
STACK   SEGMENT PARA STACK
        DB      64 DUP('MYSTACK')
STACK   ENDS

;Itt kovetkezik a program tovabbi resze
```

.286C (A 80286-os üzemmódot írja elő)

Assemblerek: Microsoft, IBM és Speedware
Leírás: A .286C pszeudo művelet megadása a 80286-os utasítások lefordításához szükséges.
Formátum: .286C (operandus nélkül)
Megjegyzés: a .286C-vel beállított üzemmód .8086-tal érvényteleníthető. A .286C pszeudo művelet lehetővé teszi valamennyi 8086-os és 80286-os utasítás lefordítását, ha azok nincsenek védett üzemmódban.
Példa:

```
;LOGARITMUS ERTEKEK KIKERESESI TABLAZATAT FELHASZNALO
;PROGRAM

.286C
STACK   SEGMENT PARA STACK
        DB      64 DUP('MYSTACK')
STACK   ENDS

;Itt kovetkezik a program tovabbi resze
```

.286P (A 80286-os védett üzemmód előírása)

Assemblerek: Microsoft

Leírás: A .286P pszeudo műveletre akkor van szükség, amikor a 80286 védett módú utasításait kívánjuk lefordítani.

Formátum: .286P (operandus nélkül)

Megjegyzés: A .286P-vel beállított 80286-os módot a 8086-os pszeudo művelettel lehet érvényteleníteni. A .286P azokat a 8086-os és 80286-os utasításokat is lefordítja, amelyek nincsenek védett módban.

Példa:

```
;LOGARITMUS ERTEKEK KIKERESESI TABLAZATAT FELHASZNALO
;PROGRAM

.286P
STACK   SEGMENT PARA STACK
        DB      64 DUP('MYSTACK')
STACK   ENDS

;Itt kovetkezik a program fo resze

CLTS    ;A TASZKVALTAS JELZOK TORLESE

;Itt kovetkezik a program tovabbi resze
```

.287 (A 80287-es üzemmód előírása)

Assemblerek: Microsoft

Leírás: A .287 pszeudo művelet lehetővé teszi a Microsoft assembler számára, hogy a kiegészítő 80287-es utasításokat is lefordítsa.

Formátum: .287 (operandus nélkül)

Megjegyzés: A .287 pszeudo művelet valamennyi 8087-es utasítást, valamint a következők lefordítását teszi lehetővé:

Példa:

```
FSETPM          ;VEDETT MOD BEALLITASA
FSTSW   AX      ;A STATUSZ-SZO ELMENTESE AX-BE (VARAKOZAS)
FNSTSW  AX      ;A STATUSZ-SZO ELMENTESE AX-BE (NINCS VARAKOZAS)
```

Példa:

```
;NEHANY SZAM KIVONASAT BEMUTATO PROGRAM
;A 80287-ES MOD BEALLITASA

.287
STACK   SEGMENT PARA STACK
        DB      64 DUP('MYSTACK')
STACK   ENDS

;Itt kovetkezik a program fo resze
```

.8086 (A 8088/8086-os üzemmód előírása)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Mindhárom assembler alapértelmezésként a 8088/8086-os üzemmódban működik. A .8086 pszeudo művelet a megelőző beállítás (pl. .286C) megszüntetését szolgálja.
Formátum: .8086 (operandus nélkül)
Megjegyzés: Ha az assemblerek ebben az üzemmódban dolgoznak, a 80286-os utasításokat nem fordítják le.
Példa:

```
;NEHANY SZAM OSSZEADASAT BEMUTATO PROGRAM
;A 80286-OS MOD BEALLITASA

.286C
STACK   SEGMENT PARA STACK
        DB      64 DUP('MYSTACK')
STACK   ENDS

;Itt kovetkezik a program fo resze

;A 8086-OS UZEMMOD VISSZAALLITASA
.8086

;Itt kovetkezik a program tovabbi resze
```

.8087 (A 8087/80287-es üzemmód előírása)

Assemlerek: IBM, Microsoft és Speedware
Leírás: A .8087 pszeudo művelet minden assembler számára lehetővé teszi, hogy a 8087/80287-es utasításokat és adatokat lefordítsa.
Formátum: .8087 (operandus nélkül)
Megjegyzés: Ennek a műveletnek ugyanaz a hatása az assemblerekre, mint az IBM vagy Microsoft assemblerek meghívásakor a /R paraméternek.
Példa:

```
;VALOS SZAMOK OSSZEADASAT BEMUTATO PROGRAM

.8087

MYDATA  SEGMENT PARA 'DATA'
NUMBER1 DQ      7123.45678912345        ;VALOS SZAM
NUMBER2 DQ      10.102                  ;VALOS SZAM
THESUM  DQ      ?                       ;AZ EREDMENY VALOSBAN

;Itt kovetkezik a program hatralevo resze
```

&

(& makro operátor)

Assemlerek: IBM
Leírás: A & pszeudo művelet a makróbővítményekben szöveg és szimbólum konkatenálására szolgál.
Formátum: *(szöveg_vagy_szimbólum)&(szöveg_vagy_szimbólum)*
Megjegyzés: Formális változót vagy paramétert, ami egy karakterláncban van, vagy nincs előtte elhatároló karakter, a & pszeudo műveletnek kell megelőznie, hogy a makrokiterjesztéskor behelyettesíthető legyen.
Példa:

```
MYLUCK  MACRO   GUESS           ;MAKRO PROGRAM
REP&T:  MOV     DX,01234H
        MOV     AX,'&T'
        ENDM

;Itt kovetkezik a program fo resze

        MYLUCK  G               ;MAKRO HIVAS

;Itt kovetkezik a makro kod

REPG:   MOV     DX,01234H
        MOV     AX,'G'
```

=

(= pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A = pszeudo művelet a program konstansainak definiálását és újradefiniálását teszi lehetővé.
Formátum: *(címke)* = *(érték)*
Megjegyzés: A = pszeudo művelet csak numerikus kifejezésekben használható. (Az EQU pszeudo művelethez nagyon hasonlít.)
Példa:

```
VAR1    =       1234H           ;UJRA DEFINIALHATO
VAR2    =       56H             ;UJRA DEFINIALHATO
VAR1    =       78H             ;UJRA DEFINIALHATO
VAR4    EQU     3456            ;NEM DEFINIALHATO UJRA
```

!

(! makro operátor)

Assemblerek: IBM
Leírás: Ha a karaktert a felkiáltójel előzi meg, akkor a karakter literális bevitele engedélyezetté válik.
Formátum: !*(karakter)*
Megjegyzés: !*(karakter)* jelölés ekvivalens a <*karakter*> jelöléssel.
Példa: !*,!§,!^

%

(% makro operátor)

Assemblerek: IBM
Leírás: Az a szimbólum, amit a % jel előz meg, egy számmá alakul át az alapértelmezés szerinti számrendszerben. A makrokiterjesztés során ez a szám a formális változót cseréli fel.
Formátum: *% (szimbólum)*
Megjegyzés: A % pszeudo művelet csak a makrókban használható.
Példa:

```
MYLUCK  MACRO   V
T       =       V-2
        MOV     AX,%T   ;;AZ EREDMENY ELHELYEZESE AX-BEN
        ENDM
```

%OUT

(% OUT pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A %OUT pszeudo művelet segítségével egy üzenet küldhető ki a kezelőpultra minden olyan esetben, ha az assembly programban egy ilyen művelet szerepel. (Így tudjuk jelezni az assembly program jelenlétét.)
Formátum: %OUT *(üzenet)*
Megjegyzés: Nincs
Példa:

```
%OUT    AZ ASSEMBLY PROGRAM KEZDETE

;Itt kovetkezik a program egy resze

%OUT    LEGY TURELEMMEL, MEG FORDITOK

;Itt kovetkezik a program hatralevo resze

%OUT    A FORDITAS ELKESZULT
```

;;

(;; makro operátor)

Assemblerek: IBM
Leírás: Ha a megjegyzési rovat előtt két pontosvesszőt használunk, akkor a megjegyzés a makrobővítménybe nem kerül bele.
Formátum: ;;*(megjegyzés_karakterlánc)*
Megjegyzés: A két pontosvessző használatával jelentős helymeg- takarítást tudunk elérni a programban, ha gyakran bővítünk makróval.
Példa:

```
MYLUCK  MACRO   GUESS           ;;MAKRO PROGRAM
REP&T   MOV     DX,01234H       ;;HELY BETOLTESE
        MOV     AX,'&T'
        ENDM                    ;;A MAKRO VEGE
```

ASSUME

(ASSUME pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Ez a pszeudo művelet jelenti be az assemblernek azt az igényt, hogy egy szegmenst feleltessen meg egy szegmensregiszternek. A megelőző igényeket az ASSUME NOTHING törli.
Formátum: ASSUME*(szegmens_regiszter:szegmens_név)*
Megjegyzés: Az érvényes szegmensregiszterek a következők: CS, DS, ES, SS
Példa:

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

COMMENT

(COMMENT pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Ez a pszeudo művelet lehetővé teszi, hogy a programban pontosvesszők nélkül helyezzünk el megjegyzést (többsoros megjegyzések esetén ez különösen hasznos).
Formátum: COMMENT *(határoló karakter)szöveg(határoló karakter)*
Megjegyzés: A határoló karaktert a programozó választhatja meg. A megjegyzés hossza nincs korlátozva.
Példa:

```
COMMENT @A SZOVEGET KET HATAROLO KARAKTER KOZE KELL ELHELYEZNI@
```

.CREF (.XCREF)

(.CREF pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A .CREF egy alapértelmezés szerinti feltétel, ami a keresztreferencia kiszolgáló program meghívása esetén lehetővé teszi a keresztreferencia kimenetet. A .XCREF az kimenet kikapcsolását szolgálja.
Formátum: .CREF (operandus nélkül)
Megjegyzés: A .XCREF-fel egy opcionális operandus is használható, amellyel megakadályozhatjuk, hogy a megnevezett változók a kereszt- referencia kimenetben megjelenjenek.
Példa:

```
MYDATA  SEGMENT PARA 'DATA'
        INFO    DB      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
        ANS     DB      ?
MYDATA  ENDS
.XCREF ANS
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALAS A MASM-NEK

:Itt kovetkezik a program hatralevo resze
```

DB (DB pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A DB (Define Byte = byte definiálás) pszeudo művelet egy változó vagy táblázat inicializálásakor byte-méretű (8 bites) hely lefoglalását szolgálja.
Formátum: *(változó)*DB *(kifejezés)*
Megjegyzés: A változó elnevezése opcionális. Kifejezésként szerepelhet konstans, karakterlánc, vagy vesszővel elválasztott konstansokból álló táblázat. Két további lehetőség a kérdőjel, ami csak a memóriahelyet foglalja le, és a DUP utasítás, amivel egy bizonyos érték többszörös beléptetése érhető el. A DB-vel definiált változó 0-tól FFH-ig (decimálisan 0-tól 255-ig) képes a számokat tárolni.
Példa:

```
HELLO   DB      23
THERE   DB      0AH
YOU     DB      'G'
LUCKY   DB      'BUDAPESTI MUSZAKI EGYETEM'
PERSON  DB      0,1,2,3,4,5,6,7,8,9
HOW     DB      45 DUP ('STACK')
ARE     DB      50 DUP (03CH)
YOU     DB      'YES',34,0FADH
```

DBIT (DBIT pszeudo művelet)

Assemblerek: Ez a pszeudo művelet a 80386-os processzor speciális adattípusa, ezért csak az ASM386-os assemblerrel használható.
Leírás: A DBIT (Define BIT = bit-definilálás) pszeudo művelet önálló bit vagy bitlánc definiálására használható. A bitlánc 1-től 32-ig tartalmazhat bináris számjegyeket, amelyeket a B karakterrel kell lezárni.
Formátum: *(változó)*DBIT *(kifejezés)*
Megjegyzés: A DBIT egy teljes byte-ot inicializál, amit rendszerint byte-határon sorakoztat fel.
Példa:

```
I       DBIT    100B    ;BYTE-BAN KIFEJEZVE 00000100
AM      DBIT    1B      ;BYTE-BAN KIFEJEZVE 00000001
SMALL   DBIT    3 DUP (1B) ;HAROM BYTE, AMELYEK MINDEGYIKE
                        ;00000001-ET TARTLAMAZ
```

DD (DD Pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A DD (Defined Doubleword = duplaszó-definiálás) pszeudo művelet egy változó vagy egy táblázat definiálását, illetve egy tárolási hely inicializálását szolgálja. Kétszavas (4 byte, azaz 32 bit) helyfoglalást tesz lehetővé.
Formátum: *(változó)*DD *(kifejezés)*
Megjegyzés: A változó elnevezése opcionális. Kifejezésként vesszővel elválasztott konstansokból álló táblázat szerepelhet. Két további lehetőség a kérdőjel, ami csak a memóriahelyet foglalja le, és a DUP utasítás, amivel egy bizonyos érték többszörös beléptetése érhető el. A DD-vel definiált változó 0-tól 0FFFFFFFFH-ig (decimálisan 0-tól 4294967295-ig) képes a

számok tárolására. Ha decimális pont, vagy exponenciális jelölés szerepel, akkor lebegőpontos szám keletkezik.

Példa:

```
HELLO   DD      0FFFFFFFFH
THERE   DD      0H
YOU     DD      1,22,333,4444,55555,666666,7777777,88888888
LUCKY   DD      45H+23H
PERSON  DD      2.17654
HOW     DD      4.567E12
ARE     DD      50 DUP (03CH)
YOU     DD      0.12345E-2
```

DP

(DP pszeudo művelet)

Assemblerek: Ez a pszeudo művelet a 80386-os processzor speciális adattípusa, ezért csak az ASM386-os assemblerrel használható.

Leírás: A DP (Define Pointer = Mutató definiálása) pszeudo művelet PWORD típusú 48 bites változó definiálására használható. A PWORD- nek egésznek kell lennie. Konstansok, változók és karakterláncok (maximálisan hat karakterig) szerepelhetnek.

Formátum: *(változó)*DP *(kifejezés)*

Megjegyzés: Nincs

Példa:

```
AA      DP      'ABCDEF'
BB      DP      ?
CC      DP      3 DUP (0A76543H)
```

DQ

(DQ pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware

Leírás: A DQ (Define Quadword = Négyesszó definiálása) pszeudo művelet egy változó vagy táblázat definiálását, ill. egy tárolási hely inicializálását szolgálja. Négyszavas (8 byte vagy 64 bit) helyfoglalást tesz lehetővé.

Formátum: *(változó)*DQ *(kifejezés)*

Megjegyzés: A változó elnevezése opcionális. Kifejezésként vesszővel elválasztott konstansokból álló táblázat szerepelhet. Két további lehetőség a kérdőjel, ami csak a memóriahelyet foglalja le, és a DUP utasítás, amivel egy bizonyos érték többszörös beléptetése érhető el. A DQ-val definiált változó 0FFFFFFFFFFFFFFFH-ig (decimálisan 0-tól 18446744073709551615-ig) képes egész számok tárolására. Ha a decimális pont, vagy exponenciális jelölés szerepel, akkor lebegőpontos szám keletkezik.

Példa:

```
THIS    DQ      100,2000,30000,400000,5000000,60000000
        DQ      700000000,8000000000,90000000000
IS      DQ      1000 DUP (45.678)
VERY    DQ      -6.345E-5,+5.0001E45
LARGE   DQ      7989H*12H
```

DT

(DT pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A DT (Define Tenbyte = tíz byte definiálása) pszeudo művelet változó, vagy táblázat definiálását, ill. egy tárolási hely inicializálását szolgálja. A 80287/80387-es számok esetében tízbyte-os (80 bites) pakolt decimális helyfoglalást tesz lehetővé.
Formátum: *(változó)*DT *(kifejezés)*
Megjegyzés: A változó elnevezése opcionális. Kifejezésként vesszővel elválasztott konstansokból álló táblázat szerepelhet. Két további lehetőség a kérdőjel, ami csak a memóriahelyet foglalja le, és a DUP utasítás, amivel egy bizonyos érték többszörös beléptetése érhető el. A DT 0-tól 9999999999-ig pakolt decimális számok tárolására is alkalmas. Ha decimális pont vagy exponenciális jelölés szerepel, akkor lebegőpontos szám keletkezik.
Példa:

```
THIS    DT      100,2000,30000,400000,5000000,60000000
        DT      700000000
IS      DT      1000 DUP (45.678)
VERY    DT      -6.345E-5,+5.0001E45
LARGE   DT      7989*12
```

DW

(DW pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A DW (Define Word = szó definilálása) pszeudo művelet egy változó vagy egy táblázat definiálását, ill. egy tárolási hely inicializálását szolgálja. Kétbyte-os (16 bites) helyfoglalást tesz lehetővé.
Formátum: *(változó)*DW *(kifejezés)*
Megjegyzés: A változó elnevezése opcionális. Kifejezésként vesszővel elválasztott konstansokból álló táblázat szerepelhet. Két további lehetőség a kérdőjel, ami csak a memóriahelyet foglalja le, és a DUP utasítás, amivel egy bizonyos érték többszörös beléptetése érhető el. A DW 0-tól 0FFFFH-ig (decimálisan 0-tól 65535-ig) alkalmas a számok tárolására.
Példa:

```
TEST1   DW      1234
TEST2   DW      0ABCDH
TEST3   DW      1,22,333,4444
TEST4   DW      23-12
TEST5   DW      23 DUP (100H)
TEST6   DW      50 DUP (?)
```

ELSE

(ELSE feltételes pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az ELSE pszeudo művelet - ha bármelyik más feltételes pszeudo művelettel kapcsolatban használjuk - műveletek közötti választást tesz lehetővé.
Formátum: ELSE (operandus nélkül)
Megjegyzés: Minden ELSE pszeudo műveletet egy IFxx-nek kell megelőznie. A műveletek tetszőleges mértékig egymásba ágyazhatók.
Példa:

```
IFNDEF LIB              ;HA A KONYVTAR NINCS DEFINIALVA,
INCLUDE B:MACLIB.MAC    ;TOLTSD BE A B:MACLIB.MAC FILE-T,
ELSE                    ;EGYEBKENT
INCLUDE B:NEWMAC.LIB    ;TOLTSD BE A B:NEWMAC.LIB FILE-T
ENDIF                   ;VEGE
```

END

(END pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az END pszeudo műveletre a forrásprogram végén van szükség. Ha az END-del egy feltételes kifejezés is áll, akkor ez a kifejezés a kód belépési pontjára mutat.
Formátum: END *(opcionális_kifejezés)*
Megjegyzés: Ha több tárgymodult szerkesztünk össze, akkor csak egyikben szerepelhet DOS belépési pontra mutató opcionális kifejezés. Ha nem adunk meg belépési pontot, akkor azt a szerkesztőprogram az első szerkesztendő tárgymodulból azonosítja be.
Példa:

```
;A program valamennyi resze ez elott a kod elott all
        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

ENDIF

(ENDIF feltételes pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az ENDIF feltételes pszeudo művelet az IFxx feltételes pszeudo művelet párja, tehát minden IFxx-hez egy ENDIF tartozik.
Formátum: ENDIF(operandus nélkül)
Megjegyzés: Nincs
Példa:

```
IFNDEF LIB              ;HA A KONYVTAR NINCS DEFINIALVA,
INCLUDE B:MACLIB.MAC    ;TOLTSD BE A B:MACLIB.MAC FILE-T,
ELSE                    ;EGYEBKENT
INCLUDE B:NEWMAC.LIB    ;TOLTSD BE A B:NEWMAC.LIB FILE-T
ENDIF                   ;VEGE
```

ENDM

(ENDM makró pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az ENDM pszeudo művelet a MACRO, REPT, IRP és IRPC pszeudo műveletek lezárását szolgálja.
Formátum: ENDM(operandus nélkül)
Megjegyzés: Ehhez a pszeudo művelethez nem kell mezőnevet hozzárendelnünk.
Példa:

```
CLEARSCREEN     MACRO       ;AZ EPPEN ERVENYES KEPERNYO TORLESE
        MOV     CX,0
        MOV     DX,2479H
        MOV     BH,7
        MOV     AX,0600H
        INT     10H
        ENDM
```

ENDP

(ENDP pszeudo művelet)

Assemblerek: IBM Microsoft és Speedware
Leírás: Az ENDP pszeudo művelettel zárjuk le az egyes eljárásokat.
Formátum: *(az_eljárás_neve)*ENDP
Megjegyzés: Mezőnév megadása szükséges.
Példa:

```
;A program valamennyi resze megelozi ezt a kodot
       RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
       END     MYPROC          ;A PROGRAM VEGE
```

ENDS

(ENDS pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az ENDS pszeudo művelettel jelezzük a struktúrák (STRUC) vagy szegmensek végét.
Formátum: *(a_szegmens_vagy_struktúra_neve)*ENDS
Megjegyzés: A szegmens vagy struktúra nevét mindig meg kell adni.
Példa:

```
;A program valamennyi resze megelozi ezt a kodot
       RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
       END     MYPROC          ;A PROGRAM VEGE
```

EQU

(EQU pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az EQU pszeudo művelettel egy változó értékadása végezhető el.
Formátum: *(változó)*EQU *(érték)*
Megjegyzés: Az EQU pszeudo művelet annyiban tér el az (=) pszeudo művelettől, hogy nem teszi lehetővé a változók újradefiniálását. Az EQU a STRUC definicióval nem használható együtt.
Példa:

```
VALUE1  EQU     1234H   ;16 BITES SZAM
VALUE2  EQU     4.56E5  ;LEBEGOPONTOS SZAM
VALUE3  EQU     [SI+6]  ;INDEX
VALUE4  EQU     DAA     ;UTASITAS
```

EVEN

(EVEN pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az EVEN pszeudo művelet az adatsorozat páros értékű határára állítja be a programszámlálót.
Formátum: EVEN(operandus nélkül)
Megjegyzés: Az EVEN pszeudo műveletnek nincs hatása, ha a a programszámláló már eleve páros határon áll. Ellenkező esetben az EVEN egy NOP műveletet hajt végre, annak érdekében, hogy a programszámláló páros sorszámú adatra mutasson.

Példa: Ha a programszámláló értéke pl. 0023H, akkor az EVEN hatására 0024H-ra változik. Ha a programszámláló 004AH-ra mutat, az EVEN nem változtatja meg az értékét.

EXITM

(EXITM pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az EXITM pszeudo művelet egy bizonyos feltételre vonatkozó pszeudo művelet és vizsgálat eredményétől függően kiléptet az ismétlésből, vagy a makróból. Az EXITM a REPT, IRP, IRP, IRPC és MACRO utasításokkal használható együtt.
Formátum: EXITM(operandus nélkül)
Megjegyzés: Blokkok egymásba ágyazása engedélyezett. Az EXITM csak azt a blokkot zárja le, amelyben szerepel.
Példa:

```
;A program egy resze megelozi ezt a programreszt

IFE     VALUE=34H       ;HA VALUE=34H, A FELTETEL IGAZ
EXITM                   ;KILEPES, HA AZ ELOBBI FELTETEL IGAZ,
ELSE                    ;EGYEBKENT
INCLUDE B:NEWMAC.LIB    ;B:NEWMAC.LIB FILE BETOLTESE
ENDIF                   ;IFXX VEGE
ENDM
```

EXTRN

(EXTRN pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az EXTRN pszeudo művelet azonosítja azokat az éppen érvényes programmodulban használt szimbólumokat vagy változókat, amelyeknek attributumait egy másik modulban definiáltuk.
Formátum: EXTRN *(név:típus)*
Megjegyzés: Az éppen érvényes programmodulban ezeket a szimbólumokat, vagy változókat EXTRN-ként kell deklarálnunk, míg a másik programmodulban PUBLIC-ként. Az EXTRN pszeudo műveletet vagy abban a szegmensben kell elhelyeznünk, ami a programmodult tartalmazza, vagy valamennyi szegmensen kívül. A bevitel típusa lehet BYTE, WORD (szó), DWORD (duplaszó) QWORD (négyesszó) TBYTE (tízbyte), NEAR (közeli), FAR (távoli) vagy ABS (abszolút).
Példa:

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
        PUBLIC  VALUE1,VALUE2,ANSWER
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ;DS ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

;Itt kovetkezik a program tovabbi resze

;Egy masik szegmens

        EXTRN   VALUE1,VALUE2,ANSWER
NUCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK

;Itt kovetkezik a program tovabbi resze
```

GROUP

(GROUP pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A GROUP pszeudo művelet a megadott szegmenseket egy aktuális 64 kbyte-os csoportba szervezi.
Formátum: *(név)*GROUP *(a_szegmens_neve)*
Megjegyzés: A névoperandus az azonosítója valamennyi olyan szegmensnek, amelyek egy csoportot alkotnak. Az azonosítónak egyedinek kell lennie, vagyis valamennyi szegmenscímkétől különböznie kell. A SEGMENT pszeudo művelettel, ill. egy SEG változóval, vagy cimke operátorral lehet a (szegmens_neve) operandus értékadását elvégezni.
Példa:

```
MEGA    GROUP   SMALL1,SMALL2
SMALL1  SEGMENT PARA 'CODE'
        ASSUME  CS:MEGA

;A szegmenshez tartozo valamennyi kodot ide kell elhelyezni

SMALL1  ENDS
SMALL2  SEGMENT PARA 'CODE'
        ASSUME  CS:MEGA

;A szegmenshez tartozo valamennyi kodot ide kell elhelyezni

SMALL2  ENDS
```

IF, IFE, IF1, IF2, IFDEF, IFNDEF IFB, IFNB, IFIDN, IFDIF

(IFxx feltételes pszeudo művelet)

Assemblerek: IBM, Microsoft és Speadware
Leírás: Az IFxx pszeudo művelet a további műveletek végrehajtása előtt egy előírt feltételt ellenőriz. Az ELSE pszeudo művelettel együtt használva különböző műveletek közötti választást tesz lehetővé.
Formátum: IFxxx *(operandus)*
(további_kódok)
ENDIF
Megjegyzés: A feltételes IFxxx pszeudo műveletek tetszőlegesen egymásba ágyazhatók. Az IFxxx valamennyi operandusáról az assemblernek az első fordítási menetben tudomást kell szereznie.

Példa:

```
IF VAR1                 ;ERTEKE IGAZ, HA VAR1 NEM ZERUS
IFE VAR1                ;ERTEKE IGAZ, HA VAR1 ZERUS
IF1(operandus nelkul)   ;ERTEKE IGAZ, HA AZ ASSEMBLER ELSO
                        ;FORDITASI MENETEROL VAN SZO
IF2(operandus nelkul)   ;ERTEKE IGAZ, HA AZ ASSEMBALER MASODIK
                        ;FORDITASI MENETEROL VAN SZO
IFDEF NEWVAL            ;ERTEKE IGAZ, HA DEFINIALT SZIMBOLUMROL
                        ;VAGY VALTOZOROL VAN SZO
IFNDEF OLDVAL           ;ERTEKE IGAZ, HA DEFINIALATLAN SZIMBOLUMROL
                        ;VAGY VALTOZOROL VAN SZO
IFB<OPERANDUS>          ;ERTEKE IGAZ, HA AZ <OPERANDUS> URES
IFNB<OPERANDUS>         ;ERTEKE IGAZ, HA AZ <OPERANDUS> NEM URES
IFIDN<OPERAND1>,        ;ERTEKE IGAZ, HA <OPERAND1> EGYENLO
 <OPERAND2>             ;<OPERAND2>-VEL
IFDIF<OPERAND1>,        ;ERTEKE IGAZ, HA <OPERAND1> NEM EGYENLO
 <OPERAND2>             ;<OPERAND2>-VEL
```

INCLUDE

(INCLUDE pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az INCLUDE pszeudo művelet egy másik file lefordítás nélküli forráskódját hozzákapcsolja az éppen érvényes file-hoz.
Formátum: INCLUDE *(meghajtó/útvonal/file-név.ext)*
Megjegyzés: Ez a pszeudo művelet különösen a makrokönyvtárak programjainak betöltésekor hasznos. A fordítás nélküli kód közvetlenül az INCLUDE pszeudo művelet után kapcsolódik a programhoz. Egymásba ágyazás lehetséges, de csak a DOS 2.0 vagy ennél újabb DOS változatoknál. A CONFIG.SYS file-paraméterének értékét alapértelmezés szerinti értéknél (8) nagyobbra kell beállítani. Ha a lefordított file-t kilistázzuk, az INCLUDE-ból származó valamennyi kódot a program egy "C" karakterrel jelöli meg a 30. oszlopban a könnyebb azonosítás és a bővítmények elhelyezése miatt.
Példa:

```
IF1
INCLUDE B:MACLIB.MAC
INCLUDE C:REGDIS.MAC
ENDIF
```

IRP

(IRP pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az IRP pszeudo művelet az IRP és ENDM közötti utasítások megismétlését szolgálja. Az ismétlések számát az operandus-listában adhatjuk meg. Az ismétlések során az operandus-listában található éppen érvényes tételt a kód blokkjában szereplő formális paraméterbe helyettesíti be a rendszer.
Formátum: IPR *(formális paraméter), <operanduslista>*
Megjegyzés: Operanduslistára szükség van. Ha nem adunk meg operandust(< >), akkor a blokkot csak egyszer ismétli meg a rendszer.

Példa:

```
IRP     VAL,<2,4,6,8>   ;A 2,4,6,8 SZAMOK OPERANDUSLISTAJA
DW      VAL             ;A VAL VALTOZOBA UJ SZAM KERUL
ENDM                    ;A BLOKK VEGE
```

IRPC

(IRPC pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az IRPC pszeudo művelet az IRPC és az ENDM közötti műveletek megismétlését szolgálja. Az ismétlések számát karakterlánc-operandussal adhatjuk meg. Az egyes ismétlések során a *karakterlánc* éppen érvényes karakterét a kód blokkjában szereplő *formális paraméter*-be helyettesíti be a rendszer.
Formátum: IRPC *(formális paraméter),karakterlánc*
Megjegyzés: A karakterláncot nem szükséges kisebb-nagyobb jelek közé foglalni.
Példa:

```
IRPC    DATA,2468       ;A 2468 KARAKTEREKBOL ALLO KAR.LANC
DB      DATA            ;DATA-BA MINDEN MENETBEN UJ SZAM KERUL
DB      DATA*2          ;DATA ERTEKE MINDEN MENETBEN 2-VEL
                        ;SZORZODIK
ENDM                    ;A BLOKK VEGE
```

LABEL

(LABEL pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A LABEL pszeudo művelet egy címkenév attributumának definiálását teszi lehetővé.
Formátum: *(név)*LABEL *(típus)*
Megjegyzés: Névként bármilyen érvényes elnevezés szerepelhet. Típusként a következők valamelyike írható elő:

BYTE
WORD (szó)
DWORD (duplaszó)
QWORD (négyesszó)
TBYTE (tízbyte)

Példa:

```
MYTABLE LABEL   BYTE
TABLE   DW      1234,5678,1111,4545,6778,4598

;A kodot ide kell beilleszteni

        MOV     CL,MYTABLE[4]   ;11 ELHELYEZESE CL-BEN
```

.LALL, .SALL, .XALL

(._ALL listázó pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Ez a pszeudo művelet a program szövegének listázását irányítja.
Formátum: .LALL(operandus nélkül)
Megjegyzés: A .LALL a .TFCOND, .LFCOND, .SFCOND-dal együtt használva kilistázza valamennyi MACRO és feltételes blokk kiterjesztését. A .SALL a kiterjesztett makrók listázá-

sát elnyomja. Az .XALL (ez az alapértelmezés) csak azt a forráskódot listázza, amiből az aktuális tárgykód elkészül.

Példa:

```
.SALL

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

.LFCOND

(.LFCOND listázó pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az LFCOND pszeudo művelet azokat a feltételes blokkokat listázza ki, amelyek kiértékelése hamis.
Formátum: .LFCOND(operandus nélkül)
Megjegyzés: Ez a feltétel vagy a .SFCOND, vagy a .TFCOND pszeudo művelettel zárható le.
Példa:

```
.LFCOND

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

.LIST, .XLIST

(.LIST, .XLIST listázó pszeudo műveletek)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az .LIST és .XLIST pszeudo műveletekkel a listafileba való kivitelt vezérelhetjük.
Formátum: .LIST(operandus nélkül)
Megjegyzés: Az .LIST alapértelmezés szerinti feltétel a forrás- és a tárgykód listázását teszi lehetővé, még az .XLIST felfüggeszti a listázást.
Példa:

```
.XLIST

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

LOCAL

(LOCAL makro pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Ha a LOCAL pszeudo műveletet használjuk, akkor az egyedi szimbólumok váltják fel

a cimkék előfordulását a makrón belül, amely által lehetővé válik, hogy a makró egy adott szegmensen belül egynél többször is kiterjeszthető legyen.
Formátum: LOCAL *(formális_paraméter_lista)*
Megjegyzés: A LOCAL pszeudo műveletnek a makrón belül az első utasításnak kell lennie. Többszörös LOCAL utasítások is lehetségesek.
Példa:

```
DELAY   MACRO                   ;;SZOFTVERREL MEGOLDOTT KESLELTETES
        LOCAL P1,P2             ;;P1,P2 LOKALIS CIMKEK DEKLARALASA
        MOV     DX,15H          ;;KB. 5 MP-ES KESLELTETES A PC-N
                                ;;ES XT-N
P1:     MOV     CX,0FF00H       ;;CX FELTOLTESE A DEC. 65280-VAL
P2:     DEC     CX              ;;CX CSOKKENTESE 1-GYEL
        JNZ     P2              ;;ZERUS? HA NEM, AKKOR P2-RE UGRAS
        DEC     DX              ;;DX CSOKKENTESE
        JNZ     P1              ;;ZERUS? HA NEM, AKKOR P1-RE UGRAS
ENDM
```

MACRO

(MACRO pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A MACRO pszeudo művelet – a szubrutinhoz hasonlóan – önálló programblokkok ismételt hívását teszi lehetővé paraméterek használatával.
Formátum: *(a_makro_neve)*MACRO *(formális_paraméter_lista)*
Megjegyzés: A MACRO pszeudo művelet a következő három fő részből áll:
1. Fejrész, ami a MACRO nevét, a MACRO pszeudo műveletet és egy formális paraméterlistát tartalmaz.
2. Törzsrész, ami a makrokódokat tartalmazza.
3. Zárórész, ami csupán csak az ENDM lezáró pszeudo műveletből áll.

A listákban egy + jel figyelmeztet bennünket arra, hogy makrókiterjesztésről van szó. A formális paraméterlista elemeinek pozíciója lényeges, mivel a behelyettesítések az előírt sorrendben történnek meg a makrokódban. A formális paraméterlistában tárolt, de használaton kívüli elemeket a rendszer figyelmen kívül hagyja. Ha túl kevés elemet szerepeltetünk, akkor a hiányzó tagok helyére nulla kerül.
Példa:

```
CLEARSCREEN MACRO               ;;A KEPERNYO LETORLESE
        MOV     CX,0            ;;PARAMETEREK NINCSENEK
        MOV     DX,2479H
        MOV     BH,7
        MOV     AX,0600H
        INT     10H
        ENDM

PRINTCHAR MACRO TEXT            ;;A 'TEXT'-BEN ATADOTT KAR.LANC
                                ;;KINYOMTATASA
        MOV     DX,OFFSET TEXT  ;;A KAR.LANC AZ ADATSZEGMENSBEN VAN
        MOV     AH,9            ;;NYOMTATAS A '$' ELVALASZTO KAR.-IG
        INT     21H             ;;NYOMTATAS AZ EPPEN ERVENYES
                                ;;KURZORPOZICIOTOL KEZDVE
        ENDM                    ;;A MAKRO VEGE
```

.MSFLOAT

(.MSFLOAT pszeudo művelet)

Assemblerek: Speedware
Leírás: Az .MSFLOAT pszeudo művelet valamennyi lebegőpontos számot Microsoft lebegőpontos formátumba alakítja.
Formátum: .MSFLOAT (operandus nélkül)
Megjegyzés: A .MSFLOAT egyszeres és dupla pontosságú lebegőpontos formátumokat támogat. Az egyszeres pontosságú számok négy byte-ot foglalnak el. Az első 23 bit az alap, az utolsó 8 bit a kitevő. A duplapontos számok nyolc byte-ot foglalnak el. Ebben az esetben az első 55 bit az alap, és az utolsó 8 bit a kitevő. Az egyszeres pontosságú számábrázolásnál 6 vagy 7 szignifikáns decimális számjegy, míg a dupla pontosságú számábrázolásnál 16 szignifikáns decimális számjegy kifejezésére van lehetőségünk.
Példa:

```
.MSFLOAT
        VALUE1  DD      1.234E-23
        VALUE2  DQ      67.65234987432E35
```

NAME

(NAME pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A NAME pszeudo műveletet a modulok elnevezésekor kell használni.
Formátum: NAME *(a_modul_neve)*
Megjegyzés: Az assembler a következő preferencia szerint ad a modulnak nevet:

1. NAME pszeudo művelet
2. A TITLE első hat karaktere vagy
3. A forrásfile nevének első hat karaktere

Példa:

```
NAME BEEPER

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAKELYEZESE A VEREMRE
```

ORG

(ORG pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: Az ORG pszeudo művelet a helyzetszámláló beállítását, és ezzel a kód kezdőcímének kijelölését végzi.
Formátum: ORG *(érték_vagy_kifejezés)*
Megjegyzés: A dollárjel ($) a helyzetszámláló éppen érvényes értékeként használható. A kifejezésről az assemblernek a fordítás első menetében tudomást kell szereznie, és 16 bites abszolut számként kell kiértékelnie.
Példa:

```
ORG     100H
ORG     $
ORG     $+100H
```

PAGE

(PAGE pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A PAGE pszeudo művelet, amelyet a forrásfile-ban kell elhelyezni, a listázandó oldalak szélességét és hosszúságát állítja be.
Formátum: PAGE *(opcio_1)(opcio_2)*
Megjegyzés: A PAGE pszeudo művelet opciók nélküli használata lapdobást jelent. A fejezetszám növekszik, ha a PAGE+-szal találkozik a rendszer. Az első opcióval a kinyomtatandó sorok száma adható meg 10-től 255-ig. Az alapértelmezés szerinti érték 58. A második opció a kinyomtatott sorok szélességét szabályozza. A megadható érték a decimális 60 és 132 között lehet. Az alapértelmezés szerinti érték 80. A PAGE pszeudo művelet nem inicalizálja a nyomtatót. Erről külön kell gondoskodnunk.
Példa:

```
PAGE                ;LAPDOBAS
PAGE    ,132        ;AZ ALAPERTELMEZES SZERINTI NYOMTATASI
                    ;SZELESSEG BEALLITASA 132-RE
PAGE    20,132      ;EGY OLDALRA 20 SOR ES 132 KARAKTER KERUL
```

PROC

(PROC pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A PROC pszeudo művelet a forráskód egy blokkjának beazonosítását szolgálja. Legalább egy FAR attributummal ellátott PROC pszeudo művelettel általában valamennyi program rendelkezik.
Formátum: *(az_eljárás_elnevezése)*PROC *(típus)*
Megjegyzés: A PROC pszeudo művelettel azonosított programblokk soron belül végrehajtható vagy hívható. (Ha külső CALL-t alkalmazunk, akkor a PROC nevének PUBLIC-ként kell szerepelnie.) Ha a PROC az .EXE file belépési pontja, vagy ha CS-nek más értéke van, a típusnak FAR-nak kell lennie. Csak akkor lehet a PROC típusa NEAR, ha az eljárást szegmensen belül hívjuk. A típustól függően a RET szegmensen belüli vagy szegmensek közötti visszatérést szolgáltat.
Példa:

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

        CALL    CURSOR          ;EGY NEAR TIPUSU ELJ. HIVASA

;Itt kovetkezik a kod

CURSOR  PROC    NEAR            ;ELJARAS UGYANAZON A SZEGMENSEN BELUL
        MOV     BX,1            ;A PARAMETEREK BEALLITASA
        MOV     AH,2
        INT     10H
        RET                     ;A CIM LEEMELESE
                                ;A VEREMROL
```

PUBLIC

(PUBLIC pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A PUBLIC pszeudo művelet lehetővé teszi, hogy a forráskódban szereplő szimbólumokat az ezzel összeszerkesztett más programok is használják.
Formátum: PUBLIC *(szám,változó_vagy_címke)*
Megjegyzés: Nincs
Példa:

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
        PUBLIC  VALUE1,ANSWER,SUM
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

PURGE

(PURGE pszeudo művelet)

Assemblerek: IBM,és Microsoft
Leírás: A PURGE pszeudo művelet törli a makrodefiníciót. A makro által előzőleg elfoglalt hely újra használható.
Formátum: PURGE *(makro_elnevezések)*
Megjegyzés: A makrót az újradefiniálása előtt nem kell törölni.
Példa: PURGE CLEARSCREEN

.RADIX

(.RADIX pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A .RADIX pszeudo művelettel 2-től 16-ig a számrendszer alapját változtathatjuk meg. Az alapértelmezés szerinti érték 10.
Formátum: .RADIX *(érték_decimális_formátumban_előírva)*
Megjegyzés: A .RADIX DB-t és DW-t közvetlenül befolyásolja, azonban nincs hatással DD-re, DQ-ra és DT-re. Helyes eredmény a következő toldalékokkal biztosítható: B – Bináris
D – Decimális
Q – vagy O – Oktális
H – Hexadecimális
R – Hexadecimális valós
Példa:

```
.RADIX  16
VALUE   DB      120             ;A VALUE VALTOZOT 120 HEXARA
                                ;ALLITJUK BE
.RADIX  2
NUM     DB      10110011        ;A NUM VALTOZOT AZ 10110011
                                ;BINARIS SZAMRA ALLITJUK BE
.RADIX  10                      ;AZ ALAPERTELMEZES SZERINITI
                                ;10-ES ALAP BEALLITASA
ANS     DB      120H            ;ANS BEALLITASA 120H-RA
```

RECORD

(RECORD pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A RECORD pszeudo művelet bitmintát hoz létre byte-ok és szavak bites pakolásához. A rekord neve a tárolási hely lefoglalásakor pszeudo műveletté válik.
Formátum: *(a_rekord_neve)*RECORD *(a_mező_neve:szélesség(=kifejezés))*
Megjegyzés: A RECORD pszeudo műveletnél mind a rekordnevet, mind a mezőnevet meg kell adni. A mezőszélesség 1-től 16-ig terjedő szám lehet és a mezőnév által definiált bitek számát adja meg. A mezők jobbra igazítottak. A kifejezés a mező kezdőértékét tartalmazza.
Példa:

```
MYRECD          RECORD  H:1,E:2,L:3,P:6
;Mezok
HEEL LLPP PPPP
;TAROLASI HELY LEFOGLALVA
STORE1          MODULE  <1,2,3,4>
;H=1, E=2, L=3, P=4
```

REPT

(REPT pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A REPT pszeudo művelettel a REPT és az ENDM közötti kód meghatározott számban megismételhető.
Formátum: REPT *(érték_vagy_kifejezés)*
Megjegyzés: A kód blokkjának nem kell a MACRO-n belül lennie.
Példa:

```
;A tobbi kodok ezt a kodot megelozik

VALUE   =       10              ;VALUE VALTOZO BEALLITASA 10-RE
        REPT    3               ;REPT BEALLITASA 3-RA
        IFE     STORE=VALUE     ;HA STORE=VALUE
        EXITM                   ;KILEPES, EGYEBKENT ISMETLES
        ENDIF                   ;VEGE, HA
        ENDM                    ;VEGE AZ ISMETELENDO RESZNEK
```

SEGMENT

(SEGMENT pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A SEGMENT pszeudo művelethez a program valamennyi kódját hozzárendelhetjük. Ugyanazon a programon belül általában több szegmenst is használunk.
Formátum: *(a_szegmens_neve)*SEGMENT *(beállítási_típus)(kombinációs _típus)(osztály)*
Megjegyzés: A sorbaállítási típus PAGE, PARA, WORD vagy BYTE lehet. PAGE típus előírása esetén a szegmens egy oldalhatáron kezdődik, vagyis egy 256-tal osztható címnél, amelyben a cím LSB-je 00H-val egyenlő. A PARA típus esetén a szegmens egy paragrafus határán kezdődik, vagyis egy olyan címnél, ami 16-tal osztható és amelynek legkisebb helyértékű bitje 0H-val egyenlő. A WORD típus esetén a szegmens egy szóhatáron kezdődik, vagyis egy páros címnél, amelyben a legkisebb helyértékű bit 0H. A BYTE típusnál a szegmens bármilyen byte-határon kezdődhet. A kombinációs típus PUBLIC, COMMON, AT *(kifejezés)*, STACK vagy MEMORY (jelenleg még nincs támogatva) lehet. A PUBLIC azt jelzi, hogy a szegmens a szerkesztés során más szegmensekhez csatlakozik majd. A COMMON azt jelzi, hogy ez a szegmens, és a többi összeszerkesztendő azonos névvel ellátott szegmens ugyanazon

a címen fog kezdődni. Az AT *(kifejezés)* azt jelzi, hogy a szegmens a kifejezés által megadott paragrafushatáron kezdődik majd. Erre a szegmensre nem keletkezik kód. A STACK azt írja elő, hogy a szegmens része lesz a futási idejű veremnek. A STACK-re csak az .EXE file-ok esetében van szükség.

Példa:

```
STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
INFO    DB      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
ANS     DB      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
```

.SFCOND

(.SFCOND listázó pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A .SFCOND pszeudo művelettel a hamisként kiértékelt feltételes blokkok listázása leállítható.
Formátum: .SFCOND(operandus nélkül)
Megjegyzés: Az .SFCOND a hamisként kiértékelt feltételes blokkokat nem listáztatja ki. A .LFCOND ezt az állapotot megfordítja.
Példa:

```
.SFCOND
IF1
        INCLUDE B:MACLIB.MAC
ENDIF

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
        PUBLIC  VALUE1,ANSWER,SUM
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

STRUC

(STURC pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A STRUC pszeudo művelet hasonló a RECORD-hoz, azzal a különbséggel, hogy a STRUC többbyte-os kapacitással rendelkezik.
Formátum: *(a_struktura_neve)*STRUC
Megjegyzés: A struktura neve pszeudo műveletté válik, amit a tárolási hely meghatározására

használ fel a rendszer. Logikailag osztott mezők nem írhatók felül, csak azok, amelyekben nem végeztünk logikai bontást. Az a mező, ami karakterláncot tartalmaz, egy másik karakterlánccal felülírható.
Példa:

```
MYSTRUC STRUC
AFIELD  DB      8       ;FELULIRHATO
BFIELD  DB      1,2,3   ;NEM IRHATO FELUL
CFIELD  DB      'SZABO EVA';FELULIRHATO
MYSTRUC ENDS

;Lefoglalas

BLUE    MYSTRUC<4,,'PAPP EVA'>

;Felulirja a 1. es a 3. mezoket
```

SUBTTL

(SUBTTL pszeudo művelet)

Assemblerek: IBM, Microsoft és Speedware
Leírás: A SUBTTL pszeudo művelet az alcím listázását teszi lehetővé közvetlenül a cím (TITLE) után.
Formátum: SUBTTL *(karakterlánc)*
Megjegyzés: A SUBTTL-ben 60 karakter helyezhető el. A SUBTTL-ek száma nincs korlátozva. A SUBTTL karakterlánc nélküli használatakor nem történik lapdobás.
Példa:

```
TITLE   LEMEZFORMALAS
SUBTTL  HIBAKODOK ELLENORZESE

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

.TFCOND

(.TFCOND pszeudo művelet)

Assemblerek: IBM, Microsoft
Leírás: A .TFCOND átkapcsolja azokat az alapértelmezés szerinti feltételeket, amelyek a hamis feltételek esetén a listázásokat vezérlik. A művelet lényegében a .SFCOND és .LFCOND hatását szünteti meg.
Formátum: .TFCONT(operandus nélkül)
Megjegyzés: A .TFCOND átkapcsolja az éppen érvényes és alapértelmezés szerinti feltételeket a nem alapértelmezés szerinti feltételekre.
Példa: .TFCOND

Assemblerek: IBM, Microsoft és Speedware
Leírás: A TITLE pszeudo művelet lehetővé teszi, hogy minden egyes listázandó oldal második sorába beírjunk egy címet.
Formátum: TITLE *(karakterlánc)*
Megjegyzés: Csak egy cím (TITLE) megadása lehetséges. Ha nem adunk meg címet, akkor a rendszer a forrásfile nevéből készít egyet. A karakterlánc-operandus maximálisan 60 karakter lehet.
Példa:

```
TITLE   LEMEZFORMALAS
SUBTTL  HIBAKODOK ELLENORZESE

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
```

7. MAKRÓK, ELJÁRÁSOK, KÖNYVTÁRAK

A makrók, eljárások és könyvtárak lehetővé teszik a programozó számára, hogy az előzetesen már hibamentesített kódokat meghívja és használja. A programkészítési eljárás idejét ezzel meglehetősen felgyorsíthatjuk, mert nem kell a már korábban megírt rutinokat ismételten megírnunk (újra feltalálni a kereket), ha új programot készítünk. Az assembly nyelven programozó szakemberek közül egyesek mindent makrókkal szeretnek megoldani, míg mások inkább az eljárásokat, vagy könyvtárakat használják. Mindegyik módszernek vannak előnyei és hátrányai. Ebben a fejezetben a a következő témaköröket tárgyaljuk:

- Makrók
- Eljárások
- Könyvtárak
- A rendelkezésre álló lehetőségek összehasonlítása.

7.1 MAKRÓK

A makrók szintaxisát az előbbi fejezetben ismertettük, az igazi hasznukat azonban csak most mutatjuk be. A makró egy olyan pszeudo művelet, amellyel önálló assembly műveletek megteremtése vagy a gyakran használt assembly kódok programba való belefoglalása válik lehetővé. A makrokódok felhasználásához csak a makró nevének megadása szükséges. Valahányszor egy makrót megnevezünk a programban, az assembler az aktuális makrokódot bemásolja és beilleszti a programnak arra a helyére, ahol a makró nevét megadjuk. Ezt makrokiterjesztésnek nevezik (macro expansion). A makrók tehát soron belül kerülnek végrehajtásra, a program folyamatának megszakítása nélkül.

Makrókat az aktuális programon belül is létrehozhatunk vagy a már elkészített könyvtárból is behívhatunk. A makrokönyvtár egy olyan makrókból álló file, ami az éppen érvényes program fordítási ideje alatt hívható. A makró vagy a makrokönyvtár kódjai nincsenek lefordítva! Ez az egyik szempont, amiben a makrokönyvtár különbözik a szerkesztés ideje alatt elérhető assembler könyvtártól. A 8088/80386-os processzorcsalád fejlesztését követniük kellett az ezekhez tartozó assembler programcsomagok fejlesztésének is. A korábbi assemblerek nem támogatták a 8087/80387-es mnemonikokat, és előfordulnak olyan assemblerek is, amelyek nem támogatják a 80286/80386-os védett virtuális üzemmódját. Ilyen esetekben makrókkal kell pótolnunk azokat az utasításokat, amelyeket az assembler nem szolgáltat a számunkra. Pl., ha egy olyan assemblerrel dolgozunk, ami nem támogatja a 8087-es processzor használatát, a 8087-es veremmutatójának növeléséhez a következő makrokódot kell létrehoznunk:

```
FINCSTP MACRO          ;;A VEREMMUTATO NOVELESE
WAIT                   ;;A 8087 ES 8088 SZINKRONIZALASA
ESC 0EH,DI             ;;A HELYES KODSOROZAT KIKULDESE
ENDM                   ;;A MAKRO VEGE
```

Ilyen makróra minden egyes 8087-es utasítás esetén szükség lenne, ez a technika azonban sok 8087-es utasítás alkalmazásánál és nyomon követésénél meglehetősen időigényes. Az IBM Makroassembler 2.0-s verziója, vagy a Microsoft Assembler 3.00-s verziója nem támogatja a 80286/80386-os processzor védett módú utasításait. Ha erre szükségünk van, akkor ezt makróműveletekkel kell megoldanunk. (A Speedware Assembler 1.02-es verziója támogatja a 26 védett vezérlőutasítást.) A makrókba foglalt kódsorozatra különösen akkor van szükség, ha azt egy programon belül sokszor akarjuk meghívni. Gyakori rutinok pl.: a képernyőtörlés, a kurzorpozícionálás, vagy az aritmetikai műveletek.

A makro elkészítése

A előző fejezetben egy makro szintaxisát már ismertettük. A lényeges dolgokat most csak átismételjük. A makro a következő három fő részből áll:

- fejrész
- törzsrész
- zárórész

A fejrész tartalmazza a makro nevét, magát a MACRO pszeudo műveletet és opcionálisan a formális paramétereket. A makro törzsrészében helyezzük el az aktuális kódokat, amelyek a makro nevének helyére kerülnek a hívó programban. A zárórésznek az ENDM utasítást kell tartalmaznia. A 7–1. ábrán olyan program listája látható, ami a makro használatát mutatja be.

```
 RGB MONITORRAL ELLATOTT 8088/80286-OS IBM GEPEKRE
 KESZITETT PROGRAM
;A MAKRO HASZNALATNAK BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
BACK    DB      2000 DUP (' ')
MYDATA  ENDS

DELAY   MACRO   TIME
        LOCAL   P1,P2           ;;P1 ES P2 LOKALIS CIMKEK
        PUSH    DX              ;;DX ES CX EREDETI TARTALMANAK
        PUSH    CX              ;;ELMENTESE
        MOV     DX,TIME         ;;A TIME VALTOZO TARTALMA DX-BE KERUL
P1:     MOV     CX,0FF00H       ;;CX-BE A 00FF00H SZAMLALO KERUL
P2:     DEC     CX              ;;A KESLELTETESI CIKLUS KEZDETE
        JNZ     P2              ;;HA CX<>0, CX CSOKKENTESE
        DEC     DX              ;;HA CX=0, DX CSOKKENTESE
```

```
JNZ     P1              ;;HA DX<>0, CX ISMETELT FELTOLTESE
POP     CX              ;;HA DX=0, CX ES DX EREDETI
POP     DX              ;;TARTALMANAK VISSZAALLITASA
ENDM                    ;;KILEPES A MAKROBOL
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,ES:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE

;A PROGRAMSZEGMENS 80*25 SZOKOZ KEPERNYORE IRASAVAL LETORLI
;A KEPERNYOT.MIVEL BL ERTEKET AZ EREDETIHEZ KEPEST MODOSITJUK,
;EZZEL A TELJES KEPERNYO SZINENEK MEGVALTOZTATASAT IS ELERJUK.
;A DELAY MAKRO HATASARA EZ A SZIN EGY BIZONYOS IDEIG MEGMARAD
;A KEPERNYON
        MOV     CX,08H          ;A CIKLUST 8-SZOR ISMETELTETJUK
        MOV     BL,00H          ;AZ EREDETI HATTER SZINENEK BEALLITASA
AGAIN:  LEA     BP,BACK         ;SZOKOZOKBOL ALLO KAR.LANC
                                ;MEGJELENITESE
        MOV     DX,0000         ;A KURZOR BEALLITASA A BAL FELSO SAROKBA
        MOV     AH,19           ;A KAR.LANC ATTRIBUTUMANAK BEALLITASA
        MOV     AL,1            ;KARAKTEREK KINYOMTATASA ES
                                ;KURZORPOZICIONALAS
        PUSH    CX              ;A CIKLUSSZAMLALO ELMENTESE
        MOV     CX,07D0H        ;2000 SZOKOZT IRUNK A KEPERNYORE
        INT     10H             ;A MEGSZAKITAS HIVASA
        DELAY   10              ;10 EGYSEGNYI KESLELTETES
        ADD     BL,10H          ;A HATTER SZINENEK MEGVALTOZTATASA
        POP     CX              ;AZ EREDETI CIKLUSSZAMLALO
                                ;VISSZAALLITASA
        LOOP    AGAIN           ;A MUVELET ELVEGZESE OSSZESEN 8-SZOR

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

7-1. ábra A makro használatának bemutatása

A program szerkezete az 5. fejezetben bemutatott programszer- ezetekkel azonos, a DELAY elnevezésű makro itt szerepel először. A makroszegmensre vonatkozó programrészlet a következő:

```
DELAY   PROC    NEAR
        PUSH    DX              ;DX ES CX EREDETI TARTALMANAK
        PUSH    CX              ;ELMENTESE
        MOV     DX,10           ;10 EGYSEGNYI KESLELTETES
P1:     MOV     CX,0FF00H       ;CX-BE A FF00H SZAMLALO KERUL
P2:     DEC     CX              ;A KESLELTETESI CIKLUS KEZDETE
        JNZ     P2              ;HA CX<>0, CX CSOKKENTESE
        DEC     DX              ;HA CX=0, DX CSOKKENTESE
```

```
JNZ     P1              ;;HA DX<>0, CX ISMETELT FELTOLTESE
POP     CX              ;;HA DX=0, CX ES DX EREDETI
POP     DX              ;;TARTALMANAK VISSZAALLITASA
ENDM                    ;;KILEPES A MAKROBOL
```

A makro neve DELAY, ha tehát ez az elnevezés szerepel egy programban, az assembler a hívó programnak erre a helyére bemásolja a DELAY makrotörzsét, a TIME változó tartalma pedig a DX regiszterbe kerül. A LOCAL deklarációval elkerülhető az ugyanarra a tárterületre szóló többszörös hivatkozás. Az önálló címkék automatikusan P1 és P2 helyére kerülnek. Ennek a makrónak a késleltetés a célja, amelyet a CX és DX regiszterek tartalmának folyamatos csökkentésével érünk el.

Az 5. fejezetben készítettünk egy olyan programot, amivel megváltoztattuk a képernyő hátterének színét és egy feliratot jelentettünk meg rajta. Az itt bemutatott technikát ebben a programban is hasznosítjuk. A háttér színe a program futása során nyolcszor változik meg, mivel a BL regiszter tartalmához minden alkalommal, valahányszor a program a ciklusba belép, 10H-t adunk. A DELAY makro határozza meg, hogy egy-egy szín meddig látható a képernyőn. A formális paraméter értékét az IBM AT 9 MHz-es frekvenciájának megfelelően állítottuk be. (Ha az Olvasó által használt gép ettől eltérő értékkel dolgozik, a változót ennek megfelelően módosítania kell.) A 7–2. ábrán a listafile látható.

```
                                ;RGB MONITORRAL ELLATOTT 8088/80286-OS IBM GEPEKRE
                                ;KESZITETT PROGRAM
                                ;A MAKRO HASZNALATNAK BEMUTATASA

0000                            STACK   SEGMENT PARA STACK
0000    40 [                            DB      64 DUP ('MYSTACK ')
           4D 59 53 54
           41 43 4B 20
                    ]

0200                            STACK   ENDS

0000                            MYDATA  SEGMENT PARA 'DATA'
0000  07D0 [                    BACK    DB      2000 DUP (' ')
           20
               ]

07D0                            MYDATA  ENDS

                                DELAY   MACRO   TIME
                                        LOCAL   P1,P2           ;;P1 ES P2 LOKALIS CIMKEK
                                        PUSH    DX              ;;DX ES CX EREDETI TARTALMANAK
                                        PUSH    CX              ;;ELMENTESE
                                        MOV     DX,TIME         ;;A TIME VALTOZO TARTALMA DX-BE KERUL
                                P1:     MOV     CX,0FF00H       ;;CX-BE A 00FF00H SZAMLALO KERUL
                                P2:     DEC     CX              ;;A KESLELTETESI CIKLUS KEZDETE
                                        JNZ     P2              ;;HA CX<>0, CX CSOKKENTESE
                                        DEC     DX              ;;HA CX=0, DX CSOKKENTESE
                                        JNZ     P1              ;;HA DX<>0, CX ISMETELT FELTOLTESE
                                        POP     CX              ;;HA DX=0, CX ES DX EREDETI
                                        POP     DX              ;;TARTALMANAK VISSZAALLITASA
                                        ENDM                    ;;KILEPES A MAKROBOL
```

```
0000                                MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
0000                                MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
                                            ASSUME  CS:MYCODE,ES:MYDATA,SS:STACK
0000  1E                                    PUSH    DS              ;A DS REGISZTER ELMENTESE
0001  2B C0                                SUB     AX,AX           ;AX TORLESE
0003  50                                    PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
0004  B8 ---- R                            MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
0007  8E C0                                MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE

                                    ;A PROGRAMSZEGMENS 80*25 SZOKOZ KEPERNYORE IRASAVAL LETORLI
                                    ;A KEPERNYOT.MIVEL BL ERTEKET AZ EREDETIHEZ KEPEST MODOSITJUK,
                                    ;EZZEL A TELJES KEPERNYO SZINENEK MEGVALTOZTATASAT IS ELERJUK.
                                    ;A DELAY MAKRO HATASARA EZ A SZIN EGY BIZONYOS IDEIG MEGMARAD
                                    ;A KEPERNYON
0009  B9 0008                              MOV     CX,08H          ;A CIKLUST 8-SZOR ISMETELTETJUK
000C  B3 00                                MOV     BL,00H          ;AZ EREDETI HATTER SZINENEK BEALLITASA
000E  8D 2E 0000 R                  AGAIN:  LEA     BP,BACK         ;SZOKOZOKBOL ALLO KAR.LANC
                                                                    ;MEGJELENITESE
0012  BA 0000                              MOV     DX,0000         ;A KURZOR BEALLITASA A BAL FELSO SAROKBA
0015  B4 13                                MOV     AH,19           ;A KAR.LANC ATTRIBUTUMANAK BEALLITASA
0017  B0 01                                MOV     AL,1            ;KARAKTEREK KINYOMTATASA ES
                                                                    ;KURZORPOZICIONALAS
0019  51                                   PUSH    CX              ;A CIKLUSSZAMLALO ELMENTESE
001A  B9 07D0                              MOV     CX,07D0H        ;2000 SZOKOZT IRUNK A KEPERNYORE
001D  CD 10                                INT     10H             ;A MEGSZAKITAS HIVASA
                                           DELAY   10              ;10 EGYSEGNYI KESLELTETES
001F  52                            +      PUSH    DX
0020  51                            +      PUSH    CX
0021  BA 000A                       +      MOV     DX,10
0024  B9 FF00                       +  ??0000: MOV  CX,0FF00H
0027  49                            +  ??0001: DEC  CX
0028  75 FD                         +      JNZ     ??0001
002A  4A                            +      DEC     DX
002B  75 F7                         +      JNZ     ??0000
002D  59                            +      POP     CX
002E  5A                            +      POP     DX
002F  80 C3 10                             ADD     BL,10H          ;A HATTER SZINENEK MEGVALTOZTATASA
0032  59                                   POP     CX              ;AZ EREDETI CIKLUSSZAMLALO
                                                                    ;VISSZAALLITASA
0033  E2 D9                                LOOP    AGAIN           ;A MUVELET ELVEGZESE OSSZESEN 8-SZOR

0035  CB                                   RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
0036                                MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
0036                                MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                            END                     ;A PROGRAM VEGE

Macros:

                N a m e                 Length

DELAY. . . . . .                        0006

Segments and groups:

                N a m e                 Size    align   combine class

MYCODE . . . . . .                      0036    PARA    NONE    'CODE'
MYDATA . . . . . .                      07D0    PARA    NONE    'DATA'
STACK. . . . . .                        0200    PARA    STACK
```

```
Symbols:

                N a m e                 Type    Value   Attr

AGAIN. . . . . . . . . . . . . .        L NEAR  000E    MYCODE
BACK . . . . . . . . . . . . . .        L BYTE  0000    MYDATA  Length =07D0
MYPROC . . . . . . . . . . . . .        F PROC  0000    MYCODE  Length =0036
??0000 . . . . . . . . . . . . .        L NEAR  0024    MYCODE
??0001 . . . . . . . . . . . . .        L NEAR  0027    MYCODE

Warning Severe
Errors  Errors
0       0
```

7–2. ábra A 7–1. ábrán látható program listafile-ja

Figyeljük meg, hogyan bővíti ki a makro a programot. Vegyük észre, hogy ott, ahol a makro először szerepel a forráskódban, a bal oldalon nem jelenik meg gépi kód, csak ott, ahol a DELAY makrót konkrétan meghívjuk. A programban használt makro vagy makrokönyvtár mindig ilyen módon szerepel a listában. A következőkben a makróhoz tartozó gépi kódu részt emeljük ki:

```
                                        DELAY   10
001F    52          +                   PUSH    DX
0020    51          +                   PUSH    CX
0021    BA  000A    +                   MOV     DX,10
0024    B9  FF00    +??0000:            MOV     CX,0FF00H
0027    49          +??0001:            DEC     CX
0028    75  FD      +                   JNZ     ??0001
002A    4A          +                   DEC     DX
002B    75  F7      +                   JNZ     ??0000
002D    59          +                   POP     CX
002E    5A          +                   POP     DX
```

A (+) jelek a makrók megjelölését szolgálják, megkönnyítve ezzel a program nyomkövetését és a hibakeresést. Mivel a DELAY makróról a programon belül csak egy másolat készült, a LOCAL deklarációra valójában nincs szükség. A lokális deklaráció eredményeként azonban P1 és P2 helyett a ??0000 és ??0001 cimkék jelennek meg. Ha a DELAY-ről ugyanazon a programon belül még egy másolatot készítenénk, akkor ezek az értékek már ??0002 és ??0003 lennének. Ha egy makrón belül a cimkéket LOCAL-nak deklaráljuk, programozási hibáktól menekülhetünk meg.

Makrokönyvtár

A makrók a 7–1. ábrán bemutatott módon önálló programokba beilleszthetők, de ha makrokönyvtárban szerepelnek, akkor bármelyik programba belefoglalhatók. Sok időt és fáradságot takaríthatunk meg magunknak, ha a hasznos és gyakran használt makróinkat makrokönyvtárba gyűjtjük össze. A 7–3. ábrán a MACLIB.MAC elnevezésű makrokönyvtárat láthatjuk. A kód beírásakor ne feledkezzünk meg arról, hogy a makrókat nem a lefordított állapotukban kell tárolnunk a lemezen. A PUSHA és POPA makrókat csak akkor kell definiálnunk, ha 8088-as és 8086-os processzorokkal dolgozunk. A PUSHA ésPOPA utasításokat valamennyi 80286/80386-os assembler értelmezi. Ha 80286-os vagy 80386-os processzorral dolgozunk, győződjünk meg arról, hogy a 0.286C vagy a 0.386C pszeudo műveletek előírása a program elején megtörtént. A PUSHA és POPA utasításokkal a számítógép regisztereinek éppen érvényes állapotát menthetjük el. Különösen akkor van erre szükség, ha bizonytalanok vagyunk abban, hogy az elkészített új rutinjaink hogyan befolyásolják majd az egyes regiszterek értékét. A könyvtárban szereplő DELAY makro pontosan megegyezik azzal, amit a 7–1. ábrán láthatunk. A CLEARSCREEN makro már szerepelt az 5. fejezet egyik programjának részeként. A PRINTNUM makro ASCII számokat helyez el a képernyőn. (Ne felejtsük el, hogy az AL regiszterben szereplő 05H értéket ASCII kóddá kell átalakítanunk, ha a képernyőn meg akarjuk jeleníteni.)

A CURSOR elnevezésű makro a LOCATE változóban tárolt pozícióba (DH = függőleges

---

```
PUSHA   MACRO                   ;;A RENDSZER REGISZTEREINEK ELMENTESE
        PUSH    AX              ;;A 80286/80386-OS GEPEKNEL ERRE A
        PUSH    CX              ;;MAKRORA NINCS SZUKSEG
        PUSH    DX
        PUSH    BX
        PUSH    SP
        PUSH    BP
        PUSH    SI
        PUSH    DI
        ENDM

POPA    MACRO                   ;;A RENDSZER REGISZTEREINEK
        POP     DI              ;;VISSZAALLITASA
        POP     SI              ;;A 80286/80386-OS GEPEKNEL ERRE A
        POP     BP              ;;MAKRORA NINCS SZUKSEG
        POP     SP
        POP     BX
        POP     DX
        POP     CX
        POP     AX
        ENDM

DELAY   MACRO   TIME            ;;SZOFTVERREL MEGOLDOTT KESLELTETES
        LOCAL   P1,P2           ;;P1 ES P2 LOKALIS CIMKEK
        PUSHA                   ;;A REGISZTEREK TARTALMANAK ELMENTESE
        MOV     DX,TIME         ;;A TIME VALTOZO TARTALMA DX-BE KERUL
P1:     MOV     CX,0FF00H       ;;CX-BE A 00FF00H SZAMLALO KERUL
P2:     DEC     CX              ;;A KESLELTETESI CIKLUS KEZDETE
        JNZ     P2              ;;HA CX<>0, CX CSOKKENTESE
        DEC     DX              ;;HA CX=0, DX CSOKKENTESE
        JNZ     P1              ;;HA DX<>0, CX ISMETELT FELTOLTESE
        POPA                    ;;HA DX=0, A REG.-EK TARTALMANAK
                                ;;VISSZAALLITASA
        ENDM                    ;;KILEPES A MAKROBOL
```

```
CLEARSCREEN  MACRO              ;;A SZINES KEPERNYO LETORLESE
        PUSHA                   ;;A REGISZTEREK TARTALMANAK ELMENTESE
        MOV     CX,0            ;;AZ ABLAK BAL FELSO SARKANAK
                                ;;BEALLITASA
        MOV     DX,2479H        ;;AZ ABLAK JOBB ALSO SARKANAK
                                ;;BEALLITASA
        MOV     BH,7            ;;A KEPERNYO ATTRIBUTUMANAK BEALLITASA
        MOV     AX,0600H        ;;MEGSZAKITASI PARAMETER
        INT     10H             ;;A MEGSZAKITAS HIVASA
        POPA                    ;;A REGISZTEREK TARTALMANAK
                                ;;VISSZAALLITASA
        ENDM                    ;;A MAKRO VEGE

PRINTNUM MACRO                  ;;ASCII SZAM MEGJELENITESE A KEPERNYON
        PUSHA                   ;;A REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    AX              ;;AX TARTALMANAK ELMENTESE
        MOV     AH,15           ;;AZ EPPEN ERVENYES KEPERNYO BETOLTESE
        INT     10H             ;;A MEGSZAKITAS HIVASA
        POP     AX              ;;AX ERTEKENEK VISSZAALLITASA
        AND     AL,0FH          ;;CSAK AZ ALSO NEGY BITET TARTJUK MEG
        OR      AL,30H          ;;ATVALTAS ASCII KODBA
        MOV     AH,10           ;;MEGSZAKITASI PARAMETER
        MOV     CX,1            ;;A KIIRANDO KARAKTEREK SZAMA
        INT     10H             ;;A MEGSZAKITAS HIVASA
        POPA                    ;;A REG.-EK TARTALMANK VISSZAALLITASA
        ENDM                    ;;A MAKRO VEGE

CURSOR  MACRO LOCATE            ;;A KURZOR POZICIONALASA A LOCATE-BAN
                                ;;ELOIRT HELYRE
        PUSHA                   ;;A REGISZTEREK TARTALMANAK ELMENTESE
        MOV     AH,15           ;;AZ EPPEN ERVENYES KEPERNYO BETOLTESE
        INT     10H             ;;A MEGSZAKITAS HIVASA, BX BEALLITASA
        MOV     DX,LOCATE       ;;A KEPERNYOHELYZET DX-BE KERUL
        MOV     AH,2            ;;A KURZOR PARAMETERENEK BEALLITASA
        INT     10H             ;;A MEGSZAKITAS HIVASA
        POPA                    ;;A REG.-EK TARTALMANAK VISSZAALLITASA
        ENDM                    ;;A MAKRO VEGE

PRINTCHAR MACRO TEXT            ;;AZ ADATSZEGMENSBEN ELHELYEZETT
                                ;;KARAKTEREK MEGJELENITESE
        PUSHA                   ;;A REGISZTEREK TARTALMANAK ELMENTESE
        LEA     DX,TEXT         ;;A KARAKTERLANCOT TEXT-BEN ADJUK AT
        MOV     AH,9            ;;NYOMTATAS A '$' KARAKTERIG AZ EPPEN
        INT     21H             ;;ERVENYES KURZORPOZICIOTOL KEZDVE
        POPA                    ;;A REG.-EK TARTALMANAK VISSZAALLITASA
        ENDM                    ;;A MAKRO VEGE
```

7–3. ábra A hasznos makrók gyűjteménye a MACLIB.MAC file-ba

koordináta, DL = vízszintes koordináta) állítja a kurzort. A PRINTCHAR makro az éppen érvényes kurzorpozíciótól kezdve kinyomtat a képernyőre egy ASCII karakterekből álló láncot. A nyomtatás az adatszesgmensben elhelyezett karakterlánc $ karakteréig tart. A 7–4. ábrán látható program a makrokönyvtár használatát és hasznosságát mutatja be. A program letörli a képernyőt, megjelenít egy üzenetet a képernyő tetején (ez egy egyszerű szamláló

program) és folyamatosan elszámol 0-tól 9-ig, amit a képernyő középpontjában láthatunk. A program öt teljes ciklus után fejeződik be. Az IF1 pszeudo művelettel jelezzük, hogy makrokönyvtárat kell a programhoz csatlakoztani. Az INCLUDE pszeudo művelettel a makrokönyvtár neve adható meg. Megjegyzés: a program az AL regiszter (számláló) legalacsonyabb helyértékű számjegyét jeleníti csak meg. Tehát ha AL-ben 45 van, csak az 5-öt látjuk a képernyőn.

```
;IBM 8088/80286-OS GEPEKRE KESZITETT PROGRAM
;A MAKROKONYVTAR HASZNALATANAK BEMUTATASA

IF1                             ;NE FELEJTSUK EL BETOLTENI AZ
      INCLUDE B:MACLIB.MAC      ;ELOZETESEN ELMENTETT MAKROKONYVTARAT
ENDIF                           ;A B MEGHAJTOROL

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
MESSAGE DB      'EZ EGY EGYSZERU SZAMLALO PROGRAM$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        CLEARSCREEN             ;A KEPERNYO LETORLESE
        CURSOR  0019H           ;A KURZOR BEALLITASA AZ UZENETHEZ
        PRINTCHAR MESSAGE       ;AZ UZENET MEGJELENITESE

        MOV     AX,00           ;A SZAMLALO INICIALIZALASA
AGAIN:  CURSOR  0C28H           ;A KURZORT A KEPERNYO KOZEPERE ALLITJUK
        PRINTNUM                ;AL TARTALMANAK KINYOMTATASA
        DELAY   10              ;10 EGYSEGNYI VARAKOZAS
        ADD     AL,01           ;AL-HEZ 1-ET ADUNK
        DAA                     ;AZ EREDMENY DECIMALIS KIIGAZITASA
        CMP     AL,05H          ;HA AZ 5 CIKLUS LEJART, KILEPES
        JE      ENDO
        JMP     AGAIN           ;ISMETLES
ENDO:   CLEARSCREEN             ;A KEPERNYO ISMETELT TORLESE

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

7–4. ábra Makrokönyvtár használatának bemutatása

```
IF1                             ;NE FELEJTSUK EL BETOLTENI AZ
        INCLUDE B:MACLIB.MAC    ;ELOZETESEN ELMENTETT MAKROKONYVTARAT
ENDIF                           ;A B MEGHAJTOROL
```

A program első három fő sora egy üzenet megjelenítéséről gondoskodik.

```
CLEARSCREEN             ;A KEPERNYO LETORLESE
CURSOR  0019H           ;A KURZOR BEALLITASA AZ UZENETHEZ
PRINTCHAR MESSAGE       ;AZ UZENET MEGJELENITESE
```

A CLEARSCREEN makro letörli a monitor képernyőjét. A CURSOR makro az első sor 25. pozíciójába állítja be a kurzort, így az üzenet középre kerül. A PRINTCHAR makro a MESSAGE változóban tárolt üzenetet megjeleníti a képernyőn.

A számok kijelzéséről a következő blokk gondoskodik:

```
        MOV     AX,00           ;A SZAMLALO INICIALIZALASA
AGAIN:  CURSOR  0C28H           ;A KURZORT A KEPERNYO KOZEPERE ALLITJUK
        PRINTNUM                ;AL TARTALMANAK KINYOMTATASA
        DELAY   10              ;10 EGYSEGNYI VARAKOZAS
        ADD     AL,01           ;AL-HEZ 1-ET ADUNK
        DAA                     ;AZ EREDMENY DECIMALIS KIIGAZITASA
        CMP     AL,05H          ;HA AZ 5 CIKLUS LEJART, KILEPES
        JE      ENDO
        JMP     AGAIN           ;ISMETLES
ENDO:   CLEARSCREEN             ;A KEPERNYO ISMETELT TORLESE
```

A PRINTNUM makro az AL regiszter legalacsonyabb helyértékű négy bitjét ASCII karakterré alakítja át és megjeleníti a képernyőn. Az egymást követő decimális számok előállítása az ADD és DAA utasításokkal történik. A program befejeződése előtt még egyszer letöröljük a képernyőt. Ha van bátorságunk fordítsuk le a forrásprogramot, és nyomtassuk ki az LST kiterjesztésű file-t. Látni fogjuk, hogy milyen sok kód beírásától mentett meg bennünket a makrokönyvtár.

7.2 ELJÁRÁSOK

Eddigi programjainkban mindig csak egyetlen eljárás szerepelt, a programok azonban több eljárásból is állhatnak. Az eljárásoknak két fő típusuk van:

- FAR (szegmensek közötti)
- NEAR (szegmensen belüli)

Amikor különböző rutinokat eljárásokba foglalunk, a szubrutinhoz hasonló helyzetet teremtünk. Ezek az aleljárások bármikor meghívhatók a főeljárásból, és a PASCAL-ban vagy a FORTRAN-ban megismert módszerekhez hasonlóan használhatók. A következőkben mind a szegmensen belüli, mind a szegmensek közötti eljárásokat tanulmányozni fogjuk.

Eljárások készítése

Legtöbb assembly program szerkezete a verem, az adat és a kódszegmens elhelyezésének tekintetében hasonló. A most bemutatott kódszegmens, amelyben magát a programot helyezzük el, a MYCODE nevet viseli. A kódszegmensen belül számos eljárás összegyűjthető.

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

(Itt kell elhelyezni az aktualis programkodot)

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Az eljárás neve MYPROC, amelyet FAR attribútummal láttunk el, mivel ez a főeljárás. Minden eljárást hasonló módon kell szerveznünk és a NEAR vagy FAR attribútummal kell ellátnunk. Vegyük észre, hogy még mielőtt a MYPROC ENDP utasítással lezárnánk az eljárást, egy RET-et is ki kell adnunk. A NEAR és FAR attribútumok megadása a CALL utasítás típusának meghatározásában segít a mikroprocesszornak. Az eljárásból való visszatérés útvonalát is meg kell adnunk, ami más a NEAR és más a FAR attribútum esetén. Ha az eljárást a NEAR attribútummal láttuk el, akkor az IP (utasításmutató) kerül rá a veremre, míg a FAR attribútum esetén mind az IP, mind a CS (kódszegmens). A 7–5. ábrán látható programlista nagyon hasonló a 7. fejezet első programjához. Alapvetően csak annyi a különbség, hogy a késleltetési rutint a makroszerkezetből egy NEAR típusú eljárásba tettük át. Hasonlítsuk össze a 7–1. ábrán és a 7–5. ábrán látható kódokat.

```
;RGB MONITORRAL ELLATOTT 8088/80286-OS IBM GEPEKRE
;KESZITETT PROGRAM
;NEAR TIPUSU ELJARAS HASZNALATNAK BEMUTATASA

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
BACK    DB      2000 DUP (' ')
MYDATA  ENDS


MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,ES:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE

;A PROGRAMSZEGMENS 80*25 SZOKOZ KEPERNYORE IRASAVAL LETORLI
;A KEPERNYOT.MIVEL BL ERTEKET AZ EREDETIHEZ KEPEST MODOSITJUK,
;EZZEL A TELJES KEPERNYO SZINENEK MEGVALTOZTATASAT IS ELERJUK.
;A DELAY ELJARAS HATASARA EZ A SZIN EGY BIZONYOS IDEIG MEGMARAD
;A KEPERNYON
        MOV     CX,08H          ;A CIKLUST 8-SZOR ISMETELTETJUK
        MOV     BL,00H          ;AZ EREDETI HATTER SZINENEK BEALLITASA
AGAIN:  LEA     BP,BACK         ;SZOKOZOKBOL ALLO KAR.LANC
                                ;MEGJELENITESE
        MOV     DX,0000         ;A KURZOR BEALLITASA A BAL FELSO SAROKBA
        MOV     AH,19           ;A KAR.LANC ATTRIBUTUMANAK BEALLITASA
        MOV     AL,1            ;KARAKTEREK KINYOMTATASA ES
                                ;KURZORPOZICIONALAS
        PUSH    CX              ;A CIKLUSSZAMLALO ELMENTESE
        MOV     CX,07D0H        ;2000 SZOKOZT IRUNK A KEPERNYORE
        INT     10H             ;A MEGSZAKITAS HIVASA
        CALL    DELAY           ;AZ ELJARAS HIVASA
        ADD     BL,10H          ;A HATTER SZINENEK MEGVALTOZTATASA
        POP     CX              ;AZ EREDETI CIKLUSSZAMLALO
                                ;VISSZAALLITASA
        LOOP    AGAIN           ;A MUVELET ELVEGZESE OSSZESEN 8-SZOR

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE

DELAY   PROC    NEAR
        PUSH    DX              ;DX ES CX EREDETI TARTALMANAK
        PUSH    CX              ;ELMENTESE
```

```
        MOV     DX,10           ;10 EGYSEGNYI KESLELTETES
P1:     MOV     CX,0FF00H       ;CX-BE A 00FF00H SZAMLALO KERUL
P2:     DEC     CX              ;A KESLELTETESI CIKLUS KEZDETE
        JNZ     P2              ;HA CX<>0, CX CSOKKENTESE
        DEC     DX              ;HA CX=0, DX CSOKKENTESE
        JNZ     P1              ;HA DX<>0, CX ISMETELT FELTOLTESE
        POP     CX              ;HA DX=0, CX ES DX EREDETI
        POP     DX              ;TARTALMANAK VISSZAALLITASA
        RET                     ;KILEPES AZ ELJARASBOL
DELAY   ENDP                    ;A DELAY ELJARAS LEZARASA

MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

7–5. ábra A NEAR típusú eljárás bemutatása

A DELAY 10 utasítást a CALL DELAY váltja fel:

```
        MOV     CX,08H          ;A CIKLUST 8-SZOR ISMETELTETJUK
        MOV     BL,00H          ;AZ EREDETI HATTER SZINENEK BEALLITASA
AGAIN:  LEA     BP,BACK         ;SZOKOZOKBOL ALLO KAR.LANC
                                ;MEGJELENITESE
        MOV     DX,0000         ;A KURZOR BEALLITASA A BAL FELSO SAROKBA
        MOV     AH,19           ;A KAR.LANC ATTRIBUTUMANAK BEALLITASA
        MOV     AL,1            ;KARAKTEREK KINYOMTATASA ES
                                ;KURZORPOZICIONALAS
        PUSH    CX              ;A CIKLUSSZAMLALO ELMENTESE
        MOV     CX,07D0H        ;2000 SZOKOZT IRUNK A KEPERNYORE
        INT     10H             ;A MEGSZAKITAS HIVASA
        CALL    DELAY           ;AZ ELJARAS HIVASA
        ADD     BL,10H          ;A HATTER SZINENEK MEGVALTOZTATASA
        POP     CX              ;AZ EREDETI CIKLUSSZAMLALO
                                ;VISSZAALLITASA
        LOOP    AGAIN           ;A MUVELET ELVEGZESE OSSZESEN 8-SZOR

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
```

Az eljárásokat a CALL utasítással kell meghívnunk. Egy másik különbség a DELAY 10 eltávolítása a programból. A paramétereket és változókat nem lehet olyan módon átadni az eljárásoknak, mint ahogy a makronál láttuk. Ahhoz, hogy 10 egységnyi késleltetést létesítsünk, a 10-et közvetlenül a DELAY eljárásba kell belefoglalnunk.

A DELAY eljárás olyan, mint egy apró program:

```
DELAY   PROC    NEAR
        PUSH    DX              ;DX ES CX EREDETI TARTALMANAK
        PUSH    CX              ;ELMENTESE
        MOV     DX,10           ;10 EGYSEGNYI KESLELTETES
P1:     MOV     CX,0FF00H       ;CX-BE A FF00H SZAMLALO KERUL
P2:     DEC     CX              ;A KESLELTETESI CIKLUS KEZDETE
        JNZ     P2              ;HA CX<>0, CX CSOKKENTESE
        DEC     DX              ;HA CX=0, DX CSOKKENTESE
        JNZ     P1              ;HA DX<>0, CX ISMETELT FELTOLTESE
        POP     CX              ;HA DX=0, CX ES DX EREDETI
        POP     DX              ;TARTALMANAK VISSZAALLITASA
        RET                     ;KILEPES AZ ELJARASBOL
DELAY   ENDP                    ;A DELAY ELJARAS LEZARASA
```

A makro és a NEAR típusú eljárás között csak néhány – azonban fontos – eltérés van. A 7-5. ábrán látható program LST file-ját a 7–6. ábrán adjuk meg. Ha ezt a listát a 7–2. ábrával összevetjük, az első szembetűnő különbség az, hogy a (+) karakterek a nyomtatásból eltűntek. Ez természetes is, hiszen a jelenlegi programunkban már nem szerepelnek makrók, a (+) jelek pedig erre utalnak.

A makrók minden olyan helyen ismételten megjelennek a programkódban, ahol azt megjelöljük, míg az eljárások esetében maga a kód csak egyszer szerepel. Az elmondottakat figyeljük meg a DELAY eljárás használata során:

```
                                ;RGB MONITORRAL ELLATOTT 8088/80286-OS IBM GEPEKRE
                                ;KESZITETT PROGRAM
                                ;NEAR TIPUSU ELJARAS HASZNALATNAK BEMUTATASA

0000                            STACK   SEGMENT PARA STACK
0000    40 [                            DB      64 DUP ('MYSTACK ')
           4D 59 53 54
           41 43 4B 20
                    ]

0200                            STACK   ENDS

0000                            MYDATA  SEGMENT PARA 'DATA'
0000  07D0 [                    BACK    DB      2000 DUP (' ')
           20
              ]

07D0                            MYDATA  ENDS


0000                            MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
0000                            MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
                                ASSUME  CS:MYCODE,ES:MYDATA,SS:STACK
0000  1E                        PUSH    DS              ;A DS REGISZTER ELMENTESE
0001  2B C0                     SUB     AX,AX           ;AX TORLESE
0003  50                        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
0004  B8 ---- R                 MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
0007  8E C0                     MOV     ES,AX           ;AX TARTALMANAK ATADASA ES-BE

                                ;A PROGRAMSZEGMENS 80*25 SZOKOZ KEPERNYORE IRASAVAL LETORLI
                                ;A KEPERNYOT.MIVEL BL ERTEKET AZ EREDETIHEZ KEPEST MODOSITJ IK,
                                ;EZZEL A TELJES KEPERNYO SZINENEK MEGVALTOZTATASAT IS ELERJUK.
                                ;A DELAY ELJARAS HATASARA EZ A SZIN EGY BIZONYOS IDEIG MEGMARAD
                                ;A KEPERNYON
0009  B9 0008                   MOV     CX,08H          ;A CIKLUST 8-SZOR ISMETELTETJUK
000C  B3 00                     MOV     BL,00H          ;AZ EREDETI HATTER SZINENEK BEALLITASA
000E  8D 2E 0000 R              AGAIN:  LEA     BP.BACK         ;SZOKOZOKBOL ALLO KAR.LANC
                                                        ;MEGJELENITESE
0012  BA 0000                   MOV     DX,0000         ;A KURZOR BEALLITASA A BAL FELSO SAROKBA
0015  B4 13                     MOV     AH,19           ;A KAR.LANC ATTRIBUTUMANAK BEALLITASA
0017  B0 01                     MOV     AL,1            ;KARAKTEREK KINYOMTATASA ES
                                                        ;KURZORPOZICIONALAS
0019  51                        PUSH    CX              ;A CIKLUSSZAMLALO ELMENTESE
001A  B9 07D0                   MOV     CX,07D0H        ;2000 SZOKOZT IRUNK A KEPERNYORE
001D  CD 10                     INT     10H             ;A MEGSZAKITAS HIVASA
001F  E8 0029 R                 CALL    DELAY           ;AZ ELJARAS HIVASA
```

```
0022  80 C3 10                        ADD     BL,10H          ;A HATTER SZINENEK MEGVALTOZTATASA
0025  59                              POP     CX              ;AZ EREDETI CIKLUSSZAMLALO
                                                              ;VISSZAALLITASA
0026  E2 E6                           LOOP    AGAIN           ;A MUVELET ELVEGZESE OSSZESEN 8-SZOR
0028  CB                              RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
0029                            MYPROC  ENDF                    ;A MYPROC ELNEVEZESU ELJARAS VEGE

0029                            DELAY   PROC    NEAR
0029  52                              PUSH    DX              ;DX ES CX EREDETI TARTALMANAK
002A  51                              PUSH    CX              ;ELMENTESE
002B  BA 000A                         MOV     DX,10           ;10 EGYSEGNYI KESLELTETES
002E  B9 FF00                   P1:   MOV     CX,0FF00H       ;CX-BE A 00FF00H SZAMLALO KERUL
0031  49                        P2:   DEC     CX              ;A KESLELTETESI CIKLUS KEZDETE
0032  75 FD                           JNZ     P2              ;HA CX<>0, CX CSOKKENTESE
0034  4A                              DEC     DX              ;HA CX=0, DX CSOKKENTESE
0035  75 F7                           JNZ     P1              ;HA DX<>0, CX ISMETELT FELTOLTESE
0037  59                              POP     CX              ;HA DX=0, CX ES DX EREDETI
0038  5A                              POP     DX              ;TARTALMANAK VISSZAALLITASA
0039  C3                              RET                     ;KILEPES AZ ELJARASBOL
003A                            DELAY   ENDP                    ;A DELAY ELJARAS LEZARASA

003A                            MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                        END                     ;A PROGRAM VEGE
Segments and groups:

                N a m e                 Size    align   combine class

MYCODE . . . . . .              003A    PARA    NONE    'CODE'
MYDATA . . . . . .              07D0    PARA    NONE    'DATA'
STACK. . . . . .                0200    PARA    STACK

Symbols:

                N a m e                 Type    Value   Attr

AGAIN. . . . . .                L NEAR  000E    MYCODE
BACK . . . . . .                L BYTE  0000    MYDATA  Length =07D0
DELAY. . . . . .                N PROC  0029    MYCODE  Length =0011
MYPROC . . . . . .              F PROC  0000    MYCODE  Length =0029
P1 . . . . . .                  L NEAR  002E    MYCODE
P2 . . . . . .                  L NEAR  0031    MYCODE

Warning Severe
Errors  Errors
0       0
```

7–6. ábra A 7–5. ábrán látható program .LIST file-ja

Valószínűleg felfigyeltünk a FAR RET és a NEAR RET aktuális gépi kódja közötti különbségre is. Mivel az eljárások hívásánál a program nem bővül ki az eljárás kódjával, az eljárásokat tartalmazó programok általában tömörebbek. Viszont ennek is ára van. Az eljárások szubrutinként történő használatakor nem létesül soron belüli azonnali végrehajtás. A főeljárásból az aleljárásba el kell ugrania a programnak, ami bizonyos időt igényel. Az időbeli eltérés még jelentősebb lehet, ha a CALL ciklusban szerepel.

Eljárásokból készített könyvtár

Az eljárásokból könyvtárat is készíthetünk, amit a felhasználói program a szerkesztési folyamat során hív meg. Az ilyen eljárásokat a FAR attribútummal kell ellátnunk, mivel ezek az éppen érvényes kódszegmenshez képest külsőnek számítanak. A 7–7. ábrán látható program hasonlít a 7–4. ábrán bemutatott programhoz.

```
;IBM 8088/80386-OS GEPEKRE KESZITETT PROGRAM
;A FAR TIPUSU ELJARASOK HASZNALATANAK BEMUTATASA

        EXTRN   PUSHA:FAR,POPA:FAR,DELAY:FAR,CLEARSCREEN:FAR
        EXTRN   PRINTNUM:FAR,CURSOR:FAR,PRINTCHAR:FAR

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA PUBLIC 'DATA'
MESSAGE DB      'EZ EGY EGYSZERU SZAMLALO PROGRAM$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        CALL    CLEARSCREEN     ;A KEPERNYO LETORLESE
        MOV     DX,19H          ;A KURZOR BEALLITASA AZ UZENETHEZ
        CALL    CURSOR
        LEA     DX,MESSAGE      ;MESSAGE HELYENEK BEOLTESE
        CALL    PRINTCHAR       ;AZ UZENET MEGJELENITESE

        MOV     AX,00           ;A SZAMLALO INICIALIZALASA
AGAIN:  MOV     DX,0C28H        ;A KURZORT A KEPERNYO KOZEPERE
        CALL    CURSOR          ;ALLITJUK
        CALL    PRINTNUM        ;AL TARTALMANAK KINYOMTATASA
        MOV     DX,10           ;10 EGYSEGNYI VARAKOZAS
        CALL    DELAY
        ADD     AL,01           ;AL-HEZ 1-ET ADUNK
        DAA                     ;AZ EREDMENY DECIMALIS KIIGAZITASA
        CMP     AL,05H          ;HA AZ 5 CIKLUS LEJART, KILEPES
        JE      ENDO
        JMP     AGAIN           ;ISMETLES
ENDO:   CALL    CLEARSCREEN     ;A KEPERNYO ISMETELT TORLESE

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

7–7. ábra A FAR típusú eljárások használatának bemutatása

Az EXTERN pszeudo művelettel teremtjük meg a külső eljárások nevét.

```
EXTRN   PUSHA:FAR,POPA:FAR,DELAY:FAR,CLEARSCREEN:FAR
EXTRN   PRINTNUM:FAR,CURSOR:FAR,PRINTCHAR:FAR
```

A file – amiben az előbbi eljárásokat elhelyeztük – elnevezésére nem kell különösebb gondot fordítanunk. Csak akkor lesz rá szükségünk, amikor a főprogrammal összeszerkesztjük. Most pedig lássuk magát a programot.

```
CALL    CLEARSCREEN     ;A KEPERNYO LETORLESE
MOV     DX,19H          ;A KURZOR BEALLITASA AZ UZENETHEZ
CALL    CURSOR
LEA     DX,MESSAGE      ;MESSAGE HELYENEK BEOLTESE
CALL    PRINTCHAR       ;AZ UZENET MEGJELENITESE
```

Hasonlítsuk össze az előbbi kódrészletet a 7–4. ábrán látható kóddal. A CURSOR eljárás hívása előtt a kurzor pozícióját DX-ben kell előírnunk. Ez a PRINTCHAR eljárás esetén is így történik, ui. a MESSAGE változó helyét DX-ben adjuk meg, ezért DX eredeti értékét el kell tárolnunk. Ezzel a programozási technikával tehát a paraméterátadás problémáját oldjuk meg úgy, hogy globális regiszterekbe helyezzük el azokat. A 7–8. ábrán egy másik forrásfile-t láthatunk, ami a külső eljárásokat tartalmazza. Hasonlítsuk össze a 7–3. ábrával.

```
MYCODE  SEGMENT
        PUBLIC PUSHA,POPA,DELAY,CLEARSCREEN,PRINTNUM,CURSOR,PRINTCHAR
        ASSUME  CS:MYCODE


PUSHA   PROC    FAR             ;A RENDSZER REGISZTEREINEK ELMENTESE
        PUSH    AX              ;A 80286/80386-OS GEPEKNEL ERRE AZ
        PUSH    CX              ;ELJARASRA NINCS SZUKSEG
        PUSH    DX
        PUSH    BX
        PUSH    SP
        PUSH    BP
        PUSH    SI
        PUSH    DI
        RET
PUSHA   ENDP

POPA    PROC    FAR             ;A RENDSZER REGISZTEREINEK
        POP     DI              ;VISSZAALLITASA
        POP     SI              ;A 80286/80386-OS GEPEKNEL ERRE AZ
        POP     BP              ;ELJARASRA NINCS SZUKSEG
        POP     SP
        POP     BX
        POP     DX
        POP     CX
        POP     AX
        RET
POPA    ENDP

DELAY   PROC    FAR             ;SZOFTVERREL MEGOLDOTT KESLELTETES
        PUSH    CX              ;REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    DX
```

```
P1:     MOV     CX,0FF00H       ;CX-BE A 00FF00H SZAMLALO KERUL
P2:     DEC     CX              ;A KESLELTETESI CIKLUS KEZDETE
        JNZ     P2              ;HA CX<>0, CX CSOKKENTESE
        DEC     DX              ;HA CX=0, DX CSOKKENTESE
        JNZ     P1              ;HA DX<>0, CX ISMETELT FELTOLTESE
        POP     DX              ;HA DX=0, A REG.-EK TARTALMANAK
        POP     CX              ;VISSZAALLITASA
        RET
DELAY   ENDP

CLEARSCREEN  PROC FAR           ;A SZINES KEPERNYO LETORLESE
        PUSH    AX              ;REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     CX,0            ;AZ ABLAK BAL FELSO SARKANAK
                                ;BEALLITASA
        MOV     DX,2479H        ;AZ ABLAK JOBB ALSO SARKANAK
                                ;BEALLITASA
        MOV     BH,7            ;A KEPERNYO ATTRIBUTUMANAK BEALLITASA
        MOV     AX,0600H        ;MEGSZAKITASI PARAMETER
        INT     10H             ;A MEGSZAKITAS HIVASA
        POP     DX              ;A REGISZTEREK TARTALMANAK
        POP     CX              ;VISSZAALLITASA
        POP     BX
        POP     AX
        RET
CLEARSCREEN ENDP

PRINTNUM PROC   FAR             ;ASCII SZAM MEGJELENITESE A KEPERNYON
        PUSH    AX              ;A REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    CX
        PUSH    AX              ;AX TARTALMANAK ELMENTESE
        MOV     AH,15           ;AZ EPPEN ERVENYES KEPERNYO BETOLTESE
        INT     10H             ;A MEGSZAKITAS HIVASA
        POP     AX              ;AX ERTEKENEK VISSZAALLITASA
        AND     AL,0FH          ;CSAK AZ ALSO NEGY BITET TARTJUK MEG
        OR      AL,30H          ;ATVALTAS ASCII KODBA
        MOV     AH,10           ;MEGSZAKITASI PARAMETER
        MOV     CX,1            ;A KIIRANDO KARAKTEREK SZAMA
        INT     10H             ;A MEGSZAKITTAS HIVASA
        POP     CX              ;A REG.-EK TARTALMANAK VISSZAALLITASA
        POP     AX
        RET
PRINTNUM ENDP

CURSOR  PROC    FAR             ;A KURZOR POZICIONALASA AZ
                                ;ELOIRT HELYRE
        PUSH    AX              ;A REGISZTER TARTALMANAK ELMENTESE
        MOV     AH,15           ;AZ EPPEN ERVENYES KEPERNYO BETOLTESE
        INT     10H             ;A MEGSZAKITAS HIVASA, BX BEALLITASA
        MOV     AH,2            ;A KURZOR PARAMETERENEK BEALLITASA
        INT     10H             ;A MEGSZAKITAS HIVASA
        POP     AX              ;A REG. TARTALMANAK VISSZAALLITASA
        RET
CURSOR  ENDP
```

```
PRINTCHAR PROC  FAR             ;AZ ADATSZEGMENSBEN ELHELYEZETT
                                ;KARAKTEREK MEGJELENITESE
        PUSH    AX              ;A REGISZTER TARTALMANAK ELMENTESE
        MOV     AH,9            ;NYOMTATAS A '$' KARAKTERIG AZ EPPEN
        INT     21H             ;ERVENYES KURZORPOZICIOTOL KEZDVE
        POP     AX              ;A REG. TARTALMANAK VISSZAALLITASA
        RET
PRINTCHAR ENDP

MYCODE  ENDS
        END
```

7–8. ábra Felhasználói programból hívható eljárások csoportja

A program fejrésze a külső eljárások esetében minimális:

```
MYCODE  SEGMENT
        PUBLIC PUSHA,POPA,DELAY,CLEARSCREEN,PRINTNUM,CURSOR,PRINTCHAR
        ASSUME  CS:MYCODE

;itt kell elhelyezni az eljarasokat

MYCODE  ENDS
END     MYPROC
```

Ebben az esetben minden eljárást a PUBLIC pszeudo művelettel deklaráltunk, amivel a szegmensek közötti információcserét tettük lehetővé. A program fejrészének a további része hagyományos. Az eljárások szerkezetének szemléltetésére a CLEARSCREEN rutint emeljük ki:

```
CLEARSCREEN  PROC FAR           ;A SZINES KEPERNYO LETORLESE
        PUSH    AX              ;REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     CX,0            ;AZ ABLAK BAL FELSO SARKANAK
                                ;BEALLITASA
        MOV     DX,2479H        ;AZ ABLAK JOBB ALSO SARKANAK
                                ;BEALLITASA
        MOV     BH,7            ;A KEPERNYO ATTRIBUTUMANAK BEALLITASA
        MOV     AX,0600H        ;MEGSZAKITASI PARAMETER
        INT     10H             ;A MEGSZAKITAS HIVASA
        POP     DX              ;A REGISZTEREK TARTALMANAK
        POP     CX              ;VISSZAALLITASA
        POP     BX
        POP     AX
        RET
CLEARSCREEN ENDP
```

Az eljárás fejrésze az elnevezésből, a PROC pszeudo műveletből és a FAR attribútumból áll. Minden eljárást a RET direktívával, az eljárás nevének megismétlésével és az ENDP pszeudo művelettel zártunk le. Az eljárások törzsrésze csak kis mértékben különbözik attól, amit a 7–3. ábrán – a makróknál – láthatunk. Jelenleg tehát két különálló programmal rendelkezünk. Az egyik a főprogram, a másik pedig az eljárásokból álló könyvtár. Mindegyik programot külön

kell lefordítanunk, majd pedig a programok OBJ kiterjesztésű file-jait a szerkesztőprogram segítségével összekapcsolnunk. Az IBM és a Microsoft DOS kézikönyvei részletesen ismertetik a szerkesztési eljárás folyamatát és a szerkesztési opciókat. A következő példa az egyik szerkesztési módszert mutatja be annak feltételezésével, hogy a szerkesztő- program az A, az összeszerkesztendő file-ok a B meghajtón vannak:

```
B>A:LINK MAINFILE.OBJ+SUBFILE.OBJ
```

Emlékeztetőül mégegyszer hangsúlyozzuk, hogy a makrokönyvtár forráskódú file-ban összegyűjtött, lefordítás nélküli rutinokból áll, az eljárásokból álló könyvtár pedig olyan tárgykódokat tartalmazó file, amiben lefordított rutinok találhatók. A makrókat a fordítás idején, az eljárásokból álló könyvtárat az szerkesztés idején kell elérnie a kiszolgálóprogramnak.

7.3 KÖNYVTÁRAK

Miben különbözik a szerkesztőprogram által igényelt könyvtár a makrokönyvtártól és az eljárásokból álló könyvtártól? Nem sokban. Mindhárom képes a kód tárolására, amelyet a későbbiekben más programok is hívhatnak. A könyvtár kifejezés inkább a DOS szerkesztőprogrammal hozható kapcsolatba. Amikor egy programot szerkesztünk, a következő menüt láthatjuk a képernyőn:

```
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985
Object Modules [.OBJ]: B:MYFILE
Run File [MYFILE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
```

A DOS szerkesztőprogram Library (könyvtár) opciója közelebb áll az eljárásokból álló könyvtárhoz. Azok a könyvtárak, amelyeket a szerkesztés idején hívunk meg, lefordított eljárások gyűjteményéből állnak. (Hasonlóan az eljárásokból álló könyvtárhoz.) Az IBM Assemblerhez (2.0 verzió) és a Microsoft Assemblerhez (3.0 verzió) könyvtárkezelő program (Library Manager) is tartozik, ami a könyvtári modulok felépítését és szerkesztését segíti. Ha egy bizonyos könyvtárra szükségünk van, a szerkesztési eljárás során a megfelelő rendszerüzenetnél be kell írnunk a nevét. A DOS szerkesztési könyvtár a makrokönyvtárhoz és az eljárásokból álló könyvtárhoz hasonlóan nagymértékben csökkenti a programfejlesztésre fordított időt, mivel az itt található eljárásokat megírták helyettünk, és a hibaszűrést is elvégezték. A könyvtárkezelő program a következő műveleteket támogatja:

- tárgyfile-ok felvétele,
- tárgyfile-ok törlése,
- tárgymodulok kiemelése,
- tárgymodulok kicserélése,
- további könyvtárak felvétele,
- könyvtárak tartalmának listázása,
- a könyvtár tárolásához használt lapok méretének megváltoztatása.

A következőkben bemutatjuk, hogyan kell létrehozni egy DOS szerkesztési könyvtárat, felhasználva ehhez a megelőző fejezetrészben bemutatott példákat. A 7–7. és a 7–8. ábrán látható programokon nem változtatunk. A 7–7. ábra programja a főprogram, míg a 7–8. ábrán az eljárásokat tartalmazó file látható. A könyvtárkezelő programmal az eljárásokat tartalmazó könyvtár beszerkeszthető a DOS könyvtárba. A következőkben a 7–8. ábra programját a NEWLIB elnevezésű könyvtárba konvertáljuk:

```
IBM Personal Computer Library Manager
Version 1.00
(C) Copyright IBM Corp 1984
(C) Copyright Microsoft Corp 1984
Library name: B:NEWLIB
Library does not exist. Create? Y
Operations: + FIG7 - 8
List file: NEWLIB.LSF
```

A már meglevő könyvtárhoz a könyvtárkezelő program segítségével bármikor további rutinok és eljárások csatlakoztathatók, csupán csak a NEWLIB elnevezést kell beírnunk annál a rendszerüzenetnél, ami a könyvtár nevét kéri be. A NEWLIB.LSF file a következő információkat tartalmazza:

```
CLEARSCREEN ..... FIG7-8   CURSOR ..... FIG7-8
DELAY ..... FIG7-8         POPA ..... FIG7-8
PRINTCHAR ..... FIG7-8     PRINTNUM ..... FIG7-8
PUSHA ..... FIG7-8
  FIG7-8 Offset: 200H Code and data size: 5E
  CLEARSCREEN CURSOR DELAY POPA
  PRINTCHAR PRINTNUM PUSHA
```

Első látásra úgy tűnik, hogy a DOS szerkesztési könyvtár nem nagyon különbözik az eljárásokat tartalmazó könyvtártól. Ez azonban csak a látszat. A szerkesztési könyvtár eljárásai a szerkesztéskor kapcsolódnak hozzá a többi tárgymodulhoz. A könyvtárkezelő program használatával a könyvtárhoz tartozó eljárások mennyisége tetszőlegesen változtatható. A DOS szerkesztési könyvtár bármelyik előző módszerhez viszonyítva sokkal dinamikusabb és könnyebben kezelhető.

7.4 A RENDELKEZÉSRE ÁLLÓ LEHETŐSÉGEK ÖSSZEHASONLÍTÁSA

A makrók, az eljárások és a könyvtárak funkciója sok szempontból azonos. A programozónak csak egyszer kell megírnia a kódokat, amelyeket aztán a későbbiekben bármelyik programjában használhat. Lehetővé válik hatékony, moduláris és strukturált programok fejlesztése. A döntés megkönnyítésének érdekében minden egyes módszernek felsoroljuk az előnyeit és hátrányait.

A makrók előnyei:

1. A makrók gyorsak, mivel soron belüli, azonnali kódvégrehajtást tesznek lehetővé.

2. A makróknál lehetőség van paraméterek fogadására és átadására.
3. A makrók editálása könnyű, mivel a forráskódú könyvtárban tárolhatók.
4. A makrókat hasznosító program fejrésze igen egyszerű. Makrokönyvtár esetén csak az IF1...ENDIF pszeudo műveleteket kell alkalmazni.

A makrók hátrányai:

1. A makrók miatt a forráskód mérete megnő, mivel forráskód minden makrohíváskor kibővül a makro kódjával.

Az eljárásokból álló könyvtár előnyei:

1. Az eljárásokat alkalmazó program forráskódja rövid marad, mivel az eljárások hívásakor azok kódja nem bővíti ki a programot.

Az eljárásokból álló könyvtár hátrányai:

1. Az eljárások alkalmazása lassítja a program futását, mivel minden CALL utasításnál a számítógépnek ki kell lépnie a főprogramból az eljárást tartalmazó programrészbe.
2. A programba az eljárás fejrészét bele kell foglalnunk. Nem szabad megfeledkeznünk a NEAR vagy FAR deklarálásról és a könyvtári file-ok external megjelöléséről sem.
3. "Makroértelmű" paraméterátadás az eljárásnál nem lehetséges.

A DOS szerkesztési könyvtár előnyei:

1. L. az eljárásokból álló könyvtár előnyeit.
2. A könyvtárkezelő program segítségével a könyvtárban szereplő rutinok száma egyszerűen módosítható.

A DOS szerkesztési könyvtár hátrányai:

1. L. az eljárásokból álló könyvtár hátrányait. Melyik a legjobb módszer? Mikor kell inkább a makrót, mint az eljárást alkalmazni? Mikor jobb a DOS könyvtár, mint az eljárásokat tartalmazó könyvtár? A kérdések megválaszolásakor §ffgondoljuk át a következő szempontokat:

1. Rövid rutinok esetében a makrókat érdemes választani.
2. Ritkán használt rutinoknál válasszuk a makrót. (Ha a rutint nem hívjuk meg túl gyakran, a kódméret növekedése nem lesz jelentős.)
3. Programfejlesztés vagy kísérletezés során használjunk makrót. (Könnyebb a létrehozás, a szövegszerkesztés és a meghívás.)
4. Ha a rutinunk hosszú, használjunk eljárást. (Igy rövidebb lesz a forráskód.)
5. Ha egy rutint gyakran hívunk, készítsünk belőle eljárást. (Az eljárások nem növelik jelentősen a program méretét.)
6. A különböző programjainkban szereplő azonos rutinokat érdemes a DOS szerkesztési könyvtárban elhelyezni a könnyű elérhetőség miatt.

Vannak olyan assembly programozók, akik sosem használnak makrókat, míg mások igyekeznek elkerülni az eljárások használatát. Egyik sem helyes módszer. Mindig azt a technikát kell alkalmaznunk, ami a konkrét feladat igényeinek legjobban megfelel.

8. FEJLETT PROGRAMOZÁSI TECHNIKÁK

A megelőző fejezetben néhány érdekes assembly programot már megismerhettünk. Bár ezek a programok egyszerűek voltak, azonban lehetővé tették az assembly utasítások gyakorlását, és a műveletek eredményének vizsgálatát. A BIOS rutinok és a DOS megszakítások felhasználásával egyszerű programokat készítettünk a képernyő színének megváltoztatására, és az üzenetek megjelenítésére. Az 5. fejezetben olyan alapelveket ismerhettünk meg, amelyek ennek a fejezetnek az alapjai. A 8. fejezet programjai összetettebbek és a kezdő programozó számára talán kevésbé érdekesek, mint az előző fejezetek programjai. Néhány példában menüvezérelt képernyő szerepel, ami a felhasználó és a gép között a kölcsönös párbeszédet teszi lehetővé. Néhány olyan programot is bemutatunk, amelyek az adatok lemezre írását és lemezről olvasását végzik. (A haladó programozókat ez a téma általában nagyon érdekli.)

sszetett programokat nem a bonyolultság kedvéért készítünk. Célunk olyan ötletek, programozási módszerek bemutatása, amelyek segítségére lehetnek az Olvasónak a programozási problémák megoldásában. Minden programnál érdemes elgondolkozni azon, hogy mire tudnánk még használni.

A fejezetben szereplő programok segítségével elsajátíthatjuk az új problémák megközelítésének módját, és az előző fejezetben megszerzett tudásunkat továbbfejleszthetjük.

8.1 GRAFIKON RAJZOLÁSA SZÍNES KÉPERNYŐRE

1. **feladat:** Szinuszhullám rajzolása nagyfelbontású grafikus képernyőre.

A mérnökök, tudósok és matematikusok számára gyakran hasznos lehet, ha a számítások és műveletek eredményét a grafikus képernyőn ábrázolni tudják. A 9. fejezetben bemutatásra kerülő még nagyobb és hatékonyabb programok előzeteseként – amelyekben egyébként már a 80287/80387-es társprocesszort is alkalmazzuk – egy szinuszhullámot jelenítünk meg a 640*200-as nagyfelbontású grafikus képernyőn táblázatban elhelyezett adatok felhasználásával. (A feladathoz színes kártya és színes monitor szükséges.) Az assembly programozásban komoly hátráltató tényező a matematikai függvények hiánya. A programozóknak maguknak kell elkészíteniük a négyzetgyök-, vagy a köbgyökvonáshoz, a trigonometriai függvényekhez, az integrál- és differenciálszámításhoz és a statisztikai analízishez szükséges rutinokat. Bonyolult függvények algoritmusát nemcsak kitalálni nehéz, hanem programozni is. Mivel a szinuszfüggvény (vagy bármely más trigonometriai függvény) értékeinek számítását a 80286/80386-os processzor nem támogatja, a programozónak az algebrai megközelítést kell alkalmaznia. Ehhez viszont a számok bizonyos hatványértékeit kell előállítani, amelyeket aztán faktoriálisokkal is

kell osztani. Egyik feladatot sem könnyű megoldani az assembly nyelven. De még mielőtt a problémát tovább részleteznénk, inkább gondoljuk végig, hogy mi szükséges egy szinuszhullám ábrázolásához. Először is el kell férnie a képernyőn létrehozott koordináta-rendszerben, ezen kívül minden pontnak, amiből a szinuszhullám összetevődik, önálló X és Y koordinátával kell rendelkeznie. Ezeket a koordinátákat – mivel most nem kívánunk az algebrai megközelítés göröngyös útján járni – már előzetesen kiszámíthatjuk és az adatszegmensben egy táblázatban tárolhatjuk.

A programnak ebben az esetben csak a megfelelő értéket kell a táblázatból kiemelnie. Ez a program tehát az 5. fejezetben bemutatott kikeresési táblázat koncepciójára építkezik. Az adatszegmens a következő lesz:

```
SINE    DB   00,02,04,05,07,09,11,12,14,16,17,19,21,23,24,26
        DB   28,29,31,33,34,36,38,39,41,42,44,45,47,49,50
        DB   52,53,55,56,57,59,60,62,63,64,66,67,68,70,71
        DB   72,73,74,76,77,78,79,80,81,82,83,84,85,86,87
        DB   88,88,89,90,91,91,92,93,93,94,95,95,96,96,97
        DB   97,97,98,98,99,99,99,99,100,100,100,100,100,100,100
```

A SINE a táblázat neve. Honnan kaptuk ezeket az értékeket és mit jelentenek? A SINE táblázat 91 értéket tartalmaz, amelyek 0-tól 90 fokig az egyes szögekhez tartozó szinuszértékeket reprezentálják. Az eredeti szinuszokat 100-zal szoroztuk, és kerekítettük, annak érdekében, hogy a képernyő felbontásának megfelelő méretarányt tudjunk beállítani. Az IBM színes monitor felbontása függőleges irányban – mind a közepes, mind a nagyfelbontás esetén – 200 képelem (pixel). Az bizonyára ismeretes, hogy a 90 fok szinusza 1-gyel egyenlő, aminek a jelenlegi méretarányban 100 felel meg. A szinuszhullámot általában 0-tól 360 fokig szokták ábrázolni. Mivel a szinuszhullám szimmetrikus, ezért a második, harmadik és negyedik szögnegyed értékeit felesleges külön feltüntetni a táblázatban. Az ábrázolási terv meglehetősen egyszerű. A SINE táblázatból ki kell keresni egy értéket, ami a képernyőn az Y eltolást (a pont függőleges helyzetét) jelenti. A pontok ábrázolását a képernyő bal oldalán kezdjük, mégpedig úgy, hogy minden egyes vízszintes képelemhez egy függőleges értéket rendelünk hozzá. Mivel 360 pontot kell ábrázolnunk (360 fok), és a nagyfelbontású képernyő vízszintesen 640 képelemből áll, bőségesen elég a hely. A SINE táblázatban szereplő információ mind a négy térnegyed ábrázolásához elégséges. A program teljes listája a 8–1. ábrán látható.

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;SZOGERTEKEK MEGHATAROZASA ADATTABLAZAT FELHASZNALASAVAL ES
;A GORBE MEGRAJZOLASA BIOS MEGSZAKITASOK SEGITSEGEVEL

SETSCREEN MACRO                 ;;NAGYFELBONTASU FEKETE/FEHER KEPERNYO
        MOV     AH,00           ;;200*640 PONT
        MOV     AL,06
        INT     10H
        ENDM

WRITEDOT MACRO                  ;;A PONT ABRAZOLASAT ELVEGZO MAKRO
        MOV     AH,12
        MOV     AL,01
        MOV     CX,ANGLE
        ADD     CX,140          ;;AZ ABRA POZICIONALASA A KEPERNYON
        MOV     DH,00
        MOV     DL,TEMP
        INT     10H
        ENDM
```

```
STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
SINE    DB      00,02,04,05,07,09,11,12,14,16,17,19,21,23,24,26
        DB      28,29,31,33,34,36,38,39,41,42,44,45,47,49,50
        DB      52,53,55,56,57,59,60,62,63,64,66,67,68,70,71
        DB      72,73,74,76,77,78,79,80,81,82,83,84,85,86,87
        DB      88,88,89,90,91,91,92,93,93,94,95,95,96,96,97
        DB      97,97,98,98,99,99,99,99,100,100,100,100,100,100,100
ANGLE   DW      0
TEMP    DB      0
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

;SZINUSZERTEKEK MEGHATAROZASA ADATTABLAZATTAL
;AZ ERTEKEKT 100-ZAL SZOROZTUK ES EGESZRE KEREKITETTUK

        SETSCREEN               ;A 200*640-ES KEPERNYO BEALLITASA
AGAIN:  LEA     BX,SINE         ;A TABLAZAT KEZDETENEK BETOLTESE BX-BE
        MOV     AX,ANGLE        ;ANGLE TARTALMA AX-BE KERUL
        CMP     AX,180          ;A SZOG 180 FOKNAL NAGYOBB?
        JLE     NEWQUAD         ;HA KISEBB, AKKOR A SZOG AZ 1.VAGY A 2.
                                ;SZOGNEGYEDBE ESIK
        SUB     AX,180          ;HA EGYENLO VAGY NAGYOBB, 180 KIVONASA
NEWQUAD: CMP    AX,90           ;A SZOG 90 FOKNAL NAGYOBB?
        JLE     SECQUAD         ;HA NAGYOBB, AKKOR A 2. NEGYEDBE ESIK
        NEG     AX              ;NEGATIV SZINUSZERTEK ELOALLITASA
        ADD     AX,180          ;KORREKCIO, HA A SZOG 90 FOK VAGY ENNEL
                                ;NAGYOBB
SECQUAD: ADD    BX,AX           ;A TABLAZATBA FOGLALT ADAT HELYENEK
        MOV     AL,SINE[BX]     ;MEGHATAROZASA ES BEKERESE
        CMP     ANGLE,180       ;HA A SZOG 180 FOKNAL NAGYOBB,
        JGE     BIGDIS          ;A KEPERNYOELTOLAST MEG HOZZAADJUK,
        NEG     AL              ;EGYEBKENT PEDIG NEGATIV SZINUSZT
                                ;KEPEZUNK
        ADD     AL,100          ;A SZINUSZ KORREKCIOJA
        JMP     READY           ;A PONT MEGJELENITESE
BIGDIS: ADD     AL,99
READY:  MOV     TEMP,AL         ;AL TAROLASA TEMP-BEN
        WRITEDOT                ;MAKROHIVAS
        ADD     ANGLE,1         ;A KOVETKEZO SZOGERTEK BEKERESE
        CMP     ANGLE,360       ;ELERTUK MAR A 360 FOKOT?
        JLE     AGAIN           ;HA NEM, AKKOR ISMETLES
;A SZINUSZGORBET ABRAZOLO PROGRAMRESZ VEGE
```

```
;VARAKOZAS EGY BILLENTYU LENYOMASARA ES VISSZAKAPCSOLAS
;SZOVEGES-KEPERNYO UZEMMODBA
        MOV     AH,07           ;BILLENTYUZET PARAMETER
        INT     21H             ;A BILLENTYUZET BEOLVASASA KILEPESHEZ
        MOV     AH,00           ;KEPERNYO PARAMETER
        MOV     AL,03           ;25*80-AS SZINES UZEMMOD
        INT     10H             ;A KEPERNYO BEALLITASA

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–1. ábra Szinuszhullám ábrázolása táblázatban tárolt értékek felhasználásával

Ez a program két hasznos makrót tartalmaz, a SETSCREEN-t és a WRITEDOT-ot. A SETSCREEN makrót emeljük ki először:

```
SETSCREEN MACRO                 ;;NAGYFELBONTASU FEKETE/FEHER KEPERNYO
        MOV     AH,00           ;;200*640 PONT
        MOV     AL,06
        INT     10H
        ENDM
```

A makro feladata az, hogy meghíváskor átkapcsoljon a nagyfelbontású grafikus üzemmódba. Az INT 10H különböző opcióit az 5. fejezetben tárgyaltuk részletesen.

A WRITEDOT makro a következő:

```
WRITEDOT MACRO                  ;;A PONT ABRAZOLASAT ELVEGZO MAKRO
        MOV     AH,12
        MOV     AL,01
        MOV     CX,ANGLE
        ADD     CX,140          ;;AZ ABRA POZICIONALASA A KEPERNYON
        MOV     DH,00
        MOV     DL,TEMP
        INT     10H
        ENDM
```

Ez a makro ugyancsak a 10H megszakítást használja. CX a vízszintes, DX a függőleges helyzetet jelenti. Mivel 640 vízszintes pont van, viszont csak 360 fok (640 − 360 = 280, és 280/2 = 140), ha balról 140 képelemnyi vízszintes eltolást alkalmazunk, a szinuszhullámot a képernyőn középre helyezzük el. A TEMP elnevezésű változó a függőleges értékek teljes tartományát tartalmazza (0-tól 100-ig). Ha a WRITEDOT makrót meghívjuk, az ANGLE és TEMP változók adják meg azt a helyet a képernyőn, ahová a pontot el kell helyezni.

```
        SETSCREEN               ;A 200*640-ES KEPERNYO BEALLITASA
AGAIN:  LEA     BX,SINE         ;A TABLAZAT KEZDETENEK BETOLTESE BX-BE
        MOV     AX,ANGLE        ;ANGLE TARTALMA AX-BE KERUL
        CMP     AX,180          ;A SZOG 180 FOKNAL NAGYOBB?
        JLE     NEWQUAD         ;HA KISEBB, AKKOR A SZOG AZ 1.VAGY A 2.
                                ;SZOGNEGYEDBE ESIK
        SUB     AX,180          ;HA EGYENLO VAGY NAGYOBB, 180 KIVONASA
NEWQUAD: CMP    AX,90           ;A SZOG 90 FOKNAL NAGYOBB?
        JLE     SECQUAD         ;HA NAGYOBB, AKKOR A 2. NEGYEDBE ESIK
        NEG     AX              ;NEGATIV SZINUSZERTEK ELOALLITASA
        ADD     AX,180          ;KORREKCIO, HA A SZOG 90 FOK VAGY ENNEL
                                ;NAGYOBB
```

A megfelelő szinuszérték kikereséséhez tudnunk kell, hogy a szög melyik szögnegyedbe esik. Az AX-ben elhelyezett szögértéket először 180 fokkal hasonlítjuk össze. Ha a szög 180 fokkal egyenlő vagy kisebb, akkor a szög az első vagy a második szögnegyedbe esik, ahol a szinusz pozitív. Ha a szög 180 foknál nagyobb, akkor a harmadik vagy negyedik szögnegyedbe esik, ahol a szinusz negatív. Ebben az esetben 180 fokot kivonunk a szög értékéből. A következő lépésben azt vizsgáljuk meg, hogy a szög 90 foknál nagyobb-e? Ha igen, akkor 180 fokból kivonjuk. Ehhez azonban egy kis trükkre van szükségünk, mivel a SUB 180,AX kódot nem használhatjuk. (Hová kerülne az eredmény?) Ezért először AX komplemensét képezzük és ezt adjuk 180-hoz. A SECQUAD címkénél kezdődő kóddal határozzuk meg a SINE táblázatban a megfelelő szinuszérték helyzetét. (AX értékét a BX regiszterhez adjuk.)

```
SECQUAD: ADD     BX,AX          ;A TABLAZATBA FOGLALT ADAT HELYENEK
         MOV     AL,SINE[BX]    ;MEGHATAROZASA ES BEKERESE
         CMP     ANGLE,180      ;HA A SZOG 180 FOKNAL NAGYOBB,
         JGE     BIGDIS         ;A KEPERNYOELTOLAST MEG HOZZAADJUK,
         NEG     AL             ;EGYEBKENT PEDIG NEGATIV SZINUSZT
                                ;KEPEZUNK
         ADD     AL,100         ;A SZINUSZ KORREKCIOJA
         JMP     READY          ;A PONT MEGJELENITESE
BIGDIS:  ADD     AL,99
READY:   MOV     TEMP,AL        ;AL TAROLASA TEMP-BEN
         WRITEDOT               ;MAKROHIVAS
         ADD     ANGLE,1        ;A KOVETKEZO SZOGERTEK BEKERESE
         CMP     ANGLE,360      ;ELERTUK MAR A 360 FOKOT?
         JLE     AGAIN          ;HA NEM, AKKOR ISMETLES
```

A programrészlet következő néhány kódsora azt határozza meg, hogyan kerüljenek a pontok a képernyőre. A képernyő tetejéhez tartozó függőleges koordináta 0, tehát a képernyő aljához a 199-es koordinátaérték tartozik. A 0 amplitudójú szinusz helyzethez 100-as képernyőeltolást rendelünk. (Igy kerül a görbe középre.) Miután a WRITEDOT makrót meghívtuk, az ANGLE változó tartalmát növeljük, és ellenőrizzük, hogy mind a 360 pontot ábrázoltuk-e. Ha nem, akkor az eljárást megismételjük, egyébként pedig a program a következő kódrészletbe lép.

```
MOV     AH,07           ;BILLENTYUZET PARAMETER
INT     21H             ;A BILLENTYUZET BEOLVASASA KILEPESHEZ
```

Ebben a kódrészletben egy DOS megszakítással (részletesen l. az 5. fejezetben) egy billentyű lenyomásáig várakoztatjuk a programot. Ezáltal lehetővé válik, hogy a programozó a grafikát tetszőleges ideig tanulmányozza. Végül a program letörli a képernyőt és visszatér 28 * 80-as karakter üzemmódba.

```
MOV     AH,00           ;KEPERNYO PARAMETER
MOV     AL,03           ;25*80-AS SZINES UZEMMOD
INT     10H             ;A KEPERNYO BEALLITASA
```

Ezzel a módszerrel sokféle matematikai függvény ábrázolható, csupán csak a megfelelő méretarányt kell kialakítani, hogy minden függvényérték elférjen a képernyőn. Ha az ábrázolandó pontok helyzetének értékét a program futása alatt számítanánk ki, a futási idő a jelenleginél hosszabb lenne. Azzal a hátránnyal azonban számolnunk kell, hogy a táblázatban rögzített értékek miatt mindig pontosan ugyanazt a hullámalakot fogjuk látni. Azok a programozók, akik a pontok helyzetének értékét a program futása alatt kívánják meghatározni, a 9. fejezetben a 80287/80387-es társprocesszor használatának tárgyalásakor ennek a problémának a megoldását is megismerhetik.

8.2 EGY PROGRAM, AMI AZ IDŐ MÚLÁSÁT SZÁMOLJA MÁSODPERCEKBEN

2. feladat: Készítsünk egy olyan programot, ami jelzi az idő múlását. A számolás a program futásakor kezdődik, és hat számjegyes kijelzést tesz lehetővé.

Ebben a programozási feladatban számos problémát kell megoldanunk:

- Hogyan lehet a számolást másodpercekben kijelezni?
- Hol helyezzük el a képernyőn a számolás eredményét?
- A számolási eljárás milyen aritmetikában történjen (ASCII, BCD vagy bináris)? Hogyan tudjuk biztosítani a számolás pontosságát?

A probléma megoldásához a következőket rögzítjük:

A program hat számjegyből álló decimális értéket jelenít meg a képernyő közepén olyan pontossággal, amit a számítógép hardvere lehetővé tesz. Pakolatlan számok ASCII összeadása helyett pakolt számok decimális összeadását – az eredmény összeadás utáni decimális kiigazításával = DAA – valósítjuk meg. A számláláshoz három változót (SEC12, SEC34, SEC56) használunk. (Egyetlen DD méretű változó is elegendő lenne, amiből a szükséges részek BYTE PTR-ekkel érhetők el; ez a technika azonban ebben a szituációban túlságosan bonyolult.)

```
;8088/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;A PROGRAM 0-TOL 999999-IG SZAMOLJA A MASODPERCEKET
;A FORDITAS SORAN FELHASZNALJUK A HASZNOS MAKROKAT
;TARTALMAZO MACLIB.MAC KONYVTARAT
PAGE ,132                       ;AZ OLDAL MERETE 66*132

IF1                             ;A MAKROKONYVTAR BEILLESZTESE
        INCLUDE C:MACLIB.MAC    ;A PROGRAMBA
ENDIF

TD      MACRO   AMT             ;;KESLELTETES 1-60 MP-IG
                                ;;A KESLELTETES ERTEKET AMT-BEN KELL
                                ;;ELHELYEZNI
        PUSH    AX              ;;REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     AH,2CH          ;;AZ EPPEN ERVENYES IDO BEOLVASASA
        INT     21H
        MOV     BH,DH           ;;KESLELTETES+EPPEN ERVENYES IDO
        ADD     BH,AMT
        CMP     BH,60           ;;KIIGAZITAS UGY, HOGY NE LEGYEN 60
                                ;;FELETT
        JL      AGAIN
        SUB     BH,60
AGAIN:  MOV     AH,2CH          ;;AZ ORA ISMETELT MINTAVETELEZESE
        INT     21H
        CMP     BH,DH           ;;A KESLELTETES KESZ?
        JNE     AGAIN
        POP     DX              ;;REG.-EK TARTALMANAK VISSZAALLITASA
```

```
        POP     CX
        POP     BX
        POP     AX
        ENDM

SCREEN  MACRO   TIME,POSITION    ;;A SZAMJEGYEK KIJELZESET SZOLGALO
        PUSH    AX              ;;MAKRO
        PUSH    CX
        MOV     AL,TIME         ;;A SZAMJEGY, AMIT KI KELL JELEZNI
        AND     AL,0FH          ;;LSB KEPZES MASZKOLASSAL
        CURSOR  POSITION        ;;A SZAMJEGY HELYZETENEK BEALLITASA
        PRINTNUM                ;;ES KIJELZESE
        MOV     AL,TIME         ;;A MASODIK SZAMJEGY ELOALLITASA
        AND     AL,0F0H         ;;MSB KEPZES MASZKOLASSAL
        MOV     CL,04           ;;ROTALAS AZ LSB POZICIOBA
        ROR     AL,CL
        CURSOR  POSITION-1      ;;KIJELZES AZ LSB-TOL BALRA
        PRINTNUM                ;;A SZAMJEGY MEGJELENITESE
        POP     CX
        POP     AX
        ENDM

STACK   SEGMENT PARA STACK      ;A VEREMSZEGMENS DEFINIALASA
        DB      64 DUP ('MYSTACK ')
STACK   ENDS                    ;A VEREMSZEGMENS VEGE

MYDATA  SEGMENT PARA 'DATA'
SEC12   DB      0
SEC34   DB      0
SEC56   DB      0
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE      ;KODSZEGMENS DEFINIALASA
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AY           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        CLEARSCREEN             ;KEPERNYOTORLES MAKROVAL
CYCLE:  TD      01              ;A KESLELTETEST VEGZO MAKRO HIVASA
        MOV     AL,SEC12        ;ADATBETOLTES
        ADD     AL,1            ;A MASODPERC SZAMLALO NOVELESE
        DAA                     ;DECIMALIS KIIGAZITAS
        MOV     SEC12,AL        ;AZ EREDMENY ELMENTESE
        MOV     AL,SEC34        ;ADATBETOLTES
        ADC     AL,0            ;NOVELES AZ ATVITELJELZO BITTOL FUGGOEN
        DAA                     ;DECIMALIS KIIGAZITAS
        MOV     SEC34,AL        ;AZ EREDMENY ELMENTESE
        MOV     AL,SEC56        ;ADATBETOLTES
        ADC     AL,0            ;NOVELES AZ ATVITELJELZO BITTOL FUGGOEN
        DAA                     ;DECIMALIS KIIGAZITAS
        MOV     SEC56,AL        ;AZ EREDMENY ELMENTESE
        SCREEN  SEC12,0C2BH     ;A KIJELZEST BIZTOSITO MAKRO MEGHIVASA
        SCREEN  SEC34,0C29H
        SCREEN  SEC56,0C27H
```

```
        CURSOR  1900H           ;A KURZOR HELYZETET BEALLITO MAKRO
                                ;HIVASA
        MOV     AH,06H          ;A BILLENTYUZET BEOLVASASA
        MOV     DL,0FFH
        INT     21H
        CMP     AL,'Q'          ;KILEPES 'Q' LENYOMASA ESETEN
        JE      ENDO
        CMP     AL,'q'          ;KILEPES 'q' LENYOMASA ESETEN
        JE      ENDO
        JMP     CYCLE           ;A CIKLUS MEGISMETLESE
ENDO:

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–2. ábra Másodperceket számláló program

A 8–2. ábrán a program listája látható.

Vegyük észre, hogy a MACLIB.MAC könyvtárat a programhoz csatoltuk. (Ezt a könyvtárat az előző fejezetben definiáltuk és a C meghajtóra mentettük el.) Két makrót találunk a program elején. A TD makróval a másodpercekben való számolást, míg a SCREEN makróval az időértékek kijelzését valósítjuk meg. Először a TD makróval foglalkozunk.

```
TD      MACRO   AMT             ;;KESLELTETES 1-60 MP-IG
                                ;;A KESLELTETES ERTEKET AMT-BEN KELL
                                ;;ELHELYEZNI
        PUSH    AX              ;;REGISZTEREK TARTALMANAK ELMENTESE
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     AH,2CH          ;;AZ EPPEN ERVENYES IDO BEOLVASASA
        INT     21H
        MOV     BH,DH           ;;KESLELTETES+EPPEN ERVENYES IDO
        ADD     BH,AMT
        CMP     BH,60           ;;KIIGAZITAS UGY, HOGY NE LEGYEN 60
                                ;;FELETT
        JL      AGAIN
        SUB     BH,60
AGAIN:  MOV     AH,2CH          ;;AZ ORA ISMETELT MINTAVETELEZESE
        INT     21H
        CMP     BH,DH           ;;A KESLELTETES KESZ?
        JNE     AGAIN
        POP     DX              ;;REG.-EK TARTALMANAK VISSZAALLITASA
        POP     CX
        POP     BX
        POP     AX
        ENDM
```

A TD makro az első menetben bekéri az éppen érvényes idő értékét, és a DH regiszter felhasználásával átadja a számítógépnek. Ezután a kívánt késleltetési értéket hozzáadja az éppen érvényes időhöz, és kiigazítja úgy, hogy ne legyen 60 felett, majd a számot a BH regiszterben tárolja. A makro ezt követően egy ciklusba lép, ami folyamatosan mintavételezi

az időt és összehasonlítja a BH-ban található értékkel. Ha a két szám megegyezik, a program kilép a ciklusból. Ilyen módon nagyon pontos késleltetés érhető el. A SCREEN makrónak ugyancsak van néhány érdekes jellemzője:

```
SCREEN  MACRO   TIME,POSITION   ;;A SZAMJEGYEK KIJELZESET SZOLGALO
        PUSH    AX              ;;MAKRO
        PUSH    CX
        MOV     AL,TIME         ;;A SZAMJEGY, AMIT KI KELL JELEZNI
        AND     AL,0FH          ;;LSB KEPZES MASZKOLASSAL
        CURSOR  POSITION        ;;A SZAMJEGY HELYZETENEK BEALLITASA
        PRINTNUM                ;;ES KIJELZESE
        MOV     AL,TIME         ;;A MASODIK SZAMJEGY ELOALLITASA
        AND     AL,0F0H         ;;MSB KEPZES MASZKOLASSAL
        MOV     CL,04           ;;ROTALAS AZ LSB POZICIOBA
        ROR     AL,CL
        CURSOR  POSITION-1      ;;KIJELZES AZ LSB-TOL BALRA
        PRINTNUM                ;;A SZAMJEGY MEGJELENITESE
        POP     CX
        POP     AX
        ENDM
```

A SCREEN makro a TIME és a POSITION elnevezésű formális paramétereket tartalmazza, amelyek átadásáról gondoskodni kell. A SEC12, SEC34 és SEC56 változók bármelyike a TIME-on keresztül átadható a SCREEN-nek. A POSITION az időérték számjegyére vonatkozó kurzorpozíciót fogadja. A TIME két pakolt BCD számjegyet tartalmaz, amelyeket a képernyőn való megjelenítésük során külön kezelünk. A TIME változót a 0FH-val maszkoljuk, hogy az LSD értéket megőrizzük. Amikor a PRINTNUM makrot a makrokönyvtárból meghívjuk, ez a mennyiség ASCII számmá alakul át, és az éppen érvényes kurzorpozícióban megjelenik a képernyőn. A TIME-ot ezután újból betöltjük, de most az F0H-val maszkolva az MSD értéket őrizzük meg, amelyet azután a fenti módszerrel megjelenítünk. Ezt az eljárást mindaddig ismételjük, amíg mind a hat számjegyet (mind a három változót) kiírjuk a képernyőre.

A decimális aritmetikáért felelős kódrészlet a következő:

```
CYCLE:  TD      01              ;A KESLELTETEST VEGZO MAKRO HIVASA
        MOV     AL,SEC12        ;ADATBETOLTES
        ADD     AL,1            ;A MASODPERC SZAMLALO NOVELESE
        DAA                     ;DECIMALIS KIIGAZITAS
        MOV     SEC12,AL        ;AZ EREDMENY ELMENTESE
        MOV     AL,SEC34        ;ADATBETOLTES
        ADC     AL,0            ;NOVELES AZ ATVITELJELZO BITTOL FUGGOEN
        DAA                     ;DECIMALIS KIIGAZITAS
        MOV     SEC34,AL        ;AZ EREDMENY ELMENTESE
        MOV     AL,SEC56        ;ADATBETOLTES
        ADC     AL,0            ;NOVELES AZ ATVITELJELZO BITTOL FUGGOEN
        DAA                     ;DECIMALIS KIIGAZITAS
        MOV     SEC56,AL        ;AZ EREDMENY ELMENTESE
        SCREEN  SEC12,0C2BH     ;A KIJELZEST BIZTOSITO MAKRO MEGHIVASA
        SCREEN  SEC34,0C29H
        SCREEN  SEC56,0C27H
        CURSOR  1900H           ;A KURZOR HELYZETET BEALLITO MAKRO
                                ;HIVASA
```

Ez a kódrészlet a TD makro segítségével először egy 1 másodperces késleltetést generál (TD hívása az 1-es paraméterrel), majd pedig aktualizálja a második számlálót tartalmazó három

változót. Ezután SEC12-őt az AL regiszterbe tölti, amelyhez 1-et ad hozzá és az eredményt decimálisan kiigazítja (DAA). Ha a számlálást decimális formában kívánjuk megjeleníteni, a számnak az AL regiszterben kell lennie. A DAA utasításnak közvetlenül az ADD után kell állnia. Amint AL-re nézve az összeadás befejeződött, az eredmény a SEC12 változóba kerül. Ha az átviteljelző bit az összeadás következtében beállítódik, akkor a SEC34 számláló értékét növelni kell. Ezt követően SEC34-et AL-be töltjük, de most az ADD utasítás helyett az ADC utasítást használjuk. Ha az átviteljelző bit értéke 1, akkor ehhez a regiszterhez 1-et hozzáadunk, míg ellenkező esetben a regiszter értékét változatlanul hagyjuk. Az eredményt decimálisan kiigazítjuk, és tároljuk. A SEC56 változó esetén teljesen hasonló módon járunk el. Ha az Olvasó úgy érzi, hogy az előbbi kódrészletet egyszerűbben és hatékonyabban is meg tudná írni, próbálja meg nyugodtan. Legyünk azonban nagyon körültekintőek.

Még egy utolsó megjegyzés: A CURSOR 1900H hatása csupán csak az, hogy a kurzort a kijelzett számoktól eltávolítsa, ellenkező esetben a számjegyek villogását tapasztalnánk.

A következő kódrészlet azt teszi lehetővé, hogy a felhasználó egy billentyű lenyomásával kiléphessen a programból.

```
MOV     AH,06H          ;A BILLENTYUZET BEOLVASASA
MOV     DL,0FFH
INT     21H
CMP     AL,'Q'          ;KILEPES 'Q' LENYOMASA ESETEN
JE      ENDO
CMP     AL,'q'          ;KILEPES 'q' LENYOMASA ESETEN
JE      ENDO
JMP     CYCLE           ;A CIKLUS MEGISMETLESE
```

A karakter beolvasása a 21H DOS megszakítással oldható meg. Jelen esetben a program a q vagy a Q lenyomására befejezi a működését. Foglaljuk röviden össze, hogy mit tanulhattunk meg ebből a programból!

1. Megismerhettük, hogyan kell pontos időtartamú késleltetési rutint írni.
2. Megtanulhattuk, hogyan lehet többszörös pontosságú decimális aritmetikát megvalósítani komplikált hexadecimális-decimális rutinok nélkül.
3. Láthattuk, hogyan kell a számjegyek elhelyezését vezérelni a képernyőn.
4. Elsajátíthattuk a folyamatos ciklusból való kilépés elegáns technikáját.

Ennyi előzetes után bizonyára nem lesz nehéz ezt a programot kibővíteni úgy, hogy óra-perc-másodperc értékek kijelzésére is képes legyen.

8.3 MENÜVEZÉRELT PROGRAM KÉSZÍTÉSE

3. feladat: Készítsünk olyan interaktív programot, aminek a segítségével érdekes információkat tudhatunk meg egy személyről.

Interaktív programrészekre szinte valamennyi kiszolgálóprogramban szükség van. Akár szövegszerkesztőről, assemblerről, fordító- programról vagy játékról van szó, az egyszerű kezelhetőség igen lényeges szempont. A programozók ezt a feladatot rendszerint menüvel oldják meg. A program mindaddig várakozik, amíg a felhasználó választ a lehetőségek közül. Megfelelő hibaellenőrzésről is gondoskodni kell annak érdekében, hogy a programot oda nem

illő válaszokkal ne lehessen megzavarni. A jelenlegi egyszerű programunkban a felhasználó egy menü segíségével információkat tudhat meg arról a személyről, akinek adatait (név, cím, telefonszám, személyi szám, kedvenc recept) előzetesen egy file-ban rögzítettük.

A programból való kilépést szolgáló opciót ugyancsak feltüntettük. A program csak a listán felsorolt betűk lenyomását fogadja el válaszként. Más hibaellenőrzést nem végez. A program listája a 8–3. ábrán látható.

```
;8088/80386-OS GEPEKRE KESZITETT
;MENUVEZERELT ASSRMBLY PROGRAM

PAGE ,132                       ;AZ OLDAL MERETE 66#132

IF1                             ;A MAKROKONYVTAR CSATOLASA
      INCLUDE C:MACLIB.MAC
ENDIF

DISPLAY MACRO   WHAT            ;;KIJELZEST SZOLGALO MAKRO, AMI
      CLEARSCREEN               ;;KEPERNYOTORLEST,
      CURSOR  0800H             ;;A KARAKTERLANC
      PRINTCHAR WHAT            ;;MEGJELE  ITESET ES
      DELAY   45H               ;;KESLELTETEST VALOSIT MEG
      ENDM

STACK   SEGMENT PARA STACK      ;A VEREMSZEGMENS DEFIFIALASA
      DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'     ;AZ ADATSZEGMENS KEZDETENEK DEFINIALASA
NAME1   DB      '          HORVATH PETER$'

ADRES1  DB      '          1038 BUDAPEST',0DH,0AH
      DB      '          ZAPOR UTCA 10.$'

TELE    DB      '           170-890$'

SSNUM   DB      '           1-580519-4110$

REC     DB      'Halpaprikas',0DH,0AH,0AH
      DB      'Hozzavalok:',0DH,0AH
      DB      '1 kg nagyobb fajta hal (harcsa, csuka, ponty)',0DH,0AH
      DB      '5 dkg zsir',0DH,0AH
      DB      '10 dkg hagyma',0DH,0AH
      DB      '2 dl tejfol',0DH,0AH
      DB      'Paprika (izles szerint)',0DH,0AH
      DB      'So',0DH,0AH
      DB      'A tisztitott halat 1-1.5 cm szeles darabokra vagjuk,',0DH,0AH
      DB      'es megsozva 1/2 oraig allni hagyjuk. A hagymat megreszeljuk, es zsirban',0DH,0AH
      DB      'megfonnyasztjuk. A tuzrol levesszuk es hozzatesszuk a pirospaprikat.',0DH,0AH
      DB      'Hozzakeverjuk a tejfolt,belerakjuk a haldarabokat (nyaron paradicsomot',0DH,0AH
      DB      'is tehetunk bele), es lassu tuzon 1/2 orat fozzuk. Konnyu galuskat is',0DH,0AH
      DB      'adhatunk hozza.'
      DB      '$'
MESS    DB      'Kerem irja be annal a menutetelnel allo betut,',0DH,0AH,0AH
      DB      'amit meg akar jeleniteni:',0DH,0AH
      DB      'N - Nev',0DH,0AH
```

```
        DB      'C - Cim',0DH,0AH
        DB      'T - Telefonszam',0DH,0AH
        DB      'S - Szemelyi szam',0DH,0AH
        DB      'R - Recept',0DH,0AH
        DB      'L - kiLepes a programbol'.0DH,0AH
        DB      '$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
KEY:    CLEARSCREEN             ;KEPERNYOTORLES MAKROVAL
        CURSOR  0200H           ;A CURSOR MAKRO MEGHIVASA
        PRINTCHAR MESS          ;A PRINTCHAR MAKRO MEGHIVASA
        MOV     AH,1            ;PARAMETERBEALLITAS A BILLENTYUZET-
                                ;MEGSZAKITASI RUTINHOZ
        INT     21H             ;BILLENTYUZET-BEOLVASAS MEGSZAKITASSAL
        CMP     AL,'N'          ;'N' OSSZEHASONLITASA AL TARTALMAVAL
        JNE     NOTN            ;UGRAS NOTN-RE, HA NEM EGYENLO,
        DISPLAY NAME1           ;EGYEBKENT A DISPLAY-VEL MEGJELENITES
NOTN:   CMP     AL,'C'          ;'C' OSSZEHASONLITASA AL TARTALMAVAL
        JNE     NOTC            ;UGRAS NOTC-RE, HA NEM EGYENLO,
        DISPLAY ADRES1          ;EGYEBKENT A DISPLAY-VEL MEGJELENITES
NOTC:   CMP     AL,'T'          ;'T' OSSZEHASONLITASA AL TARTALMAVAL
        JNE     NOTT            ;UGRAS NOTT-RE, HA NEM EGYENLO,
        DISPLAY TELE            ;EGYEBKENT A DISPLAY-VEL MEGJELENITES
NOTT:   CMP     AL,'S'          ;'S' OSSZEHASONLITASA AL TARTALMAVAL
        JNE     NOTS            ;UGRAS NOTS-RE, HA NEM EGYENLO,
        DISPLAY SSNUM           ;EGYEBKENT A DISPLAY-VEL MEGJELENITES
NOTS:   CMP     AL,'R'          ;'R' OSSZEHASONLITASA AL TARTALMAVAL
        JNE     NOTR            ;UGRAS NOTR-RE, HA NEM EGYENLO,
        DISPLAY REC             ;EGYEBKENT A DISPLAY-VEL MEGJELENITES
NOTR:   CMP     AL,'L'          ;'L' OSSZEHASONLITASA AL TARTALMAVAL
        JE      FINISH          ;HA EGYENLO, UGRAS FINISH-RE,
        JMP     KEY             ;EGYEBKENT A PROGRAM ISMETLESE
FINISH:

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–3. ábra Menüvezérelt program bemutatása

A képernyő menüjéhez szükséges valamennyi információt az adatszegmensben helyeztük el. Láthatjuk, hogy ez még egy ilyen viszonylag egyszerű program esetében is meglehetősen sok memóriahelyet (kódot) foglal el. (Nagyméretű programoknál lehetséges, hogy még a 64 K terjedelmű extraszegmensre (ES) is szükség lehet.)

A programból elsőként a DISPLAY makrót emeljük ki.

```
DISPLAY MACRO   WHAT            ;;KIJELZEST SZOLGALO MAKRO, AMI
        CLEARSCREEN             ;;KEPERNYOTORLEST,
        CURSOR  0800H           ;;A KARAKTERLANC
        PRINTCHAR WHAT          ;;MEGJELENITESET ES
        DELAY   45H             ;;KESLELTETEST VALOSIT MEG
        ENDM
```

Ez a makró felelős a felhasználó által kiválasztott opció megjelenítéséért. A DISPLAY négy olyan makrót használ fel, amit a MACLIB.MAC makrokönyvtár tartalmaz. Valahányszor a WHAT változóba egy információ kerül, a DISPLAY letörli a képernyőt, a kurzort a 0800H pozícióba állítja, és kinyomtatja az információt, amit egy adott ideig meg is őriz a képernyőn. Vegyük észre, hogy a DELAY elnevezésű rutin minden olyan esetben kényelmesen használható, ha a késleltetés időtartamának pontossága különösebben nem lényeges. (A DELAY rutinnak átadott szám csak korrelációkban áll az idővel, mivel azt határozza meg, hogy a DELAY egy bizonyos ciklust hányszor hajtson végre.) Most tehát csak az a lényeges, hogy a DELAY-nek átadott szám olyan méretű legyen, hogy a képernyőn megjelenő információt kényelmesen el tudjuk olvasni.

A következő kódrészlet a főmenüt elhelyezi a képernyőn és bekéri a választ a billentyűzetről:

```
KEY:    CLEARSCREEN             ;KEPERNYOTORLES MAKROVAL
        CURSOR  0200H           ;A CURSOR MAKRO MEGHIVASA
        PRINTCHAR MESS          ;A PRINTCHAR MAKRO MEGHIVASA
        MOV     AH,1            ;PARAMETERBEALLITAS A BILLENTYUZET-
                                ;MEGSZAKITASI RUTINHOZ
        INT     21H             ;BILLENTYUZET-BEOLVASAS MEGSZAKITASSAL
```

Ebben a programban is felhasználjuk a MACLIB.MAC makrokönyvtárat. A CLEARSCREEN makróval töröljük le a képernyőt és a PRINTCHAR makróval jelenítjük meg a menüt a CURSOR makróval beállított kurzorpozíciótól kezdve. A felhasználók tetszőleges ideig láthatják a menüt, ui. a program mindaddig várakozik, amíg a megfelelő billentyűt le nem nyomjuk. A 21H DOS megszakítás a lenyomott billentyű kódját az AL regiszterben helyezi el. A program hátralevő részében tehát AL értékét vizsgáljuk, és ettől függően a DISPLAY makróval a megfelelő üzenetet megjelenítjük. Irjuk be és futtassuk a programot! Bizonyára észrevesszük, hogy a program a felhasználó által beírt karakterekre igen gyorsan reagál. Ebből a példából már érzékelhető, hogy az assemblyben készített programok ugyanolyan hatékonyak, mint azok, amiket magas szintű nyelven írtunk, és ráadásul még sokkal gyorsabbak is.

Bonyolult menüvezérelt interaktív program készítése

4. feladat: Készítsünk egy olyan programot, ami a billentyűzetről bekér két számot, összeadja vagy összeszorozza azokat és az eredményt kijelzi a képernyőn.

Ez a programozási feladat néhány érdekes kérdést vet fel:

- Hogyan lehet a számokat bekérni a billentyűzetről?
- Milyen formátumban legyenek a számok?
- Hogyan lehet a matematikai műveleteket végrehajtani, és az eredményt kijelezni?

Az előző programokban már foglalkoztunk ilyen jellegű kérdésekkel, tehát a válasz nem lesz

olyan nehéz. Bizonyára tudjuk, hogyan kell a menüket kijelezni a képernyőn, és a billentyűzet karaktereit beolvasni. Azért, hogy a feladat ne legyen túlságosan bonyolult, bizonyos korlátozásokat teszünk:

1. A billentyűzetről csak egyjegyű decimális számok írhatók be.
2. Csak az összeadás és a szorzás engedélyezett. (Ez azt jelenti, hogy a legnagyobb összeg 18 (9+9), és a legnagyobb szorzat 81 (9 * 9)).

Emlékeztetőül:

Amikor a 21H DOS megszakítással vizsgáltatjuk a billentyűzetet, a lenyomott gomb ASCII kódja az AL regiszterbe kerül. A számok ASCII kódjai 30H-tól 39H-ig terjednek. Mivel a programban csak egyetlen számjegyből álló decimális számok beírását engedélyezzük, a számoknak ebbe a tartományba kell esniük. A legjobb az lenne, ha a számokat ebben a formátumban hagyhatnánk, mivel ez az alak szükséges a monitoron való megjelenítéshez is. Azonban az aritmetikai műveletek elvégzéséhez decimális számokra van szükségünk, tehát a 30H értéket a szám ASCII kódjából le kell vonnunk. Vizsgáljuk meg figyelmesen a program listáját a 8–4. ábrán.

```
;80286/80386-OS GEPEKRE KESZITETT PROGRAM
;2 EGYJEGYU SZAM OSSZEADASA VAGY OSSZESZORZASA MENUVEZERELT
;INTERAKTIV PROGRAM SEGITSEGEVEL
PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

IF1                             ;MAKROKONYVTAR CSATOLASA
        INCLUDE C:MACLIB.MAC
ENDIF

ARITH   MACRO                   ;;ARITMETIKAI MAKRO
        PUSH    AX              ;;A BEFOLYASOLT REGISZTEREK ELMENTESE
        PUSH    CX
        SUB     AX,AX           ;;AX NULLAZASA
        MOV     AL,NUM1         ;;AZ ELSO SZAM ELHELYEZESE AX-BEN
        CMP     OPER,'A'        ;;OSSZEADAS VAGY SZORZAS ?
        JE      ADDER           ;;HA OSSZEADAS, AKKOR ADDER-RE UGRAS,
MULTP:  MUL     NUM2            ;;EGYEBKENT SZORZAS
        AAM                     ;;AZ EREDMENY ASCII KIIGAZITASA
        JMP     FINISH          ;;UGRAS FINISH-RE
ADDER:  ADD     AL,NUM2         ;;AZ OSSZEADAS ELOKESZITESE
        AAA                     ;;AZ EREDMENY ASCII KIIGAZITASA
FINISH: AND     AH,0FH          ;;AH-BOL CSAK AZ ALSO NEGY BITET TARTJUK MEG
        MOV     CL,04H          ;;A BITLEPTETES ELOKESZITESE
        SHL     AH,CL           ;;BITLEPTETES BALRA
        ADD     AL,AH           ;;PAKOLT SZAM ELOALLITASA (AL+AH)
        MOV     ANS,AL          ;;AZ EREDMENY ANS-BA KERUL
        POP     CX              ;;A BEFOLYASOLT REGISZTEREK VISSZAALLITASA
        POP     AX
        ENDM                    ;;A MAKRO VEGE

READKEY MACRO   THISKEY         ;;A BILLENTYU BELOLVASASA
        PUSH    AX              ;;A BEFOLYASOLT REGISZTER ELMENTESE
        MOV     AH,01H          ;;A MEGSZAKITASHOZ SZUKSEGES PARAMETER BEALLITASA
        INT     21H             ;;A BILLENTYUZETET BEOLVASO MEGSZAKITAS
        MOV     THISKEY,AL      ;;A MEGSZAKITASBOL SZARMAZO ERTEK ELMENTESE
        POP     AX              ;;A BEFOLYASOLT REGISZTER VISSZAALLITASA
        ENDM
```

```
STACK   SEGMENT PARA STACK      ;VEREMSZEGMENS DEFINIALASA
        DB      64 DUP ('MYSTACK ')
STACK   ENDS                    ;A VEREMSZEGMENS VEGE

MYDATA  SEGMENT PARA 'DATA'
PLUS    DB      '+$'
TIMES   DB      '*$'
EQUAL   DB      '=$'
MENU1   DB      '     A MUVELET KIVALASZTASAHOZ IRJA BE A KOVETKEZO KARAKTEREK VALAMELYIKET: '
        DB      '                                                                            '
        DB      '                              A - KET SZAM OSSZEADASA '
        DB      '                                                                            '
        DB      '                              M - KET SZAM OSSZESZORZASA '
        DB      '                                                                            '
        DB      '                              E - A PROGRAM BEFEJEZESE '
        DB      '                                                                            '
        DB      '$'
MENU2   DB      '  IRJA BE AZ ELSO SZAMOT (0-9) :    $'
MENU3   DB      '  IRJA BE A MASODIK SZAMOT (0-9) :  $'
VALUE   DB      ?
POSTION DW      ?
OPER    DB      ?
NUM1    DB      0
NUM2    DB      0
ANS     DB      0
MYDATA  ENDS
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

;A FOMENU MEGJELENITESE A KEPERNYON
MMENU:  CLEARSCREEN             ;KEPERNYOTORLES MAKROVAL
        CURSOR  0C00H           ;KURZORPOZICIONALAS MAKROVAL
        PRINTCHAR MENU1         ;A FOMENU MEGJELENITESE MAKROVAL

;BILLENTYUZET BEOLVASAS
KEY:    READKEY OPER
        CMP     OPER,'E'        ;A FELHASZNALO BE AKARJA FEJEZNI A PROGRAMOT?
        JNE     GETNU           ;HA NEM, AKKOR AZ ELSO SZAM BEKERESE
        JMP     END             ;HA IGEN, A PROGRAM BEFEJEZODIK
GETNU:  CLEARSCREEN             ;A MENU LETORLESE
        CURSOR  0C00H           ;KURZORMOZGATAS
        PRINTCHAR MENU2         ;AZ ELSO SZAM BEKERESE

;AZ ELSO SZAM BEOLVASASA
        READKEY NUM1            ;A KARAKTER BEOLVASASA
        SUB     NUM1,30H        ;ATVALTAS ASCII-BOL DECIMALISBA
        CURSOR  0C00H           ;KURZORMOZGATAS
        CLEARSCREEN             ;A KEPERNYO LETORLESE
        PRINTCHAR MENU3         ;A 3. MENU KOVETKEZIK
```

```
;A MASODIK SZAM BEOLVASASA
        READKEY NUM2            ;A KARAKTER BEOLVASASA
        SUB     NUM2,30H        ;ATVALTAS ASCII-BOL DECIMALISBA
        CLEARSCREEN             ;A KEPERNYO LETORLESE

;AZ ARITMETIKAI MUVELET VEGREHAJTASA
        ARITH                   ;OSSZEADAS VAGY SZORZAS VEGREHAJTASA
        MOV     AL,NUM1         ;NUM1 ELOKESZITESE A KINYOMTATASHOZ
        MOV     VALUE,AL
        MOV     POSTION,0C1FH
        CALL    PRINTIT
        CURSOR  0C22H
        CMP     OPER,'A'
        JNE     T
        PRINTCHAR PLUS          ;HA OSSZEADAS, AKKOR '+' JELET KELL NYOMTATNUNK
        JMP     MORE
T:      PRINTCHAR TIMES         ;HA SZORZAS, AKKOR '*' JELET KELL NYOMTATNUNK
MORE:   MOV     AL,NUM2         ;NUM2 ELOKESZITESE A KINYOMTATASHOZ
        MOV     VALUE,AL
        MOV     POSTION,0C24H
        CALL    PRINTIT
        CURSOR  0C27H
        PRINTCHAR EQUAL         ;AZ '=' JEL MEGJELENITESE
        MOV     AL,ANS          ;ANS ELOKESZITESE A KINYOMTATASHOZ
        MOV     VALUE,AL
        MOV     POSTION,0C29H
        CALL    PRINTIT
        DELAY   25H             ;VARAKOZAS A BEOLVASASRA
        JMP     MMENU           ;UGRAS A FOMENURE ES PROGRAMISMETLES
END:
        RET                     ;VISSZATERES A DOS-BA
MYPROC  ENDP                    ;AZ MYPROC ELJARAS VEGE
PRINTIT PROC    NEAR            ;A SZAMOK MEGJELENITESE ELJARASSAL
        PUSH    AX              ;A BEFOLYASOLT REGISZTEREK ELMENTESE
        PUSH    CX
        PUSH    DX
        MOV     AL,VALUE        ;A SZAM BETOLTESE
        AND     AL,0F0H         ;A FELSO NEGY BIT KIEMELESE ES
        MOV     CL,4            ;ROTALASA
        ROR     AL,CL           ;AZ ALSO NEGY BITPOZICIOBA
        CURSOR  POSTION         ;KURZORPOZICIONALAS
        PRINTNUM                ;A SZAM MEGJELENITESE
        MOV     AL,VALUE        ;A SZAM BETOLTESE
        AND     AL,0FH          ;AZ ALSO NEGY BITET TARTJUK MEG
        ADD     POSTION,01H     ;KINYOMTATAS AZ ELSO SZAMTOL BALRA KEZDVE
        CURSOR  POSTION         ;KURZORPOZICIONALAS
        PRINTNUM                ;A SZAM MEGJELENITESE
        POP     DX              ;A BEFOLYASOLT REGISZTEREK VISSZAALLITASA
        POP     CX
        POP     AX
        RET                     ;AZ ELJARAS VEGE
PRINTIT ENDP

MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END         MYPROC      ;A PROGRAM VEGE
```

8–4. ábra Egyjegyű számok összeadása vagy összeszorzása menüvezérelt program segítségével

Láthatjuk, hogy a makrokönyvtáron kívül még két további makrót is alkalmazunk. Azt is vegyük észre, hogy egy NEAR típusú eljárással helyezzük el az eredményeket a monitor képernyőjén. Bizonyára felmerült az Olvasóban a kérdés: Az ARITH és READKEY rutinokból miért készítettünk makrót, a PRINTIT-ből pedig eljárást? A READKEY mindössze öt kódsorból áll, és a programban csak háromszor hívjuk meg. Természetes tehát, hogy ebből makro készült. Az ARITH és a PRINTIT rutinok kb. egyforma hosszúak. Melyik megoldást válasszuk? Az ARITH rutint csak egyszer hívjuk meg, tehát az ehhez tartozó kód csak egyszer jelenik meg a programban, akármelyik módszert is alkalmazzuk. Mivel a makro használatával soron belüli – azonnali, tehát gyorsabb – végrehajtást tudunk előidézni, ezért a makro mellett döntöttünk. A PRINTIT esetében a NEAR típusú eljárás szinte természetes. Az ehhez tartozó kód meglehetősen hosszú, és mivel háromszor hívjuk meg, jelentősen növelné a program méretét, ha makróként szerepelne. Mielőtt a program magyarázatát tovább folytatnánk, nézzük végig alaposan ezt a három rutint. Először az ARITH kódjával foglalkozunk.

```
ARITH   MACRO                   ;;ARITMETIKAI MAKRO
        PUSH    AX              ;;A BEFOLYASOLT REGISZTEREK ELMENTESE
        PUSH    CX
        SUB     AX,AX           ;;AX NULLAZASA
        MOV     AL,NUM1         ;;AZ ELSO SZAM ELHELYEZESE AX-BEN
        CMP     OPER,'A'        ;;OSSZEADAS VAGY SZORZAS ?
        JE      ADDER           ;;HA OSSZEADAS, AKKOR ADDER-RE UGRAS,
MULTP:  MUL     NUM2            ;;EGYEBKENT SZORZAS
        AAM                     ;;AZ EREDMENY ASCII KIIGAZITASA
        JMP     FINISH          ;;UGRAS FINISH-RE
ADDER:  ADD     AL,NUM2         ;;AZ OSSZEADAS ELOKESZITESE
        AAA                     ;;AZ EREDMENY ASCII KIIGAZITASA
FINISH: AND     AH,0FH          ;;AH-BOL CSAK AZ ALSO NEGY BITET TARTJUK MEG
        MOV     CL,04H          ;;A BITLEPTETES ELOKESZITESE
        SHL     AH,CL           ;;BITLEPTETES BALRA
        ADD     AL,AH           ;;PAKOLT SZAM ELOALLITASA (AL+AH)
        MOV     ANS,AL          ;;AZ EREDMENY ANS-BA KERUL
        POP     CX              ;;A BEFOLYASOLT REGISZTEREK VISSZAALLITASA
        POP     AX
        ENDM                    ;;A MAKRO VEGE
```

Az adatszegmens többek között a NUM1 és NUM2 byte-nak definiált változókat is tartalmazza. Ezek tárolják a felhasználó által beírt egyjegyű számokat. A szorzás kiválasztása az M, az összeadás kiválasztása pedig az A betű lenyomásával történik. Ezt a betűt az OPER (operandus) változóban tároljuk. A NUM1-ben szereplő értéket átmásoljuk az AL regiszterbe, és az A, ill. M előírásától függően elágaztatjuk a programot a MULTP vagy az ADDER címekre. Figyeljük meg, hogy a matematikai műveletek mindkét esetben pakolatlan decimális számokkal történnek, ehhez azonban a MUL művelet után egy AAM, ill. az ADD művelet után egy AAA utasításra van szükség. (Az AAM, ill. az AAA utasítások a számok ASCII kiigazítását szolgálják.) Az eredmény tárolása előtt a két decimális számjegyet szétválasztjuk és pakolva helyezzük el az AL regiszterbe. Ez az érték kerül aztán az ANS változóba.

A READKEY makro egy olyan műveletet valósít meg, amit már korábban is használtunk.

```
READKEY MACRO   THISKEY         ;;A BILLENTYU BELOLVASASA
        PUSH    AX              ;;A BEFOLYASOLT REGISZTER ELMENTESE
        MOV     AH,01H          ;;A MEGSZAKITASHOZ SZUKSEGES PARAMETER BEALLITASA
        INT     21H             ;;A BILLENTYUZETET BEOLVASO MEGSZAKITAS
        MOV     THISKEY,AL      ;;A MEGSZAKITASBOL SZARMAZO ERTEK ELMENTESE
        POP     AX              ;;A BEFOLYASOLT REGISZTER VISSZAALLITASA
        ENDM
```

A READKEY makro a THISKEY formális változót használja a billentyűzetről származó információ tárolására. Figyeljük meg, hogy amikor ezt a makrót meghívjuk, az OPER, a NUM1, vagy a NUM2 változók valamelyike a THISKEY helyére kerül. A READKEY a lenyomott billentyű ASCII kódját adja vissza a megfelelő változóba.

A PRINTIT eljárás kódja majdnem teljesen megegyezik azzal a másodpercek kijelzését szolgáló kóddal, amit a 8–2. ábrán látható programnál használtunk:

```
PRINTIT PROC    NEAR            ;A SZAMOK MEGJELENITESE ELJARASSAL
        PUSH    AX              ;A BEFOLYASOLT REGISZTEREK ELMENTESE
        PUSH    CX
        PUSH    DX
        MOV     AL,VALUE        ;A SZAM BETOLTESE
        AND     AL,0F0H         ;A FELSO NEGY BIT KIEMELESE ES
        MOV     CL,4            ;ROTALASA
        ROR     AL,CL           ;AZ ALSO NEGY BITPOZICIOBA
        CURSOR  POSTION         ;KURZORPOZICIONALAS
        PRINTNUM                ;A SZAM MEGJELENITESE
        MOV     AL,VALUE        ;A SZAM BETOLTESE
        AND     AL,0FH          ;AZ ALSO NEGY BITET TARTJUK MEG
        ADD     POSTION,01H     ;KINYOMTATAS AZ ELSO SZAMTOL BALRA KEZDVE
        CURSOR  POSTION         ;KURZORPOZICIONALAS
        PRINTNUM                ;A SZAM MEGJELENITESE
        POP     DX              ;A BEFOLYASOLT REGISZTEREK VISSZAALLITASA
        POP     CX
        POP     AX
        RET                     ;AZ ELJARAS VEGE
PRINTIT ENDP
```

A PRINTIT eljárás a két eredeti egyjegyű szám, és az eredmény megjelenítéséért egyaránt felelős. Mivel az eljárások nem teszik lehetővé a formális változók használatát, a VALUE változóban adjuk át az eljárásoknak azt a számot, amit ki akarunk íratni. Ez az eljárás először a szám magasabb helyértékű tagját nyomtatja ki, majd a kurzor áthelyezése után az alacsonyabb helyértékű tagot. A VALUE változóba pakolt decimális számot kell elhelyezni. Ez az eljárás a MACLIB.MAC könyvtárból két makrót használ. A CURSOR makróban a POSITION változóval határozható meg a nyomtatás kezdőpozíciója. A PRINTNUM csupán csak a pakolatlan decimális számot alakítja át ASCII-vé, ami a képernyőn való megjelenítéshez szükséges. Miért szükséges a számok pakolatlan, pakolt, majd ismét pakolatlan átalakítása? A választ a rendelkezésre álló rutinjainkban kell keresnünk:

– A szorzó és összeadó rutinjaink pakolatlan számokat igényelnek.
– Az AAA utasítás AL-ben, az AAM AL-ben és AH-ban helyezi el az eredményt. Itt tehát a pakolt alak mellett döntöttünk, mivel a PRINTIT kétjegyű pakolt BCD számot már képes megjeleníteni.

A felhasználó az adatszegmensben elhelyezett menük segítségével dönthet arról, hogy szorozni, összeadni kíván-e, vagy esetleg kilépni a programból.

```
;A FOMENU MEGJELENITESE A KEPERNYON
MMENU:  CLEARSCREEN             ;KEPERNYOTORLES MAKROVAL
        CURSOR  0C00H           ;KURZORPOZICIONALAS MAKROVAL
        PRINTCHAR MENU1         ;A FOMENU MEGJELENITESE MAKROVAL

;BILLENTYUZET BEOLVASAS
KEY:    READKEY OPER
```

```
        CMP     OPER,'E'        ;A FELHASZNALO BE AKARJA FEJEZNI A PROGRAMOT?
        JNE     GETNU           ;HA NEM, AKKOR AZ ELSO SZAM BEKERESE
        JMP     END             ;HA IGEN, A PROGRAM BEFEJEZODIK
GETNU:  CLEARSCREEN             ;A MENU LETORLESE
        CURSOR  0C00H           ;KURZORMOZGATAS
        PRINTCHAR MENU2         ;AZ ELSO SZAM BEKERESE
```

A kódrészlet első három sora a képernyő letörléséről, a kurzorpozíció beállításáról és a főmenü megjelenítéséről gondoskodik. A következő rész a billentyűzet beolvasását végzi. Ha a beírt karakter nem az E betű – ami a program befejezését jelenti –, a program bekér egy számot a felhasználótól.

```
;AZ ELSO SZAM BEOLVASASA
        READKEY NUM1            ;A KARAKTER BEOLVASASA
        SUB     NUM1,30H        ;ATVALTAS ASCII-BOL DECIMALISBA
        CURSOR  0C00H           ;KURZORMOZGATAS
        CLEARSCREEN             ;A KEPERNYO LETORLESE
```

A fentihez hasonló az eljárás a második számnál is:

```
;A MASODIK SZAM BEOLVASASA
        READKEY NUM2            ;A KARAKTER BEOLVASASA
        SUB     NUM2,30H        ;ATVALTAS ASCII-BOL DECIMALISBA
        CLEARSCREEN             ;A KEPERNYO LETORLESE
```

Amint a program ezeket az információkat megkapja, a főrutin elvégzi az aritmetikai műveletet és előkészíti a képernyőt a kivitelhez:

```
;AZ ARITMETIKAI MUVELET VEGREHAJTASA
        ARITH                   ;OSSZEADAS VAGY SZORZAS VEGREHAJTASA
```

A főrutin meghívja az ARITH makrót, amelyhez a NUM1 és NUM2 változók szolgáltatják a bemenő adatokat. A kétszámjegyű pakolt BCD eredményt az ANS változóban kapjuk meg.

```
MOV     AL,NUM1         ;NUM1 ELOKESZITESE A KINYOMTATASHOZ
MOV     VALUE,AL
MOV     POSTION,0C1FH
CALL    PRINTIT
```

A megelőző kódrészlet a NUM1-ben tárolt számot átadja a PRINTIT eljárásnak. A kurzort a 0C1FH pozícióba állítjuk. Ez a négy sor tehát a felhasználó által beírt első számot a képernyő középpontjától kissé balra helyezi el, majd pedig a plusz- vagy a mínuszjel megjelenítése következik.

```
        CURSOR  0C22H
        CMP     OPER,'A'
        JNE     T
        PRINTCHAR PLUS          ;HA OSSZEADAS, AKKOR '+' JELET KELL NYOMTATNUNK
        JMP     MORE
T:      PRINTCHAR TIMES         ;HA SZORZAS, AKKOR '*' JELET KELL NYOMTATNUNK
```

Az összeadás szimbólumát az adatszegmens PLUS elnevezésű változójában tároltuk, míg a szorzás jelét a TIMES változóban. Mivel mindkettő karakteres adat, a PRINTCHAR makróval az éppen érvényes kurzorpozícióban kijelezhető.

```
MORE:   MOV     AL,NUM2         ;NUM2 ELOKESZITESE A KINYOMTATASHOZ
        MOV     VALUE,AL
        MOV     POSTION,0C24H
        CALL    PRINTIT
```

Ezek után a második szám megjelenítése következik a PRINTIT makróval, valamint az egyenlőségjel kinyomtatásának előkészítése. (Az egyenlőségjelet az EQUAL elnevezésű változóban tároltuk karakteres adatként.)

```
CURSOR  0C27H
PRINTCHAR EQUAL         ;AZ '=' JEL MEGJELENITESE
```

Végül az eredményt az egyenlőségjeltől jobbra nyomtatjuk ki:

```
MOV     AL,ANS          ;ANS ELOKESZITESE A KINYOMTATASHOZ
MOV     VALUE,AL
MOV     POSTION,0C29H
CALL    PRINTIT
```

Az eredmény a képernyőn bizonyos előre meghatározott ideig látható és ezután a program visszatér a főmenübe.

```
DELAY   25H             ;VARAKOZAS A BEOLVASASRA
JMP     MMENU           ;UGRAS A FOMENURE ES PROGRAMISMETLES
```

Az egyszerűbb késleltetési rutint használjuk, mivel a késleltetés pontos ideje lényegtelen. A továbbiakban néhány ötletet adunk a program bővítéséhez.

1. Az elvégezhető műveletek sora kiterjeszthető mind a négy alapműveletre (egész számokkal).
2. Tegyük lehetővé, hogy az aritmetikai feladat megoldását a felhasználó beírhassa, amit a számítógép által meghatározott eredménnyel össze lehet vetni, és pontozni.
3. A program felkészíthető kétjegyű számok fogadására. (L. a 9. fejezet négyzetre emelést elvégző programját.)

Legyünk nagyon óvatosak, mert ez az utóbbi módosítás nem egyszerű. A kétjegyű szám bekérése még viszonylag könnyű feladat. Az összeadás és a szorzás, valamint az eredmény kijelzése a képernyőre azonban komoly programozási feladatot jelent.

8.4 FEJLETT KARAKTERLÁNC-KEZELŐ UTASÍTÁSOK HASZNÁLATA

5. feladat: Készítsünk egy olyan menüvezérelt programot, ami a felhasználó által beírt szó helyesírását ellenőrzi.

Ez a program teljesen különbözik az előbbi menüvezérelt programtól. Az eltérés nem a menü elkészítésének, hanem az adatok beírásának és programon belüli felhasználásának módjában van. Számos programozási problémát kell most is megoldanunk:

– Hogyan kérjük be a szavakat a billentyűzetről és hogyan tároljuk?

- Hogyan befolyásolja a szó hossza az adat inputját?
- Hol helyezzük el azt a szótárat, amelyben a szavakat tároljuk?

A program nem lesz túlságosan bonyolult, ui. a szavak abc sorrendbe rendezését nem végzi el, valamint lineáris keresést valósít meg, vagyis a felhasználó által beírt szót a szótár összes tételével összehasonlítja. A szótárban egyelőre csak nyolc szót helyezünk el, amelyek, mindegyike C betűvel kezdődik. A szavak száma természetesen bővíthető. A szótár méretének csak az adatszegmensek, ill. a szavakat beíró személy türelme szab határt. A program teljes listája a 8–5. ábrán látható.

```
;80286/80386-OS GEPEKRE KESZITETT PROGAM
;HELYESIRAS ELLENORZESE HATEKONY KARAKTERLANC-KEZELO
;UTASITASOK FELHASZNALASAVAL

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

IF1
        INCLUDE C:MACLIB.MAC    ;HASZNOS MAKROKAT TARTALMAZO EXTERNAL
ENDIF                           ;FILE CSATOLASA A PROGRAMHOZ

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'     ;A PROGRAM ADATTERULETE
NAME1   DB      11,11 DUP(' '),'$'
MAIN    DB      'KEREM IRJA BE A SZOT',0DH,0AH,'$'
MESS1   DB      0DH,0AH,'NINCS A SZOTARBAN!$'
MESS2   DB      0DH,0AH,'A HELYESIRAS KORREKT!$'
DICTION DB      'CAFRANG   ','CAKK      ','CAMMOG    '
        DB      'CEFRE     ','CELLULOID ','CENTRUM   '
        DB      'CIKLON    ','CIKLUS    '
MYDATA ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK,ES:MYDATA
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     ES,AX           ;AX ERTEKENEK ELHELYEZESE ES-BEN

;A PROGRAM FOMENUJE
        CLEARSCREEN             ;A KEPERNYO LETORLESE
        PRINTCHAR MAIN          ;A FOMENU MEGJELENITESE

;SZAVAK BEIRASA A BILLENTYUZETROL
        LEA     DX,NAME1        ;ITT TAROLJUK A BEIRT KAR.LANCOT
        MOV     AH,0AH          ;PARAMETER A BILLENTYUZET BEOLVASASAHOZ
        INT     21H             ;A BILLENTYUZET BEOLVASASA

;EZ A PROGRAMSZEGMENS A 10 KARAKTERNEL ROVIDEBB  SZAVAKBOL
;ELTAVOLITJA A KOCSIVISSZA KARAKTERT
        MOV     AL,NAME1+1      ;A KARAKTERSZAMLALO BEKERESE
```

```
        ADD     AL,2            ;ELTOLAS 2-VEL
        CBW                     ;KIBOVITES SZOVA
        MOV     BX,AX           ;AX ERTEKENEK ELHELYEZESE BX-BEN
        MOV     NAME1[BX],' '   ;A KOCSIVISSZA KARAKTERT EGY SZOKOZZEL
                                ;IRJUK FELUL

;A BEIRT SZO ES A SZOTARBAN SZEREPLO SZAVAK OSSZEHASONLITASA
        CLD                     ;A DIREKCIOS JELZO BEALLITASA
        MOV     BX,00H          ;BX -MINT INDEX- NULLAZASA
        MOV     AX,00H          ;A SZAVAK SZAMANAK KEZDOERTEKE ZERUS
NXTWRD: MOV     CL,0AH          ;A SZAVAK MAX. MERETENEK BEALLITASA
        LEA     DI,NAME1+2      ;AZ ADAT EFFEKTIV CIMENEK BETOLTESE
        LEA     SI,DICTION[BX]  ;A SZOTAR HELYZETE+BX ELTOLAS
        REPE    CMPSB           ;A SZOTAR 10 BYTE-JANAK
                                ;OSSZEHASONLITASA
        JE      CONGRD          ;MIND A 10 MEGEGYEZIK?
        ADD     BX,0AH          ;HA NEM, AKKOR AZ ELTOLASI ERTEK
                                ;BEALLITASA A KOVETKEZO SZOHOZ
        ADD     AX,01H          ;A SZAVAK SZAMLALOJAT NOVELJUK
        CMP     AX,08H          ;VEGEZTUNK MAR MINDEGYIK SZOVAL?
        JNE     NXTWRD          ;HA NEM, AKKOR ISMETLES

;A MEGFELELO UZENET MEGJELENITESE A KEPERNYON
        PRINTCHAR MESS1         ;A SZO NEM SZEREPEL A SZOTARBAN
        JMP     OUT
CONGRD: PRINTCHAR MESS2         ;A SZO SZEREPEL A SZOTARBAN
OUT:

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–5. ábra Helyesírást ellenőrző program készítése fejlett karakterlánc-kezelő utasítások alkalmazásával

Először az adatszegmenssel foglalkozunk.

```
MYDATA  SEGMENT PARA 'DATA'     ;A PROGRAM ADATTERULETE
NAME1   DB      11,11 DUP(' '),'$'
MAIN    DB      'KEREM IRJA BE A SZOT',0DH,0AH,'$'
MESS1   DB      0DH,0AH,'NINCS A SZOTARBAN!$'
MESS2   DB      0DH,0AH,'A HELYESIRAS KORREKT!$'
DICTION DB      'CAFRANG   ','CAKK      ','CAMMOG    '
        DB      'CEFRE     ','CELLULOID ','CENTRUM   '
        DB      'CIKLON    ','CIKLUS    '
MYDATA ENDS
```

A NAME1 változót a felhasználói bevitelnek tartjuk fenn. A könyvtárban szereplő szavak hosszúságát tíz karakterre korlátozzuk. (Ez a szám természetesen növelhető.) Amikor a felhasználó által beírt szót összehasonlítjuk a szótár szavaival, a következő válaszok egyike jelenik meg:

NINCS A SZOTARBAN
A HELYESIRAS KORREKT

(Ha a szó nem szerepel a könyvtárban, attól még lehetséges, hogy a helyesírása korrekt.) A menü csak egyetlen sorból áll: KEREM IRJA BE A SZOT.

A program a következő három fő részre osztható:

- Adatbevitel.
- A szavak összehasonlítása.
- Üzenet megjelenítése.

Az adatbevitelre vonatkozó programrész hasonlít ahhoz, amit az 5. fejezetben alkalmaztunk:

```
;SZAVAK BEIRASA A BILLENTYUZETROL
      LEA     DX,NAME1          ;ITT TAROLJUK A BEIRT KAR.LANCOT
      MOV     AH,0AH            ;PARAMETER A BILLENTYUZET BEOLVASASAHOZ
      INT     21H               ;A BILLENTYUZET BEOLVASASA
```

A 21H DOS megszakítás a billentyűzetről származó inputot a NAME1 változóban helyezi el. A beírható betűk száma kizárólag a NAME1 inicializálásától függ. Jelen esetben a NAME1 változót 11 szóközzel inicializáltuk, és így 10 karakter beírását tettük lehetővé, mivel a 21H megszakítás az AH-ban beállított 0AH paraméterrel a beírt szó karakterszámát az első pozícióban adja meg. Amikor a betűk száma a megadott maximumnál kisebb, a karakterláncba egy kocsivissza karakter is belekerül, amit – mivel a szótárban a szavak enélkül szerepelnek – el kell távolítani a szó végéről. Ezt a műveletet a következő rutin végzi el:

```
;EZ A PROGRAMSZEGMENS A 10 KARAKTERNEL ROVIDEBB  SZAVAKBOL
;ELTAVOLITJA A KOCSIVISSZA KARAKTERT
      MOV     AL,NAME1+1        ;A KARAKTERSZAMLALO BEKERESE
      ADD     AL,2              ;ELTOLAS 2-VEL
      CBW                       ;KIBOVITES SZOVA
      MOV     BX,AX             ;AX ERTEKENEK ELHELYEZESE BX-BEN
      MOV     NAME1[BX],' '     ;A KOCSIVISSZA KARAKTERT EGY SZOKOZZEL
                                ;IRJUK FELUL
```

A NAME1 változó definiciójában szereplő első 11-es szám a NAME1 által elfogadható karakterek maximális számát, míg a második 11-es szám a változó feltöltését jelenti 11 szóközzel. A $ egy határoló jel, ami a DOS megszakításhoz szükséges. (Tehát a NAME1 a karakterek maximális számát, 11 szóközt és a $ jelet tartalmazza.) A felhasználó által beírt karakterek aktuális száma a NAME1 + 1 pozícióban van. Ahhoz, hogy a kurzort az utolsó betű utáni helyre (a kocsivissza helyre) állítsuk, 2 byte-os eltolást kell alkalmaznunk. Jogosan merül fel a kérdés: Az eltolási érték miért 2 byte-os és nem 1 byte? A programnak az első 11-es számot is át kell ugrania. A NAME1 változón belüli indexelést a BX regiszterrel valósítjuk meg, ezért AL-t a CBW utasítással szó méretűvé alakítjuk át. Utolsó lépésként a kocsivissza karaktert szóközzel cseréljük fel. A felhasználó által beírt szó összehasonlítását a szótárban szereplő szavakkal a következő kódok teszik lehetővé:

```
;A BEIRT SZO ES A SZOTARBAN SZEREPLO SZAVAK OSSZEHASONLITASA
      CLD                       ;A DIREKCIOS JELZO BEALLITASA
      MOV     BX,00H            ;BX -MINT INDEX- NULLAZASA
      MOV     AX,00H            ;A SZAVAK SZAMANAK KEZDOERTEKE ZERUS
NXTWRD: MOV   CL,0AH            ;A SZAVAK MAX. MERETENEK BEALLITASA
      LEA     DI,NAME1+2        ;AZ ADAT EFFEKTIV CIMENEK BETOLTESE
      LEA     SI,DICTION[BX]    ;A SZOTAR HELYZETE+BX ELTOLAS
      REPE    CMPSB             ;A SZOTAR 10 BYTE-JANAK
                                ;OSSZEHASONLITASA
```

```
JE      CONGRD          ;MIND A 10 MEGEGYEZIK?
ADD     BX,0AH          ;HA NEM, AKKOR AZ ELTOLASI ERTEK
                        ;BEALLITASA A KOVETKEZO SZOHOZ
ADD     AX,01H          ;A SZAVAK SZAMLALOJAT NOVELJUK
CMP     AX,08H          ;VEGEZTUNK MAR MINDEGYIK SZOVAL?
JNE     NXTWRD          ;HA NEM, AKKOR ISMETLES
```

A felhasználó és a könyvtár szavait a CMPSB utasítás segítségével hasonlítjuk össze. Ha a program nem talál a karakterek között teljes egyezést, akkor a következő szót veszi elő a könyvtárból.

Az eljárás mindaddig ismétlődik, amíg a rutin nem talál a beírt szóval megegyezőt, ill. amíg el nem fogynak a könyvtár szavai. A BX regiszter tartalmazza a könyvtáron belüli eltolási értéket. A könyvtár következő tételére való rámutatáshoz, BX-et az egyes szavak hosszával meg kell növelnünk. Minden szó 10 karakter hosszú. (A helyet kitöltő szóközök is számítanak.) Az AX regiszter tartja nyilván az összehasonlított szavak számát, amelyet a szótárban szereplő összes szó számával (jelen esetben 8-cal) vetünk össze. A CL-ben szerepel az összehasonlítandó byte-ok száma (jelen esetben 10). A NAME1 változó effektív címe NAME1 + 2, mivel a betűk maximális számát (az első 11-et!) és a változó első karakterpozíciójában tárolt aktuális karakterszámot át kell ugranunk. A program azonnal kilép a rutinból, amint egyező szavakat talál, és a szükséges üzenetet is megjeleníti. Ellenkező esetben BX-et tízzel (0AH-val) növeli, és a szótár következő tíz byte-ját hasonlítja össze az általunk beírt szóval. A program a megfelelő üzenettel jelez, ha nem találja a beírt szó párját. Szótárunknak – a jelenlegi formájában – két fő hiányossága van:

1. Mivel a szavak sorrendbe rendezését nem oldottuk meg, a beírt szavakat a szótár minden tételével össze kell hasonlítanunk, olyan esetben is, amikor a szó egyáltalán nem szerepel a szótárban.
2. A szótárban szereplő minden egyes tételt 10 karakterre kellett kiegészítenünk.

Első látásra talán úgy tűnik, hogy egyszerűbb lenne, ha a szavakat nem egészítenénk ki, és csak a betűket ellenőriznénk. Miért nem végzünk csak annyi összehasonlítást, ahány karaktert a billentyűzetről beírtunk? Ezzel az elképzeléssel az lenne a baj, hogy ha pl. a cellu szót írjuk be, akkor a program helyesnek értelmezi annak ellenére, hogy ilyen szó nincs is a szótárban. (A celluloid szó szerepel csak a szótárban.) Az első öt betű összehasonlítása tehát helyes eredményt adna. A program tehát továbbfejleszthető, de minden lépést nagyon alaposan fontoljunk meg.

8.5 LEMEZFILE-OK LÉTREHOZÁSA ÉS HASZNÁLATA

A következő három program különböző file-kezelési eljárásokat mutat, ami minden nyelvben alapvető fontosságú, viszont sok assembly nyelvet oktató könyv ezzel a témával nem, vagy csak felületesen foglalkozik. Igyekeztünk ezért ezt a hiányt pótolni.

A DOS 2.0 változata az assembly programok file-kezelési technikájában jelentős változást hozott. Az még mindig igaz, hogy a file-ok kezelése BIOS szinten történik, azonban most már nem ez az elsődleges módszer. A DOS megszakítások több komplett szolgáltatás igénybevételét teszik lehetővé. Az 1.0-s és 1.1-es DOS verziók esetén az FCB-re (File Control Block) volt

szükség, ha a 21H megszakítással el akartuk érni a file-okat. (AH = 0FH-tól 29H-ig). Az FCB különböző információkat tartalmazott a file-ról (meghajtószám, file-név, kiterjesztés, éppen érvényes blokkszám, rekordméret, file-méret, dátum, rekordszám, relatív rekordszám). Az FCB-ről a 3.0-s DOS verzióig részletes leírás található a DOS kézikönyvben. (A DOS 3.0 feletti verziószámtól kezdve ezt az információt egy különálló DOS kezelői kézikönyv tartalmazza.) Bár az FCB-vel a programozó az egyes file-okfelett szabadon rendelkezhetett, meglehetősen nehézkes megoldásokhoz kellett folyamodnia minden egyes file-elérés során. A jelenlegi DOS verziók még mindig támogatják ezeket a megszakítási lehetőségeket, de használatukat már nem javasoljuk.

A 2.0-s, és ez feletti DOS verzióknál a file-ok eléréséhez már nem szükséges FCB-t használnunk. A file-kezelésre vonatkozó DOS szolgáltatások még mindig használják a 21H megszakítást, de a AH = 39H-tól AH = 46H-ig terjedő paraméterekkel. (Az 5. fejezetben részlegesen szerepel a 21H DOS megszakítás paraméterezése.) Ha a file-t ezekkel a kiemelt DOS utasításokkal hozzuk létre, a file-hoz egy ún. file-kezelő (file-handle) létesül, amit a file nyitása után a program felhasználhat. A file-kezelőt az AX regiszterben kapja vissza a program a file megnyitása után (l. OPENFIL makro). A program minden házimunkát – amit régebben az FCB-vel valósítottunk meg – automatikusan elvégez, de ennek az az ára, hogy elveszítjük az uralmat az FCB felett.

Ha a file létrehozása, megnyitása, olvasása, írása és lezárása valamilyen oknál fogva nem lehetséges, AX-be egy hibakód kerül. A DOS 2.0-ban szereplő 18 hibaüzenet a 3.0-s és ez feletti DOS változatoknál 39-re bővült. Pl. az AX = 1 érvénytelen műveleti kódot jelent, az AX = 2 pedig azt, hogy a rendszer a file-t nem találja. A hibakódok részletes ismertetését a DOS felhasználói kézikönyvében találjuk.

A 6. feladatban egy lemezfile-t hozunk létre, a 7. feladatban megnyitjuk a file-t, írunk bele és lezárjuk, míg a 8. feladatban a felhasználó részére lehetővé tesszük a file olvasását.

---

6. feladat: lemezfile létrehozása.

A 8–6. ábrán bemutatott programból láthatjuk, hogy a tovább- fejlesztett DOS parancsok segítségével a file-ok létrehozása viszonylag egyszerű.

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;FILE LETREHOZASA TOVABBFEJLESZTETT DOS
;SZOLGALTATASOK SEGITSEGEVEL

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

CRETFIL MACRO                   ;;A FILE LETREHOZASAT SZOLGALO MAKRO
        MOV     AH,3CH          ;;A FILE LETREHOZASAHOZ SZUKSEGES
                                ;;DOS PARAMETER BEALLITASA
        MOV     CX,00H          ;;NORMAL FILE ATTRIBUTUM
        LEA     DX,FILENAM      ;;A FILE NEVENEK BETOLTESE DX-BE
        INT     21H             ;;A LETREHOZASI MUVELET
        ENDM

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
FILENAM DB      'A:NUMBERS.DAT',0 ;A FILE NEVE
MYDATA  ENDS
```

```
MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        CRETFIL                 ;A FILE LETREHOZASA

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–6. ábra File létrehozása továbbfejlesztett DOS segítségével

lünk – csupán csak a CRETFIL makrót hívja meg és a munka máris elkészül. (Az AH = 3CH paraméter beállítása szükséges.)

```
CRETFIL MACRO                   ;;A FILE LETREHOZASAT SZOLGALO MAKRO
        MOV     AH,3CH          ;;A FILE LETREHOZASAHOZ SZUKSEGES
                                ;;DOS PARAMETER BEALLITASA
        MOV     CX,00H          ;;NORMAL FILE ATTRIBUTUM
        LEA     DX,FILENAM      ;;A FILE NEVENEK BETOLTESE DX-BE
        INT     21H             ;;A LETREHOZASI MUVELET
```

A CRETFIL makro állítja be a 21H DOS megszakítással történő file-nyitási vagy -létrehozási művelethez a paramétereket. A CX-ben tárolt érték a file attributumát határozza meg a következők szerint:

0H – Normál file
1H – Csak olvasható file
2H – Rejtett file
4H – Rendszerfile
8H – Lemezcimke
10H – Alkatalógus
20H – Archív file

A FILENAM arra az ASCIIZ karakterláncra mutat az adatszegmensben, ami a meghajtót az útvonalnevet, a file-nevet és a kiterjesztést tartalmazza.

```
MYDATA  SEGMENT PARA 'DATA'
FILENAM DB      "A:NUMBERS.DAT",0 ;A FILE NEVE
MYDATA  ENDS
```

A következő feladatban (a program a 8–7. ábrán látható) a felhasználó a B:NUMBERS.DAT elnevezésű file-ba neveket és telefonszámokat helyezhet el.

A file kiterjesztése után álló zérus egy elhatároló jel, ami a karakterlánc végét jelzi. A CRETFIL makro az AH = 3CH paraméter hatására létrehozza a file-t, és a hosszát zérusra állítja be. (Emiatt csak új file létrehozására, vagy a már meglévő törlésére használható). A file-kezelő az AX regiszterben kapjuk vissza. Kiváló programozási gyakorlat, ha a korábbi progamok felhasználásával a billentyűzetről bevitt adatokat írunk a létrehozott file-ba.

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;ELOZETESEN LETREHOZOTT FILE MEGNYITASA,
;IRAS A FILE-BA ES A FILE LEZARASA

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

IF1
      INCLUDE C:MACLIB.MAC      ;HASZNOS FUNKCIOKAT TARTALMAZO
ENDIF                           ;EXTERNAL FILE CSATLAKOZTATASA

OPENFIL MACRO                   ;;ELOZETESEN LETREHOZOTT FILE
                                ;;MEGNYITASA MAKROVAL
      MOV     AL,01H            ;;NYITAS CSAK IRASHOZ
      MOV     AH,3DH            ;;PARAMETERBEALLITAS
      LEA     DX,FILENAM        ;;A FILE NEVENEK BEKERESE
      INT     21H               ;;MAGA A NYITASI MUVELET
      MOV     FHAND,AX          ;;A FILE-KEZELO ELMENTESE
      ENDM

INDEXNUM MACRO                  ;;EZ A MAKRO A FILE VEGET KERESI MEG
      MOV     AL,02H            ;;PARAMETERBEALLITAS
      MOV     AH,42H
      MOV     BX,FHAND
      MOV     CX,0H
      MOV     DX,0H
      INT     21H               ;;AZ INDEX BEKERESE
      ENDM

WRITFIL MACRO                   ;;FILE-BA IRAS MAKROVAL
      MOV     AH,40H            ;;AZ IRAS ELOKESZITESE
      MOV     BX,FHAND          ;;A FILE-KEZELO ELHELYEZESE
                                ;;BX-BEN
      MOV     CX,80             ;;AZ IRASRA KERULO BYTE-OK SZAMA
      LEA     DX,NAMEFLD        ;;RAMUTATAS A KARAKTEREKET TARTALMAZO
                                ;;MEZORE
      INT     21H               ;;MAGA AZ IRASI MUVELET
      ENDM

CLOSFIL MACRO                   ;;A FILE LEZARASA MAKROVAL
      MOV     AH,3EH            ;;A LEZARASI MUVELET ELOKESZITESE
      MOV     BX,FHAND          ;;A FILE-KEZELO ELHELYEZESE
                                ;;BX-BEN
      INT     21H               ;;A LEZARASI MUVELET
      ENDM

STACK   SEGMENT PARA STACK
      DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NAMEPAR LABEL   BYTE
MAXLEN  DB      81              ;AZ INFORMACIO MAX. HOSSZA+1
ACTLEN  DB      ?               ;AZ INFORMACIO AKTUALIS HOSSZA
NAMEFLD DB      80 DUP(' '),'$' ;AZ INFORMACIO, AMIT LEMEZEN
                                ;AKARUNK ELHELYEZNI
FILENAM DB      'A:NUMBERS.DAT',0 ;A FILE NEVE
```

```
FHAND   DW      ?               ;A FILE-KEZELO TAROLASAT
                                ;BIZTOSITO VALTOZO
PROMPT  DB      'NEV                                    TELEFONSZAM$'
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK,ES:MYDATA
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     ES,AX           ;AX TARTALMANAK ELHELYEZESE ES-BEN

        CLEARSCREEN             ;A KEPERNYO LETORLESE
        OPENFIL                 ;A FILE MEGNYITASA
        INDEXNUM
AGAIN:
        CALL    PASINFO         ;A BILLENTYUZETROL SZARMAZO INFORMACIO
                                ;ATADASA
        CMP     ACTLEN,00       ;HA NINCS BEVITEL, A PROGRAM BEFEJEZODIK,
        JNE     AGAIN           ;EGYEBKENT TOVABBI ADATOK BEKERESE
        CLOSFIL                 ;A FILE LEZARASA
        RET
MYPROC  ENDP

PASINFO PROC    NEAR            ;AZ INFORMACIO ATADASA A LEMEZFILE-BA
        CURSOR  0000H           ;A KURZOR BEALLITASA A LAP TETEJERE
        PRINTCHAR PROMPT        ;AZ ADATOK BEKERESE
        CURSOR  0100H           ;KURZORPOZICIONALAS AZ IRASHOZ
        MOV     AH,0AH          ;A BEVITEL ELFOGADASAHOZ SZUKSEGES
                                ;PARAMETER
        LEA     DX,NAMEPAR      ;A KAR.LANC HELYENEK BEKERESE
        INT     21H             ;A BEVITEL BEOLVASASA
        CLEARSCREEN             ;A LEGUTOLJARA BEIRT NEV TORLESE
        CMP     ACTLEN,00       ;VOLT BEVITEL?
        JE      ENDO            ;HA NEM, A PROGRAM BEFEJEZODIK
        MOV     BH,00           ;A KAR.LANC HOSSZA BX-BE KERUL
        MOV     BL,ACTLEN
        MOV     NAMEFLD[BX],' ' ;A LEMEZRE IRAS ELOKESZITESE
        WRITFIL                 ;IRAS A LEMEZFILE-BA

        CLD                     ;A IRANYJELZO BEALLITASA
        LEA     DI,NAMEFLD      ;A SZOKOZOK TAROLASANAK ELOKESZITESE
        MOV     CX,80           ;A NAMEFLD VALTOZOBA
        MOV     AL,20H          ;A SZOKOZ ASCII ERTEKE: 20H
        REP     STOSB           ;ELMENTES 80-SZOR
ENDO:

        RET
PASINFO ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE

MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–7. ábra Előzetesen létrehozott file megnyitása, írás a file-ba és a file lezárása

7. feladat: Lemezfile megnyitása, írás a file-ba és lezárás.

Egy-egy makrót használunk fel a file megnyitására a file végéhez mutató index beállítására, a file-ba történő írásra és a file lezárására. A 8–7. ábrán a teljes programlista látható. Ahhoz, hogy egy file-ba írni vagy egy file-ból olvasni tudjunk, a file-t először meg kell nyitni. Ha ekkor az AH = 3CH paramétert használnánk, a file hosszát zérusra állítanánk be, ami azt jelenti, hogy a file tartalmát töröljük. Ezért tehát az AH = 3DH paraméterrel nyitjuk meg a file-t.

```
OPENFIL MACRO                   ;;ELOZETESEN LETREHOZOTT FILE
                                ;;MEGNYITASA MAKROVAL
        MOV     AL,01H          ;;NYITAS CSAK IRASHOZ
        MOV     AH,3DH          ;;PARAMETERBEALLITAS
        LEA     DX,FILENAM      ;;A FILE NEVENEK BEKERESE
        INT     21H             ;;MAGA A NYITASI MUVELET
        MOV     FHAND,AX        ;;A FILE-KEZELO ELMENTESE
        ENDM
```

Az OPENFIL makróban az AL értékének beállításával határozhatjuk meg, hogy a file csak írható, csak olvasható vagy írható-olvasható legyen. Ha AL = 0, akkor a file csak olvasható, az AL = 1 jelenti a csak írható file-elérést, míg az AL = 2 mindkét variációt lehetővé teszi. Ez az információ az AL byte alsó három bitjét foglalja el. A felső öt bit a 3.0-s és az e feletti DOS verziókban a file-kezeléssel kapcsolatos feladatmegosztást támogatja. A FILENAM változóban azt a teljes file-nevet kell megadnunk, amit a file létrehozásakor használtunk. (A file teljes nevét a meghajtó, az útvonal, a file-név, a kiterjesztés és az elhatárolójel adja.) A FILENAM egy ASCIIZ karakterlánc, amit az adatszegmensben helyezünk el. A filekezelőt, amit az AX regiszterben kapunk vissza, az adatszegmens FHAND elnevezésű változójában tároljuk. Mielőtt a file-ba íráshoz hozzákezdenénk, meg kell határoznunk a file végét (EOF), különben az írás során tönkretehetjük az előzetesen tárolt információkat. Az INDEXNUM makro ezt a funkciót látja el.

```
INDEXNUM MACRO                  ;;EZ A MAKRO A FILE VEGET KERESI MEG
        MOV     AL,02H          ;;PARAMETERBEALLITAS
        MOV     AH,42H
        MOV     BX,FHAND
        MOV     CX,0H
        MOV     DX,0H
        INT     21H             ;;AZ INDEX BEKERESE
        ENDM
```

Ha a 21H DOS megszakítást az AH = 42H paraméterrel hívjuk meg, akkor lehetőségünk lesz arra, hogy a file-mutatót átállítsuk. Mielőtt azonban ténylegesen meghívnánk ezt a megszakítást, BX-be az előzetesen elmentett file-kezelőt kell betöltenünk, az AL-nek a mutató kiinduló helyzetét kell tartalmaznia. Ha AL = 0, akkor az eltolási értéket a file kezdőpontjától kell mérni. Ha AL = 1, akkor az eltolást az éppen érvényes helyzettől, AL = 2 esetében pedig file végétől kell számítani. Az eltolás értékét CX:DX regiszterpárnak kell tartalmaznia. CX általában zérus, kivéve, ha a file 64 K-nál nagyobb. Ha AL = 2, akkor a program keresi meg az eltolást (vagy a file-méretet), tehát CX-et és DX-et ekkor zérusra állítjuk be. Amikor a 21H megszakítást meghívjuk, a visszakapott eltolási érték DX:AX-be kerül. (Jelen esetben csak arra volt szükség, hogy a mutatót a file végére állítsuk.) Amikor továbbfejlesztett DOS szolgáltatással írunk egy megnyitott file-ba, a WRITFIL makro beállítja a szükséges DOS paramétereket.

```
WRITFIL MACRO                   ;;FILE-BA IRAS MAKROVAL
        MOV     AH,40H          ;;AZ IRAS ELOKESZITESE
        MOV     BX,FHAND        ;;A FILE-KEZELO ELHELYEZESE
                                ;;BX-BEN
        MOV     CX,80           ;;AZ IRASRA KERULO BYTE-OK SZAMA
        LEA     DX,NAMEFLD      ;;RAMUTATAS A KARAKTEREKET TARTALMAZO
                                ;;MEZORE
        INT     21H             ;;MAGA AZ IRASI MUVELET
        ENDM
```

Az AH = 40H a DOS írás-a-file-ba paramétere. BX-et ismét az előzetesen tárolt file-kezelővel töltjük fel. CX-be kerül a file-ba írandó byte-ok száma, ami a jelenlegi programban minden egyes híváskor egy teljes szövegsort jelent. A DS:DX regiszterpárral mutatunk rá a byte-ok helyére. Ebben a programban a NAMEFLD változót az adatszegmensben helyeztük el, és 80 szóközzel, valamint a $ határolójellel inicalizáltuk.

```
MYDATA  SEGMENT PARA 'DATA'
NAMEPAR LABEL   BYTE
MAXLEN  DB      81              ;AZ INFORMACIO MAX. HOSSZA+1
ACTLEN  DB      ?               ;AZ INFORMACIO AKTUALIS HOSSZA
NAMEFLD DB      80 DUP(' '),'$' ;AZ INFORMACIO, AMIT LEMEZEN
                                ;AKARUNK ELHELYEZNI
FILENAM DB      'A:NUMBERS.DAT',0 ;A FILE NEVE
FHAND   DW      ?               ;A FILE-KEZELO TAROLASAT
                                ;BIZTOSITO VALTOZO
PROMPT  DB      'NEV                                    TELEFONSZAM$'
MYDATA  ENDS
```

Ha az írási művelettel elkészültünk, a file-t le kell zárnunk. Ezt végzi el a CLOSFIL makro az AH = 3EH paraméterrel.

```
CLOSFIL MACRO                   ;;A FILE LEZARASA MAKROVAL
        MOV     AH,3EH          ;;A LEZARASI MUVELET ELOKESZITESE
        MOV     BX,FHAND        ;;A FILE-KEZELO ELHELYEZESE
                                ;;BX-BEN
        INT     21H             ;;A LEZARASI MUVELET
        ENDM
```

A file-kezelőt elő kell írni, azonban a FILENAM információt nem. Másképp fogalmazva: BX-nek a legutoljára megnyitott file file-kezelő kódját kell tartalmaznia. Ha a kódszegmens fejrészét gondosan végignézzük, bizonyára észrevesszük, hogy ez a rogram fejlett karakterlánc-kezelő utasításokat használt. (A 8–7. ábrán keressük meg a REP STOSB utasítást.) A főprogram meglehetősen rövid, csak néhány sorból áll:

```
        CLEARSCREEN             ;A KEPERNYO LETORLESE
        OPENFIL                 ;A FILE MEGNYITASA
        INDEXNUM
AGAIN:
        CALL    PASINFO         ;A BILLENTYUZETROL SZARMAZO INFORMACIO
                                ;ATADASA
        CMP     ACTLEN,00       ;HA NINCS BEVITEL, A PROGRAM BEFEJEZODIK,
        JNE     AGAIN           ;EGYEBKENT TOVABBI ADATOK BEKERESE
        CLOSFIL                 ;A FILE LEZARASA
        RET
MYPROC  ENDP
```

A képernyőt a MACLIB.MAC könyvtárban szereplő CLEARSCREEN makróval először letöröljük. A B:NUMBERS.DAT file-t megnyitjuk, (ez a file a 8–6. ábrán látható program futtatásával hozható létre) és a filemutatót az INDEXNUM makro segítségével a file végére állítjuk. A NEAR típusu PASINFO eljárás lehetővé teszi, hogy a felhasználó által a billentyűzeten beírt karakterek a NAMEFLD változóba, majd pedig a file-ba kerüljenek. Ha csak a kocsivissza karaktert (0DH) írjuk be, akkor a program a CLOSEFIL makro futtatásával befejeződik.

A PASINFO eljárás számos műveletet tartalmaz:

```
PASINFO PROC    NEAR            ;AZ INFORMACIO ATADASA A LEMEZFILE-BA
        CURSOR  0000H           ;A KURZOR BEALLITASA A LAP TETEJERE
        PRINTCHAR PROMPT        ;AZ ADATOK BEKERESE
        CURSOR  0100H           ;KURZORPOZICIONALAS AZ IRASHOZ
```

Az első néhány kódsor a kurzort a bal felső sarokba állítja be. Ehhez a művelethez MACLIB.MAC könyvtár CURSOR makróját használjuk fel. A PRINTCHAR makro – ugyanebből a könyvtárból – a képernyő tetejére írja a felhasználó részére szánt üzenetet.

Az adatszegmensből már látható, hogy ez az üzenet a nevet és a telefonszámot kéri be. (Valójában a felhasználó 80 byte terjedelemben bármilyen információt beírhat.) Ezután az input bekéréséhez a kurzort a következő sor elejére állítjuk.

```
MOV     AH,0AH          ;A BEVITEL ELFOGADASAHOZ SZUKSEGES
                        ;PARAMETER
LEA     DX,NAMEPAR      ;A KAR.LANC HELYENEK BEKERESE
INT     21H             ;A BEVITEL BEOLVASASA
CLEARSCREEN             ;A LEGUTOLJARA BEIRT NEV TORLESE
CMP     ACTLEN,00       ;VOLT BEVITEL?
JE      ENDO            ;HA NEM, A PROGRAM BEFEJEZODIK
MOV     BH,00           ;A KAR.LANC HOSSZA BX-BE KERUL
MOV     BL,ACTLEN
MOV     NAMEFLD[BX],' ' ;A LEMEZRE IRAS ELOKESZITESE
WRITFIL                 ;IRAS A LEMEZFILE-BA
```

Az adatszegmens NAMEPAR LABEL BYTE definiciója 80 karakteres bevitelt tesz lehetővé, amit aztán a NAMEFLD változóba helyez el. Csak emlélkeztetőül: Ez a megszakítás a karakterlánc hosszát a karakterlánc első byte-jában adja vissza, így a MAXLEN változót 81-re kell beállítani. Amint a karakterlánc beírása és tárolása megtörtént, a további adatbevitel előkészítéséhez a képernyőt letöröljük. Ezt követően a program megvizsgálja ACTLEN értékét, és eldönti, hogy csupán csak a kocsivissza karakterről van szó, ami a program befejezését jelenti, vagy pedig feldolgozandó információról. Az aktuális karakterlánc hossza a BX regiszterbe kerül, és a kocsivissza karakterét egy szóközzel írja felül a program. A WRITFIL makro átadja az információt a lemezre, és a file-mutatót automatikusan a file végére állítja.

Mielőtt azonban a további adatbevitelekre rátérnénk, egy kis takarítást kell végeznünk:

```
CLD                     ;A IRANYJELZO BEALLITASA
LEA     DI,NAMEFLD      ;A SZOKOZOK TAROLASANAK ELOKESZITESE
MOV     CX,80           ;A NAMEFLD VALTOZOBA
MOV     AL,20H          ;A SZOKOZ ASCII ERTEKE: 20H
REP     STOSB           ;ELMENTES 80-SZOR
```

A NAMEFLD még mindig az előzőleg bevitt adatokat tartalmazza. Ha ezt a tartalmat nem írjuk felül szóközökkel, és a következő adatbevitel az előzőnél rövidebb, akkor az első adattartalom maradékát a másodikkal együtt ismét tárolja a rendszer. Ennek elkerülése végett a REP STOPSB művelet felhasználásával 80 szóközzel (20H) töltjük fel a NAMEFLD változót.

A PASINFO ciklus mindaddig ismétlődik, amíg a felhasználó szöveg nélküli kocsivissza karaktert szerepeltet bevitelként. Ekkor a program lezárja a file-t és befejezi a futást. A létrehozott file – DOS szinten – a következők beírásával jeleníthető meg a képernyőn:

```
A>TYPE B:NUMBERS.DAT
```

A következő program azt mutatja be, hogyan lehet egy file besolvasását assembly szinten megoldani a 21H DOS megszakítás felhasználásával.

8. feladat: Lemezfile nyitása, olvasása és lezárása.

A két utóbbi programmal egy adatfile-t létesítettünk, amelybe a felhasználó az információit elhelyezte. Ezt a file-t fogjuk most megnyitni és az adatait beolvasni. A 8–8. ábrán a program

---

```
;8088/80386-OS GEPEKRE KESZITETT PROGRAM
;ELOZETESEN ELMENTETT FILE OLVASASA

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

IF1
        INCLUDE C:MACLIB.MAC    ;HASZNOS FUNKCIOKAT TARTALMAZO
ENDIF                           ;EXTERNAL FILE CSATLAKOZTATASA

OPENFIL MACRO                   ;;ELOZETESEN LETREHOZOTT FILE
                                ;;MEGNYITASA MAKROVAL
        MOV     AL,00H          ;;NYITAS CSAK OLVASASHOZ
        MOV     AH,3DH          ;;PARAMETERBEALLITAS
        LEA     DX,FILENAM      ;;A FILE NEVENEK BEKERESE
        INT     21H             ;;NYITASI MUVELET
        MOV     FHAND,AX        ;;A FILE-KEZELO ELMENTESE
        ENDM

READFIL MACRO                   ;;FILE OLVASASA MAKROVAL
        MOV     AH,3FH          ;;AZ OLVASAS ELOKESZITESE
        MOV     BX,FHAND        ;;A FILE-KEZELO ELHELYEZESE
                                ;;BX-BEN
        MOV     CX,80           ;;A BEOLVASASRA KERULO BYTE-OK SZAMA
        LEA     DX,NAMEFLD      ;;A BEOLVASANDO KARAKTEREK MUTATOJA
        INT     21H             ;;OLVASASI MUVELET
        ENDM

CLOSFIL MACRO                   ;;A FILE LEZARASA MAKROVAL
        MOV     AH,3EH          ;;A LEZARASI MUVELET ELOKESZITESE
        MOV     BX,FHAND        ;;A FILE-KEZELO KOD ELHELYEZESE
                                ;;BX-BEN
        INT     21H             ;;LEZARASI MUVELET
        ENDM

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NAMEPAR LABEL   BYTE
MAXLEN  DB      81              ;AZ INFORMACIO MAX. HOSSZA+1
```

```
ACTLEN  DB      ?               ;AZ INFORMACIO AKTUALIS HOSSZA
NAMEFLD DB      80 DUP(' '),'$' ;AZ INFORMACIO, AMIT LEMEZEN
                                ;AKARUNK ELHELYEZNI
FILENAM DB      'A:NUMBERS.DAT',0 ;A FILE NEVE
FHAND   DW      ?               ;A FILE-KEZELO KOD TAROLASAT
                                ;BIZTOSITO VALTOZO
PROMPT  DB      'NEV                                    TELEFONSZAM$'
POSITION DW     0100H
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        CLEARSCREEN             ;A KEPERNYO LETORLESE
        OPENFIL                 ;A FILE MEGNYITASA
AGAIN:
        CALL    PASINFO         ;LEMEZOLVASASI IGENY
        CMP     ACTLEN,00       ;HA NINCS INFORMACIO,
                                ;A PROGRAM BEFEJEZODIK,
        JNE     AGAIN           ;EGYEBKENT TOVABBI BEVITEL BEKERESE
        CLOSFIL                 ;A FILE LEZARASA
        RET
MYPROC  ENDP

PASINFO PROC    NEAR            ;AZ INFORMACIO BEOLVASASA A
                                ;LEMEZFILE-ROL
        CURSOR  0000H           ;A KURZOR BEALLITASA A LAP TETEJERE
        PRINTCHAR PROMPT        ;A BEVITEL BEKERESE
REPTN:  CURSOR  POSITION        ;KURZORPOZICIONALAS AZ IRASHOZ
        READFIL                 ;A FILE OLVASASA
        MOV     ACTLEN,AL
        CMP     ACTLEN,00       ;A BEVITEL ZERUS?
        JE      ENDO            ;HA IGEN, A PROGRAM BEFEJEZODIK
        PRINTCHAR NAMEPAR+2
        ADD     POSITION,0100H
        JMP     REPTN
ENDO:

        RET
PASINFO ENDP                    ;AZ ELJARAS VEGE

MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

8–8. ábra Előzetesen tárolt file beolvasása

listája látható. A programban három makrót és egy NEAR típusú eljárást használunk. Először a három makrót tanulmányozzuk.

```
OPENFIL MACRO                   ;;ELOZETESEN LETREHOZOTT FILE
                                ;;MEGNYITASA MAKROVAL
        MOV     AL,00H          ;;NYITAS CSAK OLVASASHOZ
        MOV     AH,3DH          ;;PARAMETERBEALLITAS
        LEA     DX,FILENAM      ;;A FILE NEVENEK BEKERESE
        INT     21H             ;;NYITASI MUVELET
        MOV     FHAND,AX        ;;A FILE-KEZELO ELMENTESE
        ENDM

READFIL MACRO                   ;;FILE OLVASASA MAKROVAL
        MOV     AH,3FH          ;;AZ OLVASAS ELOKESZITESE
        MOV     BX,FHAND        ;;A FILE-KEZELO ELHELYEZESE
                                ;;BX-BEN
        MOV     CX,80           ;;A BEOLVASASRA KERULO BYTE-OK SZAMA
        LEA     DX,NAMEFLD      ;;A BEOLVASANDO KARAKTEREK MUTATOJA
        INT     21H             ;;OLVASASI MUVELET
        ENDM

CLOSFIL MACRO                   ;;A FILE LEZARASA MAKROVAL
        MOV     AH,3EH          ;;A LEZARASI MUVELET ELOKESZITESE
        MOV     BX,FHAND        ;;A FILE-KEZELO KOD ELHELYEZESE
                                ;;BX-BEN
        INT     21H             ;;LEZARASI MUVELET
        ENDM
```

Az OPENFIL és CLOSEFIL makrókat közelebbről megvizsgálva látható, hogy ezek az előző programban a file-nyitásra és file-lezárásra használt makrókkal teljesen megegyeznek. Az AH = 3FH paraméter megadása esetén a READFIL makro azt a file-t olvassa be, amit a BX-ben elhelyezett file-kezelővel előírtunk. CX-ben, mint mindig, most is a beolvasásra kerülő byte-ok száma van. A NAMEFLD effektív címét DX-ben helyezzük el. A DS regiszter az adatszegmens címéhez mutat. A NAMEFLD változóval az adatszegmensben annyi memóriahelyet foglalunk el, hogy a CX-ben előírt byte-mennyiség elhelyezése biztosítva legyen. Amikor a 21H DOS megszakítást meghívjuk, AX-be a beolvasott byte-ok aktuális száma kerül. Ha AX zérus, akkor a file végébe ütközünk. Ennek a programnak az adatszegmense többé-kevésbé hasonlít a 8–7. ábrán látható adatszegmenshez. Ez nem meglepő, hiszen az információ beolvasásának formátuma meg kell egyezzen a beírás formátumával.

```
MYDATA  SEGMENT PARA 'DATA'
NAMEPAR LABEL   BYTE
MAXLEN  DB      81              ;AZ INFORMACIO MAX. HOSSZA+1
ACTLEN  DB      ?               ;AZ INFORMACIO AKTUALIS HOSSZA
NAMEFLD DB      80 DUP(' '),'$' ;AZ INFORMACIO, AMIT LEMEZEN
                                ;AKARUNK ELHELYEZNI
FILENAM DB      'A:NUMBERS.DAT',0 ;A FILE NEVE
FHAND   DW      ?               ;A FILE-KEZELO KOD TAROLASAT
                                ;BIZTOSITO VALTOZO
PROMPT  DB      'NEV                                    TELEFONSZAM$'
POSITION DW     0100H
MYDATA  ENDS
```

A FILENAM változóban – mint már az előző estekben is láthattuk – elhelyeztünk még egy byte-ot, ami a file nevének végét jelzi. A PROMPT teljesen megegyezik azzal, amit az előző programban használtunk.

A főeljárás aktuális kódja szinte magától értetődő:

```
        CLEARSCREEN             ;A KEPERNYO LETORLESE
        OPENFIL                 ;A FILE MEGNYITASA
AGAIN:
        CALL    PASINFO         ;LEMEZOLVASASI IGENY
        CMP     ACTLEN,00       ;HA NINCS INFORMACIO,
                                ;A PROGRAM BEFEJEZODIK,
        JNE     AGAIN           ;EGYEBKENT TOVABBI BEVITEL BEKERESE
        CLOSFIL                 ;A FILE LEZARASA
```

A MACLIB.MAC könyvtárból származó CLEARSCREEN makro letörli a képernyőt, még mielőtt az OPENFIL makróval a file-t megnyitnánk. Ha a file nyitása szerencsésen végbemegy, a PASINFO eljárás kiemeli az információt a file-ból. Amennyiben az információ hosszúsága zérus (ACTLEN = 0), a program a CLOSEFIL makro futtatásával befejeződik. A PASINFO eljárást külön is kiemeljük.

```
PASINFO PROC    NEAR            ;AZ INFORMACIO BEOLVASASA A
                                ;LEMEZFILE-ROL
        CURSOR  0000H           ;A KURZOR BEALLITASA A LAP TETEJERE
        PRINTCHAR PROMPT        ;A BEVITEL BEKERESE
REPTN:  CURSOR  POSITION        ;KURZORPOZICIONALAS AZ IRASHOZ
        READFIL                 ;A FILE OLVASASA
        MOV     ACTLEN,AL
        CMP     ACTLEN,00       ;A BEVITEL ZERUS?
        JE      ENDO            ;HA IGEN, A PROGRAM BEFEJEZODIK
        PRINTCHAR NAMEPAR+2
        ADD     POSITION,0100H
        JMP     REPTN
ENDO:
```

A PASINFO a MACLIB.MAC könyvtárból több makrót is hasznosít. A CURSOR makro a kurzort a képernyő bal felső sarkába helyezi el. A PRINTCHAR kinyomtatja a PROMPT-ban található üzenetet a képernyőre. Ezt követően a kurzor pozícióját a POSITION változóban tárolt értékkel aktualizáljuk. A POSITION értéke a ciklus ismétlésekor megváltozik, annak érdekében, hogy az információ mindig új sorba kerüljön. A READFIL – amit már korábban is alkalmaztunk – megkeresi a kívánt információt és a NAMEPAR adatszerkezetbe helyezi el. A ciklus folytatása előtt ellenőrizzük, hogy a file végét elértük-e, és a PRINTCHAR makro a NAMEPAR adatszerkezet tartalmát kinyomtatja. A két byte hozzáadására azért van szükség, mert az adatszerkezet első byte-ja a karakterlánc maximális hosszúságát, míg a második byte-ja az aktuális hosszúságát tartalmazza. A POSITION változó értékének növelésével érjük el azt, hogy a kurzor a következő sorba kerüljön.

A telefonlista kinyomtatása addig tart, amíg a képernyő utolsó sora is betelik. Ettől kezdve a nevek és telefonszámok nem lesznek már láthatók. Bizonyos programváltoztatások szükségesek ahhoz, hogy az adatokat tetszőleges mennyiségben megjeleníthessük. Ez tehát egy megoldásra váró feladat.

A file-kezeléssel kapcsolatos műveletek egyetlen menüvezérelt programmal is megolddhatók, azonban ez már az Olvasó feladata. Javasoljuk, hogy a menü a következő opciókat tartalmazza:

1. File létrehozása
2. Írás a file-ba
3. Olvasás a file-ból
4. A file lezárása

Ha a felhasználó az 1., 2. vagy a 3. opciót választja, akkor be kell kérni a lemezfile teljes nevét (meghajtó + file-név + kiterjesztés).Ezen a módon tehát mindhárom funkciót (létrehozás, olvasás, írás) használhatjuk anélkül, hogy ehhez külön programokat kellene betöltenünk és futtatnunk.

9. PROGRAMOZÁS A 80287/80387-ES TÁRSPROCESSZORRAL

Az Intel 8087/80387-es processzorai a megfelelő 8088/80386-os mikroprocesszorral együtt dolgoznak. A 8088/80386-os processzort általános célú processzornak, míg a 8087/80387-est aritmetikai processzornak szokták nevezni. Mivel a 8087/80387-est mindig a megfelelő 8088/80386-os processzorral használjuk, társprocesszornak is nevezzük. Valamennyi eddig elkészített programunkban egész-szám aritmetikát használtunk, mivel a 80286/80386-os processzorok bizonyos aritmetikai műveleteket a valós számokra vonatkozó speciális szoftver konverzió segítsége nélkül is el tudnak végezni. A szoftverkonverziós rutinoknak – amelyeket az olyan magas szintű nyelvek használnak, mint a BASIC, a FORTRAN vagy a PASCAL – az a céljuk, hogy a valós számokat olyan számsorozattá alakítsák, ami a 80286/80386-os számára egésznek látszik. Az ily módon átalakított számmal elvégezhetők a kívánt aritmetikai műveletek, majd a szám egy másik szoftver rutinnal valós számmá alakítható vissza. Ez az oda-vissza alakítás annyira időigényes volt, hogy a korábbi mikroszámítógépek pl. kétféle BASIC-kel rendelkeztek. Az egyik változat csak egész, a másik egész és valós aritmetikával dolgozott. (Az egész számokkal dolgozó BASIC változat volt a gyorsabb, és a számítógépes játékokhoz használták.)

Szükségessé vált tehát egy olyan mikroprocesszor készítése, ami valós számok gyors és pontos aritmetikáját tudja megvalósítani. Körülbelül ugyanebben az időben az IEEE (Institute of Electrical and Electronic Engineers) megállapította a valós számok kifejezésének szabványát. Az Intel cég átvette ezt a szabványt, és a kis méretű számítógépeknél általánosan elterjesztette. Jelenleg kétféle valós szám formátum használatos.

1. Microsoft valós-szám formátum, amit a legtöbb magas szintű nyelv használ, és ami nem támogatja a társprocesszor alkalmazását.
2. Az Intel, vagy IEEE formátum, amit a 80287/80387-es társprocesszor használ.

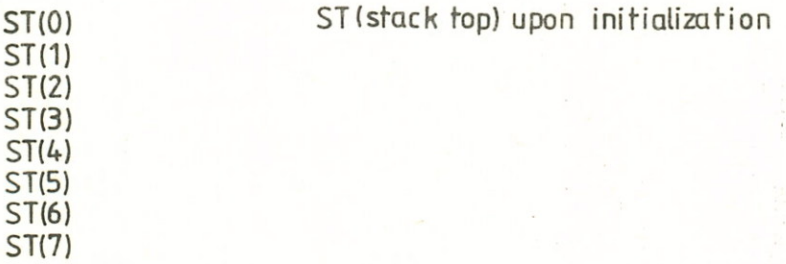
Az assembly nyelvben a .8087-es direktíva felhasználásával a programot arra késztethetjük, hogy a valós számokat IEEE formátumban tárolja. Ha ezt a direktivát nem adjuk meg, a valós számok Microsoft formátumban kerülnek tárolásra, amit a társprocesszorok nem tudnak használni. A számok valós formátumának kialakítása a fordítási idő alatt történik. Azonban nincs olyan hasonló fordított eljárás, ami a valós formátumot visszaalakítaná, ezért a társprocesszorokból származó eredmények kódolt valós formátumban szerepelnek, ami a felhasználó számára elég nehezen érthető. A megfelelő formátum kialakítását tehát egy külön szoftver rutinnak kell megoldani.

9.1 MIKROPROCESSZOR SPECIFIKÁCIÓK

A 8087-es processzor 1979-ben jelent meg a piacon. 1982-ben az Intel a 80287-es, három évvel később pedig a 80387-es processzorral jelentkezett. Mind a 80287-es, mind a 80387-es processzor 80 bites belső szerkezetű, IEEE lebegőpontos formátumot használ és hét adattípust támogat (32 bites egyszeres pontosságú valós, 64 bites dupla pontosságú valós, 80 bites kibővített valós, 16 bites egész szó, 32 bites rövid egész, 64 bites hosszú egész, 18 számjegyes BCD egész), valamint a 80286/80386-es utasításkészletét kibővíti valamennyi adattípus trigonometrikus, logaritmikus, exponenciális és aritmetikai utasításaival. Ezen kívül a 80387-es társprocesszor támogatja a 80386-os adatbusszal való teljes 32 bites csatlakozást, és rendelkezik a szinusz, koszinusz és tangens trigonometrikus függvényekkel. (A 80287-es csak a tangensfüggvényt támogatja a 0-tól 45 fokig terjedő értelmezési tartományban.) A 80287-es chipet egy szabványos 40 lábú kerámia DIP tokban helyezték el, míg a 80387-est egy 68 lábú kerámia GRID tokban (hasonlóan a 80286/80386-oshoz).

A 80287-es órajel frekvenciája 5, 8, 10 vagy 12 MHz lehet, a 80387-esé pedig 12, vagy 16 MHz. Bár a társprocesszorok mind a hét adattípust támogatják, az adatokat kibővített valós formátumban tárolják. Azok az utasítások, amelyek az adatokat a veremre ráhelyezik, automatikusan végrehajtják a hét támogatott adattípusból a kibővített valós formátumba történő átalakítást. A 80287/80387-es processzorral elvégezhető műveleteket a 9–1. táblázatban foglaltuk össze.

Már az előző fejezetben említettük, hogy a társprocesszorok veremorientáltak. A vermet a 9–1. ábrán látható formában képzelhetjük el.



9–1. ábra A 80287/80387-es verem

Művelet	Jelentés	
Egész összeadás	Négyzetgyökvonás	
Egész kivonás	Osztás/szorzás kettő hatványozott értékkel	
Egész szorzás	Parciális maradékképzés	
Egész osztás	Kerekítés egész számra	
Egész összehasonlítás	A hatvány és a mantissza kiemelése	
Valós összeadás	Abszolút érték	
Valós kivonás	Az előjel megváltoztatása	
Valós szorzás	Vizsgálat zérusra	
Valós osztás	A verem tetejének vizsgálata	
Valós összehasonlítás		
Zérus betöltése		
1 betöltése		
PI betöltése		
A $\log_2 10$ betöltése		
A $\log_2 e$ betöltése		
A $\log_{10} 2$ betöltése		
A ln 2 betöltése		
	80287-es:	**80387-es:**
Tangens	0-tól PI/4 radiánig	Forgásszögekre
Árkusz tangens	Forgásszögekre	Forgásszögekre
Szinusz	Nincs támogatva	Forgásszögekre
Koszinusz	Nincs támogatva	Forgásszögekre
Egyidejű szinusz, koszinusz	Nincs támogatva	Forgásszögekre
$2^x - 1$	$0 <= x <= 0.5$	$0 <= x <= 0.5$
$y(\log_2 x)$	Minden pozitív x értékre	Minden pozitív x értékre
$y(\log_2 (x+1))$	Minden pozitív x értékre	Minden pozitív x értékre

9–1. táblázat A 80287/80387 processzorral elvégezhető műveletek

9.2 EGÉSZ SZÁM ARITMETIKA ÉS AZ INTEL TÁRSPROCESSZORAI

Mind a 80286/80386-os, mind a 80287/80387-es képes egész szám aritmetika végrehajtására. Felmerülhet a kérdés, miért van szükség erre a duplázásra. Talán a társprocesszor mégsem kezeli megfelelően a valós számokat? Három oka van annak, hogy az egész aritmetika a társprocesszornál is szerepel:

1. Gyakran előfordul, hogy valós számokat kell egészekkel összeszoroznunk.
2. A nagy pontosságot igénylő számítások gyorsabban és jobban elvégezhetők egész számokon végrehajtott műveletekkel.
3. Lehetővé válik a valós aritmetika végrehajtása és a kerekített egész eredmény képzése és tárolása (ez a módszer számos grafikus feladatnál segítséget jelent).

Először két olyan példát mutatunk be, amelyekben a társprocesszorral egész szám aritmetikát valósítunk meg.

Két egész szám összeadása

Különösebb bevezető nélül bemutatunk egy programot, ami nagyon hasonlít az előző fejezetekben ismertetett programokhoz.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;EGESZ SZAMOK OSSZEADASA A 80287/80387-ES TARSPROCESSZORRAL
;AZ EREDMENY A MYOMKOVETO PROGRAMMAL FIGYELHETO MEG

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM1    DQ      123456789ABCDEF0H
NUM2    DD      12345678H
ANS     DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FILD    NUM1            ;NUM1 RAHELYEZESE A TARSPROCESSZOR
                                ;VERMERE
        FIADD   NUM2            ;NUM2 HOZZAADASA NUM1-HEZ
        FISTP   ANS             ;LEEMELES A VEREMROL ES AZ ERTEK
                                ;ELMENTESE
        FWAIT                   ;SZINKRONIZALAS

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A programban szereplő direktívának (.8087) kettős célja van:

1. Lehetővé teszi a társprocesszor utasításainak lefordítását.
2. A direktíva után szereplő valamennyi valós számot a Microsoft formátum helyett IEEE formátumba alakítja át a rendszer.

A verem- és adatszegmensben mindössze annyi az eltérés a korábbi programokhoz viszonyítva, hogy NUM1 és NUM2 változókban igen nagy egész számokat helyeztünk el. A kódszegmensben a következő öt sor jelent újdonságot:

```
FINIT                ;A TARSPROCESSZOR INICIALIZALASA
FILD    NUM1         ;NUM1 RAHELYEZESE A TARSPROCESSZOR
                     ;VERMERE
FIADD   NUM2         ;NUM2 HOZZAADASA NUM1-HEZ
FISTP   ANS          ;LEEMELES A VEREMROL ES AZ ERTEK
                     ;ELMENTESE
FWAIT                ;SZINKRONIZALAS
```

Vegyük észre, hogy a kódrészlet valamennyi mnemonikja az F betűvel kezdődik. (A társprocesszor összes mnemonikja az F betűvel kezdődik.) Amikor a 80286/80386-os processzor ezeknek a kódoknak a végrehajtásához ér, a vezérlést átadja a társprocesszornak. Amíg a társprocesszor az adott utasításokat végrehajtja, a 80286/80386-os a következő utasításra tér át. A társprocesszor tehát valóban kiegészítő tevékenységet folytat.

Az FINIT úgy viselkedik, mint egy hardver alaphelyzetbe állítás. Mi is történikilyenkor? A verem valamennyi eleme, valamennyi foglaltsági megszakítás és kizárási jelző törlődik. Az FILD az egész szám veremre való ráhelyezésének mnemonikja. A NUM1 forrás memoriaoperandust az előírt memóriahelyről veszi elő a rendszer, és mialatt a számot a társprocesszor vermének tetejére (ST0) helyezi, bináris egész formátumból ideiglenes valós (80 bites) formátumba alakítja. A forrásoperandus típusának a három egész típus valamelyikének kell lennie. [Rövid (32 bites, mint a DD), szó (16 bites, mint a DW) vagy hosszú (64 bites, mint a DQ)]. A FIADD utasítás a forrásoperandust (NUM2) a verem tetején álló elemhez (ST) adja, és az eredményt ugyancsak ST-ben helyezi el. Az FIADD két egész típust támogat, a DW-t és a DD-t. Még egyszer felhívjuk az Olvasó figyelmét arra, hogy amikor egy szám rákerül a társprocesszor vermére, akkor a 80 bites ideiglenes valós formátumot veszi fel. Az FISTP a verem legfelső elemének leemelését és egy előírt céloperandusba (ANS) való elmentését szolgálja, amelynek során a társprocesszor az ideiglenes valós számot egésszé alakítja a kontrollszó RC mezőjétől függően (a kontrollszót l. később). Végül az FWAIT utasítás elvégzi a társprocesszor és a 80286/80386-os processzor szinkronizációját. Ezt az utasítást általában a 80286/80386-os assembler mnemonikokba való visszatérés előtt használjuk. A program futtatása után az adatszegmens tárkiíratása a következő listát adja:

```
85B0 : 0100   1E 2B C0 50 B8 C2 85 8E-D8 9B DF 2E 00 00 9B DA  . . + @ P8B . . X _ ..... Z
85B0 : 0110   06 08 00 9B DF 3E 0C 00-0B CB 00 00 00 00 00 00  . . . . _ > . . . K ........
85B0 : 0120   F0 DE BC 9A 78 566 34 12-78 56 34 12 68 35 F1 AC  p ∧ < . xV4 . xV4 . h5q .
85B0 : 0130   78 56 34 12 00 00 00 00-00 00 00 00 00 00 00 00  xV4 ................
85B0 : 0140   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
85B0 : 0150   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
85B0 : 0160   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
85B0 : 0170   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
```

A tárkiíratás egy részét külön is kiemeljük:

```
1E 2B C0 50 B8 C2 85 8E-D8 9B DF 2E 00 00 9B DA
06 08 00 9B DF 3E 0C 00-9B CB 00 00 00 00 00 00
F0 DE BC 9A 78 56 34 12-78 56 34 12 68 35 F1 AC
78 56 34 12 00 00 00 00-00 00 00 00 00 00 00 00
```

Az F0 DE BC 9A 78 56 34 12 jelenti NUM1 értékét byte-okban kifejezve. A felső byte az alacsonyabb memóriahelyre, míg az alsó byte a magasabb memóriahelyre kerül. A 78 56 34 12 byte-ok NUM2-t jelentik, a 68 35 F1 AC 78 56 34 12 pedig a 12345678ACF13568 hexadecimális összeget.

Gondoljuk csak el, hogy ha ezt az összegzést a 80286/80386-os processzorral végeztetnénk el, mennyivel több kódsorra lenne szükségünk. A társprocesszorokat veremorientált mikroprocesszoroknak is nevezik, mivel valamennyi művelet a processzor belső hardver vermét valamilyen módon felhasználja. Ezzel ellentétben a 80286/80386-os memória- vagy regiszterorientált mikroprocesszor, mivel valamennyi műveletet regiszterekkel hajt végre, és az információk a memóriából származnak, ill. a memóriába kerülnek vissza.

Táblázatban elhelyezett egészek összeadása

A következő példában olyan egész számok összegét képezzük, amelyeket az adatszegmensben létesített táblázatban helyeztünk el. Az adatszegmens erre vonatkozó részét emeljük ki először:

```
NUMS    DD      11111111H,22222222H,33333333H,44444444H,55555555H
        DD      66666666H,77777777H,88888888H,99999999H,0AAAAAAAAH
        DD      0BBBBBBBBH,0CCCCCCCCH,0DDDDDDDDH,0EEEEEEEEH
        DD      0FFFFFFFFH,12345678H,9ABCDEF0H
```

Valamennyi számot a NUMS változónévhez rendeltünk és duplaszónak (DD) definiáltuk (32 bit). (Emlékeztetőül: az FIADD utasítás által közvetlenül kezelhető rövid egész 32 bites.) A program, amit most bemutatunk, az összeadást egy ciklusban oldja meg.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;TABLAZATBAN ELHELYEZETT NAGYMERETU EGESZ SZAMOK OSSZEADASA
;A MATEMATIKAI TARSPROCESSZOR FELHASZNALASAVAL
;AZ EREDMENY A MYOMKOVETO PROGRAMMAL FIGYELHEO MEG

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A SEGEDPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUMS    DD      11111111H,22222222H,33333333H,44444444H,55555555H
        DD      66666666H,77777777H,88888888H,99999999H,0AAAAAAAAH
        DD      0BBBBBBBBH,0CCCCCCCCH,0DDDDDDDDH,0EEEEEEEEH
        DD      0FFFFFFFFH,12345678H,9ABCDEF0H
ANS     DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        MOV     CX,14           ;A NUMS TABLAZATBAN ELHELYEZETT
                                ;ELSO 15 SZAM OSSZEADASA
```

```
        LEA     BX,NUMS
        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FILD    NUMS[BX]        ;AZ ELSO SZAM BEKERESE A TABLAZATBOL
AGAIN:  ADD     BX,04H          ;LEPTETES A KOVETKEZO SZAMRA
        FIADD   NUMS[BX]        ;OSSZEADAS A VEREM LEGFELSO ELEMEVEL
        LOOP    AGAIN
        FISTP   ANS             ;A VEREM TETEJEN KEPZODOTT EREDMENY
                                ;ELMENTESE
        FWAIT                   ;SZINKRONIZALAS

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A kódszegmens egy részét külön kiemeljük:

```
        MOV     CX,14           ;A NUMS TABLAZATBAN ELHELYEZETT
                                ;ELSO 15 SZAM OSSZEADASA
        LEA     BX,NUMS
        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FILD    NUMS[BX]        ;AZ ELSO SZAM BEKERESE A TABLAZATBOL
AGAIN:  ADD     BX,04H          ;LEPTETES A KOVETKEZO SZAMRA
        FIADD   NUMS[BX]        ;OSSZEADAS A VEREM LEGFELSO ELEMEVEL
        LOOP    AGAIN
        FISTP   ANS             ;A VEREM TETEJEN KEPZODOTT EREDMENY
                                ;ELMENTESE
        FWAIT                   ;SZINKRONIZALAS
```

Ebben a példában a táblázat első 15 mennyiségét adjuk össze, amelyhez egy ciklust használunk. A CX regiszter tölti be a ciklusszámláló szerepét.

CX-be 1-gyel kisebb számot állítunk be, mint az összeadások száma, mivel a táblázatból az első számot még a ciklusba való belépés előtt elhelyeztük a veremben. A BX regiszter tartalmazza a táblázaton belüli eltolási értéket, ezért az első elem címét a LEA (effektív cím betöltése) utasítással töltjük be a BX regiszterbe. Az FINIT inicializálja a segédprocesszort és az FILD rátölti az első egész számot (ideiglenes valós formátumban) a segédprocesszor vermére (ST-be). Amint a program a ciklusba belép, a BX regiszter tartalmához 4-et adunk, annak érdekében, hogy a következő számra rá tudjunk mutatni. A FIADD NUMS[BX] utasítás ezt a számot ST-re helyezi, kicserélve a verem legfelső elemének tartalmát az új részösszeggel. Ezt az eljárást mindaddig ismételjük, amíg mind a 15 számot összeadjuk.

A műveletek során az összeg mindvégig a társprocesszoron belül marad. Amikor valamennyi összeadást elvégeztük, az eredményt a veremből az ANS változóba tesszük. Egy hasznos jótanács: hagyjuk a részösszegeket mindaddig a társprocesszor vermében, amíg csak lehetséges, ily módon ui. a kerekítési hibák minimálissá válnak. A program az FWAIT szinkronizálási utasítással fejeződik be.

Futtatás után az adatszegmenst a nyomkövető programmal vizsgálhatjuk meg.

```
7EAC : 0100   1E 2B C0 50 B8 BF 7E 8E-D8 B9 0E 00 8D 1E 00 00   . + @P8??~ . X9 .....
7EAC : 0110   9B DB 87 00 00 83 C3 04-9B DA 87 00 00 E2 F6 9B   ..[..... C .... Z .... bv .
7EAC : 0120   DF 3E 44 00 9B CB 00 00-00 00 00 00 00 00 00 00   _> D ..... K ..........
7EAC : 0130   11 11 11 11 22 22 22 22-33 33 33 33 44 44 44 44   ..........  """"3333DDDD
7EAC : 0140   55 55 55 55 66 66 66 66-77 77 77 77 88 88 88 88   UUUUffffwwww ..........
7EAC : 0150   99 99 99 99 AA AA AA AA-BB BB BB BB CC CC CC CC   .... * * * * ; ; ; ; LLLL
7EAC : 0160   DD DD DD DD EE EE EE EE-FF FF FF FF 78 56 34 12   ]]]] nnnn ..... xV4 .....
7EAC : 0170   F0 DE BC 9A F8 FF FF FF-FF FF FF FF 00 00 00 00   p ∧ < . x ..............
```

A eredményt közvetlenül a NUMS változó utolsó tétele után kell keresnünk, ami a tárkiíratásban F0 DE BC 9A-ként szerepel. Ez azt jelenti, hogy az eredménynek a F8 FF FF FF FF FF FF FF számnak kell lennie. A byte-ok átszervezése után a következő hexadecimális eredményt kapjuk: FFFFFFFFFFFFFFF8. Ha az összeadást manuálisan végezzük el, akkor a 7FFFFFF8 eredményt kapjuk. Mit csináltunk rosszul?

Valójában semmit. A társprocesszor az egész számokon előjeles aritmetikát hajt végre. Rövid egészek esetén (32 bit) minden szám, ami 7FFFFFFF-nél kisebb, vagy azzal egyenlő pozitívnak, míg minden szám, ami 8000000-val egyenlő vagy ennél nagyobb, negatívnak tekintendő. Ugyanez az elv más egész méretekre is kiterjeszthető. Az előbbi összeadásnál az utolsó nyolc szám mint negatív szám került az összeghez. Az eredmény, amit kaptunk a számok algebrai összege volt. (Megjegyzés: Ha az összegzést kézzel végezzük el, a 7FFFFFFF feletti részösszeget is negatívnak kell tekinteni.)

9.3 EGÉSZ ÉRTÉKEK KIJELZÉSE MAKRO SEGÍTSÉGÉVEL

Az előző fejezetben a makrókat a képernyő letörléséhez, a kurzor pozícionálásához és szövegek megjelenítéséhez készítettünk. Az a makro, amit most bemutatunk, egy négyesszót ír ki a képernyőre attól a kurzorpozíciól kezdve, amire a DX regiszterrel rámutatunk.

```
COMMENT /Ez a makro a DX regiszterrel beallitott kurzorpoziciotol
        kezdve egy negyes szo erteket nyomtatja ki a kepernyore.
        A kurzorpzicio es a BX mutato ertekenek novelesevel
        kozvetlenul a kepernyon jelenithetok meg az ertekek.
        Ez a makro szuksegtelenne teszi a nyomkoveto program
        hasznalatat a programok tesztelesekor.

        FORMATUM:       LEA     BX,VARIABLE     ;A VALTOZO HELYE
                        MOV     0100H           ;A KURZOR POZICIOJA
                        QUADIS                  ;A MAKRO HIVASA

QUADIS  MACRO                           ;;A FORUTIN
        LOCAL   MORE,FINISH
        PUSH    DX
        PUSH    CX
        PUSH    BX
        PUSH    AX
        MOV     SI,07H                  ;;AZ INDEXREGEISZTER
                                        ;;SZAMLALOJANAK BEALLITASA
MORE:   MOV     AL,BYTE PTR [BX+SI]     ;;A NEGYES SZO EGY BYTE-JANAK
                                        ;;KIEMELESE
        MOV     CL,04H                  ;;CL BEALLITASA A ROTALASHOZ
        ROR     AL,CL                   ;;A BYTE ROTALASA 4 HELLYEL
        AND     AL,0FH                  ;;AZ ALSO 4 BITET TARTJUK MEG
        SETPXY                          ;;A KURZOR MOZGATASA
        DISPXY                          ;;A 4 BIT KIJELZESE
        MOV     AL,BYTE PTR [BX+SI]     ;;VEGYUK MEGEGYSZER UGYANAZT
                                        ;;A BYTE-OT
        AND     AL,0FH                  ;;MASZKOLAS
```

```
        INC     DX                      ;;A KURZORPOZICIO ERTEKENEK
                                        ;;CSOKKENTESE
        SETPXY                          ;;A KURZOR MOZGATASA
        DISPXY                          ;;A 4 BIT KIJELZESE
        INC     DX                      ;;A KURZORPOZICIO ERTEKENEK
                                        ;;CSOKKENTESE
        DEC     SI                      ;;A KOV. BYTE BEKERESENEK
                                        ;;ELOKESZITESE
        CMP     SI,00H                  ;;VEGEZTUNK A 8 BYTE-TAL?
        JL      FINISH                  ;;HA IGEN, UGRAS FINISH-RE
        JMP     MORE                    ;;NA NEM, UGRAS MORE-RA
FINISH: POP     AX
        POP     BX
        POP     CX
        POP     DX
        ENDM                            ;;A FOMAKRO VEGE

SETPXY  MACRO                           ;;A KURZORBEALLITAST SZOLGALO
                                        ;;MAKRO
        PUSH    BX                      ;;CSAK KIJELZESROL VAN SZO
        PUSH    AX                      ;;A POZICIO BX-BEN
        MOV     BX,01H                  ;;INPUT PARAMETEREK
        MOV     AH,02H
        INT     10H
        POP     AX
        POP     BX
        ENDM

DISPXY  MACRO                           ;;MAKRO A SZAMOK ES HEXA-
        LOCAL   ADJUST                  ;;BETUK KIJELZESEHEZ
        PUSH    CX                      ;;AZ ERTEKET AZ AL REGISZTERBEN
        PUSH    BX                      ;;KELL ATADNI
        PUSH    AX
        MOV     AH,10                   ;;A KARAKTEREK MEGJELENITESEHEZ
        MOV     BX,01H                  ;;SZUKSEGES PARAMETEREK
        MOV     CX,01H
        AND     AL,0FH
        CMP     AL,09H
        JLE     ADJUST                  ;;A SZAM NEM AZ A,B,C,D,E,F
                                        ;;MENNYISEGEK VALAMELYIKE
        ADD     AL,07H                  ;;A SZAM AZ A,B,C,D,E,F
                                        ;;MENNYISEGEK VALAMELYIKE
ADJUST: ADD     AL,30H                  ;;ATALAKITAS ASCII-BE
        INT     10H                     ;;A MEGSZAKITAS HIVASA
        POP     AX
        POP     BX
        POP     CX
        ENDM
```

Ezt a makrót az éppen érvényes lemezre kell tárolni. Nem szabad sem lefordítani, sem szerkeszteni, ezek a műveletek ui. akkor történnek majd meg, amikor a felhasználói program fordítását és szerkesztését végezzük. A QUADIS makro használatának szintaxisa egyszerű. Annak a négyesszónak a helyét, amit meg akarunk jeleníteni, a LEA utasítással a BX regiszterbe kell töltenünk, és a kívánt kurzorpozíciót a DX regiszterbe beállítanunk. DH-ban szerepel a függőleges, DL-ben pedig a vízszintes pozíció. Példaként tegyük fel, hogy a program adatszegmense egy négyesszót tartalmaz, amihez a VARIABLE elnevezést rendeltük hozzá.

A VARIABLE tartalmának megjelenítéséhez a következő kód szükséges:

```
LEA     BX,VARIABLE     ;A VALTOZO HELYE
MOV     0100H           ;A KURZOR POZICIOJA
QUADIS                  ;A MAKRO HIVASA
```

9.4 NAGYMÉRETŰ POZITÍV EGÉSZ SZÁMOK ÖSSZESZORZÁSA

A bemutatásra kerülő program adatszegmense a következő három pozitív egész számot tartalmazza: 1234H, 5678H, 12345678H. Ezeket a számokat fogjuk összeszorozni. Az eredményt az ANS (DQ) változóban tároljuk, amit a QUADIS makro segítségével jelenítünk meg a képernyőn.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;NEHANY NAGYMERETU EGESZ SZAM OSSZESZORZASA
;A MATEMATIKAI TARSPROCESSZOR FELHASZNALASAVAL
;AZ EREDMENYT A QUADIS.MAC MAKRO SEGITSEGEVEL JELENITJUK MEG

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET
IF1
        INCLUDE C:MACLIB.MAC    ;A MACLIB.MAC KONYVTAR CSATOLASA ES
        INCLUDE C:QUADIS.MAC    ;A QUADIS.MAC KONYVTAR CSATOLASA A
                                ;PROGRAMHOZ
ENDIF

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUMS    DD      1234H,5678H,12345678H
ANS     DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        MOV     CX,2            ;AZ ELSO HAROM SZAM OSSZESZORZASA
        LEA     BX,NUMS         ;A NUMS VALTOZO CIMENEK BEKERESE
        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FILD    NUMS[BX]        ;AZ ELSO SZAM BEKERESE NUMS-BOL
```

```
MORE:   ADD     BX,04H          ;A KOVETKEZO SZAM BEKERESE NUMS-BOL
        FIMUL   NUMS[BX]        ;A SZORZAS EREDMENYE A VEREM LEGFELSO
                                ;ELEMEBEN TALALHATO
        LOOP    MORE
        FISTP   ANS             ;AZ VEREMROL SZARMAZO ERTEK ELMENTESE
                                ;EGY VALTOZOBA
        FWAIT                   ;SZINKRONIZALAS

        CLEARSCREEN             ;A KEPERNYO LETORLESE A KIJELZESHEZ
        MOV     DX,0100H        ;AZ EREDMENY KIJELZESE A MASODIK SORBAN
        LEA     BX,ANS          ;ANS HELYE BX-BE KERUL
        QUADIS

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A két külső makrokönyvtár azonosításáról az IF1 pszeudo művelettel gondoskodunk.

```
IF1
        INCLUDE C:MACLIB.MAC    ;A MACLIB.MAC KONYVTAR CSATOLASA ES
        INCLUDE C:QUADIS.MAC    ;A QUADIS.MAC KONYVTAR CSATOLASA A
                                ;PROGRAMHOZ
ENDIF
```

A MACLIB.MAC CLEARSCREEN makrója a képernyőt törli le, míg a QUADIS.MAC QUADIS makrója az ANS változó tartalmát jeleníti meg a képernyőn.

A szorzást megvalósító kód szerkezete hasonlít ahhoz, amit az előző összeadási példánál láttunk:

```
        MOV     CX,2            ;AZ ELSO HAROM SZAM OSSZESZORZASA
        LEA     BX,NUMS         ;A NUMS VALTOZO CIMENEK BEKERESE
        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FILD    NUMS[BX]        ;AZ ELSO SZAM BEKERESE NUMS-BOL
MORE:   ADD     BX,04H          ;A KOVETKEZO SZAM BEKERESE NUMS-BOL
        FIMUL   NUMS[BX]        ;A SZORZAS EREDMENYE A VEREM LEGFELSO
                                ;ELEMEBEN TALALHATO
        LOOP    MORE
        FISTP   ANS             ;AZ VEREMROL SZARMAZO ERTEK ELMENTESE
                                ;EGY VALTOZOBA
        FWAIT                   ;SZINKRONIZALAS
```

A FIMUL utasítás esetében ugyanaz a méretkorlátozás, mint az FIADD-nál. A forrásoperandusnak 32 bites rövid egésznek, vagy 16 bites egész szónak kell lennie. Ebben a programban CX-et ismét 1-gyel kevesebbre állítottuk be, mint azoknak a mennyiségeknek a száma, amit össze akarunk szorozni, mert az első számot még a ciklusba való belépés előtt ráhelyeztük a veremre.

A kód, ami az eredményt kinyomtatja a képernyőre a következő:

```
CLEARSCREEN             ;A KEPERNYO LETORLESE A KIJELZESHEZ
MOV     DX,0100H        ;AZ EREDMENY KIJELZESE A MASODIK SORBAN
LEA     BX,ANS          ;ANS HELYE BX-BE KERUL
QUADIS
```

A futtatás során a képernyőn a következő eredmény jelenik meg:

006FEDD279706D00

ami a három szám összeszorzásának korrekt hexadecimális eredménye.

9.5 EGÉSZ SZÁMOK CSOPORTJÁNAK MEGJELENÍTÉSE A KÉPERNYŐN

A QUADIS makro ismételt hívásával lehetővé válik, hogy több számot is kijelezzünk a képernyőn. A következő példában, amivel az 1-től 20-ig terjedő egész számok négyzetgyökét számítjuk ki, ezt a programozási technikát mutatjuk be.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;A PROGRAMBAN EGYSZERU VALOS-SZAM ARITMETIKAT VALOSITUNK MEG.
;A VEGEREDMENY EGESZ ALAKBAN LESZ LATHATO. A PROGRAM KISZAMITJA
;AZ 1-20 SZAMOK NEGYZETGYOKET, ES A QUADIS MAKROVAL MEGJELENITI
;AZ EREDMENYEKET HEXADECIMALIS ALAKBAN LATJUK MAJD.

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET
IF1
        INCLUDE C:MACLIB.MAC    ;A MACLIB.MAC KONYVTAR CSATOLASA ES
        INCLUDE C:QUADIS.MAC    ;A QUADIS.MAC KONYVTAR CSATOLASA A
                                ;PROGRAMHOZ
ENDIF

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
NUM     DD      1
CONST   DQ      100000000
ANS     DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        CLEARSCREEN             ;A KEPERNYO LETORLESE
```

```
        MOV     DX,0000H        ;A KURZOR BEALLITASA AZ EREDMENY
                                ;KIJELZESEHEZ
        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
MORE:   FILD    NUM             ;A NUM VALTOZO ERTEKENEK BEKERESE
        FSQRT                   ;A SZAM NEGYZETGYOKENEK KEPZESE
        FILD    CONST           ;AZ EREDMENY SZORZASA EGY KONSTANSSAL
        FMUL                    ;ES RAHELYEZESE A VEREMRE
        FISTP                   ;AZ EREDMENY LEEMELESE A VEREMROL
                                ;ES ELMENTESE EGY VALTOZOBA
        FWAIT                   ;SZINKRONIZALAS

        LEA     BX,ANS          ;AZ ANS VALTOZO HELYE BX-BE KERUL
        QUADIS
        INC     DH              ;A KURZORT EGY SORRAL LEJJEBB HELYEZZUK
        INC     BYTE PTR NUM    ;NUM-HOZ 1-ET ADUNK
        CMP     BYTE PTR NUM,21 ;MIND A 20 SZAMMAL VEGEZTUNK MAR?
        JE      ENDO            ;HA IGEN, VEGE A PROGRAMNAK
        JMP     MORE            ;HA NEM, FOLYTATAS
ENDO:

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          :A PROGRAM VEGE
```

Az egész számok, ha a társprocesszor vermére kerülnek, 80 bites ideiglenes valós alakot vesznek fel. Valójában valamennyi szám, akár egész, akár valós, ilyen formátumban szerepel a veremben. Olyan utasítások, mint az FIADD vagy az FIMUL, csupán az egész formátumban szereplő számot alakítják át 80 bites valós formátummá. A négyzetgyökértékek törtrészeket is tartalmaznak (Pl. a 2 négyzetgyöke 1.4142...). Ha az FISTP utasítást használjuk, csak az 1-et kapja meg az ANS változó.

A pontosabb válasz kialakítása érdekében a négyzetgyökértéket 1000000000-val szorozzuk, és csak az ezután emeljük le a veremről. Ezen a módon igen pontos négyzetgyökértékhez juthatunk. A QUADIS makro megkívánja, hogy a felhasználó a tizedespontot a megfelelő helyre illessze be. Az 5. fejezetben bemutatott rutinokkal a tizedespont elhelyezése nem jelenthet különösebb nehézséget. Mivel a társprocesszor mnemonikjaiban nem szerepelhet azonnali címzésű érték, a konstansokat mindig az adatszegmensben kell elhelyezni.

```
NUM     DD      1
CONST   DQ      100000000
ANS     DQ      ?
```

Az assembly kód első része a következő:

```
        CLEARSCREEN             ;A KEPERNYO LETORLESE
        MOV     DX,0000H        ;A KURZOR BEALLITASA AZ EREDMENY
                                ;KIJELZESEHEZ
        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
MORE:   FILD    NUM             ;A NUM VALTOZO ERTEKENEK BEKERESE
        FSQRT                   ;A SZAM NEGYZETGYOKENEK KEPZESE
        FILD    CONST           ;AZ EREDMENY SZORZASA EGY KONSTANSSAL
        FMUL                    ;ES RAHELYEZESE A VEREMRE
        FISTP                   ;AZ EREDMENY LEEMELESE A VEREMROL
                                ;ES ELMENTESE EGY VALTOZOBA
        FWAIT                   ;SZINKRONIZALAS
```

A CLEARSCREEN makrót a kijelzés előkészítéseként használjuk. A DX regiszter tartalmazza azt a pozíciót, ahol az eredményeket a képernyőre kijelezzük. DH-val mutatunk rá a függőleges (0-tól 24-ig), DL-lel pedig a vízszintes pozícióra (0-tól 64-ig). Ebben a programban DL értékét zérusnak rögzítjük, így az eredmények a képernyő bal szélén egy oszlopot fognak alkotni.

A NUM változót az FILD utasítással helyezzük rá a társprocesszor vermére (ST-be). (A szám ekkor a 80 bites valós formátumot veszi fel.) A négyzetgyökképzés eredménye ugyancsak a verem tetejére kerül, felülírva az eredeti ST számot. Ezt követően a CONST értéket töltjük rá a veremre, amelynek következtében a négyzetgyökérték egy pozícióval lejjebb, ST(1)-be lép. Az FMUL utasítás az alapértelmezése szerint ST-t cél-, ST(1)-et forrásoperandusként kezeli. Ha ez az elrendezés számunkra kedvezőtlen, a nyolc verempozíció bármelyikét előírhatjuk a következő szintaxissal:

```
FMUL ST,ST(6)
```

Jelen esetben azonban az alapértelmezés a megfelelő. A következő lépésben a verem tetején képződött valós eredményt leemeljük, és az ANS változóba helyezzük el. (Az ANS változóban a szám már egészként szerepel!) A hátralevő kódrészlet a kivitel megfelelő formátumát biztosítja:

```
LEA     BX,ANS          ;AZ ANS VALTOZO HELYE BX-BE KERUL
QUADIS
INC     DH              ;A KURZORT EGY SORRAL LEJJEBB HELYEZZUK
INC     BYTE PTR NUM    ;NUM-HOZ 1-ET ADUNK
CMP     BYTE PTR NUM,21 ;MIND A 20 SZAMMAL VEGEZTUNK MAR?
JE      ENDO            ;HA IGEN, VEGE A PROGRAMNAK
JMP     MORE            ;HA NEM, FOLYTATAS
```

Az ANS változó helyét a BX regiszterben kell elhelyeznünk. A QUADIS makro az egész számot az éppen érvényes kurzorpozícióban jeleníti meg. DH-t úgy állítjuk be, hogy a következő szám a képernyő következő sorában jelenjen majd meg. A NUM-ot 1-gyel növeljük, és 21-gyel összehasonlítjuk.

A következő eredmények keletkeznek:

```
0000000005F5E100
00000000086DEB2C
000000000A52E659
000000000BEBC200
000000000D53F80E
000000000E999FEE
000000000FC5189B
0000000010DBD658
0000000011E1A300
0000000012D940B6
0000000013C4C48F
0000000014A5CCB2
00000000157DA278
00000000164D50EB
000000001715B41F
0000000017D78400
0000000018935C23
000000001949C185
0000000019FB26E6
000000001AA7F01B
```

Az eredmények hexadecimális formátumban szerepelnek.

Ha pl. az utolsó tételt (000000001AA7F01B) decimálissá alakítjuk át, a következő lesz az eredmény: 447213595. Ezt a számot 1000000000-val elosztva 20 négyzetgyökét (4.47213595) kapjuk. (A CONST értékét növelve ennél is nagyobb pontosság érhető el.)

9.6 AZ INTEL TÁRSPROCESSZORAI ÉS A VALÓS SZÁM ARITMETIKA

A fejezet eddigi példáiban a számokat egész formában írtuk be és az eredményeket egész alakban jelenítettük meg. Miért? A választ megkaphatjuk, ha megvizsgáljuk, hogyan kódolja a rendszer a valós számokat a társprocesszor számára. Nézzük át figyelmesen a következő listát:

```
MYDATA  SEGMENT PARA 'DATA'
RADIUS  DQ      8.567
AREA    DQ      ?
MYDATA  ENDS
```

Ha a .8087-es direktívát elhagyjuk, a RADIUS változó 8.567-es értéke a Microsoft formátumot veszi fel, vagyis olyan alakot, amit azok a nyelvek használnak, amelyek nem támogatják a társprocesszort. Igy viszont biztosítjuk az IEEE lebegőpontos formátumot. A 80286/80386-os processzorral a valós számokat csak egész alakban tárolhatjuk. A fordítóprogram gondoskodik az előírt formátum kialakításáról. Sajnálatosan a társprocesszorból származó eredmények dekódolását nem végzi el a rendszer. Ezt a feladatot a felhasználónak kell megoldania, amihez azonban a szoftverkészítő szakemberek kétféle segítséget is nyújtanak:

1. A Microsoft szimbolikus nyomkövető programjának egyik hasznos tulajdonsága, hogy lehetővé teszi az adatkijelzést valós formátumban. Tehát ha rendelkezünk a Microsoft Assemblerrel, akkor ezt a nehézséget könnyen áthidaljuk.
2. Az IBM az IBMUTIL.LIB könyvtárában számos konverziós programot találunk, amelyek segítségével a számok formátumának kialakítása viszonylag egyszerűen megoldható.

Ha a fenti lehetőségek egyikével sem rendelkezünk, akkor magunknak kell ezeket a konverziós rutinokat megírni.

IEEE valós szám formátumok

Számos módszer létezik, amivel a valós számok hexadecimális számjegyek felhasználásával kódolhatók. Szerencsére az IEEE egy olyan lebegőpontos szabványt állított elő, amelyhez az Intel mérnökei a 80287/80387-es társprocesszort gondosan hozzáalakították. Bár a társprocesszorok az adatokat ideiglenes valós formátumban (80 bit) tárolják, minket most a 32 bites rövid valós és a 64 bites hosszú valós formátumok érdekelnek. Az adatszegmensben a rövid valós számokat duplaszónak (DD), a hosszú valós számokat négyesszónak (DQ) definiáljuk.A rövid valós szám mint 32 bites kódolás nélküli szám szerepel a memóriában.

Az MSB az előjelet, a következő 8 bit a kitevőt, a többi 23 bit a mantisszát jelenti. A hosszú valós szám mint 64 bites kódolatlan szám szerepel a memóriában. Az MSB-t mint előjelet, a következő 11 bitet, mint kitevőt és a maradék 52 bitet, mint mantisszát kell értelmeznünk. A konverzió általános egyenlete meglehetősen egyszerű:

$$\text{valós szám} = (-1)^{\text{előjel}} * (\text{mantissza}) * 2^{\text{kitevő}}$$

Vegyünk egy példát. Tegyük fel, hogy a program futtatása után az adatszegmens tárkiíratása a következő adatokat tartalmazza:

82 ED FC 79 51 D2 6C 40

Ha a biteket helyes sorrenbe állítjuk, a következő kódolt hexadecimális valós szám keletkezik:

406CD25179FCED82

Alakítsuk át ezt a számot binárissá:

0100000001101100110100100101000101111001111111001110110110000010

Válasszuk szét az előjel, a kitevő és a mantissza részeket:

0 10000000110 1100110100100101000101111001111111001110110110000010

Ha az általános egyenletet alkalmazzuk, a következő eredményt kapjuk: Az MSB = 0, tehát ha -1-et a 0-dik hatványra emeljük, akkor 1-et kapunk, tehát az előjel pozitív. A kitevő (10000000110) decimális alakja 1030. A hosszú valós számoknál a kitevőből 1023-at ki kell vonni, tehát a valóságos kitevő 7 lesz.
A mantissza az eddig fel nem használt számjegyekből képződik, amely elé egy decimális pontot és egy 1-est kell képzelnünk:

1.1100110100100101000101111001111111001110110110000010

Ahhoz, hogy ezt a bináris számot decimálissá alakíthassuk, törtszámok súlyozott konverzióját kell megvalósítanunk, amivel a zsebszámológépek rendszerint nem rendelkeznek. Mit is jelent ez? A tizedesponttól jobbra eső első szám az 1/2-et, a második az 1/4-et, a harmadik az 1/8-ot jelenti, és így tovább. Minden olyan helyen, ahol 1-es áll, az adott helyértéket (ami 1/2 megfelelő hatványa) kell az összegzésbe belevonnunk.

```
1.0
 .5
 .25
 .03125
 .015625
 .00390625
 .00048828125
 .00006103515625
1.80133156640625
```

Az előbbi összegzésnél csak az első nyolc egyest vettük figyelembe, a célunk ui. az elmélet bemutatása volt. Ha a fenti értéket a képletbe behelyesítjük, a következő eredményt kapjuk:

$$\begin{aligned} \text{valós szám} &= +1.80133156640625 * 2^7 \\ &= +1.80133156640625 * 128 \\ &= +230.5703124 \end{aligned}$$

A végeredmény 230.5724458637233.

A rövid valós számok konverziója a most bemutatotthoz hasonlóan végezhető el azzal a különbséggel, hogy a mantissza 23 bit hosszú és a kitevőből 127-et kell levonni.

Lássunk erre is egy példát! Tegyük fel, hogy egy program futtatása után a következő rövid valós szám került az adatszegmensbe:

00 00 58 BE

Ha a számokat átrendezzük, a

BE580000

értéket kapjuk. Váltsuk át ezt a számot bináris alakba:

10111110010110000000000000000000

Válasszuk külön az előjelet, a kitevőt és a mantisszát!

1 01111100 10110000000000000000000

Az előjelbit 1. Ha – az egyenlet szerint – a −1-et 1-re emeljük, negatív előjelet kapunk. A kitevőt – 01111100 – decimálissá alakítva 124-et kapunk. A valóságos kitevő 124 – 127 vagyis −3. A mantissza a következő:

1.10110000000000000000000

A szám decimális alakja pedig:

```
1.0
 .5
 .125
 .0625
------
1.6875
```

Helyettesítsünk be az egyenletbe!

$$\begin{aligned} \text{valós szám} &= -1.6875 * 2 \\ &= -1.6875 * 0.125 \\ &= -0.2109375 \end{aligned}$$

A következő példában a manuális konverzió helyett a Microsoft és az IBM assembler programcsomagjaiban található konverziós rutinokat alkalmazzuk.

Valós szám aritmetikát alkalmazó egyszerű program

Ebben a programban az A = PI * R egyenlet felhasználásával egy kör területét számítjuk ki és az eredményt a MICROSOFT szimbólikus nyomkövető programjával nézzük meg. A sugár értéke és az eredmény 64 bites valós számok.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;A PROGRAMBAN EGYSZERU VALOS-SZAM ARITMETIKAT VALOSITUNK MEG.
;AZ EREDMENYT A MICRSOFT SZIMBOLIKUS NYOMKOVETOJEVEL TESSZUK
;LATHATOVA.
;A PROGRAM EGY KOR TERULETET SZAMITJA KI.

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET


STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
RADIUS  DQ      8.567
AREA    DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FLD     RADIUS          ;A SUGAR ERTEKENEK RAHELYEZESE A VEREMRE
        FMUL    RADIUS          ;A SUGAR NEGYZETENEK KEPZESE
        FLDPI                   ;PI RAHELYEZESE A VEREMRE
        FMUL                    ;A KET ERTEK OSSZESZORZASA (PI,R*R)
        FSTP    AREA            ;AZ EREDMENY LEEMELESE A VERMEROL
                                ;ES ELMENTESE EGY VALTOZOBA
        FWAIT                   ;SZINKRONIZALAS

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Az adatszegmensben elhelyezett valós számokat DD-nek, vagy DQ-nak kell definiálnunk. A DD-t a 32 bites valós számoknál, míg DQ-t a 64 bites valós számoknál használjuk. A valós számok beírására a következő lehetőségeink vannak:

```
   8.567
  +12.345
 -234.9
  -12.4E2
   1.234E-10
```

A kódszegmens egy lényeges részét külön kiemeljük:

```
FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
FLD     RADIUS          ;A SUGAR ERTEKENEK RAHELYEZESE A VEREMRE
FMUL    RADIUS          ;A SUGAR NEGYZETENEK KEPZESE
FLDPI                   ;PI RAHELYEZESE A VEREMRE
FMUL                    ;A KET ERTEK OSSZESZORZASA (PI,R*R)
FSTP    AREA            ;AZ EREDMENY LEEMELESE A VERMEROL
                        ;ES ELMENTESE EGY VALTOZOBA
FWAIT                   ;SZINKRONIZALAS
```

Az FLD utasítással a valós számot a rendszer memóriájából a társprocesszor vermére (ST-re) töltjük. Az FMUL utasítás a RADIUS szám és a verem tetején található érték összeszorzását valósítja meg. A szorzás eredménye a verem tetején marad. Az FLDPI utasítás PI értékét rátölti a veremre, így R ST(1)-be kerül. Az FMUL ST-t és ST(1)-et összeszorozza és az eredményt ST-ben hagyja. Az utolsó utasítás – FSTP – az eredményt valós alakban az AREA változónak adja át.

Ha sikeresen futtattuk a programot, a MICROSOFT nyomkövetőjével nézzük meg az eredményt. Ugyanazon memóriahelyről két különböző tárkiíratást készítünk:

85B0:0100-tól 85B0:014F-ig a memóriatartalom hagyományos formátumban jelenik meg. A kód második részét a DL opcióval hívtuk meg:

```
-DL DS:0100 014F
```

DL a hosszú tárkiíratást írja elő, amelyben a valós számok 8 bites csoportokat alkotnak. Számunkra most csak a következő két szám érdekes:

```
85B0:0120 FC A9 F1 D2 4D 22 21 40 +0.8567E+1
85B0:0128 82 ED FC 79 51 D2 6C 40 +0.2305724458637233E+3
```

Az első szám a sugár (8.567), a második szám a kör területe (230.5724458637233).

9.7 AZ IBM MAKROASSEMBLER ADATKONVERZIÓS RUTINJAI

Az IBM Makroasembler 2.0 verzióját és az ez után készült verziókat adatkonverziós rutinokkal is ellátták. Ezek a rutinok a programozó számára lehetővé teszik

- az ASCII karakterláncok 80287/80387-es lebegőpontos formátummá alakítását,
- hosszú valós lebegőpontos alakból ASCII karakterláncok képzését,
- Microsoft formátum és a 80287/80387-es alak közötti oda-vissza alakítást.

A következő opciók állnak rendelkezésre:

$I8_INPUT	ASCII karakterláncok átalakítása 80287/80387-es alakra
$I8_OUTPUT	Hosszú valós 80287/80387-es számok átalakítása ASCII-be.
$I4_M4	Egyszeres pontosságú Microsoft szám átalakítása 80287/80387-es rövid valósba
$I8_M8	Dupla pontosságú Microsoft szám átalakítása 80287/80387-es hosszú valósba
$M4_I4	80287/80387-es rövid valós átalakítása Microsoft rövid valósba
$M8_I8	80287/80387-es hosszú valós átalakítása Microsoft duplapontos valósba
$I4_I8	80287/80387-es hosszú valós átalakítása 80287/80387-es rövid valósba
$I8_I4	80287/80387-es rövid valós átalakítása 80287/80387-es hosszú valósba

Ezek a konverziós rutinok különösen akkor hasznosak, amikor az assembler kódot egy olyan magas szintű nyelvhez csatlakoztatjuk, ami nem támogatja a társprocesszort. Valójában a konverziós rutinok könyvtára, ami valamennyi előbb felsorolt választási lehetőséget tartalmazza, az IBM szerkesztőprogram felhasználásával a hívó programhoz szerkeszthető. Ez a könyvtár az IBMUTIL elnevezést viseli.
Az itt bemutatott példában a konverziós rutinok közül csak az egyikre lesz szükségünk. Olyan módszer kell, ami a 80287/80387-es valós számokat lebegőpontos alakra hozza. A $I8_OUTPUT rutin a kódolt valós formátumot ASCII karakterlánccá alakítja, amit aztán közvetlenül megjelentíhetünk a képernyőn. A $I8_OUTPUT konverziós rutin használatakor a következő feltételeknek kell eleget tennünk:

1. A kódszegmenst a következőképpen kell deklarálni:

```
MATHCODE    SEGMENT    BYTE        PUBLIC      'CODE'
            EXTRN      $I8_OUT-
                       PUT:NEAR
(Itt következik az adatszegmens tartalma)

MATHCODE    ENDS
```

2. Az adatszegmenst a következőképpen kell deklarálni:

```
DATA        SEGMENT    WORD        PUBLIC      'DATA'
(Itt következik az adatszegmens tartalma)

DATA        ENDS
DGROUP      GROUP      DATA
```

3. Valamennyi rutinnak NEAR típusú eljárásnak kell lennie.
4. A DS és az ES regiszter tartalmának azonosnak kell lennie.
5. Minden regiszter értéke megváltozik, kivéve a szegmensregisztereket ill. SP-t.
6. Az IS:DS-sel kell rámutatni arra a helyzetcímre, ahol a hosszú valós számot a rutin hívása előtt elhelyeztük.
7. Az SI:DS mutat az LSTRING címére. Ez egy 17 byte-os memóriaterület, ami az ASCII karakterlánc hosszát az első byte-ban tárolja. A decimális pontot a bal oldalon tételezi fel.
8. AX értéke 1, ha az eredeti számot használja a rendszer, és a szám zérus, ha az határozatlan volt.
9. BL pozitív szám esetén egy nullát, negatív szám esetén egy minuszjelet tartalmaz.
10. DX a szám kitevőjét 10-es számrendszerben tárolja.

Első látásra lehet, hogy ennek az eljárásnak a használata bonyolultnak tűnik.

A következő példában megbizonyosodhatunk arról, hogy a követel- ménynek könnyű eleget tenni. A rutin visszatérési értéke nincs normál alakban, ezért ezt egy külön rutinnal állítjuk elő. Mielőtt tovább mennénk, vessünk egy pillantást a 9–2. ábrára, ahol a fordítási és szerkesztési művelet során könyvtárat is használunk.

```
B>C:MASM PROGNAME;
IBM Personal Computer MACRO Assembler      Version 2.00
(C) Copyright IBM Corp. 1981,1984
(C) Copyright Microsoft Corp 1981, 1983, 1984

48774 Bytes free§0
 Warning Severe
Errors Errors
0 0

B>C:LINK PROGNAME,PROGNAME,NUL,C:IBMUTIL;

IBM Personal Computer Linker

Version 2.30 (C) Copyright IBM Corp. 1981, 1985

B>
```

9–2. ábra Fordítás és szerkesztés könyvtárhasználattal

Az IBM Assemblert és a szerkesztőprogramot a C meghajtón, a programot pedig a B meghajtón tételezzük fel. A fordítási eljárás a már eddig megszokott módon történik egészen a szerkesztési szakaszig. Ekkor a C:IBMUTIL megadásával írhatjuk elő annak a külső könyvtárnak a nevét, amit a programunkhoz hozzá kell szerkeszteni. Ez a könyvtár a továbbiakban állandóan .EXE formában kapcsolódik a programhoz.

9.8 AZ IBM KISZOLGÁLÓPROGRAMOKAT TARTALMAZÓ KÖNYVTÁRÁNAK HASZNÁLATA

Az előző alfejezetben az IBM cég $I8_OUTPUT konverziós rutinját ismertettük, ami a duplapontos (DQ) lebegőpontos valós számot ASCII karakterláncba alakítja át. A BL regiszter tartalmazza az előjel karakterét, DX pedig a kitevőt tízes alapú számrendszerben. Ebből az információból állítjuk elő a szám normál alakját.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;A PROGRAMBAN VALOS-SZAM ARITMETIKAT VALOSITUNK MEG. AZ EREDMENYT
;AZ IBM ADATKONVERZIOS RUTINJAVAL JELENITJUK MEG.
;AZ IBMUTIL.LIB KONYVTARAT A FUTASI IDO ALATT KELL A
;PROGRAMHOZ HOZZASZERKESZTENI
;A PROGRAM EGY GOMB TERFOGATAT SZAMOLJA KI.

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET
IF1
        INCLUDE C:MACLIB.MAC    ;A MACLIB.MAC KONYVTAR CSATOLASA A
                                ;PROGRAMHOZ
ENDIF

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'     ;AZ ADATSZEGMENSNEK PUBLIC-NAK KELL
                                ;LENNIE
CONT1   DD      4.0             ;NUMERIKUS KONSTANS
CONT2   DD      3.0             ;NUMERIKUS KONSTANS
RADIUS  DD      123.45E2        ;A GOMB SUGARA
VOLUME  DQ      ?               ;VALTOZO DEKLARALASA VALOS FORMATUMBAN
PADDER  DB      20 DUP (' ')    ;HELYFOGLALAS AZ IBM RUTIN SZAMARA
ANSWER  DB      17 DUP (?),'$'  ;AZ IBM RUTINBOL VISSZAKAPOTT
                                ;KARAKTEREK TAROLASI HELYE
FIRDIG  DB      ' ','$'         ;AZ ELSO SZAMJEGY HELYENEK BIZTOSITASA
POWER   DW      ?               ;A KITEVO TAROLASI HELYE
TBUFF   DB      4 DUP(' ')      ;4 BYTE A KITEVO KARAKTEREI SZAMARA
SIGN    DB      '-$'            ;NEGTIV ELOJEL
POINT   DB      '.$'            ;DECIMALIS PONT
EXP     DB      ' E $'          ;A KITEVO SZIMBOLUMA

MYDATA  ENDS

MATHCODE  SEGMENT BYTE PUBLIC  'CODE'     ;KODSZEGMENS DEFINIALASA
        EXTRN   $I8_OUTPUT:NEAR         ;A RUTIN KULSO KONYVTARBAN VAN
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MATHCODE,DS:MYDATA,SS:STACK,ES:MYDATA
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
```

```
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     ES,AX           ;AX ERTEKENEK ELHELYEZESE ES-BEN

        LEA     SI,VOLUME       ;VOLUME-BAN A KARAKTERLANC
                                ;FORRASINDEXE TALALHATO

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FLD     RADIUS          ;A SUGAR ERTEKENEK RAHELYEZESE A
                                ;VEREMRE
        FMUL    RADIUS          ;A SZAM NEGYZETENEK KEPZESE
                                ;AZ EREDMENY ST-BEN
        FMUL    RADIUS          ;A SZAM KOBENEK KEPZESE
                                ;AZ EREDMENY ST-BEN KELETKEZIK
        FLDPI                   ;PI RAHELYEZESE A VEREMRE
        FMUL                    ;PI OSSZESZORZASA A SUGAR KOBEVEL
        FMUL    CONT1           ;SZORZAS 4.0-VAL
        FDIV    CONT2           ;OSZTAS 3.0-VAL
        FSTP    QWORD PTR [SI]  ;AZ EREDMENY LEEMELESE A
                                ;TARSPROCESSZOR VERMEROL ES
                                ;ELHELYEZESE EGY VALTOZOBAN
        FWAIT                   ;SZINKRONIZALAS
        CALL    $I8_OUTPUT      ;AZ IBM ELJARAS MEGHIVASA
        CALL    FORMAT          ;A FORMAT ELJARAS AZ EREDMENYT NORMAL
                                ;ALAKRA HOZZA

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE

COMMENT  /A kovetkezo eljaras a $I8_OUTPUT-bol szarmazo eredmenyt
normal alakra hozza. Pl.: -3.5678912345 E -123                  /

FORMAT  PROC    NEAR
        SUB     DX,01H          ;A KITEVO CSOKKENTESE 1-GYEL
        MOV     POWER,DX        ;DX AZ ANS VALTOZO KORRIGALT KITEVOJE
        CMP     BL,'-'          ;AZ EREDMENY + VAGY - ?
        JNE     NONEG
        LEA     DX,SIGN         ;HA NEGATIV, A '-' KARAKTER
                                ;MEGJELENIETESE
        MOV     AH,09
        INT     21H
NONEG:
        CLD                     ;A KAR.LANC ANS-BA HELYEZESENEK
                                ;ELOKESZITESE
        MOV     CL,17           ;A KAR.LANC 17 BYTE HOSSZU
        LEA     DI,ANSWER       ;A MOZGATAS IRANYA. SI A FORRASRA
                                ;MUTAT
REP     MOVSB                   ;ATHELYEZES
        MOV     CL,1            ;ANSWER ELSO SZAMJEGYE FIRDIG-BE KERUL
        LEA     SI,ANSWER[1]
        LEA     DI,FIRDIG
REP     MOVSB
        LEA     DX,FIRDIG       ;AZ ELSO SZAMJEGY KINYOMTATASA
        MOV     AH,09
        INT     21H
        LEA     DX,POINT        ;A DECIMALIS PONT MEGJELENITESE
        MOV     AH,09
        INT     21H
```

```
        LEA     DX,ANSWER[2]    ;AZ ANSWER VALTOZO TOVABBI RESZENEK
                                ;MEGJELENITESE
        MOV     AH,09
        INT     21H
        LEA     DX,EXP          ;AZ 'E' BETU KINYOMTATASA
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A DX-BEN LEVO SZAM ATALAKITASA
                                ;KARAKTERLANCBA
        CMP     DX,8000H        ;POZITIV VAGY NEGATIV
        JB      POSIT           ;HA POZITIV, UGRAS POSIT-RA
        LEA     DX,SIGN         ;HA NEGATIV, AKKOR '-' JELET NYOMTATUNK
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A NEGATIV HEXA SZAM KORREKCIOJA
        XOR     DX,0FFFFH
        ADD     DX,01
POSIT:  MOV     CX,0
        LEA     DI,TBUFF        ;A TBUFF 4 BYTE-OS KARAKTERES TAROLASI
POWER1: PUSH    CX              ;HELYET BIZTOSIT A DX-BEN SZEREPLO
        MOV     AX,DX           ;HEXA SZAMOK DECIMALISSA ALAKITASABAN
        MOV     DX,0            ;A KEPERNYORE NYOMTATAS ELOKESZITESE
        MOV     CX,10
        DIV     CX
        XCHG    AX,DX
        ADD     AL,30H          ;A SZAM ATALAKITASA ASCII SZAMMA
        MOV     [DI],AL         ;ES ELMENTESE TBUFF-BA
        INC     DI              ;UJ HELY KIJELOLESE TBUFF-BAN
        POP     CX
        INC     CX              ;CX-BEN A SZAMJEGYEK DARABSZAMA
                                ;SZEREPEL
        CMP     CX,0
        JNZ     POWER1
PRIT:   DEC     DI              ;AZ ERTEKEK MEGJELENITESENEK
                                ;ELOKESZITESE
        MOV     AL,[DI]         ;TBUFF-BOL EGY SZAMJEGY BEKERESE
        PUSH    DX              ;DX EREDETI TARTALMANAK ELMENTESE
        MOV     DL,AL           ;A SZAM ELHELYEZESE A NYOMTATASHOZ
        MOV     AH,2            ;A DOS MEGSZAKITASHOZ SZUKSEGES PARAM.
        INT     21H             ;A HATVANY A KEPERNYORE KERUL
        POP     DX              ;DX EREDETI TARTALMANAK VISSZAALLITASA
        LOOP    PRIT            ;A RUTINT ADDIG ISMETELJUK, AMIG
                                ;VALAMENNYI SZAM MEGJELENIK
        RET
FORMAT  ENDP
MATHCODE ENDS                   ;A MATHCODE ELNEVEZESU KODSZEGMENS VEGE

        END                     ;A PROGRAM VEGE
```

Ez a program a $V = 4/3 * PI * R^3$ egyenlet felhasználásával kiszámítja egy gömb térfogatát, és az eredményt (a megfelelő formátumban) megjeleníti a képernyőn. Az adatszegmensben definiált adatokat két csoportra oszthatjuk: Az egyikre konkrét számoláskor, a másikra az eredmény normál alakjának kialakításakor lesz szükségünk. A számolási műveletek során felhasznált adatok a következők:

```
CONT1   DD      4.0             ;NUMERIKUS KONSTANS
CONT2   DD      3.0             ;NUMERIKUS KONSTANS
RADIUS  DD      123.45E2        ;A GOMB SUGARA
VOLUME  DQ      ?               ;VALTOZO DEKLARALASA VALOS FORMATUMBAN
```

Mivel a társprocesszor operandusait a memóriából kell előhívnunk, valamennyi szükséges állandót előzetesen tárolnunk kell. A CONT1 és CONT2 tartalmazza azokat az egész számokat (3, 4), amikkel az egyenletben szorzunk és osztunk. RADIUS értéke 12345, amit – annak ellenére, hogy egész szám – vegyes alakban helyeztük el. A konverziós rutin használata miatt a VOLUME változót négyesszámnak kell definiálnunk.

Az adatszegmens további részében az ANSWER változó formátumának kialakításához szükséges értékek szerepelnek.

```
PADDER  DB      20 DUP (' ')    ;HELYFOGLALAS AZ IBM RUTIN SZAMARA
ANSWER  DB      17 DUP (?),'$'  ;AZ IBM RUTINBOL VISSZAKAPOTT
                                ;KARAKTEREK TAROLASI HELYE
FIRDIG  DB      ' ','$'         ;AZ ELSO SZAMJEGY HELYENEK BIZTOSITASA
POWER   DW      ?               ;A KITEVO TAROLASI HELYE
TBUFF   DB      4 DUP(' ')      ;4 BYTE A KITEVO KARAKTEREI SZAMARA
SIGN    DB      '-$'            ;NEGTIV ELOJEL
POINT   DB      '.$'            ;DECIMALIS PONT
EXP     DB      ' E $'          ;A KITEVO SZIMBOLUMA
```

A $I8_OUTPUT rutin meghívásakor az az érték, amire az SI regiszter rámutat (jelen esetben ez a VOLUME), karakterláncba alakul át, amit az SI-vel megjelölt helyen kapunk vissza. A néhány üres szóközt tartalmazó PADDER változó a karakterlánc és az adatszegmensben szereplő egyéb adatok elválasztására való. Az ANSWER 17 byte-os karakterlánc tárolását valósítja meg. (A $ karakter már arra utal, hogy valószínüleg a 21H megszakítással fogjuk az eredményt a képernyőn megjeleníteni.) A FIRDIT az eredmény első számjegyét tárolja. A program az eredményeket a normál alakban szerepelteti. A POWER változó tartalmazza majd az eredmény kitevő részét, ami a $I8_OUTPUT meghívása után a DX regiszterbe kerül. (A TBUFF változó a DX-ben tárolt kitevő ASCII konverziójához szükséges.) A SIGN, POINT és EXP változók a mínuszjelet, a decimális pontot és a kitevőt jelző karaktereket (E) tartalmazzák.

Ebben a programban a kódszegmens eltér az előző programok kódszegmenseitől, mivel a $I8_OUTPUT hívásához a megfelelő környezetet biztosítani kell.

```
MATHCODE  SEGMENT BYTE PUBLIC  'CODE'     ;KODSZEGMENS DEFINIALASA
          EXTRN   $I8_OUTPUT:NEAR         ;A RUTIN KULSO KONYVTARBAN VAN
MYPROC    PROC    FAR             ;AZ ELJARAS NEVE MYPROC
          ASSUME  CS:MATHCODE,DS:MYDATA,SS:STACK,ES:MYDATA
```

A MATHCODE elnevezésű kódszegmenst az IBMUTIL.LIB könyvtárral való kapcsolatteremtés miatt PUBLIC-nak kell deklarálnunk. Az $I8_OUTPUT két rutin közötti információcsere létrehozása érdekében EXTERN-ként szerepel.

A társprocesszorra vonatkozó kód talán a program legkisebb egysége:

```
FLD     RADIUS          ;A SUGAR ERTEKENEK RAHELYEZESE A
                        ;VEREMRE
FMUL    RADIUS          ;A SZAM NEGYZETENEK KEPZESE
                        ;AZ EREDMENY ST-BEN
FMUL    RADIUS          ;A SZAM KOBENEK KEPZESE
                        ;AZ EREDMENY ST-BEN KELETKEZIK
```

A sugár értékének köbe a verem tetején marad.

```
FLDPI                   ;PI RAHELYEZESE A VEREMRE
FMUL                    ;PI OSSZESZORZASA A SUGAR KOBEVEL
```

Az FLDPI utasítás PI értékét ráhelyezi a veremre, a köbre emelt mennyiség pedig ST(1)-be lép. Az FMUL utasítás összeszorozza ST(1) és ST tartalmát és az eredményt ST-ben helyezi el.

```
FMUL    CONT1           ;SZORZAS 4.0-VAL
FDIV    CONT2           ;OSZTAS 3.0-VAL
FSTP    QWORD PTR [SI]  ;AZ EREDMENY LEEMELESE A
                        ;SEGEDPROCESSZOR VERMEROL ES
                        ;ELHELYEZESE EGY VALTOZOBAN
```

Ezt követően ST tartamát 4-gyel szorozzuk és 3-mal osztjuk. Az FSTP utasítás az eredményt - valós szám formátumban – a VOLUME változóba helyezi el.

```
CALL    $I8_OUTPUT      ;AZ IBM ELJARAS MEGHIVASA
CALL    FORMAT          ;A FORMAT ELJARAS AZ EREDMENYT NORMAL
                        ;ALAKRA HOZZA
```

Az $I8_OUTPUT rutin hívásakor az SI-vel kijelölt DQ szám (jelen esetben a VOLUME) ugyanazon a memóriahelyen karakterláncba alakul át.

A FORMAT eljárás alakítja ki az IBM rutinból származó érték normál alakját, és az eredmény megjelenítéséről is gondoskodik. Az eredmény karakteres formátumban van. (Nem numerikus információ!) A normál alakot kialakító eljárás a következő lépésekből áll:

1. Negatív előjel kinyomtatása szükséges?
2. Az eredmény első számjegyének kinyomtatása.
3. A decimális pont kinyomtatása.
4. Az eredmény további számjegyeinek kinyomtatása.
5. Az "E" exponenciális szimbólum megjelenítése.
6. A kitevőre vonatkozó negatív előjel kinyomtatása szükséges?
7. A kitevő értékének kinyomtatása.

A következő kódrészlet dönti el a negatív előjel megjelenítésének szükségességét:

```
FORMAT  PROC    NEAR
        SUB     DX,01H          ;A KITEVO CSOKKENTESE 1-GYEL
        MOV     POWER,DX        ;DX AZ ANS VALTOZO KORRIGALT KITEVOJE
        CMP     BL,'-'          ;AZ EREDMENY + VAGY - ?
        JNE     NONEG
        LEA     DX,SIGN         ;HA NEGATIV, A '-' KARAKTER
                                ;MEGJELENITESE
        MOV     AH,09
        INT     21H
NONEG:
```

A SUB DX,01H és a MOV POWER,DX kódok a kitevő értékét tárolják, mivel erre a továbbiakban még szükségünk lesz. A BL regiszter tartalmát a mínusz karakterrel hasonlítjuk össze. Ha BL a minuszjelet tartalmazza, a SIGN tartalma jelenik meg a képernyőn. Egyébként nem nyomtatunk ki semmit. A VOLUME-ban szereplő 17 ASCII számjegyet a következő kód segítségével másoljuk át az ANSWER változóba:

```
        CLD                     ;A KAR.LANC ANS-BA HELYEZESENEK
                                ;ELOKESZITESE
        MOV     CL,17           ;A KAR.LANC 17 BYTE HOSSZU
        LEA     DI,ANSWER       ;A MOZGATAS IRANYA. SI A FORRASRA
                                ;MUTAT
REP     MOVSB                   ;ATHELYEZES
```

Csak emlékeztetőül: az SI (forrás index) már VOLUME-hoz mutat. Az eredmény első számjegyét ANSWER-ből FIRDIG-be másoljuk át:

```
        MOV     CL,1            ;ANSWER ELSO SZAMJEGYE FIRDIG-BE KERUL
        LEA     SI,ANSWER[1]
        LEA     DI,FIRDIG
REP     MOVSB
```

A CL regisztert 1-re állítjuk be, mivel csak egyetlen karakterről van szó. A karakter az ANSWER(1) helyen, nem pedig az ANSWER(0) helyen található. (Az ANSWER(0) a karakterlánc karaktereinek számát tartalmazza.)

Az első számjegy (FIRDIG) a decimáis pont és a további ASCII karakterek megjelenítését a következő kódrészlet biztosítja:

```
LEA     DX,FIRDIG       ;AZ ELSO SZAMJEGY KINYOMTATASA
MOV     AH,09
INT     21H
LEA     DX,POINT        ;A DECIMALIS PONT MEGJELENITESE
MOV     AH,09
INT     21H
LEA     DX,ANSWER[2]    ;AZ ANSWER VALTOZO TOVABBI RESZENEK
                        ;MEGJELENITESE
MOV     AH,09
INT     21H
```

A fenti eljárásban az 5. fejezetben bemutatott karakterkeket megjelenítő technikára ismerhetünk rá.

A következő feladat a kitevő értékének kinyomtatása. Az E szimbólummal egyszerű a dolgunk:

```
LEA     DX,EXP          ;AZ 'E' BETU KINYOMTATASA
MOV     AH,09
INT     21H
```

Mielőtt a kitevőt megjelenítenénk, megállapítjuk, hogy milyen az előjele és ASCII formátumba alakítjuk át.

```
        MOV     DX,POWER        ;A DX-BEN LEVO SZAM ATALAKITASA
                                ;KARAKTERLANCBA
        CMP     DX,8000H        ;POZITIV VAGY NEGATIV
        JB      POSIT           ;HA POZITIV, UGRAS POSIT-RA
        LEA     DX,SIGN         ;HA NEGATIV, AKKOR '-' JELET NYOMTATUNK
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A NEGATIV HEXA SZAM KORREKCIOJA
        XOR     DX,0FFFFH
        ADD     DX,01
POSIT:  MOV     CX,0
```

A POWER változó a kitevőt decimális számjegyek alakjában tartalmazza. Ha POWER értéke 8000H-nál kisebb, akkor a kitevő pozitív, tehát a POSIT cimkére kell ugranunk. Ha a POWER egyenlő, vagy nagyobb mint 8000H, akkor a kitevő negatív előjelű, tehát egy mínuszjelet kell a képernyőn megjelenítenünk. A kettes komplemenssel kifejezett negatív számból pozitívot képezünk.

```
POSIT:  MOV     CX,0
        LEA     DI,TBUFF        ;A TBUFF 4 BYTE-OS KARAKTERES TAROLASI
POWER1: PUSH    CX              ;HELYET BIZTOSIT A DX-BEN SZEREPLO
        MOV     AX,DX           ;HEXA SZAMOK DECIMALISSA ALAKITASABAN
        MOV     DX,0            ;A KEPERNYORE NYOMTATAS ELOKESZITESE
        MOV     CX,10
        DIV     CX
        XCHG    AX,DX
        ADD     AL,30H          ;A SZAM ATALAKITASA ASCII SZAMMA
        MOV     [DI],AL         ;ES ELMENTESE TBUFF-BA
        INC     DI              ;UJ HELY KIJELOLESE TBUFF-BAN
        POP     CX
        INC     CX              ;CX-BEN A SZAMJEGYEK DARABSZAMA
                                ;SZEREPEL
        CMP     CX,0
        JNZ     POWER1
PRIT:   DEC     DI              ;AZ ERTEKEK MEGJELENITESENEK
                                ;ELOKESZITESE
        MOV     AL,[DI]         ;TBUFF-BOL EGY SZAMJEGY BEKERESE
        PUSH    DX              ;DX EREDETI TARTALMANAK ELMENTESE
        MOV     DL,AL           ;A SZAM ELHELYEZESE A NYOMTATASHOZ
        MOV     AH,2            ;A DOS MEGSZAKITASHOZ SZUKSEGES PARAM.
        INT     21H             ;A HATVANY A KEPERNYORE KERUL
        POP     DX              ;DX EREDETI TARTALMANAK VISSZAALLITASA
        LOOP    PRIT            ;A RUTINT ADDIG ISMETELJUK, AMIG
                                ;VALAMENNYI SZAM MEGJELENIK
```

A POWER-ben tárolt kitevő négy számjegyből állhat. Annak érdekében, hogy a konverziós rutint megértsük, tegyük fel, hogy a kitevő értéke 34. A MOV AX,DX művelet 34-et az AX regiszterbe helyezi el. Ezután DX-be zérust teszünk. CX 10-et tartalmaz, és az osztási műveletben mint osztó szerepel. A DIV CX után AX-ben 3, DX-ben 4 szerepel majd. A XCHG utasítás ezt a sorrendet felcseréli, tehát AX-be 4, DX-be 3 kerül. (Valójában AL tartalmazza a 4-et.) Amikor AL-hoz 30H-t adunk, a korrekt ASCII formátumot kapjuk, amit TBUFF-ban mint MSD-t tárolunk. Az eljárást ezután megismételjük, és ez alkalommal a 3-at mint ASCII karaktert helyezzük el TBUFF-ban a következő helyre. Mivel DX az utolsó DIV és XCHG utasítás befejezése után 0-t tartalmaz, az eljárás befejeződik. A TBUFF-ban a számok megfordított sorrendben szerepelnek, ami a későbbiekben semmilyen problémát nem okoz majd. A következő feladatunk a kitevő karaktereinek kijelzése a képernyőre:

```
PRIT:   DEC     DI              ;AZ ERTEKEK MEGJELENITESENEK
                                ;ELOKESZITESE
        MOV     AL,[DI]         ;TBUFF-BOL EGY SZAMJEGY BEKERESE
        PUSH    DX              ;DX EREDETI TARTALMANAK ELMENTESE
        MOV     DL,AL           ;A SZAM ELHELYEZESE A NYOMTATASHOZ
        MOV     AH,2            ;A DOS MEGSZAKITASHOZ SZUKSEGES PARAM.
        INT     21H             ;A HATVANY A KEPERNYORE KERUL
        POP     DX              ;DX EREDETI TARTALMANAK VISSZAALLITASA
        LOOP    PRIT            ;A RUTINT ADDIG ISMETELJUK, AMIG
                                ;VALAMENNYI SZAM MEGJELENIK
```

Mivel a karaktereket fordított sorrendben tároltuk, csökkenő sorrendben jelenítjük meg. A kinyomtatott számjegyek számát CX-szel szabályozzuk.

A későbbi munkáink során elképzelhető, hogy gyakran szükségünk lesz valós számok normál alakjára. Ha úgy találjuk, hogy ez az eljárás kielégíti az igényeinket, és az IBM Makroassemblerét használjuk, helyezzük el ezt a programot az IBMUTIL.LIB kiszolgálóprogramokat tartalmazó könyvtárba.

9.9 ADOTT SZÖG TANGENSÉNEK MEGHATÁROZÁSA

Az Intel társprocesszora támogatja a tangensfüggvényt. A 80287-es akkor ad helyes eredményt, ha az argumentum 0–PI/4 tartományba esik. A 80387-es forgásszögek tangensét is képes kiszámítani. A trigonometrikus függvények használatát bemutató első programban egy szög tangensét határozzuk meg, és a valós eredményt a kiszolgálóprogramokat tartalmazó IBM könyvtár segítségével megjelenítjük. A szöget fokokban adjuk meg. A radiánba való átszámítást a program végzi.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;A PROGRAMBAN VALOS-SZAM ARITMETIKAT VALOSITUNK MEG. AZ EREDMENYT
;AZ IBM ADATKONVERZIOS RUTINJAVAL JELENITJUK MEG.
;AZ IBMUTIL.LIB KONYVTARAT A FUTASI IDO ALATT KELL A
;PROGRAMHOZ HOZZASZERKESZTENI. A PROGRAM EGY OLYAN SZOG TANGENSET
;SZAMITJA KI, AMI A 0-45 FOKOS TARTOMANYBA ESIK.

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET
IF1
        INCLUDE C:MACLIB.MAC    ;A MACLIB.MAC KONYVTAR CSATOLASA A
                                ;PROGRAMHOZ
ENDIF

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'     ;AZ ADATSZEGMENSNEK PUBLIC-NAK KELL
                                ;LENNIE
ANGLE   DD      25.5            ;A SZOGERTEK FOKOKBAN
CONST   DD      180.0           ;NUMERIKUS KONSTANS
TANGENT DQ      ?               ;A TANGENS VALTOZO DEKLARALASA
PADDER  DB      20 DUP (' ')    ;HELYFOGLALAS AZ IBM RUTIN SZAMARA
ANSWER  DB      17 DUP (?),'$'  ;AZ IBM RUTINBOL VISSZAKAPOTT
                                ;KARAKTEREK TAROLASI HELYE
FIRDIG  DB      ' ','$'         ;AZ ELSO SZAMJEGY HELYE
POWER   DW      ?               ;A KITEVO TAROLASI HELYE
TBUFF   DB      4 DUP(' ')      ;4 BYTE A KITEVO KARAKTEREI SZAMARA
SIGN    DB      '-$'            ;NEGTIV ELOJEL
```

```
POINT   DB      '.$'            ;DECIMALIS PONT
EXP     DB      ' E $'          ;A KITEVO SZIMBOLUMA

MYDATA  ENDS

MATHCODE  SEGMENT BYTE PUBLIC  'CODE'     ;KODSZEGMENS DEFINIALASA
        EXTRN   $I8_OUTPUT:NEAR         ;A RUTIN KULSO KONYVTARBAN VAN
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MATHCODE,DS:MYDATA,SS:STACK,ES:MYDATA
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     ES,AX           ;AX ERTEKENEK ELHELYEZESE ES-BEN

        LEA     SI,TANGENT      ;A TANGENT VALTOZO  A KAR.LANC
                                ;FORRASINDEXE
        FINIT                   :A TARSPROCESSZOR INICIALIZALASA
        FLDPI                   ;PI RAHELYEZESE A VEREMRE
        FLD     CONST           ;CONST RAHELYEZESE A VEREMRE
        FDIV                    ;CONST OSZTASA PI-VEL
        FLD     ANGLE           ;A SZOGERTEK RAHELYEZESE A VEREMRE
        FMUL                    ;A MEGELOZO EREDMENY ES SZOG
                                ;OSSZESZORZASA
        FPTAN                   ;A FENTI SZORZAT TANGENSENEK KEPZESE
        FDIV                    ;ST OSZTASA ST(1)-GYEL
        FSTP    QWORD PTR [SI]  ;AZ EREDMENY LEEMELESE A
                                ;TARSPROCESSZOR VERMEROL ES
                                ;ELHELYEZESE EGY VALTOZOBAN
        FWAIT                   ;SZINKRONIZALAS
        CALL    $I8_OUTPUT      ;AZ IBM ELJARAS MEGHIVASA
        CALL    FORMAT          ;A FORMAT ELJARAS AZ OUTPUTOT NORMAL
                                ;ALAKRA HOZ7A

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK

MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGF

COMMENT  /A kovetkezo eljaras a $I8_OUTPUT-bol szarmazo eredmenyt
normal alakra hozza. Pl.  -3.5678912345 E -123

FORMAT  PROC    NEAR
        SUB     DX,01H          ;A KITEVO CSOKKENTESE 1-GYEL
        MOV     POWER,DX        ;DX AZ ANS VALTOZO KORRIGALT KITEVOJE
        CMP     BL,'-'          ;AZ EREDMENY + VAGY - ?
        JNE     NONEG
        LEA     DX,SIGN         ;HA NEGATIV, A '-' KARAKTER
                                ;MEGJELENIETESE
        MOV     AH,09
        INT     21H
NONEG:
        CLD                     ;A KAR.LANC ANS-BA HELYEZESENEK
                                ;ELOKESZITESE
        MOV     CL,17           ;A KAR.LANC 17 BYTE HOSSZU
        LEA     DI,ANSWER       ;A MOZGATAS IRANYA. SI A FORRASRA
                                ;MUTAT
REP     MOVSB                   ;ATHELYEZES
```

```
        MOV     CL,1            ;ANSWER ELSO SZAMJEGYE FIRDIG-BE KERUL
        LEA     SI,ANSWER[1]
        LEA     DI,FIRDIG
REP     MOVSB
        LEA     DX,FIRDIG       ;AZ ELSO SZAMJEGY KINYOMTATASA
        MOV     AH,09
        INT     21H
        LEA     DX,POINT        ;A DECIMALIS PONT MEGJELENITESE
        MOV     AH,09
        INT     21H
        LEA     DX,ANSWER[2]    ;AZ ANSWER VALTOZO TOVABBI RESZENEK
                                ;MEGJELENITESE
        MOV     AH,09
        INT     21H
        LEA     DX,EXP          ;AZ 'E' BETU KINYOMTATASA
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A DX-BEN LEVO SZAM ATALAKITASA
                                ;KARAKTERLANCBA
        CMP     DX,8000H        ;POZITIV VAGY NEGATIV
        JB      POSIT           ;HA POZITIV, UGRAS POSIT-RA
        LEA     DX,SIGN         ;HA NEGATIV, AKKOR '-' JELET NYOMTATUNK
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A NEGATIV HEXA SZAM KORREKCIOJA
        XOR     DX,0FFFFH
        ADD     DX,01
POSIT:  MOV     CX,0
        LEA     DI,TBUFF        ;A TBUFF 4 BYTE-OS KARAKTERES TAROLASI
POWER1: PUSH    CX              ;HELYET BIZTOSIT A DX-BEN SZEREPLO
        MOV     AX,DX           ;HEXA SZAMOK DECIMALISSA ALAKITASABAN
        MOV     DX,0            ;A KEPERNYORE NYOMTATAS ELOKESZITESE
        MOV     CX,10
        DIV     CX
        XCHG    AX,DX
        ADD     AL,30H          ;A SZAM ATALAKITASA ASCII SZAMMA
        MOV     [DI],AL         ;ES ELMENTESE TBUFF-BA
        INC     DI              ;UJ HELY KIJELOLESE TBUFF-BAN
        POP     CX
        INC     CX              ;CX-BEN A SZAMJEGYEK DARABSZAMA
                                ;SZEREPEL
        CMP     CX,0
        JNZ     POWER1
PRIT:   DEC     DI              ;AZ ERTEKEK MEGJELENITESENEK
                                ;ELOKESZITESE
        MOV     AL,[DI]         ;TBUFF-BOL EGY SZAMJEGY BEKERESE
        PUSH    DX              ;DX EREDETI TARTALMANAK ELMENTESE
        MOV     DL,AL           ;A SZAM ELHELYEZESE A NYOMTATASHOZ
        MOV     AH,2            ;A DOS MEGSZAKITASHOZ SZUKSEGES PARAM.
        INT     21H             ;A HATVANY A KEPERNYORE KERUL
        POP     DX              ;DX EREDETI TARTALMANAK VISSZAALLITASA
        LOOP    PRIT            ;A RUTINT ADDIG ISMETELJUK, AMIG
                                ;VALAMENNYI SZAM MEGJELENIK
        RET
FORMAT  ENDP
MATHCODE ENDS                   ;A MATHCODE ELNEVEZESU KODSZEGMENS VEGE

        END                     ;A PROGRAM VEGE
```

Az adatszegmensben két valós szám szerepel. Az ANGLE változó a szögértéket tartalmazza fokokban, a CONST pedig a radiánba való átváltáshoz szükséges. A program szerkezete a megelőző programmal azonos, ezért csak azt a kódrészletet emeljük ki, amelyben a szög tangensének meghatározása történik:

```
FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
FLDPI                   ;PI RAHELYEZESE A VEREMRE
FLD     CONST           ;CONST RAHELYEZESE A VEREMRE
FDIV                    ;CONST OSZTASA PI-VEL
FLD     ANGLE           ;A SZOGERTEK RAHELYEZESE A VEREMRE
FMUL                    ;A MEGELOZO EREDMENY ES SZOG
                        ;OSSZESZORZASA
FPTAN                   ;A FENTI SZORZAT TANGENSENEK KEPZESE
FDIV                    ;ST OSZTASA ST(1)-GYEL
FSTP    QWORD PTR [SI]  ;AZ EREDMENY LEEMELESE A
                        ;TARSPROCESSZOR VERMEROL ES
                        ;ELHELYEZESE EGY VALTOZOBAN
FWAIT                   ;SZINKRONIZALAS
```

Először PI-t, aztán CONST-t töltjük rá a veremre. Az FDIV utasítás tehát PI-t 180-nal osztja, és az eredményt ST-ben hagyja. Az FLD utasítással ANGLE tartalmát ráhelyezzük a veremre, amely által az előző osztás eredménye egy hellyel lejjebb lép. Az FMUL ST-t összeszorozza ST(1)-gyel. Az eredmény ST-ben marad. Ha most kiadjuk az FPTAN utasítást, a tangensértéket mint két szám (Y és X) hányadosát kapjuk meg. Y kerül először a veremre, majd pedig X. (Ekkor Y egy hellyel lejjebb lép.) FDIV elvégzi az osztást és a hányadost ST-ben hagyja. Az FSTP utasítás a valós eredményt a TANGENT változóba teszi, az IBM kiszolgáló-rutin pedig a normál alakot adja.

9.10 ADOTT SZÖG SZINUSZÁNAK MEGHATÁROZÁSA

Ha az Olvasó a 80287-es társprocesszorral dolgozik, a szögek szinuszát és koszinuszát a tangensfüggvényből kell származtatnia a trigonometriai összefüggések felhasználásával, míg a 80387-es társprocesszor forgásszögek tangensét, szinuszát és koszinuszát közvetlenül megadja.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;A PROGRAMBAN A VALOS SZAM ARITMETIKARA LATUNK PELDAT. AZ
;EREDMENYT A MICROSOFT NYOMKOVETO PROGRAMJA SEGITSEGEVEL
;NEZZUK MEG. A PROGRAM A 0-90 FOK KOZE ESO KEREK FOKERTEKEK
;SZINUSZAT SZAMOLJA KI.

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
```

```
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
ANGLE   DW      65
TEMP    DW      ?
CONST   DD      180.0
SINE    DQ      ?
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TURLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

;#########################################################################

        MOV     AX,ANGLE        ;A SZOGERTEK BETOLTESE AX-BE
        CMP     AX,45           ;A SZOG 45 FOKNAL NAGYOBB ?
        JG      FIXIT           ;HA IGEN, UGRAS FIXIT-RE
        JMP     CONT            ;HA NEM, UGRAS CONT-RA
FIXIT:  NEG     AX              ;90-AX KEPZESE
        ADD     AX,90           ;AZ EREDMENYHEZ 90 FOKOT HOZZAADAUNK
CONT:   MOV     TEMP,AX         ;ES TEMP-BEN ELHELYEZZUK

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FLDPI                   ;PI RAHELYEZESE A VEREMRE
        FLD     CONST           ;CONST RAHELYEZESE A VERMERE
        FDIV                    ;A KONSTANS OSZTASA PI-VEL
        FILD    TEMP            ;A KORRIGALT SZOGERTEK RATOLTESE A
        FMUL                    ;VEREMRE, ES SZORZASA AZ ELOZO
                                ;EREDMENNYEL
        FPTAN                   ;TANGENSKEPZES
        FWAIT

        MOV     AX,ANGLE        ;A SZOG MERETENEK ELLENORZESE
        CMP     AX,45           ;HA 45 FOK VAGY ENNEL KISEBB, AKKOR
                                ;SZINUSZT KEPZUNK
        JG      COSIN           ;HA NAGYOBB, AKKOR KOSZINUSZT KEPZUNK
        JMP     SININ

COSIN:  FXCH    ST(1)           ;HA KOSZINUSZ, AKKOR CSERE A VEREMBEN
SININ:  FMUL    ST(0),ST        ;A SZINUSZNAL MINDEN A HELYEN MARAD
        FXCH    ST(1)           ;SZINUSZ VAGY KOSZINUSZ SZAMITAS
        FLD     ST(0)           ;TRIGONOMETRIKUS UTON
        FMUL    ST(0),ST        ;
        FADD    ST(0),ST(2)     ;
        FSQRT                   ;
        FDIVP   ST(1),ST
        FSTP    SINE            ;AZ EREDMENY ELMENTESE EGY TAROLASI
                                ;HELYRE
        FWAIT                   ;SZINKRONIZALAS

;#########################################################################
```

```
        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

Ha a 80387-es segédprocesszorral dolgozunk, akkor a csillagokból álló két sor közé eső programrész a következő néhány programsorral cserélhető fel:

```
FINIT
FLDPI
FLD     CONST
FDIV
FILD    ANGLE
FMUL
FSIN
FSTP    SINE
FWAIT
```

A programban a következő szabályt használjuk:

$$SIN(X) = COS(PI/2 - X)$$

Emlékeztetünk arra is, hogy a PI/4 radián (45 fok) szinusza és koszinusza egyaránt 0.707. A program három fő részből áll:

1. Az eredeti szög radiánba alakítása (PI/4 kivonása, ha szükséges).
2. A szög tangensének képzése.
3. Ha a szög PI/4-nél nagyobb, koszinusz konverzió, egyébként pedig szinusz konverzió.

A következő kódrészlet a szöget 0 és PI/4 közé állítja be:

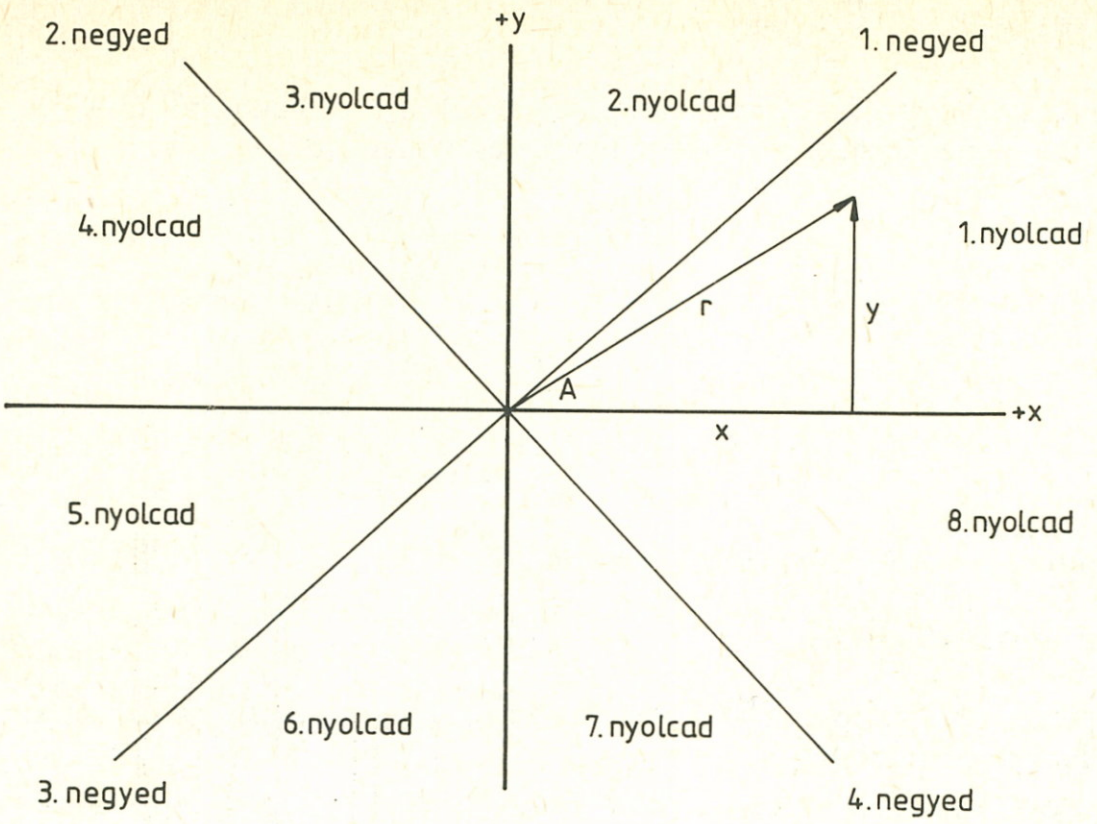
```
        MOV     AX,ANGLE        ;A SZOGERTEK BETOLTESE AX-BE
        CMP     AX,45           ;A SZOG 45 FOKNAL NAGYOBB ?
        JG      FIXIT           ;HA IGEN, UGRAS FIXIT-RE
        JMP     CONT            ;HA NEM, UGRAS CONT-RA
FIXIT:  NEG     AX              ;90-AX KEPZESE
        ADD     AX,90           ;AZ EREDMENYHEZ 90 FOKOT HOZZAADAUNK
CONT:   MOV     TEMP,AX         ;ES TEMP-BEN ELHELYEZZUK
```

Az ANGLE változó, amit egész szónak definiáltunk, az eredeti szögértéket fokokban tartalmazza. Ha a szög 45 foknál nagyobb, a FIXIT cimkére, egyébként pedig a CONT cimkére ágaztatjuk el a programot. A FIXIT-nél található utasítás először AX komplemensét képezi (kivonás 90 fokból), és az eredményt 90 fokhoz hozzáadja. A TEMP változóban tárolt eredmény tangensének képzése a következő programrészben történik:

```
FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
FLDPI                   ;PI RAHELYEZESE A VEREMRE
FLD     CONST           ;CONST RAHELYEZESE A VERMERE
FDIV                    ;A KONSTANS OSZTASA PI-VEL
FILD    TEMP            ;A KORRIGALT SZOGERTEK RATOLTESE A
FMUL                    ;VEREMRE, ES SZORZASA AZ ELOZO
                        ;EREDMENNYEL
FPTAN                   ;TANGENSKEPZES
FWAIT
```

Ez a kódrészlet hasonlít ahhoz, ami az előző példában szerepelt, azzal a különbséggel, hogy TEMP most egy egész érték. Amikor az FPTAN utasítást végrehajtjuk, X az ST-ben, Y pedig ST(1)-ben lesz.

A 9–3. ábrán a szögek szinuszának képzése követhető nyomon.



$$R = \sqrt{X^2 + Y^2}$$

Az első negyedben: $\sin = \cos(\pi/4 - \theta)$

$$\sin\theta = \frac{Y}{R} = \frac{Y}{\sqrt{X^2 + Y^2}}$$

$$\cos\theta = \frac{X}{R} = \frac{X}{\sqrt{X^2 + Y^2}}$$

9–3. ábra Színusz- és koszinusz-számítás

Mivel az ábrán látható X és Y értékeket a megelőző művelet eredményeképpen készen kapjuk, csak arra van szükségünk, hogy R-t (sugár) kiszámítsuk.

```
(1)   COSIN:  FXCH    ST(1)          ;HA KOSZINUSZ, AKKOR CSERE A VEREMBEN
(2)   SININ:  FMUL    ST(0),ST       ;A SZINUSZNAL MINDEN A HELYEN MARAD
(3)           FXCH    ST(1)          ;SZINUSZ VAGY KOSZINUSZ SZAMITAS
(4)           FLD     ST(0)          ;TRIGONOMETRIKUS UTON
(5)           FMUL    ST(0),ST       ;
(6)           FADD    ST(0),ST(2)    ;
(7)           FSQRT                  ;
(8)           FDIVP   ST(1),ST
(9)           FSTP    SINE           ;AZ EREDMENY ELMENTESE EGY TAROLASI
                                     ;HELYRE
(10)          FWAIT                  ;SZINKRONIZALAS
```

	Szinusz			Koszinusz		
	ST(0)	ST(1)	ST(2)	ST(0)	ST(1)	ST(2)
(0)	X	Y		X	Y	
(1)				Y	X	
(2)	X^2	Y		Y^2	X	
(3)	Y	X^2		X	Y^2	
(4)	Y	Y	X^2	X	X	Y^2
(5)	Y^2	Y	X^2	X^2	X	Y^2
(6)	Y^2+X^2	Y	X^2	X^2+Y^2	X	Y^2
(7)	$(Y^2+X^2)^{.5}$	Y	X^2	$(X^2+Y^2)^{.5}$	X	Y^2
(8)	R*	Y/R	X^2	R	X/R	Y^2
(9)	R	Sin	X^2	R	Sin	Y^2

*$R = \sqrt{X^2+Y^2}$

9–2. táblázat A verem tartalma a szinusz-, ill. koszinuszértékek kiszámítása után

A 9–2. táblázat segítségével kövessük végig, hogy mi történik a társprocesszor vermében akkor, amikor az előbbi kódrészletet a rendszer végrehajtja. Ha az eredeti szögérték PI/4 (45 fok) vagy ennél kisebb, akkor szinuszt, egyébként pedig koszinuszt képezünk. A 2. lépésben vagy X^2 vagy Y^2 négyzetét állítjuk elő ST-ben. A 9–2. táblázatból látható, hogy ST-ben végül az $R = (Y^2 + X^2)^{0.5}$ érték szerepel. A 8. lépésben, amikor Y-t R-rel (szinusz) vagy X-et R-rel (koszinusz) elosztjuk, a hányados ST(1)-be lép. A 9. lépés előtt a hányados ST-ben található, majd pedig a SINE változóban.

Az eredményt a Microsoft nyomkövető programjával nézzük meg.

```
46FB:0158   F9 9B DD 1E 08 00 9B CB --0.1655102779799079E+57
46FB:0160   41 00 19 00 00 00 34 43  +0.5629499535851585E+16
46FB:0168   67 D7 2D 30 79 00 ED 3F  +0.9063077870366499E+0
46FB:0170   4D 59 53 54 41 43 4B 20  +0.4066692443677049E--152
```

A 65 fok szinusza 0.9063077870366499E+0. Micsoda pontosság! A 80387-es processzor használata a trigonometriai és algebrai számításokat szükségtelenné teszi.

Nagy pontosságú szinusztáblázat készítése

Az előbbi fejezetrészben bemutatott programon, ha néhány kisebb átalakítást elvégzünk, a szögértékek szinusztáblázatát készíthetjük el 0-tól 90 fokig. A következő listában minden szükséges változtatás szerepel.

```
;80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETT PROGRAM
;A PROGRAMBAN A VALOS SZAM ARITMETIKARA LATUNK PELDAT. AZ
;EREDMENYT A MICROSOFT NYOMKOVETO PROGRAMJA SEGITSEGEVEL
;NEZZUK MEG. A PROGRAM A 0-TOL 90 FOKIG A KEREK FOKERTEKEK
;SZINUSZAT SZAMOLJA KI, ES A MEMORIABAN TAROLJA

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET
```

```
STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
ANGLE   DW      0               ;KIINDULO SZOGERTEK
TEMP    DW      ?               ;IDEIGLENES TAROLASI HELY
CONST   DD      180.0           ;KONSTANS A KONVERZIOHOZ
SINE    DQ      91 DUP (?)      ;AZ EREDMENYEK TAROLASI HELYE
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE


        LEA     BX,SINE
AGAIN:  MOV     AX,ANGLE        ;A SZOGERTEK BETOLTESE AX-BE
        CMP     AX,45           ;A SZOG 45 FOKNAL NAGYOBB ?
        JG      FIXIT           ;HA IGEN, UGRAS FIXIT-RE
        JMP     CONT            ;HA NEM, UGRAS CONT-RA
FIXIT:  NEG     AX              ;90-AX KEPZESE
        ADD     AX,90           ;AZ EREDMENYHEZ 90 FOKOT HOZZAADAUNK
CONT:   MOV     TEMP,AX         ;ES TEMP-BEN ELHELYEZZUK

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FLDPI                   ;PI RAHELYEZESE A VEREMRE
        FLD     CONST           ;CONST RAHELYEZESE A VERMERE
        FDIV                    ;A KONSTANS OSZTASA PI-VEL
        FILD    TEMP            ;A KORRIGALT SZOGERTEK RATOLTESE A
        FMUL                    ;VEREMRE, ES SZORZASA AZ ELOZO
                                ;EREDMENNYEL
        FPTAN                   ;TANGENSKEPZES
        FWAIT                   ;SZINKRONIZALAS

        MOV     AX,ANGLE        ;A SZOG MERETENEK ELLENORZESE
        CMP     AX,45           ;HA 45 FOK VAGY ENNEL KISEBB, AKKOR
                                ;SZINUSZT KEPZUNK
        JG      COSIN           ;HA NAGYOBB, AKKOR KOSZINUSZT KEPZUNK
        JMP     SININ

COSIN:  FXCH    ST(1)           ;HA KOSZINUSZ, AKKOR CSERE A VEREMBEN
SININ:  FMUL    ST(0),ST        ;A SZINUSZNAL MINDEN A HELYEN MARAD
        FXCH    ST(1)           ;SZINUSZ VAGY KOSZINUSZ SZAMITAS
        FLD     ST(0)           ;TRIGONOMETRIKUS UTON
        FMUL    ST(0),ST        ;
        FADD    ST(0),ST(2)     ;
        FSQRT                   ;
        FDIVP   ST(1),ST
        FSTP    SINE[BX]        ;AZ EREDMENY ELMENTESE EGY TAROLASI
                                ;HELYRE
        FWAIT                   ;SZINKRONIZALAS
```

```
        ADD     BX,08H          ;RAMUTATAS A KOVETKEZO SZINUSZERTEK
                                ;TAROLASI HELYERE
        ADD     ANGLE,1         ;A SZOGERTEK NOVELESE
        CMP     ANGLE,90        ;A CIKLUS 0-TOL 90 FOKIG TART
        JLE     AGAIN           ;HA A SZOG KISEBB MINT 91, ISMETLES

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A lényeges eltéréseket külön kiemeljük.

```
FSTP    SINE[BX]        ;AZ EREDMENY ELMENTESE EGY TAROLASI
                        ;HELYRE
FWAIT                   ;SZINKRONIZALAS

ADD     BX,08H          ;RAMUTATAS A KOVETKEZO SZINUSZERTEK
                        ;TAROLASI HELYERE
ADD     ANGLE,1         ;A SZOGERTEK NOVELESE
CMP     ANGLE,90        ;A CIKLUS 0-TOL 90 FOKIG TART
JLE     AGAIN           ;HA A SZOG KISEBB MINT 91, ISMETLES
```

Az adatszegmensben a SINE változót úgy deklaráltuk, hogy 91 darab négyesszó tárolására legyen képes. BX-et tehát 8-cal kell növelnünk ahhoz, hogy a következő tárolási helyre rá tudjunk mutatni. Az ANGLE változó tartalmát 90-nel összehasonlítva eldönthető, hogy a számítás megismétlésére, vagy a program befejezésére van szükség.

Szinuszhullám ábrázolása

Az előbbi programban a 0 és 90 fok közötti szögek szinuszát számítottuk ki, és a valós eredményt SINE[BX]-ben helyeztük el. Az értékek ábrázolásához számos módosítást kell elvégeznünk az eredeti programon. Először is az eredményeket egész számokká kell átalakítanunk, másodszor olyan méretarányt kell felvennünk, hogy a koordináta-értékek elférjenek a képernyőn, harmadszor a 91-től 360 fokig kell kiterjeszteni az értelmezési tartományt.

A valós szám egésszé alakítása igen egyszerű. Az FSTP SINE utasítást fel kell cserélni az FISTP SINE utasítással.

Az IBM PC/AT nagyfelbontású képernyője függőlegesen 200, vízszintesen 640 képelem elhelyezésére képes. A szinuszértékek $+1.00$ és -1.00 között változnak. Ha a szorzótényezőt 100-nak választjuk, akkor a hullám $+100$ és -100 között helyezkedik el. Ha egy-egy szögérték egy-egy vízszintes pozíciót jelent, akkor a 640 vízszintes képelemből 360-at használunk fel. Annak érdekében, hogy a hullám a képernyőn középre kerüljön, a képernyő bal szélétől 140 képelemes eltolást használunk ($640 - 360 = 280$, és $280/2 = 140$). Az első két lépést megvalósító kódrészlet a következő:

```
FIMUL   MULTIP
FISTP   SINE[SI]        ;AZ EREDMENY ELMENTESE
```

ahol MULTIP értékét az adatszegmensben 100-nak definiáltuk. Az SI regiszter az egész számokat tartalmazó SINE táblázat indexelését szolgálja. A képernyő felbontóképessége miatt, a táblázat szinuszértékeinek pontossága jelentős mértékben csökken. Az első szögnegyed (0–90) kiterjesztése a következő három szögnegyedre (91–360) elég komoly feladat. Két lehetőség kínálkozik:

1. Megváltoztatjuk az algoritmust, és 361 ponthoz számoljuk ki a szinuszt.
2. A 0 és 90 közötti értékeket – figyelembe véve a trigonometriai szabályokat – kiterjesztjük a következő három szögnegyedre.
   (A 8. fejezetben ezt a megoldást választottuk.)

```
+0.0E+0
+0.1745240643728351E)−1
+0.3489949670250097E)−1
+0.5233595624294383E)−1
+0.697564737441253E)−1
+0.8715574274765818E)1
+0.1045284632676535E+0
+0.1218693434051475E+0
+0.1391731009600654E+0
+0.1564344650402309E+0
+0.1736481776669304E+0
+0.1908089953765448E+0
+0.2079116908177593E+0
+0.224951054343865E+0
+0.2419218955996677E+0
+0.2588190451025207E+0
+0.2756373558169992E+0
+0.2923717047227367E+0
+0.3090169943749475E+0
+0.3255681544571566E+0
+0.3420201433256687E+0
+0.3583679495453003E+0
+0.374606593415912E+0
+0.3907311284892738E+0
+0.4067366430758002E+0
+0.4226182617406994E+0
+0.4383711467890774E+0
+0.4539904997395468E+0
+0.4694715627858908E+0
+0.484809620246337E+0
+0.5E+0
+0.5150380749100542E+0
+0.5299192642332049E+0
+0.5446390350150271E+0
+0.5591929034707468E+0
+0.573576436351046E+0
+0.5877852522924731E+0
+0.6018150231520483E+0
+0.6156614753256583E+0
+0.6293203910498375E+0
+0.6427876096865394E+0
+0.6560590289905073E+0
+0.6691306063588582E+0
+0.6819983600624985E+0
+0.6946583704589973E+0
+0.7071067811865476E+0
+0.7193398003386512E+0
+0.7313537016191705E+0
+0.7431448254773942E+0
+0.754709580222772E+0
+0.766044443118978E+0
+0.7771459614569709E+0
+0.7880107536067219E+0
+0.7986355100472928E+0
+0.8090169943749475E+0
+0.8191520442889918E+0
+0.8290375725550417E+0
+0.8386705679454241E+0
+0.848048096156426E+0
+0.8571673007021123E+0
+0.8660254037844386E+0
+0.8746197071393959E+0
+0.882947592858927E+0
+0.8910065241883679E+0
+0.898794046299167E+0
+0.9063077870366499E+0
+0.9135454576426009E+0
+0.9205048534524404E+0
+0.9271838545667874E+0
+0.9335804264972017E+0
+0.9396926207859084E+0
+0.945518575599316 8E+0
+0.951056516295153 5E+0
+0.956304755963035 4E+0
+0.9612616959383189E+0
+0.9659258262890683E+0
+0.9702957262759965E+0
+0.9743700647852352E+0
+0.9781476007338057E+0
+0.981627183447664E+0
+0.984807753012208E+0
+0.9876883405951378E+0
+0.9902680687415704E+0
+0.992546151641322E+0
+0.9945218953682733E+0
+0.9961946980917455E+0
+0.9975640502598242E+0
+0.9986295347545738E+0
+0.9993908270190958E+0
+0.9998476951563913E+0
+0.1E+1
```

9–3. táblázat 0 és 90 fok közötti egész szögértékek nagy pontosságú szinuszértékei

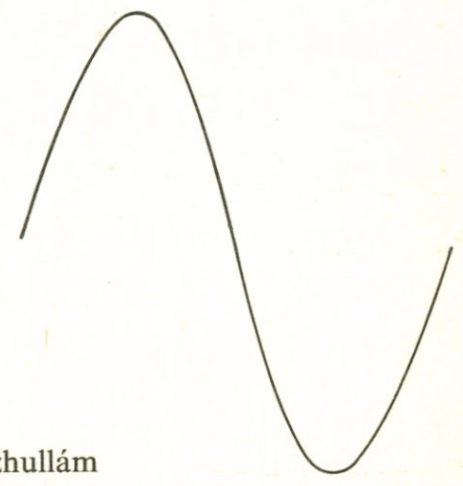
A jelenlegi és az előző programok között az a fő különbség, hogy ebben a programban az értékek akkor kerülnek a táblázatba, amikor kiszámítjuk azokat. (Nem nekünk kell előzetesen feltöltenünk a táblázatot.) Ez a program a 8–1. ábrán látható program, és a 9. fejezet két utóbbi programjának a keveréke. A programlistán próbáljuk felfedezni az előző programokból már ismerős kódrészleteket.

```
;A PROGRAMOT 80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETTEK.
;VALOS SZAM ARITMETIKAT BEMUTATO PROGRAM, AMELYBEN AZ
;EREDMENYEKET EGESZEKKE ALAKITJUK AT,ES EGY KONSTANSSAL
;BESZOROZZUK. A PROGRAM 0-TOL 90 FOKIG AZ EGESZ SZOGERTEKEK
;SZINUSZAT SZAMOLJA KI, ES A SZINUSZHULLAMOT A 0-360
;ERTELMEZESI TARTOMANYBAN ABRAZOLJA A KEPERNYON.

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET

SETSCREEN MACRO                 ;;NAGYFELBONTASU GRAFIKUS KEPERNYO
        MOV     AH,00           ;;BEALLITASA
        MOV     AL,06           ;;200*640 FEKETE/FEHER PONT
        INT     10H
        ENDM

WRITEDOT MACRO                  ;;A PONT MEGJELENITESET SZOLGALO MAKRO
        MOV     AH,12
        MOV     AL,01
        MOV     CX,POS
        ADD     CX,140          ;;AZ ABRA KOZEPRE ALLITASA A KEPERNYON
        MOV     DH,00
        MOV     DL,INDEP
        INT     10H
        ENDM

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
ANGLE   DW      0               ;A KIINDULO SZOGERTEK
TEMP    DW      ?               ;IDEIGLENES TAROLASI HELY
CONST   DD      180.0           ;KONSTANS A SZOG_RADIANBA VALO
                                ;ATSZAMITASAHOZ
MULTIP  DW      100             ;AZ ERTEKKESZLET BEALLITASA A 0-100
                                ;TARTOMANYBA
POS     DW      0
INDEP   DB      0
SINE    DW      91 DUP (?)      ;AZ EREDMENYEK TAROLASI HELYE
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER HELYENEK ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESA A VEREMRE
```

```
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE
        MOV     SI,0            ;SI-VEL INDEXELUNK A SINE VALTOZOBAN


AGAIN:  MOV     AX,ANGLE        ;A SZOGERTEK BETOLTESE AX-BE
        CMP     AX,45           ;A SZOG 45 FOKNAL NAGYOBB ?
        JG      FIXIT           ;HA IGEN, UGRAS FIXIT-RE
        JMP     CONT            ;HA NEM, UGRAS CONT-RA
FIXIT:  NEG     AX              ;90-AX KEPZESE
        ADD     AX,90           ;AZ EREDMENYHEZ 90 FOKOT HOZZAADAUNK
CONT:   MOV     TEMP,AX         ;ES TEMP-BEN ELHELYEZZUK

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FLDPI                   ;PI RAHELYEZESE A VEREMRE
        FLD     CONST           ;CONST RAHELYEZESE A VERMERE
        FDIV                    ;A KONSTANS OSZTASA PI-VEL
        FILD    TEMP            ;A KORRIGALT SZOGERTEK RATOLTESE A
        FMUL                    ;VEREMRE, ES SZORZASA AZ ELOZO
                                ;EREDMENNYEL
        FPTAN                   ;TANGENSKEPZES
        FWAIT

        MOV     AX,ANGLE        ;A SZOG MERETENEK ELLENORZESE
        CMP     AX,45           ;HA 45 FOK VAGY ENNEL KISEBB, AKKOR
                                ;SZINUSZT KEPZUNK
        JG      COSIN           ;HA NAGYOBB, AKKOR KOSZINUSZT KEPZUNK
        JMP     SININ

COSIN:  FXCH    ST(1)           ;HA KOSZINUSZ, AKKOR CSERE A VEREMBEN
SININ:  FMUL    ST(0),ST        ;A SZINUSZNAL MINDEN A HELYEN MARAD
        FXCH    ST(1)           ;SZINUSZ VAGY KOSZINUSZ SZAMITAS
        FLD     ST(0)           ;TRIGONOMETRIKUS UTON
        FMUL    ST(0),ST        ;
        FADD    ST(0),ST(2)     ;
        FSQRT                   ;
        FDIVP   ST(1),ST
        FIMUL   MULTIP
        FISTP   SINE[SI]        ;AZ EREDMENY ELMENTESE A SINE
                                ;TABLAZATBA
        FWAIT                   ;SZINKRONIZALAS

        ADD     SI,02H          ;RAMUTATAS A KOVETKEZO SZINUSZERTEK
                                ;TAROLASI HELYERE
        ADD     ANGLE,1         ;A SZOGERTEK NOVELESE
        CMP     ANGLE,90        ;A CIKLUS 0-TOL 90 FOKIG TART
        JLE     AGAIN           ;HA A SZOG KISEBB MINT 91, ISMETLES

;EZ A RUTIN A 8-1-ES ABRAN LATHATO PROGRAMHOZ HASONLOAN
;JELENITI MEG  A PONTOKAT A NAGYFELBONTASU KEPERNYON

        SETSCREEN               ;A 200*640-ES GRAFIKUS KEPERNYO
                                ;BEALLITASA
REPT:   MOV     SI,0            ;RAMUTATAS A TABLAZAT KEZDOELEMEHEZ
        MOV     AX,POS          ;A SZOG ERTEKE AL-BE KERUL
        CMP     AX,180          ;NAGYOBB MINT 180 ?
        JLE     NEWQUAD         ;HA KISEBB, AKKOR A SZOG AZ ELSO VAGY
                                ;A MASODIK SZOGNEGYEDBE ESIK
```

```
        SUB     AX,180          ;A SZOG KIIGAZITASA, HA AZ 180-NAL
                                ;NAGYOBB
NEWQUAD: CMP    AX,90           ;A SZOG NAGYOBB MINT 90 FOK ?
        JLE     SECQUAD         ;HA IGEN, AKKOR A MASODIK SZOGNEGYEDBE
                                ;ESIK
        NEG     AX              ;A SZOGERTEK BEALLITASA NEGATIVRA
        ADD     AX,180          ;HA A SZOG>=90, KIIGAZITAS
SECQUAD: ADD    SI,AX           ;ELTOLASI ERTEK BETOLTESE SI-BE
        SHL     SI,1            ;2-VEL SZORZAS
        MOV     AL,BYTE PTR SINE[SI]  ;TABLAZATOS SZINUSZERTEK
                                ;BEKERESE
        CMP     POS,180         ;HA A SZOG<180 , MEGJELENITES A
                                ;KEPERNYON
        JGE     BIGDIS
        NEG     AL              ;EGYEBKENT NEGATIV A SZINUSZ
        ADD     AL,100          ;A SZINUSZERTEK ELHELYEZESE A KEPERNYON
        JMP     READY           ;A VIZSZINTES ELTOLASI ERTEK 100
BIGDIS: ADD     AL,99
READY:  MOV     INDEP,AL        ;A PONTOT MEGJELENITO RUTIN
                                ;ELOKESZITESE
        WRITEDOT                ;MAKRO HIVAS
        ADD     POS,1           ;A KOVETKEZO SZOG BEKERESE
        CMP     POS,360         ;MIND A 360 FOKKAL VEGEZTUNK ?
        JLE     REPT            ;HA NEM, ISMETLES
;A SZINUSZHULLAMOT ABRAZOLO RUTIN VEGE

;VARAKOZAS EGY BILLENTYU LENYOMASARA
        MOV     AH,07           ;BILLENTYUZET PARAMETER
        INT     21H             ;A BILLENTYUZET BEOLVASASA
        MOV     AH,00           ;KEPERNYO PARAMETER
        MOV     AL,03           ;25*80-AS SZINES UZEMMOD
        INT     10H             ;BEALLITASA
        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A 9–4. ábrán a program futtatásának végeredménye látható. A társprocesszor felhasználásával a pontok helyzetének kiszámítása és ábrázolása igen gyors.



9–4. ábra Színuszhullám

9.11 FOURIER-SOROK KISZÁMÍTÁSA ÉS ÁBRÁZOLÁSA

A most bemutatásra kerülő grafikus programmal a 80286/80386-os processzorral elérhető gyorsaságot és precizitást mutatjuk be, felhasználva az Intel társprocesszort is. A feladatot egy 9 MHz-es 80286-os IBM AT számítógépen írtuk meg és futtatuk, kihasználva a 80287-es lehetőségeit is.

Jean Baptiste Fourier (1768–1830) fedezte fel, hogy majdnem minden periódikus hullám előállítható harmónikus szinuszhullámok összességeként. A jelenséget leíró általános Fourier-egyenlet a következő:

$$y = A + A1(\text{SIN } \omega t) + A2(\text{SIN } 2\omega t) + A3(\text{SIN } 3\omega t) + A4(\text{SIN } 4\omega t)\ldots$$

Néhány hullámforma esetén csak a páros vagy csak a páratlan harmonikusok szerepelnek, míg más esetekben mindkettő. Bizonyos esetekben a Fourier-sor egymást követő elemeinek előjele váltakozó (+ és −).

Ebben a példában négyszögjelet szerkesztünk a Fourier-sor harmonikus tagjainak összeadásával. Minél több tagot vonunk bele az összegzésbe, annál pontosabb eredményt kapunk. Először a négyszögjel általános Fourier-egyenletét közöljük:

$$y = (\text{SIN } \omega t) + (1/3)\,(\text{SIN } 3\omega t) + (1/5)\,(\text{SIN } 5\omega t) + (1/7)\,(\text{SIN } 7\omega t)\ldots.$$

A végeredmény kialakításában csak a páratlan harmónikusok vesznek részt. Ha az egyenletben csak egy harmónikust választunk, akkor az eredmény maga a szinuszhullám lesz. Az egymást követő tagok előtt egy-egy tört szorzó áll, és mivel ez a tört egyre kisebb, a hozzá tartozó harmonikus tag befolyása a hullám alakjára egyre csökken.

A Fourier-sor egyes elemeit külön-külön számítjuk ki a programban és a tagok összegét az új elem hozzáadásával mindig aktualizáljuk. Ha pl. egy Fourier-sor 500 elemét akarjuk a számításainkba bevonni, akkor 500 különböző szinuszértéket kell a programnak meghatároznia és összeadnia ahhoz, hogy a képernyőn a görbe egyetlen pontját ábrázolni tudja. Ezt az eljárást természetesen minden újabb pont esetén meg kell ismételni. A számítógép tehát abban az esetben, ha 360 pontot ábrázolunk, 500 * 360 = 180 000 számítást végezne el. Meddig tartana ezt kiszámolni egy számológéppel?! Tegyük fel, hogy 5 perc alatt 50 elemet tudunk meghatározni – ami nem is olyan rossz munkatempó –, akkor 50 perc lenne szükséges 500 elemhez. 50 perc szorozva 360 ponttal 18000 perc, azaz 12 és fél nap megállás nélküli és hibamentes! munkát jelentene. Meddig tartana ez a feladat a BASIC, a PASCAL vagy a PLI nyelv felhasználásával? A későbbiekben erre a kérdésre is választ adunk.

Most lássuk a program listáját!

```
;A PROGRAMOT 80287/80387-ES TARSPROCESSZORRAL ELLATOTT SZAMITOGEPEKRE
;KESZITETTEK.
;A PROGRAM NEGYSZOGHULLAMOT ALLIT ELO A FOURIER-SOR MEGFELELO
;ELEMEINEK OSSZEADASAVAL. AZ EREDMENYT A KEPERNYON ABRAZOLJUK.
;HA 500 HARMONIKUS TAGOT IRUNK ELO, AKKOR A PROGRAM FUTASI
;IDEJE KB. 1.8 PERC.


PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA
```

```
.8087                           ;A TARSPROCESSZORRA VONATKOZO
                                ;PSZEUDO MUVELET

HARMONIC MACRO                  ;;A BEVITT ADAT EGY NEGYJEGYU DECIMALIS
        LOCAL   MOREIN,DONE     ;;SZAM, AMIT A BILLENTYUZETROL IRUNK BE
        PUSH    AX              ;;A BEFOLYASOLT REGISZTEREK ELMENTESE
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     DX,0            ;;DX FELTOLTESE NULLAKKAL
        MOV     CX,5            ;;CSAK 4 SZAMJEGYBOL ALLO BEVITELT
                                ;;ENGEDUNK MEG
MOREIN:MOV      AH,1            ;;PARAMETERBEALLITAS A DOS
                                ;;MEGSZKITASHOZ
        INT     21H             ;;ASCII KARAKTER BEKERESE A
                                ;;BILLENTYUZETROL
        CMP     AL,30H          ;;NAGYOBB MINT 30H ?
        JL      DONE            ;;HA IGEN, AKKOR ADATBEKERES TORTENIK
        CMP     AL,39H          ;;KISEBB MINT 39H ?
        JG      DONE            ;;HA IGEN, AKKOR ADATBEKERES TORTENIK
        AND     AX,000FH        ;;AZ ALSO 4 BITET MEGORIZZUK ES
        PUSH    AX              ;;RAHELYEZZUK A VEREMRE
        MOV     AX,DX           ;;AZ OSSZEGYUJTOTT SZAM AX-BE KERUL
        MOV     BX,10           ;;SZORZAS 10-ZEL
        MUL     BX
        MOV     DX,AX           ;;A SZORZAS EREDMENYE DX-BE KERUL
                                ;;VISSZA
        POP     AX
        ADD     DX,AX           ;;DX ERTEKET A LEGUTOLJARA BELEPTETETT
                                ;;SZAMJEGYHEZ ADJUK HOZZA
        LOOP    MOREIN
DONE:   MOV     HARM,DX         ;;A BILLENTYUZETEN BEIRT SZAM A
                                ;;HARMONIKUS TAGOK SZAMAT JELENTI
        POP     DX              ;;AZ ELMENTETT REGISZTEREK TARTALMANAK
        POP     CX              ;;VISSZAALLITASA
        POP     BX
        POP     AX
        ENDM

SETSCREEN MACRO                 ;;NAGYFELBONTASU GRAFIKUS KEPERNYO
        PUSH    AX              ;;BEALLITASA
        MOV     AH,00           ;;200*640 FEKETE/FEHER PONT
        MOV     AL,06
        INT     10H
        POP     AX
        ENDM
WRITEDOT MACRO                  ;;A PONT MEGJELENITESET SZOLGALO MAKRO
        PUSH    AX
        MOV     AH,12
        MOV     AL,01
        MOV     CX,ANGLE
        ADD     CX,140          ;;AZ ABRA KOZEPRE ALLITASA A KEPERNYON
        MOV     DH,00
        INT     10H
        POP     AX
        ENDM
```

```
STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

MYDATA  SEGMENT PARA 'DATA'
ANGLE   DW      0               ;A SZOG KEZDOERTEKE
FOUR    DW      4               ;KONSTANS TAROLASA
MULTIP  DW      50              ;AZ ERTEKKESZLETET 0-100-RA ALLITJUK BE
RADIAN  DW      180             ;KONSTANS TAROLASA
REDUCE  DW      360             ;KONSTANS TAROLASA
TEMP1   DW      ?
TEMP2   DW      ?
TEMP3   DW      ?
STATWD  DW      ?               ;HELYFOGLALAS A 80287 STATUSZ-SZAVA
                                ;SZAMARA
SINE    DD      361 DUP (0)     ;A VALOS EREDMENYEK TAROLASI HELYE
ISINE   DW      361 DUP (0)     ;AZ EGESZ EREDMENYEK TAROLASI HELYE
MESSG1  DB      'KEREM AZ OSSZEADANDO HARMONIKUS TAGOK SZAMANAK'
        DB      '       BEIRASAT (0-9999) : $',13H
MESSG2  DB      'SZAMOLOK $'
HARM    DW      ?               ;A FELHASZNALO ALTAL BEIRT HARMONIKUS
                                ;TAGOK SZAMA
MYDATA  ENDS

MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE,DS:MYDATA,SS:STACK
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE

        SETSCREEN               ;A KEPERNYO LETORLESE
        LEA     DX,MESSG1       ;A BEMENO ADAT BEKERESE
        MOV     AH,9
        INT     21H
        HARMONIC                ;A BEMENO ADAT LEKEZELESE
        SETSCREEN               ;A KEPERNYO LETORLESE ES VARAKOZAS
        LEA     DX,MESSG2       ;A 'SZAMOLOK' UZENET MEGJELENITESE
        MOV     AH,9
        INT     21H
        MOV     SI,0            ;SI AZ INDEX A SINE VALTOZOBAN
NXTPT:  MOV     TEMP1,01H       ;LASSUK AZ ELSO SZOGET
ADMORE: MOV     AX,ANGLE        ;AZ EPPEN SORON KOVETKEZO SZOG
                                ;ELHELYEZESE AX-BEN
        MOV     DX,TEMP1        ;A HARM. TAG KISZAMITASA
        SHL     DX,1            ;SZORZAS 2-VEL
        SUB     DX,1            ;1 KIVONASA
        MOV     TEMP2,DX        ;AZ EGYUTTHATO ELMENTESE A KESOBBI
                                ;SZAMITASOKHOZ
        MUL     TEMP2           ;A SZOG OSSZESZORZASA AZ EGYUTTHATOVAL
        DIV     REDUCE          ;A SZOG OSZTASA 360-NAL
        MOV     TEMP3,DX        ;A MARADEK SZOGERTEK ELMENTESE

        FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
        FILD    RADIAN          ;A SZOGEK RADIANBA VALO ATSZAMITASANAK
                                ;ELOKESZITESE
```

```
      FLDPI                   ;PI RAHELYEZESE A VEREMRE
      FDIV    ST(0),ST(1)     ;OSZTAS A KONSTANSSAL
      FILD    TEMP3           ;A SZOG RAHELYEZES A VERMERE (FOKOKBAN)
      FMUL                    ;A SZOGET RADIANBA KAPJUK MEG

COMMENT / Ha rendelkezesunkre all a 80387-es processzor kibovitett
      trigonometriai utasitaskeszlete, akkor a csillagok kozotti
      kodreszlet az FSIN utasitassal helyettesitheto.         /

;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      FLDPI                   ;PI RAHLEYEZESE A VERMERE
      FIDIV   FOUR            ;ES OSZTASA 4-GYEL. AZ EREDMENY A
                              ;VERMERE KERUL
      FXCH                    ;A SZOG POZICIOJANAK KICSERELESE
      FPREM                   ;A SZOGET A 0-PI/4 TARTOMANYBA KELL
                              ;BEALLITANI A TANGENS KEPZESEHEZ
      FSTSW   STATWD          ;AZ EPPEN ERVENYES-STATUSZ SZO
                              ;ELMENTESE
      FWAIT                   ;SZINKRONIZALAS
      MOV     AX,STATWD       ;A STATUSZ-SZO ELHLYEZESE AX-BE
      TEST    AH,00000010B    ;HA BIT-TESZT EREDMENYE ZERUS, A SZOG
                              ;46 FOKNAL KISEBB
      JZ      CALTAN          ;A SZOG TANGENSENEK KISZAMITASA
      FSUBP   ST(1),ST(0)     ;EGYEBKENT A SZOGET 45 FOKBOL KIVONJUK
CALTAN:
      FPTAN                   ;A SZOG TANGENSENEK KISZAMITASA
      TEST    AH,01000010B    ;+- SZINUSZ
      JPE     TST2            ;HA IGEN, EGY MASIK VIZSGALAT
                              ;KOVETKEZIK
      JMP     FIXIT           ;EGYEBKENT ST ES ST(1) FELCSERELESE
TST2: TEST    AH,00000000B    ;+- SZINUSZ
      JPE     CALSIN          ;HA MINDKET VIZSGALAT EREDMENYE
FIXIT: FXCH                   ;ZERUS VAGY 1,AKKOR AZ ERTEKEKET A
                              ;KOSZINUSZ SZAMITASHOZ FELCSERELJUK
CALSIN:                       ;EGYEBKENT SZINUSZT SZAMOLUNK
      FMUL    ST(0),ST        ;AZ ATSZAMITASHOZ A TRIGONOMETRIAI
      FXCH    ST(1)           ;OSSZEFUGGEST HASZNALJUK
      FLD     ST(0)
      FMUL    ST(0),ST
      FADD    ST(0),ST(2)
      FSQRT
      FDIV    ST(1),ST
      TEST    AH,00000001B    ;A ZERUS POZITIV EREDMENYT JELENT
      JZ      POSSIG
      FCHS                    ;EGYEBKENT AZ EREDMENY ELOJELENEK
                              ;MEGVALTOZTATASA
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
POSSIG: FIMUL   MULTIP        ;AZ EREDMENY KIIGAZITASA A KEPERNYON
                              ;VALO ABRAZOLASHOZ
      FIDIV   TEMP2           ;OSZTAS A HARMONIKUS EGYUTTHATOVAL
      FADD    SINE[SI]        ;UJ OSSZEG KEPZESE
      FSTP    SINE[SI]        ;AZ UJ OSSZEG ELMENTESE A MEMORIABA
      FWAIT                   ;SZINKRONIZALAS


      INC     TEMP1           ;A KOVETKEZO MAGASABB HARMONIKUS TAG
                              ;KEPZESENEK ELOKESZITESE
```

```
        MOV     CX,HARM         ;A HARM. TAGOK MAX. SZAMA CX-BE KERUL
        CMP     TEMP1,CX        ;A JELENELGI ERTEK OSSZEHASONLITASA
                                ;A MAXIMUMMAL
        JG      IDXPOS          ;HA NAGYOBB, A KOVETKEZO SZOGET KELL
                                ;VENNI
        JMP     ADMORE          ;EGYEBKENT FOLYTATJUK A HARM. TAGOK
                                ;KEPZESET ES OSSZEGYUJTESET

IDXPOS: ADD     SI,04H          ;RAMUTATAS A KOV. SZINUSZERTEK HELYERE
        INC     ANGLE           ;A KOV. SZOGERTEK BEALLITASA
        CMP     ANGLE,360       ;A CIKLUS ISMETLESE 0-TOL 360-IG
        JG      TRANS           ;HA KESZ, AZ EREDMENY ATADASA
        JMP     NXTPT           ;HA NEM, AZ ELJARAS MEGISMETLESE

;A 361 VALOS SZAM A SINE TABLAZATBOL ISINE-BE KERUL EGESZ SZAMOK
;FORMAJABAN. AZ ERTEKKESZLETET +/- 100 KOZOTT ALLITJUK BE.
TRANS:  MOV     SI,00           ;AZ INDEXREGISZTER BEALLITASA 0-RA
        MOV     DI,00
        LEA     BX,SINE         ;A TABLAZATOK CIMENEK
        LEA     BP,ISINE        ;BETOLTESE
        MOV     CX,361          ;361 ERTEK ATALAKITASANAK ELOKESZITESE
TFMOR:  FLD     SINE[SI]        ;A VALOS SZAM RAHLEYEZESE A VERMRE
        FISTP   ISINE[DI]       ;EGESZKENT TORTENO LEEMELES ES ELMENTES
        ADD     SI,04           ;RAMUTATAS A KOV. VALOS SZAM HELYERE
        ADD     DI,02           ;RAMUTATAS A KOV. EGESZ SZAM HELYERE
        LOOP    TFMOR

;A PONTOK MEGJELENITESE A KEPERNYON
PLOTP:  SETSCREEN               ;A GRAFIKUS KEPERNYO BEALLITASA
        LEA     BP,ISINE        ;ISINE EFFEKTIV CIMENEK BETOLTESE
        MOV     SI,0            ;AZ ELSO ELEM KIJELOLESE
        MOV     ANGLE,0         ;AZ ELSO MEGJELENITENDO SZOG
DOALL:  MOV     DL,BYTE PTR ISINE[SI]  ;AZ ERTEK BEKERESE ES
                                ;ELHELYEZESE DL-BEN
        NEG     DL              ;KIVONAS 100-BOL
        ADD     DL,100          ;A PONT FUGGOLEGES KOORDINATAJA
        WRITEDOT                ;A PONT MEGJELENITESE MAKROVAL
        ADD     SI,02           ;A KOVETKEZO SZO HELYE
        ADD     ANGLE,1         ;A KOVETKEZO SZOG
        CMP     ANGLE,360       ;VEGEZTUNK MAR A 360 SZOGGEL ?
        JLE     DOALL           ;HA NEM, AZ ELJARAST MEGISMETELJUK

;A SZOVEGES KEPERNYO UZEMMODBA VALO VISSZAKAPCSOLASHOZ
;EGY BILLENTYU LENYOMASA SZUKSEGES
        MOV     AH,07           ;BILLENTYUZET PARAMETER
        INT     21H             ;A BILLENTYUZET BEOLVASASA
        MOV     AH,00           ;KEPERNYO PARAMETER
        MOV     AL,03           ;25*80-AS SZINES UZEMMOD
        INT     10H             ;BEALLITASA

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A program első látásra talán bonyolultnak tűnik, közelebbről megvizsgálva azonban látható, hogy bizonyos részei az előző programokból származnak.

A HARMONIC, a SETSCREEN és a WRITEDOT makrók szerepelnek a programban. A SETSCREEN és a WRITEDOT a grafikus képernyő beállításáról és a pont elhelyezéséről gondoskodik. (Ez a két makro az előző programból bemásolható.) A HARMONIC egy igen érdekes makro. Annak érdekében, hogy a programot minél rugalmasabbá tegyük, a felhasználó a program futtatásakor előírhatja a Fourier-sorban szereplő harmónikus tagok számát. Ez azt jelenti, hogy lehetővé kell tennünk több számjegyből álló mennyiség bekérését a billentyűzetről. A feladat megoldására többféle lehetőség is kínálkozik. Az általunk kiválasztott módszer remélhetőleg mindenki számára könnyen érthető és követhető lesz. A makrónak az a leglényegesebb része, amiben a négy jegyből álló decimális számot bekérjük a billentyűzetről.

```
        MOV     DX,0            ;;DX FELTOLTESE NULLAKKAL
        MOV     CX,4            ;;CSAK 4 SZAMJEGYBOL ALLO BEVITELT
                                ;;ENGEDUNK MEG
MOREIN:MOV      AH,1            ;;PARAMETERBEALLITAS A DOS
                                ;;MEGSZAKITASHOZ
        INT     21H             ;;ASCII KARAKTER BEKERESE A
                                ;;BILLENTYUZETROL
        CMP     AL,30H          ;;NAGYOBB MINT 30H ?
        JL      DONE            ;;HA IGEN, AKKOR ADATBEKERES TORTENIK
        CMP     AL,39H          ;;KISEBB MINT 39H ?
        JG      DONE            ;;HA IGEN, AKKOR ADATBEKERES TORTENIK
        AND     AX,000FH        ;;AZ ALSO 4 BITET MEGORIZZUK ES
        PUSH    AX              ;;RAHELYEZZUK A VEREMRE
        MOV     AX,DX           ;;AZ OSSZEGYUJTOTT SZAM AX-BE KERUL
        MOV     BX,10           ;;SZORZAS 10-ZEL
        MUL     BX
        MOV     DX,AX           ;;A SZORZAS EREDMENYE DX-BE KERUL
                                ;;VISSZA
        POP     AX
        ADD     DX,AX           ;;DX ERTEKET A LEGUTOLJARA BELEPTETETT
                                ;;SZAMJEGYHEZ ADJUK HOZZA
        LOOP    MOREIN
DONE:   MOV     HARM,DX         ;;A BILLENTYUZETEN BEIRT SZAM A
                                ;;HARMONIKUS TAGOK SZAMAT JELENTI
```

A CX regiszter a ciklusszámláló szerepét tölti be. A ciklus minden egyes periódusában egy-egy újabb számjegy begyűjtése történik meg. A CX tartalma 4, ami négy számjegy beírását teszi lehetővé. Az AH-ban elhelyezett érték (1) a billentyűzet beolvasását teszi lehetővé a 21H megszakítás aktualizálásakor. A billentyűzetről származó érték természetesen ASCII formátumban lesz. A makro először azt ellenőrzi, hogy a beírt érték decimális számjegy-e. (A decimális számok ASCII kódja 30H-tól 39H-ig tart.) Ha a beírt karakter nem esik ebbe a tartományba, a rutin befejeződik. AX és 000FH AND művelete az ASCII karakterből a decimális részt őrzi meg AX-ben, amit aztán ráhelyezünk a veremre. Ezután a DX-ben tárolt értéket AX-be tesszük át. A 10-es szorzót BX-ben helyezzük el, amivel az AX-ben szereplő számot beszorozzuk. Az eredmény most is AX-ben képződik és innen kerül DX-be. AX tartalmát ezután leemeljük a veremről és hozzáadjuk DX-hez, így kialakul a decimális szám. Ha helytelen számot írunk be vagy ha a négy számjegy begyűjtése megtörtént, a makro befejeződik.

Az elmondottakat próbáljuk ki egy példával is. A kiinduló helyzet legyen a következő:

1. DX egy megelőző művelet eredményeképpen tartalmazzon 9-et.
2. A program a cikluson most haladjon át másodszor.
3. A billentyűzeten 5-öt írjunk be.

A program az 5 ASCII értékét (35-öt) kapja meg, amit helyes bemenő adatnak értelmez, tehát elvégzi az (AX) AND (000FH) műveletet és az eredményt (decimális 5-öt) ráhelyezi a veremre. DX tartalma (9) AX-be kerül és 10-zel szorzódik, tehát AX-ben már 90 szerepel. Ezt az értéket kapja meg DX. Ezután az AX regiszter előzőleg tárolt tartalmát (5-öt) leemeljük a veremről. DX-et és AX-et összeadva 95 lesz az eredmény, ami a DX regiszterbe kerül. Ha ekkor a kocsivissza karakterét léptetjük a programba, a 95-öt a HARM változó kapja meg és a makro befejeződik. Még mielőtt a program tanulmányozásához hozzákezdenénk, nézzük végig, hogy milyen értékek szerepelnek az adatszegmensben:

```
ANGLE   DW      0               ;A SZOG KEZDOERTEKE
FOUR    DW      4               ;KONSTANS TAROLASA
MULTIP  DW      50              ;AZ ERTEKKESZLETET 0-100-RA ALLITJUK BE
RADIAN  DW      180             ;KONSTANS TAROLASA
REDUCE  DW      360             ;KONSTANS TAROLASA
TEMP1   DW      ?
TEMP2   DW      ?
TEMP3   DW      ?
STATWD  DW      ?               ;HELYFOGLALAS A 80287 STATUSZ-SZAVA
                                ;SZAMARA
SINE    DD      361 DUP (0)     ;A VALOS EREDMENYEK TAROLASI HELYE
ISINE   DW      361 DUP (0)     ;AZ EGESZ EREDMENYEK TAROLASI HELYE
MESSG1  DB      'KEREM AZ OSSZEADANDO HARMONIKUS TAGOK SZAMANAK
        DB              BEIRASAT (0-9999) : $',13H
MESSG2  DB      'SZAMOLOK $'
HARM    DW      ?               ;A FELHASZNALO ALTAL BEIRT HARMONIKUS
                                ;TAGOK SZAMA
```

Az ANGLE a vízszintes pozíciót jelenti a képernyőn. A FOUR, MULTIP, RADIAN és REDUCE olyan konstansok, amiket a program a különböző fázisaiban felhasznál. A TEMP1, TEMP2 és TEMP3 a részeredmények tárolását szolgálja. A STATWD tárolja a társprocesszorra vonatkozó státusz-szót, valamint segít megállapítani, hogy a szög melyik szögnegyedbe esik. A SINE táblázat 361 különböző valós összeget tartalmaz majd. Annak érdekében, hogy az eredmények pontosságát megőrizzük, a valós számokból képzett egész értékeket az ISINE táblázatban tároljuk. MESSG1 kéri be a felhasználótól a harmonikus tagok számát. A MESSG2 tájékoztat arról, hogy a számítógép hozzákezdett a számításokhoz. Semmit nem láthatunk a képernyőn mindaddig, amíg az összes tag kiszámításával a számítógép elkészül. (Ehhez egy 9 MHz-es gépnek 1,8 percre van szüksége.) A kód első részében tehát az input bekéréséről gondoskodunk:

```
SETSCREEN               ;A KEPERNYO LETORLESE
LEA     DX,MESSG1       ;A BEMENO ADAT BEKERESE
MOV     AH,9
INT     21H
HARMONIC                ;A BEMENO ADAT LEKEZELESE
SETSCREEN               ;A KEPERNYO LETORLESE ES VARAKOZAS
LEA     DX,MESSG2       ;A 'SZAMOLOK' UZENET MEGJELENITESE
MOV     AH,9
INT     21H
```

A SETSCREEN makro a nagyfelbontású grafikus üzemmódot állítja be, amialatt a képernyőt is letörli. A MESSG1 üzenet bekéri a harmonikus tagok számát. A HARMONIC makro lekezeli a felhasználó által beírt adatot. (A számok 0-tól 9999-ig szerepelhetnek. A későbbiekben majd látjuk, hogy 500 harmonikus tag figyelembevételével már igen jó négyszögjelet kaphatunk.) A MESSG2 üzenet jelzi az input elfogadását.

Ennek a programnak a programozási filozófiája eltér attól, amit az előző példáknál megszoktunk. A legutóbbi programban pl. csak a 0-tól 90-ig terjedő értékeket számoltuk ki, mivel kihasználtuk a szinuszhullám szimmetrikus tulajdonságát. Jelen esetben ez a megközelítési mód helytelen, mert nem mindegyik Fourier-sor szimmetrikus az összes nyolcadban. Most tehát mind a 361 pontot kiszámítjuk, tároljuk és ábrázoljuk. Ez egyben azt is jelenti, hogy a harmonikus szögek értelmezési tartományát 0 < = szög < = 360-ra kell korlátoznunk. A képlet egy másik érdekessége az, hogy a harmonikus tagok számának növelésével a szög szorzótényezője is egyre nagyobb lesz.

Tanulmányozzuk át figyelmesen a következő kódrészletet, ami a program leglényegesebb része.

```
        MOV     SI,0            ;SI AZ INDEX A SINE VALTOZOBAN
NXTPT:  MOV     TEMP1,01H       ;LASSUK AZ ELSO SZOGET
ADMORE: MOV     AX,ANGLE        ;AZ EPPEN SORON KOVETKEZO SZOG
                                ;ELHELYEZESE AX-BEN
        MOV     DX,TEMP1        ;A HARM. TAG KISZAMITASA
        SHL     DX,1            ;SZORZAS 2-VEL
        SUB     DX,1            ;1 KIVONASA
        MOV     TEMP2,DX        ;AZ EGYUTTHATO ELMENTESE A KESOBBI
                                ;SZAMITASOKHOZ
        MUL     TEMP2           ;A SZOG OSSZESZORZASA AZ EGYUTTHATOVAL
        DIV     REDUCE          ;A SZOG OSZTASA 360-NAL
        MOV     TEMP3,DX        ;A MARADEK SZOGERTEK ELMENTESE
```

SI-t 0-ra állítjuk be, amit a SINE táblázatban mint eltolást használunk majd. A TEMP1 változóba 1 kerül. Ez lesz a továbbiakban minden egyes szögnél a kezdőérték. TEMP1 nyomon követi a ciklusban éppen kiszámítandó Fourier harmonikus tagot. (Minden kiszámított szögnél TERMP1 értéke 1 és HARM között változik.) Az ANGLE változó az éppen érvényes vízszintes helyzetet, vagyis a szög aktuális értékét tartalmazza. A TEMP1 egy adott harmónikus tag esetén a Fourier-tényezőket segíti kiszámítani. TEMP1 értékét DX-be másoljuk, amin egy bitléptetés balra műveletet hajtunk végre (ez valójában 2-vel való szorzást jelent), majd pedig 1-gyel csökkentjük. DX most egy pártalan számot tartalmaz, amit a későbbi felhasználáshoz TEMP2-be tárolunk. Az AX-ben található szögértéket ezzel a tényezővel szorozzuk. Az AX/DX-ben képződő eredményt 360 fokkal elosztjuk, hogy a szög a 0–360 fokos tartományba essen. Csak az osztás maradékát helyezzük el a DX regiszterben. Az AX-ben szereplő egész értékek lényegtelenek, hiszen ezek úgyis csak a 360 fok többszörösét jelentik (360 fokonként ismétlődő periodikus hullámról van szó!). A maradékot TEMP3-ba tároljuk. A programnak ebben a szakaszában TEMP3-ban egy 0 és 360 fok közé eső szögérték található. A következő kódrészlet ezt a szöget radiánba alakítja át.

```
    FINIT                   ;A TARSPROCESSZOR INICIALIZALASA
    FILD    RADIAN          ;A SZOGEK RADIANBA VALO ATSZAMITASANAK
                            ;ELOKESZITESE
    FLDPI                   ;PI RAHELYEZESE A VEREMRE
    FDIV    ST(0),ST(1)     ;OSZTAS A KONSTANSSAL
    FILD    TEMP3           ;A SZOG RAHELYEZES A VERMERE (FOKOKBAN)
    FMUL                    ;A SZOGET RADIANBA KAPJUK MEG
```

A kódrészlet végrehajtásának eredményeként a szög a társprocesszor vermére kerül.

A csillagok közötti kódrészlet 0-tól 2PI radiánig bármely szög szinuszát kiszámítja. Ha 80387-es processzort használunk, ezt a kódrészletet egyetlen utasítással (FSIN) cserélhetjük fel. Egy korábbi művelettel már valamennyi szöget a 0–360 tartományba redukáltuk. Ha azonban nem a 80387-es társprocesszorral dolgozunk, akkor a szöget – a tangensképzés miatt – a 0–PI/4

tartományba kell beállítanunk. A megelőző programokból már láthatjuk, hogyan lehet egy 0–90 tartományba eső szög szinuszát meghatározni egyszerű trigonometriai összefüggések felhasználásával. Jelen esetben egy kissé más megközelítést alkalmazunk.

9.12 A STÁTUSZ-SZÓ

A 80287-es processzor státuszregisztere egy olyan 16 bites regiszter, ami a társprocesszor éppen érvényes állapotát írja le. A társprocessszorra vonatkozó kizárási feltételek a következők:

0. bit	IE	Érvénytelen művelet (Invalid Operation)
1. bit	DE	Normalizálás nélküli művelet (Denormalized Operation)
2. bit	ZE	Zérus osztás (Zero Devide)
3. bit	OE	Túlcsordulás (Overflow)
4. bit	UE	Alulcsordulás (Underflow)
5. bit	PE	Pontosság (Precision)
6. bit		Fenntartva
7. bit	IR	Megszakítás igény (Interrupt Request)
8. bit	C0	Feltételkód bit (Condition Code Bit)
9. bit	C1	Feltételkód bit (Condition Code Bit)
10. bit	C2	Feltételkód bit (Condition Code Bit)
11. bit	ST	Veremtetőmutató bit (Stack Top Pointer Bit)
12. bit	ST	Veremtetőmutató bit (Stack Top Pointer Bit)
13. bit	ST	Veremtetőmutató bit (Stack Top Pointer Bit)
14. bit	C3	Feltételkod bit (Condition Code Bit)
15. bit	B	Foglaltság (Busy)

A státusz-szó feltételi bitjeisegítségével a programban döntések hozhatók, amelyre egy példát is mutatunk:

```
FLDPI                   ;PI RAHLEYEZESE A VERMERE
FIDIV   FOUR            ;ES OSZTASA 4-GYEL. AZ EREDMENY A
                        ;VERMERE KERUL
FXCH                    ;A SZOG POZICIOJANAK KICSERELESE
FPREM                   ;A SZOGET A 0-PI/4 TARTOMANYBA KELL
                        ;BEALLITANI A TANGENS KEPZESEHEZ
FSTSW   STATWD          ;AZ EPPEN ERVENYES STATUSZ-SZO
                        ;ELMENTESE
FWAIT                   ;SZINKRONIZALAS
MOV     AX,STATWD       ;A STATUSZ-SZO ELHLYEZESE AX-BE
TEST    AH,00000010B    ;HA BIT-TESZT EREDMENYE ZERUS, A SZOG
                        ;46 FOKNAL KISEBB
JZ      CALTAN          ;A SZOG TANGENSENEK KISZAMITASA
FSUBP   ST(1),ST(0)     ;EGYEBKENTA SZOGET 45 FOKBOL KIVONJUK
```

Ebben a kódrészletben először PI-t rátöltjük a veremre, aminek következtében annak előző értéke egy hellyel lejjebb lép. A verem tetején levő értéket (PI) 4-gyel osztjuk. Ebben a helyzetben tehát ST-ben PI/4, míg ST(1)-ben a megelőző műveletből származó szögérték van. ST-t és ST(1)-gyet felcseréljük, és végrehajtunk egy FPREM utasítást. FPREM végzi el ST moduló osztását ST(1)-gyel, ami azt jelenti, hogy a verem tetején levő szöget PI/4-gyel eloszt-

juk. Tegyük fel, hogy a szög 4.88692 radián. Ha ezt PI/4-gyel elosztjuk, 6.22222-t kapunk. A parciális maradék 0.22222, ami kb. 12.73 foknak felel meg. Ha a szöget forgásszögnek értelmezzük, 282.73 fokot kapunk (sin(282.73) = −sin(90−12.73)). De honnan tudhatjuk azt, hogy a szög előjele negatív és 90 fokból ki kell vonni? Ha PI/4-gyet használjuk arányossági tényezőnek egy olyan szögnél, ami 0-tól 2PI-ig terjedhet, a teljes tartományt nyolcadokra osztjuk. Próbaképpen jelöljünk ki egy-egy szöget egy-egy nyolcadban és próbáljunk bizonyos törvényszerűségeket felderíteni (9–4. táblázat).

Szögnyolcad	Szög (radián)	Szög (PI/4)	Hányados	Maradék
1	0.1745	0.2222	0	.22222
2	1.3963	1.77778	1	.77778
3	1.7453	2.22222	2	.22222
4	2.9671	3.77778	3	.77778
5	3.3161	4.22222	4	.22222
6	4.5379	5.77778	5	.77778
7	4.8869	6.22222	6	.22222
	6.1087	7.77778	7	.77778

9–4. táblázat Nyolcadok meghatározásának módja

Egy dolog már bizonyára mindenkinek feltünt. A hányados egész része és a nyolcadok között lineáris kapcsolat van. Tehát a hányados megmondja, hogy egy szög hány egész szögnyolcadból áll, amelynek segítségével a szöget képezni tudjuk, és a megfelelő trigonometriai összefüggést használhatjuk.

Ennél a lépésnél lehetséges lenne az FDIV használata a FPREM helyett – ha ezt a megodást választjuk –. Az egész (a hányados) jelenti a szögnyolcadok számát, a maradékot pedig a tangens képzéséhez, azután a szinusz meghatározásához használhatjuk. A 9–5. táblázatban látható, hogy ugyanezek az információk az FPREM utasítás végrehajtása után a státuszszóban megtaláhatók.

Szögnyolcad	Hányados	C0	C3	C1	Művelet
1	0	0	0	0	Sin
2	1	0	0	1	Cos(PI/4)
3	2	0	1	0	Cos
4	3	0	1	1	Sin(PI/4)
5	4	1	0	0	−Sin
6	5	1	0	1	−Cos(PI/4)
7	6	1	1	0	−Cos
8	7	1	1	1	−Sin(PI/4)

9–5. táblázat A programban alkalmazott nyolcadokra vonatkozó teszt

Elsőként C1-en hajtunk végre egy vizsgálatot. Az STATWD státuszszó az AX regiszterbe kerül és a rendszer a 00000010B(02H) bináris számmal összehasonlítja AH-t. Ha a C1-gyel jelölt bit értéke ugyancsak 1, akkor a vizsgálat eredménye igaz, és nem következik be ugrás. A maradékot (ST(0)) PI/4-ből kivonja a program és az eredményt leemeli a veremről. Ennek következtében az új szög lép a verem tetejére.

```
CALTAN:
        FPTAN                   ;A SZOG TANGENSENEK KISZAMITASA
        TEST    AH,01000010B    ;+- SZINUSZ
        JPE     TST2            ;HA IGEN, EGY MASIK VIZSGALAT
                                ;KOVETKEZIK
        JMP     FIXIT           ;EGYEBKENT ST ES ST(1) FELCSERELESE
TST2:   TEST    AH,00000000B    ;+- SZINUSZ
        JPE     CALSIN          ;HA MINDKET VIZSGALAT EREDMENYE
FIXIT:  FXCH                    ;ZERUS VAGY 1, AKKOR AZ ERTEKEKET A
                                ;KOSZINUSZ SZAMITASHOZ FELCSERELJUK
```

A program a szög tangensének képzése után két különböző vizsgálatot hajt végre. Ha C3 és C1 egyaránt 0 vagy 1, a szinuszra vonatkozó trigonometriai függvényt hajtja végre, egyébként pedig ST és ST(1) felcserélésével koszinuszt képez. Ez a módszer azonos azzal, mint amit a szinuszhullámot ábrázoló programnál megismertünk.

```
CALSIN:                         ;EGYEBKENT SZINUSZT SZAMOLUNK
        FMUL    ST(0),ST        ;AZ ATSZAMITASHOZ A TRIGONOMETRIAI
        FXCH    ST(1)           ;OSSZEFUGGEST HASZNALJUK
        FLD     ST(0)
        FMUL    ST(0),ST
        FADD    ST(0),ST(2)
        FSQRT
        FDIV    ST(1),ST
```

Miután a FDIVP utasítást végrehajtottuk, a verem tetején a helyes szinuszérték lesz, ami azonban nem tartalmazza az előjelre vonatkozó információt. A szög előjele C0 vizsgálatával határozható meg. C0 értéke ui. negatív eredmény esetén 1, pozitív eredmény esetén 0.

```
        TEST    AH,00000001B    ;A ZERUS POZITIV EREDMENYT JELENT
        JZ      POSSIG
        FCHS                    ;EGYEBKENT AZ EREDMENY ELOJELENEK
                                ;MEGVALTOZTATASA
```

A verem tetején szereplő értéket ezután az ábrázoláshoz szükséges tényezővel szorozni és a megelőző számítások eredményével összesíteni kell:

```
POSSIG: FIMUL   MULTIP          ;AZ EREDMENY KIIGAZITASA A KEPERNYON
                                ;VALO ABRAZOLASHOZ
        FIDIV   TEMP2           ;OSZTAS A HARMONIKUS EGYUTTHATOVAL
        FADD    SINE[SI]        ;UJ OSSZEG KEPZESE
        FSTP    SINE[SI]        ;AZ UJ OSSZEG ELMENTESE A MEMORIABA
        FWAIT                   ;SZINKRONIZALAS
```

A program elején TEMP2-ben tárolt harmonikus szorzótényező négyszöghullám esetén a szög szorzója és ugyanakkor az egyes harmonikus tagok amplitudójának osztója is. Az FADD utasítás hozzáadja az előzetesen összegyűjtött szöget a verem tetején álló értékhez, majd az új eredményt ugyanitt helyezi el. Ha pl. 50 harmonikus tagot veszünk figyelembe, akkor 50 különböző értéket kell összeadnunk, mielőtt SI-vel a következő vízszintes pozícióra rámutatnánk.

Az iterációk számát TEMP1 szabályozza.

```
INC     TEMP1           ;A KOVETKEZO MAGASABB HARMONIKUS TAG
                        ;KEPZESENEK ELOKESZITESE
MOV     CX,HARM         ;A HARM. TAGOK MAX. SZAMA CX-BE KERUL
CMP     TEMP1,CX        ;A JELENELGI ERTEK OSSZEHASONLITASA
                        ;A MAXIMUMMAL
JG      IDXPOS          ;HA NAGYOBB, A KOVETKEZO SZOGET KELL
                        ;VENNI
JMP     ADMORE          ;EGYEBKENT FOLYTATJUK A HARM. TAGOK
                        ;KEPZESET ES OSSZEGYUJTESET
```

Egyszerű vizsgálattal megállapítható (összehasonlítás HARM-mal), hogy szükséges-e további harmonikus tagok kiszámítása. Ha igen, akkor a program az ADMORE cimkére ugrik, egyébként pedig beállítja a következő vízszintes hely indexét, és hozzákezd a következő szögértékhez tartozó harmonikus tagok szorzatának a kiszámításához.

```
IDXPOS. ADD     SI,04H          ;RAMUTATAS A KOV. SZINUSZERTEK HELYERE
        INC     ANGLE           ;A KOV. SZOGERTEK BEALLITASA
        CMP     ANGLE,360       ;A CIKLUS ISMETLESE 0-TOL 360-IG
        JG      TRANS           ;HA KESZ, AZ EREDMENY ATADASA
        JMP     NXTPT           ;HA NEM, AZ ELJARAS MEGISMETLESE
```

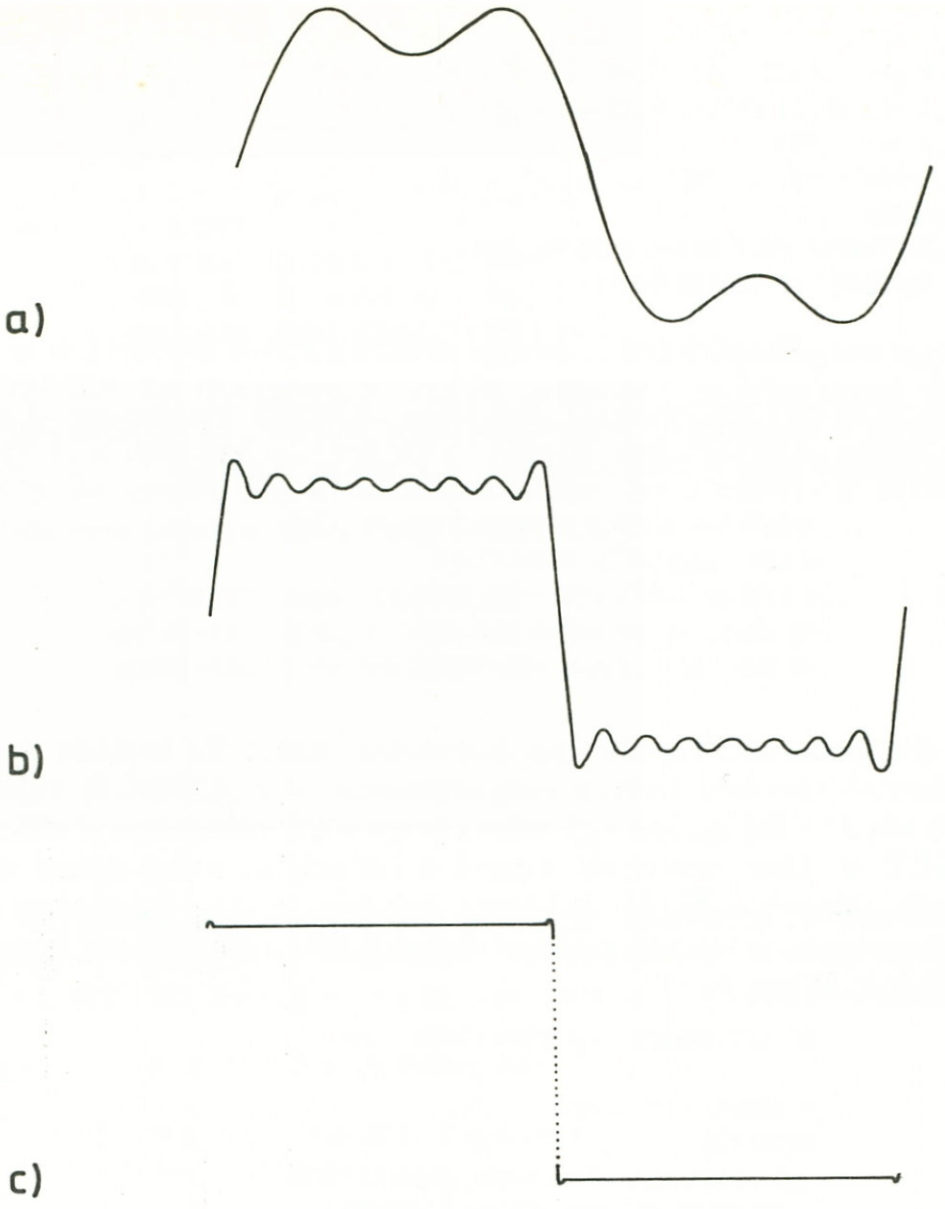
A SINE táblázat duplaszavak tárolására alkalmas, ezért SI értékét 4-gyel kell növelnünk ahhoz, hogy a következő tárolási helyre rámutassunk. Az ANGLE változóban a vízszintes pozíciók találhatók 0-tól 360 fokig. Ha egy adott pozícióra valamennyi előírt harmonikus tagot kiszámoltunk, ANGLE értékét növeljük 1-gyel mindaddig, amig mind a 361 pozíció (szög) számítása megtörténik. Mivel a SINE táblázat értékeit olyan pontosan kell kiszámolnunk, amennyire ez csak lehetséges, valós számokkal dolgozunk, azonban a képernyőn való ábrázolás előtt egészekké alakítjuk át azokat:

```
TRANS:  MOV     SI,00           ;AZ INDEXREGISZTER BEALLITASA 0-RA
        MOV     DI,00
        LEA     BX,SINE         ;A TABLAZATOK CIMENEK
        LEA     BP,ISINE        ;BETOLTESE
        MOV     CX,361          ;361 ERTEK ATALAKITASANAK ELOKESZITESE
TFMOR:  FLD     SINE[SI]        ;A VALOS SZAM RAHLEYEZESE A VERMRE
        FISTP   ISINE[DI]       ;EGESZKENT TORTENO LEEMELES ES ELMENTES
        ADD     SI,04           ;RAMUTATAS A KOV. VALOS SZAM HELYERE
        ADD     DI,02           ;RAMUTATAS A KOV. EGESZ SZAM HELYERE
        LOOP    TFMOR
```

A SINE táblázatban tárolt végredményekből mindig csak 1-et töltünk vissza a társprocesszorba, ahonnan az ISINE táblázatban már mint egész számot tároljuk. Ez a táblázat tartalmazza a szinusz-értékekből képzett képernyőn ábrázolható koordinátákat. Már csak két feladatunk maradt. Az egyik a pontok ábrázolása, a másik az ábra képernyőn tartása mindaddig, amíg egy billentyűt lenyomunk. A 9–5. ábrán három külöböző esetet figyelhetünk meg.

Az első esetben (a) 2 harmonikus tagot, a másodikban (b) 8-at, a harmadikban pedig 500-at vettünk figyelembe.

Láthatjuk, hogy a program futási ideje rövid. Az összehasonlítás végett hasonló feladatot megvalósító programokat készítettünk a BASIC, a PASCAL és az APL nyelveken (a maximális futási sebességre való kiélezés nélkül). Az ábrázoláshoz szükséges idők a 9–6. táblázatban láthatók. A futási idők összehasonlításából kiderül, hogy az assembly kóddal jelentős az időmegtakarítás.



9–5. ábra Fourier-sorok ábrázolása
a) 2 harmonikus tag figyelembevétele
b) 8 harmonikus tag figyelembevétele
c) 500 harmonikus tag figyelembevétele

Négyszöghullám-rutin 500 harmonikus tag figyelembevételével (9 MHz IBM AT computer)

BASICA	fordítás nélkül	69.75	perc
APL	fordítás nélkül	19.58	perc
BASIC	lefordítva	10.66	perc
PASCAL	lefordítva	3.21	perc
ASSEMBLY	lefordítva	1.80	

9–6. táblázat Futási idők összehasonlítása

10. KAPCSOLÓDÁS A MAGAS SZINTŰ NYELVEKHEZ

Sok programozó először azért hívja segítségül az assembly programozást, mert olyan feladatot vagy függvényt kell készítenie, amit az általa használt magas szintű nyelv nem támogat. Azok a programozók, akik az APL, BASIC, C, FORTRAN vagy PASCAL nyelveket használják, csak olyan lehetőségekkel élhetnek, amiket a fordítóprogram kínál, vagyis amit egy másik programozó elkészített és a fordítói programcsomagban elhelyezett.

Például az IBM első PASCAL fordítóprogramja nem támogatta a grafikát. Az ilyen feladatokat tehát assembly programmal kellet megoldani. Azonban sok más esetben is kerülhetünk olyan helyzetbe, hogy assembly programot kell írnunk. Például ha az eredeti fordítóprogram nem tartalmaz játékadapter, párhuzamos, port, soros port, vagy egyéb más interface támogatást, az assembly programozás eszközeit kell igénybe vennünk.

Legtöbb ilyen esetben a magas szintű program, és a különleges feladatot megvalósító assembly rutin kapcsolódását, valamint az egyik programból a másikba való paraméterátadást meg kell oldanunk. Ebben a fejezetben olyan példákat igyekeztünk összegyűjteni, amelyekkel szemléltetni tudjuk az assembly programrészletek beillesztését különböző magas szintű nyelvekbe.

Minden egyes feladatban az IBM AT számítógép játékadapterét kérdezzük le az assembly rutinnal. Lehetővé tesszük, hogy a felhasználó az alapprogramból egy paramétert adjon át, amivel meghatározza, hogy a játékadapter potenciómétere vagy nyomógombjai mintavételezése történjen meg. A feladatok figyelmes tanulmányozásával elsajátítható a paraméterátadás és -fogadás technikája. Az assembly kódnak az a része, ami minden magas szintű programba való beillesztéskor állandó, a következő:

```
;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
      MOV     AH,84H          ;BOTKORMANY INTERFACE
      MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                              ;BEKERESEHEZ
      INT     15H             ;A FELADAT ELVEGZESE
      MOV     CL,04           ;ROTALAS 4 BITTEL
      ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
      AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                              ;BITJET TARTJUK MEG
      JMP     SEND            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
POTS: MOV     AH,84H          ;BOTKORMANY INTERFACE
      MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
      INT     15H             ;BEOLVASASA
```

A botkormány interface az IBM AT számítógépnél az INT 15H-val olvasható be. AH-t ekkor 84H-ra kell beállítani. Ha DX értéke zérus, akkor a négy nyomógombra vonatkozó státuszt az AL regiszterbe kapjuk vissza.

Ha DX-et 1-re állítjuk be, akkor a négy potenciométerre vonatkozó érték (minden botkormány esetén kettő) az AX, BX, CX és DX regiszterekbe kerül. Ha tehát DX = 0, a gombokra vonatkozó információt az AL regiszter 7–4 bitjeibe kapjuk vissza, amit az alsó négy bitbe (0–3) forgatunk, és ezt az információt adjuk át a hívó programnak. Ha DX = 1, akkor a megszakítás a potenciométer értékeit a botkormány fajtájától függően (0–270 közötti egészeket) kapjuk meg a négy általános célú regiszterben. A rutin csatlakoztatását az STSC cég APL-jéhez, a Borland cég TURBO PASCAL-jához, a Microsoft BASIC és C fordítóprogramjaihoz, az IBM FORTRAN-jához és PASCAL-jához adjuk meg. A példákat figyelmesen tanulmányozva sok hasonlóság és sok finom különbség vehető észre.

10.1 ASSEMBLY-RUTIN CSATLAKOZTATÁSA AZ APL NYELVHEZ

Az APL nyelv – annak ellenére, hogy sokféle szolgáltatása van és jelentős számítási hatékonyságú – nem nagyon terjedt még el a közhasználatban. Az IBM számítógépcsaládra az APL nyelv egyik legjobb implementációját az STSC Inc. cég készítette. Ez az APL változat hagyományos APL jellemzőkön kívül teljes képernyős szövegszerkesztőt és grafikát, valamint file-kezelő rutinokat is tartalmaz.

Az APL egy interpreter típusú nyelv, amit másként kell kezelni, mint a fordítóprogrammal ellátott eseteket. Ez a példa tehát eltér a fejezet többi példáitól. A fejezetben bemutatott APL programnyelv a gépi kódú rutinok meghívását a □CALL utasítással teszi lehetővé. A művelet szintaxisa a következő:

```
eredmény←                          □CALL rutin
eredmény←          regiszterek     □CALL rutin
```

Az előbbi szintaxisban szereplő eredmény, regiszter és rutin fogalmakon a következőket kell érteni: Az eredmény a rutin által visszaadot explicit eredmény. A regiszter az AX, BX, CX, DX, BP, SI, DI regiszterekre vonatkozó 0–7 egész számokból álló vektor. A rutin az a nem üres tömb, ami a végrehajtandó rutint tartalmazza. Annak a rutinnak, amibe be akarunk lépni, a következő követelményeknek kell eleget tennie:

1. FAR típusú visszatérési utasítással (0CBH) kell lezárni.
2. Az SS és SP regiszterek eredeti tartalmát vissza kell állítani.
3. 498 byte-nál mélyebb verem nem használható.
4. A rutinnak áthelyezhetőnek kell lennie.
5. Ha bármelyik cím eltolással képződik, a kód elején 10 byte-os eltolásra van szükség.

A 10–1. ábrán látható az a program, ami a rutin használatához szükséges tényezőket tartalmazza. A fordítási eljárás során .LST file-t is készítettünk, amit a 10–2. ábrán láthatunk. Az .LST file listájának bal szélén a sorszámozás, majd ettől jobbra az assembler mnemonikok gépi kódú megfelelői helyezkednek el. Minket most elsősorban a CMP DX,0 utasítás és a második INT 15H közötti kódok érdekelnek. További kiegészítő kódra nincs szükség, mivel az APL □CALL utasítás kezeli majd a szükséges programfejet. A gépi kódot hexadecimális és decimális alakban a 10.1. táblázat tartalmazza. A 10–3.ábrán az assembly kód APL függ-

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT. ES A HIVOPROGRAMNAK ATADJA

        PUBLIC  GAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
GAME    PROC    FAR             ;AZ ELJARAS NEVE GAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+22      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
        MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                ;BEKERESEHEZ
        INT     15H             ;A FELADAT ELVEGZESE
        MOV     CL,04           ;ROTALAS 4 BITTEL
        ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
        AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                ;BITJET TARTJUK MEG
        JMP     SEND            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
POTS:   MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
        INT     15H             ;BEOLVASASA

;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

SEND:   MOV     DI,[BP]+18      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+14      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETER
        POP     BP
        RET     10

GAME    ENDP                    ;A GAME ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     GAME            ;A PROGRAM VEGE
```

10–1. ábra Assembly kód csatlakozása APL hívóprogramhoz

```
                                ;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
                                ;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
                                ;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA

                                        PUBLIC  GAME
0000                            MYCODE  SEGMENT 'CODE'
                                        ASSUME  CS:MYCODE
0000                            GAME    PROC    FAR             ;AZ ELJARAS NEVE GAME
0000  55                               PUSH    BP              ;A BP REGISZTER ELMENTESE
0001  8B EC                            MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

0003  8B 76 16                         MOV     SI,[BP]+22      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
0006  8B 14                            MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
0008  83 FA 00                         CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                                                ;VONATKOZO INFORMACIOT OLVASSA BE
000B  75 11                            JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

                                ;A NEGY NYOMOGOMB MITAVETELEZESENEK KODJA
000D  B4 84                             MOV     AH,84H          ;BOTKORMANY INTERFACE
000F  BA 0000                           MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                                                ;BEKERESEHEZ
0012  CD 15                             INT     15H             ;A FELADAT ELVEGZESE
0014  B1 04                             MOV     CL,04           ;ROTALAS 4 BITTEL
0016  D2 C8                             ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
0018  25 000F                           AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                                                ;BITJET TARTJUK MEG
001B  EB 08 90                          JMP     SEND            ;AZ INFORMACIO ATADASA

                                ;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
                                ;KODJA

001E  B4 84                     POTS:   MOV     AH,84H          ;BOTKORMANY INTERFACE
0020  BA 0001                           MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
0023  CD 15                             INT     15H             ;BEOLVASASA

                                ;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

0025  8B 7E 12                 SEND:   MOV     DI,[BP]+18      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
0028  89 05                             MOV     [DI],AX
002A  8B 7E 0E                          MOV     DI,[BP]+14      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
002D  89 1D                             MOV     [DI],BX
002F  8B 7E 0A                          MOV     DI,[BP]+10      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
0032  89 0D                             MOV     [DI],CX
0034  8B 7E 06                          MOV     DI,[BP]+6       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
0037  89 15                             MOV     [DI],DX

0039  8B E5                             MOV     SP,BP           ;VISSZATERESI PARAMETER
003B  5D                                POP     BP
003C  CA 000A                           RET     10

003F                            GAME    ENDP                    ;A GAME ELNEVEZESU ELJARAS VEGE
003F                            MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                        END                     ;A PROGRAM VEGE
```

```
Segments and groups:

                N a m e                 Size    align   combine class

MYCODE . . . . . .              003F    PARA    NONE    'CODE'

Symbols:

                N a m e                 Type    Value   Attr

GAME . . . . . .                F PROC  0000    MYCODE  Global  Length
                                =003F
POTS . . . . . .                L NEAR  001E    MYCODE
SEND . . . . . .                L NEAR  0025    MYCODE

Warning Severe
Errors  Errors
0       0
```

10–2. ábra Az APL hívóprogramhoz csatlakoztatott gépi kód .LST listája

Hexadecimális	Decimális	
83	131	
FA	250	
00	0	
75	117	
11	17	
B4	180	
84	132	
BA	186	
0000	0	
	0	
CD	205	
15	21	
B1	177	
04	4	
D2	210	
C8	200	
25	37	
000F	15	(Az alsó byte-ot először)
	0	
EB	235	
08	8	
90	144	
B4	180	
84	132	
BA	186	
0001	1	(Az alsó byte-ot először)
	0	
CD	205	
15	21	
(CB)	203	

10–1. táblázat A játékadaptert lekérdező program gépi kódú megfelelője

vényhez való csatlakoztatása látható. Ez az egyszerű APL függvény a gépi kód decimális megfelelőjét a CODE elnevezésű tömbbe tárolja. Ahhoz, hogy a programot futtatni tudjuk, egy másik APL függvényt is el kell készítenünk, ami a 10–4. ábrán látható.

```
      ∇LOADGAME[0]∇
[0]   LOADGAME;A;B;C;DIO
[1]    DIO←1
[2]    ⍝ MACHINE LANGUAGE PROGRAM FOR READING GAME ADAPTER PORT
[3]    A←131 250 0 117 17 180 132 186 0 0 205 21 177 4
[4]    B←210 200 37 15 0 235 8 144 180 132 186 1 0 205 21 203
[5]    C←A,B
[6]    CODE←DAV[0IO+C]
```

10–3. ábra A LOADGAME elnevezésű APL program

```
      ∇READGAME[0]∇
[0]   READGAME;DIO
[1]    DIO←1
[2]    LOADGAME
[3]    ⍝EXECUTE THE PROGRAM STORED IN SYSTEM MEMORY
[4]    ⍝TO READ POTS,SET VECTORE TO 0 0 01
[5]    ⍝TO READ PUSHBUTTONS(RETURNED IN AX ONLY)SET VECTOR TO 00 00
[6]   REPEAT:R←0 0 01 0CALL CODE
[7]     AX←R[1]
[8]     AY←R[2]
[9]     BX←R[3]
[10]    BY←R[4]
[11]   ⍝SHOW A CONTINUOUS READING FOR PUSHBUTTONS OR JOYSTICK A AND B
[12]    D←AX,AY,BX,BY
[13]   →REPEAT
```

10–4. ábra A READGAME elnevezésű APL program

A példánkban a DX regisztert 1-gyel töltjük fel, mivel a potenciométerek értékét akarjuk beolvasni. A műveletet a következő kód végzi el:

```
REPEAT:R← 0001 □CALL CODE
```

Ezután a négy érték az R tömbből az AX, BX, CX és DX változókba kerül vissza. Ha a nyomógombok adatait akarjuk megtudni, akkor az előbbi kódon a következő módosítást kell végrehajtanunk:

```
REPEAT:R← 0000 □CALL CODE
```

A program futtatásával a négy ellenállásérték folyamatos beolvasását és kijelzését valósítjuk meg. A Ctrl/Exit billentyűk lenyomásával a program leállítható. (A gépi kódú programok APL nyelvhez csatlakoztatásáról az STSC APL kézikönyvben részletesen olvashatunk.)

10.2 ASSEMBLY-RUTIN CSATLAKOZTATÁSA A TURBO PASCAL NYELVHEZ

A PASCAL nyelv nemcsak a gyakorlatban, hanem a számítástechnika oktatásában is nagyon elterjedt. A BORLAND cég által készített PASCAL interpretációt nemcsak teljes képernyős szövegszerkesztővel látták el, de kívánságra 80287-es és BCD matematikai támogatással is szállítják. A fordítóprogram olyan környezetet teremt, mintha egy interpreter típusú nyelvvel dolgoznánk.

A most bemutatásra kerülő PASCAL program egy billentyű lenyomásáig folyamatosan beolvassa a játékadaptert. A 10–5. ábrán látható assembly rutint olyan kóddal láttuk el, ami a két program közötti információcserét segíti.

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA


MYCODE  SEGMENT
        ASSUME  CS:MYCODE
GAME    PROC    NEAR            ;AZ ELJARAS NEVE GAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+20      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
        MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                ;BEKERESEHEZ
        INT     15H             ;A FELADAT ELVEGZESE
        MOV     CL,04           ;ROTALAS 4 BITTEL
        ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
        AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                ;BITJET TARTJUK MEG
        JMP     SEND            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
POTS:   MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
        INT     15H             ;BEOLVASASA

;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

SEND:   MOV     DI,[BP]+16      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
```

```
        MOV     DI,[BP]+8       ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+4       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETER
        POP     BP
        RET     10

GAME    ENDP                    ;A GAME ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     GAME            ;A PROGRAM VEGE
```

10–5. ábra Assembly program csatlakoztatása Turbo Pascal hívóprogramhoz

Ezt a kódrészletet külön is kiemeljük:

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA


MYCODE  SEGMENT
        ASSUME  CS:MYCODE
GAME    PROC    NEAR            ;AZ ELJARAS NEVE GAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+20      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE

;-->A JATEKADAPTERRE VONATKOZO KODOT MOST INNEN ELTAVOLITOTTUK<--

;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

SEND:   MOV     DI,[BP]+16      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+8       ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+4       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETEREK
        POP     BP
        RET     10
```

Az assembly rutint fordítanunk és szerkesztenünk kell, majd pedig .COM file-ba konvertálnunk. Emiatt a TURBO PASCAL programoknak csak egyetlen szegmensük lehet. A GAME elnevezésű eljárást NEAR-nak deklaráltuk. BP-t tároltuk a verembe annak érdekében, hogy SP értékét BP-nek átadhassuk. Ha az assembly rutinokat magas szintű nyelven írt programokhoz csatlakoztatjuk, rendszerint a verem az a hely, amint keresztül az információkat átadjuk, ill. ahonnan átvesszük. A hívóprogramból az assembly programba az SI regiszteren keresztül juttathatunk el értékeket, míg az assembly rutinból a hívó rutinba a DI regiszteren keresztül.

A jelenlegi esetben öt – szó méretűnek definiált – változó kölcsönös átadását oldjuk meg. Mivel ez egy NEAR típusú eljárás, a visszatérési cím a verem első négy byte-ját foglalja el. A hívóprogram BY változójának címe [BP] + 4. Ez a cím kerül a DI regiszterbe. A DX-ben tárolt egész érték ezután a MOV [DI],DX utasítással jut a megfelelő címre. Négy byte-tal arrébb a következő változó címe található és így tovább. Az assembly program tehát egy változót, a hívóprogram pedig négy egész értéket kap. Amennyiben DX-et a rutin meghívásakor 1-re állítjuk be, ez a négy érték a botkormány potenciométerének értékeit jelenti. Ha DX-nek 0 értéket adunk 1 helyett, akkor csak az AX regiszternek van értelme, ez tartalmazza ui. a nyomógombokra vonatkozó kódolt információt.

A [BP] + 20-nál levő változón keresztül történik a hívóprogram és az assembly rutin közötti paraméterátadás. Ily módon állítható be a DX regiszter 1-re vagy 0-ra. A [BP] + 20-nál található címet az SI regiszter kapja meg, az SI hely tartalmát pedig a DX regiszter. Ezzel a módszerrel a hívóprogram és az assembly rutin között tetszőleges számú változó átadása valósítható meg mindkét irányban. Mielőtt a rutinból kilépnénk, a BP regiszter tartalmát vissza kell állítani, és egy NEAR típusú RET utasítást kell kiadni. Az operandusok oszlopában álló 10 hatására a program törli a veremnek azt a 10 byte-ját, amit korábban oda ráhelyeztünk.

```
VAR
   OPER,AX,AY,BX,BY:INTEGER;

PROCEDURE REP(VAR OPER,AX,AY,BX,BY:INTEGER); EXTERNAL 'B:TGAME.COM';

PROCEDURE INFORM;
BEGIN
  OPER:=1;
  REP(OPER,AX,AY,BX,BY);
  WRITELN(AX,'    ',AY,'    ',BX,'    ',BY,'    ');
END;

BEGIN  {EZ A FOELJARAS}
WHILE NOT KEYPRESSED DO
    INFORM
END.
```

10–6. ábra Turbo Pascal program, ami meghívja a játékadapter portját beolvasó gépi kódú programot

A 10–6. ábrán maga a TURBO PASCAL program látható.

Az assembly programot – amit a B meghajtón helyeztünk el és a TGAME.COM elnevezést viseli –, mint külső (external) eljárást hívjuk meg. Egy másik érdekesség a változók átadásának sorrendje. Az OPER, AX, AY, BX és BY változókat egésznek deklaráltuk. Az OPER változón keresztül adjuk át az 1 vagy a 0 értéket az assembly program DX regiszterébe. Az OPER változó címe [BP] + 20. Az AX, AY, BX, BY értékek az assembly programtól származnak. A BY regiszter címe [BP] + 4.

Az assembly programot fordítanunk és szerkesztenünk kell, valamint az EXE2BIN kiszolgálóprogrammal .COM file-t kell létrehoznunk. A fordításhoz és szerkesztéshez tartozó képernyő output a következő:

```
B>C:MASM TGAME;
IBM Personal Computer Macro Assembler      Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984
```

```
  50096 Bytes free

  Warning Severe
  Errors Errors
  0       0
  B>C:LINK TGAME,TGAME,NUL,;
  IBM Personal Computer Linker
  Version 2.30 (C) Copyright IBM Corp. 1981, 1985
  Warning: no stack segment§0 §xx B:§0 §xx B>C:EXE2BIN TGAME TGAME.COM
```

Ezután a TURBO szövegszerkesztő programja segítségével be kell írnunk a 9–6. ábrán látható PASCAL programot. Ha ezzel elkészültünk, lépjünk ki a szövegszerkesztőből, és hozzuk létre belőle a .COM file-t.

Most már csak a TURBO PASCAL-ból való kilépés és futtatás van hátra. Mivel a PASCAL fordítóprogram .COM file-t hoz létre, a program maga igen rövid. Még egy utolsó megjegyzés: Mind a PASCAL, mind az assembly file-oknak futási időben kell elérhetőnek lenniük, ui. ezeket a file-okat nem szerkesztettük össze, így nem alkotnak egyetlen önálló programot.

10.3 ASSEMBLY-RUTIN CSATLAKOZTATÁSA A BASIC PROGRAMHOZ

A MICROSOFT cég QuickBASIC fordítóprogramja egyike a szoftverpiacon előforduló leghatékonyabb BASIC fordítóprogramoknak. A QuickBASIC-ben minden olyan utasítás szerepel, amit az interpreter típusú BASIC-ek támogatnak, beleértve a grafikus szolgáltatásokat is. A QuickBASIC ideális BASIC fordítóprogram az alkalmilag programozó felhasználók számára. Sok szempontból az ennél sokkal drágább fordítóprogramok riválisának tekinthető. A most bemutatásra kerülő interface technika az IBM PC számítógépcsalád legtöbb BASIC fordítóprogramjával működik. Annak érdekében, hogy a példa minél egyszerűbb legyen, a BASIC program az assembly programot ismételten mindig meghívja. Az output leállítása a Ctrl/Alt/Del billentyűk lenyomásával lehetséges. Az assembly rutin tartalmazza azt a kódot, amivel a két program közötti információcsere lehetővé válik. A 10–7. ábrán látható a program teljes listája.

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA

        PUBLIC  BGAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
BGAME   PROC    FAR             ;AZ ELJARAS NEVE BGAME
        PUSH    DS
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+16      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
```

```
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
        MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                ;BEKERESEHEZ
        INT     15H             ;A FELADAT ELVEGZESE
        MOV     CL,04           ;ROTALAS 4 BITTEL
        ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
        AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                ;BITJET TARTJUK MEG
        JMP     SEND            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
POTS:   MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
        INT     15H             ;BEOLVASASA

;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

SEND:   MOV     DI,[BP]+14      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+8       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETER
        POP     BP
        POP     DS
        RET     10

BGAME   ENDP                    ;A BGAME ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     BGAME           ;A PROGRAM VEGE
```

10–7. ábra Assembly kód csatlakoztatása QuikBASIC hívóprogramhoz

Az információcserét támogató kódrészletet külön kiemeljük:

```
        PUBLIC  BGAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
BGAME   PROC    FAR             ;AZ ELJARAS NEVE GAME
        PUSH    DS
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+16      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;-->ITT KELL ELHELYEZNI A JATEKADAPTERRE VONATKOZO KODOT <--
```

```
SEND:   MOV     DI,[BP]+14      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+8       ;AZ B(Y) ERTEK ATADASANAK ELOKÉSZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETER
        POP     BP
        POP     DS
        RET     10
```

Ezt az assembly rutint le kell fordítani és össze kell szerkeszteni a már lefordított BASIC programmal. Azok az assembly rutinok, amelyeket BASIC programokkal összeszerkesztünk, egy szegmensnél többet is tartalmazhatnak. A BASIC-ből meghívott eljárást PUBLIC-nak kell deklarálnunk annak érdekében, hogy az eljárás és a program között a változók átadása lehetővé váljék. A BGAMBE eljárást FAR-nak kell deklarálnunk, amihez természetesen egy FAR típusú RET is tartozik majd. Mivel BP tartalmát a veremben tároljuk, SP értéke elhelyezhető BP-ben. Ha assembly rutinokat magas szintű nyelven írt programokhoz csatlakoztatunk, rendszerint a verem az a hely, amin keresztül az információkat átadjuk, ill. ahonnan átvesszük. A hívóprogramból az assembly programba az SI regiszteren keresztül juttathatunk el értékeket, míg az assembly rutinból a hívó rutinba a DI regiszteren keresztül. A jelenlegi esetben öt változó – amit szó méretűnek deklaráltunk – kölcsönös átadását oldjuk meg. A hívóprogramba történő visszatéréshez szükséges információ a verem első nyolc byte-ját foglalja el. A hívóprogram BY változójának címe [BP] + 8. Ez a cím kerül a DI regiszterbe. A DX-ben szereplő egész érték ezután a MOV [DI],DX utasítás adja át a a megfelelő címre. Két byte-tal arrébb található a következő változó memóriacíme és így tovább. Az assembly program tehát egy változót, a hívóprogram pedig négy egész értéket kap. Amennyiben DX-et a rutin meghívásakor 1-re állítjuk be, ez a négy érték a botkormány pontenciométerének értékeit jelenti. Ha DX-nek 0 értéket adunk 1 helyett, akkor csak az AX regiszternek van értelme, ui. ez tartalmazza a nyomógombokra vonatkozó kódolt információt.

A [BP] + 16-nál levő változón keresztül valósul meg a hívóprogram és az assembly rutin közötti paraméterátadás. Ilyen módon állítható be a DX regiszter 1-re vagy 0-ra. A [BP] + 16-nál található címet az SI regiszter kapja meg, SI hely tartalmát pedig a DX regiszter. Ezzel a módszerrel a hívóprogram és az assembly rutin között tetszőleges számú változó átadása valósítható meg mindkét irányban. Mielőtt a rutinból kilépnénk, a BP és a DS regiszterek tartalmát vissza kell állítanunk, és egy FAR típusú RET utasítást kell kiadnunk. Az operandusok oszlopában álló 10 hatására a program törli a veremnek azt a 10 byte-ját, amit korábban oda ráhelyeztünk. A 10.8. ábrán az eljárást meghívó BASIC programot láthatjuk. Az assembly programot, ami a BGAME.ASM elnevezést viseli, mint külső eljárást hívjuk meg, és az .OBJ file-lal együtt a B meghajtón helyezzük el. Figyeljük meg a hívóprogramban a változók átadásának sorrendjét. Az OPER%, AX%, AY%, BX%, BY% változókat egésznek deklaráltuk. Az OPER% változó, aminek a címe [BP] + 16, az 1 vagy a 0 átadását biztosítja. Az AX%, AY%, BX%, BY% változók értékeit az assembly programtól kapjuk meg. BY címe [BP] + 8.

```
LET OPER%=1
FOR I=1 TO 200
   CALL BGAME(OPER%,AX%,BX%,CX%,DX%)
   PRINT AX%,BX%,CX%,DX%
NEXT I
END
```

10–8. ábra QuikBASIC program, ami a játékadapter portját lekérdező gépi kódú programot meghívja

Az assembly programot tehát be kell írnunk, és le kell fordítanunk. A képernyőn a következőket látjuk:

```
B>C:MASM BGAME;
IBM Personal Computer Macro Assembler Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984
50096 Bytes free
Warning Severe
Errors Errors
0      0
```

A következő feladat a BASIC program beírása egy szövegszerkesztő segítségével (ilyen pl. a Peter NORTON féle teljes-képernyős szövegszerkesztő). Ha ezzel a feladattal is elkészültünk, le kell fordítanunk a BASIC programot is.

```
C:BASCOM B:MSGAME/O,B:,,;
Microsoft Quick Basic Compiler
Version 1.00

(C) Copyright Microsoft Corp. 1982, 1983, 1984, 1985
49158 Bytes Available
48704 Bytes Free
0 Warning Error(s)
0 Severe Error(s)
```

A /O kapcsolót azért használtuk, hogy a programot a BASRUN file nélkül is tudjuk működtetni. A legutolsó lépés a BASIC és az assembly program összeszerkesztése:

```
C:LINK B:MSBGAME+B:BGAME,B:,,C:;
```

Az utasításból látható, hogy mind a BASIC program lefordított állapotát tartalmazó MSBGAME.OBJ file, mind a BGAME.OBJ assembly program a B meghajtón található. Ha az assembly programot hozzászerkesztjük a hívóprogramhoz, a továbbiakban erre a programra már nem lesz szükségünk a lemezen. Az MSBGAME.EXE teljes, és önállóan működőképes program.

10.4 ASSEMBLY-RUTIN CSATLAKOZTATÁSA C NYELVŰ PROGRAMHOZ

A C-t eredetileg a nagyszámítógépek operációs rendszerének kialakításához használták. Később már adatbáziskezelő nyelvet és fordítóprogramot is készítettek ezen a nyelven. A C nyelvet a magas szintű nyelvek és az assembly kód között lehet elhelyezni. A MICROSOFT cég C fordítóprogramja a 3.0 változattól kezdve a kisszámítógépes rendszereknél is elterjedt. A MICROSOFT cég a C nyelv felhasználásával teljesen átdolgozta az IBM számítógépcsaládhoz készített makroassemblerét. A 4.0 verziójú makroassembler és az ezt követő implementációk majdnem háromszor olyan gyorsan futnak, mint a MICROSOFT Makroassembler 3.0 és az IBM Makroassembler 2.0. Ez a rugalmasság és gyorsaság a jól elkészített C program jellemzője.

Jelen esetben egy C nyelven elkészített programhoz csatlakoztatjuk a játékadaptert beolvasó rutint. A hívóprogrammal az eredmények képernyőre való kijelzését is megoldjuk. A program a Ctrl/Alt/Del billentyűk lenyomásáig fut. Az assembly program tartalmazza a két program közötti információcserét. A 10–9. ábrán látható a teljes assembly kód.

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA

        public  _cgame          ;LENYEGES "C" DEKLARACIOK
_text   segment byte public 'code'
        assume  cs:_text
_cgame  PROC    NEAR            ;AZ ELJARAS NEVE _cgame
        push    bp              ;A BP REGISZTER ELMENTESE
        mov     bp,sp           ;A VEREMMUTATO ERTEKE BP-BE KERUL
        push    di              ;
        push    si

        mov     si,[bp+4]       ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        mov     dx,[si]         ;AZ INFORMACIO ELMENTESE DX-BE
        cmp     dx,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        jne     pots            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
        mov     ah,84H          ;BOTKORMANY INTERFACE
        mov     dx,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                ;BEKERESEHEZ
        int     15H             ;A FELADAT ELVEGZESE
        mov     cl,04           ;ROTALAS 4 BITTEL
        ror     al,cl           ;A 7-4-BOL A 3-0 BITPOZICIOBA
        and     ax,0fH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                ;BITJET TARTJUK MEG
        jmp     send            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
pots:   mov     ah,84H          ;BOTKORMANY INTERFACE
        mov     dx,01H          ;A POTENCIOMETEREK ERTEKENEK
        int     15H             ;BEOLVASASA
```

```
;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

send:   mov     di,[bp+6]       ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],ax
        mov     di,[bp+8]       ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],bx
        mov     di,[bp+10]      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],cx
        mov     di,[bp+12]      ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],dx

        pop     si
        pop     di
        mov     sp,bp           ;VISSZATERESI PARAMETER
        pop     bp
        ret

_cgame  endp                    ;A GAME ELNEVEZESU ELJARAS VEGE
_text   ends                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        end     _cgame          ;A PROGRAM VEGE
```

10–9. ábra Assembly kód csatlakoztatása C nyelvű hívóprogramhoz

A következő kódrészletben a C programot támogató programrészletet emeltük ki.

```
        public  _cgame          ;LENYEGES "C" DEKLARACIOK
_text   segment byte public 'code'
        assume  cs:_text
_cgame  PROC    NEAR            ;AZ ELJARAS NEVE _cgame
        push    bp              ;A BP REGISZTER ELMENTESE
        mov     bp,sp           ;A VEREMMUTATO ERTEKE BP-BE KERUL
        push    di              ;
        push    si

        mov     si,[bp+4]       ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        mov     dx,[si]         ;AZ INFORMACIO ELMENTESE DX-BE
        cmp     dx,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        jne     pots            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;-->ITT KELL ELHELYZNI A JATEKADAPTERRE VONATKOZO KODOT <--

send:   mov     di,[bp+6]       ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],ax
        mov     di,[bp+8]       ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],bx
        mov     di,[bp+10]      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],cx
        mov     di,[bp+12]      ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        mov     [di],dx

        pop     si
        pop     di
        mov     sp,bp           ;VISSZATERESI PARAMETER
        pop     bp
        ret
```

A programban kisbetűket kell használnunk, ha az alkalmazott C fordítóprogram erre érzékeny. A szegmensek és az eljárások aláhúzás karakterrel kezdődnek. A cgame eljárást NEAR-nek és public-nak deklaráltuk és a NEAR RET utasítással fejeztük be. Az assembly program tehát egy változót, a hívóprogram pedig 4 egész értéket kap. Amennyiben DX-et a rutin meghívásakor 1-re állítjuk be, ez a négy érték a botkormány pontenciométerének értékeit jelenti. Ha DX-nek 0 értéket adunk 1 helyett, akkor csak az AX regiszternek van értelme, ui. ez tartalmazza a nyomógombokra vonatkozó kódolt információt. Az az információ, amit a hívóprogramnak vissza kell adnunk, a verem első négy byte-ját foglalja el. A hívóprogram &oper változójának címe [BP] + 4. Ezt a címet kapja meg az SI regiszter. Az [SI] címnél található egész érték ezután a dx regiszterbe kerül a mov dx, [si] utasítás segítségével. A következő két byte (szó) a következő változó (&ax) memóriacímét adja, és így tovább. Ezzel a módszerrel a hívóprogramból egy változót adunk át az assembly programba, a hívóprogram pedig négy változót kap vissza. Ez a négy egész érték a botkormány potenciométerének beolvasását jelenti, feltételezve azt, hogy DX-et 1-re állítottuk be. Ha DX-nek a 0 értéket adtuk, csak az AX regiszternek van jelentése, ui. ez tartalmazza a nyomógombokra vonatkozó kódolt információt. Ezzel a módszerrel tetszőleges számú változó értékének átadása is megoldható. Mielőtt a rutinból kilépnénk, az SI, BI és BP regiszterek eredeti tartalmát vissza kell állítanunk, és kiadnunk egy NEAR típusú, RET utasítást.

A 10–10. ábrán a C nyelvű hívóprogram látható.

```
int  oper, ax, ay, bx, by;

main()
{
        extern cgame();
        oper = 1;
repeat: cgame(&oper,&ax,&ay,&bx,&by);
        printf("%d %d %d %d\n", ax, ay, bx, by);
        goto repeat;
}
```

10–10. ábra Az assembly rutint meghívó C nyelvű program

Az assembly programot mint külső eljárást hívtuk meg. Figyeljük meg, hogy híváskor az aláhúzás karakter a CGAME előtt nem szerepel. Az assembly program neve CGAME.ASM, amit a lefordított .OBJ file-lal együtt a B meghajtón helyeztünk el. A hívóprogramban az egésznek deklarált oper%, ax%, ay%, bx%, by% változók átadásának sorrendje is érdekes. Az oper% változón keresztül helyezhetjük el az 1 vagy a 0 értéket az assembly program dx regiszterébe. Az oper% címe [BP] + 4. Az ax%, ay%, bx%, by% értékét az assembly program határozza meg. A by változó címe [BP] + 12. Most is az assembly program beírása az első feladatunk (pl. NORTON teljes képernyős szövegszerkesztő programjával). A következő lépés a program lefordítása, aminek a képernyőképét bemutatjuk:

```
B>C:MASM CGAME;
IBM Personal Computer Macro Assembler Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984
50096 Bytes free
Warning Severe
Errors Errors
0       0
```

Ez az eljárás hozza létre az .OBJ file-t a B meghajtón. A következő feladat a C program beírása, és egy .OBJ file létrehozása a C fordítóprogrammal. (Az általunk használt C fordítóprogramot a Microsoft cég készítette.)

```
C>MSC B:MSCGAME,B:,/G2/FPc87;
```

Az utolsó művelet a két .OBJ file összeszerkesztése a rendszer szerkesztőprogramjával. (Az általunk használt szerkesztőprogramot a Microsoft cég készítette. Verziószáma: 3.01.)

```
C>LINK B:MSCGAME+B:CGAME,B:,,C:;
```

A jelenlegi esetben mind az MSCGAME.OBJ, ami a C nyelvű program lefordított változata, mind a CGAME.OBJ, ami az assembly program lefordított változata, a B meghajtón helyezkedik el. Ezt a két programot a (+) opcióval tudjuk összeszerkeszteni. Amint ezzel a művelettel elkészülünk, az MSCGAME.EXE program önállóvá válik, tehát az assembly programra nincs már szükség a lemezen.

10.5 ASSEMBLY-RUTIN CSATLAKOZTATÁSA FORTRAN PROGRAMHOZ

Sok programozó legelőször a FORTRAN nyelvet sajátítja el. Bár manapság az ADA, a PASCAL és a C nyelveket nevezik a legjobb programozási nyelveknek, a FORTRAN továbbra is népszerű maradt. Az IBM cég FORTRAN fordítóprogramja a kis méretű számítógépes környezetek szabványos FORTRAN fordítóprogramjának tekinthető. Annak érdekében, hogy a mintapéldát minél egyszerűbbé tegyük, a FORTRAN programmal mindig ismételten meghívjuk a gépi kódú programot. Az output leállítása a Ctrl/Alt/Del billentyűk lenyomásával lehetséges. A gépi kódú program tartalmazza azt a támogató kódot, amivel a két program közötti információcsere megoldható. A 10–11. ábrán az assembly program teljes listáját látjuk.

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA

        PUBLIC  FGAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
FGAME   PROC    FAR             ;AZ ELJARAS NEVE FGAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+22      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
        MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                ;BEKERESEHEZ
```

```
        INT     15H             ;A FELADAT ELVEGZESE
        MOV     CL,04           ;ROTALAS 4 BITTEL
        ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
        AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                ;BITJET TARTJUK MEG
        JMP     SEND            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
POTS:   MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
        INT     15H             ;BEOLVASASA

;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

SEND:   MOV     DI,[BP]+18      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+14      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETER
        POP     BP
        RET     10

FGAME   ENDP                    ;A FGAME ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     FGAME           ;A PROGRAM VEGE
```

10–11. ábra Assembly kód csatlakoztatása FORTRAN hívóprogramhoz

A programok kapcsolódását biztosító kódot külön kiemeljük:

```
        PUBLIC  FGAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
FGAME   PROC    FAR             ;AZ ELJARAS NEVE FGAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+22      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;-->ITT KELL ELHELYEZENI A JATEKADAPTERRE VONATKOZO KODOT <--

SEND:   MOV     DI,[BP]+18      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+14      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
```

```
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETER
        POP     BP
        RET     10
```

Az assembly kód lefordított változatát a FORTRAN kód lefordított változatához kapcsoljuk. Az FGAME eljárást FAR-nak és PUBLIC-nak deklaráljuk, emiatt a FGAME eljárást egy FAR típusú RET utasítással kell lezárnunk. BP-t a verembe tároltuk annak érdekében, hogy SP értékét BP-nek átadhassuk. Ha az assembly rutinokat magas szintű nyelven írt programokhoz csatlakoztatjuk, rendszerint a verem az a hely, amin keresztül az információkat átadjuk, ill. ahonnan átvesszük. A hívóprogramból az assembly programba az SI regiszteren keresztül juttathatunk el értékcket, míg az assembly rutinból a hívó rutinba a DI regiszteren keresztül. A jelenlegi esetben öt – szó méretűnek definiált – változó kölcsönös átadását oldjuk meg. A BY változó címe [BP] + 6. Ez a cím kerül a DI regiszterbe. A DX-ben szereplő egész értéket a MOV [DI],DX utasítás a [DI] címre helyezi. A következő változó 4 byte-tal arrébb a [BP] + 10 memóriahelyen található. Az assembly program tehát egy változót, a hívóprogram pedig négy egész értéket kap. Amennyiben DX-et a rutin meghívásakor 1-re állítjuk be, ez a négy érték a botkormány pontenciométerének értékeit jelenti. Ha DX-nek zérus értéket adunk 1 helyett, akkor csak az AX regiszter használatának van értelme, ez tartalmazza ui. a nyomógombokra vonatkozó kódolt információt.

Mielőtt a rutinból kilépnénk, a BP regiszter tartalmát vissza kell állítani, és egy FAR típusú RET-et kell kiadni. A RET utasítás mellett álló 10-es szám a változók veremből történő eltávolítását szolgálja.

A 10–12. ábrán az assembly kód – mint külső eljárás – hívását biztosító FORTRAN program látható.

```
      INTEGER  OPER,AX,AY,BX,BY
      OPER=1
      DO 3 I=1,500
1     CALL FGAME(OPER,AX,AY,BX,BY)
      WRITE(*,2)AX,AY,BX,BY
2     FORMAT  (4I5)
3     CONTINUE
      END
```

10–12. ábra A játékadaptert lekérdező gépi kódú programot meghívó FORTRAN program

Az FGAME.ASM assembly program és a lefordított változatát tartalmazó .OBJ file egyaránt a B meghajtón helyezkedik el. Érdemes felfigyelnünk a hívóprogramban a változók átadásának sorrendjére is. Az OPER, AX, AY, BX, BY változókat egésznek deklaráltuk. Az OPER változóval adjuk át az 1 vagy a 0 értéket a DX regiszternek az assembly programba. Az OPER változó címe [BP] + 22. Az AX, BY, BX, BY változók értékeit az assembly program állítja be. A BY változó címe [BP] + 6.

Először az assembly programot írjuk be (pl. a NORTON teljes képernyős szövegszerkesztő programmal). A fordítási művelet a következő lépés, amelyhez a képernyőn megjelenő output bemutatásával nyújtunk segítséget:

```
B>C:MASM FGAME;
IBM Personal Computer Macro Assembler Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984
```

```
50096 Bytes free
Warning Severe
Errors Errors
0       0
```

Ezután a FORTRAN programot kell beírnunk, majd a FORTRAN fordítóprogrammal a fordítást elkészítenünk. Az .OBJ file létrehozásának menete a következő:

```
A>FOR1 B:MSFGAME,B:MSFGAME,NUL,NUL;
IBM Personal Computer FORTRAN Compiler
Version 2.00
(C)Copyright IBM Corp 1982, 1984
(C)Copyright Microsoft Corp 1982, 1984
Pass One No errors Detected
8 Source Lines
A>FOR2
Code Area Size =
00D1 (  209)
Code Area Size =
000C (   12)
Code Area Size =
0021 (   33)

Pass Two No Errors Detected
```

Ha a két lépésből álló fordítási eljárással elkészültünk, a FORTRAN program .OBJ file-ja a B meghajtón helyezkedik el.

Az utolsó lépés a két .OBJ file összeszerkesztése.

```
A>LINK B:MSFGAME+B:FGAME,B:,,A:;
```

Az utasításból látható, hogy mind a FORTRAN program lefordított állapotát tartalmazó MSFGAME.OBJ file, mind a FGAME.OBJ assembly program a B meghajtón található. Ha az assembly programot a (+) opcióval hozzászerkesztjük a hívóprogramhoz, a továbbiakban erre a programra már nem lesz szükségünk a lemezen. Az MSFGAME.EXE teljes és önállóan működőképes program.

10.6 ASSEMBLY-RUTIN CSATLAKOZTATÁSA PASCAL PROGRAMHOZ

Az IBM PASCAL fordítóprogramnak nincs annyi előnyös tulajdonsága, mint a BORLAND cég TURBO PASCAL fordítóprogramjának. Az IBM PASCAL-nak nincs saját szövegszerkesztője, a fordítás ideje jóval hosszabb, mint a TURBO PASCAL-é, és ráadásul még az ára is magasabb. Csak igazán komoly és nagyméretű feladatok esetén ajánljuk a használatát. A legutolsó PASCAL verzió a 80287-es társprocesszort támogatja. Az IBM PASCAL fordítóprogram az előző példákhoz hasonlóan olyan .OBJ file-okat hoz létre, amelyek a több szegmensből álló assembler programokhoz hozzászerkeszthetők.

A most bemutatásra kerülő példa egy billentyű lenyomásáig folyamatosan lekérdezi a játékadaptert. A 10–13. ábrán látható assembly rutint egy olyan kóddal láttuk el, ami a két program közötti információcserét segíti.

```
;80286/80386-OS SZAMITOGEPRE KESZITETT PROGRAM
;A PROGRAM EGY BIOS MEGSZAKITASSAL BEKERI AZ A ES B BOTKORMANY
;PORTJANAK  ERTEKEIT, ES A HIVOPROGRAMNAK ATADJA

        PUBLIC  PGAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
PGAME   PROC    FAR             ;AZ ELJARAS NEVE PGAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+14      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;A NEGY NYOMOGOMB MINTAVETELEZESENEK KODJA
        MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,0            ;PARAMETERBEALLITAS AZ INFORMACIO
                                ;BEKERESEHEZ
        INT     15H             ;A FELADAT ELVEGZESE
        MOV     CL,04           ;ROTALAS 4 BITTEL
        ROR     AL,CL           ;A 7-4-BOL A 3-0 BITPOZICIOBA
        AND     AX,0FH          ;AZ INFORMACIONAK CSAK AZ ALSO 4
                                ;BITJET TARTJUK MEG
        JMP     SEND            ;AZ INFORMACIO ATADASA

;A NEGY POTENCIOMETER - A(X),A(Y),B(X),B(Y) - MINTAVETELEZESENEK
;KODJA
POTS:   MOV     AH,84H          ;BOTKORMANY INTERFACE
        MOV     DX,01H          ;A POTENCIOMETEREK ERTEKENEK
        INT     15H             ;BEOLVASASA

;AZ AX, BX, CX, DX REGISZTEREK ERTEKENEK ATADASAT SZOLGALO KOD

SEND:   MOV     DI,[BP]+12      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+10      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+8       ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETEREK
        POP     BP
        RET     10

PGAME   ENDP                    ;A PGAME ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     PGAME           ;A PROGRAM VEGE
```

10–13. ábra Assembly kód csatlakoztatása Pascal hívóprogramhoz

Ezt a kódrészletet külön is kiemeljük:

```
        PUBLIC  PGAME
MYCODE  SEGMENT 'CODE'
        ASSUME  CS:MYCODE
PGAME   PROC    FAR             ;AZ ELJARAS NEVE PGAME
        PUSH    BP              ;A BP REGISZTER ELMENTESE
        MOV     BP,SP           ;A VEREMMUTATO ERTEKE BP-BE KERUL

        MOV     SI,[BP]+14      ;A HIVOPROGRAMTOL SZARMAZO PARAMETER
        MOV     DX,[SI]         ;AZ INFORMACIO ELMENTESE DX-BE
        CMP     DX,0            ;ZERUS ESETEN A PROGRAM A NYOMOGOMBOKRA
                                ;VONATKOZO INFORMACIOT OLVASSA BE
        JNE     POTS            ;HA NEM ZERUS, UGRAS A POTS CIMKERE

;-->ITT KELL ELHELYEZNI A JATEKADAPTERRE VONTAKOZO KODOT <--

SEND:   MOV     DI,[BP]+12      ;AZ A(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],AX
        MOV     DI,[BP]+10      ;AZ A(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],BX
        MOV     DI,[BP]+8       ;AZ B(X) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;AZ B(Y) ERTEK ATADASANAK ELOKESZITESE
        MOV     [DI],DX

        MOV     SP,BP           ;VISSZATERESI PARAMETEREK
        POP     BP
        RET     10
```

Az assembly kód lefordított változatát a PASCAL kódhoz kapcsoljuk hozzá. A PGAME eljárást FAR-nak és PUBLIC-nak deklaráljuk, emiatt a PGAME eljárást egy FAR típusú RET-tel kell majd lezárnunk. BP-t a verembe tároltuk annak érdekében, hogy SP értékét BP-nek átadhassuk. Ha az assembly rutinokat magas szintű nyelven írt programokhoz csatlakoztatjuk, rendszerint a verem az a hely, amin keresztül az információkat átadjuk, ill. ahonnan átvesszük. A hívóprogramból az assembly programba az SI regiszteren keresztül, míg az assembly rutinból a hívó rutinba a DI regiszteren keresztül juttathatunk el értékeket. A jelenlegi esetben öt – szó méretűnek definiált – változó kölcsönös átadását oldjuk meg. A hívóprogramba való visszatéréshez szükséges információ a verem első hat byte-ját foglalja el. A hívóprogram BY változójának címe [BP] + 6. Ez a cím kerül a DI regiszterbe. A DX-ben szereplő egész érték a MOV [DI],DX utasítással kerül a [DI]-vel kijelölt címre. Két byte-tal arrébb a [BP] + 8 memóriacímen található a következő változó (BX) értéke. Ezzel a technikával tetszőleges számú változó közötti információcsere oldható meg.

Mielőtt a rutinból kilépnénk, a BP regiszter értékét vissza kell állítanunk, és egy FAR típusú RET utasítást kiadnunk. A RET utasítást követő 10-es szám hatására a program törli a veremnek azt a 10 byte-ját, amit korábban oda ráhelyeztünk.

A 10–14. ábrán az assembly kód – mint külső eljárás – hívását szolgáló PASCAL program látható.

A PGAME.ASM assembly program és a lefordított változatát tartalmazó .OBJ file egyaránt a B meghajtón vannak. Érdemes arra is felfigyelnünk, hogy a hívóprogramban milyen a változók átadásának sorrendje. Az OPER, AX, AY, BX, BY változókat egészeknek deklaráltuk. Az OPER változóval - aminek a címe [BP] + 4, - adjuk át az 1 vagy 0 értéket a DX regiszterbe. Az AX, AY, BX, BY változók értékeit az assembly program állítja be. A BY változó címe [BP] + 6.

```
PROGRAM MSPGAME(INPUT,OUTPUT);

VAR
   OPER,AX,AY,BX,BY:INTEGER;
   CH:CHAR;

PROCEDURE PGAME(VAR OPER,AX,AY,BX,BY:INTEGER);
  EXTERNAL;

PROCEDURE INFORM;
BEGIN
  OPER:=1;
  PGAME(OPER,AX,AY,BX,BY);
  WRITELN(AX,'    ',AY,'    ',BX,'    ',BY,'    ');
END;

BEGIN  {EZ A FOELJARAS}
    REPEAT
     INFORM;
    UNTIL CH='Q'
END.
```

10–14. ábra A játékadapter portját lekérdező assembly programot meghívó Pascal program

Először az assembly programot írjuk be egy szövegszerkesztővel, majd fordítsuk le. A fordításhoz és a szerkesztéshez tartozó képernyő a következő:

```
B>C:MASM PGAME;
IBM Personal Computer Macro Assembler Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984
50096 Bytes free
Warning Severe
Errors Errors
0       0
```

Ezután a PASCAL programot is készítsük el ugyanezzel a szövegszerkesztővel és fordítsuk le pl. az IBM cég PASCAL fordítóprogramjával. Az .OBJ file létrehozásának menete a következő:

```
A>PAS1 B:MSPGAME,B:,,;
IBM Personal Computer Pascal Compiler
Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1984
Pass One No errors Detected
A>PAS2
Code Area Size =
00D1 (    19)
Code Area Size =
000C (    13)
Code Area Size =
0021 (    36)

Pass Two No Errors Detected.
```

Az utolsó lépés a két .OBJ file összeszerkesztése:

```
A>LINK B:MSPGAME+B:PGAME,B:,,A:;
```

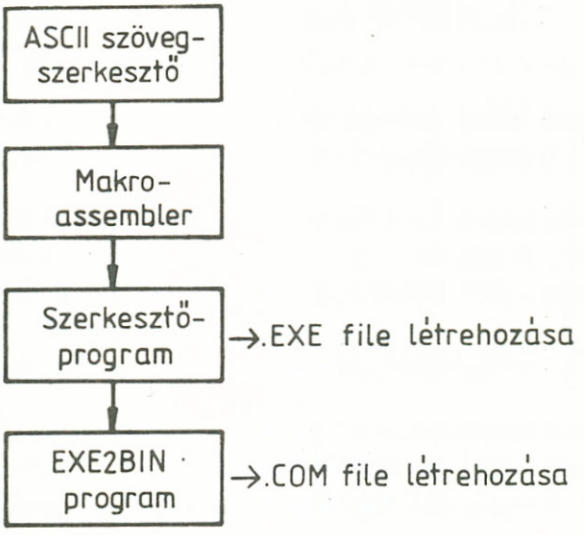
Az utasításból látható, hogy mind a FORTRAN program lefordított állapotát tartalmazó MSPGAME.OBJ file, mind a PGAME.OBJ assembly program a B meghajtón található. Ha az assembly progamot a (+) opcióval hozzászerkesztjük a hívóprogramhoz, a továbbiakban erre a programra már nem lesz szükségünk a lemezen. Az MSPGAME.EXE teljes és önállóan működőképes program.

AZ IBM CÉG MAKROASSEMBLER PROGRAMJA

Ebben a mellékletben az Olvasó az IBM Makroassembler jellemzőivel és használatával ismerkedhet meg, és egy mintaprogram segítségével megtanulhatja, hogyan kell létrehozni, lefordítani és szerkeszteni az assembly programokat (l. az A–1. ábrát).

Azt is megtudhatjuk, hogyan használja fel a forraskódot a fordítóprogram az .OBJ tárgyfile létrehozására, mit kell érteni a listázó és a keresztreferenca file-okon és miért van ezekre szükség. Miután megtanultuk, hogyan kell az .OBJ file-t létrehozni, azt is meglátjuk, hogyan alakítja át a szerkesztőprogram az .OBJ file-okat .EXE file-okká. (Ebben a szakaszban opcionális .MAP file-t is létrehoztunk.)

A mintaprogramon keresztül elmagyarázzuk, hogyan kell ezt a file-t létrehozni és rámutatunk a hasznosságára. A mellékletből még az is kiderül, hogy mi a különbség az .EXE és a .COM file-ok között. Elsajátíthatjuk az EXE2BIN.EXE program használatát is, amivel az .EXE file-okból .COM készíthető.



A–1. ábra Az assembly program létrehozásának lépései

Általános információk

Az IBM Makroassemblert kis és nagy (makro) assemblerrel szállítják. Bár a kis assembler kevesebb helyet foglal, nem támogatja a makroassembler által nyújtott valamennyi opciót és függvényt. Ezért úgy gondoltuk, hogy a makroassembler ismertetésével nagyobb segítséget nyújtunk az Olvasónak.

A makroassembler működéséhez minimálisan 128 K memória és 2.0-s vagy ennél később

készült DOS verzió szükséges. A fordítási eljárás során a makroassembler akár 192 K memóriát is felhasználhat, ha ennyi hely a rendelkezésére áll.

Az assembly forráskód létrehozása

Az assembly nyelvű programfejlesztés első lépéseként mindig a megoldandó probléma pontos körbehatárolását és definiálását kell elvégeznünk. Például elképzelhető az a feladat, hogy a számítógéphez csatlakoztatott játékadapter potenciométer értékeit akarjuk megtudni.

Mielőtt a konkrét programozáshoz hozzákezdenénk, információt kell szereznünk arról, hogyan fér hozzá a számítógép a játékadapterhez. A kérdésre az IBM Műszaki Kézikönyve BIOS listáiban találhatjuk meg a választ.

Ezek után elkészíthetjük a forráskódot. Mint a 2. fejezetben már ismertettük, a forráskódban olyan mnemonikok szerepelnek, amelyek gépi kódú utasításokat jelentenek. A forráskódot szigorú szintaktikai (helyesírási) szabályok szerint kell elkészíteni azért, hogy a fordítóprogram mindent helyesen tudjon értelmezni. A forráskódot bármilyen sor, vagy teljes képernyős szövegszerkesztővel elkészíthetjük, csak az a lényeg, hogy a file ASCII formátumban legyen.

A következőkben bemutatásra kerülő forráskódot, amelynek a GAME.ASM file-nevet adtuk, egy ASCII szövegszerkesztővel (pl. a Norton Editor-ral) készítettünk el. Az .ASM kiterjesztés tudatja a rendszerrel, hogy assembly nyelven készített forráskódról van szó.

```
;80286/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;A PROGRAM BEOLVASSA A JATEKADAPTERT, ES A POT. ERTEKEKET
;A KIJELOLT REGISZTEREKBE ADJA VISSZA

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA


MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

;A JATEKADAPTER POT. ERTEKEINEK BEOLVASASA MEGSZAKITAS
;SEGITSEGEVEL
        MOV     AH,84H          ;A BOTKORMANY MINTAVETELEZESE
        MOV     DX,01H          ;A POT. ERTEKEK MINTAVETELEZESE
        INT     15H             ;A MUVELET VEGREHAJTASA


        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A forráskód elkészítése után a programfejlesztés következő fázisába léphetünk, vagyis a forráskódot a makroassembler felhasználásával átalakítjuk olyan formába, amit a mikroprocesszor megért.

A makroassembler használata

Az IBM Makroassebler az .ASM file-okból (forráskódokból) .OBJ file-okat, vagyis az .ASM file-ok gépi kódú megfelelőit készíti el két menetben.

Az első menetben (fázisban) készül el az ún. relatív helyzet (offset) a forráskód minden egyes sorához, és felépül a szimbólumtáblázat, ami még nem jelenti a tárgykód elkészültét. (L. a A.1. táblázatban a /D paraméterre vonatkozó megjegyzést.) Az első menetben definiált értékekből a második menetben készül el az .OBJ file.

Paraméter	Magyarázat
/A	A paraméter hatására a forráskód szegmenseinek listázása történik alfabetikus sorrendben. A fordítóprogramban ez az alapértelmezés szerinti paraméter.
/D	Fázishiba akkor keletkezik, ha a fordítási eljárás első és második menetében keletkezett értékek között eltérés van. A /D paraméter megadásakor a fordítóprogram az első menetről is listát készít, amelyet összehasonlíthatunk a második menetben automatikusan elkészülő listával.
/E	Ez a paraméter fordítja le a 80287/80387-es processzorokra vonatkozó forráskódot és olyan lebegőpontos konstansokat készít, amelyek a társprocesszor számára szükségesek. Hasonló módon működik, mint az /R paraméter.
/N	Ennek a paraméternek a hatására a fordítóprogram nem készít szimbólumtáblát a listázás végén.
/O	Ennek az opciónak a hatására az assembler elkészíti az assembly program oktális megfelelőjét is.
/R	Ez a paraméter a 80287/80387-es társprocesszor részére vonatkozó forráskódot úgy fordítja le, hogy a numerikus értékek a társprocesszor által elvárt formátumban keletkezzenek. A /R paraméterrel létrehozott gépi kódok a társprocesszor nélküli rendszereken nem futnak megfelelően.
/S	Ez a paraméter felülírja az alapértelmezés szerinti /A paramétert. A szegmensek olyan sorrendben kerülnek listázásra, ahogy a forráskódban megjelennek.

A–1. táblázat Az IBM Makroassembler paraméterei

Fázishibák

A fázishibák a fordítási művelet második menetében keletkeznek akkor, amikor a fordítóprogram valamely változóra, címkére vagy eljárásra más értéket talál, mint amit az első menetben a szimbólumtábla felépítésekor tárolt.

A következőkben felhasznált utasítások azt feltételezik, hogy a
GAME.ASM forrásfile a B meghajtóban, a IBM Makroassebler pedig az

A meghajtóban van, valamint azt, hogy a B meghajtó az alapértelmezés szerinti meghajtó. A>B rendszerüzenetnek kell a képernyőn lennie. A fordítóprogram futtatásához a következőket kell beírnunk:

```
B>A:MASM
```

Ennek hatására a képernyőn egy üzenet jelenik meg:

```
IBM Personal Computer MACRO Assembler Version 2.00
(C) Copyright IBM Corp 1981, 1984
(C) Copyright Microsoft Corp 1981, 1983, 1984
Source filename [.ASM]:
```

A fordítóprogram mindig azt adja meg a szögletes zárójelek között, amit alapértelmezés szerint feltételez. A forrásfile kiterjesztése alapértelmezés szerint .ASM . Mivel a GAME elnevezésű forrásfile-t az .ASM kiterjesztéssel tároltuk, elegendő a **GAME** beírása és az ENTER gomb lenyomása.

Megjegyezzük, hogy a Microsoft Makroassembler lehetővé teszi a fixlemezek használata során előforduló útvonalnevek megadását is. Ilyen eseteken a rendszerüzenetekre adott válaszoknak tartalmazniuk kell a meghajtót és az alkatalógust is.

Az ENTER gomb lenyomása után egy új rendszerüzenetet láthatunk:

```
Object filename [GAME.OBJ]:
```

A fordítóprogram ezzel az üzenettel azt jelzi számunkra, hogy a célfile-nak a GAME.OBJ file-nevet adja, hacsak nem adjuk meg másképpen A file-név elfogadásához az ENTER gombot kell lenyomnunk. Ekkor a harmadik rendszerüzenet jelenik meg:

```
Source listing [NUL.LST]:
```

Az .LST kiterjesztésű file az eredeti forráskód lefordított változata. A forráskód itt annyiban módosul, hogy sorszámozást, az egyes utasítások gépi kódú megfelelőit, és szimbólumtáblát tartalmaz. Az üzenetben a NUL mnemonik jelzi, hogy a fordítóprogram az alapértelmezés szerint nem készíti el ezt a file-t. Ha igénybe akarjuk venni ezt a hasznos szolgáltatást, be kell írnunk a **GAME** nevet és lenyomnunk az ENTER gombot. A fordítóprogram a kiterjesztést automatikusan szolgáltatja, ezért az .LST beírása szükségtelen.

Az ENTER gomb lenyomása után megjelenik a negyedik és egyben utolsó rendszerüzenet is:

```
Cross reference [NUL.CRF]:
```

A fordítóprogram a .CRF kiterjesztésű file-okat, amelyeket kereszreferencia-listának nevezünk, mint közbenső lépést készíti el a fordítási folyamat során. (A .CRF kiterjesztésű file-okat a melléklet későbbi részében még tárgyaljuk.) Az alapértelmezés szerint a program nem készíti el ezt a file-t sem, amit a NUL-lal jelez a számunkra. Ha igényt tartunk erre a file-ra, a **GAME** file-nevet kell beírnunk, majd lenyomnunk az ENTER gombot. Ezek után a fordítóprogram végrehajtja a GAME.ASM forrásfile-on a megadott műveleteket, és elkészíti az .OBJ, az .LST és .CRF file-okat. Kérjünk egy katalógust a B meghajtóról, és győződjünk meg arról, hogy fordítóprogram ténylegesen végrehajtotta a fordítási műveleteket.

A file-ok listája a következőhöz hasonló lesz:

```
GAME  ASM      758          12-27-85   6:47p
GAME  OBJ      57           12-29-85   2:55p
GAME  LST      1656         12-29-85   2:55p
GAME  CRF      69           12-29-85   2:55p
```

Annyit már tudunk, hogy a GAME.ASM tartalmazza a program ASCII kódú változatát, és hogy a GAME.OBJ file nem listázható, mivel ez a programunk gépi kódú megfelelője. A GAME.CRF file sem listázható, mivel ez csak a keresztreferencia-táblázat elkészítéséhez szükséges közbenső lépcső. A GAME.LST file azonban listázható.

Mint már korábban említettük, a .LST kiterjesztésű file-ok a sorszámokkal ellátott forrásprogramot az egyes utasítások gépi kódú megfelelőit és a szimbólumtáblát tartalmazzák. A hardcopy készítés legegyszerűbb módja a **TYPE GAME.LST** paranccsal való listázás. (Ne felejtsük el bekapcsolni a nyomtatót és a parancs beírása után lenyomni a CTRL - PRTSC + ENTER billentyűket.)

A monitor képernyőjén a következő listának kell megjelennie:

```
                                ;80286/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
                                ;A PROGRAM BEOLVASSA A JATEKADAPTERT, ES A POT. ERTEKEKET
                                ;A KIJELOLT REGISZTEREKBE ADJA VISSZA

                                PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA


0000                            MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
0000                            MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
                                        ASSUME  CS:MYCODE
0000  1E                                PUSH    DS              ;A DS REGISZTER ELMENTESE
0001  2B C0                             SUB     AX,AX           ;AX TORLESE
0003  50                                PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

                                ;A JATEKADAPTER POT. ERTEKEINEK BEOLVASASA MEGSZAKITAS
                                ;SEGITSEGEVEL
0004  B4 84                             MOV     AH,84H          ;A BOTKORMANY MINTAVETELEZESE
0006  BA 0001                           MOV     DX,01H          ;A POT. ERTEKEK MINTAVETELEZESE
0009  CD 15                             INT     15H             ;A MUVELET VEGREHAJTASA


000B  CB                                RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
                                MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
000C                            MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                        END                     ;A PROGRAM VEGE

Segments and Groups:

                N a m e                 Size    Align   Combine Class

MYCODE . . . . . .              000C    PARA    NONE    'CODE'

Symbols:

                N a m e                 Type    Value   Attr

MYPROC . . . . . .              F PROC  0000    MYCODE  Length = 000C


    26 Source  Lines
    26 Total   Lines
    24 Symbols
```

```
50524 Bytes symbol space free

    0 Warning Errors
    0 Severe  Errors
```

A–2. ábra Programlista

Paraméterek

A IBM Makroassembler a programok lefordítása során számos választható paramétert (opciót) bocsát a felhasználó rendelkezésére. Az opciókat, amelyeket az A.1. táblázatban soroltunk fel abc sorrendben, a parancssorban kell megadni a / jel után. (Az opciókról részletesen a IBM Makroassembler referencia kézikönyvében olvashatunk.)

Parancsok rövidítése

A fordítóprogram elindításához számos rövidítést használhatunk. Mint már a 2. fejezetben említettük, a

B>A:MASM GAME;

sor beírásával a fordítóprogram közvetlenül elkészíti az .OBJ file-t. Nem generál azonban .LST és .CRF file-okat. A következő parancsrövidítések elfogadják a .LST és .CRF file-okra vonatkozó alapértelmezésként szereplő elnevezést, ezáltal lehetővé teszik, hogy a programozónak ne kelljen mindig válaszolnia a rendszerüzenetekre.

B>A:MASM GAME,,,;

A vesszők letiltják az assemler rendszerüzeneteket, és az adott opciónak a forráskódból következően az alapértelmezés szerinti file-nevet adják. A NUL paranccsal az adott opció letiltható.

A következőkben a parancs-sorban szereplő opcionális paraméter- használatra mutatunk példát:

B>A:MASM GAME/N;

Az előbbi opció hatására a fordítóprogram a listázás végén nem készíti el a szimbólumtáblát.

Keresztreferencia-listázás: CREF.EXE

A keresztreferencia-lista nagyon hasznos a hibakereséskor, ui. az adatok, változók, címkék, konstansok és egyéb szimbólumok alfabetikus listáját készíti el. A lista azt a sorszámot tartalmazza, ahol a szimbólumot először definiáltuk, valamint valamennyi további sort, amelyben a szimbólum szerepel. A keresztreferencia-lista elkészítéséhez a negyedik assembler rendszerüzenetre be kell írnunk a file nevét, amelynek hatására elkészül a filenev.CRF közbenső file. Ezt a file-t a CREF.EXE használja fel az aktuális keresztreferencia-lista elkészítéséhez.

A következő parancsot írjuk be tehát:

B>CREF

majd nyomjuk le az ENTER billentyűt. A képernyőn a következőhöz hasonló felirat jelenik meg:

```
The IBM Personal Computer CREF, Version 2.00
(C) Copyright IBM Corp 1981, 1984
(C) Copyright Microsoft Corp 1981, 1983, 1984
Cross-reference [.CRF]:
```

A CREF.EXE kiszolgálóprogram a .CRF kiterjesztésű file-nevet kéri. A mi példánkban a **GAME** a file neve, tehát ezt kell beírnunk, majd lenyomnunk az ENTER billentyűt. A .CRF kiterjesztést nem szükséges leírnunk, mert a kiszolgálóprogram automatikusan hozzáteszi a file-névhez.

Miután a program a megelőző lépést végrehajtotta, ismét egy üzenettel jelentkezik:

```
List filename [GAME.REF]:
```

A CREF.EXE kiszolgálóprogram alapértelmezésként a GAME.REF programnevet adja az elkészítendő file-nak, amit a felhasználó természetesen felülírhat, ha ettől eltérő meghajtót akar használni. Ha azonban elfogadjuk a program által javasolt file-nevet, csak az ENTER billentyűt kell lenyomnunk.

Rövidítések a CREF.EXE parancs használatakor

Mint a makroassemblernél láttuk, a keresztreferencia-lista elkészítéséhez is használhatunk rövidítéseket. A

```
B>CREF GAME;
```

beírásával a CREF.EXE program automatikusan megkeresi a GAME.CRF file-t és elkészíti ugyanilyen névvel a .REF kiterjesztésű file-t.

Ebben az esetben ennek a file-nak a GAME.REF nevet adja a program. Ha a file hiba nélkül létrejön, az alapértelmezés szerinti meghajtó szimbóluma jelenik meg a képernyőn. Készítsünk a B meghajtóról egy tartalomjegyzéket, hogy lássuk, a file tényleg létrejött. A **TYPE GAME.REF** parancs beírásával a következőhöz hasonló keresztreferencia-lista jelenik majd meg a képernyőn.

```
B>TYPE GAME.REF
Symbol Cross Reference                                  (is definition)
Cref-1

CODE                8

MYCODE              8#                  10      23

MYPROC              9#                  22
  3 Symbols
  63048 Bytes Free
```

A listából megállapítható a programban található szegmensek száma. A mi esetünkben csak egy ilyen szegmens van, a CODE. A MYCODE és MYPROC deklarációk is láthatóak. A # szimbólum, amely egy sorszámot követ, azt jelenti, hogy az adott szimbólum ebben a sorban szerepelt először. A további sorszámok azokra a sorokra utalnak, amelyekben az adott szimbólumot még felhasználtuk.

Még ebből a rövid példából is látható, hogy a keresztreferencia-lista segítségével meggyőződhetünk arról, hogy a kódszegmensek tartalmazzák-e az END utasításokat (pl. a MYCODE a 23. vagy a MYPROC a 22. sorban).

Minél nagyobb az assembly program, annál hasznosabb a keresztreferencia-lista a hibakeresésben. Az előbbi file-ról nagyon egyszerűen készíthető hardcopy, csupán csak be kell kapcsolni a nyomtatót és beírni a **TYPE** után a file nevét a következők szerint:

```
B>TYPE GAME.REF
```

Ezek után a CTRL-PRTSC, majd az ENTER billentyűket kell lenyomnunk. A listázás befejezése után ne feledkezzünk meg a CTRL-PRTSC ismételt lenyomásáról, különben mindent kinyomtat a nyomtató, ami a képernyőn megjelenik.

Akár készítettünk keresztreferencia-listát, akár nem, az assembly proramfejlesztés következő lépése az .OBJ file szerkesztése, amit a LINK.EXE programmal tudunk megvalósítani.

Szerkesztés a LINK.EXE programmal

A LINK.EXE kiszolgálóprogram az ún. áthelyezhető programot készíti el. Ha az .OBJ file-hoz egy kiegészítő fejrészt adunk, a szerkesztőprogram engedélyezi az operációs rendszernek, hogy az assembly programot a rendelkezésre álló memória bármely részére elhelyezze.

Ez egy nagy előny, ezáltal ui. lehetővé válik, hogy egyszerre több programot töltsünk be a memóriába és futtassunk. Például legyen egy olyan programunk, ami a 0100H címnél kezdődik a RESULT elnevezésű felhasználó által definiált változóval, amelynek értékét a 4F3AH memóriahelyen tárolta a rendszer. Ha egy másik programnak is ugyanerre a memóriahelyre van szüksége, akkor ebből probléma keletkezik. Az áthelyezhető prgramot azonban a rendelkezésre álló memóriában feljebb vagy lejjebb helyezhetjük. A szerkesztőprogram minden szükséges címet módosít, és ezen kívül néhány rendkívül hasznos műveletet is végrehajt.

ssze tudja kapcsolni a különállóan lefordított .OBJ file-okat, ami a kis modulokból építkező nagy assembly programoknál hasznos. A szerkesztőprogram a forráskódban szereplő szubrutin hívásokhoz megkeresi a szubrutinokat a programkönyvtárban.

A szerkesztőprogram életre keltéséhez csak a **LINK** szót kell beírnunk, és ekkor a következő üzenet jelenik meg a képernyőn:

```
IBM Personal Computer Linker
Version 2.20 (C)Copyright IBM Corp 1981, 1982, 1983, 1984
Object Modules [.OBJ]:
```

A szerkesztőprogram annak az .OBJ file-nak a nevét várja tőlünk, amelyet szerkesztetni akarunk.

Térjünk vissza az eredeti példánkhoz, és írjuk be a **GAME** szót. Az .OBJ kiterjesztést felesleges megadnunk, mert a szerkesztőprogram ezt automatikusan hozzáfűzi a file-névhez. Ezt követően egy üzenetet láthatunk:

```
Run File [GAME.EXE]:
```

Alapértelmezésként a szerkesztőprogram a futtatható program nevét ajánlja fel, amit felülírhatunk, azonban a szerkesztőprogram mindig az .EXE kiterjesztést támogatja.

A fenti file-név elfogadásához az ENTER billentyűt kell lenyomnunk, amelyre a következő üzenet jelenik meg:

```
List File [NUL.MAP]:
```

A .MAP kiterjesztéssel ellátott file a forráskód minden szegmense számára egy listát

tartalmaz, ami a futtatható file helyzetcímeit adja meg. A hibakeresésnek ez is egy hasznos eszköze.

Alapértelmezés szerint a szerkesztőprogram ezt a file-t nem készíti el. (Erre a szögletes zárójelben látható NUL file-név is utal.) A .MAP file elkészítéséhez be kell írnunk a **GAME** nevet majd lenyomnunk az ENTER gombot. Ekkor megpillanthatjuk a képernyőn a negyedik és egyben utolsó üzenetet is:

```
Libraries [.LIB]:
```

A szerkesztőprogram azon könyvtárak nevének megadását várja tőlünk, amelyekben a forráskódban előírt rutinok szerepelnek. A jelenlegi példánkban nem szerepel felhasználó által definiált könyvtár, tehát az ENTER gomb lenyomásával ezen a lépésen továbbjuthatunk.

A következő üzenetet látjuk:

```
Warning: No STACK segment   (Figyelmeztetés: nincs veremszegmens)
There was 1 error detected  (1 hiba felderítve)
```

Ez az üzenet érvényes ránk nézve, mivel a forráskód nem tartalmaz előre definiált veremszegmenset. Jelen esetben az üzenet a keletkező file típusa miatt elhanyagolható, ui. a GAME.ASM elnevezésű mintaprogramot úgy írtuk meg, hogy .COM kiterjesztésű tömörített (compact) file keletkezzen belőle.

Készítsünk egy katalógust a B meghajtóban nyilvántartott file-okból. Ha a szoftverfejlesztés minden egyes lépését a leírtak szerint végeztük el, akkor a következő listához hasonlót kell megpillantanunk a képernyőn:

```
GAME  ASM      758          12-27-85          6:47p
GAME  OBJ       57           1-02-86          6:36p
GAME  LST     1656           1-02-86          6:36p
GAME  CRF       69           1-02-86          6:36p
GAME  REF      268           1-02-86          6:36p
GAME  MAP      154           1-02-86          6:36p
GAME  EXE      524           1-02-86          6:36p
       7 File(s) 4159488 bytes free
```

A GAME.EXE file, amely áthelyezhető, a GAME program futtatható változata.

Mielőtt azonban ezt a file-t .COM-má alakítanánk, ismerkedjünk meg néhány szerkesztési paraméterrel és parancsrövidítési módozattal.

Szerkesztési paraméterek és parancsok rövidítései

Az A–2. táblázatban felsorolt opcionális paraméterek a / (törtjel) segítségével kapcsolhatók a szerkesztőparancshoz. A makroassemblerhez hasonlóan a szerkesztőprogram is használható rövidített parancsokkal, amelyeket az B–3. táblázatban soroltunk fel.

Paraméter	Magyarázat
/DSALLOCATION	A /DS-ként rövidíthető paraméter arra utasítja a szerkesztőprogramot, hogy az adatcsoportot tartalmazó szegmensben a csoport a lehető legnagyobb címre kerüljön. Alapértelmezés szerint az adatok első byte-jának helyzetcíme 0.
/HIGH	A /H-ként rövidíthető paraméter a szerkesztőprogramot arra utasítja, hogy a futtatható programot olyan magasra helyezze a memóriában, amennyire ez csak lehetséges annak érdekében, hogy a COMMAND.COM tranziens részével nehogy fedésbe kerüljön.
/LINE	A /L vagy /LINE opció hatására az input modulok forrásutasításainak címét és sorszámát a szerkesztőprogram belefoglalja a listázható file-ba.
/NODEFAULPT LIBRARYSEARCH	Ez a paraméter /NOD-ként rövidíthető és azt jelenti a szerkesztőprogram számára, hogy az alapértelmezés szerinti könyvtárakban ne hajtsa végre a szokásos keresési folyamatot.
/NOGROUP ASSOCIATION	Az /NOG-ként rövidített paraméter arra utasítja a szerkesztőprogramot, hogy a hosszú external címzéseket kijavítsa még akkor is, ha a szimbólumot egy definiált csoporton belüli szegmensben adtuk meg.
/PAUSE	A /P hatására a szerkesztőprogram felszólít a futtatható file-t tartalmazó lemez meghajtóba helyezésére.
/STACK: méret	A /S paraméter, amelyet egy 0 és 65 536 közé eső számnak kell követnie, lehetővé teszi, hogy a programozó megadja a veremszegmens méretét. A 0–512 tartományba eső értékek 512 byte-os minimális veremméret megadását jelentik.

A–2. táblázat Szerkesztési paraméterek

Utasítás	Magyarázat
LINK **file-nev**;	Ez az utasítás indítja a szerkesztőprogramot, ami a **file-nev**.OBJ file-t használja inputként. Az output file-t automatikusan **file-nev**.EXE-nek nevezi el.
LINK **file-nev**,,;	Ez a rövidített parancs arra készteti a szerkesztőprogramot, hogy a **file-nev**.OBJ file-t keresse inputként, és készítse el a **file-nev**.EXE és **file-nev**.MAP output file-okat.
Megjegyzés:	Az előbbi parancsban szereplő vesszők hatására nem jelennek meg a szerkesztőprogram üzenetei és minden egyes opció az alapértelmezésként megadott file-nevet veszi fel. Ha valamelyik opciót el akarjuk hagyni, a vessző helyett a NUL parancsot kell behelyettesítenünk.

A–3. táblázat Szerkesztési parancsok rövidítése

EXE. kontra .COM

Az .EXE kiterjesztésű file-oknak kétszeres előnyük van. Az egyik előny az áthelyezhetőség, így ui. lehetővé válik, hogy egynél több programot töltsünk be a memóriába, kihasználva a rendelkezésre álló helyet. A másik előny a

STACK (verem)
DATA (adat)
CODE (kód)
EXTRA

szegmensek használata, amely által lehetővé válik nagy méretű programok kifejlesztése moduláris programozással.

Ezzel szemben a .COM file csak egyetlen szegmenst tartalmazhat, ami a STACK, DATA és CODE információkat hordozza. Ha az assembly program kis méretű, a .COM file bőségesen elegendő valamennyi adat, utasítás és veremművelet tárolására.

Az .EXE file-oknak – amelyekben a szegmensek maximális mérete 64 K – az a hátrányuk, hogy a szerkesztőprogram ezeket a file-okat egy bizonyos mérettel megnöveli annak érdekében, hogy az áthelyezési művelet lehetővé váljon.

Példánkban a GAME.EXE file mérete 524 byte. Az .EXE file-tól eltérően a .COM file nem helyezhető át a memóriában, és a 0100H címnél kell kezdődnie. A .COM file lényegesen kisebb méretű, mint az .EXE file.

A .COM file-ok elkészítése

Az EXE2BIN.EXE kiszolgálóprogram futtatásával az .EXE file-ból .COM-ot készíthetünk. Irjuk be tehát a következőket:

```
B>A:EXE2BIN GAME GAME.COM
```

majd nyomjuk le az ENTER gombot.

Feltételezve azt, hogy az EXE2BIN program az A meghajtóban van, az A:EXE2BIN parancs elindítja a programot. Jelen esetben a GAME elnevezésű .EXE file-t kell konvertálni. (Vegyük észre, hogy a GAME után nem szükséges az .EXE kiterjesztést megadni.) A keletkező .COM file neve GAME.COM lesz. Ebben az esetben azonban lényeges a file-kiterjesztés megadása, ellenkező esetben ui. az EXE2BIN program a .BIN kiterjesztést fűzi a file-névhez. Igy azonban a file nem lenne futtatható, és a RENAME (átnevezés) paranccsal meg kellene változtatnunk a kiterjesztését. Most pedig hasonlítsuk össze a GAME.EXE és a GAME.COM file-ok méretét. A következőkhöz hasonlót kell látnunk a képernyőn:

```
GAME  EXE      524          1-02-86          7:45p
GAME  COM      12           1-02-86          7:45p
```

Figyeljük meg a két file-méret közötti lényeges különbséget. Az A–4.táblázatban az IBM Makroassembler kiszolgálóprogramjait foglaltuk össze, feltüntetve a bemeneti és kimeneti formátumokat.

Kiszolgálóprogram	Leírás	
MASM	A bemeneti file formátuma:	MYFILE.ASM (ASCII szövegfile)
	A kimeneti file formátuma:	MYFILE.OBJ (Gépi kód) MYFILE.LST (Listafile) MYFILE.CRF (A CREF által felhasznált file)
CREF	A bemeneti file formátuma:	MYFILE.CRF (A CREF által felhasznált file)
	A kimeneti file formátuma:	MYFILE.REF (Keresztreferencia-file)
LINK	A bemeneti file formátuma:	MYFILE.OBJ (Gépi kód) MYFILE.LIB (A felhasználó által definiált könyvtár)
	A kimeneti file formátuma:	MYFILE.EXE (Futtatható program) MYFILE.MAP (Memóriatérkép)
EXE2BIN	A bemeneti file formátuma:	MYFILE.EXE (Egyszegmenses kód)
	A kimeneti file formátuma:	MYFILE.COM (Futtatható program)

A–4. táblázat A kiszolgálóprogramok Input/Output file-formátumai

B MELLÉKLET

A MIKROSOFT MAKROASSEMBLER

Ebben a mellékletben az Olvasóa Microsoft Makroassembler jellemzőivel és használatával ismerkedhet meg, és egy mintaprogram segítségével megtanulhatja, hogyan kell létrehozni, lefordítani és szerkeszteni az assembly programokat.

Azt is megtudhatja, hogyan használja fel a forráskódot a fordítóprogram az .OBJ célfile létrehozására, mit kell érteni a listázó és a keresztreferenca file-okon és miért van ezekre szükség. Miután megtanultuk, hogyan kell az .OBJ file-t létrehozni, azt is meglátjuk, hogyan alakítja át a szerkesztőprogram az .OBJ file-okat .EXE file-okká. (Ebben a szakaszban opcionális .MAP file-t is létrehozhatunk.)

A mintaprogram segítségével elmagyarázzuk, hogyan kell ezt a file-t létrehozni és rámutatunk a hasznosságára.

A mellékletből még az is kiderül, hogy mi a különbség az .EXE és a .COM file között. Elsajátíthatjuk az EXE2BIN.EXE program használatát is, amivel az .EXE file-okból .COM készíthető.

Általános információk

A Microsoft Makroassembler olyan programok lefordítására képes, amelyek a 8086-os, 80186-os és 80286-os mikroprocesszorokon, valamint a 8087-es és 80287-es matematikai társprocesszszorokon futtathatók.

A Microsoft Makroassembler 2.0-ás vagy ennél később készült DOS változat használatát, és minimálisan 128 K belső memóriát tételez fel.

Az assembly forráskód létrehozása

Az assembly nyelvű programfejlesztés első lépéseként mindig a megoldandó probléma pontos körbehatárolását és definiálását kell elvégeznünk.

Például elképzelhető az a feladat, hogy a számítógéphez csatlakoztatott játékadapter potenciométer értékeit akarjuk megtudni.

Mielőtt a konkrét programozáshoz hozzákezdenénk, információt kell szereznünk arról, hogyan fér hozzá a számítógép a játékadapterhez. A kérdésre az IBM Műszaki Kézikönyve BIOS listáiban találhatjuk meg a választ.

Ezek után elkészíthetjük a forráskódot. Mint a 2. fejezetben már ismertettük, a forráskódban olyan mnemonikok szerepelnek, amelyek gépi kódú utasításokat jelentenek. A forráskódot szigorú szintaktikai (helyesírási) szabályok szerint kell elkészíteni azért, hogy a fordítóprogram mindent helyesen tudjon értelmezni. A forráskódot bármilyen sor, vagy teljes képernyős

szövegszerkesztővel elkészíthetjük, csak az a lényeg, hogy az output file ASCII formátumban legyen.

A következőkben bemutatása kerülő forráskódot, amelynek a **GAME.ASM** file-nevet adtuk, egy ASCII szövegkezelővel (pl. a Norton Editor-ral) készítettünk el. Az .ASM kiterjesztés tudatja a rendszerrel, hogy assembly nyelven készített forráskódról van szó.

```
;80286/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
;A PROGRAM BEOLVASSA A JATEKADAPTERT, ES A POT. ERTEKEKET
;A KIJELOLT REGISZTEREKBE ADJA VISSZA

PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA


MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MYCODE
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

;A JATEKADAPTER POT. ERTEKEINEK BEOLVASASA MEGSZAKITAS
;SEGITSEGEVEL
        MOV     AH,84H          ;A BOTKORMANY MINTAVETELEZESE
        MOV     DX,01H          ;A POT. ERTEKEK MINTAVETELEZESE
        INT     15H             ;A MUVELET VEGREHAJTASA


        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A forráskód elkészítése után a programfejlesztés következő fázisába léphetünk, vagyis a forráskódot a makroassembler felhasználásával átalakítjuk olyan formába, amit a mikroprocesszor megért.

A makroassembler használata

A Microsoft Makroassebler az .ASM file-okból (forráskódokból) .OBJ file-okat, vagyis az .ASM file-ok gépi kódú megfelelőit készíti el két menetben. Az első menetben (fázisban) készül el az ún. relatív helyzet (offset) a forráskód minden egyes sorához, és felépül a szimbólumtáblázat, ami még nem jelenti a tárgykód elkészültét. (L. a B–1. táblázatban a /D paraméterre vonatkozó megjegyzést.)

Az első menetben definiált értékekből a második menetben készül el az .OBJ file.

Fázishibák

A fázishibák a fordítási művelet második menetében akkor keletkeznek, ha a fordítóprogram valamely változóra, címkére vagy eljárásra más értéket talál, mint amit az első menetben a szimbólumtábla felépítésekor tárolt.

A következőkben felhasznált utasítások azt feltételezik, hogy a GAME.ASM forrásfile a B meghajtóban, a Microsoft Makroassebler pedig az A meghajtóban van, valamint azt, hogy

Paraméter	Magyarázat
/A	A paraméter hatására a forráskód szegmenseinek listázása történik alfabetikus sorrendben. A fordítóprogramban ez az alapértelmezés szerinti paraméter.
/B	Ez a paraméter állítja be a file pufferméretét a forráskód számára. A definiálás kbyte-okban történhet és 32 kbyte az alapértelmezés szerinti érték. A file puffermérete maximálisan 63 K lehet (nem 64 K!).
/C	Ez a paraméter utasítja a Makroasseblert (MASM), a keresztreferencia-file elkészítésére.
/D	Fázishiba akkor keletkezik, ha a fordítási eljárás első és második menetébn keletkezett értékek között eltérés van. A /D paraméter megdásakor a fordítóprogram az első menetről is listát készít, amelyet összehasonlíthatunk a második menetben automatikusan elkészülő listával.
/E	Ez a paraméter fordítja le a 80287/80387-es processzorokra vonatkozó forráskódot, és olyan lebegőpontos konstansokat készít, amelyek a társprocesszor számára szükségesek. Hasonló módon működik, mint az /R parméter.
/I	Ezzel a paraméterrel állítható be bármelyik include file részére a kereső útvonal. Maximálisan 10 kereső útvonal lehetséges.
/ML	Ez a paraméter utasítja a MASM-et, hogy a változónevekben szereplő kis- és nagybetűk között tegyen különbséget. Ebben az esetben tehát a MEM_WORD és a Mem_Word két különböző változót jelent majd.
/MU	Ennek az opciónak a hatására a MASM valamennyi kisbetűt nagybetűvé alakítja a public és external nevekben.
/MX	Ez a paraméter arra utasítja a MASM-et, hogy fenntartsa az eset érzékenységét valamennyi public és external változóval szemben.
/N	Ennek a paraméternek a hatására a fordítóprogram nem készít szimbólumtáblát a listázás végén.
/P	Ez az opció a MASM-et a 80286-os processzor váratlan védett módú kizárásainak ellenőrzésére készteti. Az opció csak a 80286/80386-os mikroprocesszorok vezérlése alatt aktiválható.
/R	Ez a paraméter a 80287/80387-es társprocesszor részére vonatkozó forráskódot úgy fordítja le, hogy a numerikus értékek a társprocesszor által elvárt formátumban keletkezzenek. A /R paraméterrel létrehozott gépi kódok a társprocesszor nélküli rendszereken nem futnak megfelelően.
/S	Ez a paraméter felülírja az alapértelmezés szerinti /A paramétert. A szegmensek olyan sorrendben kerülnek listázásra, ahogy a forráskódban megjelennek.
/T	Ez a paraméter a hibajelzéstől eltérő assembler üzenetek kinyomtatását letiltja.
/V	Ez a paraméter arra készteti a fordítóprogramot, hogy a fordítás során további statisztikákat is készítsen.
/X	Az opció hatására a fordítóprogram a listázófile részére másolatot készít azokból az utasításokból, amelyek az IF direktíva törzsét képezik, ha az IF vizsgálatakor hamis érték keletkezik.
/Z	Ez egy nagyon hasznos opció, amely arra utasítja a MASM-t, hogy listázza a hibákat tartalmazó kódsorokat.

B–1. táblázat A Mikrosoft Makroassembler paraméterei

a B meghajtó az alapértelmezés szerinti meghajtó. A B> rendszerüzenetnek kell a képernyőn lennie.

A fordítóprogram futtatásához a következőket kell beírnunk:

```
B>A:MASM
```

Ennek hatására a képernyőn egy üzenet jelenik meg:

```
Microsoft (R) Macro Assembler Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985. All rights
reserved.
Source filename [.ASM]:
```

A fordítóprogram mindig azt adja meg a szögletes zárójelek között, amit alapértelmezés szerint feltételez. A forrásfile kiterjesztése alapértelmezés szerint .ASM . Mivel a GAME elnevezésű forrásfile-t az .ASM kiterjesztéssel menettük el, elegendő a **GAME** beírása és az ENTER gomb lenyomása.

Megjegyezzük, hogy a Microsoft Makroassembler lehetővé teszi a fixlemezek használata során előforduló útvonalnevek megadását is. Ilyen eseteken a rendszerüzenetekre adott válaszoknak tartalmazniuk kell a meghajtót és az alkatalógust.

Az ENTER gomb lenyomása után egy új rendszerüzenetet láthatunk:

```
Object filename [GAME.OBJ]:
```

A fordítóprogram ezzel az üzenettel azt jelzi számunkra, hogy a célfile-nak a GAME.OBJ file-nevet adja, hacsak nem adjuk meg másképpen. A file-név elfogadásához az ENTER gombot kell lenyomnunk.

Ekkor a harmadik rendszerüzenet jelenik meg:

```
Source listing [NUL.LST]:
```

Az .LST kiterjesztésű file az eredeti forráskód lefordított változata. A forráskód itt annyiban módosul, hogy sorszámozást, az egyes utasítások gépi kódú megfelelőit és szimbólumtáblát is tartalmaz. Az üzenetben a NUL mnemonik jelzi, hogy a fordítóprogram az alapértelmezés szerint nem készíti el ezt a file-t. Ha igénybe akarjuk venni ezt a hasznos szolgáltatást, be kell írnunk a **GAME** nevet és le kell nyomnunk az ENTER gombot. A fordítóprogram a kiterjesztést automatikusan szolgáltatja, ezért az .LST beírása szükségtelen.

Az ENTER gomb lenyomása után megjelenik a negyedik és egyben utolsó rendszerüzenet is:

```
Cross reference [NUL.CRF]:
```

A fordítóprogram a .CRF kiterjesztésű file-okat, amelyeket kereszreferencia-listának nevezünk, mint közbenső lépést készíti el a fordítási folyamat során. (A .CRF kiterjesztésű file-okat a melléklet későbbi részében még tárgyaljuk.) Az alapértelmezés szerint a program nem készíti el ezt a file-t sem, amit a NUL-lal jelez. Ha igényt tartunk ennek a file-nak az elkészítésére, a **GAME** file-nevet kell beírnunk, majd lenyomnunk az ENTER gombot. Ezek után a fordítóprogram végrehajtja a GAME.ASM forrásfile-on a megadott műveleteket, és elkészíti az .OBJ, .LST és .CRF file-okat. Kérjünk egy katalógust a B meghajtóról, és győződjünk meg arról, hogy fordítóprogram ténylegesen végrehajtotta a fordítási műveleteket. A file-ok listája a következőhöz hasonló lesz:

```
GAME    ASM        758          12-27-85          6:47p
GAME    OBJ         57          12-29-85          2:55p
GAME    LST       1656          12-29-85          2:55p
GAME    CRF         69          12-29-85          2:55p
```

Annyit már tudunk, hogy a GAME.ASM tartalmazza a program ASCII kódú változatát, és hogy a GAME.OBJ file nem listázható, mivel ez a programunk gépi kódú megfelelője. A GAME.CRF file sem listázható, mivel ez csak a keresztreferencia-táblázat elkészítéséhez szükséges közbenső lépcső. A GAME.LST file azonban listázható. Mint már korábban említettük, a .LST kiterjesztésű file-ok a sorszámokkal ellátott forrásprogramot az egyes utasítások gépi kódú megfelelőit és a szimbólumtáblát tartalmazzák.

A hardcopy készítés legegyszerűbb módja a **TYPE GAME.LST** paranccsal történő listázás. (Ne felejtsük el bekapcsolni a nyomtatót és a parancs beírása után lenyomni a CTRL PRTSC + ENTER billentyűket.) A monitor képernyőjén a következő listának kell megjelennie:

```
                                ;80286/80386-OS SZAMITOGEPEKRE KESZITETT PROGRAM
                                ;A PROGRAM BEOLVASSA A JATEKADAPTERT, ES A POT. ERTEKEKET
                                ;A KIJELOLT REGISZTEREKBE ADJA VISSZA

                                PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA


0000                            MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
0000                            MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
                                        ASSUME  CS:MYCODE
0000  1E                                PUSH    DS              ;A DS REGISZTER ELMENTESE
0001  2B C0                             SUB     AX,AX           ;AX TORLESE
0003  50                                PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE

                                ;A JATEKADAPTER POT. ERTEKEINEK BEOLVASASA MEGSZAKITAS
                                ;SEGITSEGEVEL
0004  B4 84                             MOV     AH,84H          ;A BOTKORMANY MINTAVETELEZESE
0006  BA 0001                           MOV     DX,01H          ;A POT. ERTEKEK MINTAVETELEZESE
0009  CD 15                             INT     15H             ;A MUVELET VEGREHAJTASA


000B  CB                                RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
000C                            MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
000C                            MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                        END                     ;A PROGRAM VEGE


Segments and groups:

                N a m e                 Size    align   combine class

MYCODE . . . . . .              000C    PARA    NONE    'CODE'

Symbols:

                N a m e                 Type    Value   Attr

MYPROC . . . . . .              F PROC  0000    MYCODE  Length =000C

Warning Severe
Errors  Errors
0       0
```

B–1. ábra Programlista

Paraméterek

A Microsoft Makroassembler a programok lefordítása során számos választható paramétert (opciót) bocsát a felhasználó rendelkezésére. Az opciókat, amelyeket a B–1. táblázatban sorolunk fel abc sorrendben, a parancs-sorban kell megadni a / jel után. (Az opciókról részletesen a Mikrosoft Makroassembler referencia kézikönyvében olvashatunk.)

Parancsok rövidítése

A fordítóprogram elindításához számos rövidítést használhatunk. Mint már a 2. fejezetben említettük, a

```
B>A:MASM GAME;
```

sor beírásával a fordítóprogram közvetlenül elkészíti az .OBJ file-t. Nem generál azonban .LST és .CRF file-okat. A következő parancsrövidítések elfogadják a .LST és .CRF file-okra vonatkozó alapértelmezésként szereplő elnevezést, ezáltal lehetővé teszik, hogy a programozónak ne kelljen mindig válaszolnia a rendszerüzenetekre.

```
B>A:MASM GAME,,,;
```

A vesszők letiltják az assembler rendszerüzeneteket, és az adott opciónak a forráskódból következően az alapértelmezés szerinti file-nevet adják. A NUL paranccsal az adott opció letiltható. A következőkben a parancssorban szereplő opcionális paraméterhasználatra mutatunk példát:

```
B>A:MASM GAME/N;
```

Az előbbi opció hatására a fordítóprogram a listázás végén nem készíti el a szimbólumtáblát.

Keresztreferencia-listázás: CREF.EXE

A keresztreferencia-lista nagyon hasznos a hibakereséskor, ui. az adatok, változók, címkék, konstansok és egyéb szimbólumok alfabetikus listáját tartalmazza. A listában az a sorszám is szerepel, ahol a szimbólumot először definiáltuk, valamint valamennyi további sort, amelyben a szimbólum még előfordul. A keresztreferencia-lista elkészítéséhez a negyedik assembler rendszerüzenetre válaszolnunk kell, amelynek hatására elkészül a filenév.CRF közbenső file. Ezt a file-t a CREF.EXE használja fel az aktuális keresztreferencia-lista elkészítéséhez.

A következő parancsot írjuk be tehát:

```
B>CREF
```

majd nyomjuk le az ENTER billentyűt. A képernyőn a következőhöz hasonló felirat jelenik meg:

```
Microsoft (R) Cross-Reference Utility Version 4.00
Copyrigh (C) Microsoft Corp 1981,1983,1984,1985.
All rights reserved.
Cross-reference [.CRF]:
```

A CREF.EXE kiszolgálóprogram a .CRF kiterjesztésű file-nevet kéri. A mi példánkban a **GAME** a file neve, tehát ezt kell beírnunk, majd lenyomnunk az ENTER billentyűt. A .CRF

kiterjesztést nem szükséges leírnunk, mert a kiszolgálóprogram automatikusan hozzáfűzi a file-névhez.

Miután a program a megelőző lépést végrehajtotta, ismét egy üzenettel jelentkezik:

```
List filename [GAME.REF]:
```

A CREF.EXE kiszolgálóprogram alapértelmezésként a GAME.REF programnevet adja az elkészítendő file-nak, amit a felhasználó természetesen felülírhat, ha ettől eltérő meghajtót akar használni. Ha azonban elfogadjuk a program által javasolt file-nevet, csak az ENTER billentyűt kell lenyomnunk.

Rövidítések a CREF.EXE parancs használatakor

Mint a makroassemblernél láttuk, a keresztreferencia-lista elkészítéséhez is használhatunk rövidítéseket. A

```
B>CREF GAME;
```

beírásával a CREF.EXE program automatikusan megkeresi a GAME.CRF file-t és elkészíti ugyanilyen névvel a .REF kiterjesztésű file-t. A jelenlegi esetben ennek a file-nak a GAME.REF nevet adja a program. Ha a file hiba nélkül létrejön, az alapértelmezés szerinti meghajtó szimbóluma jelenik meg a képernyőn.

Készítsünk a B meghajtóról egy tartalomjegyzéket, hogy lássuk, a file tényleg létrejött. A **TYPE GAME.REF** parancs beírásával a következőhöz hasonló keresztreferencia-lista jelenik majd meg a képernyőn:

```
B>TYPE GAME.REF
Microsoft Cross-Reference    Version 4.00    Mon Dec 0202:35:00 1985
  Symbol Cross-Reference     (#is definition)
Cref-1
CODE                 8
MYCODE               8    8#    10    23
MYPROC               9    9#
  3 Symbols
  Symbol Cross-Reference     (#is definition)
Cref-1
CODE                 8
MYCODE               8#    10    23
MYPROC               9#    22

  3 Symbols
  63048 Bytes Free
```

A listából megállapítható a programban található szegmensek száma. A mi esetünkben csak egy ilyen szegmens van, a CODE. A MYCODE és MYPROC deklarációk is láthatóak. A # szimbólum, amely egy sorszámot követ, azt jelenti, hogy az adott szimbólum ebben a sorban szerepelt először. A további sorszámok azokra a sorokra utalnak, amelyekben az adott szimbólum még ismételten szerepel.

Még ebből a rövid példából is látható, hogy a keresztreferencia-lista segítségével meggyőződhetünk arról, hogy a kódszegmensek az END utasításokat is tartalmazzák (pl. a MYCODE a 23. vagy a MYPROC a 22. sorban).

Minél nagyobb az assembly program, annál hasznosabb a keresztreferencia-lista a hibakeresésben. Az előbbi file-ról nagyon egyszerűen készíthető hardcopy, csupán csak be kell kapcsolni a nyomtatót és be kell írni a **TYPE** után a file nevét a következők szerint:

```
B>TYPE GAME.REF
```

Ezek után a CTRL-PRTSC, majd az ENTER billentyűket kell lenyomnunk. A listázás befejezése után ne feledkezzünk meg a CTRL-PRTSC ismételt lenyomásáról, különben mindent kinyomtat a nyomtató, ami a képernyőn megjelenik.

Akár készítettünk keresztreferencia-listát, akár nem, az assembly proramfejlesztés következő lépése az .OBJ file szerkesztése, amelyet a LINK.EXE prorammal tudunk megvalósítani.

Szerkesztés a LINK.EXE programmal

A LINK.EXE kiszolgálóprogram az ún. áthelyezhető programot készíti el. Ha az .OBJ file-hoz egy kiegészítő fejrészt adunk, a szerkesztőprogram engedélyezi az operációs rendszernek, hogy az assembly programot a rendelkezésre álló memória bármely részére elhelyezze.

Ez egy nagy előny, ui. így lehetővé válik, hogy egyszerre több programot töltsünk be a memóriába és futtassunk. Például legyen egy olyan programunk,amely a 0100H címnél kezdődik a RESULT elnevezésű felhasználó által definiált változóval, amelynek értékét a 4F3AH memóriahelyen tárolta a rendszer. Ha egy másik programnak is ugyanerre a memóriahelyre van szüksége, akkor ebből probléma keletkezik. Az áthelyezhető programot azonban a rendelkezésre álló memóriában feljebb vagy lejjebb helyezhetjük. A szerkesztőprogram minden szükséges címet módosít, és ezen kívül néhány rendkívül hasznos műveletet is végrehajt. ssze tudja kapcsolni a különállóan lefordított .OBJ file-okat, ami a kis modulokból építkező nagy assembly programoknál hasznos.

A szerkesztőprogram a forráskódban szereplő szubrutin hívásokhoz megkeresi a szubrutinokat a programkönyvtárban.

A szerkesztőprogram futtatásához csak a **LINK** szót kell beírnunk, és akkor a következő üzenet jelenik meg a képernyőn:

```
Microsoft (R) 8086 Object Linker Version 3.05
Copyright (C) Microsoft Corp 1983, 1984, 1985.All rights
reserved.
Object Modules [.OBJ]:
```

A szerkesztőprogram annak az .OBJ file-nak a nevét várja tőlünk, amelyet szerkesztetni akarunk.

Térjünk vissza az eredeti példánkhoz, és írjuk be a **GAME** szót. Az .OBJ kiterjesztést felesleges megadnunk, mert a szerkesztőprogram ezt automatikusan hozzáfűzi a file-névhez.

Ezt követően egy üzenetet láthatunk:

```
Run File [GAME.EXE]:
```

Alapértelmezésként a szerkesztőprogram a futtatható program nevét ajánlja fel, amit felülírhatunk, azonban a szerkesztőprogram mindig az .EXE kiterjesztést fogja támogatni.

A fenti file-név elfogadásához az ENTER billentyűt kell lenyomnunk, amelyre a következő üzenet jelenik meg:

```
List File [NUL.MAP]:
```

A .MAP kiterjesztéssel ellátott file a forráskód minden szegmense számára egy listát tartalmaz, ami a futtatható file helyzetcímét adja meg.

A hibakeresésnek ez is egy hasznos eszköze. Alapértelmezés szerint a szerkesztőprogram ezt a file-t nem készíti el. (Erre a szögletes zárójelben látható NUL file-név is utal.) A .MAP file elkészítéséhez be kell írnunk a **GAME** nevet, majd lenyomnunk az ENTER gombot.

Ekkor megpillanthatjuk a képernyőn a negyedik és egyben utolsó üzenetet is:

```
Libraries [.LIB]:
```

A szerkesztőprogram azon könyvtárak nevének megadását várja tőlünk, amelyekben a forráskódban előírt rutinok szerepelnek. A jelenlegi példánkban nem szerepel felhasználó által definiált könyvtár, tehát az ENTER gomb lenyomásával ezen a lépésen továbbjuthatunk.

A következő üzenetet fogjuk látni:

```
Warning: No STACK segment      (Figyelmeztetés: nincs veremszegmens)
```

Ez az üzenet érvényes ránk nézve, mivel a forráskód nem tartalmaz előre definiált veremszegmenset. Jelen esetben az üzenet a keletkező file típusa miatt elhanyagolható, ui. a GAME.ASM elnevezésű mintaprogramot úgy írtuk meg, hogy .COM kiterjesztésű tömörített (compact) file keletkezzen belőle.

Most pedig készítsünk egy katalógust a B meghajtóban nyilvántartott file-okból. Ha a szoftverfejlesztés minden egyes lépését a leírtak szerint végeztük el, akkor a következő listához hasonlót kell megpillantanunk a képernyőn:

```
GAME  ASM      758          12-27-85          6:47p
GAME  OBJ      57           1-02-86           6:36p
GAME  LST      1656         1-02-86           6:36p
GAME  CRF      69           1-02-86           6:36p
GAME  REF      268          1-02-86           6:36p
GAME  MAP      154          1-02-86           7:13p
GAME  EXE      524          1-02-86           7:13p
     7 File(s)  4159488 bytes free
```

A GAME.EXE file, amely áthelyezhető, a GAME program futtatható változata.

Mielőtt azonban ezt a file-t .COM-má alakítanánk, ismerkedjünk meg néhány szerkesztési paraméterrel és parancs rövidítési módozattal.

Szerkesztési paraméterek és parancsok rövidítései

A B–2. táblázatban felsorolt opcionális paraméterek a / (törtjel) segítségével kapcsolhatók a szerkesztőparancshoz. A makroassemblerhez hasonlóan a szerkesztőprogram is használható rövidített parancsokkal, amelyeket a B–3. táblázatban soroltunk fel.

.EXE kontra .COM

Az .EXE kiterjesztésű file-oknak kétszeres előnyük van. Az egyik előny az áthelyezhetőség, így ui. lehetővé válik, hogy egynél több programot töltsünk be a memóriába, kihasználva a rendelkezésre álló helyet. A másik előny a

STACK (verem)
DATA (adat)
CODE (kód)
EXTRA

szegmensek használata, amely által lehetővé válik nagy méretű programok kifejlesztése moduláris programozással.

Az .EXE file-oknak – melyekben a szegmensek maximális mérete 64 kbyte – az a hátrányuk, hogy a szerkesztőprogram ezeket a file-okat egy bizonyos mérettel megnöveli, annak érdeké-

Paraméter	Magyarázat
/HELP	Rövidített formája: /HE, ami arra utasítja a szerkesztőprogramot, hogy a rendelkezésre álló opciókat nyomtassa ki a képernyőre.
/DSALLOCATION	A /DS-ként rövidíthető paraméter arra utasítja a szerkesztprogramot, hogy az adatcsoportot tartalmazó szegmensben a csoport a lehető legnagyobb címre kerüljön. Alapértelmezés szerint az adatok első byte-jának helyzetcíme 0.
/EXEPACK	A /E-ként rövidíthető paraméter arra készteti a szerkesztőprogramot, hogy törölje az ismételt byte-ok sorozatát, ezáltal az áthelyezési táblázat méretét csökkentse. (Bár ez az opció a file méretét valójában megnövelheti.) Az opció használatát akkor javasoljuk, ha a forráskód sok ismételt karaktert tartalmaz.
/HIGH	A /H-ként rövidíthető paraméter a szerkesztőprogramot arra utasítja, hogy a futtatható programot olyan magasra helyezze a memóriában, amennyire ez csak lehetséges annak érdekében, hogy a COMMAND.COM tranziens részével nehogy fedésbe kerüljön.
/LINENUMBERS	A /L vagy /LINE opció hatásra az input modulok forrásutasításainak címét és sorszámát a szerkesztőprogram belefoglalja a listázható file-ba.
/MAP	A /M paraméter hatására a szerkesztőprogram a forráskódban szereplő PUBLIC szimbólumokról listát készít.
/NOIGNORECASE	A /NOI-ként rövidíthető paraméter hatására a szerkesztőprogram a kis- és nagybetűk között különbséget tesz. Ezt az opciót a /ML vagy a /MX opciókkal kombinálva a MASM-ben a public változók esetérzékennyé válnak.
/NODEFAULT LIBRARYSEARCH	Ez a paraméter /NOD-ként rövidíthető és azt jelenti a szerkesztőprogram számára, hogy az alapértelmezés szerinti könyvtárakban ne hajtsa végre a szokásos keresési folyamatot.
/NOGROUP ASSOCIATION	Az /NOG-ként rövidített paraméter arra utasítja a szerkesztőprogramot, hogy a hosszú external címzéseket kijavítsa még akkor is, ha a szimbólumot egy definiált csoporton belüli szegmensben adtuk meg.
/PAUSE	A /P hatására a szerkesztőprogram felszólít a futtatható file-t tartalmazó lemez meghajtóba helyezésére.
/STACK :méret	A /S paraméter, amelyet egy 0 és 65 536 közé eső számnak kell követnie, lehetővé teszi, hogy a programozó megadja a veremszegmens méretét. A 0–512 tartományba eső értékek 512 byte-os minimális veremméret megadását jelentik.
/CPARMAXALLOC	A /C-ként rövidíthető paraméter a szükséges 16 byte-os paragrafusok maximális számát állítja be, amikor a programot betöltjük a memóriába.

B–2. táblázat Szerkesztési paraméterek

Utasítás	Magyarázat
LINK **file-nev**;	Ez az utasítás elindítja a szerkesztőprogramot, ami a **file-nev**.OBJ file-t használja inputként. Az output file-t automatikusan **file-nev**.EXE-nek nevezi el.
LINK **file-nev**,,;	Ez a rövidített parancs arra készteti a szerkesztőprogramot, hogy a **file-nev**.OBJ file-t keresse inputként, és készítse el a **file-nev**.EXE és **file-nev**.MAP output file-okat.
Megjegyzés:	Az előbbi parancsban szereplő vesszők hatására nem jelennek meg a szerkesztőprogram üzenetei, és minden egyes opció az alapértelmezésként megadott file-nevet veszi fel. Ha valamelyik opciót el akarjuk hagyni, a vessző helyett a NUL parancsot kell behelyettesítenünk.

B–3. táblázat Szerkesztési parancsok rövidítése

ben, hogy az áthelyezési művelet lehetővé váljon. A jelenlegi példánkban a GAME.EXE file mérete 524 byte.

A .COM kiterjesztésű file azonban csak egyetlen, 64 kbyte-os szegmenset tartalmazhat, amely magában foglalja a STACK,DATA (adat) CODE (kód) információkat. Ha az assembly program kis méretű, 64 K bőségesen elegendő az adat, az utasítás és a veremmanipulációk tárolására.

Az .EXE file-tól eltérően a .COM file nem helyezhető át a memóriában, és a 0100H címnél kell kezdődnie. A .COM file lényegesen kisebb méretű mint, az .EXE file.

A .COM file-ok elkészítése

Az EXE2BIN.EXE kiszolgálóprogram futtatásával az .EXE file-ból .COM-ot készíthetünk.

Írjuk be tehát a következőket:

```
B>A:EXE2BIN GAME GAME.COM
```

majd nyomjuk le az ENTER gombot.

Feltételezve azt, hogy az EXE2BIN program az A meghajtóban van, az A:EXE2BIN parancs elindítja a programot. Jelen esetben a GAME elnevezésű .EXE file-t kell konvertálni. (Vegyük észre, hogy a GAME után nem szükséges az .EXE kiterjesztést megadni.) A keletkező .COM file neve GAME.COM lesz. Ebben az esetben azonban lényeges a file-kiterjesztés megadása, ellenkező esetben ui. az EXE2BIN program .BIN kiterjesztést fűzi a file-névhez. Így azonban a file nem lenne futtatható és a RENAME (átnevezés) paranccsal meg kellene változtatnunk a kiterjesztését.

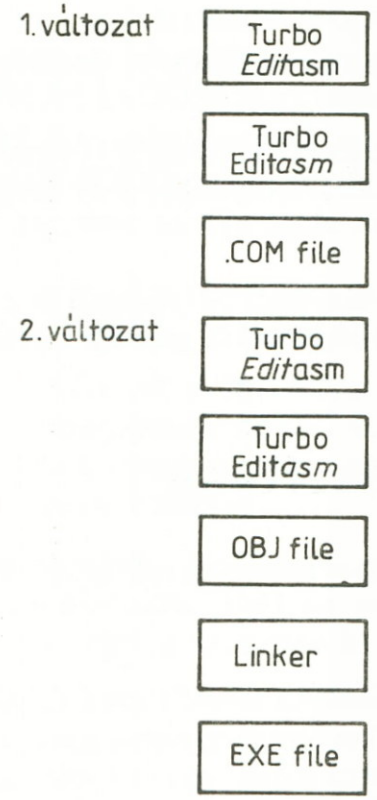
Most pedig hasonlítsuk össze a GAME.EXE és a GAME.COM file-ok méretét. A következőkhöz hasonlót kell látnunk a képernyőn:

```
GAME  EXE      524          1-02-86          7:45p
GAME  COM      12           1-02-86          7:45p
```

Figyeljük meg a két file-méret közötti lényeges különbséget.

A TURBO EDITASM PROGRAM HASZNÁLATÁNAK ISMERTETÉSE

A C melléklet célja az, hogy a TURBO EDITASM program különleges tulajdonságait bemutassa. A fordítási eljárás különböző lépéseit - a forráskód generálásától egészen a futtatható program létrehozásáig) egy mintafeladat segítségével ismertetjük. A C.1. ábrán az assembly kód készítésének két lehetséges módja látható.



C–1. ábra Az assembly program létrehozásának kétféle módja

Általános információk

A TURBO EDITASM program teljes képernyős szövegszerkesztővel rendelkezik. Az IBM és a MICROSOFT Makroassemblertől eltérően – amelyeknek ASCII file-okat létrehozó önálló szövegszerkesztő programra van szükségük – a TURBO EDITASM program az ASCII szövegszerkesztőt az assembleren belül valósítja meg. Ezáltal lehetővé válik, hogy a fordítási folyamatból ahhoz a kódsorhoz térjünk vissza, ami a hibát generálta. A TURBO EDITASM program ezen kívül még számos hasznos tulajdonsággal rendelkezik, mint pl. a beépített keresztreferencia kiszolgálóprogram vagy a sorkonkénti fordítás lehetősége.

A TURBO EDITASM olyan utasításkészletet tartalmaz, ami a 80286-os mikroprocesszor

mindkét üzemmódját (valós, védett) támogatja, valamint értelmezi a 8087-es és 80287-es matematikai társprocesszorok utasításait is, beleértve az ötféle lebegőpontos adattípust, a BCD és a LONG egészeket, és annak a lehetőségét, hogy mind a MICROSOFT, mind a 8087-es adattípusokat ugyanabba a forrásfile-ba foglaljuk.

A TURBO EDITASM programmal mind .EXE, mind .COM file-ok létrehozhatók. Ha a program lefordítása során a kódszegmens előtti adatszegmens elhelyezésével problémák jelentkeznek, az adatszegmenst a kódszegmens után helyezzük el.

Az assembly forráskód létrehozása

Az assemblyben megoldandó feladat nagyon egyszerű lesz. Egy üzenetet jelenítünk meg a monitor képernyőjén, amelyben a korábban már ismertetett DOS megszakítást alkalmazzuk majd.

Első feladatunk tehát a forráskód létrehozása. A TURBO EDITASM programot tartalmazó lemezt – amire a továbbiakban TASMB-ként fogunk hivatkozni – helyezzük be az A meghajtóba, a munkalemezünket pedig a B meghajtóba. Ha az A az alapértelmezés szerinti meghajtó, írjuk be a következőt:

```
TASMB
```

mire a következőket látjuk a képernyőn:

```
TURBO EDITASM          Ver 1.03B
                              PC-DOS
Copyright (C) 1984, 1985 by SPEEDWARE
Assem Source     Edit Source     Get Source     Write Source
Run Codefile     Hexdump File    Kill File      List File
Symbol List      Xrefer List     Directory      New Drive – Directory
Asm Options      Value      Quit
65278 Byte(s) Available.
    0 Byte(s) Used.
(A)
```

A kiemelt betűk az utasítások rövidítésének módját jelentik. A képernyő alján látható (A) betű jelzi azt a helyet, ahol a TASMB utasítások beírhatók.

Írjuk be az E betűt (Edit Source = forrásprogram szövegszerkesztése). Egy üres képernyőt fogunk látni, amelynek a jobb felső sarkában a következő információ jelenti, hogy szerkesztő üzemmódban vagyunk:

```
Line 1     Col 1     Insert
```

Írjuk be a következő kódot pontosan úgy, ahogy itt a könyvben szerepel.

```
PAGE ,132                    ;AZ OLDAL MERETENEK BEALLITASA

MYDATA  SEGMENT PARA 'DATA'
BLOCK   DB      'A 80286/80386 HATEKONY MIKROPROCESSZOROK'
        DB      '$'
MYDATA  ENDS


MYCODE  SEGMENT PARA 'CODE'     ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
```

```
        ASSUME  CS:MYCODE,DS:MYDATA
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATSZEGMENS HELYENEK BETOLTESE
        MOV     DS,AX           ;AX TARTALMAT DS-BE KERUL

        LEA     DX,BLOCK        ;A BLOCK HELYENEK BETOLTESE DX-BE
        MOV     AH,9H           ;PARAMETERBEALLITAS
        INT     21H             ;AZ UZENET MEGJELENITESE

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
        END     MYPROC          ;A PROGRAM VEGE
```

A TASMB szövegszerkesztőjének utasításai hasonlóak ahhoz, amit a jól ismert WordStar szövegszerkesztő program használ. A kínálat természetesen nem olyan bőséges (DEL billentyű = karakterek törlése, INS billentyű = karakterek beszúrása, Kurzormozgató gombok = kurzor pozíciónálás).

Ha a beírással elkészültünk, tároljuk a forráskódot. Nyomjuk le az F2 billentyűt, amire a képernyő bal felső sarkában a következő üzenet jelenik meg:

```
Write Source File:
```

Válaszoljunk erre a

```
B:PROGEXE
```

beírásával, és nyomjuk le az ENTER billentyűt. A TASMB automatikusan az .ASM file kiterjesztését támogatja. A program lefordításához a CTRL-K-D gombok lenyomásával ki kell lépnünk a szövegszerkesztőből, és az A billentyű lenyomásával aktivizálnunk kell a fordítóprogramot. A képernyőn a következő üzenetet pillanthatjuk meg:

```
Use File:B:PROGEXE.OBJ (Y/N) ?
```

Az Y lenyomásával egy .OBJ kiterjesztésű file jön létre, a PROGEXE elnevezéssel. A TASMB két menetben végzi el a fordítást, amelyről a következő üzenetekkel tájékoztat bennünket:

```
Assembling
Pass One
Pass Two
```

Feltételezve azt, hogy a program beírásakor nem hibáztunk, a megjelenő üzenet a következőhöz lesz hasonló:

```
  25 Source Line(s), No Assembly Error(s).
  12 Object Byte(s),54716 Byte(s) Free.
  Assembly Time: 1 second(s)
(A)
```

Assembly opciók

A TASMB igen könnyen használható assembly opciókat szolgáltat. Ha a főmenüből az O beírásával a Options tételt választjuk, akkor a megjelenő almenüből kijelölhetjük azokat a szolgáltatásokat (keresztreferencia- és szimbólumtábla előállítása, képernyőkijelzés, lemezre írás stb.), amire szükségünk van. Az almenü listája a következő:

```
* Editasm Options *
  F1 – Screen     (ON)
  F2 – Printer    (OFF)
  F3 – Symbols    (OFF)
  F4 – Xrefer.    (OFF)
  F5 – Memory     (OFF)
  F6 – ErrWait    (OFF)
  F7 – OBJ File   (OFF)
  F8 – COM File   (OFF)
  F9 – LST File   (OFF)
  F10– Undefd.    (OFF)
Select option or <CR> to Exit.
```

.OBJ file-ok létrehozása

Az .OBJ file-ok szerkesztésével jön létre a futtatható .EXE file, ami olyan kiegészítő fejrészt tartalmaz, amelynek segítségével a kód a rendelkezésre álló memórián belül áthelyezhető lesz. Az .OBJ file létrehozásához a TASMB főmenüjéből az O opciót, majd a megjelenő almenüből az F7 billentyű lenyomásával a .OBJ opciót kell választanunk. Az ENTER billentyű megnyomásával a főmenübe térhetünk vissza. Az A (Assemble) betűt kell most beírnunk, mivel a kódot le akarjuk fordítani. A TASMB egy file-nevet ajánl fel a számunkra:

```
B:PROGEXE.OBJ (Y/N)?
```

Az Y betű lenyomása a file-név elfogadását jelenti.

Győződjünk meg arról, hogy a TASMB valójában elkészítette az .OBJ file-t. A D (Directory) opció segítségével listát kérhetünk a lemezen szereplő file-okról. A képernyőn a D billentyű lenyomása után a

```
PROGEXE.ASM
PROGEXE.OBJ
```

file-neveket pillanthatjuk meg. (Persze csak akkor, ha eredetileg egy üres lemezt használtunk a TASMB kipróbálásához.)

A folyamat következő lépése az .OBJ file-on a szerkesztési művelet végrehajtása, amire a listafile tárgyalása után térünk vissza.

Listafile készítése

Hibakereséskor gyakran hasznos, ha rendelkezünk a lefordított programról is egy listával, amelynek létrehozásához a TASMB főmenüjéből az Asm opciót, majd a megjelenő almenüből az F9 billentyűhöz tartozó funkciót kell választanunk. Az Asm opcióhoz tartozó menüből való kilépés az ENTER gomb lenyomásával lehetséges. A program lefordításához az A betű beírására van szükségünk. A TASMB program két üzenetet is küld. Az elsőben a lefordított kódokat tartalmazó file nevének jóváhagyását várja tőlünk:

Use Object File:B:PROGEXE.OBJ (Y/N)?

a másodikban pedig a listafile nevére tesz ajánlatot:

Use File:B:PROGEXE.LST (Y/N)?

A PROGEXE.LST file listázásához a TASMB szövegszerkesztője, vagy a DOS TYPE utasítása egyaránt felhasználható. Amennyiben hardcopy-t is szeretnénk készíteni, ne felejtsük el bekapcsolni az F2 opciót, amelynek hatására a rendszer a listát a nyomtatóra is kiküldi.

A PROGEXE.ASM file lefordított kódokat is tartalmazó .LST file-ja a következőhöz lesz hasonló:

```
                                PAGE ,132                       ;AZ OLDAL MERETENEK BEALLITASA

0000                            MYDATA  SEGMENT PARA 'DATA'
0000  41 20 38 30 32 38         BLOCK   DB      'A 80286/80386 HATEKONY MIKROPROCESSZOROK'
      36 2F 38 30 33 38
      36 20 48 41 54 45
      4B 4F 4E 59 20 4D
      49 4B 52 4F 50 52
      4F 43 45 53 53 5A
      4F 52 4F 4B
0028  24                                DB      '$
0029                            MYDATA  ENDS


0000                            MYCODE  SEGMENT PARA 'CODE'      ;KODSZEGMENS DEFINIALASA AZ MASM-NEK
0000                            MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
                                        ASSUME  CS:MYCODE,DS:MYDATA
0000  1E                                PUSH    DS              ;A DS REGISZTER ELMENTESE
0001  2B C0                             SUB     AX,AX           ;AX TORLESE
0003  50                                PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
0004  B8 ---- R                        MOV     AX,MYDATA       ;AZ ADATSZEGMENS HELYENEK BETOLTESE
0007  8E D8                             MOV     DS,AX           ;AX TARTALMAT DS-BE KERUL

0009  8D 16 0000 R-                     LEA     DX,BLOCK        ;A BLOCK HELYENEK BETOLTESE DX-BE
000D  B4 09                             MOV     AH,9H           ;PARAMETERBEALLITAS
000F  CD 21                             INT     21H             ;AZ UZENET MEGJELENITESE

0011  CB                                RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
0012                            MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
0012                            MYCODE  ENDS                    ;A MYCODE ELNEVEZESU KODSZEGMENS VEGE
                                        END                     ;A PROGRAM VEGE
```

```
25 Source Line(s), No Assembly Error(s).
63 Object Byte(s),54619 Byte(s) Free.
Assembly Time: 1 Second(s)
```

C–2. ábra Programlista

Szimbólumtábla és keresztreferencia-lista készítése. A szimbólumtábla és a keresztreferencia-lista a program nyomon követéséhez és hibakereséshez használható, ui. ezek segítségével a forráskód minden szimbóluma, valamint az a sor, ahol a szimbólum szerepelt, listázható. Az F3 opcióval a szimbólumtáblát, az F4 opcióval a keresztreferencia-listát tudjuk elkészíteni. A PROGEXE.ASM program szimbólumtáblája a következőhöz lesz hasonló:

```
Assembling
Pass One                                                    2:51:39
Pass Two
p.m.
Segs & Groups

Name            Size            Align           Line            Combine         Class
MYCODE          0012            PARA            8               NONE            'CODE'
MYDATA          002D            PARA            3               NONE            'DATA'

Symbols:

Name            Type            Value           Line            Attr
BLOCK           Byte            0000            4               MYDA
MYPROC          F Proc          0000            9               MYCO            Lenght = 0012

25 Source Line(s), No Assembly Error(s).
63 Object Byte(s),54575 Byte(s) Free.
Assembly Time:

(A)
```

Ahhoz, hogy a keresztreferencia-listát is megjeleníthessük, a TASMB főmenüjéből az Xrefer List opciót kell választanunk az X betű beírásával. A lista hasonló lesz a következőhöz:

```
Name            Cross Refernce(s)
BLOCK           4,17
MYCODE          8,23
MYDATA          3,6,14
MYPROC          9,22

    6 Symbol(s) Used
28672 Byte(s) for Symbols.
28671 Byte(s) for Workarea.
```

.EXE file-ok készítése

Az .OBJ file-ból bármelyik MICROSOFT-tal kompatibilis szerkesztőprogrammal elkészíthető a .EXE futtatható programváltozat. A szerkesztőprogram látja el a programot azzal a kóddal, amely által a program a rendelkezésre álló memórián belül áthelyezhető lesz.

A szerkesztőprogram használatához ki kell lépnünk a TASMB programból a főmenü opciójával, és be kell helyeznünk az A meghajtóba a szerkesztőprogramot tartalmazó lemezt. Ha az .OBJ file a B meghajtóban van, és az alapértelmezés szerinti meghajtó az A, írjuk be a következőket:

```
A>LINK
```

A program egy üzenettel jelentkezik be:

```
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Object Modules [.OBJ]:
```

Írjuk be annak az .OBJ file-nak a nevét, amit szerkeszteni akarunk. Például: **B:PROGEXE**. Az .OBJ kiterjesztést nem szükséges kiírni. A szerkesztőprogram a következő háromsoros üzenettel válaszol:

```
Run File [B:PROGEXE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
```

Az alapértelmezés szerinti paraméterek elfogadásához egyszerűen csak az ENTER billentyűt kell lenyomnunk. (Az A és a B mellékletben részletesen tárgyaltuk az IBM és a MICROSOFT szerkesztőprogramok működését.)

A következő hibaüzenetre lehetünk figyelmesek:

```
Warning: no stack segment
```

Ez csupán csak egy figyelmeztetés arra vonatkozólag, hogy nem definiáltunk veremszegmenst. (Ennél a programnál ui. nem volt szükség veremszegmensre.)

A futtatható .EXE file tehát elkészült, amiről meggyőződhetünk, ha a B meghajtó file-jait listázzuk. Ha a B meghajtó a TURBO EDITASM program használata előtt üres volt, a következő listát kell kapnunk:

```
PROGEXE.ASM
PROGEXE.OBJ
PROGEXE.EXE
```

A B meghajtón létrehozott .EXE file futtatásához csupán a program nevét kell beírnunk:

```
A>B:PROGEXE
```

A következő üzenet jelenik meg a képernyőn:

```
a 80286/80386 hatekony mikroprocesszor
```

.COM file-ok készítése

Mielőtt hozzálátnánk a .COM file elkészítéséhez, a forráskódnak a megfelelő formátumban kell lennie. Vessünk egy pillantást arra a PROGEXE.ASM file-ra, amiből az .EXE file-t létrehoztuk. Vegyük észre, hogy két különálló kódszegmenst használtunk fel, az egyiket az adat, a másikat a program tárolásához. A programok szegmentálá- sával a 80286-os számítógépek .COM file-jaira vonatkozó 64 kbyte-os korlátozást fel tudjuk oldani.

A következő listán szereplő forráskódot, mivel csak egyetlen szegmenst használ, .COM file-ba fordíthatjuk mind a 80286-os, mind a 80386-os processzor esetében. A .COM file gyorsabban fut, mint az ennek megfelelő .EXE változat, két ok miatt:

1. Mivel csak egy szegmensről van szó, nincs szükség a szegmenscímek időrabló kiszámítására.
2. A .COM file nem helyezhető át a memóriában, tehát nem tartalmazza azt a programfejet, ami az áthelyezéshez szükséges.

Mivel ezáltal a program rövidebb, gyorsabban is fut. A PROGCOM.ASM forráskód a .COM file-ra vonatkozó korlátozásokat testesíti meg. A TASMB teljes képernyős szövegszerkesztőjét felhasználva írjuk be a PROGCOM.ASM programot pontosan úgy, ahogy itt szerepel.

```
        LEA     DX,BLOCK        ;A BLOCK HELYENEK BETOLTESE DX-BE
        MOV     AH,9H           ;PARAMETERBEALLITAS
        INT     21H             ;AZ UZENET MEGJELENITESE

        INT     20H             ;VEZERLES VISSZAADASA A DOS-NAK

BLOCK   DB      'A 80286/80386 HATEKONY MIKROPROCESSZOROK'
        DB      '$'

        END     MYPROC          ;A PROGRAM VEGE
```

A PROGCOM.ASM file kevesebb sort tartalmaz, mint a PROGEXE.ASM, mivel a szegmensdeklarációk .COM file esetén szükségtelenek. A .COM file elkészítéséhez a TASMB opciói közül válasszuk az O-t.

```
* Editasm Options *
  F1 – Screen     (ON)
  F2 – Printer    (OFF)
  F3 – Symbols    (OFF)
  F4 – Xrefer.    (OFF)
  F5 – Memory     (OFF)
  F6 – ErrWait    (OFF)
  F7 – OBJ File   (OFF)
  F8 – COM File   (OFF)
  F9 – LST File   (OFF)
  F10– Undefd.    (OFF)
Select option or <CR> to Exit.
```

A .COM file opció kiválasztásához az F8, majd az ENTER billentyűk lenyomása szükséges. Bizonyosodjunk meg arról, hogy az F7-es opció nincs aktivizálva, ui. a PROGCOM.ASM nem olyan forrásprogram, amiből .EXE file-t lehetne készíteni. Az opció kiválasztása után a képernyőn a következő üzenet jelenik meg:

```
Use File: B:PROGCOM.COM (Y/N)?
```

Az Y lenyomásával fogadhatjuk el a felajánlott file-nevet. A képernyőn ezután a következőket látjuk:

```
Assembling
Pass One

Pass Two

11 Source Line(s), No Assembly Error(s).
55 Object Byte(s),54783 Byte(s) Free.
Assembly Time: 1 second(s)
```

A B meghajtóról készített katalógusban a következő két file-névnek is szerepelnie kell:

```
PROGCOM.ASM
PROGCOM.COM
```

A .COM file futtatásához csak a program nevét **(PROGCOM)** kell beírnunk, amire a képernyőn az éppen érvényes kurzorpozíciótól kezdve a következő üzenet jelenik meg:

```
A 80286/80386 hatekony processzorok
```

A TASMB fordítási opciói

A megelőző példákból már láthattuk, hogy a TASMB számos opcióval rendelkezik. Az opciókat tartalmazó listát az O betű beírásával jeleníthetjük meg a képernyőn. A lista hasonló lesz a következőhöz (néhány ON/OFF beállítás természetesen eltérhet):

```
* Editasm Options *

  F1 – Screen     (ON)
  F2 – Printer    (OFF)
  F3 – Symbols    (OFF)
  F4 – Xrefer.    (OFF)
  F5 – Memory     (OFF)
  F6 – ErrWait    (OFF)
  F7 – OBJ File   (OFF)
  F8 – COM File   (OFF)
  F9 – LST File   (OFF)
  F10– Undefd.    (OFF)

Select option or <CR> to Exit.
```

A C–1. táblázatban a fordítási opciók magyarázatát adjuk meg.

Funkcióbillentyű	Az opció leírása
F1	Képernyő – Az opció aktivizálásával a fordítási lista megjelenik a képernyőn. Ha F1 kikapcsolt (OFF) állapotban van, akkor csak a fordítás során kiderített hibákat látjuk a képernyőn. Ha F1-et kikapcsoljuk, a fordítóprogram sokkal gyorsabban fut, mivel nem kell minden egyes lefordított sor képernyőre írásához a fordítási folyamatot leállítani.
F2	Nyomtató – Az opció bekapcsolásakor a lefordított assebly kódot a nyomtató kinyomtatja. A képernyőre és a nyomtatóra a lista nem küldhető ki egyidejűleg.
F3	Szimbólumok – Ez az opció arra utasítja a fordítóprogramot, hogy az assembly forráslista végéhez hozzákapcsolja a szimbólumtáblát is, amit aztán a kiválasztott eszközzel (F1, F2) kinyomtathatunk.
F4	XREF – Az F4-es opció aktivizálásával arra késztethetjük az assemblert, hogy minden egyes felhasznált szimbólumnak elkészítse a keresztreferencia-listáját azzal a forráskóddal együtt, ahol a szimbólumokra utalás történt. A lista arra a kiviteli eszközre kerül, amit az F1 vagy F2 opciókkal kiválasztottunk.
F5	Memória – Az F5 opció bekapcsolásával a TASMB a program a gépi kódú verzióját közvetlenül a memóriába küldi, amit aztán a TASMB főmenűjének RUN parancsával futtathatunk. (Megjegyzés: Az opció működéséhez INT20H-t kell a forráskódba elhelyeznünk oda, ahonnan vissza akarunk térni a TASMB programba.)
F6	Errwait – Ez az opció hosszú listák esetén hasznos, ui. ennek hatására a fordítóprogram minden hibánál, amit a forrásprogramban talál, várakozni fog. Ez idő alatt a programozónak lehetősége van arra, hogy visszaugorjon abba a sorba, ami a hibát okozta, és kijavítsa – az ESC billentyű lenyomásával –, vagy folytassa a fordítást az ENTER billentyű lenyomásával.
F7	OBJ FILE – Ennek az opciónak a hatására készül el az .OBJ file. (L. az .OBJ file-ok készítése című fejezetrészt.)
F8	COM FILE – Ennek az opciónak a hatására készül el a .COM file. (L. a .COM file-ok készítése című fejezetrészt.)
F9	LST FILE – Ezzel az opcióval határozhatjuk meg, hogy készüljön-e listafile, ill. hogy a lista a szimbólumtáblába és/vagy a keresztreferencia-listába bekerüljön-e? Ha az F9 bekapcsolt állapotban van, a TASMB bekéri a listafile nevét. Pl. a PROGEXE.ASM program esetén a TASMB a következő üzenettel jelentkezik be: *Use File: PROGEXE.LST (Y/N)?* Ha az F3 (szimbólumtábla opció) bekapcsolt helyzetű, a listát a szimbólumtábla követi. Hasonlóan, ha az F4 opciót kiválasztottuk, a keresztreferencia-lista is hozzákapcsolódik a forrásprogram listájához.
F10	Jelenleg nincs funkciója

C–1. táblázat Asm opciók

D MELLÉKLET

ASCII KARAKTEREK

A D–1. táblázat a karakterek ASCII kódját sorolja fel

DEC	OCTAL	HEX	ASCII
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL
8	010	08	BS
9	011	09	HT
10	012	0A	LF
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	SPACE
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	—
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C

DEC	OCTAL	HEX	ASCII
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	^
95	137	5F	—
96	140	60	‘
97	141	61	a

DEC	OCTAL	HEX	ASCII
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

D–1. táblázat ASCII karakterkódok

E MELLÉKLET

A KÖNYVTÁRKEZELŐ PROGRAM HASZNÁLATA

Az IBM Makroassembler (2.0-s és ennél későbbi változatok) és a MICROSOFT Makroassembler (3.0-s és ennél későbbi változatok) programok mindegyike rendelkezik könyvtárkezelő programmal, aminek az a feladata, hogy a gyártó cég vagy felhasználó által létesített könyvtárak létrehozását és karbantartását segítse. Ebben a mellékletben egy példán keresztül mutatjuk be, hogyan kell egy már meglevő könyvtárhoz eljárást hozzákapcsolni. A 7. fejezetben a könyvtár karbantartására több egyszerű példát is láthattunk. Érdemes ezt a fejezetet még egyszer áttekinteni, mielőtt ennek a mellékletnek a tanulmányozását folytatnánk.

A feladat leírása

A 9. fejezetben az IBM kiszolgálóprogramokat tartalmazó könyvtárát (IBMUTIL.LIB) hívtuk segítségül a 80287-es társprocesszor kimeneti formátumának kialakításához. Ez a könyvtár számos konverziós rutint használ, amelyek közül a $I8_OUTPUT-ot alkalmaztuk a fejezet példáiban. Mivel a $I8_OUTPUT az eredményt nem a kívánt formában adta vissza, további kiegészítő programra is szükség volt. A program kódja emiatt 60 sornál hosszabb lett. Erre a kiegészítő rutinra a 9. fejezet mindegyik programjában szükségünk volt, tehát jogos az az igény, hogy ezzel ne terheljük a forráskódot, hanem helyezzük el a kiszolgálóprogramokat tartalmazó könyvtárba. Ebben a mellékletben ezt a kódot fogjuk hozzáilleszteni az IBM kiszolgálóprogramokat tartalmazó könyvtárhoz a könyvtárkezelő program segítségével.

Megjegyzés: Mind az IBM, mind a MICROSOFT Makroassembler rendelkezik könyvtárkezelő programmal, azonban csak az IBM szállítja IMBUTIL.LIB-et az MASM lemezen.

Információk a programról

A 9. fejezetben található programok meglehetősen hosszúak, és nagy részük az eredmény formátumának kialakításával foglalkozik. Az E–1. és E–2. ábrán látható programok nagyon hasonlóak ehhez. Az E–1. ábrán egy olyan program listája látható, ami két valós szám összegét képezi a 80287-es társprocesszor felhasználásával. Az E–2. ábra programja alakítja át az E–1. ábrán látható program eredményét a megfelelő formátumba. A korábbi esetben E–1. és E–2. egyetlen programban szerepelt, most pedig a formátumot kialakító programrészt a kiszolgálóprogramokat tartalmazó könyvtárba helyezzük el. Igy a későbbiekben, ha egy szám normál alakjára van szükségünk, elegendő a kiszolgálóprogramokat tartalmazó könyvtári rutint meghívni.

Mielőtt ebben a fejezetben továbbhaladnánk, tanulmányozzuk át mégegyszer az IBM kiszolgálóprogramokat tartalmazó könyvtárának használatát a 9. fejezetben és az IBM Makroas-

sembler kézikönyvében, különös tekintettel a változók, adatszegmensek és kódszegmenesek PUBLIC deklarációjára.

```
COMMENT  /EBBEN A PROGRAMBAN A 80287-ES TARSPROCESSZORT VALOS
        SZAMOK OSSZEADASAHOZ HASZNALJUK. AZ EREDMENY NORMAL
        ALAKBAN TORTENO MEGJELENITESEHEZ MINIMALIS MERETU
        PROGRAMFEJ SZUKSEGES. AZ IBMUTIL KONVERZIOS RUTINOKAT
        TARTALMAZO KONYVTARAT A SZERKESZTES ELOIRASAKOR KELL
        MEGADNUNK. AZ AKTUALIS KONVERZIOS RUTIN A PRTREAL-T
        HASZNALJA. EZ A RUTIN A 80287-ES FORMATALT SZAMOT
        KARAKTERLANCCA ALAKITJA AT.AZ EREDMENY AZ EPPEN ERVENYES
        KURZORPOZICIOTOL KEZDVE JELENIK MEG.
PAGE ,132
.8087
COMMENT /A VALOS SZAMOK A KOVETKEZO ALAKUAK LEHETNEK:
                123.4567        0.000048976
                1.3E20          -4.5789E-3


        PUBLIC  RESULT          ;EZT A VALTOZOT HIVJA A KONYVTAR
MYDATA  SEGMENT PARA PUBLIC 'DATA';EZERT PUBLIC-NAK KELL LENNIE
NUMBER1 DQ      1.2345E21       ;VALOS SZAM
NUMBER2 DQ      -13.456789E20   ;VALOS SZAM
RESULT  DQ      ?               ;AZ EREDMENY 80287-ES FORMATUMBAN VAN
MYDATA  ENDS

CODEGRP GROUP   MATHCODE
MATHCODE SEGMENT BYTE PUBLIC 'CODE'  ;KODSZEGMENS DEFINIALASA AZ
                                     ;MASM-NEK
MYPROC  PROC    FAR             ;AZ ELJARAS NEVE MYPROC
        ASSUME  CS:MATHCODE,DS:MYDATA,ES:MYDATA
        EXTRN   PRTREAL:NEAR    ;AZ EXTERNAL KONYVTAR MIATT SZUKSEGES
        PUSH    DS              ;A DS REGISZTER ELMENTESE
        SUB     AX,AX           ;AX TORLESE
        PUSH    AX              ;ZERUS RAHELYEZESE A VEREMRE
        MOV     AX,MYDATA       ;AZ ADATOK HELYENEK BETOLTESE AX-BE
        MOV     DS,AX           ;AX TARTALMANAK ATADASA DS-BE ES ES-BE
        MOV     ES,AX


        LEA     SI,RESULT       ;A VALTOZO HELYENEK BEKERESE
        FLD     NUMBER1         ;A VALOS SZAM RATOLTESE A VEREMRE
        FADD    NUMBER2         ;ES EGY SZAM HOZZAADASA
        FSTP    QWORD PTR [SI]  ;AZ EREDMENY VEREMROL EGY VALTOZOBA
                                ;KERUL
        FWAIT                   ;SZINKRONIZALAS

        CALL    PRTREAL         ;AZ EREDMENY FORMATUMANAK KIALAKITASA

        RET                     ;A VEZERLES VISSZAADASA A DOS-NAK
MYPROC  ENDP                    ;A MYPROC ELNEVEZESU ELJARAS VEGE
MATHCODE ENDS
        END     MYPROC          ;A PROGRAM VEGE
```

E–1. ábra Két valós szám összegének számítása

```
        PUBLIC PRTREAL
MYDATA  SEGMENT PARA PUBLIC 'DATA'  ;AZ ADATSZEGMENSNEK PUBLIC-NAK KELL
                                    ;LENNIE
        EXTRN   RESULT:QWORD        ;EZ AZ A VALTOZO, AMIT ATADUNK
PADDER  DB      20 DUP (' ')        ;HELYFOGLALAS AZ IBM RUTIN SZAMARA
ANSWER  DB      17 DUP (?),'$'      ;AZ IBM RUTINBOL VISSZAKAPOTT
                                    ;KARAKTEREK TAROLASI HELYE
FIRDIG  DB      ' ','$'             ;AZ ELSO SZAMJEGY HELYENEK BIZTOSITASA
POWER   DW      ?                   ;A KITEVO TAROLASI HELYE
TBUFF   DB      4 DUP(' ')          ;4 BYTE A KITEVO KARAKTEREI SZAMARA
SIGN    DB      '-$'                ;NEGTIV ELOJEL
POINT   DB      '.$'                ;DECIMALIS PONT
EXP     DB      ' E $'              ;A KITEVO SZIMBOLUMA
MYDATA  ENDS

CODEGRP GROPU   LIBSEG
LIBSEG  SEGMENT BYTE PUBLIC  'CODE'     ;KODSZEGMENS DEFINIALASA
        ASSUME:CS:CODEGRP,DS:MYDATA
        EXTRN   $I8_OUTPUT:NEAR ;A RUTIN KULSO KONYVTARBAN VAN
PRTREAL PROC    NEAR            ;AZ ELJARAS NEVE PRTREAL
        CALL    $I8_OUTPUT
        SUB     DX,01H          ;A KITEVO CSOKKENTESE 1-GYEL
        MOV     POWER,DX        ;DX AZ ANS VALTOZO KORRIGALT KITEVOJE
        CMP     BL,'-'          ;AZ EREDMENY + VAGY - ?
        JNE     NONEG
        LEA     DX,SIGN         ;HA NEGATIV, A '-' KARAKTER
                                ;MEGJELENIETESE
        MOV     AH,09
        INT     21H
NONEG:
        CLD                     ;A KAR.LANC ANS-BA HELYEZESENEK
                                ;ELOKESZITESE
        MOV     CL,17           ;A KAR.LANC 17 BYTE HOSSZU
        LEA     DI,ANSWER       ;A MOZGATAS IRANYA. SI A FORRASRA
                                ;MUTAT
REP     MOVSB                   ;ATHELYEZES
        MOV     CL,1            ;ANSWER ELSO SZAMJEGYE FIRDIG-BE KERUL
        LEA     SI,ANSWER[1]
        LEA     DI,FIRDIG
REP     MOVSB
        LEA     DX,FIRDIG       ;AZ ELSO SZAMJEGY KINYOMTATASA
        MOV     AH,09
        INT     21H
        LEA     DX,POINT        ;A DECIMALIS PONT MEGJELENITESE
        MOV     AH,09
        INT     21H
        LEA     DX,ANSWER[2]    ;AZ ANSWER VALTOZO TOVABBI RESZENEK
                                ;MEGJELENITESE
        MOV     AH,09
        INT     21H
        LEA     DX,EXP          ;AZ 'E' BETU KINYOMTATASA
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A DX-BEN LEVO SZAM ATALAKITASA
                                ;KARAKTERLANCBA
```

```
        CMP     DX,8000H        ;POZITIV VAGY NEGATIV
        JB      POSIT           ;HA POZITIV, UGRAS POSIT-RA
        LEA     DX,SIGN         ;HA NEGATIV, AKKOR '-' JELET NYOMTATUNK
        MOV     AH,09
        INT     21H
        MOV     DX,POWER        ;A NEGATIV HEXA SZAM KORREKCIOJA
        XOR     DX,0FFFFH
        ADD     DX,01
POSIT:  MOV     CX,0
        LEA     DI,TBUFF        ;A TBUFF 4 BYTE-OS KARAKTERES TAROLASI
POWER1: PUSH    CX              ;HELYET BIZTOSIT A DX-BEN SZEREPLO
        MOV     AX,DX           ;HEXA SZAMOK DECIMALISSA ALAKITASABAN
        MOV     DX,0            ;A KEPERNYORE NYOMTATAS ELOKESZITESE
        MOV     CX,10
        DIV     CX
        XCHG    AX,DX
        ADD     AL,30H          ;A SZAM ATALAKITASA ASCII SZAMMA
        MOV     [DI],AL         ;ES ELMENTESE TBUFF-BA
        INC     DI              ;UJ HELY KIJELOLESE TBUFF-BAN
        POP     CX
        INC     CX              ;CX-BEN A SZAMJEGYEK DARABSZAMA
                                ;SZEREPEL
        CMP     CX,0
        JNZ     POWER1
PRIT:   DEC     DI              ;AZ ERTEKEK MEGJELENITESENEK
                                ;ELOKESZITESE
        MOV     AL,[DI]         ;TBUFF-BOL EGY SZAMJEGY BEKERESE
        PUSH    DX              ;DX EREDETI TARTALMANAK ELMENTESE
        MOV     DL,AL           ;A SZAM ELHELYEZESE A NYOMTATASHOZ
        MOV     AH,2            ;A DOS MEGSZAKITASHOZ SZUKSEGES PARAM.
        INT     21H             ;A HATVANY A KEPERNYORE KERUL
        POP     DX              ;DX EREDETI TARTALMANAK VISSZAALLITASA
        LOOP    PRIT            ;A RUTINT ADDIG ISMETELJUK, AMIG
                                ;VALAMENNYI SZAM MEGJELENIK
        RET
PRTREAL ENDP                    ;A PRTREAL ELNEVEZESU ELJARAS VEGE
LIBSEG  ENDS                    ;A LIBSEG ELNEVEZESU KODSZEGMENS VEGE

        END     PRTREAL         ;A PROGRAM VEGE
```

E–2. ábra Az outputot kialakító program

A könyvtárkezelő program használata

Az E–1. és E–2. ábrán látható programok mindegyike lefordítható a megfelelő MASM programmal. Amint sikeresen lefordítottuk az E–2. ábra programját, hozzákapcsolhatjuk a kiszolgálóprogramokat tartalmazó könyvtárhoz. A könyvtárkezelő program a következő öt opció megadását várja:

Könyvtárfile
Oldalméret
Műveletek
Listafile
j könyvtár

A műveletre vonatkozó szintaxis a következő:

```
LIB lib_file [oldal_meret][muveletek],[lista_file],[uj_konyvtár]...[;]
```

Az IBM MASM kézikönyvében a rendszerüzenetekről részletes leírást találunk.
A jelenlegi példánkban a könyvtár az IMUTIL.LIB név alatt már létezik. Ha az IBM MASM program a C meghajtón van, írjuk be a következőt:

```
C>LIB

IBM Personal Computer Library Manager
Version 1.00
(C) Copyright IBM Corp. 1984

(C) Copyright Microsoft Corp 1984

Library name:IBMUTIL
Operations:
List File:C:CONTENTS
```

Ennél a műveletnél a könyvtárkezelő program beolvassa az IBMUTIL tartalmát, és ugyanezen a meghajtón egy listafile-t képez a CONTENTS név alatt. A listafile egy ASCII file, amit a következők szerint lehet kinyomtatni:

```
$I4_I8          ifconv          $I4_M4          bfconv
$I8_I4          ifconv          $I8_INPUT       i8fin
$I8_M8          bfconv          $I8_OUTPUT      i8fout
$I8_TMUL        i8tmul          $I8_TPWR10      i8tmul
$M4_I4          bfconv          $M8_I8          bfconv

bfconv          Offset: 200     HCode and data size: F0
  $I4_M4          $I8_M8          $M4_I4          $M8_I8

ifconv          Offset: 400H    Code and data size: C0
  $I4_I8          $I8_I4

i8fin           Offset: 600H    Code and data size: 2FD
  $I8_INPUT

i8fout          Offset: C00H    Code and data size: 1A2
  $I8_OUTPUT

i8tmul          Offset: 1000H   Code and data size: 1EA
  $I8_TMUL        $I8_TPWR10
```

A könyvtárkezelő program segítségével az E–2. ábrán látható program .OBJ változatát hozzákapcsolhatjuk a kiszolgálóprogramokat tartalmazó könyvtárhoz. (Csak az .OBJ file-ok csatlakoztathatók a könyvtárhoz, ezért volt szükség a program lefordítására.)

A program csatlakoztatását elvégző rutint a következő utasítással hívhatjuk meg:

```
C>LIB IBMUTIL.LIB+PRTREAL.OBJ
```

Ellenőrizzük a művelet sikerességét:

```
C>LIB

IBM Personal Computer Library Manager
Version 1.00
(C) Copyright IBM Corp. 1984
(C) Copyright Microsoft Corp 1984

Library name: IBMUTIL
Operations:
List File: C:NEWINFO
```

A listafile-t a NEWINFO néven a C meghajtón tárolta a rendszer. Ha a NEWINFO-t a képernyőre kilistázzuk, a következőket kapjuk:

```
$I4_I8          ifconv          $I4_M4          bfconv
$I8_I4          ifconv          $I8_INPUT       i8fin
$I8_M8          bfconv          $I8_OUTPUT      i8fout
$I8_TMUL        i8tmul          $I8_TPWR10      i8tmul
$M4_I4          bfconv          $M8_I8          bfconv
PRTREAL         PRTREAL

bfconv          Offset: 200H    Code and data size: F0
  $I4_M4          $I8_M8          $M4_I4          $M8_I8

ifconv          Offset: 400H        Code and data size: C0
  $I4_I8          $I8_I4
  $I8_INPUT

i8fout          Offset: C00H    Code and data size: 1A2
$I8_OUTPUT

i8tmul          Offset: 1000H   Code and data size: 1EA
  $I8_TMUL        $I8_TPWR10

PRTREAL         Offset: 1400H   Code and data size: DB
  PRTREAL
```

Ha a PRTREAL csatlakoztatása sikeres volt, akkor a rutin eléréséhez csak az IBMUTIL előírására lesz szükség a szerkesztési folyamat során.

```
C>LINK

IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Object Modules [.OBJ]:ADDER
Run File [ADDER.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:IBMUTIL
```

A könyvben bemutatott rutinok legtöbbje a most ismertetett módon hozzákapcsolható a könyvtárhoz. A könyvtárkezelő hatékony programfejlesztést tesz lehetővé.

Készült a Somogy Megyei Nyomdaipari
Vállalat kaposvári üzemében – 181708
Felelős vezető: MIKE FERENC igazgató

392,– Ft

80386/80286

Osborne McGraw-Hill