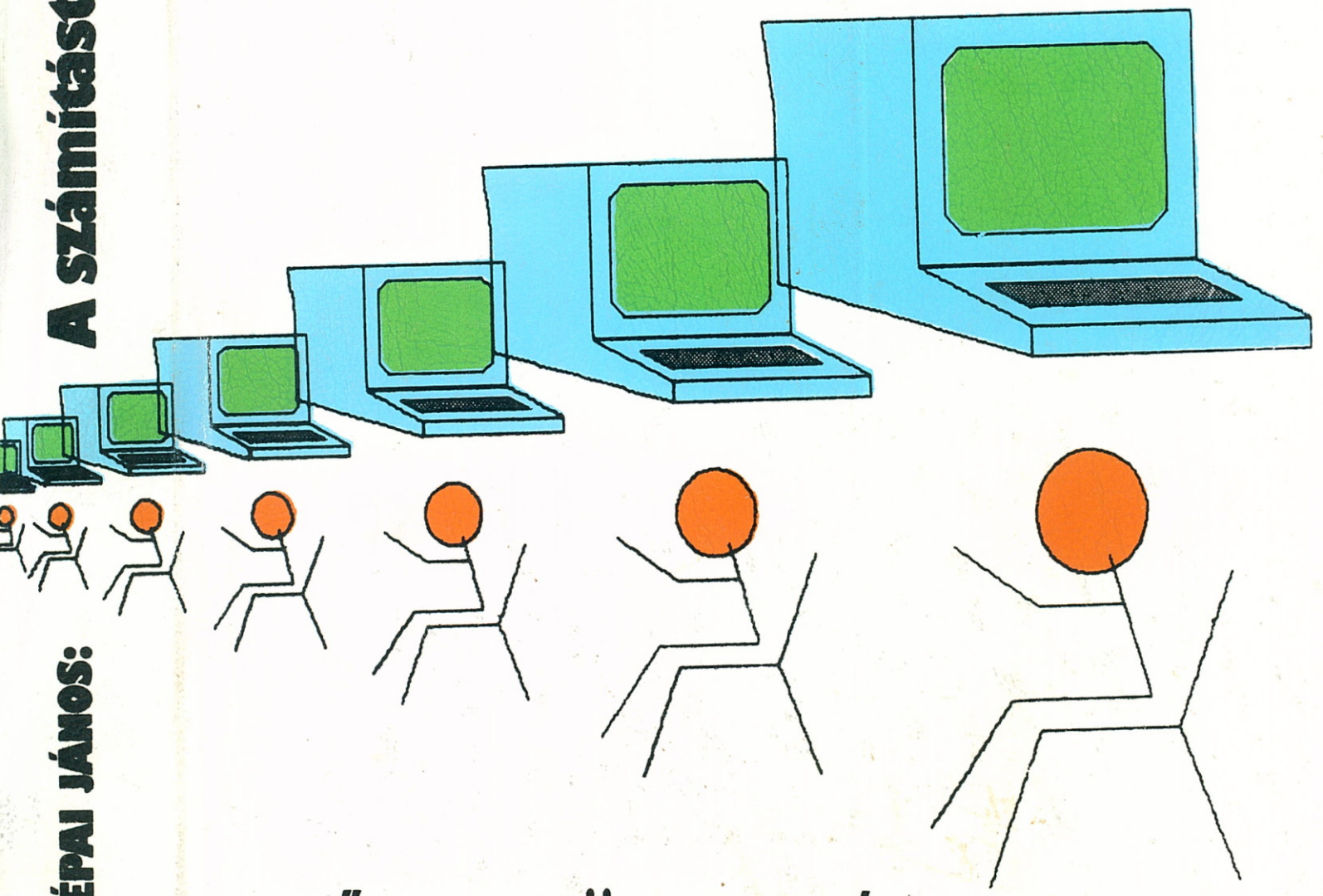
CSÉPAI JÁNOS

A számítástechnika alapjai



CSÉPAI JÁNOS: A számítástechnika alapjai

MŰSZAKI KÖNYVKIADÓ

CSÉPAI JÁNOS

A számítástechnika alapjai

MŰSZAKI KÖNYVKIADÓ, BUDAPEST, 1985

Csépai János: A számítástechnika alapjai c. tankönyv
változatlan kiadása

Lektorok:

GARÁDI JÁNOS
DR. QUITTNER PÁL

Szerkesztő:

DR. BALÁZS BÉLA

ISBN 963 10 6738 6

ETO 681.3 (0753)

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató
Felelős szerkesztő: Tarr Kálmánné

85.0488 Kossuth Nyomda, Budapest
Felelős vezető: Bede István vezérigazgató

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő és kötéstervező: Dornizs László
A könyv ábráit rajzolta: Bokori István és Varga Árpád
A könyv formátuma: B5. Ívterjedelme: 33,625 (A5)
Ábrák száma: 166. Papír minősége: 80 g ofszet
Betűcsalád és -méret: New Times garmond/12
Azonossági szám: 90051
MŰ: 3776—i—8588
Készült az MSZ 5601 és 5602 szerint

Tartalomjegyzék

IV. FEJEZET

Adatábrázolás a számítógépben 64

V. FEJEZET

Központi tár, központi egység 78

VI. FEJEZET

Programozási nyelvek, fordítóprogramok 100

VII. FEJEZET

Input-output egységek, háttértárak 128

VIII. FEJEZET

Fájlszerkezetek 158

IX. FEJEZET

Operációs rendszer 177

X. FEJEZET

Számítóközpont 204

ANNOTÁCIÓS RÉSZ 217

Előszó

A mikroszámítógépek hazai megjelenése és rohamos terjedése új távlatokat nyitott a hazai számítástechnikai alkalmazások területén. Ma már minden magyar középiskola rendelkezik legalább egy iskolaszámítógéppel, és nem ritka, hogy otthon a családban is számítógéppel játszik a gyerek vagy dolgozik a szülő. A professzionális személyi számítógépek teljesítőképessége alkalmas arra, hogy kisebb vállalatok is használják feladataik megoldásához.

Ugyanakkor a népgazdaság legkülönbözőbb területein, a nagyvállalatoktól kezdve a tervezés és az irányítás intézményein át a népességnyilvántartásig, a nagyszámítógépek továbbra is fontos szerepet játszanak.

Ez a tankönyv, amely a gimnáziumok III. osztályában fakultatív tárgyként oktatott számítástechnika tárgyhoz készült — az MM ide vonatkozó tanterve alapján —, alapvetően nagygépes ismereteket tartalmaz. Az Egységes Számítógép Rendszerhez tartozó gépek alapján kíván olyan **számítástechnikai alapokat** biztosítani, amelyek lehetővé teszik, hogy mindazok a tanulók, akik számítóközpontokban számítástechnikai középkáderként kívánnak elhelyezkedni vagy számítástechnikai területen kívánnak továbbtanulni, megismerkedjenek ezeknek a gépeknek a felépítésével, működésük alapjaival.

A tankönyv ugyanakkor bevezetést nyújt a programozáshoz, a számítógépes programozás elő- és alapismereteinek tárgyalásával. Áttekintést ad a mikroszámítógépekről is — a látókör bővítése céljából. Ez a tananyag a IV. osztályban szervezési és operátori ismeretekkel bővül.

Útmutató a könyv használatához

Ez a könyv elsősorban felépítésében tér el az eddig ismert hasonló munkáktól. Az eltérések azonban több vonatkozásban *sajátossá teszik használatának módszertani elemeit* is.

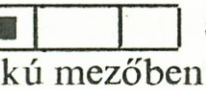
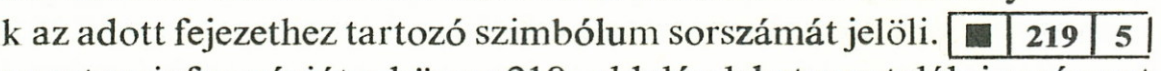
A könyv feldolgozásának leglényegesebb módszertani specifikuma az *annotációk alkalmazása.* A tanulási feladatok megoldásába ui. szervesen iktatódnak be a tananyag annotációs elemei, amelyekre a feldolgozás során lehet és — bizonyos esetekben — kell is támaszkodni. [Az annotáció legáltalánosabban: „1. rövid pár soros ismertetés; tartalmi kivonat 2. feljegyzés, széljegyzet." (Bakos Ferenc: Idegen szavak és kifejezések szótára.)]

A könyv fejezetekre tagolt két főrésze: *a tanulás új elemeit* tartalmazó és *az annotációs információkat* magába foglaló részek a 13—216., ill. a 217—317. oldalakon találhatók.

Az annotációs rész különböző típusú információi *szimbolikus jelek* segítségével határolódnak el egymástól. Ezek a szimbólumok *a tanulás új elemeit* tartalmazó részben is megtalálhatók, ahol *azt jelzik*, hogy a feldolgozás során *hol és milyen ismeretekkel összefüggésben* lehet ezeket az információkat felhasználni.

A szimbólumok jelentése a következő:

A **négyzet** ■ szimbólummal jelölt információk általában olyan ismereteket tartalmaznak, amelyekkel tanulmányaik során — esetenként más tantárgyak keretében — *már találkoztak*, de feltételezhetően vannak olyan tanulók, akik ezeket már vagy *elfelejtették*, vagy *nem pontosan emlékeznek tartalmukra.* Ennek megfelelően az olvassa el az információkat, *akinek szüksége van* ezek felidézésére.

A *négyzet*  szimbólumhoz — és a többi szimbólumhoz is — csatlakozó *téglalap* alakú mezőben két szám található. Ezek közül az első a tankönyv oldalszámát, a második az adott fejezethez tartozó szimbólum sorszámát jelöli.  pl. azt jelenti, hogy ezt az információt a könyv 219. oldalán lehet megtalálni a négyzet szimbólum fejezeten belüli 5. sorszámmal jelzett információjában.

A **csillag** ★ szimbólummal jelölt információk azt jelzik, hogy az adott fogalommal, kategóriával, tétellel tanulmányaik során már találkoztak, de ezeknek a jelölt kategóriától *eltérő volt a tartalma.* Az annotációban szereplő információ arra ad választ, *mi a különbség* a korábbi és a most megjelölt tartalom között, *mi tette szükségessé* az új tartalom kialakítását.

Ezeket az információkat azok veszik igénybe, akik ezekre a kérdésekre *nem tudnak válaszolni*. Az információkat célszerű az *otthoni* munka során tanulmányozni.

A **háromszög** ▲ szimbólummal jelölt információk ún. *segítő információkat* tartalmaznak. Ezeknek az a szerepe, hogy figyelembe véve a különböző tanulók tudása, felkészültsége, képességei közötti nagy eltéréseket, biztosítsa *minden tanuló* eredményes, teljes értékű tanulását. A segítő információk olyan ismeretek, amelyek felhasználásával az adott szinten legkisebb előképzettséggel, felkészültséggel rendelkező tanulók számára is lehetővé válik, hogy az e szimbólummal jelölt tételek, fogalmak, kategóriák lényegét, összefüggéseit mélyen megértsék és elsajátítsák.

A különböző bonyolult ismeretek elsajátításánál természetesen az veszi igénybe ezeket a mélyebb, pontosabb megértést segítő információkat, akiknek a könyv *otthoni tanulmányozása* alkalmával szüksége van tartalmuk ismeretére.

A **kör** ● szimbólummal jelölt információk a különböző ismeretek *elmélyültebb tanulmányozását*, különböző összefüggések feltárásával *mélyebb megértését* és *hatékonyabb elsajátítását* biztosítják. Ezeket az információkat *mindenkinek el kell olvasni*, és az otthoni tanulás munkájában a könyv első részében található különböző tételeket, fogalmakat, kategóriákat ezek tanulmányozásának eredményével kell kiegészíteni.

A **kettős kör** ⊙ szimbólummal jelölt információk tartalmát *a kötelező tananyagot meghaladó* ismeretek alkotják. Azok vehetik igénybe, akik egy-egy tétellel, fogalommal, művelettel, kategóriával összefüggésben *a kötelező tananyagnál többet* szeretnének tudni, akik *mélyebben szeretnének a számítástechnikával foglalkozni*. Az információk feldolgozását az otthoni önálló tanulás keretében kell elvégezni.

Az annotációs elemek módszertani sajátosságainak e rövid áttekintése is azt bizonyítja, hogy az annotációs elemek a tankönyvben ugyan *külön részben* szerepelnek, a tanulás konkrét folyamatában a tananyag első részével *egységet* alkotnak.

A könyv jellemző sajátossága az is, hogy az egyes fejezetek végén nemcsak különböző *kérdések és feladatok* találhatók, hanem biztosítva van a *válaszok és megoldások helyességének ellenőrzése* is. Ez a könyv 319—376. oldalain található információk segítségével történik. Arra kérjük azonban, hogy az itt található szöveget csak akkor olvassa el, ha *már önállóan válaszolt*, *önállóan megoldotta* a feladatot. Saját fejlődését akadályozná, ha ezek elolvasásával kezdené a munkát, hiszen gátolná önálló feladatmegoldó képességének a fejlődését. Természetesen *ellenőrzés céljából* el kell olvasni a helyes megoldásokat is, de csak az *önállóan elvégzett megoldások után*. Ezek alapján persze ki is kell javítani esetleges tévedéseit, biztosítani kell a helyes megoldások rögzítését.

A könyv használatának ezek a főbb sajátosságai természetesen csak *fokozatosan* valósulnak majd meg munkájukban, de érdemes ezeket a sajátosságokat is megismerni, és az ezekből fakadó követelményeknek eleget tenni. Ezek az ismeretek is *feltételei* annak, hogy a könyv a gyakorlatban is eredményesen töltse be funkcióját, segítse a számítástechnika alapjainak megismerését, sikeresen készítsen fel — más tudományterületekre is támaszkodva — a gyakorlatra.

Balázs Béla

I. FEJEZET

Számítógépek

Az embereket mindig izgatta, hogy milyen lesz a jövő. A XIX. század nagy álmodozói — a magyar Jókai Mór, a francia Jules Verne és sokan mások — izgalmas történeteket találtak ki, amelyek a jövőben játszódnak. Ha ezek a szerzők feltámadnának és körülnéznének a XX. század utolsó negyedének világában, zavartan vonnák össze szemöldöküket. Lépten-nyomon beleütköznének valamibe, amiről még legmerészebb történeteikben sem álmodtak. Ezt a valamit a mai emberek **számítógépnek** [★ | 219 | 1] nevezik. Ők azon csodálkoznának, hogy ezek az emberalkotta gépek mennyi mindent tudnak, sok ember pedig azon csodálkozna, hogy ez számukra miért olyan elképzelhetetlen.

Pedig mi magunk, a XX. század gyermekei is sokszor álmélkodva kapjuk fel a fejünket, amikor a számítógépek újabb és újabb térhódításáról kapunk híreket. Kétségtelen, hogy századunk sok kiemelkedő műszaki alkotása között különös jelentősége van a számítógépnek. Vannak tudósok, akik jelentőségét a termonukleáris energia felfedezésének jelentőségéhez hasonlítják.

A számítógépek kitörtek arról a hagyományos területről, ahová mindenki elképzeli őket. Szétfeszítették a matematikai alkalmazások falait és elkezdték meghódítani az emberi gondolkodás legkülönbözőbb területeit.

Jókai és Verne megdöbbenése éppen ebből táplálkozna. Verne pl. nagyon sok gépet és berendezést álmodott meg, de ezek mind valamilyen konkrét célt szolgáltak: tenger alatti utazást, holdbarepülést stb. Olyan gép, amely sok célra használható, Verne képzeletét is felülmúlja.

Hiszen számítógépek vezérlik az űrhajókat és a korszerű atomerőműveket. Működésük megoldhatatlan a számítógépek alkalmazása nélkül. Nap mint nap olvashatunk számítógép-vezérlésű gépekről, amelyek teljesítményét pontosság és gyorsaság tekintetében az ember nem képes elérni. A nyelvi kapcsolatok létrejöttének eddig nem ismert lehetőségei kezdenek feltárulni, mert úgy tűnik, hogy számítógépek segítségével lehetséges lesz az egyik nyelven megjelent szövegnek egy másik nyelvre való lefordítása.

Új utak lehetősége tárult fel az oktatásban, amikor kiderült, hogy számítógépek segítségével oktatni lehet.

A számítógépek alkalmazása átszövi a hétköznapi élet gyakorlatát is.

Hazánkban már az újszülöttet számítógépes nyilvántartásba veszik. Később

— felnőtt korában — számítógéppel tartják nyilván OTP-tartozásait, háztartásának energiafogyasztását, átutalási betétszámlájának alakulását stb.

Mindezek elismerést és csodálkozást váltanak ki belőlünk. Nem állhatunk meg azonban a csodálkozásnál.

A számítástechnikában eddig végbement fejlődés és különösen az utóbbi években megfigyelhető változások alapján nyilvánvaló, hogy a számítógépek használata rövid időn belül minden művelt embernek szükségletévé válik. Már napjainkban sem a szakemberek szűk csoportja dolgozik a számítógépekkel. A személyi számítógépek mindennapjaink használati tárgyává lesznek, nagy segítséget nyújtva különböző problémáink megoldásához, segítve az ember fejlődését. Ezért nem elégséges a számítógépek csodálata. Meg kell ismernünk működésüket, hiszen az álmélkodás még nem biztosítja, hogy használni is tudjuk őket. A puszta csodálat nem azonos az ismeretekkel, s ha itt megállunk, ellentmondás jön létre az ember mindennapi munkája és a számítógépek által nyújtott teljesítmény között.

Hogyan lehet megismerni a számítógépeket? A számítógépek elméleti és gyakorlati kérdéseivel a *számítógép tudomány* foglalkozik.

Mi a számítógép tudomány és milyen fő területei vannak? Találó megfogalmazását adta e fogalomnak egy amerikai professzor, amikor így írt: „A számítógép tudomány teljesen hasonló értelemben foglalkozik az információval, | ★ | 220 | 2 | mint a fizika az energiával; témája az információ ábrázolása, tárolása, kezelése és közlése... Ahogyan a fizika energiaátalakító eszközöket használ, úgy alkalmaz a számítógép tudomány információátalakító eszközöket." De mint ahogy a fizika sem csak az energiával foglalkozik, éppen úgy a számítógép tudomány sem csak az információval kapcsolatos problémákat öleli fel.

Ide tartozik a numerikus matematika területe, az automataelmélet, a matematikai logika, a programozási nyelvek elmélete és gyakorlata, a mesterséges intelligenciával foglalkozó tudományterületek. És természetesen ide tartozik a programozási rendszerek fejlesztése, valamint a logikai tervezés, a számítógép-tervezés, nem utolsó sorban a már említett terület: az információtárolás és -visszakeresés elméleti és gyakorlati kérdései. | ● | 222 | 3 |

A számítógép tudomány különböző területeit kell tanulmányoznia annak, aki a számítógépeket meg akarja ismerni. Az egyes területekkel való foglalkozás mértékét és sorrendjét az dönti el, hogy az illető milyen szempontból, milyen nézőpontból, milyen célból kíván vizsgálódni. Akit a későbbiekben pl. a mesterséges intelligencia filozófiai-társadalmi kérdései érdeklik, az más tanulmányokat fog folytatni, mint egy olyan személy, aki a számítógépek konkrét alkalmazásának kérdéseivel akar foglalkozni.

Számunkra a számítástudomány egy szűkebb területe fontos, az, amelyet összefoglalóan számítástechnikának neveznek.

A számítástechnika a számítógépek felépítésével, működésével és működtetésével, programozásával foglalkozik. Vizsgálódásainkat ebbe az irányba folytatjuk.

A SZÁMÍTÓGÉP ÁLTALÁNOS FOGALMA

Mi teszi lehetővé a korábban elképzelhetetlen teljesítményeket? Amint neve is mutatja, a számítógép létrejöttét az emberi ész felbecsülhetetlen értékű tevékenységének, a számolásnak a területén végbement radikális változásnak köszönheti. Az emberiség története folyamán a számolás igen nagy fejlődésen ment át. Viszonylag korán felismerték, hogy a különböző matematikai műveletek gépek segítségével is elvégezhetők — a *számolás gépesíthető.*

A sort — jelenlegi történelmi ismereteink szerint — az abacus (a golyós számvető) nyitotta meg még az ókorban. A XVII. században szerkesztett, majd a későbbiekben tökéletesített számológépek mechanikus elven működtek. A fejlődést itt is, mint oly sok más területen, a szükségletnek és a technikai megvalósíthatóságnak a kölcsönhatása alakította.

A XX. század második harmadában a technikai fejlődés megteremtette a *lehetőséget*, a felmerülő feladatok jellege pedig az *igényt* arra, hogy a számolás területén is megvalósulhasson annak *automatizálása*, ezzel minőségileg új utakat nyitva a számológépek történetében. [● | 222 | 4]

A magyar származású **Neumann János** [■ | 219 | 5] és mások (eleinte egymástól függetlenül zajló) kutatási eredményeiként olyan gép jött létre, amely úgy képes hosszú számítási folyamatok, különböző műveletek elvégzésére, hogy az indításhoz szükséges bemenőjelek hatására saját belső állapotát emberi beavatkozás nélkül változtatva, kimenőjelek formájában a folyamat végeredményét megadja. Úgy is megfogalmazhatjuk, hogy mivel saját állapotát folyamatos emberi beavatkozás nélkül képes megváltoztatni, a *számítógép automata.* [⊙ | 225 | 6] A minőségi különbséget funkciója szerint éppen az jelenti, hogy a különböző bonyolult műveletek végzéséhez *nincsen szükség állandó emberi beavatkozásra.* Ezt jól mutatja, ha összehasonlítjuk egy egyszerű asztali számológép működtetésével. Míg a számítógépnél a különböző matematikai műveletek végrehajtásakor nincsen szükség az ember közreműködésére, az asztali számológépnél egy-egy matematikai művelethez szükséges operandusokat [▲ | 221 | 7] billentyűzet segítségével egyenként kell bevinni a számológépbe, utána meg kell nyomni a műveletet kiváltó billentyűt, meg kell nyomni az egyenlőségjelet is és csak ekkor jutunk a várt eredményhez.

Mi teszi lehetővé, hogy a számítógépeknél nincsen szükség emberi beavatkozásra és a gép képes akár több ezer különböző művelet automatikus elvégzésére? Döntően az, hogy a gép olyan tárral rendelkezik, amelynek az a sajátossága, hogy nemcsak a belső állapotváltozást jelentő műveletek végzéséhez szükséges adatokat (számokat) képes tárolni, hanem tárolja a megoldásra váró feladatok műveleteit leíró utasítások sorozatát is.

Egy feladat megoldását adó utasítások együttesét *programnak* nevezik. A *tárolt program* végrehajtása biztosítja, hogy a számítógép az emberi teljesítőképességet meghaladó automatikus működésével másodpercenként több százezer vagy millió műveletet végez el. A programot annak végrehajtása előtt természetesen el kell készíteni és

a gépben el kell helyezni. Egy megoldandó feladathoz tartozó program elkészítésének folyamatát *programozásnak* nevezik. A programozáshoz azonban sajátos *programozási nyelv* ismeretére is szükség van, és elengedhetetlen, hogy a programozó képes legyen a megoldandó feladatot véges számú részlépés sorozatára felbontani (algoritmizálni).

Ezek után megfogalmazhatjuk, hogy **mi is a számítógép.**

Számítógépnek nevezzük azt a műszakilag megvalósított rendszert, amely képes adatok tárolását, visszakeresését és feldolgozását (beleértve az adatbevitelt és az eredményközlést) emberi beavatkozás nélkül megvalósítani a számítógépben korábban elhelyezet programok működtetésével. ⊙ | 226 | 8

Hogyan valósulnak meg a számítógépeknek ezek a sajátosságai?

A számítástechnikában *erőforrásoknak* nevezik azokat az elemeket, amelyek a számítógépnek a program végrehajtására vonatkozó képességeit megvalósítják. Ezeket az erőforrásokat két nagy csoportba szokás sorolni. Az első csoportba tartozó erőforrásokat összefoglalóan hardvernek, a másodikba tartozókat szoftvernek nevezzük.

A **hardver** a számítógép műszakilag megalkotott, megépített technikai elemeinek összefoglaló neve, különböző egységekből áll.

A **szoftver** a programok és a programrendszerek összefoglaló neve. Ezek közvetlenül a hardver nyújtotta alaplehetőségekre támaszkodva vagy már meglevő szoftverre építve a számítógép használatának lehetőségeit bővítik. ● | 224 | 9

A számítógépek fejlődése során mind a hardver, mind a szoftver nagy fejlődésen ment át. Állandóan gyarapodott (és gyarapszik) a hardver, ill. a szoftver körébe tartozó eszközök, ill. programok fajtája, ugyanakkor egyre tökéletesebb és megbízhatóbb elektronikai egységekből épülnek maguk a hardver eszközök. Ez pedig visszahat a szoftver hatékonyságára is. Ennek megfelelően különböző számítógép generációkról beszélhetünk.

SZÁMÍTÓGÉP GENERÁCIÓK

Az első számítógépekben az elektronikus áramköröket elektroncsövek alkották. Az üzemeltető mérnökök alapvető feladata az üzemképes — mégpedig hosszú ideig üzemképes — berendezés biztosítása volt. Ezeket a számítógépeket sorolják ma az *első generációs* számítógépek közé. Ebben az időben maga a hardver jelentette a teljes számítógépet, ami azt jelenti, hogy a gép adottságai, „képességei" azonosak voltak a hardver „képességeivel".

A tranzisztorok megjelenése a számítógép építésében is jelentős változásokat hozott. A tranzisztorokból épített *második generációs* gépek üzembiztonsága már lényegesen javult, miközben a gép méretei is számottevően csökkentek. A második generációs gépekkel megjelennek az első szoftver termékek, amelyek egyszerűbbé teszik a programozást, és a gép működése során is segítik a felhasználót.

A *harmadik generációt* az integrált áramkörök megjelenése és a szoftver súlyának

további növekedése jellemzi. A korszerű számítógépek vételárában a rendszerrel szállított szoftver értéke meghaladja a teljes vételár 50%-át. Ugyancsak egyre több önálló — hardvertől független — szoftver termék jelenik meg a piacokon. Ha a számítógépek első, körülbelül 10 évét a hardver korszakának nevezzük, az azóta eltelt időről elmondható, hogy egyre inkább a szoftver korszakává válik.

A korszerű számítógépet igénybe vevő felhasználó gyakorlatilag nem tudja, hogy az általa használt számítógép hardverje mire képes. A felhasználó egy számára látszólagos számítógéppel áll szemben, amely a hardverre „épített" szoftver rétegek együtteséből áll. A hetvenes évek közepétől a számítástechnika fejlődésében — amely addig is meglehetősen dinamikus volt — egy olyan robbanás történt, amelynek hatására szinte elképzelhetetlen távlatok nyíltak meg. Ennek a robbanásnak az oka nagybonyolultságú elektronikus félvezető áramkörök (szuper integrált áramkörök), a VLSI-k [■ | 219 | 10] megjelenése volt. Ezek segítségével több tízezernyi tranzisztor teljesítménye néhány mm^2 nagyságú lapkába (chip-be) zsugorodott össze. Ez a félelmetes méretcsökkenés nem csupán az elképesztő miniatürizálást eredményezte (pl. a karórába építhető számológépet), hanem a megbízhatóság továbbnövekedését is, hiszen ezek a lapkák szinte nem tudnak elromlani.

Egyre több, korábban csak programmal megvalósított funkció „hardveresedik", ami azt jelenti, hogy programnyi utasítás-sorozatok tevékenységét egy vagy néhány lapka valósítja meg. Ez a végrehajtás sebességét lényegesen gyorsítja. Neumann János szobányi gépének képességeit messze meghaladó képességű számítógép ma egy íróasztalon elfér.

Új fogalom is bevonult a számítástechnikába: a **mikroszámítógép** vagy személyi számítógép, ez az aktatáskányi masina, amely a TV képernyőjéhez csatlakoztatható, jó minőségű kazettás magnetofont tud adat- és programtárolásra használni, a lakásban levő telefonvonalon keresztül más hasonló (vagy nagyobb) gépekhez tud csatlakozni. Egyszerűen programozható és játékra éppúgy felhasználható, mint a háztartási kiadások nyilvántartására, a családi költségvetés elkészítésére stb. A mikroszámítógépek megjelenésével (és alacsony árával) a számítástechnika látványosan bevonul a mindennapi életbe.

EGY KORSZERŰ SZÁMÍTÓGÉP STRUKTÚRÁJA

A számítógép automata jellegéből következik, hogy működése során a külvilágból bemenőjeleket észlel. Ezeket a jeleket (adatokat) a gép érzékeli, majd belső állapotát változtatva (valamiféle feldolgozást végezve) kimenőjeleket (eredményadatokat) állít elő. [▲ | 221 | 11] Ez a nagyon általános megfogalmazás minden számítógépes feladatmegoldásra értelmezhető, ezért azt mondhatjuk, hogy minden számítógépes feladatmegoldást **adatfeldolgozásnak** tekinthetünk, mindegyik felírható

B(evitel) — **F**(eldolgozás) — **E**(redményközlés)

képlettel.

A feldolgozás irányulhat az adatok tárolására, visszakeresésére vagy velük való műveletvégzésre. E három lehetőséget összefoglalóan *adatfeldolgozásnak* nevezzük. | ★ | 220 | 12 | A számítógépeknek olyan egységekkel kell rendelkezniök, amelyek megvalósítják az adatfeldolgozás lépéseit:

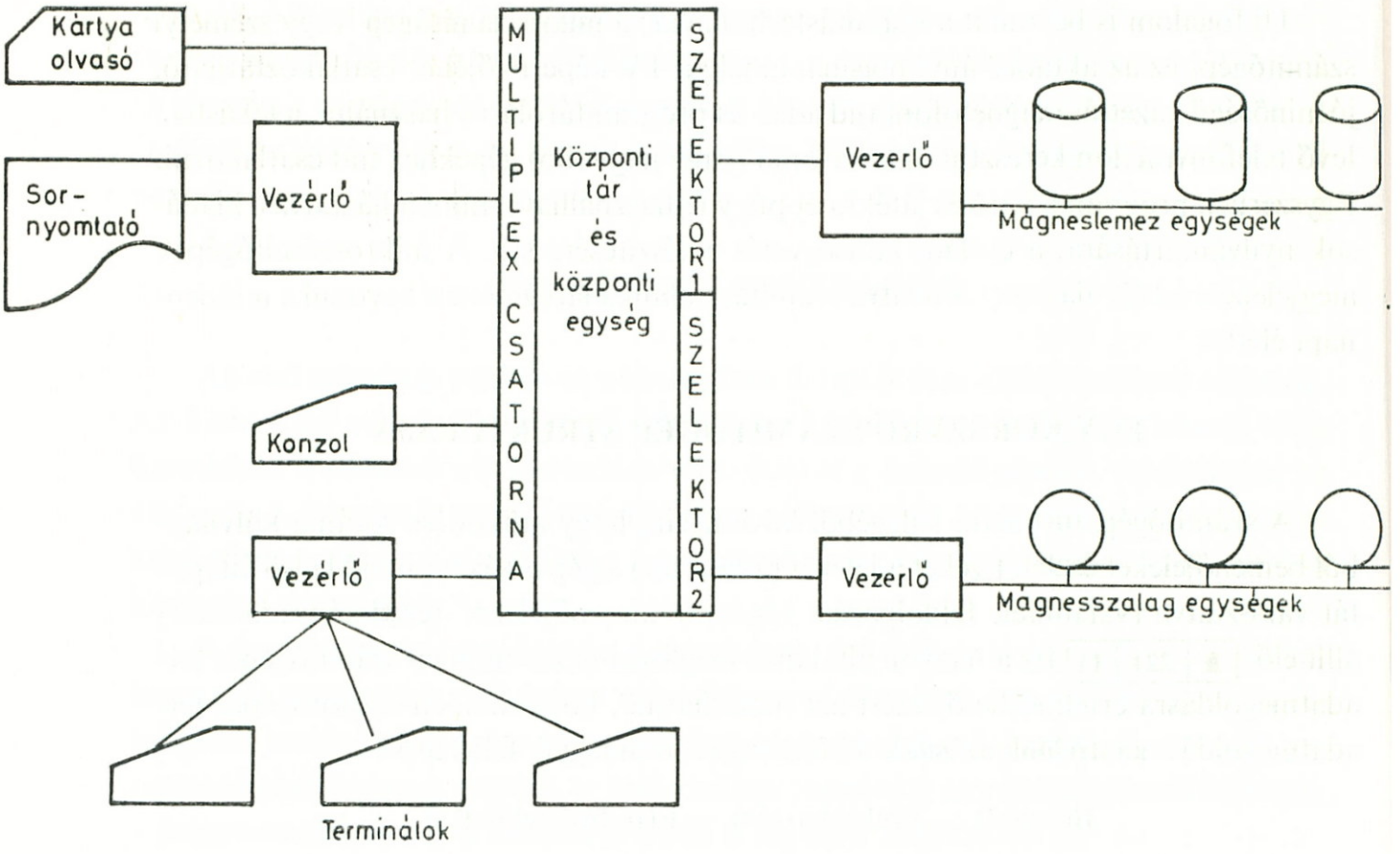
— az adatbevitelt,
— a feldolgozást,
— az eredményközlést.

Ha figyelembe vesszük, hogy a feldolgozandó adatok tömege olykor több százezer vagy millió is lehet, még egy fontos funkcióra kell alkalmassá tenni a számítógépet. Ez pedig

— a nagy tömegű adatok (hatékony) tárolása.

Hogyan tagozódik a hardver?

Egy korszerű számítógép hardverjének tagozódását vizsgálva látjuk, hogy mindezek a szükséges egységek megtalálhatók. Az adatbevitelt a beviteli (meghonosodott idegen szóval: input) egységek segítségével valósítjuk meg. A leggyakrabban használt adatbeviteli egységek: a lyukkártyaolvasó berendezés, a terminálok, a különböző mágneses input lehetőségek.



1. ábra. A számítógép struktúrája

A feldolgozást, beleértve a feldolgozásra kerülő és az eredményadatok tárolását a számítógép két alapvető egysége végzi el. Ezek:

— a központi tár,

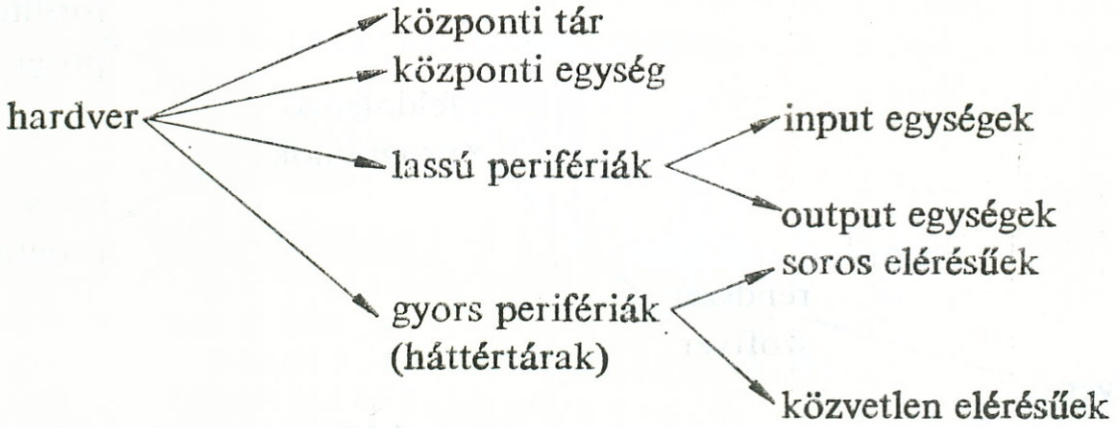
— a központi egység.

Az eredményközlés leggyakrabban használt eszköze a sornyomtató, de gyakran jelenítik meg az eredményeket (idegen szóval output adatokat) pl. függvényrajzoló berendezéssel vagy képernyős kijelzővel.

A nagy tömegű adattárolást szolgálja a mágnesszalag és a mágneslemez. Ezeket háttértáraknak is nevezik, mivel leggyakrabban olyan adatállományokat tárolunk rajtuk, amelyek a feldolgozás során teljes terjedelmükben soha nem szükségesek. Így lehetőségünk van, hogy a korlátozott méretű központi tárba csak a legfontosabb (vagyis a megoldás menetében éppen szükséges) adatokat tartsuk bent, a többi adat a „háttérből" kérhető, ha rájuk kerül sor. A háttértárakat, mivel viszonylag nagy sebességű adatátvitelt biztosítanak, szokás gyors perifériáknak is nevezni, szemben az adatbevitel és -kihozatal korábban említett eszközeivel, amelyeket összefoglalóan lassú perifériáknak hívunk.

A perifériák a központi egységhez csatornákon keresztül csatlakoznak. Az elmondottakat foglalja össze az 1. ábra.

A következő séma pedig összefoglalja a hardver tagozódását:



A soros elérés (gyakorlatban a mágnesszalagon levő adatok elérése) azt jelenti, hogy a fizikai rögzítés sorrendjében lehet csak az adatokat újraolvasni. A közvetlen elérés (gyakorlatban a mágneslemezről lehet megvalósítani), azt jelenti, hogy — megfelelő elérési technikák segítségével — az adatok a fizikai rögzítéstől eltérő sorrendben is hatékonyan visszakereshetők.

A számítástechnikai eszközök jelenlegi alkalmazási szintjén egyre szélesebb körben használatosak azok a berendezések, amelyek egy-egy nagy teljesítményű számítógéphez postai telefon- vagy telexvonalakon csatlakoznak. Az ilyen készülékeket terminálóknak vagy adatállomásoknak nevezzük. Közülük is említést érdemelnek azok, amelyek a számítógéppel párbeszédes kapcsolatot képesek fenntartani.

A felhasználó egy képernyős írógép-szerű készülék mellett ülve gépeli be parancsait és utasításait a számítógép számára, amely azokat kiértékeli és végrehajtja. Egy

ilyen „párbeszéd" eredményeképpen néhány óra alatt programok készíthetők, javíthatók és próbálhatók ki.

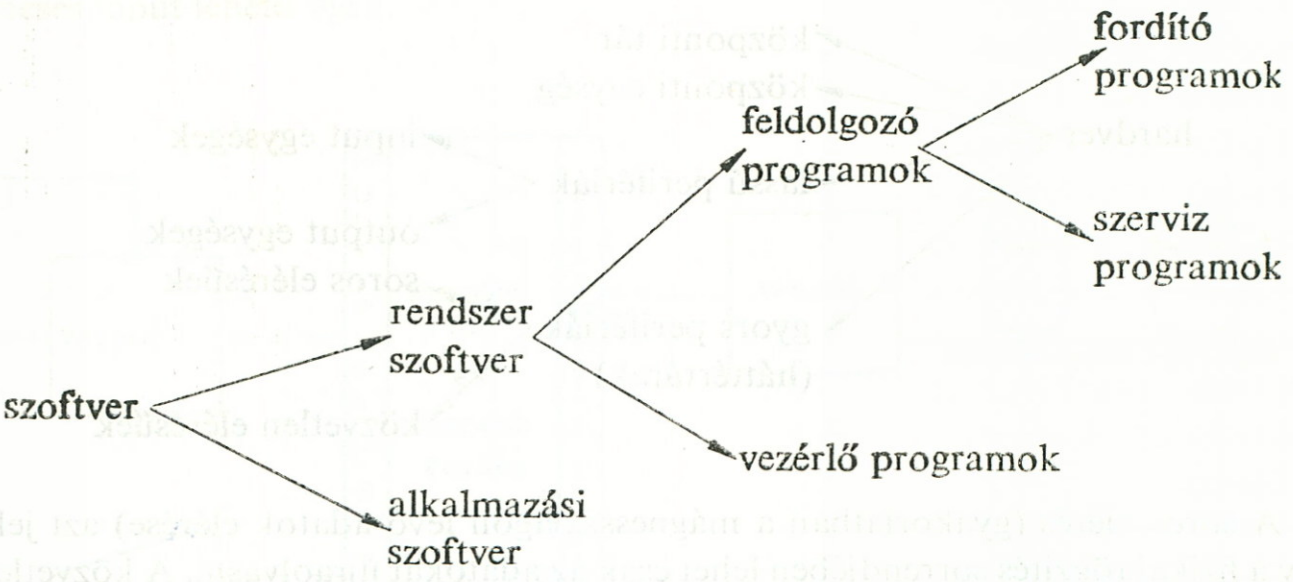
A mikroszámítógépek megjelenésével elterjedtek az „intelligens terminálok". Ezek olyan nagyszámítógéphez kapcsolt mikrogépek, amelyek önállóan is képesek működni, de adott esetben a nagygéphez kapcsolva folytathatják a működésüket.

Hogyan tagozódik a szoftver?

A hardver eszközök hatékony felhasználását az erőforrások másik csoportja, a szoftver biztosítja. A szoftver erőforrások legfontosabb feladatai közé tartozik:

— a hardver eszközökkel való gazdálkodás,
— a gép működésének felügyelete,
— a háttértárolók és azok adattartalmának karbantartása,
— a programozási munka megkönnyítése,
— ismétlődő feladattípusok megoldásához kész programok biztosítása.

A különböző fontosságú és nagyságú feladatok megoldását a szoftver különböző alkotóelemei végzik. Az elvégzendő feladatok funkciója szerint egy korszerű számítógép szoftverje a következő módon tagozódik.



A felhasználó számára a legközvetlenebb segítséget a feldolgozó programok biztosítják. Ez a segítség a programírás megkönnyítésében és az állandóan ismétlődő programok, ill. programelemek beépíthetőségében realizálódik.

A ma ismert programozási nyelvek száma több százra tehető, bár széles körben ezeknek csupán töredéke terjedt el.

Hány nyelven ért egy számítógép? Minden számítógép *alapvetően egyetlen nyelvet* ért, egyetlen nyelv utasításait képes közvetlenül végrehajtani: saját anyanyelvét, a **gépi kódot**.

A különböző típusú gépeknek eltérő gépi kódja van. Ezek az egymástól különböző kódok egyben mégis azonosak: számjegykombinációk sorozatából állnak, s

ezért felkészültség nélkül szinte érthetetlenek. Példaképpen vessünk egy pillantást egy gépi kódban írott programrészletre:

```
4754    00    2    00000
4734    00    1    00000
0560    00    2    00000
0500    00    1    00000
0040    00    0    01002
```

Ezek a számjegyoszlopok semmit nem árulnak el arról, hogy milyen utasításokat jelentenek. Megértése bonyolult, nehézkes tanulmányokat igényel.

A számítógépek széles körű elterjedését éppen ez a bonyolult, az ember köznapi gondolkodásától távol álló és idegen programozhatóság gátolta. Olyan közös nyelvre

```
      DIMENSION TOMB(50,50),SOR(50)
C
C  KETDIMENZIOS TOMB SOROSSZEGEINEK KISZAMITASA
C
C  A SOROK ES OSZLOPOK SZAMANAK BEOLVASASA
C
      READ(5,1) M,N
 1    FORMAT(2I3)
C
C  A SOR ELEMEINEK KINULLAZASA
C
      DO 2 I=1,N
        SOR(I)=0.
 2    CONTINUE
C
C  A TOMB OLVASASA - EGY SORA EGY REKORD
C
      DO 3 I=1,N
        READ(5,4) (TOMB(I,J),J=1,M)
 4    FORMAT(16F5.1)
 3    CONTINUE
C
C  A SOROSSZEGEK SZAMITASA
C
      DO 10 I=1,N
        DO 11 J=1,M
          SOR(I)=SOR(I)+TOMB(I,J)
 11     CONTINUE
 10   CONTINUE
C
C  AZ EREDMENY KIIRATASA
C
      WRITE(6,15) (SOR(I),I=1,N)
 15   FORMAT(1H1,// '  A SOROSSZEGEK:'(1H ,
     1 10F10.1))
      STOP
      END
```

2. ábra. FORTRAN nyelvű program

volt szükség, amely az emberi kifejezés és a gépi megértés közötti ellentmondást képes feloldani. Ma már a felhasználó a feladatot megoldó programot a gépi kódhoz képest *magas szintű nyelven* (pl. FORTRAN nyelven) írja meg. A számítógép szoftver erőforrásai között rendelkezésre áll egy **fordítóprogram**, amely működése során a FORTRAN nyelven írott programot elemzi, feldolgozza és végeredményben lebontja gépi kód szintre. Az így *lefordított* programot azután — bizonyos kiegészítések elvégzése után — a számítógép már közvetlenül végre tudja hajtani. Az említett ellentmondás feloldására az ember magát a számítógépet használta fel.

A magas szintű nyelvek emberközelségét szemlélteti a 2. ábra szerinti FORTRAN nyelven írott programrészlet, amely — némi angol nyelvtudással — szinte elolvasható.

A felhasználó számára más jellegű segítséget nyújtanak a *vezérlőprogramok*. Ezeket a programokat **operációs rendszernek** is szokás nevezni.

Kinek és milyen előnyöket nyújtanak a vezérlőprogramok?

Az operációs rendszer szolgáltatásokat nyújt a számítógép üzemeltetőjének: pl. automatikusan regisztrálja a program végrehajtása során elhasznált erőforrásokat, s ezek alapján — hónap végén — elvégzi a számlázást is.

A gépkezelő (operátor) az operációs rendszer segítségével bepillanthat a feldolgozás folyamatába: információkat kaphat a gép adott pillanatban végzett tevékenységéről, a hardver pillanatnyi állapotáról, a gépben levő munkák készültségi fokáról.

Ugyancsak az operációs rendszer jelzései segítik a gépkezelőt a géptermi munkában: pl. jelzi, ha egy periféria nincs bekapcsolva, ha egy mágnesszalag vagy mágneslemez tévesen lett feltéve stb. A gépkezelő az operációs rendszerrel az operátori parancsnyelv segítségével érintkezik.

A programozó az általa összeállított munka elvégzéséhez szükséges hardver és szoftver erőforrásigényt a munkavezérlő nyelv segítségével közli az operációs rendszerrel.

Az ember szerepe

Hogyan kapcsolódik össze az ember munkájában a hardver és a szoftver?

A jó hardver és szoftver önmagában még kevés a számítógép hatékony működéséhez. E két erőforráson kívül — amelyek az elvégzendő munka eszközei — a munkát végző személy, az ember is fontos „erőforrás". Egy asztali számológép vagy egy személyi számítógép működtetéséhez egyetlen ember elegendő. A nagyszámítógépeknek a gazdasági életben való alkalmazásához különböző munkakörökben dolgozó, különböző végzettségű szakemberekre van szükség. Természetesen ezek a munkakörök a számítógép megjelenése előtt nem léteztek. Új munkaterületekről van szó, amelyek többsége nagyon komoly szakmai felkészültséget kíván. Ebből is láthatjuk, hogy miközben a számítógépek sok rutin jellegű szellemi munka alól felszabadítják az embereket, ugyanakkor érdekes és sok tanulást igénylő munkalehetőséget biztosítanak az emberek számára.

A számítógépekhez kapcsolódó legfontosabb munkaterületeket a következő *csoportokba* sorolhatjuk:

A *rendszerszervezők* teremtik meg a kapcsolatot egy adott szakterület szakemberei és a számítógépes szakemberek között. Az ő feladatuk a problémák megértésén túl a számítógépes megoldás hatékony módszerének a kidolgozása. Megfogalmazzák a rendszer elé tűzött célokat, kijelölik a rendszer döntési pontjait, kijelölik az ehhez szükséges adatok körét és azok forrásait, megtervezik a feldolgozás menetét.

A *programozók* végzik el a rendszert alkotó programok megtervezését és kódolását. Munkájukhoz tartozik a programok tesztelése és a hibák javítása.

A *rendszerprogramozók* feladata a rendszer működéséhez szükséges szoftver generálása, karbantartása, fejlesztése. Fő feladatuk a szoftver javításával a számítógép hatékonyságának növelése.

Az *alkalmazási programozók* feladata a számítógéppel megoldani kívánt problémák programjainak elkészítése, ill. a felhasználók segítése ilyen jellegű munkájukban.

A számítóközpontokban ezek a munkaterületek sokszor nem válnak el élesen egymástól. A rendszerszervezők sokszor részt vesznek a programozási munkában, és a programozók is végezhetnek szervezési feladatokat.

Az *operátorok* (számítógép-kezelők) fontosabb feladatai:

— a programok bevitelének előkészítése,
— a vezérlőpulton keresztül a rendszer működtetése,
— az előírt rend fenntartása a gépteremben,
— az időnyilvántartások vezetése,
— a programok levétele a gépről,
— a fellépő hibák jelzése a karbantartók számára.

Bár nem tartoznak szorosan a gépkezelők közé, itt említjük az *adatrögzítőket*, akik a feldolgozandó adatokat rögzítik a számítógép számára elfogadható formában.

A *karbantartó mérnökök* és *műszakiak* a hibamentes hardver biztosításáért felelősek.

A rendszerszervezők és a programozók általában munkacsoportokban dolgoznak. Jó felkészültségük biztosítja a számítógép hatékony működését, mert a számítógép mindig csak azt végzi el, amire utasítást kap.

A számítógépek alkalmazásának kezdeti időszakában vita folyt arról, hogy lehet-e gondolkodó gépet készíteni. A vitát az a meggondolás is motiválta, ha ilyenek vannak vagy lesznek, nem fogják-e uralmuk alá vonni az emberi gondolkodást?

A számítógépek története azt bizonyítja, hogy ezek a gépek *nem az ember ellen, hanem az emberért* vannak. A fejlődés minden szakasza azt bizonyította, hogy minél fejlettebb számítógépek jöttek létre, annál magasabb szintet ért el az emberi alkotó és kreatív gondolkodás is. A számítógépek minden fejlődési szakasza szintén ennek az alkotó tevékenységnek a terméke. A tapasztalatok és az elméleti megfontolások azt bizonyítják, hogy mennél többre lesznek képesek a gépek, annál nagyobb lehetőségek nyílnak meg az ember alkotó gondolkodása előtt. Azért kell foglalkozni a számító-

gépek megismerésével, mert ez is egyik eszköze egyéni fejlődésünknek, személyiségünk kiteljesedésének, s ezzel együtt társadalmunk fejlődésének is. | ● | 224 | 13 |

Kérdések

1. Mi jellemzi az automatát?
2. Mit jelent a tárolt program fogalma?
3. Mi a számítógép?
4. Milyen számítógépes generációkat ismersz?
5. Milyen számítógépes erőforrásokat ismersz?
6. Mi a hardver?
7. Mi a szoftver?
8. Mi jellemzi a mikroszámítógépeket?
9. Milyen foglalkozási csoportokat ismersz a számítógép környezetében?
10. Mi az operátor (gépkezelő) feladata?

Feladatok

1. Az automata mosógép — mint a neve is mutatja — szintén felfogható automataként. Gyűjtsétek össze azokat a tulajdonságait, amelyek közösek a számítógéppel és olyanokat, melyekkel a számítógép rendelkezik, de az automata mosógép nem.

2. Tanáraitok segítségével írjátok le, hogy mi jellemezte az elektroncsöves, tranzisztoros és az integrált áramkörökkel készült rádiókat. Milyen közös vonást fedeztek fel a rádió és a számítógép fejlődésében?

3. Automatának tekinthető-e az ember?

4. Az annotációs részben leírt áruházi számítógépes rendszer kialakítása során — véleményed szerint — milyen sorrendben működtek közre a felsorolt szakemberek: programozók, áruházi szakemberek, rendszerszervezők, számítógépkezelők? Nyilakkal jelöld, hogy ki kivel került kapcsolatba a feladat megoldása során!

II. FEJEZET

Adatok, adatstruktúrák, adathordozók

Tudjuk, hogy a számítógép adatfeldolgozást végez. Az „adat" elnevezéssel a mindennapi életben is találkozunk: „A Központi Statisztikai Hivatal nyilvánosságra hozta a gazdaság és a társadalom múlt évi fejlődéséről szóló adatokat". „A tárgyaláson az ügyész bemutatta a terhelő adatokat". „A laboráns a műszerről leolvasta a mérési adatokat".

Az előbbi idézetekből is érezzük, hogy az adat szónak különböző tartalma van az egyes mondatokban. A számítógéptechnikában — mint a tudomány bármely területén — csak jól definiált fogalmakkal szabad dolgozni. Mi magunk is, ha a társadalomban meg akarjuk állni a helyünket, csak a pontosan végzett munkánkkal leszünk képesek ezt elérni. Ezért pontosan és egyértelműen meg kell fogalmaznunk, hogy **mit értünk adaton,** ill. — most már e fogalom pontos elnevezését használva — **elemi adaton. Milyen jellemzői vannak az adatnak, milyen a felépítése, hogyan kerül a számítógépbe és onnan hogyan kerül az emberhez?** Addig amíg ezekre a kérdésekre nem tudunk válaszolni, nem tudjuk a számítógépet eredményesen használni problémáink megoldásban.

MI AZ ELEMI ADAT?

Az elemi adat fogalmának tisztázásához az **információ** fogalmából induljunk ki. Köznapi értelemben információnak tekintünk olyan értesüléseket, amelyek előttünk ismeretlen vagy kevéssé ismert tényekről, múlt, jelen vagy jövőbeni történésekről tudósítanak. Az ilyen jellegű fogalmazás nem jár messze az információ egzaktabb megfogalmazásától, amely úgy foglalható össze röviden, hogy információnak tekintjük a *számunkra új ismereteket hordozó jelek tartalmi jelentését.* Az információnak fontos velejárója, hogy határozatlanságot (bizonytalanságot) szüntethet meg. [★ | 229 | 1] [▲ | 229 | 2]

Az elemi adat az információnak olyan megjelenített formája, amely kisebb egységekre már nem osztható fel anélkül, hogy az általa hordozott információ ne torzulna. E meghatározás két fontos elemet tartalmaz:

— az elemi adat megjelenített (rögzített) információ;

— ha bármit is elhagyunk az elemi adatból, az általa megjelenített információ torzul vagy elvész. [▲ | 230 | 3]

Arra a kérdésre, hogy az információ milyen jelsorozatok segítségével rögzíthető, a későbbiekben térünk vissza.

Ember és gép információcseréje

Vizsgáljuk előbb meg, hogy az ember—számítógép viszony milyen információ, tehát végeredményben milyen adatmozgáson alapszik. A számítógépnek a külvilággal való kapcsolata ui. — mint minden önmagában zárt rendszernél — kétirányú információáramlás útján valósul meg. Az ember közli a géppel a kívánságait, a számítógép pedig ezeket a kívánságokat kielégíti és eredményeket produkál, vagy elutasítja, miközben közli az elutasítás okát.

Az ember által közölt kívánságok rögzített jelsorozatok, amelyek jellegüket tekintve háromfélék lehetnek:

— vezérlő információk,
— programok,
— e programokkal kezelni kívánt adatok.

A számítógépbe bekerülő jelsorozatoknak ez a hármas tagozódása a felhasználó nézőpontjából adódik. A felhasználó *programot* készít annak érdekében, hogy a számítógép az általa megadott *adatokon* végrehajtott műveletek eredményeképpen az output adatokat előállítsa. Ennek a tevékenységnek az elvégzéséhez a számítógép igénybevételére vonatkozó *vezérlő információkat* is közölni kell a számítógéppel. A számítógép oldaláról vizsgálva, az adat dinamikusabb fogalom. A számítógép működése során ui. a különböző időpillanatokban más-más jelsorozatokon hajt végre különböző műveleteket. Ezt a tevékenységet a hardver valósítja meg. A hardver működése — ma már — mindig valamilyen szoftveren keresztül érvényesül; pl. a nem közvetlenül hardver szintű parancsokat valamilyen program bontja le a gépi berendezések „színvonalára". A számítógép szempontjából tehát az adat fogalma általánosabb, mint a felhasználó szempontjából, hiszen a számítógép a programot és a vezérlő információkat is kezelheti adatként. Hogy a számítógép a három típusú jelsorozatot hogyan tudja megkülönböztetni, erre a kérdésre későbbi fejezetekben térünk vissza.

Néhány példa

Tekintsünk néhány példát az elemi adat fogalmának jobb megértése érdekében. A következő megfogalmazások mindegyike valamilyen adatra vonatkozik anélkül, hogy megadnánk az adat számszerű mennyiségét.

„A beteg vérsüllyedése"
„A III. b osztály tanulmányi átlaga"
„A III. b osztály létszáma"
„Az iskola telefonszáma"

És egy olyan adat, amelynél „számszerű mennyiségről" nem beszélhetünk:

„Az állampolgár neve"

Ezek az adatok sok mindenben különböznek egymástól, pl. keletkezésük módjában. Az egyiket mérték, a másikat számították (más adatokból), a negyediket a posta valamilyen elv szerint kiosztotta, végül az utolsót az ember örökölte, kapta.

Van-e hasonlóság a felsorolt adatok között? Egyik hasonlóság lehet a véletlen számszerű egyezőség. Pl. a vérsüllyedés lehet 3,5 (milliméter), az osztály átlaga is lehet 3,5 (ez persze nem milliméter). Egy sokkal fontosabb hasonlóság, hogy mindegyik megfogalmazás a „valaminek a valamije" vagy a „valakinek a valamije" séma szerint történt. Ezt a hasonlóságot már nem a véletlen okozta.

Az elemi adat — és az általa hordozott információ — ui. mindig

— tartozik valakihez/valamihez, nevezzük ezt a valamit a továbbiakban egyednek;

— kifejez valamit (az egyedről), nevezzük ezt a továbbiakban az egyed tulajdonságának vagy röviden tulajdonságnak.

Egyetlen egyednek több tulajdonsága is lehet (pl. az állampolgár neve, az állampolgár születési éve), amit más-más adatok fognak kifejezni. Másrészt ugyanaz a tulajdonság több egyedhez is tartozhat (különböző állampolgárok születési adatai). A világban előforduló egyedek és a hozzájuk tartozó tulajdonságok száma végtelenül sok. Egy-egy adatfeldolgozási feladat megoldásának előkészítése során fontos feladat

— kiválasztani azoknak az egyedeknek a halmazát, amelyeket a feldolgozás érinteni fog (pl.: a népességnyilvántartás minden magyar állampolgárt érint);

— meghatározni a feldolgozás szempontjából szükséges tulajdonságok halmazát (pl. legmagasabb iskolai végzettség, foglalkozás stb., de nem szükséges: testmagasság, hajszín stb.).

Az így kialakított egyed- és tulajdonsághalmazokból létrehozható egy táblázat, a következő struktúrával.

tulajdonságok	1.	2.	3.	4.	...	k.	k+1	...	m
egyedek									
A.									
B.									
...									
K.									
L.									
...									
P.	*a táblázatban pedig a soroknak megfelelő egyednek az adott oszlopban levő tulajdonságra vonatkozó adata van.*								

A táblázatos elrendezésből az is észrevehető, hogy az egyes elemi adatok más elemi adatokkal (a velük egy sorban vagy egy oszlopban levőkkel) valamilyen logikai kapcsolatban vannak. Az elemi adatok meghatározott részhalmaza tehát egy-egy adatcsoportot képez. ▲ 230 4

AZ ELEMI ADAT JELLEMZŐI

A számítógépes feldolgozásban részt vevő adat legfontosabb jellemzője az *adat típusa*. Az adattípus meghatározza azt az *értékhalmazt*, amelyből az adott típusú adat értékeket vehet fel. Az egy típusú adatokkal végzett műveletek eredménye is általában ebbe az értékhalmazba tartozó eredményt ad.

Az adat típusa ugyanakkor meghatározza azt a *művelethalmazt* is, amelyből adott típusú adatokkal végzendő művelet kiválasztható. Az adat típusa meghatározható az adat formája (vagy programbeli leírása) alapján. ▲ 230 5 Az adatnak típustól függően különböző fajta értéke és formája lehet. Adattípust magunk is tudunk definiálni, legfeljebb azzal a kérdéssel kell szembenéznünk, hogy az általunk kitalált típus a számítógépen értelmezett-e?

Pl. kitaláltunk „hónap" típusú adatot, amelynek lehetséges értéktartománya a hónapok neveiből áll.

típus: hónap {január, február, március, április, május, június, július, augusztus, szeptember, október, november, december}

A programozási nyelvek általában ilyen típusú adatot nem tudnak kezelni, az egyes hónapokat egy-egy sorszámmal azonosítják. Vannak azonban olyan adattípusok, amelyeket a legtöbb programozási nyelv valami módon tud kezelni. Ezek a következők:

egész típusú adat,
valós típusú adat,
logikai típusú adat,
karakter típusú adat.

Az egész típusú adat mindig csak egész szám lehet. Pl. a korábban már előfordult „A III. b osztály létszáma" körülírású adat bármilyen osztálylétszám mellett csak egész értéket vehet fel. A valós típusú adat értéke mindig csak valós szám lehet. Pl. „A III. b osztály tanulmányi átlaga" valós értéket jelent (még akkor is, ha értéke mondjuk 4,0, hiszen ez is tartalmaz tizedest). A logikai típusú adat két lehetséges értéket vehet fel, az „igaz" és a „hamis" logikai változóknak megfelelően. ■ 227 6

A karakter típusú adat valamilyen betűkombinációt tartalmazhat. „Az állampolgár neve" körülírású adat esetében pl. CSÉPAI ORSOLYA.

Az adat tehát attól függően, hogy milyen típusú lehet:,

— numerikus (csak számjegyeket tartalmazó): egész és valós típusúnál;

— alfanumerikus (betűket, számjegyeket és speciális karaktereket egyaránt tartalmazó): karakteres típusúnál.

Speciális karakterek pl. az írásjelek, matematikai műveleti jelek stb.

Az információt megjelenítő adatok tehát számjegyeket, betűket, speciális írásjeleket tartalmazhatnak. A speciális jelek zömét azonban az adatokkal végzett műveleteknél használjuk. Az adatokkal végezhető műveletekről a következő fejezetben lesz szó, most csak néhány példát mutatunk be, amely azt szemlélteti, hogy az adat típusa befolyásolja a vele végezhető műveletek körét. Összeadás pl. végezhető egész és valós

típusú adatokkal, de nem végezhető karakter típusú vagy logikai típusúval. A matematikában tanult osztási műveletet valós típusú adatokkal lehet végezni, egész típusúakkal az ilyen osztás nem végezhető el, hiszen az osztás általában nem egész hányadost eredményez. (Ezért létezik egy egész osztás nevű művelet, amely a matematikai osztás hányadosának csak az egész részét szolgáltatja eredményül.)

ADATSTRUKTÚRÁK

Az elemi adatokból különböző adatstruktúrák állíthatók össze. A szakirodalom többféle adatstruktúrát ismer, ezek közül kettővel ismerkedünk meg. Ezek:

a tömbstruktúra, vagy röviden **tömb**,

a rekordstruktúra, röviden **rekord**.

Vizsgáljuk meg az annotációs szöveg | ▲ | 230 | 4 | jelű részében bemutatott táblázat második oszlopát! Ebben az oszlopban a tanulók születési éve található. Látható, hogy ebben az oszlopban minden adat azonos típusú, maga a teljes oszlop tehát egy típusú adatokat tartalmaz. A születési évszámok sorrendje megegyezik a tanulók sorrendjével, vagyis meghatározott sorrendről van szó.

Az egy típusú adatokból álló rendezett adatsort *tömbnek* nevezzük. A későbbiekben, az algoritmusok kapcsán a tömbre vonatkozó ismereteinket tovább fogjuk bővíteni.

Tekintsük most ugyanennek a táblázatnak egy sorát. Azonnal észrevehetjük, hogy itt már nincs szó arról, hogy az egy sorban levő adatok mind azonos típusúak, hiszen már az első (tanuló neve) és a második (születési éve) adat sem azonos típusú.

Tetszőleges típusú adatok egy egységgé való összekapcsolásakor keletkezik a *rekord*. Adatfeldolgozási feladatoknál egy-egy rekordban éppen a feldolgozás szempontjából szükséges tulajdonságok vannak összegyűjtve. Sok olyan feldolgozás van, amelynél szükséges, hogy egy egyedet (ill. a hozzá tartozó rekordot) azonosítsuk, vagyis megkülönböztessük a többitől. Ilyenkor szükséges, hogy a rekordban előforduljon egy azonosító. Az azonosító az egyednek olyan tulajdonsága, amely adott konkrét értékkel csak egy egyednél fordul elő. A hazánkban megvalósított népességnyilvántartásban pl. az egyedet (állampolgárt) a személyi szám azonosítja.

E népességnyilvántartás esetében pl. egy rekord a következőképpen nézhetne ki:

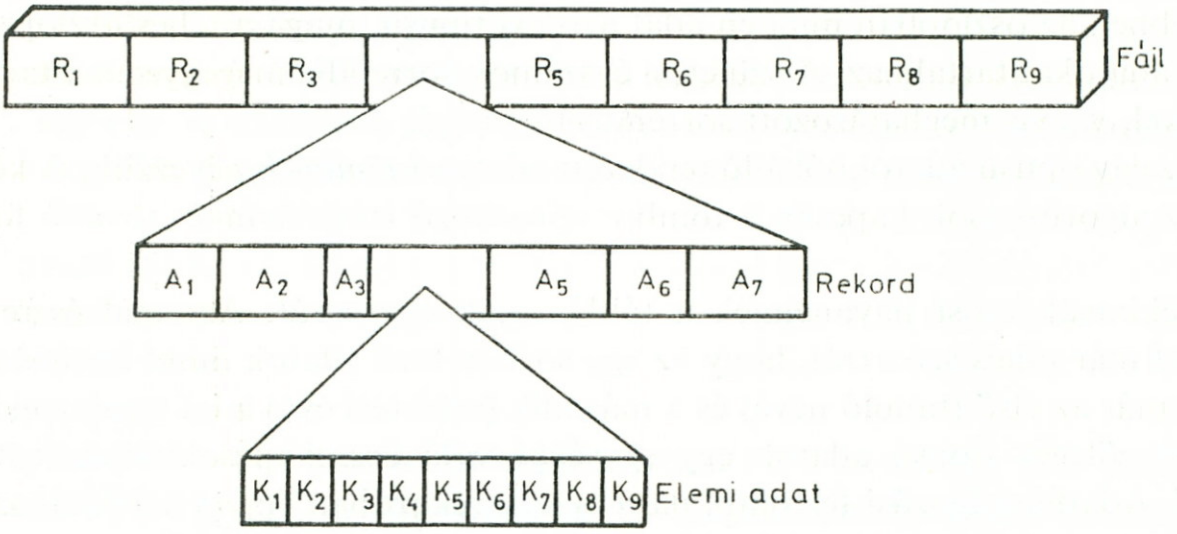
Személyi szám
Családi név
Családi utónév
Leánykori név
Születési anyakönyvi kivonat száma
Születési hely
Születési hely országkód
Anyja neve
Állampolgárság

Családi állapot
Vércsoport
Legmagasabb iskolai végzettség
Foglalkozási kódja
Állandó lakhelye

Tételezzük fel, hogy a nyilvántartás céljaira ezek az adatok elegendők. (A valóságban ennél sokkal több adat szükséges.)

Ha egy feldolgozás céljaira a feldolgozás adott szempontja szerinti valamennyi egyed rekordját összegyűjtjük, egy olyan adathalmazhoz jutunk, amely halmaznak egy-egy elemét egy-egy rekord alkotja. Az ilyen halmazokat **adatállománynak** vagy röviden állománynak nevezzük. Ugyanennek a fogalomnak szokásos elnevezése még a **fájl** (amely az angol file szó fonetikus átírásából került a magyar nyelvű szakirodalomba).

Az elemi adat a rekord és az állomány hierarchikus viszonyát a 3. ábra szemlélteti.



3. ábra. Fájl, rekord, elemi adat kapcsolata

Lehetséges olyan feldolgozás, amelynek csak egy input fájlra van szüksége, más feldolgozások több fájllal dolgoznak. Valamennyi fájl jellemzője, hogy a funkciója által meghatározott szempontból a fájl minden szükséges rekordot tartalmaz.

▲	231	7

A számítógépes feldolgozások nagy százaléka adatállományokkal dolgozik. Egy vagy több input állomány feldolgozása során output állományok jönnek létre. Az output állományok rekordokból állnak, a rekordok elemi adatokból. Ezek pedig információt hordoznak. Ezeknek az információknak a birtokában az ember döntéseket képes hozni, felhasználhatja őket a tudomány, a termelés területén.

A következő probléma, amit meg kell válaszolnunk, hogy miképpen kerülnek az adatstruktúrák kapcsolatba a számítógéppel. A számítógéppel való kapcsolatteremtés — vagyis a számítógéppel való adatközlés — előfeltétele, hogy a bemenő adatok olyan módon legyenek rögzítve, amelyet a számítógép érzékelni és értelmezni tud. Az eredményadatok pedig olyan formában kerüljenek ki a gépből, amelyet az ember könnyen tud megérteni. Az érzékelés problémája azokkal az adatokkal kapcsolatos, amelyeket a számítógépbe be kívánunk juttatni, az értelmezés viszont akkor szükséges, amikor ezek az adatok már bent vannak a számítógépben.

Az adat a számítógéphez viszonyítva háromféle lehet:

— beszélhetünk olyan adatokról, amelyeket fel kívánunk dolgoztatni a számítógéppel (pl. egy másodfokú egyenletben a három együttható: a, b és c értéke, ha számítógéppel végezzük a megoldást);

— beszélhetünk olyan adatról, amely már elhagyta a számítógépet (pl. az előző egyenlet gyökei, amelyeket a gép kinyomtatott);

— végül lehetnek olyan adatok, amelyek a számítógépben tartózkodnak (pl. az előző feladat megoldása közben a diszkrimináns értéke).

Az első két esetben külső (számítógépen kívüli) ábrázolásról, a harmadik esetben belső ábrázolásról beszélünk.

A számítógép — ma még — nem érti a közvetlen emberi beszédet, csak drága berendezések segítségével képes elolvasni a kézzel írott szövegeket. Mondanivalónkat (programjainkat, adatainkat) ezért olyan formára kell kódolni, ami a számítógép számára könnyen érthető. Erre a célra kódrendszereket használunk. Kódrendszernek nevezünk egy jelsorozatot a hozzárendelési szabállyal kiegészítve. A jeleket használjuk a kódolásra, a szabály mondja meg, hogy ezt hogyan tegyük. Így lehetséges, hogy egy kódolt szöveget a szabály ismerője vissza tudjon alakítani. | ▲ | 231 | 8 |

A számítógépbe bejuttatandó adatokat a számítógép számára érzékelhetővé kell tenni. Ehhez olyan eszközre van szükség, amelyre a kiválasztott kódrendszer karaktereivel az adatokat feljegyezhetjük (rögzíthetjük).

Azokat az eszközöket, amelyekre adatokat rögzítünk tárolási, megőrzési (későbbi feldolgozási) célokra, **adathordozóknak** nevezzük. Adathordozó pl. egy rendőrségi bejelentőlap, egy könyvtári nyilvántartó karton, egy iskolai értesítő. Ezek tartalmát azonban közvetlenül nem tudjuk a számítógéppel elolvastatni. Amennyiben egy számítógépes feldolgozás során ilyen tartalmú adatokra van szükség, gondoskodni kell az adatoknak — a számítógép számára elfogadható — adathordozókra való átviteléről. Ezért szokás megkülönböztetni

— *elsődleges adathordozókat és*

— *másodlagos adathordozókat.*

A másodlagos adathordozók tartalma számítógépes feldolgozásra mindig alkalmas.

A számítógépek alkalmazásának korai szakaszában másodlagos adathordozóként a lyukkártyát és a lyukszalagot használták. Mindkettőt egyre inkább kiszorítják a

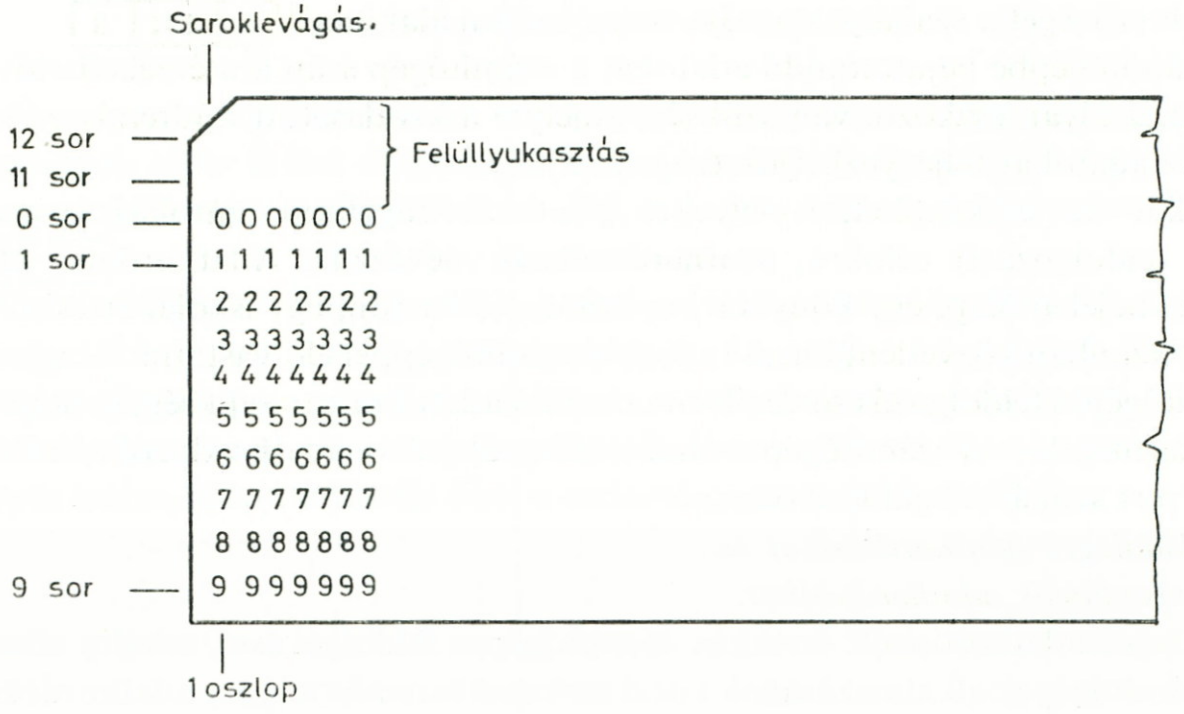
mágneses elven működő adathordozók. Miután azonban ezek tömeges elterjedéséhez olyan berendezésekre van szükség, amelyek ezeket kezelni tudják (s ezek beszerzéséhez pénzre van szükség), ez a kiszorítás — érthetően — elsősorban ott valósul meg, ahol az adatok tömege ezt égetően szükségessé teszi. Más számítóközpontokban a meglevő kártyalyukasztó berendezések még jó néhány évig használatban maradnak, de itt is elsősorban programok rögzítésére. A lyukszalag kiszorulása ennél látványosabban történik. A harmadik generációs gépekhez már csak ott illesztettek lyukszalagolvasót, ahol nem volt elég pénz lecserélni a lyukszalaglyukasztó eszközöket.

A lyukkártya

A mai formában használatos lyukkártyát először a múlt század végén használtak az Egyesült Államokban népszámlálási adatok összesítéséhez. Mechanikusan működő berendezések végezték az adatok feldolgozását, nehézkes és bonyolult módon, bár kétségtelen, hogy gyorsabban és pontosabban, mint ha kézi erővel végezték volna el az adatok csoportosítását. A lyukkártyás technika azóta bevonult az adatfeldolgozás képzeletbeli múzeumába, magát a lyukkártyát az elektronika átvette, s úgy tűnik, még egy ideig meg is fogja tartani.

A lyukkártya vékony, kemény papírból készül, mérete 18,7×8,3 cm. A kártya a 4. ábrán látható módon 80 oszlopra és 12 sorra van osztva. A lyukkártya saroklevágása a kártyakötegbe rosszul behelyezett kártyák vizuális felismerését segíti elő. Egy önálló karakter ábrázolásához egy lyukkártyaoszlop szükséges:

— számjegy ábrázolásához az oszlopban a számjegy értékének megfelelő sornál egyetlen lyukat helyezzünk el;



4. ábra. A lyukkártya struktúrája

— betűt egy oszlopban elhelyezett két lyuk segítségével tudunk ábrázolni. Az egyik lyuk mindig 1—9 közötti értéket, a másik lyuk mindig 0, 11 vagy 12 értéket reprezentál;

— speciális jelek ugyanazon oszlopban elhelyezett több (két vagy három) lyuk segítségével ábrázolhatók.

A lyukak téglalap formájúak. Ennek a „lyukírásnak" tehát minden karakterét lyukak és „nem lyukak" kombinációi képezik. Az 5. ábra egy teljes kódrendszert mutat be, amelyben megtalálhatók a számjegyek, betűk és speciális jelekhez rendelt lyukkombinációk.

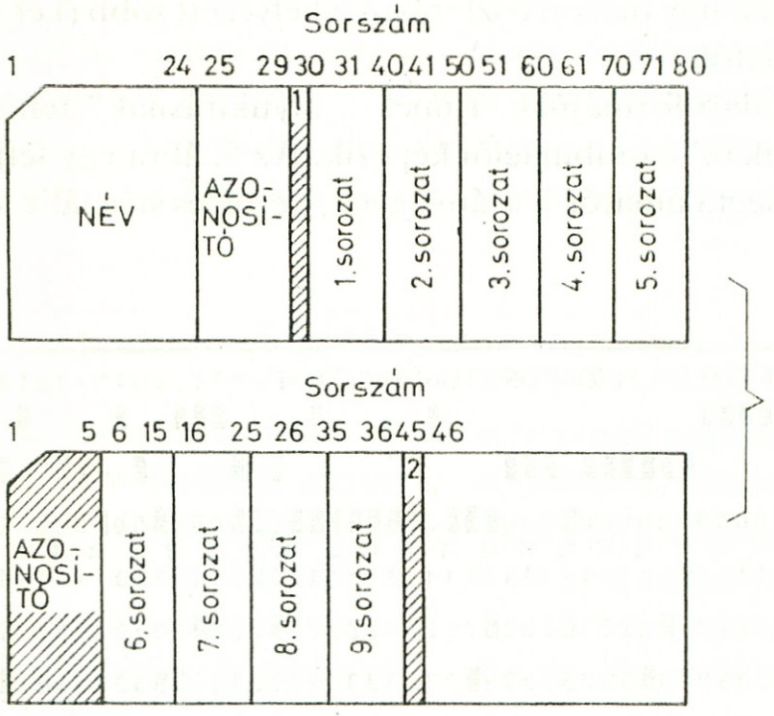
5. ábra. A lyukkártya kódrendszere

Több karakter hosszúságú adat ábrázolásához természetesen egynél több oszlopra van szükség. *Lyukasztási mezőnek* nevezzük azokat az egymást követő oszlopokat, amelyekre egy adat rögzítése történik.

Fizikai rekord

Próbáljuk meg elhelyezni egy rekord szavait egy lyukkártyán. Miután egy adat egy lyukasztási mezőben van, a rekord képe az egymást követő lyukasztási mezők együttese lesz. Rekordunk helyigényét a mezők hosszának (karakterszámának) összege adja meg. Mindaddig, amíg a rekordhossz nem haladja túl a nyolcvan karaktert (vagyis a nyolcvan oszlopot), fizikailag egyetlen lyukkártyát veszünk csak igénybe. Ilyenkor a logikailag összetartozó adatok fizikailag is egy egységet = egy kártyát al-

kotnak. Természetesen előfordulhat olyan eset is, amikor a rekord mérete meghaladja a lyukkártya befogadóképességét. Ilyen esetet szemléltet a 6. ábra.



6. ábra. Két kártyás rekord

Az egy kártyára rögzíthető adatmezők együttesét (függetlenül attól, hogy egy teljes rekordot alkotnak-e) *fizikai rekordnak* nevezzük.

Kártyaterv

Lyukkártyás adathordozók esetében a feldolgozás előkészítése során törekedni kell arra, hogy a rekord és a fizikai rekord ne különüljön el, vagyis az egyed tulajdonságainak mért értékei egy lyukkártyára ráférjenek. Vegyük észre, hogy az 5. ábrán néhány mező sötétebb színű. Ezek szerepe mindegyik kártyán a rekord azonosítása és a rekordon belüli sorrend biztosítása. (Látszólag erre nincs szükség, valójában azonban mindig fennáll a veszélye, hogy egy kártyacsomag kártyái összekeverednek a nem gondos kezelés miatt.)

A fizikai rekord megtervezése a kártyaterv elkészítésével történik. A kártyaterven jelölik ki a rekord egyes szavainak helyét, vagyis itt határozzák meg a fizikai rekordképet. | ● | 232 | 9 |

Mágneses adatrögzítés

Az adatrögzítés területén sokáig egyeduralkodó lyukkártyát egyre inkább kiszorítják a mágneses adatrögzítők. Ennek fő oka, hogy a rögzített adatok lényegesen kisebb helyen férnek el, s magát az adathordozót többször fel lehet használni. Mindez

lényeges költségmegtakarítást eredményez. A mágneses adatrögzítés során nem jelentkezik olyan mereven a kártyánál meglevő fizikai rekordhatár. Ugyanakkor a beolvasás biztonságosabb és gyorsabb.

A mágneses adatrögzítés történhet

— mágnesszalagra,

— kazettás mágnesszalagra,

— hajlékony mágneslemezre (diszkettre).

Az utóbbi kettőről az adatokat át lehet másolni szabványos mágnesszalagra (ezt a műveletet azonban már megfelelő berendezéssel, emberi beavatkozás nélkül meg lehet valósítani).

Egy gépkezelő egynapi munkája kb. 10 méternyi mágnesszalagon elfér. Ezt az adatmennyiséget a számítógép néhány másodperc alatt beolvassa. Ezért kialakultak a csoportos adatrögzítő berendezések. Ezek működésének lényege, hogy ugyanarra a mágnesszalagra több munkahelyről kerülnek fel az adatok. Természetesen ilyenkor gondoskodni kell a felvitt adatok szétválogatásáról és szortírozásáról. Ez a tevékenység is programozottan történhet.

Valamennyi mágneses adatrögzítési mód egyik jellemzője, hogy a munkavégzés során, vagy azt követően egy kisszámítógép segítségével kerülnek az adatok olyan szabványos adathordozóra, amelyet már a nagyszámítógép is kezelni tud.

Optikai bizonylatolvasás

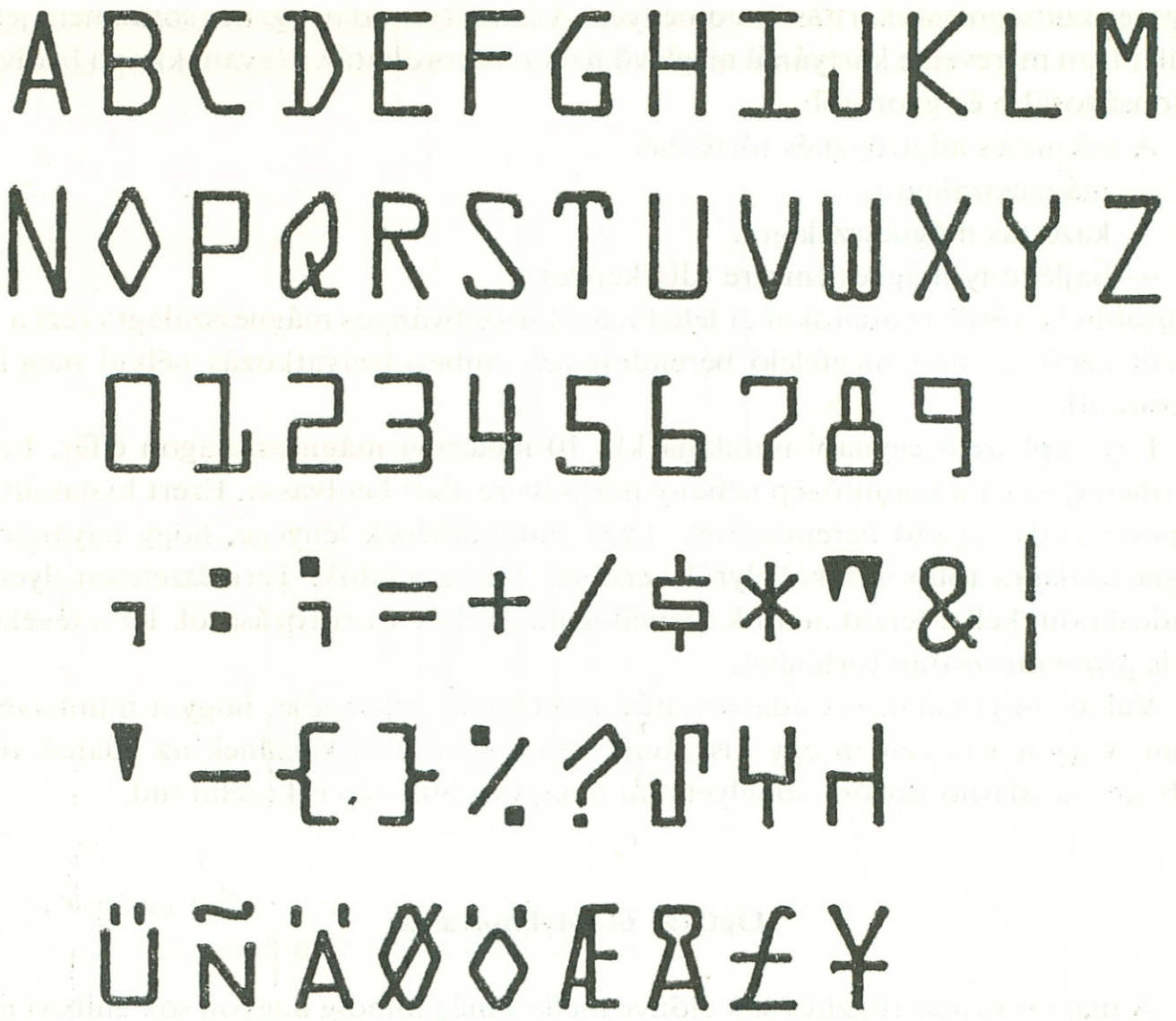
A mágneses adatrögzítés sok előnye mellett még mindig nagyon sok emberi munkát igényel. Ennek kiküszöbölésére már régebben kidolgozták az optikai bizonylatolvasókat. Elterjedésüket nagyban gátolta, hogy alkalmazásuk jelentős beruházási igénnyel járt, így csak nagyon nagy tömegű adattömegnél fizetődött ki. A mikroszámítógépek terjedésével itt is megfigyelhető az árak csökkenése.

A bizonylatolvasók alapvetően két elvet használnak:

— a felismerendő jel mágneses tintával készült;

— a felismerendő jel sötét tónusa másképp veri vissza a fényt, mint fehér környezete.

1966-ban szabványosítottak olyan jelkészletet, amelyben minden karakternek előírt formája van. Ezeket a formákat a számítógép képes felismerni. Egy ilyen betűkből álló abc-t szemléltet a 7. ábra. Miután bármelyik írógép (az írókarok vagy a gömbfej kicserélésével) alkalmassá tehető ezeknek a jeleknek a rögzítésére, viszonylag olcsón előállítható olyan berendezés, amelynek segítségével az alapbizonylat és a számítógépes input megegyezik egymással. Ez sok munkától és tévesztési lehetőségtől kíméli meg az embert.



7. ábra. Az OCR-A írás jelkészlete

A HIBÁS ADATOK KIKÜSZÖBÖLÉSE

Az emberi tevékenység gyakori velejárója a tévedés. A számítógépes feldolgozásban viszont a tévedéseket ki kell küszöbölni. Ezzel lehet csak egyértelmű, pontos feldolgozásokat biztosítani. A számítógép ebben az ember segítőtársa tud lenni, hiszen a hibák feltárásának nagy része különböző adatok összehasonlításával lehetséges, és erre maga a számítógép is felhasználható. Milyen eszközeink vannak a probléma megoldására?

A tévedések kiküszöbölése nagyon fontos mozzanata a számítógép alkalmazásának. Mit ér ui. a tévedésmentesen számoló számítógép, ha hibás adatokkal dolgozik? De mikor hibás egy adat? — kérdezhetjük rögtön. A kérdésre könnyen tudunk válaszolni, hiszen tudjuk, hogy az adat információt hordoz. Hibás egy adat akkor, ha téves információt hordoz. Arra a kérdésre, hogy mitől téves az információ, már bonyolultabb a válasz, mert ez függ attól, hogy mikor és hol „romlott el" az adat. Hiba keletkezhet akkor, amikor az adatot elsődlegesen rögzítették (pl. pontatlan mérés, hibás

számolás miatt). Keletkezhetett a hiba akkor is, amikor az adatot elsődleges adathordozóról másodlagosra rögzítették.

A számítógépes adatfeldolgozás előkészítésének egyik fontos állomása az adathibák kiküszöbölése, ill. minimálisra szorítása. A kétféle tévedés közül kétségtelenül az elsőnek említett a nehezebben korrigálható. Persze, ha a mérés nagyon pontatlan volt, mondjuk a beteg vérnyomását 150 helyett 250-nek olvassa az asszisztens, ez „szembetűnő". Az adatfelvételkor elkövetett hiba — utólag — akkor észlelhető és javítható, ha a hibás adat kívül esik azon az értékkészleten, amelyet normális körülmények között felvehet. (Egy 5,2 osztályátlagról mindenki azonnal látja, hogy lehetetlen.) Az ilyen jellegű vizsgálatok persze számítógépes programmal elvégezhetők.

A másodlagos adatrögzítéskor fellépő hibák javítása egyszerűbb, mert ilyenkor már van mihez viszonyítani. Az adatrögzítéskor fellépő hibázás kiküszöbölésének legegyszerűbb formája, amikor a már rögzített adatot kontrollvizsgálatnak vetik alá. Lyukkártyás vagy mágneses adatrögzítés esetén ez úgy történik, hogy a rögzített rekordokat egy ellenőrző berendezésbe helyezve valaki az elsődleges bizonylatokról ismételten billentyűzi az adatokat. (Ez persze többlet munkaráfordítást jelent.) Minden olyan esetben, amikor a beütött jel és a korábban rögzített jel nem egyezik meg, a berendezés hangjelzést ad. Egy másik módszer, amikor az egyes rekordokhoz egy ellenőrzőszámot is rögzítenek. Megfelelő matematikai eljárással az ellenőrzőszám segítségével megállapítható a hibás adatrögzítés. Az adatbevitel — úgy tűnik — még sokáig gyenge pontja lesz a számítógépek hatékony alkalmazásának, elsősorban azokon a területeken, ahol nagy tömegű adatot kell előkészíteni. Éppen ezért világszerte kísérletek folynak arra, hogy a nagy teljesítményű berendezéseket hogyan lehet hibamentes adatokkal ellátni. A jelenlegi kutatások azt mutatják, hogy néhány év múlva a számítógép képes lesz az emberi hangot adatokká változtatni, sőt bizonyos esetekben „emberi hangon" válaszolni.

Kérdések

1. Mit nevezünk információnak?
2. Mi az elemi adat?
3. Milyen viszonyban van az egyed az elemi adattal?
4. Mit határoz meg az adat típusa?
5. Hányféle adattípust ismersz? Sorold fel ezeket!
6. Mit jelent az, hogy egy adat numerikus, és mit, hogy alfanumerikus?
7. Milyen adatstruktúrákat ismersz?
8. Mi jellemzi a tömböt?
9. Mi jellemzi a rekordot?
10. Mi az adatállomány (fájl)?
11. Mi a kódrendszer?
12. Mi az adathordozók feladata?

Feladatok

1. A halmazelméletben tanult jelölésekkel a következő „adattípusok"-hoz rendeld hozzá azt az értékkészletet, amit felvehetnek:

típus „napok"

típus „családtagok"

típus „négyszög"

2. Készítsetek kódrendszert, amely az abc minden betűjéhez azt a sorszámot rendeli hozzá, ahányadik a betű az abc-ben! Készítsetek rövid (egymondatos) kódolt szöveget!

3. A 2. feladat jó megoldásához elégséges volt-e csak a betűket kódolni? Mi hiányzik még? Egészítsétek ki a kódrendszereteket!

4. Az annotációs rész | ▲ | 230 | 4 | jelzése alatt talált táblázathoz készítsetek kártyatervet, azaz e rekordok feldolgozásához határozzátok meg, hogy az adatok hogyan kerüljenek a lyukkártyára!

III. FEJEZET

Algoritmusok, blokkdiagramok

Az automatáról tanultak alapján már tudjuk, hogy egy bemenőjel hatására

— az automata belső állapotában változás történik;

— valamilyen kimenőjel keletkezik.

A számítógép automata jellegéből következően a bemenő adatok hatására itt is

— a gép belső állapotában változás történik (felveszi a soron következő állapotot);

— eredmények keletkeznek.

Tudjuk azt is, hogy a számítógép alkalmazásának célja az eredményadatok által hordozott információk hasznosítása. Az áruházi pénztárgépekhez csatolt mikroszámítógép esetében a vevő számára készült output-lista információtartalma: a vásárolt cikkek listája és a fizetendő végösszeg. Ugyanakkor a központi számítógép a saját feldolgozásához (raktárkészlet-figyelés, rendelésfeladás) is adatokat kap.

Ha egy számítógépes feldolgozás eredményei senki számára sem adnak információt, a feldolgozás felesleges.

Minden feladat esetében kijelölhető egy induló és egy végállapot. Az induló állapothoz a bemenő adatokat, a végállapothoz az eredményadatokat rendelhetjük hozzá. A kérdés tulajdonképpen az, hogy hogyan lehet eljutni a kezdeti állapotból a végállapotba? A válasz egyszerű: ha egy lépésben nem lehet eljutni, közbülső lépések segítségével olyan közbülső állapotokat kell létrehozni, amelyeken keresztülhaladva a végállapotba juthatunk. ▲ 235 1

Ahogyan az induló és a végállapothoz egy-egy adathalmaz tartozik, így a közbülső állapotok mindegyikéhez is hozzá lehet rendelni egy-egy adathalmazt. Az egyes közbülső állapotokhoz rendelt adathalmazok a bennük foglalt adatok körét és értékét tekintve általában különböznek egymástól, de nem biztos, hogy ez valamennyi lépés esetében fennáll.

A számítógép működése során a közbülső állapotokhoz rendelt adathalmazok általában az ember számára nem válnak láthatóvá. Ez azonban nem változtat a lényegen, ami számunkra most fontos. A számítógép a kiinduló (kezdeti) állapotból több lépésen keresztül eljut egy végső állapotba s ennek hatására az input adatokból feldolgozott eredményadatok állnak elő.

Felvetődnek azonban a következő kérdések:

— Mi történik az adatokkal a feldolgozás során?

— Mi vezérli a kezdeti állapotból a végeredményekhez való eljutást?

— Hogyan írható le a vezérlés menete?

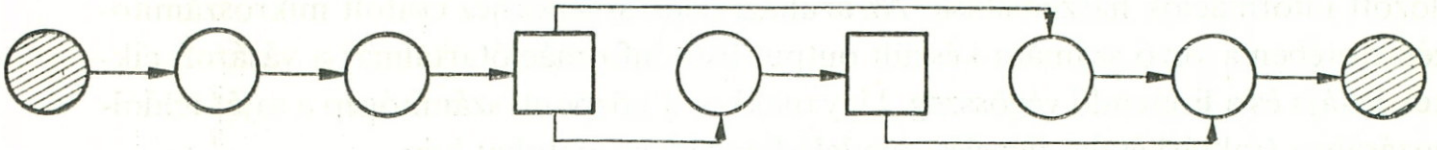
Az eddig tanultak alapján ezekre a kérdésekre még nem tudunk kielégítő választ adni. Pedig válaszolni kell ezekre a kérdésekre, mert nem lehet az adatokat akárhogyan kezelni és feldolgozni. A számítógépek csak abban az esetben végeznek megbízható munkát, ha mi magunk is megbízhatóan tevékenykedünk. Egyéni érdekünk és a közösség érdeke is az, hogy minél pontosabban, egyértelműbben legyünk képesek feladataink megoldására. Erre sarkall bennünket a munkánk iránt érzett felelősségtudatunk is. Ebben a vonatkozásban az alapvető problémánk így fogalmazható meg: **Milyen általános feltételei vannak az adatok feldolgozásának?**

Általános feltételen azt értjük, hogy mi vezérli a feldolgozást, milyen összefüggések érvényesülnek a vezérlés során, milyen műveletek végezhetők, és hogyan valósítható meg mindez a gyakorlatban.

A feldolgozás lényegét jelentő állapotváltozást, vagyis a gép belső működését utasítások sorozata vezérli. Egy-egy utasítás alapvetően

— műveletvégzési feladatot lát el, vagy

— vezérlési feladatot lát el, ahogyan azt a 8. ábra szemlélteti.



8. ábra. A közbülső állapotok sorrendje

Az ábrán körrel jelzett utasítás funkciója, hogy az adatokkal valamilyen műveletet vagy műveletsort valósítson meg. Vegyük észre, hogy minden ilyen funkciójú utasítás után a továbblépés sorban következő utasításhoz vezet.

Az ábrán négyszöggel jelzett utasítások vezérlési feladatot hajtanak végre. Ennek eredménye a továbblépés helyének meghatározása. ▲ | 236 | 2

A PROGRAM

Azt az utasításcsoportot, amely egy kijelölt input adatstruktúrájából egy előre meghatározott eredménystruktúrába való átmenet lépéseit írja elő, **programnak** *nevezzük.*

Milyen részekből áll a program? A program két részből áll, az adatdefiníciós részből és a vezérlésleíró részből.

Az *adatdefiníciós részben* kerül meghatározásra, hogy a programban előforduló adatok milyen típusúak, a vezérlésleíró részben történik meg az eredményt előállító műveletsor sorrendjének és tartalmának meghatározása. ● | 237 | 3 ▲ | 236 | 4

A program tehát mindig egyértelműen meghatározható a hozzá tartozó adatok

és utasítások véges halmazának a megadásával. A program számítógéppel való végrehajtása a dinamikus program. A program dinamikájának megtervezése a programozandó feladat **algoritmusának** a megadásával történik. | ★ | 235 | 5 |

Mi az algoritmus? *Az algoritmus egy probléma megoldásának véges számú részlépésben való egyértelmű és teljes leírása.* A részlépések egyértelműségét az biztosítja, hogy minden részlépés után egyértelműen meghatározható a soron következő részlépés, a teljességet pedig az, hogy minden részlépésre van rákövetkező, kivéve az utolsót, amely az algoritmus végét jelzi.

Az algoritmus fogalma bizonyos fokig független a számítógép, ill. a számítógépes program fogalmától. Beszélhetünk olyan tevékenységek algoritmusáról, amely tevékenységek számítógéppel való elvégzése nem lehetséges vagy nem kívánatos. | ● | 238 | 6 | Magának az algoritmusnak a leírási módjára nincsenek megkötések, sok esetben elképzelhető a verbális (szavakkal történő) leírás. A programtervezéshez használt algoritmusok természetesen igyekeznek ennél egzaktabbak és rövidebbek lenni, ezért matematikai és más előre definiált jelöléseket használnak.

Egy számítógéppel megoldandó probléma algoritmusának az elkészítésekor tulajdonképpen a készítendő program *vezérlési részét* fogalmaztuk meg, oly módon, hogy eltekintünk a leírási nyelvtől.

A program másik részét, az adatdefiníciós részt, általában a program írásakor szokták elkészíteni. Ennek oka, hogy ez utóbbi jobban függ a használni kívánt programozási nyelvtől, mint a vezérlésleíró rész. A korszerű programtervezési irányzatok azt tanítják, hogy e két részt a programnak nem szabad egymástól függetlenül kezelni. A II. fejezetben az adattípusra adott meghatározás is alátámasztja ezt. Az adattípusról tanultakból tudjuk, hogy — egyebek mellett — az adat típusa meghatározza azt a művelethalmazt, amelyből az adott típusra vonatkozó műveletek kiválaszthatók.

De milyen műveleteket lehet végezni az adatokkal? Beszélhetünk olyan műveletekről, amelyeket a tanult adattípusok mindegyikére értelmezhetünk, és lehetségesek olyan műveletek, amelyek csak meghatározott adattípusra (vagy néhány adattípusra) értelmezhetők.

Műveletek adatokkal

Három olyan művelettel ismerkedünk most meg, amelyet az összes tanult adattípusra értelmezhetünk.

Ezek:

— a hasonlítás (egyenlőségvizsgálat),
— az értékadás,
— az átvitel.

A *hasonlítás*ra alapvetően a = (úgy olvassuk, hogy „egyenlő") szimbólumot használjuk. Kikötjük, hogy az egyenlőségjel két oldalán csak azonos típusú adat szerepelhet. Maga a művelet kétféle eredményt adhat: a logikai igaz, ill. a logikai hamis értéket, attól függően, hogy fennáll-e az egyenlőség a hasonlítás két oldala között.

A hasonlításra használható még a $>$ (nagyobb) $\geq$ (nem kisebb), $<$ (kisebb), $\leq$ (nem nagyobb) és a $\neq$ (nem egyenlő) reláció is.

Az *értékadás*ra a := szimbólumot használjuk (úgy olvassuk, hogy „legyen egyenlő"). A műveletet úgy definiáljuk, hogy az értékadás bal oldala felveszi azt az értéket, amivel a jobb oldal rendelkezik.

Az *átviteli* műveletek — az átvitel irányától függően — kétfélék lehetnek. A „kívülről" való adatbevitelre a **READ** (ejtsd: ríd) (olvasni) utasításszót használjuk. A külvilág felé való adatmozgatást a **WRITE** (ejtsd: rájt) (írni, kiírni) utasításszóval adjuk meg. | ● | 238 | 7 |

Az előbbi általános műveleteken kívül az egyes adattípusokra még a következő műveleteket fogjuk megengedni:

Az egész és a valós típusú adatokkal a *matematikai műveleteket*. (Nem csupán a négy alapműveletet, hanem pl. a gyökvonást vagy a hatványozást is.)

A logikai típusú adatokkal a *logikai műveleteket* közül a következő hármat:
a konjunkciót („ÉS" művelet),
a diszjunkciót (megengedő „VAGY" művelet) és
a negációt („NEM" művelet).

A karakter típusú adatokra a már bevezetett általános műveleteken kívül újabbakat nem értelmezünk.

Az adatokra való hivatkozás

Egy adat egy algoritmuson belül vagy nem változtatja meg értékét, vagy pedig időről időre újabb értékekkel bír. Azt az adatot, amely állandó értékű, konstansnak nevezzük. Konstans pl. a π értéke, bármely kör területének kiszámításánál. Ezeket az adatokat konkrét értékükkel szerepeltetjük, a π-t tehát pl. 3,14 értékkel, ha a két tizedesre való megadás elegendő. Azt az adatot, amely az algoritmus során változtatja az értékét, változónak nevezzük. Változó pl. a kör sugara abban az algoritmusban, amelyben különböző körök területét kell kiszámítani. | ▲ | 236 | 8 |

A programozási nyelvekben a különböző változókra különböző szimbolikus nevekkel hivatkozunk. Az egyes nyelvek pontosan definiálják, hogy e szimbólumoknak milyen formai szabálynak kell eleget tenniök. Általában a szimbolikus nevek (szokásos elnevezéssel változónevek) egy vagy több alfanumerikus jelből álló jelsorozatok, pl. A, B, MIN, TOMB, SUGAR stb.

A változóneveket szabadon választhatjuk (a nyelv szabályain belül), de ugyanarra az adatra (pl. a kör sugara) mindig ugyanazzal a szimbólummal kell hivatkozni egy algoritmuson, ill. programon belül. Egy-egy változónévnek a központi tárban egy-egy konkrét tárhely felel meg. E nevekkel tehát a tár egyes helyeit azonosítjuk.

Érték hozzárendelése a változónévhez

Amikor a változónevekkel szimbolizált adatokkal műveleteket végzünk, felmerül az igény, hogy a számítások eredményét megőrizzük egy újabb változónévvel jelölt területen. Ehhez van szükségünk az értékadás műveletére. *Az értékadás fogalmán egy hozzárendelést értünk: egy változónévhez hozzárendeljük egy műveletsor eredményét.* (Speciálisan a „műveletsor" állhat egyetlen változónévből vagy konstans számértékből.)

Az értékadás szimbólumaként a már bevezetett := (legyen egyenlő) jelet használjuk. Az értékadás művelete a következő szabályok figyelembevételével alkalmazható:

— egy értékadás jobb oldalán csak konstans vagy olyan változónevek szerepelhetnek, amelyek a korábbiakban valami módon már értéket kaptak;

— ha egy értékkel rendelkező változónév a jobb oldalon megjelenik, a változónévhez korábbiakban hozzárendelt érték továbbra is megmarad;

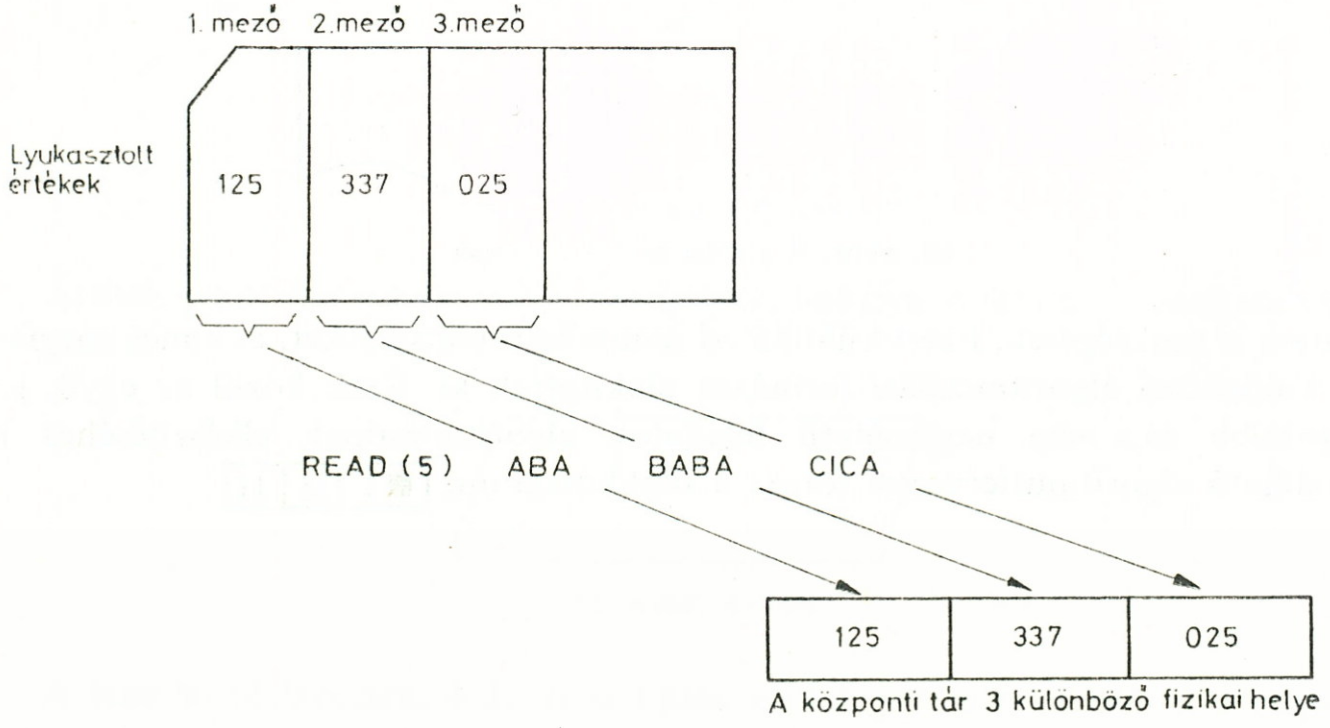
— ha egy értékkel rendelkező változónév egy értékadás bal oldalán jelenik meg, akkor korábbi tartalma elvész, és a változónévhez az új érték kerül hozzárendelésre.

Változókhoz az értékadáson kívül még egy módon tudunk értéket rendelni, az *adatbeviteli (olvasási) művelet* segítségével. Az adatbevitelt a már bevezetett READ művelettel valósítjuk meg:

READ (n) változónevek listája,

ahol READ (olvass) az adatátvitelt előíró parancsszó,

n annak a perifériális egységnek az azonosítója, amelyről az adatokat be kell olvasni. (Ezt az azonosítást a továbbiakban — az egyszerűség kedvéért — egy számmal fogjuk elvégezni, a kártyaolvasót a továbbiakban pl. az 5 számmal azonosítjuk.);



9. ábra. Az olvasási művelet hatása

lista: azoknak a változóneveknek a felsorolása, amelyek a beolvasás hatására értéket kapnak.

A beolvasási művelet hatását a 9. ábra szemlélteti. A listaelemek együttesét egy rekordnak tekintjük. ▲ | 236 | 10

Eredményközlés

Az eredményeket azok létrejötte után ki kell íratni. Ehhez a kiírási utasítás szolgál, amelynek formája

WRITE (n) változónevek listája,

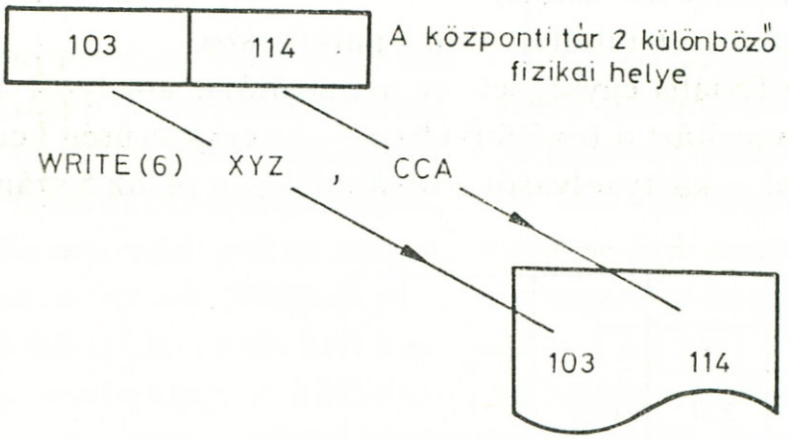
ahol WRITE (írj) az adatátvitelt előíró parancsszó;

n annak a perifériális egységnek az azonosítója, amelyre a kiírás történik (a sornyomtatót a továbbiakban a 6 számmal azonosítjuk);

lista: azoknak a változóneveknek a felsorolása, amelyek tartalmát meg kívánjuk jeleníteni.

Az írási művelet hatását a 10. ábra szemlélteti. Itt is a listaelemek együttese alkot egy rekordot.

Az algoritmusok elkészítése során ezeket a műveleteket fogjuk használni. Kiderült azonban, hogy egy-egy algoritmus leírásának elősegítése érdekében további szim-



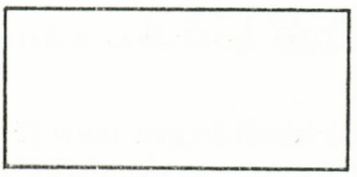
10. ábra. A kiírási művelet hatása

bólumok is szükségesek. Ezért különböző szimbólumrendszereket, és ennek megfelelően különböző algoritmizálási formákat alakítottak ki. Ezek közül az egyik legelterjedtebb és a nem nagyméretű feladatok algoritmusainak elkészítéséhez jól használható algoritmustervezési forma a blokkdiagram. ■ | 235 | 11

BLOKKDIAGRAMOK

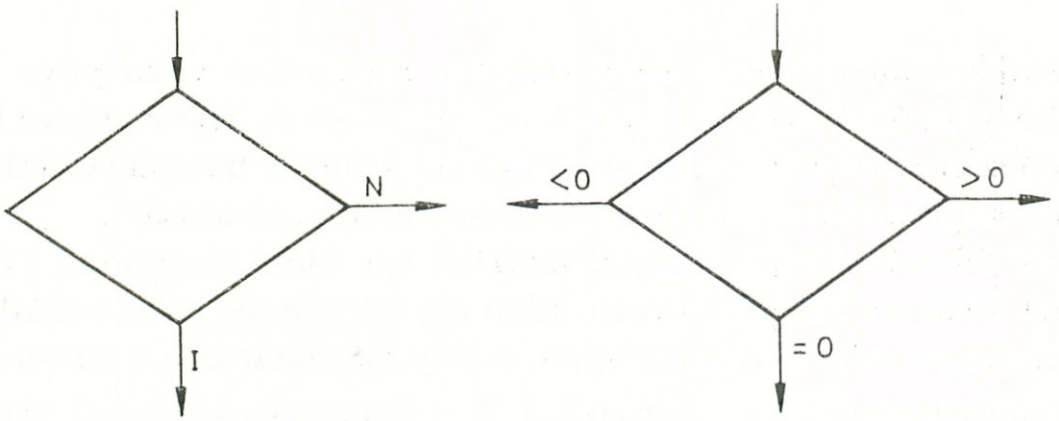
A blokkdiagram formában való algoritmizálás esetén a következő jelöléseket használjuk:

Általános tevékenység: jele téglalap, az elvégzendő tevékenységet a téglalapba írjuk. A tevékenység leírásához felhasználjuk a definiált műveleteket (11. ábra).



11. ábra. Általános művelet

Döntés: egyik csúcsán álló rombusz, amelyben egy hasonlítási művelet van. Attól függően, hogy a hasonlítás eredménye igaz-e vagy hamis, kétfelé ágazhat a feldolgozás. Néha célszerűnek látszik valamilyen kiszámított érték előjelének függvényében háromfelé ágaztatni a feldolgozást: a negatív, a pozitív és a nulla (előjel nélkül) eredménynek megfelelően. Így kétféle döntést használhatunk, a kétfelé ágazó logikai és a háromfelé ágazó aritmetikai döntést (12. ábra).



12. ábra. Döntések

Átvitel: jelölésére a romboidot használjuk, beleírva magát az átviteli műveletet (13. ábra).

13. ábra. Átvitel

A vizuális tájékozódás érdekében külön szokás jelölni a **kezdeti és a végpontot** a 14. ábrán bemutatott jelöléssel.

Az egyes tevékenységeket folyamatvonalak, **nyilak** kötik össze. Az algoritmus végrehajtása e nyilak mentén fog bekövetkezni. Bár a blokkdiagram jelölésrendszeré-

14. ábra. Kezdeti és végpont

ben az egyes részlépésekhez tartozó közbülső adatok nincsenek jelölve, azok helye éppen e nyilak mentén lenne.

Amikor a blokkdiagram méreténél fogva nem fér rá egyetlen lapra vagy a továbbhaladást mutató nyilat túl távoli pontba kellene húzni, a **csatlakozási pontok** használata segít.

A 15. ábra szerinti kör csatlakozó párja a lap egy más helyén, az ötszög párja egy másik lapon található. Az összetartozó csatlakozókba ugyanazt a betűt vagy számjegyet írjuk.

15. ábra. Csatlakozók

A blokkdiagramban levő tevékenységeket egymáshoz viszonyítva úgy szokás elhelyezni, hogy az őket összekötő nyilak *felülről lefelé és balról jobbra* haladjanak. Az ettől esetleg eltérő irányokat azok irányának különös hangsúlyozásával célszerű ábrázolni. Ez nagyban megkönnyíti a blokkdiagramok „olvasását".

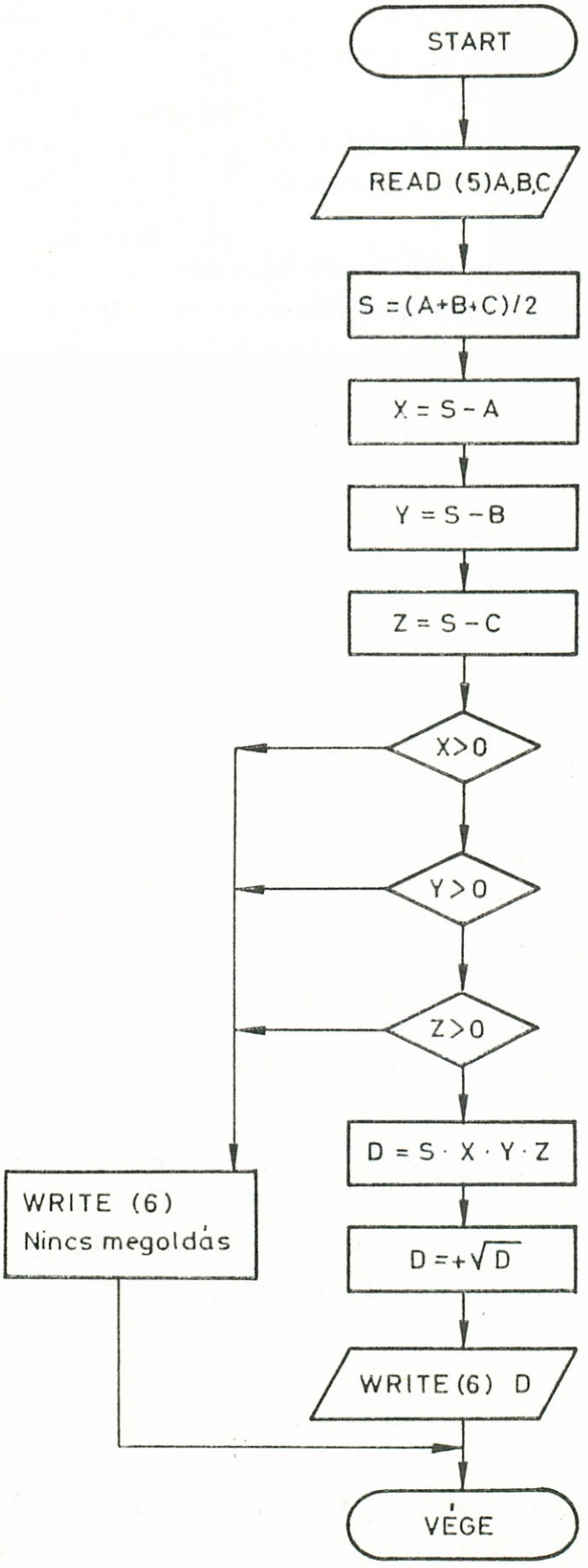
Idézzünk fel másodikos tanulmányainkból egy blokkdiagramot. (Technika II. 142. oldal.) A feladat a következő volt: adott egy háromszög három oldalának mérőszáma, készítsük el azt a blokkdiagramot, amely meghatározza e háromszög területét. A megoldás matematikai „háttere" a $T = \sqrt{s(s-a)(s-b)(s-c)}$ területszámító képlet. ▲ 237 12 A blokkdiagram formájában rögzített algoritmust felkészítjük arra az esetre is, amikor a három input adatból nem szerkeszthető háromszög (amikor bármelyik oldal hosszabb a másik kettő összegénél). ▲ 237 13

A 16. ábra tanulmányozásakor vegyük észre a következőket:

— az algoritmusban szereplő adatok típusáról nincs információnk (hiányzik az adatdefiníciós rész, amit a blokkdiagramból készítendő programnak majd tartalmaznia kell). Természetesen matematikai tanulmányainkból tudjuk, hogy valamennyi adatunk valós típusú;

— az algoritmusban előfordul valamennyi tanult általános művelet, s ezen kívül a valós adatokkal végezhető matematikai műveletek majd mindegyike;

— az algoritmus pontos és áttekinthető képet ad a feladat megoldásának vezérlési struktúrájáról. ● 239 14



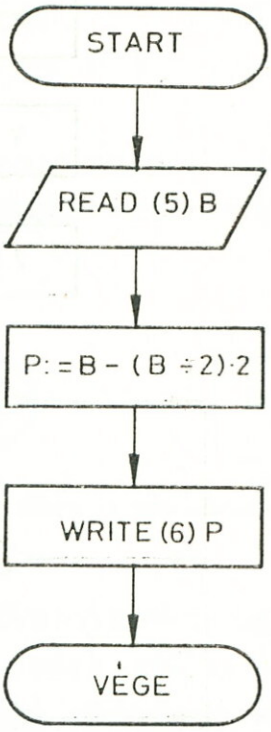
16. ábra. A háromszög területének számítása

Annak szemléltetésére, hogy milyen könnyen kezelhetővé válnak feladatok, ha sikerül az adattípusnak megfelelő művelettípust használni, bemutatunk egy példát.

A feladat a következő. Készítsünk olyan algoritmust, amely segítségével meghatározható egy egész típusú számról, hogy páros-e vagy páratlan. Ha a szám páros, P legyen 0, ha páratlan, P legyen 1.

Definiáljuk az egész osztás műveletét. Jele legyen ÷ és hatása a következő: két egész szám osztásának eredménye az osztás egész része. (Az osztás törtrésze tehát elvész.)

Gondolnánk-e, hogy az algoritmusunk egyetlen utasításból áll, ha eltekintünk az adatbeviteli és az eredménykiírási utasításoktól? (17. ábra).



17. ábra. B egész típusú változó párosság vizsgálata

A gyakorlatban viszonylag kevés olyan problémával találkozunk, amelynek megoldása során az algoritmus az adatokat elemi adatok szintjén kezeli. A legtöbb esetben az input valamilyen adatstruktúra formájában áll rendelkezésre. Vizsgáljuk meg, hogyan tudjuk blokkdiagrammal a tanult adatstruktúrákat kezelni.

Tömbök kezelése

A II. fejezet annotációs részében (▲ | 236 | 4 jelzés alatt) bemutatott táblázatának bármelyik oszlopa egy olyan adategyüttes, amelynek elemei hasonló tulajdonságúak, de más-más egyedre vonatkoznak. Az osztály tanulóinak születési dátumait — az egyszerűbb kezelhetőség kedvéért — egy olyan hatjegyű szám formájában képzeljük most el, ahol az első két számjegy a születés éve, a második két számjegy a születés hónapjának sorszáma, az utolsó két számjegy pedig a születés napjának dátuma. Meg kell határozni, hogy sorrendben hányadik tanuló a legfiatalabb az osztályban.

Mielőtt a megoldási algoritmust gondolnánk végig, jelölési problémáink vannak. Hogyan lehet erre az adatsorra hivatkozni? Célszerű megoldás, ha az adatsort egy olyan rendezett halmazként próbáljuk értelmezni, amelynek egyes elemeit valamilyen sorszámmal különböztetjük meg egymástól. A halmaz azonosítására — célszerűen — ugyanolyan szimbólumokat (változó neveket) használunk, mint az egyedi adatok esetében tettük. A halmazon belüli azonosításra pedig e változónév mellé egy sorszámot írunk, amely a rendezettségen belüli „helyét" adja meg az adatnak. Pl. $ÉV_2$ jelenti az ÉV névvel azonosított halmaz 2. elemét. (A sorszámot alsó indexbe írhatjuk, bár megjegyezzük, hogy a programozási nyelveknél az index a következő formában jelenik meg: pl. ÉV (2) — tehát a tömb neve után zárójelben.)

Egy ilyen elrendezés szerint megadott adathalmaz a tömb. Egy tömböt akkor tekintünk megadottnak, ha ismerjük a benne levő elemek számát és az egyes tömbelemeket.

Most már visszatérünk a feladat algoritmusának elkészítéséhez. Egészítsük ki a feladatot azzal, hogy az osztály létszámát egy N nevű változóba olvassuk be a legelső adatkártyáról. A tömbelemek egyetlen rekordot alkotnak, a második adatkártyától kezdődően. A legfiatalabb tanuló nyilván az, aki legkésőbb született, akihez matematikai értelemben a legnagyobb születési szám tartozik. (Ezért vettük fel a születési dátumot hatjegyű szám formájában, így nem kell külön az évet, a hónapot és a napot vizsgálni.)

A feladat tehát így is megfogalmazható: egy N elemű tömb legnagyobb eleméhez tartozó indexet kell meghatározni. A megoldás a 18. ábrán látható.

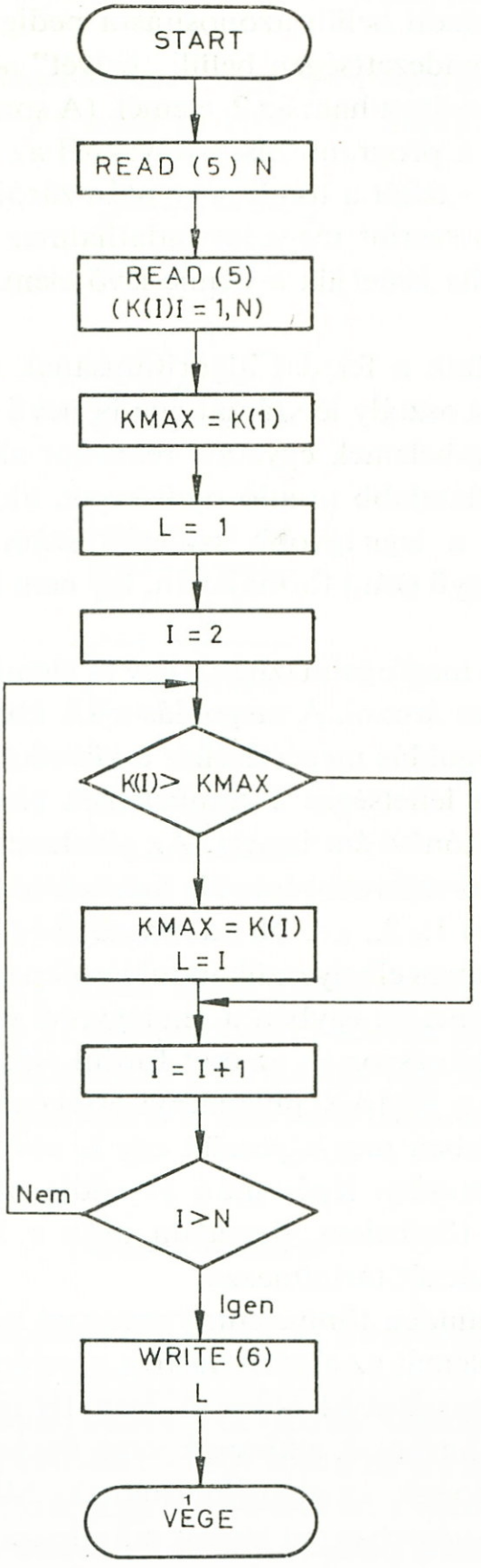
Az ábrán látható megoldás megértéséhez a következő kiegészítések szükségesek. A tömb beolvasása nem lehetséges a tömbelemek tételes felsorolásával, mivel az elemek száma csak változónévként ismert. Az alkalmazott READ lista értelmezése: „olvasd be a soron következő rekordot, oly módon, hogy az K_1, $K_2 \ldots K_N$ változók vegyék fel az input rekord 1., 2., ... N. adatmezőjében található értékét.

KMAX nevű változóban elhelyezzük indulóértéknek az első tömbelemet. (Ha a tömb csak egy elemű lenne, ez egyben a legnagyobb érték is.) Utána sorravesszük a tömbelemeket, és valahányszor az az eset fordul elő, hogy az éppen vizsgálat alá vett tömbelem nagyobb a KMAX pillanatnyi értékénél, a KMAX-ba e tömbelem értékét tesszük be, miközben megjegyezzük egy L nevű változóban a most vizsgált tömbelem indexét. Így minden lépés után az eddig talált legnagyobb elem van a KMAX-ban. Az utolsó tömbelem vizsgálata után a KMAX a tömb legnagyobb elemét, L pedig ennek indexét tartalmazza.

A megoldás során minden tömbelemre ugyanazt a vizsgálatot kellett elvégezni. Látható, hogy ezt a problémát az algoritmusban a vezérlésnek az algoritmus egy korábbi pontjára való visszaadásával oldottuk meg. Ily módon az algoritmusban egy hurok állt elő. Ezeket a hurkokat **ciklusnak** vagy **iterációnak** nevezzük. A ciklusok nagyon fontos elemét képezik az algoritmusoknak. Minden ciklus alapvetően két részből áll. Az ismételten végrehajtani kívánt műveletsorból — ezt nevezzük a ciklus

magjának — és a ciklus működését biztosító részből, ezt nevezzük ciklus adminisztráló résznek.

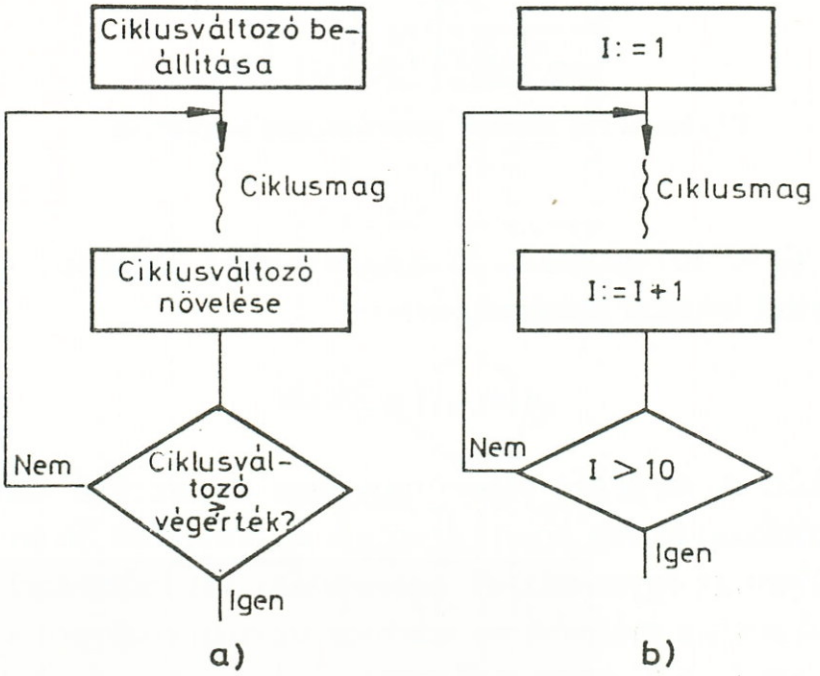
A ciklus adminisztrálásában az ún. ciklusváltozónak van szerepe. A ciklusváltozó először felveszi az előírt kezdőértéket, majd a ciklusmag végrehajtása után a meg-



18. ábra. A legnagyobb tömbelem keresése

adott lépésközzel megváltozik, és amennyiben az így megváltozott érték nem haladja meg az előírt végértéket, a ciklusmag ismét végrehajtásra kerül. A lépésköz általában 1, de lehet más szám is, ahogy a feladat megkívánja. | ● | 240 | 15 |

Az elmondottakat a 19. ábra szemlélteti, az ábra *a)* változatán általában, *b)* változaton a lépésköz 1, a kezdőérték 1 és a végérték 10.



19. ábra. A ciklus szerkezete

Az algoritmus véges jellegéből következően minden ciklusból véges számú menet után ki kell lépni. Az eddig tárgyalt ciklus menetszáma a kezdőérték, a végérték és a lépésköz ismeretében előre kiszámítható. | ● | 240 | 17 | | ⊙ | 242 | 16 |

Kétdimenziós tömbök

Gyakori eset, hogy egy adathalmazt táblázatos formában célszerű értelmezni. Tekintsük pl. a következő feladatot:

Egy osztály létszáma 36 fő. A tanulókról öt tantárgyra vonatkozóan ismerjük az előző tanév végi osztályzataikat. (Az öt tárgy sorrendje: magyar, történelem, orosz, fizika, biológia.) Készítsünk algoritmust, amely meghatározza a tantárgyakból elért osztályátlagokat minden tantárgyra külön-külön. Próbáljuk meg a tömb elemeit a tanult módon sorszámmal (index-szel) ellátni. Összesen $36 \times 5 = 180$ adatunk van. Ha a sorszámozást úgy végezzük el, hogy az első tanuló magyar jegyéhez az 1-et, a második tanuló magyar jegyéhez a 2-t ..., a 36. tanuló magyar jegyéhez a 36-ot rendeljük hozzá, akkor a gondolatmenetünk úgy folytatódna, hogy az első tanuló történelem jegye lesz a 37. stb., ahogy a 20. ábra szemlélteti. Olyan algoritmus kellene ezek után, amely az 1—36, 37—72 ... stb. elemeket kezeli ebben a tömbben.

1	37	73	109	145
2	38	74	110	146
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
36	72	108	144	180

20. ábra. Az elemek sorszámának alakulása

A k. tanuló ($k \leq 36$) m. ($m \leq 5$) tárgyból elért jegyének sorszámát (indexét) a következő képlettel lehetne kiszámítani:

$$(m-1) * 36 + k.$$

Ez eléggé bonyolult. A megoldó algoritmus igen egyszerűvé válik, ha ezt a 36×5 adatot egy olyan táblázat formájában képzeljük el, amelynek 36 sora és 5 oszlopa van. Egy ilyen táblázatban az egyes adatok azonosítása két index-szel lehetséges: az első az adathoz tartozó sort, a második az adathoz tartozó oszlopot azonosítja.

A $T_{3,4}$ tehát azt jelenti, hogy a T táblázat 3. sorában levő 4. elemről van szó, esetünkben a sorrendben 3. tanuló 4. tantárgyból — fizikából — elért osztályzatáról. Ha az input adatokat ilyen elrendezésben kezeljük, a megoldandó feladat abból áll, hogy minden oszlopban képezni kell az ott levő osztályzatok összegét, majd az így kapott eredményeket el kell osztani a tanulók számával, vagyis az oszlop hosszával (ez esetben 36-tal).

A táblázatos elrendezésben elhelyezkedő adatok együttesét *kétdimenziós tömbnek* nevezzük. A tömb méreteit sorainak és oszlopainak száma adja meg. A tömbelemre való hivatkozás az aktuális sor és oszlopindex megadásával történik. Az *elsőnek* megadott érték mindig arra utal, hogy az adott elem hányadik *sorban* található, a *másodiknak* megadott érték pedig, hogy ezen belül hányadik *oszlopban.*

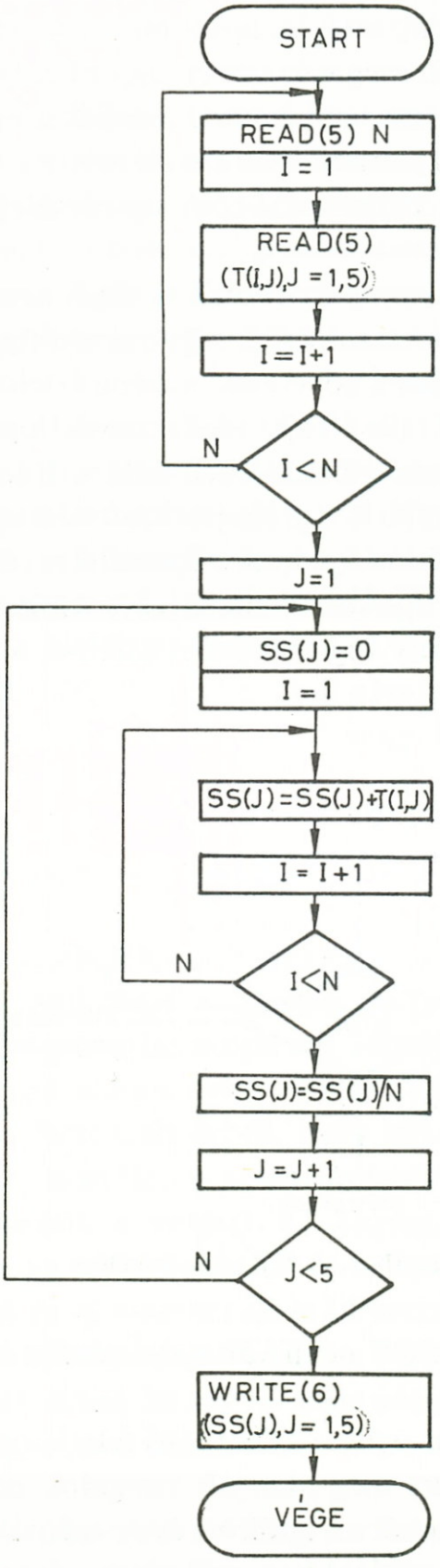
A feladatunk inputjának célszerű formája a következő:

Az első adatkártyán megadjuk az osztály létszámát. Ez esetünkben 36, de olyan algoritmust akarunk tervezni, amely ettől eltérő létszám esetén is működőképes. Minden tanulóról egy-egy kártyát lyukasztatunk. Az egyes kártyákon (a tantárgyak kötött sorrendjében) az elért osztályzatok szerepelnek. A tanuló azonosításától eltekintünk, miután a feladat megoldásához erre nincs szükség. A megoldást a 21. ábra szemlélteti.

A megoldás megértéséhez a következő kiegészítések szükségesek:

Láthatjuk, hogy két egymásba épített ciklus működik. A belső ciklus egyetlen oszlop elemeit adja össze, az összeget egy egydimenziós tömb egy elemében helyezve el. Ennek az egydimenziós tömbnek a pillanatnyi indexe megegyezik az éppen feldol-

gozott oszlop sorszámával. A külső ciklus biztosítja, hogy az összegzés *valamennyi* oszlopra végbemenjen. Eredményül 5 adatot kapunk: az öt tantárgyból elért osztályátlagot.



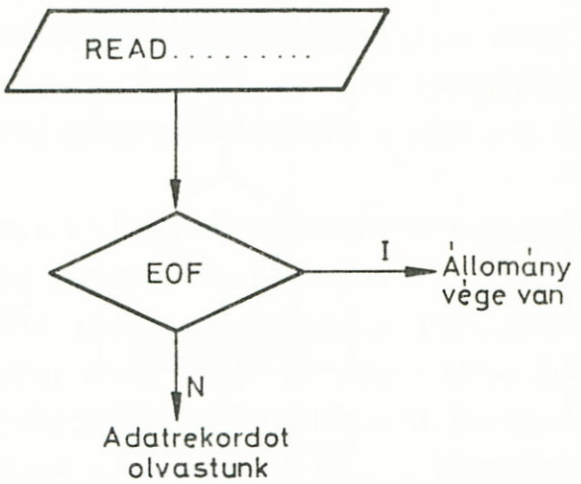
21. ábra. A kétdimenziós tömb kezelése

A rekordstruktúra kezelése a tömbstruktúrához képest több szempontból különbözik. Az egyik különbség, hogy míg a tömb elemei mindig azonos típusúak, a rekord elemei általában nem. Egy másik eltérés, hogy míg egy tömb esetében a benne szereplő elemek száma ismert, tehát meg van adva (vagy a beolvasás során leszámolható), addig a rekordok száma általában nem ismert. Ráadásul egy állomány rekordjainak a száma akár a több tízezret is elérheti. Mindezek miatt a rekordok feldolgozásakor

— sohasem töltjük be az összes rekordot egyidejűleg a tárba, hanem azokat egymás után dolgozzuk fel;

— a feldolgozás addig tart, amíg az állomány végét nem érjük el. Erre a tényre az állományt lezáró speciális rekord, az ún. fájl-vége-rekord figyelmeztet.

A fájl-vége-rekord érzékelését a következő módon értelmezzük: Az algoritmusban értelmezünk egy **EOF** (end of file = fájl vége) nevű, logikai típusú változót. Ezt a változót a programozónak nem kell külön definiálnia. Ennek a változónak az értéke mindig hamis, kivéve, ha az utoljára végrehajtott művelet egy olyan input adatátvitel (READ) volt, amely az adott állományt lezáró speciális rekordot olvasta. Ekkor a változó értéke igaz lesz. | • | 241 | 18 | Annak ellenőrzése tehát, hogy az olvasási műveletünk adatrekordot vagy állomány vége rekordot talált-e, a 22. ábrán látható módon lehetséges.



22. ábra. A fájl vége kezelése

Fontos tudnunk, hogy az EOF nevű változó értékére közvetlenül a READ után kell rákérdezni.

Az eddig tanultakat alkalmazzuk a következő feladat megoldása során. Egy vállalatnál felmérést végeztek, mert meg akarják vizsgálni, hogy miképp függ a dolgozók fizetése a beosztástól, a munkában eltöltött évek számától és attól a ténytől, hogy az illető férfi-e vagy nő. A számításokhoz szükséges adatokat tartalmazó adatállomány rekordfelépítése a következő:

Dolgozó azonosítója

Munkában eltöltött évek száma
Neme (1 — ha férfi, 2 — ha nő)
Fizetése
Beosztása (1 — ha felsőszintű vezető, 2 — ha középvezető, 3 — ha beosztott)

Az állományban levő rekordok számát nem ismerjük, az állományt a fájl-vége-jel zárja. Az állomány a 19-es számmal azonosítható.

Ennek az állománynak a felhasználásával válaszoljunk a következő kérdésekre:
— hány férfi és hány női vezető van a vállalatnál;
— mekkora a férfi, ill. a női vezetők átlagfizetése.

A feladat megoldásához a következő gondolatmenet nyújt segítséget:
— egymás után megvizsgáljuk az állomány rekordjait;
— ha egy rekord nem felsőszintű vezető beosztású dolgozóról készült, semmi dolgunk, vehetjük a következőt;
— ha felsőszintű vezető beosztásúról készült a rekord, akkor külön a férfiakról és külön a nőkről gyűjteni kell a fizetéseket és a létszámot;
— ha az állomány véget ért, két osztást kell végezni (férfi vezetők összes fizetése/férfi vezetők száma, ill. női vezetők összes fizetése/női vezetők száma. Ha adott nemű felsőszintű vezető nincs a vállalatnál, az átlag értéke legyen 0.)

A feladat megoldását a 23. ábrán láthatjuk. A megoldásban kételemű tömböket használtunk, amelyeknek az indexét a rekordból olvasott NEM nevű változó aktuális értéke adta (a feladat szerint ez az érték vagy 1 vagy 2).

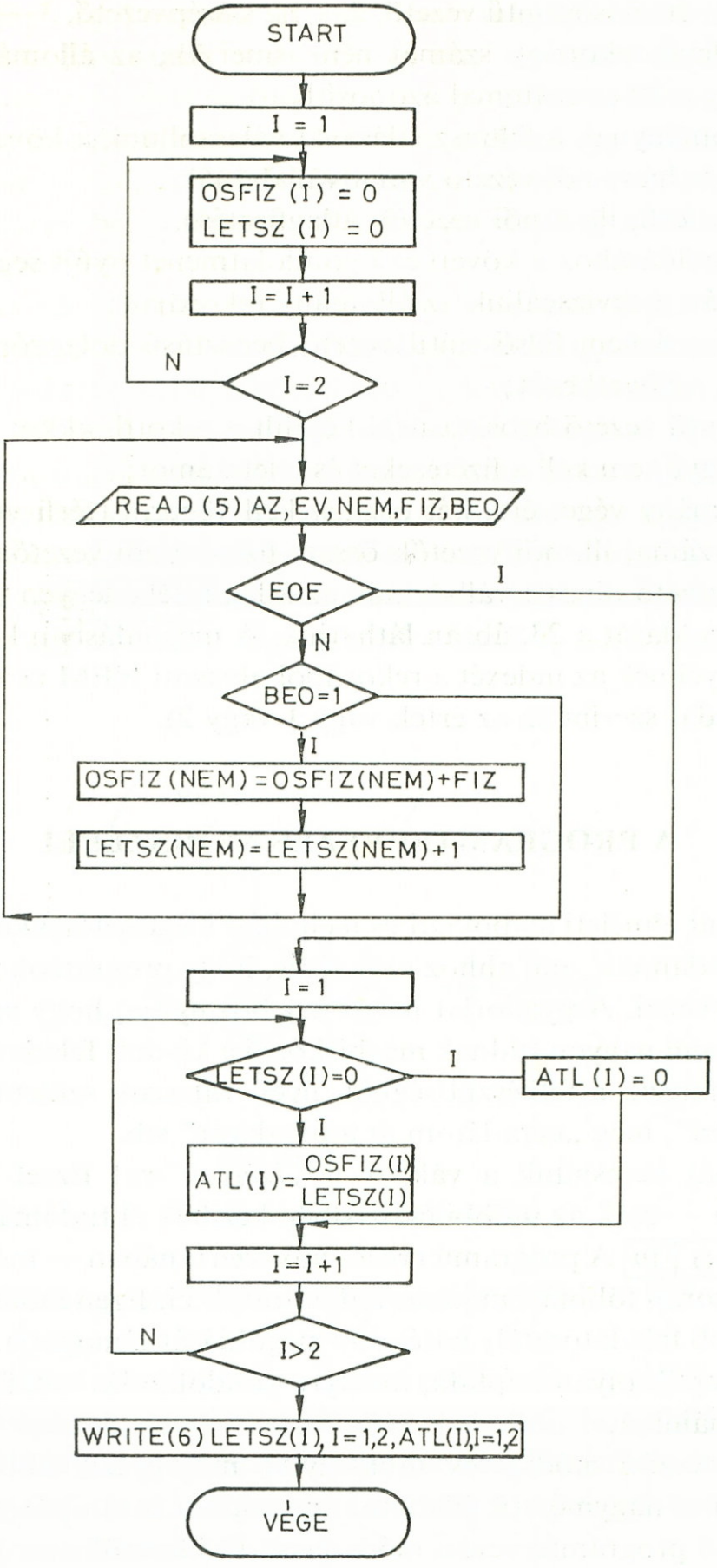
A PROGRAMTERVEZÉS MÓDSZEREI

Az eddig tanult elméleti alapokkal és technikai kiegészítésekkel birtokában érezzük magunkat mindannak, ami ahhoz szükséges, hogy programok vezérlési szerkezetét meg tudjuk tervezni. A gyakorlat mégis azt bizonyítja, hogy az eddig tárgyaltak alapján a kezdők nem nagyon tudnak megbirkózni a kiadott feladatokkal. Arra a kérdésre, hogy mi hiányzik a felkészültségből, ilyen válaszok születnek: „nem tudom, hogy kezdjek hozzá", meg „nem látom át a feladatot" stb.

Meg kell tehát keresnünk a választ a *„hogyan"*-ra! Ezzel a problémakörrel — érdekes módon — csak az utóbbi évtizedben kezdtek el tudományos alapossággal foglalkozni. [●] [241] [19] A programtervezés módszertanában — mert erről van szó — az elmúlt évtized során többféle módszert alakítottak ki. Ezen módszerek mindegyike valamilyen nagyobb feladatosztály hatékony megoldását támogatja. Nincs tehát egyedül üdvözítő „recept", olyan kaptafa, amelyre gondolkodás nélkül rá lehetne húzni felmerülő problémáinkat.

A programtervezési módszerek mindegyike megegyezik abban, hogy előnyös oldaluk elsősorban a nagyméretű feladatok megoldása során jelentkezik. Így a most bemutatásra kerülő programtervezési módszernél is felmerülhet a „minek a dolgokat komplikálni" kérdés. De vigyázat! Ami egy egyszerűbb és a középiskolás diák szintjén

tárgyalható algoritmus esetében komplikált megközelítési mód, az nagy, életből adódó probléma megoldásakor értékes segítség.



23. ábra. Fájl feldolgozás EOF kezeléssel

A funkcionális lebontás módszere

E programtervezési módszer lényegét a következőkben foglalhatjuk össze:

A program vezérlési szerkezetének (tehát algoritmusának) a tervezése előtt kialakítjuk a megoldandó feladat funkcionális leírását, vagyis elkészítjük:

— az eredményadatok leírását,

— a bemenő (input) adatok leírását,

— a megoldást befolyásoló tényezők vizsgálatát.

Ezt követően az elkészítendő vezérlési részt elkezdjük lebontani funkciókra. Ennek során alapvetően azzal foglalkozunk, hogy a készülő programnak MIT kell csinálnia és nem törődünk azzal, hogy ezt HOGYAN valósíthatjuk meg. Ily módon kialakul egy tevékenységlánc, amelynek egy-egy „láncszeme" a készülő program egy-egy önálló funkciót megvalósító része. Ezeket a funkciójukban önálló részeket *moduloknak* nevezzük. A modulnak három jellegzetessége van:

— A modul funkciója: vagyis, hogy a modul *mit* valósít meg a végrehajtáskor, milyen adattranszformációk (adatátalakítások) történnek a modulban.

— A modul logikája: a funkcióban leírtak *hogyan* valósulnak meg, milyen a vezérlés szerkezete.

— A modul kapcsolata a környezettel: milyen bemenő és kimenő adatokkal *kapcsolódik* a modul a környezetéhez. ▲ 237 20

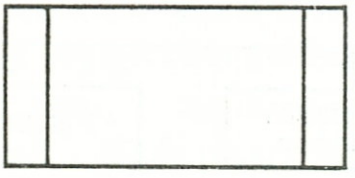
A funkcionális lebontáson alapuló programtervezés egyik problematikus pontja, hogy ugyanazt a feladatot többféleképpen lehet funkciókra bontani, s ezek között a lebontások között lehet jó és kevésbé jó lebontás. Jónak tekinthetünk egy lebontást akkor, ha a következő feltételek teljesülnek:

— egy modul egy funkciót valósít meg, vagyis a modul belső szerkezete egységes;

— a modulok közötti kapcsolódás csak adatkapcsolat szintjén valósul meg (vagyis egy funkciót nem több modul valósít meg).

Az elmondottak gyakorlati alkalmazásához szükségünk van egy új fogalom és jelölés bevezetésére a blokkdiagram jelölésrendszerében, s ez a modul programozástechnikai megfelelője, az *alprogram.*

Az alprogram (vagy idegen szóval *szubrutin*) formailag önálló program, amely egy konkrét funkciót valósít meg. A vezérlést a főprogramtól vagy egy másik alprogramtól kapja, ahonnan a rá vonatkozó hívási utasítás van. Az alprogramok egymással és a főprogrammal csak adatkapcsolatban vannak, ami azt jelenti, hogy egy alprogram input adatszükségletét, valamint az általa kiszámított outputokat „közlekedtetjük" a hívó és a hívott programrészek között. Blokkdiagram formában történő program



24. ábra. Alprogram

tervezésekor az alprogramot a 24. ábrán látható szimbólummal jelöljük. A négyzet belsejébe a hívott alprogram nevét tüntetjük fel, és azoknak az adatoknak, ill. tömböknek a nevét, amelyek „közlekednek" a két rutin között.

A funkcionális lebontás — mint módszertan — nem követeli meg, hogy a tervezés során kialakított valamennyi modulból önálló alprogram legyen. Elképzelhető ui., hogy egy modulról kiderül, hogy a megvalósítása (a logikája) egy vagy két számítógépes utasítással lehetséges. Ilyen esetekben e néhány utasítás önálló alprogramként való szerepeltetése csak akkor kívánatos, ha valami nagyon fontos vagy nagyon gyakori funkciót valósít meg.

A funkcionális lebontás módszeréről elmondottakat a következő feladat megoldásán kövessük végig.

Egy gyár összevont exportértékesítési adatairól a következő rekordképű állomány áll rendelkezésre:

— a gyártott termék kódja;
— értékesítési reláció (1 — tőkés; 2 — szocialista);
— értékesített mennyiség;
— egységár (dollár vagy rubel, az értékesítési reláció szerint);
— a vevő kódja.

A vevők adatait egy másik állomány tartalmazza, amelynek rekordképe:

— a vevő kódja,
— a vevő neve,
— a vevő székhelye.

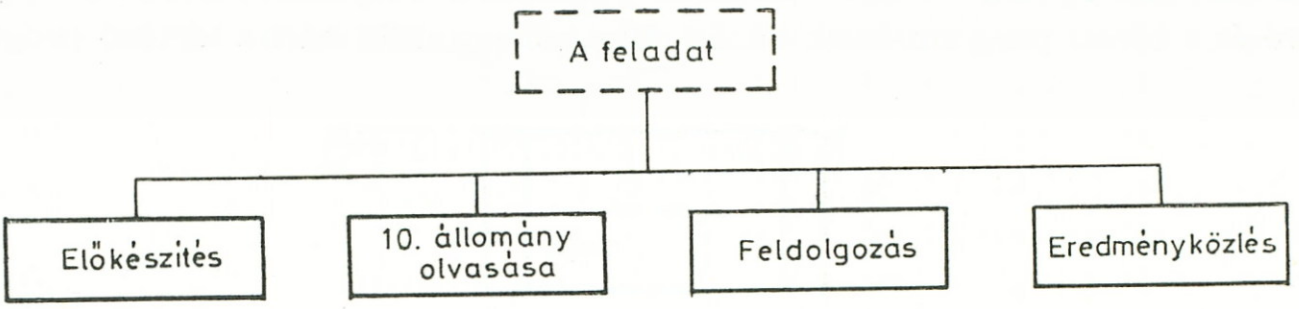
Készítsünk algoritmust, amely meghatározza a szocialista relációkban értékesített termékek összértékét, a tőkés piacra való értékesítés összértékét és egy listát, amely tartalmazza a tőkés vevő nevét, székhelyét és az általa vásárolt áru értékét. (Annak érdekében, hogy a feladat ne legyen túl nehéz, feltételezzük, hogy egy vevő számára csak egy értékesítés történt. Erre utal a feladatban szereplő „összevont" szó.)

A termék-fájl azonosítója legyen 10, a vevő-fájlé 20. Az input, az output és a megoldást befolyásoló feltételek megadásával első, induló lépésünkön túl is vagyunk, rendelkezésünkre áll a feladat funkcionális leírása (tudjuk, hogy mit kell csinálni).

A feladatot elemezve a következő funkciók adódnak:

1. Előkészítés.
2. Olvasás a 10. állományból.
3. Az olvasott rekord feldolgozása.
4. Eredményközlés

Grafikusan ábrázolva pl. így:



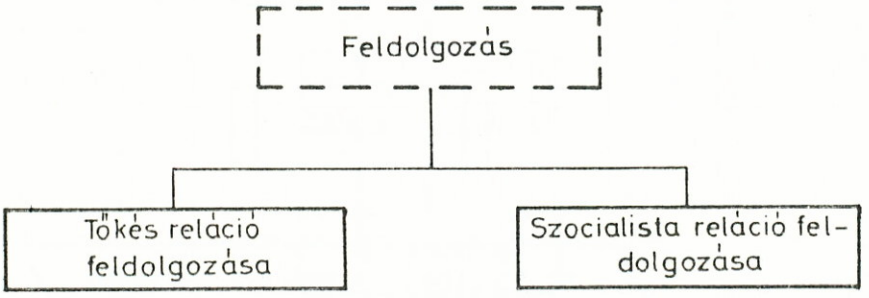
25. ábra. Funkciók a feladatban

Az egyes funkciókat tovább vizsgálva a 3. kíván további funkcionális lebontást.
A feldolgozási tevékenység kétféle lehet:

3.1. a tőkés relációba menő áru rekordjának feldolgozása;

3.2. a szocialista relációba menő áru rekordjának feldolgozása.

Ismét grafikusan szemléltetve:



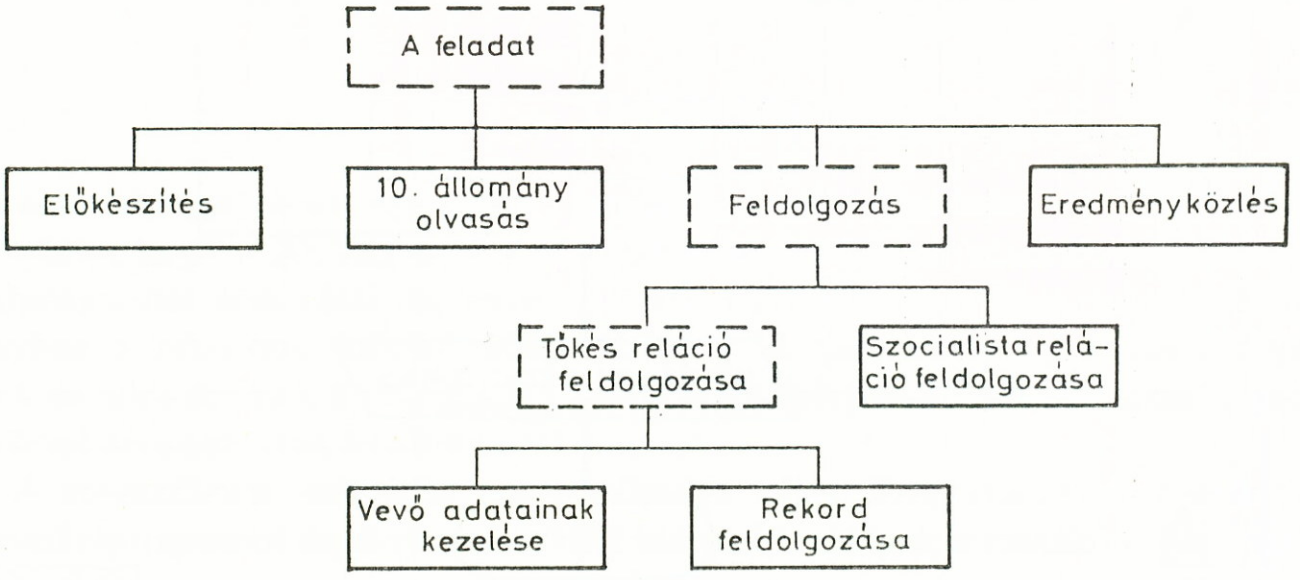
26. ábra. Egy funkció tovább-bontása

További meggondolást csak a tőkés reláció érdemel, ez is két újabb funkciót tartalmaz:

3.1.1. a vevő adatainak kikeresése a vevőkódhoz;

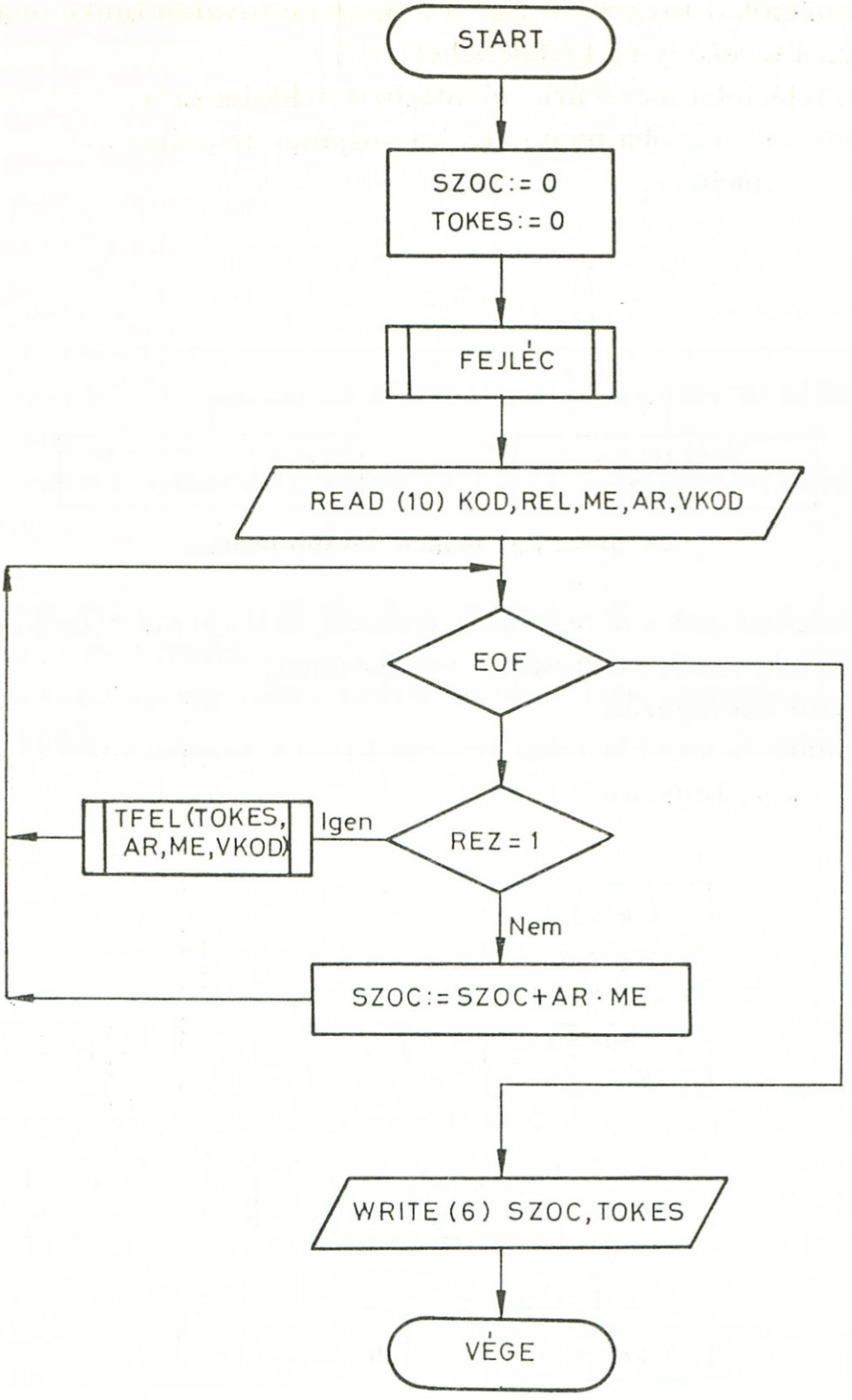
3.1.2. a rekord feldolgozása.

A teljes funkcionális hierarchia tehát így néz ki: (A szaggatottan rajzolt funkciók „szétbomlottak" újabb funkciókra).



27. ábra. Teljes lebontás

A 27. ábrán egybefüggően rajzolt funkciók egyenkénti elemzése alapján kialakíthatjuk ezeknek a funkcióknak a logikai szerkezetét. Ekkor derül ki, hogy egy-egy funkció (pl. a szocialista relációban való értékesítés feldolgozása) egyetlen vagy néhány művelettel megoldható. Mindezek végiggondolása után úgy döntünk, hogy első lépésben két alprogramot fogunk készíteni, az egyik az előkészítési fázisban a készítendő lista fejlécét valósítja meg, a másik a tőkés piacra menő export feldolgozását. Az így kapott megoldást a 28. ábra szemlélteti.



28. ábra. A feladat megoldása

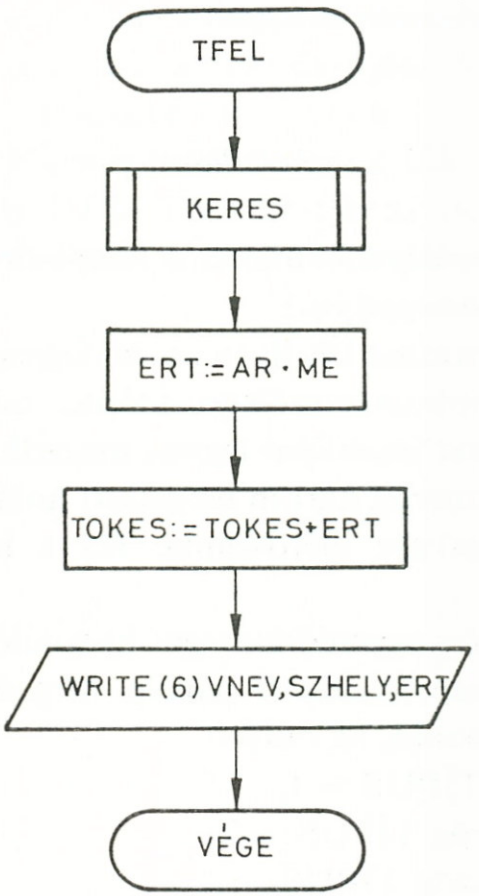
(A fejlécíró alprogram funkciója, hogy a lista elején magyarázó szöveg jelenjék meg a lista tartalmára vonatkozóan.) Ilyen fejlécet mutat a 29. ábra.

```
TOKES PIACRA TORTENO ERTEKESITESEK
VEVO NEVE        SZEKHELYE        ARU ERTEKE
```

29. ábra. Fejléc

Ezt az alprogramot nem készítjük el, mert a szövegkezelést nem tanultuk, ehhez konkrét programozási nyelv ismerete volna szükséges.

A tőkés piacra való értékesítést feldolgozó alprogram a következő formájú:



30. ábra. A TFEL alprogram

Ebben az alprogramban egy újabb alprogram került meghívásra (KERES). Ennek a feladata, hogy a 20. fájlból a kódszámhoz tartozó vevő adatait megkeresse. Ezt az alprogramot sem készítjük most el, mert működése attól függ, hogy a 20. állományban a rekordok milyen módon érhetők el. (Hatékony megoldás érdekében ennek az állománynak a rekordjait közvetlen elérésűre kell szervezni, ezzel a problémakörrel azonban csak később találkozunk.)

A megszületett megoldás ennek ellenére teljes. Programozásakor kell a két hiányzó alprogramot megírni a most még hiányzó ismeretek birtokában. | ● | 243 | 22 | | ⊙ | 241 | 21 |

Kérdések

1. Mi a program, milyen részekből áll?
2. Mi az algoritmus?
3. Milyen általános műveleteket ismersz?
4. Milyen műveleteket értelmezünk logikai típusú adatokkal?
5. Milyen szabályokat kell figyelembe venni az értékadó utasítás használatakor?

6. Mi a blokkdiagram szerepe?
7. Hogyan hivatkozunk tömbelemekre?
8. Mi a ciklus, milyen szerepe van a programozásban?
9. Mi az alprogram, és mi a szerepe a programozásban?
10. Mi a funkcionális lebontás módszerének lényege?

Feladatok

1. Készíts algoritmust, amely alkalmas arra, hogy óra, perc, másodperc formában megadott számhármasokat összeadjon!

a) Készítsd el az algoritmust két ilyen szám összeadására!

b) Készítsd el az algoritmust tetszőleges számú számhármas összeadására! Az összeadandó időadatok száma inputként legyen megadható.

2. Szerkeszthető-e háromszög három megadott hosszúságú szakaszból? Az algoritmus inputja a három szakasz mérőszáma. Ha a háromszög megszerkeszthető P:= 1, ha nem, P:= —1.
(A megoldáshoz a háromszög egyenlőtlenséget használd fel!)

3. Milyen háromszög szerkeszthető három megadott hosszúságú szakaszból? Az input a 2. feladathoz hasonló, az output
— ha általános háromszög TÍPUS:=1,
— ha egyenlőszárú háromszög TÍPUS:=2,
— ha egyenlőoldalú háromszög TÍPUS:=3.

4. A 2. és 3. feladat megoldását egy-egy alprogrammá alakítva készíts algoritmust, amely megvizsgálja, hogy három megadott szakaszhosszból szerkeszthető-e háromszög, és ha igen, akkor milyen!

5. Egy háromszög három oldalának mérőszáma X, Y, Z. (Ezek input adatok.) Derékszögű-e a háromszög?
Ha igen, DEREK:=1, ha nem, DEREK:=—1.

6. Egy háromszög három oldalának mérőszáma A, B, C.
Egy másik háromszög három oldalának mérőszáma D, E, F. Egybevágó-e a két háromszög?

a) a két háromszög megfelelő oldalainak megadási sorrendje megegyezik;

b) ez a sorrend nem egyezik meg a két háromszögnél.

7. ALFA, BETA, GAMMA, DELTA egy négyszög négy szögének mérőszáma fokban. Húrnégyszöghöz tartoznak-e? Ha igen, egy HUR nevű változó legyen 1, ha nem, akkor legyen —1.

8. Mértani közép számítás.
Számkettesek alkotnak egy-egy rekordot. Számítsuk ki és nyomtassuk a számkettesek mértani közepét! Az inputban az utolsó számkettest egy dupla 0-t tartalmazó zárórekord követi.

9. Egy rekordként olvassunk be egy 30 elemű tömböt!
Igaz-e az a feltevés, hogy a páros indexű elemek összege nagyobb egy előre megadott értéknél? (Ezt az értéket — még a tömb elemei előtt — inputként olvastassuk be.) Ha a feltevés igaz, az IGAZ változó értéke legyen 1, ha nem igaz, legyen —1.

10. Az I. fejezet olvasmányában szereplő áruházi rendszer mintájára készítsetek algoritmust, amelynek inputját kódszámok alkotják. A kódszámokról tudjuk, hogy 1 és 100 közé esnek. Az utolsó kódszám után —1 áll.
A tárban található egy NEVEK és egy ÁRAK nevű 100—100 elemű tömb. Készítsünk listát a vevő számára, amely kiírja a vásárolt áruk nevét és árát, a számla végére pedig a fizetendő összeget! Ha hibás kódszámot találunk, írjuk ki a 8. sz. hibafájlba!

IV. FEJEZET

Adatábrázolás a számítógépben

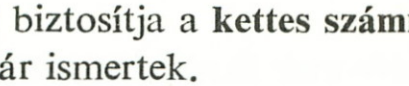
A számítógépes adatfeldolgozás folyamatából ez ideig megismerkedtünk a programokat alkotó adatstruktúrák és a vezérlést leíró algoritmusok fontosabb kérdéseivel. Ezek ismeretében azonban újabb problémákkal kerülünk szembe.

A feldolgozáshoz az szükséges, hogy mind a program, mind az adatstruktúrák bekerüljenek a számítógépbe, s ott végbemenjen a feldolgozásnak nevezett tevékenység, amelynek végén megszületnek a feldolgozási eredmények.

Mivel a feldolgozást a gép végzi, lényeges kérdésként vetődik fel, hogy a programok és az adatok hogyan ábrázolódnak a gépben. Hiába ismerjük az adatstruktúrákat és az algoritmusokat, ezek önmagukban még nem biztosítják a gép működését. A feldolgozás ideje alatt az adatok a gépben tárolódnak, ezért a *gépben való tárolás belső adatábrázolással* történik. Alapvető problémánk tehát az lesz, hogy **hogyan történik a számítógépben a belső adatábrázolás.**

A számítógépben való adattárolást belső adatábrázolásnak nevezzük. Az adatok belső ábrázolása többféle lehet. Másképp ábrázoljuk pl. az alfanumerikus jelsorozatokat (szövegeket), és másképp a számkonstansokat. Sőt, maguk a számértékek is többféle módon jelenhetnek meg a számítógépben. Ismét másképpen kerülnek tárolásra a programutasítások.

A belső ábrázolásnak alapvetően a gép hardver adottságához kell igazodnia. Ez a hardver adottság pedig röviden úgy fogalmazható meg, hogy a számítógép minden információt **két állapotú elemek** sorozatával képes tárolni. Olyan belső kódrendszerre van tehát szükség, amelyben az alkalmazott jelek száma mindössze kettő.

Ilyen kódrendszer alapját biztosítja a **kettes számrendszer,**  amelyet korábbi tanulmányaitokból már ismertek.

A kettes számrendszerben való ábrázolás tehát azért fontos, mert jól fel lehet használni az adatok számítógépben való ábrázolásának fizikai megvalósításakor. Ugyanakkor fennáll e számrendszernek az a hátránya, hogy már viszonylag nem nagy tízes rendszerbeli számok bináris megfelelője is igen hosszú bitsorozatot eredményez, amely az ember számára kényelmetlen.

E probléma áthidalására használható a számrendszerek egymásba való átszámításának speciális esete.

Ha egy számrendszer alapszáma egy másik számrendszer alapszámának egész kitevős hatványa, az átszámítás (konvertálás) igen egyszerűen elvégezhető. A konver-

tálás menetét egy konkrét példán mutatjuk be, anélkül, hogy a matematikai bizonyításba belemennénk.

Legyen az egyik számrendszer alapszáma 8, a másiké 2. Mivel $8 = 2^3$, ezért a következők szerint járhatunk el. A kettes számrendszerbeli számot a bináris ponttól balra (s ha a szám nem egész szám, attól jobbra is) bit-hármasokra (triádokra) osztjuk, majd az így kapott bithármasokat mint önálló nyolcas számrendszerbeli számjegyeket értelmezzük. Pl.:

$$(1100110.1)_2 = \underbrace{001}_{1}\ \underbrace{100}_{4}\ \underbrace{110}_{6}\ \underbrace{100}_{4} = (146{,}4)_8$$

Ezt tízes számrendszerbelivé már könnyen átalakíthatjuk az ellenőrzés céljából:

$$64+32+6+4/8 = 102{,}5.$$

Az előbbiek alapján a **nyolcas számrendszerben** való ábrázolás a bináris ábrázolásnak triádonként való értelmezése, s ez megkönnyíti a bináris értékekre való hivatkozást.

Ha az elmondottakat újraértelmezzük oly módon, hogy nem triádokat, hanem bitnégyeseket (tetrádokat) képezünk, új számrendszerbeli ábrázoláshoz jutunk. Ennek a számrendszernek az alapszáma 16, mivel $2^4 = 16$. Ez a számrendszer tizenhat számjegyet értelmez. Mivel a decimális rendszerből csak tíz számjegy kölcsönözhető, megállapodás szerint a latin abc első hat nagybetűje szolgál a hiányzó számjegyek szimbolizálására.

A következő táblázatban a **tizenhatos számrendszerben** értelmezett 16 „számjegy" értékét találhatjuk tízes, kettes és nyolcas rendszerben felírva.

Tízes	Kettes	Nyolcas	Tizenhatos
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

A tizenhatos számrendszer a benne szereplő betű alakú számjegyek miatt nehézkesnek látszik. A tízes számrendszerben gondolkodó ember számára első ránézésre furcsának tűnik pl. ez a szám:

10AB,

ami nem más, mint $1\times16^3+0\times16^2+10\times16^1+11\times16^0$, vagyis $4096+160+11 = 4267$.

A számítástechnikában mindezek ellenére azért használatos, mert — mint később látni fogjuk — a számítógépek nagy része a számjegyeket négy biten ábrázolja. A tizenhatos számrendszer a négy bitet teljesen kihasználja.

Hogyan lehetséges ezek után számok ábrázolása? A számítógépen kívüli adatábrázolás adathordozókon történik. A számítógépben tárolásra kerülő adatok tárolóközege a hardver. A központi tár hardver kialakítása különböző hosszúságú bitsorozatok tárolását biztosítja. A tárolás alapegysége a bájt, amely nyolc bit hosszúságú információ befogadására és tárolására képes. A bájtokból a programozó a következő méretű tárolóelemeket alakíthatja ki:

1 bájt = 8 bit,
2 bájt = 16 bit (félszó),
4 bájt = 32 bit (szó),
8 bájt = 64 bit (dupla szó).

A leggyakrabban használt tárolóelem a szó. ● 254 2

FIXPONTOS ÁBRÁZOLÁS

Egész számok esetében legegyszerűbb a fixpontos ábrázolás, ami az adott szám kettes számrendszerbeli alakjában való ábrázolást jelent.

Miután negatív számok is előfordulnak, ezért az első bitet előjelbitnek tekintjük a következők szerint:

A pozitív számokat bináris alakjuk ábrázolja, az *előjelbit értéke 0.*

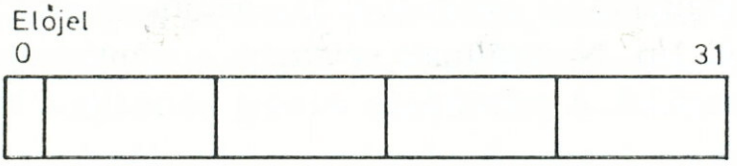
A negatív számokat a **kettes komplemens** ábrázolja, *az előjelbit értéke 1.*

Egy bináris szám kettes komplemensét úgy kapjuk meg, hogy a szám minden bitjét az ellenkezőjére változtatjuk, majd az így kapott számhoz hozzáadunk egyet. Az elmondottakat szemlélteti a következő példa:

alapszám:	10111
	01000
	1
kettes komplemens:	1001

● 254 3

A pozitív számok esetében a 32 bitet nem igénylő számok felső bitjei nullák. Negatív számoknál ezek a bitek csupa 1-et tartalmaznak (31. ábra).



31. ábra. Fixpontos ábrázolás

Az előjelet követő 31 biten a legnagyobb ábrázolható pozitív érték az, amely 31 darab 1 értékű bitből áll, vagyis $2^{32} - 1 = 2147483647 \sim 2 \cdot 10^9$. A negatív számok között a legkisebb (a komplemens ábrázolásból következően) csupa 0 bitet tartalmaz. Az előjelbit természetesen 1.

E belső ábrázolás lehetősége a felhasználó számára az általa használt programozási nyelv előírásaiból következik. Vannak nyelvek, amelyek lehetővé teszik a dupla szóban — tehát 64 biten való egész érték ábrázolását is. Ebben az esetben a legnagyobb ábrázolható pozitív szám $2^{64} - 1$, ami közelítőleg 1.84×10^{19}.

Az adatfeldolgozások nagy részénél olyan jellegű kérdésekre kell választ adni, hogy „hány egyed rendelkezik valamilyen meghatározott tulajdonságkombinációval" (pl. hány kitűnő rendű tanuló van az iskolában, aki fizikai foglalkozású szülők gyermeke). Ilyen esetekben a fixpontos ábrázolás kielégítő, egyben kívánatos is, mivel a számítógép a matematikai műveleteket a fixpontos számokkal végzi el a leggyorsabban, és a fixpontos ábrázolás mindig pontos. ▲ 253 4

Az olyan jellegű kérdésekre azonban, mint pl.: „a tanulók hány százaléka kitűnő rendű", már nem tudunk válaszolni, ha csak egész számokat lehet ábrázolni.

Mi módon lehetséges a nem egész (valós) számok ábrázolása?

LEBEGŐPONTOS ÁBRÁZOLÁS

A nem egész számok (valós számok) ábrázolása **lebegőpontos formában** lehetséges. A bináris lebegőpontos ábrázolás minden számot szorzat formájában ad meg. A szorzat első tényezője, a mantissza, egy valódi tört. A szorzat másik tényezője — a karakterisztika — azt adja meg, hogy a mantissza törtpontját hány helyértékkel kell áthelyezni (az előjeltől függő irányban). A törtpont áthelyezése után az ábrázolandó értékhez jutunk.

Negatív karakterisztika esetén a törtpont balra, pozitív karakterisztika esetén a törtpont jobbra mozdítandó el.

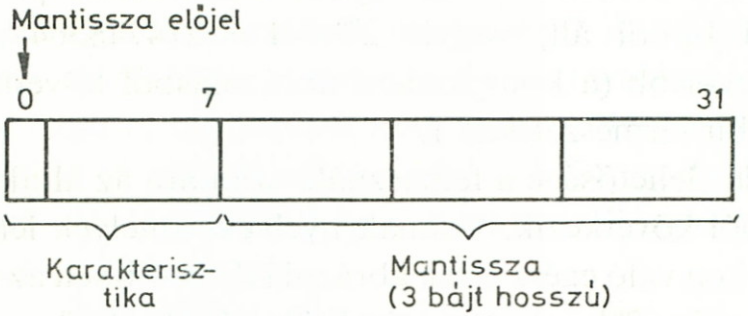
Az elmondottak alapján egy bináris lebegőpontos szám általános alakja a következő:

$$L = M \times 2^k,$$

ahol L az ábrázolandó számérték; M a mantissza, amelyre a következő nagyságrendi előírás áll fenn: $1/2 \leq |M| < 1$; k a karakterisztika.
A mantisszára fennálló előírás azt jelenti, hogy a valódi tört első értékes számjegye nem lehet 0. Tízes számrendszer esetében 1/10 a mantissza alsó korlátja, általában p alapú számrendszernél 1/p. Normáltnak nevezzük a mantisszát, ha az előbbi nagyságrendi előírásnak megfelel. A számítógép mindig normálja a lebegőpontos számot.

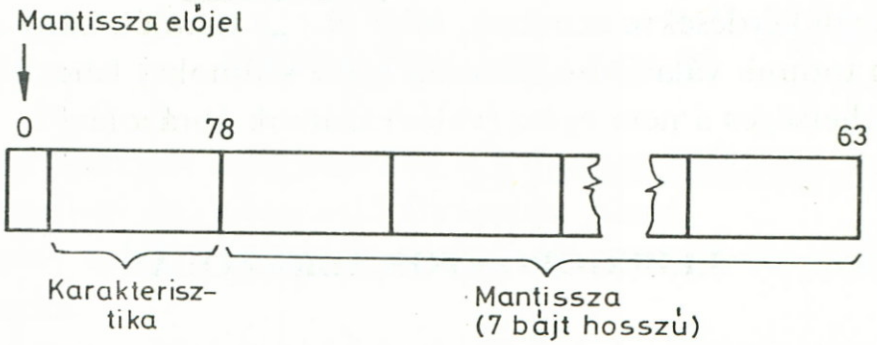
▲ 253 5

A lebegőpontos ábrázolás, amennyiben egy szóban történik, a 32. ábrán látható formájú.



32. ábra. Lebegőpontos ábrázolás

Duplaszóban való ábrázolás esetében az ábrázolt szám pontossága nő meg, vagyis a mantissza lesz hosszabb a 33. ábrán látható módon.



33. ábra. Dupla pontosságú lebegőpontos ábrázolás

Mindkét ábrázolás esetében a bal oldali első bájt legelső bitje a mantissza előjelét tartalmazza. A további hét biten helyezkedik el a karakterisztika. Értéke 0 és 127 közé eshet. Annak érdekében, hogy negatív kitevőket is lehessen ábrázolni, a karakterisztikában levő érték a tényleges kitevőnél 64-gyel nagyobb szám. Ily módon (ha ez a 64-es konstans levonásra kerül), a tényleges kitevő -64 és 63 közé eshet. Mivel a számítógép a karakterisztikában levő értéket 16 hatványaként értelmezi, egy lebegőpontosan ábrázolható szám a $5{,}4 \cdot 10^{-79}$ és $7{,}2 \cdot 10^{75}$ közötti értéket vehet fel.

Egy-egy számítógépes feladat megoldásának előkészítése során a programozó az input alapján dönti el, hogy melyek a valós adatok. Ezek lebegőpontosan kerülnek

tárolásra. Ezen belül ugyancsak az ő feladata eldönteni, hogy egy-egy adat — nagyságrendjénél fogva — megkívánja-e a dupla szóban való tárolást.

A számítógép mindig normált alakra hozza (normálja) a beírt számot. Ha egy művelet elvégzése során olyan eredmény keletkezik, amelynek mantisszája nem normált alakú, a karakterisztika egyidejű módosításával automatikusan megtörténik a normálás.

A lebegőpontos ábrázolásnak az ábrázolható számok körére vonatkozó előnyével szemben áll az ábrázolásból következő viszonylagos hátrány: a műveletvégzési idő lebegőpontos számokkal nagyobb, mint fixpontosokkal. Ennek okát abban kell keresni, hogy a műveletek végzésekor a mantisszán kívül a karakterisztikával is törődni kell.

BETŰK ÁBRÁZOLÁSA

Mindaddig amíg a számítógépeket csak olyan feladatok megoldására használták, amelynél kizárólag számításokat kellett végezni, a fixpontos és a lebegőpontos (közös névvel bináris) ábrázolás kielégítő volt. Amikor azonban a számítógépeket a tudományos számításokon túl adatfeldolgozási célokra is elkezdték használni, szükségessé vált a karakter adatok bevezetése és belső ábrázolhatóságának biztosítása.

Hogyan lehetséges ez? Miután a bináris kódrendszerek mindegyikében az 1 és 0 bitek meghatározott értéket (2 valamely hatványának meglétét vagy hiányát) reprezentálják, betűk ábrázolására ezek a kódrendszerek alkalmatlanok.

A megoldás csak az lehetett, hogy az egyes betűkhöz, speciális jelekhez és a számjegyekhez *hozzárendeljenek* egy-egy meghatározott bitkombinációt.

Az első probléma, hogy hány bitből kell állnia egy ilyen bitsorozatnak. A kérdést akkor tudjuk megválaszolni, ha tudjuk, hogy hány egymástól különböző karaktert tartalmaz a jelkészletünk.

Bitek száma	Lehetséges bitkombinációk száma
1	2
2	4
3	8
4	16
5	32
6	64
.	.
.	.
.	.
K	2^K

A számjegyek száma 10, a betűk száma 26 (az angol abc-ben, csak a nagybetűket figyelembevéve). A speciális jelek (írásjelek, matematikai jelek) nélkül tehát 36. Mielőtt azon fáradoznánk, hogy összegyűjtsük a szükséges speciális jeleket, nézzük meg a következő táblázatot, amely arra ad választ, hogy milyen kapcsolat van a bitek száma és az adott számú bittel felírható különböző bitkombinációk között:

A táblázat tanúsága szerint 5 bit ($2^5 = 32$) még a betűk és a számjegyek számára is kevés. Hat biten viszont 64 lehetőség van. Ebből következik, hogy $64 - 36 = 28$ egyéb jel számára marad hely, ha a keresett kódrendszerünket hat bit hosszúságúra választjuk.

Természetesen nagyon sokféle hatbites kód képzelhető el, a valóságban is több ilyen létezik. Az egyik legelterjedtebb ezek között az ún. **BCD kód** (nevét a Binary Coded Decimal Code = binárisan kódolt decimális kód elnevezés rövidítéséből nyerte).

A BCD kód

A decimális, vagyis a tízes rendszerbeli számok bináris kódolása:

Decimális alak	BCD kód
0	000000
1	000001
2	000010
3	000011
4	000100
5	000101
6	000110
7	000111
8	001000
9	001001

A bal oldali nullák elhagyásával éppen a számjegyek bináris alakját kapjuk meg. Ebben a kódrendszerben a számokat számjegyenként átírva (kódolva) ábrázolják.

Pl. az 1849 BCD alakja (tehát karakteres formája) a következő:

000001	001000	000100	001001
1	8	4	9

A BCD kódban a betűk és a speciális jelek kódolása az előbbi táblázat folytatásából adódik. A „decimális alak" oszlopba a 9 számjegyet követik az angol abc nagybetűi (mivel csak ezeket értelmezi a kódrendszer), majd a speciális jelek meghatározott sorrendben. A „BCD kód" oszlopban pedig tovább kell írni a bináris számokat egész 111111-ig. A táblázat további néhány sora tehát így néz ki:

A	001010
B	001011
C	001100
D	001101
E	001110
F	001111

A BCD kódot azok a számítógépek használják, amelyekben a szóhossz 24 bit (az általunk tárgyalt 32 bit helyett). Ezeknél a számítógépeknél egy szóban 4 karakter fér el, vagyis bármilyen szöveget 4 betűnként tud eltárolni. A BCD kódban levő számokkal műveletet végezni nem lehet.

Az EBCDIC kód

Az adatfeldolgozásra orientált számítógépek a BCD kód helyett az **EBCDIC** kódot használják, amely a BCD kód kiterjesztett változata. Nevét az angol Extended Binary Coded Decimal Interchange Code = kiterjesztett BCD kód elnevezésből kapta.

Az EBCDIC kód nyolc bit hosszúságú.

Nyolc biten $2^8 = 256$ egymástól különböző bitkombináció lehetséges. 256 különböző karakter azonban nem használatos, a nyolc bites kód hosszúnak látszik. A kód kialakítói azonban nem az egyszerű hozzárendelést választották a kódképzés szabályának. A nyolc bit két, egyenként négy-négy bit hosszúságú részre oszlik. Az első négy bit (amelyben egy tizenhatos rendszerbeli számjegy ábrázolható), a zónarész, a második négy bit (amelyen szintén egy tizenhatos rendszerbeli számjegy ábrázolható) a számrész.

binárisan:	1 1 0 0	1 0 0 1
	zóna-	szám-
	rész	
hexadecimálisan:	C	9

A 34. ábrán egy 16×16 méretű táblázatból mutatjuk be a legfontosabb részletet. Kiolvasható, hogy az EBCDIC kódban értelmezett fontosabb karaktereknek milyen zóna- és számrész felel meg.

A táblázatból az ábrázolni kívánt karakter oszlopához tartozó oszlopszámot — mint tizenhatos rendszerbeli számot — leolvassuk és zónarésznek tekintjük, a sorok sorszámát — szintén tizenhatos rendszerbeli számként — mint számrészt értelmezzük. A két négybites érték megadja az ábrázolni kívánt karakter EBCDIC kódját.

Vegyük példaképpen a következő karaktersorozatot: ADAT. EBCDIC kódban a bitsorozat így alakul (a bitsorozatot most a tizenhatos rendszerbeli megfelelő „számjeggyel" helyettesítjük):

C 1 | C 4 | C 1 | E 3

		ZÓNA									
Tizenhatos	Kettes	6	7	8	9	A	B	C	D	E	F
		0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	–									0
1	0001	/		a	j			A	J		1
2	0010			b	k	s		B	K	S	2
3	0011			c	l	t		C	L	T	3
4	0100			d	m	u		D	M	U	4
5	0101			e	n	v		E	N	V	5
6	0110			f	o	w		F	O	W	6
7	0111			g	p	x		G	P	X	7
8	1000			h	q	y		H	Q	Y	8
9	1001			i	r	z		I	R	Z	9
A	1010	^	:								
B	1011	'	#								
C	1100	%	@								
D	1101	–	'								
E	1110		=								
F	1111	?	"								

SZÁM

34. ábra. EBCDIC kódtáblázat (részlet)

Egy 32 bites szóban 4 bájt tárolható, a példánkban szereplő belső ábrázolás tehát éppen egy szóban fér el. Hosszabb szövegek természetesen több szóban, célszerűen tömbökben tárolhatók.

Zónázott decimális ábrázolás

Vizsgáljuk meg a 34. ábrán levő táblázat alapján, hogy hogyan lehet számokat ábrázolni. Vegyük észre, hogy valamennyi tízes rendszerbeli számjegy kódjának zónarésze 1111 (vagyis tizenhatos rendszerben F). Előjel nélküli számok ábrázolása tehát a 35. ábrán látható módon lehetséges. Az ábrázolt szám 269.

Binárisan	1111 0010	1111 0110	1111 1001
Hexadecimálisan	F 2	F 6	F 9

35. ábra. Előjel nélküli zónázott decimális szám

Minden számjegy ábrázolásához nyolc bit szükséges. Ezek a számok — akár csak a matematikában — pozitívnak értelmezendők. Előjeles számok esetén az előjel

négybites kombinációja az ábrázolandó szám legalacsonyabb helyértékű számjegyének zónarészében jelenik meg. Az előjeleknek megfelelő négybites kombinációk:

Bináris kép	Tizenhatos rendszerbeli alak	Előjel
1010	A	+
1011	B	—
1100	C	+
1101	D	—
1110	E	+
1111	F	+

Az EBCDIC kódban az 1100 és az 1101 előjelkódokat használjuk. Előjeles számok ábrázolását mutatja be a 36. ábra.

Bináris	1111 0010	1111 0110	1100 1001	= + 269
Hexadecimális	F 2	F 6	C 9	

Bináris	1111 0010	1101 0111	= − 27
Hexadecimális	F 2	D 7	

36. ábra. Előjeles zónázott decimális számok

A zónázott decimális kód az elmondottak alapján nem más, mint a számok karakteres ábrázolása. A 35. ábrán levő számok tehát az őket alkotó számjegyek karakterképeinek egymásutánja.

Éppen ezért azokkal a számokkal, amelyek ilyen belső kódban vannak nem is lehet műveletet végezni, ezek annak ellenére, hogy számjegyekből állnak, ugyanúgy karakter típusúak, mint a betűk. ● | 254 | 6

Az eddigiekből nem derült ki, hogy milyen előnyei vannak a 8 bites EBCDIC kódnak a 6 bites BCD kóddal szemben, csupán az a hátrány, hogy minden karakter 2 bittel több helyet igényel a számítógép központi tárában. A továbbiakban e kérdésre keressük a választ.

Tömörített decimális ábrázolás

A 34. ábrán levő EBCDIC kódrészletben látható, hogy valamennyi számjegy zónarésze 1111, vagyis F. Amennyiben tehát csak számokat kívánunk ábrázolni, ez a zónarész el is maradhat. Ha viszont elhagyjuk a zónát, nyolc biten nem egy, hanem két decimális számjegy ábrázolása válik lehetővé. Ilyen tömörítéssel az ábrázolandó szám által lefoglalt bitek száma nem egészen a felére csökken. ● | 254 | 7

A számok tömörítése:

Zónázott forma: F 1 F 8 F 4 F 9
Tömörített alak: 0 1 8 4 9 F

Az előjel nélküli számokat tehát úgy zónázza, hogy a jobb oldali bájt hátsó négy bitjén (ezen a helyen van az előjel zónázott esetben) egy F-et helyez el. (Ez előjel nélküli számot jelent, ami ugyanúgy mint a matematikában, pozitívnak értelmeződik.) Az 1 előtti 0 az 1-et tartalmazó bájt első négy bitje. Ez üres marad.

Ha a tömörítendő szám páratlan számú számjegyet tartalmaz, akkor a tömörített ábrázolásnál valamennyi bájt „tele lesz". Pl.:

Zónázott forma: F 2 F 3 F 4
Tömörített alak: 2 3 4 F

Előjeles számok tömörítése esetén ugyanez az eljárás, de ilyenkor a jobb oldali bájt hátsó négy bitjén megjelenik az előjel, ami lehet negatív (D), vagy pozitív (C) (ez most előjeles pozitív számot jelent). Pl.:

Zónázott forma: F 2 F 6 D 9
Tömörített alak: 2 6 9 D

Ha az előjeles szám páros számú számjegyből áll, a tömörítésnél itt is 4 üres bit keletkezik:

Zónázott forma: F 2 C 7
Tömörített alak: 0 2 7 C

A tömörített decimális ábrázolás esetén egy számjegy helyigénye 4 bit.

Vajon egy átlagosnak tekinthető adatfeldolgozási feladatban szereplő adatok hány százaléka numerikus, és hány százaléka alfabetikus?

Erre a kérdésre pontos választ adni nehéz. Azt bizonyosan állíthatjuk, hogy a numerikus adatok aránya nagyobb 50%-nál.

Mennyi az egy karakterre jutó átlagos helyfoglalás (bitekben)?

Ha fele-fele arányban lennének számjegyek és betűk, úgy $(8+4):2 = 6$ bit lenne az átlagos helyigény. Mivel a kisebb helyigényű számjegyek aránya nagyobb, ezért ez az átlag kevesebb 6-nál.

Így már érthető az EBCDIC kód előnye, hiszen $2^8 = 256$ különböző kódérték értelmezhető benne a BCD kód $2^6 = 64$ értékével szemben, miközben az átlagos helyigénye alatta marad a 6 bitnek. (Az EBCDIC kód nem használja ki a 256 féle lehetőséget, az ábrán is látszik, hogy üres, vagyis nem értelmezett helyek vannak a táblázatban.)

A számok tehát négyféle módon ábrázolhatók a számítógépben:

fixpontos bináris,
lebegőpontos bináris,
zónázott decimális,
tömörített decimális alakban.

A számítógépbe a kívülről bekerülő adatok mindig zónázott formájúak. Az alfabetikus adatok, továbbá azok a numerikus adatok, amelyekkel nem végzünk műveleteket, ebben a kódban is maradnak. Azokat a számadatokat, amelyeket műve-

letekhez használunk fel — programozási nyelv adta lehetőségek segítségével —, át kell alakítani fixpontos bináris, lebegőpontos bináris vagy tömörített decimális alakba.

A számítások eredményei a számítógépből való kihozatal előtt ismét zónázott formába kerülnek. [• | 255 | 8]

ADATÁBRÁZOLÁS A MIKROSZÁMÍTÓGÉPEKBEN

A mikroszámítógépek elsőnek megjelent típusainak belső szervezése a nyolc bit hosszú adat- és utasításábrázolást tette lehetővé. Ez azt jelenti, hogy szemben a nagyszámítógépek 32 bites belső tárolási lehetőségével, itt csak nyolc bites egységek állnak rendelkezésre. Ez elsősorban azért jelent problémát, mert fix- és lebegőpontos adatábrázolásra ugyanúgy szükség van, mint a nagyszámítógépekben. Ugyanígy igény van a karakteres ábrázolás lehetőségére is.

E probléma alapvető megoldása az volt, hogy több, egyenként nyolc bites egységben történik mind az adatok, mind az utasítások tárolása. (Ez persze a végrehajtás során jelent időveszteséget, s éppen ezért megindult a 16 és 32 bites processzorok fejlesztése is. Ma már ilyenek is vannak a piacon.)

Ugyanakkor a mikrogépekben való karakteres ábrázoláshoz az ASCII kódot használták fel. (Nevét az angol American Standard Code for Information Interchange elnevezés rövidítéséből kapta.) Az ASCII kód is nyolcbites kód, de létezik hat- és hétbites „szűkített" változata is. A mikrogépek a hétbites változatot használják. Az ASCII kódkészletét a 37. ábra szemlélteti.

Milyen előnyei vannak a kódrendszernek? Alapvetően az, hogy a nyolcbites gépben levő hétbites kód lehetővé teszi, hogy szükség esetén további információkat is tartalmazzon a „felesleges" nyolcadik bit. Ilyen más jellegű információt lehet pl. az input/output esetében bizonyos állapotok jelzése vagy olyan ellenőrzési funkció, amelyet EBCDIC kód esetében külön-külön kell megoldani, ahogy azt az V. fejezetben látni is fogjuk.

Kérdések

1. Mi a kettes számrendszer lényege?
2. Mi a bit?
3. Miért előnyös a kettes számrendszer használata a tízessel szemben?
4. Miért hátrányos a kettes számrendszer használata a tízessel szemben?
5. Hány „számjegyet" értelmezünk a tizenhatos számrendszerben?
6. Mit jelent a tizenhatos számrendszerben felírt A, B, C, D, E, F?
7. Mit jelent a fixpontos ábrázolás?
8. Milyen korlátai vannak a fixpontos ábrázolásnak?
9. Milyen adatokat célszerű fixpontosan tárolni?

sorszám	karakter	sorszám	karakter	sorszám	karakter	sorszám	karakter
0	NUL	32	SP (szóköz	64	(a)	96	
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	’	76	L	108	l
13	CR	45	—	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

37. ábra. ASCII kódkészlet

10. Mit jelent a lebegőpontos ábrázolás?
11. Mit értünk egy szám normálalakján?
12. Milyen értékeket vehet fel a mantissza normál alakban, lebegőpontos ábrázolással?
13. Milyen előnyei vannak a lebegőpontos ábrázolásnak a fixpontossal szemben?
14. Milyen hátrányai vannak a lebegőpontos ábrázolásnak a fixpontossal szemben?
15. Mit jelent a tömörített ábrázolás?
16. Hogyan változik az előjel tömörítéskor?

1. Írd fel a következő számok EBCDIC kódbeli megfelelőjét:

+378;693; — 5966!

2. Írd fel a következő számok tömörített ábrázolású megfelelőjét:

+378;693; — 5966!

3. Készíts algoritmust, amely legfeljebb 40 karakter hosszúságú szöveg karakteres tárolását biztosítja egy SZÖVEG nevű tömbben!

4. Készíts algoritmust, amely egy tetszőleges hosszúságú szöveg karakteres ábrázolását biztosítja! Az input első adata az aktuális szöveg karaktereinek száma, ezt követi, új rekordként, a tárolandó szöveg.

5. Egy BEHOZ nevű alprogram hívásának hatására a tárba töltődik egy 26 elemű BETŰK nevű tömb. A tömb i. eleme az angol abc i. betűjét tartalmazza karakteresen.

Rendelkezésünkre áll egy 72 karakter hosszúságú szöveg, amelyet úgy kódoltak, hogy a szövegben minden betű helyére az abc-ben néggyel rákövetkezőket írták (pl. „a" helyére „e"-t). A 23.—26. betűk helyére az 1.—4. betűket kódolták.

Készítsünk algoritmust, amely az input szöveget megfejti és kinyomtatja! A szövegben szóköz és írásjel nem fordul elő, ha mégis lenne, ezeket ne vegyük figyelembe.

Az alprogramot nem kell elkészíteni! A megoldásnál alkalmazzátok a funkcionális lebontás módszerét.

V. FEJEZET

Központi tár, központi egység

Az eddigiek során megismerkedtünk a leggyakoribb adattípusokkal, az egyszerűbb adatstruktúrákkal, megtanultuk, hogy miképpen lehetséges a problémák algoritmusát elkészíteni, valamint találkoztunk a belső adatábrázolás alapeseteivel.

Nem tudjuk azonban eddigi ismereteink alapján, hogy

— milyen tárolóközegben valósul meg a belső ábrázolás;

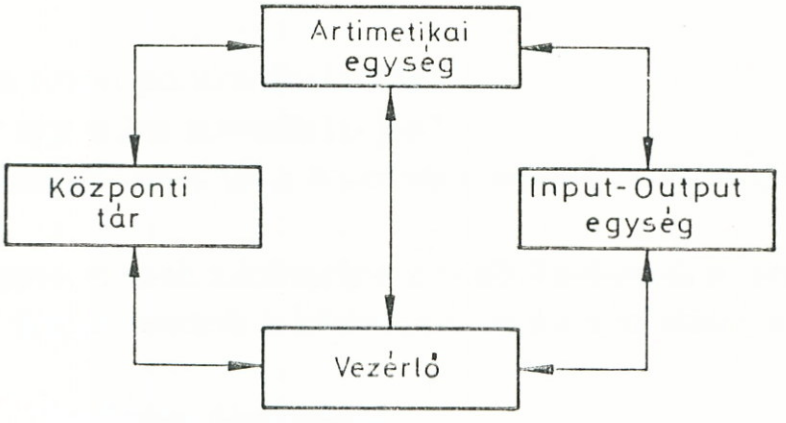
— miképpen realizálható az algoritmusban leírt vezérlés;

— hogyan kapcsolódik egymáshoz a megvalósított tárolóközeg és a vezérlést megvalósító eszköz.

Ennek a fejezetnek az alapvető problémája tehát, amelyet meg kell válaszolnunk, így fogalmazható meg: **Hogyan történik a számítógépben az adatok és a programok belső ábrázolása és tárolása? Milyen eszköz biztosítja ennek megvalósítását? Hogyan valósul meg egy program által leírt vezérlés a számítógépben? Milyen eszköz valósítja ezt meg? Egyszóval: hogyan történik a program végrehajtása?**

E kérdések megválaszolásához közelebbről kell megismerkednünk a számítógép felépítésével, az első fejezetben hardver erőforrásoknak nevezett eszközök közül **a központi tárral és a központi egységgel.** ● 258 1

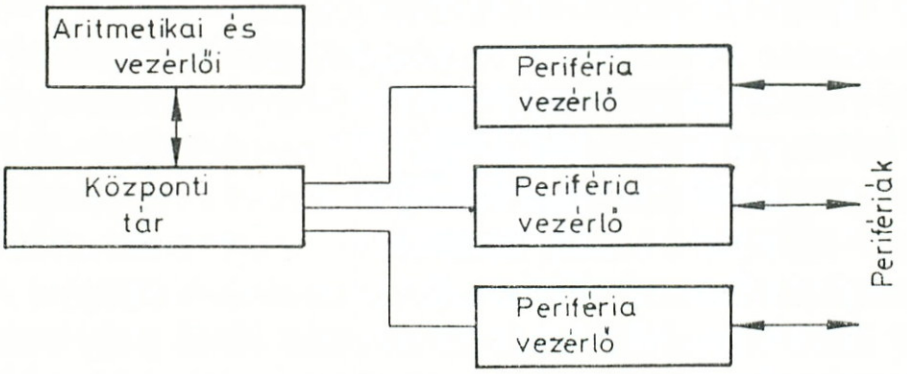
Az első számítógépek hardver felépítését a 38. ábra szemlélteti. Négy önálló funkcióval rendelkező egységet láthatunk az ábrán. Ezek: Az **aritmetikai egység,** amely a program által kijelölt műveleteket valósította meg. A **vezérlőegység,** amely a számítógép működése során az egymást követő tevékenységeket vezérelte. **A tár,** ahol a végrehajtandó program és a hozzá tartozó adatok kerültek elhelyezésre.



38. ábra. Az első számítógépek hardver felépítése

Az **input/output** berendezések, amelyeken keresztül a számítógép a külvilággal kapcsolatot tartott. A működés logikája szerint a vezérlő- és az aritmetikai egység volt a középpontban, ezt szolgálta ki a tár és az I/O berendezések.

Ehhez a kezdeti állapothoz képest az elmúlt évtizedekben alapvetően megváltozott a számítógép hardver felépítése. Ez a változás jól nyomonkövethető a 39. ábrán.



39. ábra. A korszerű számítógépek hardver felépítése

Az új hardver elemek megjelenésén túlmenőleg az első, ami szembetűnik, hogy az egész rendszer középpontjába a tár került, ehhez csatlakozik a hajdani aritmetikai és vezérlőegységek utóda, valamint a tárhoz csatlakoznak (most már áttételesen) az input/output berendezések is. A **központi tár** megtartotta eredeti funkcióját: az adatok, a programok és a gépet vezérlő információk tárolására szolgál. A központi egység két funkcionális részből tevődik össze: a végrehajtandó utasítást kiválasztó, valamint a végrehajtást előkészítő **vezérlőegységből,** ill. az utasításvégrehajtást elvégző **aritmetikai-logikai** egységből. Ezek működése során történik a szükséges tárcímek meghatározása (generálása), az elvégzendő utasítások sorrendjének eldöntése. A központi tár és a központi egység közötti adatmozgatásban fontos szerepet játszanak a **regiszterek** és az **adatsínek,** az input-output egységekkel való kapcsolattartásban a **csatornák.** Az adatmozgatás során esetleg fellépő adatátviteli hibák felderítése és kiküszöbölése szintén fontos feladata a számítógépnek. Végül, de nem utolsó sorban a modern számítógépek hatékony működésének elengedhetetlen hardver feltétele **a megszakítási rendszer.**

Ismerkedjünk meg ezeknek a funkcióknak a megvalósulásával.

A KÖZPONTI TÁR

A központi tár feladata, hogy egy probléma megoldása során mindazokat az információkat tárolja, amelyek a megoldás adott pillanatában szükségesek. Így itt kerülnek tárolásra a számítógép erőforrásaival gazdálkodó és a programok végrehajtását segítő rendszerprogramok — amikor szükség van rájuk —, a végrehajtásra kijelölt program (vagy annak egy része) és a hozzátartozó adatokból az éppen szükségesek.

A tárolás fizikai megvalósítása mindaddig, amíg nem sikerült olcsón előállítani magas integráltságú áramköröket, ferritgyűrűk segítségével történt. | ● | 259 | 2 |

A korszerű számítógépeknél a ferritgyűrűt is felváltották az integrált áramkörök. Ezzel a tár fizikai kiterjedése a korábbi méretek töredékére csökkent. | ● | 260 | 3 |

Tudjuk, hogy az információ alapegysége a bit, a kettes számrendszer egy alapjele. A központi tárban megvalósított tárolás alapegysége a **bájt**. Egy bájt nyolc bitet tartalmaz. Az egy bájtnyi információ elhelyezésére szolgáló tárhely a **rekesz.** A központi tár tehát a felhasználó szemszögéből rekeszekből épül fel. Minden rekesznek önálló címe van. A **cím** a rekesz azonosítására szolgáló bináris szám, amelyet tizenhatos rendszerbeli számként szokás megadni. | ▲ | 256 | 4 | A címek fixen vannak a tárrekeszekhez rendelve. Egy konkrét cím tehát mindig ugyanazt a tárrekeszt azonosítja. A cím segítségével válik lehetővé a rekesz feltöltése és az ott tárolt érték visszanyerése. A rekeszekben elhelyezett információt a *rekesz tartalmának* tekintjük. A tartalom természetesen időről időre változhat. A rekesz tartalma mindig egy bitsorozat, amelynek értéke attól függ, hogy milyen kódrendszerben történt a belső ábrázolás. | ▲ | 256 | 5 |

A belső kódrendszertől függően egy bájtban ábrázolható: egy 0—255 közötti bináris egész szám (fixpontos bináris ábrázolás esetén), két decimális számjegy (tömörített decimális kód alkalmazása esetén), egy alfanumerikus karakter (EBCDIC kódban való ábrázolás esetén). Miután a számítógépes feldolgozás során ennél nagyobb tárolóegységekre is szükség van (pl. lebegőpontos ábrázoláskor), a bájtból mint a tárolás alapegységéből 2 bájtos *félszó*, 4 bájtos *szó* és 8 bájtos *dupla szó* méretű fizikai tárolási egységek alakíthatók ki. Ezek kezdő címe mindig a legalacsonyabb bájt címével azonos.

Amikor egy adatot meg kívánunk őrizni a központi tárban, három dolgot kell megadnunk:

— azt a kezdőcímet, amelytől kezdődő rekeszekben a tárolás történjék;

— azt a hosszt, amely megadja, hogy a tárolás hány rekeszben történjék (ez értelemszerűen 1, 2, 4, 8 lehet); | ▲ | 256 | 6 |

— magát a tárolni kívánt adatot.

A ma használatos programozási nyelvekben a tárban őrzött adatokra való hivatkozás *szimbolikus címekkel* történik. Minden szimbolikus cím — az adott programozási nyelv szabályainak megfelelően — egy **változónév.** Amikor tehát a korábbiakban, az algoritmusok tervezésekor azt mondtuk, hogy egy-egy változó valamilyen értéket vesz fel, ugyanezt most már úgy fogalmazhatjuk, hogy egy szimbolikus címhez tartozó tárterületen (egy vagy több rekeszben) egy adat került tárolásra.

Neumann János óta a központi tárban nemcsak adatokat, hanem programutasításokat is tárolhatunk.

Az utasítások — a tár fizikai felépítése miatt — szintén bináris jelek, vagyis 1 és 0 bitek sorozataként tárolódnak.

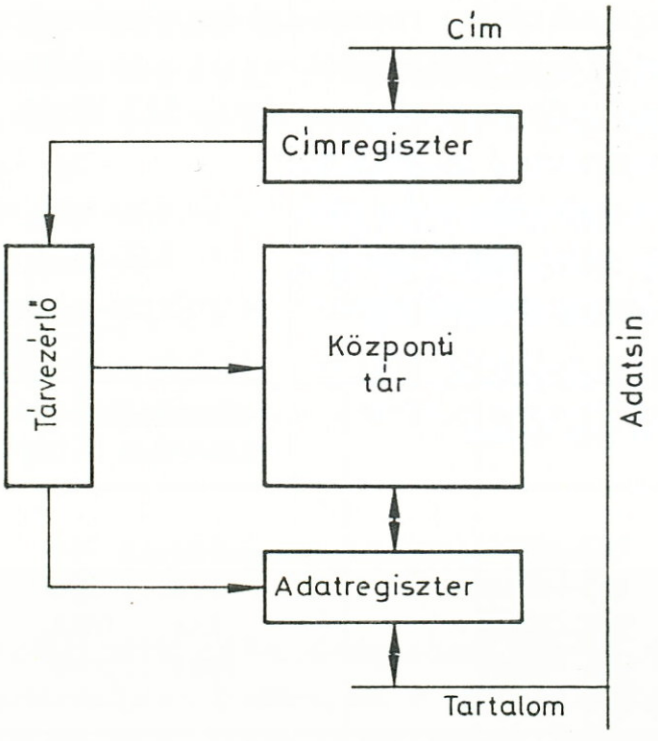
Amikor egy futtatandó program számára tárterület lesz kijelölve, ezen a területen belül vannak az adatok és a programutasítások is. Ha a programozó minden adathivatkozása korrekt, minden szimbolikus címen a megfelelő adatot tudja kezelni. Hibás

hivatkozásoknál (pl. egy 23 elemű tömb 34. elemébe akarunk értéket elhelyezni) egy más célú tárterületre illegális beírás történik. Ha ez a tárcím a program számára biztosított területen belül van, a program tovább fut, de biztos, hogy valami hiba lesz. Ha a hivatkozott cím a program saját területén kívül esne, ezt a felülírást a vezérlőprogram nem engedi meg. A saját területen való hibás tárhasználat eredményeképpen vagy egy olyan cím lesz felülírva, ahol adat van, vagy egy olyan cím, ahol utasítás. Ez utóbbi okozza a nagyobb problémát, hiszen ilyen esetben a tárban levő program nem azonos a programozó által elképzelttel. ⊙ | 263 | 7

A központi tár a számítógép többi részéhez **regisztereken** keresztül kapcsolódik. A regiszter adatok és utasítások átmeneti ideig való tárolására szolgáló, igen gyors működésű áramkörökből felépített hardver elem. A számítógépben levő regiszterek egy része speciális célú, más részük a programozó által is használható általános célú.

Speciális célú a tárhoz kapcsolódó két regiszter is, a **címregiszter** és az **adatregiszter**. A címregiszterben kerül elhelyezésre az a tárcím, amelyre a művelet vonatkozik. (Ez a művelet vagy a címre való beírás, vagy a címről való olvasás.) A művelet végrehajtását a tárvezérlő irányítja. Az adatregiszterből íródik be az információ a tárcímre, ill. ide lesz elhelyezve a kiolvasás eredményeképpen a cím tartalma. Az elmondottakat a 40. ábra szemlélteti.

Az az időmennyiség, amely alatt a tár egy beírási vagy egy kiolvasási műveletet végrehajt, a tár egyik fontos jellemzője. Ezt az időszükségletet **elérési időnek** nevezzük. A félvezetős tárak megjelenése előtt a tárból való olvasás ún. destruktív olvasás volt, ami azt jelenti, hogy az olvasás hatására az érintett rekeszekben levő információ törlődött. Mivel ez nem megengedhető, ezért a kiolvasott bitsorozatot vissza kellett állítani (idegen szóval regenerálni kellett a tártartalmat). Ehhez persze időre volt szükség.



40. ábra. A tár fontosabb regiszterei

Amíg az érintett cím tartalma nem volt visszaállítva, a tárhoz nem lehetett ismételten hozzáfordulni (nem lehetett újabb parancsot adni a tárnak).

A tárhoz való kétszeri hozzáférés közötti legrövidebb időt a tár *ciklusidejének* nevezzük. Regenerálást igénylő táraknál (ilyen volt a ferrites tár is) a ciklusidő az elérési idő többszörösét is elérte.

A számítógép központi tárával kapcsolatosan a felhasználó alapvetően két kérdést szokott feltenni:

— mekkora az adott tár kapacitása, vagyis egyidejűleg hány adat helyezhető el;

— mekkora az adott tár ciklusideje, vagyis mennyi idő szükséges egy adat tárolásához, ill. visszakereséséhez.

A központi tár kapacitásának mérésére használatos mérőszám a címezhető alapegység 2^{10}-szerese. Az alapegység esetünkben a bájt. | • | 261 | 8 | A 2^{10} bájt, vagyis az 1024 bájt szokásos jelölése *1 Kbájt (kilóbájt)*.

A számítógépek központi tára modulárisan építhető ki. Minden gép esetében létezik egy minimális tárméret, amelyet egy maximális méretig lehet növelni. A tárméret növelését a feladatok számának állandó növekedése indokolhatja. A legkisebb méretnek a működőképesség szab határt, a legnagyobbnak gazdasági és technikai megfontolások.

Néhány évvel ezelőtt a számítógépek nagyságrendi kategóriákba való besorolásánál az egyik szempont a központi tár mérete volt. Miután az integrált áramkörök a tárépítés területére is betörtek, már nem az az alapvető probléma, hogy a nagyobb méretű tár több helyet foglal el, hanem elsősorban az, hogy mekkora az a legnagyobb cím (legnagyobb bináris szám), amelyet a gép központi egysége meg tud címezni (ábrázolni tud.) Ennek a kérdésnek a megválaszolására a központi egység tárgyalásakor visszatérünk.

A számítógépek központi tárára vonatkozó kapacitásadatok igen eltérőek lehetnek az alkalmazási területtől és a számítógéphez tartozó szoftvertől függően. Nagyobb teljesítményű számítógépek központi tárkapacitása 512 Kbájt és 2 Mbájt közötti értéket mutat **1 Mbájt** / 1 megabájt / = 2^{10} Kbájt).

A félvezetős tárak megjelenése után már találkozhatunk olyan számítógépekkel, amelyeknek központi tára eléri, sőt meghaladja a 16 Mbájtot.

Az ESZR sorozat néhány számítógépének tárméretei a következők szerint alakul:

A gép típusa	A központi tár kapacitása (Kbájt)
EC—1020	64... 256
EC—1030	128... 256
EC—1040	256...1024
EC—1050	1024...2048

Ne feledkezzünk meg azonban arról, hogy ezek a tárméretek nem azt jelentik, hogy egy-egy felhasználó ekkora területet le is köthet a programja számára. A központi tárban a felhasználó programján kívül más programok is tartózkodnak és működnek, amint az a későbbiek során kiderül.

Vizsgáljuk most meg azt a kérdést, hogy mekkora a központi tár működési sebessége. Mint tudjuk, ezt a sebességet a ciklusidővel mérhetjük. A központi tár ciklusideje a különböző konkrét számítógép-típusoknál eltérő értéket mutat. Értékét alapvetően a gép belső felépítése és a megvalósítás technológiája határozza meg.

A már korábban tárgyalt destruktív táraknál a ciklusidő értéke 1—2 μs között volt (1 μs = 10^{-6} s).

A félvezetős tárak megjelenésével ez az idő csökkent. Egy minden szempontból korszerűnek számító gépnél 100 ns nagyságrendű (1 ns = 10^{-9} s). Várható ennek az értéknek a további csökkenése, miután megjelentek az első olyan számítógépek, ahol a ciklusidő 30 ns körüli értékkel rendelkezik.

A tár valamennyi címére vonatkoztatva azonos a ciklusidő. A központi tárban levő adatok tehát, függetlenül a táron belüli helyüktől, ugyanannyi idő alatt érhetők el. A ciklusidő emberi fogalmaink szerint olyan kicsi, hogy jogosan mondhatjuk: az adat, amely a központi tárban van, „azonnal" hozzáférhető.

A KÖZPONTI EGYSÉG

A központi egység feladata a gépben levő program végrehajtásának előkészítése, a végrehajtás irányítása és az előírt műveletek elvégzése. E feladatokat a központi egység akkor tudja elvégezni, ha képes:

— meghatározni egy tárterület címét;
— adott cím tartalmát lekérdezni vagy oda adatot beírni;
— adatokon aritmetikai és logikai műveleteket végrehajtani;
— az utasításokat előírt sorrendben kezelni.

Ezeket a funkciókat a központi egység két önálló része valósítja meg. A műveletvégrehajtást az aritmetikai-logikai egység, az előkészítést és az irányítást a vezérlőegység végzi el. Az amerikai-logikai egység olyan hardver elemekből épül fel, amelyek dekódolják (megfejtik) a tárból kiolvasott utasítást, s az utasításban kijelölt műveletet az ugyanott megadott tárcímtartalmakkal végrehajtják.

Az aritmetikai-logikai egység

Az aritmetikai-logikai egység által elvégezhető műveletek csoportosíthatók a műveletben részt vevő operandusok hossza, ill. a művelet fajtája szerint. Az operandus hossza szerint vizsgálva megállapítható, hogy vannak műveletek, amelyek bájt méretű

(tehát 8 bites), és vannak, amelyek félszavas, szavas, duplaszavas adatokkal dolgoznak.

A műveletek fajtái szerint a következő főbb csoportok különböztethetők meg:

— regiszterekre vonatkozó műveletek;

— regiszter és tárterület közötti műveletek;

— fixpontos matematikai műveletek (összeadás, kivonás, szorzás, osztás, összehasonlítás);

— lebegőpontos matematikai műveletek;

— logikai műveletek (logikai ÉS, logikai VAGY, léptetés);

— decimális kódban ábrázolt operandusokra vonatkozó műveletek.

Ezeket a műveleteket az aritmetikai-logikai egység áramkörök segítségével valósítja meg. Az aritmetikai-logikai egység a különböző műveleteket különböző idő alatt képes megvalósítani. A végrehajtás ideje függ:

— a művelet operandusainak helyétől (a regiszterben levő operandussal ugyanaz a művelet gyorsabb, mint a tárban levővel);

— az adatábrázolástól (fixpontos operandusokkal gyorsabb a műveletvégzés, mint lebegőpontosokkal);

— az elvégzendő művelettől (összeadáshoz kevesebb idő kell, mint szorzáshoz).

Az aritmetikai-logikai egység sebességének szemléltetésére bemutatunk néhány műveletvégrehajtási időt:

Fixpontos összeadás:

Operandusok a regiszterekben:	0,08...3,8 μs.
Egyik operandus a regiszterben, másik operandus a tárban:	0,16...4,8 μs.
Decimális ábrázolással, mindkét operandus dupla szóban:	2,1 ...29,3 μs.

Lebegőpontos összeadás:

Operandusok a regiszterekben:	0,38...13,6 μs.
Egyik operandus a regiszterben, másik operandus a tárban:	0,38...17,7 μs.

A műveletvégzés előkészítését és a művelet-végrehajtás irányítását a *vezérlőegység* végzi.

A vezérlőegység

A vezérlőegység működésének legfontosabb feltétele egy utasításnyelv, amely a számítógép számára közvetlenül érthető módon (bináris formában) adja meg az elvégzendő feladat programját. Minden számítógépnek megvan a saját „anyanyelve", amelyet *gépi kódnak* nevezünk. Minden programnak a végrehajtáskor gépi kódúnak kell lennie.

Mielőtt a vezérlőegység egy gépi kódú utasítás végrehajtásához hozzákezd, négy információra van szüksége. Ezek:

— az elvégzendő művelet megnevezése (kódja),

— az első operandus címe,

— a második operandus címe,

— a megszülető eredmény leendő címe.

Ezenkívül még egy ötödik információ is szükséges, legkésőbb az utasítás végrehajtásakor:

— a soron következő utasítás tárbeli címe. ▲ 256 9

Ha minden utasítás ennyi információt tartalmazna, igen sok helyet foglalna el a program a központi tárban. Ezért az információk számát különböző módon csökkentik. Így pl. az eredmény általában valamelyik operandus címén keletkezik, a következő utasítás címének kijelölésére pedig egy külön regiszter szolgál.

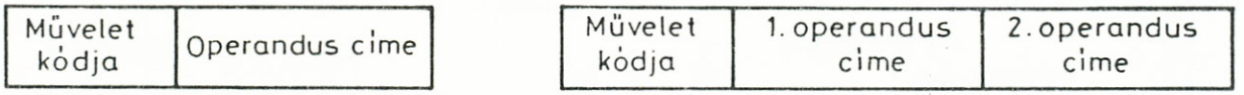
Egy utasításnak így alapvetően kétféle információt kell hordoznia:

— meg kell határoznia, hogy mi az elvégzendő művelet;

— meg kell adnia, hogy hol található(k) a művelet operandusa(i).

Az elsőt az utasítás *műveleti része*, a másodikat az utasítás *címrésze* határozza meg. Ez utóbbi egy vagy két címet tartalmazhat.

A 41. ábrán egy- és kétcímű utasításformát mutatunk.



a) b)

41. ábra. Egycímű (a) és kétcímű (b) utasításforma

A gépi kódú utasításban levő címek számától függően egycímű és kétcímű gépekről szokás beszélni. A gépi kódú utasítások, attól függően, hogy a számítógép mely részét kívánják működésbe hozni, négyfélék lehetnek:

— vezérlőutasítások, amelyek a vezérlőegységet működtetik;

— aritmetikai utasítások, amelyek az aritmetikai-logikai egységet működtetik;

— átviteli utasítások, amelyek a központi tárat működtetik;

— input-output utasítások, amelyek a perifériákat működtetik.

Az utasítások alakja számítógéptípusonként eltérő, hasonlóan a többi hardver adottsághoz. A bájt szervezésű központi tárral rendelkező számítógépeknél az utasításoknak öt fajtáját különböztethetjük meg, attól függően, hogy a címrészben milyen információk találhatók. Az ötféle utasítást a 42. ábra szemlélteti. Valamennyi utasításforma első bájtja a műveleti kódot tartalmazza. A műveleti kód első két bitje az utasítás hosszát adja meg:

00 egy félszó hosszú utasítás (RR típusú utasítás);

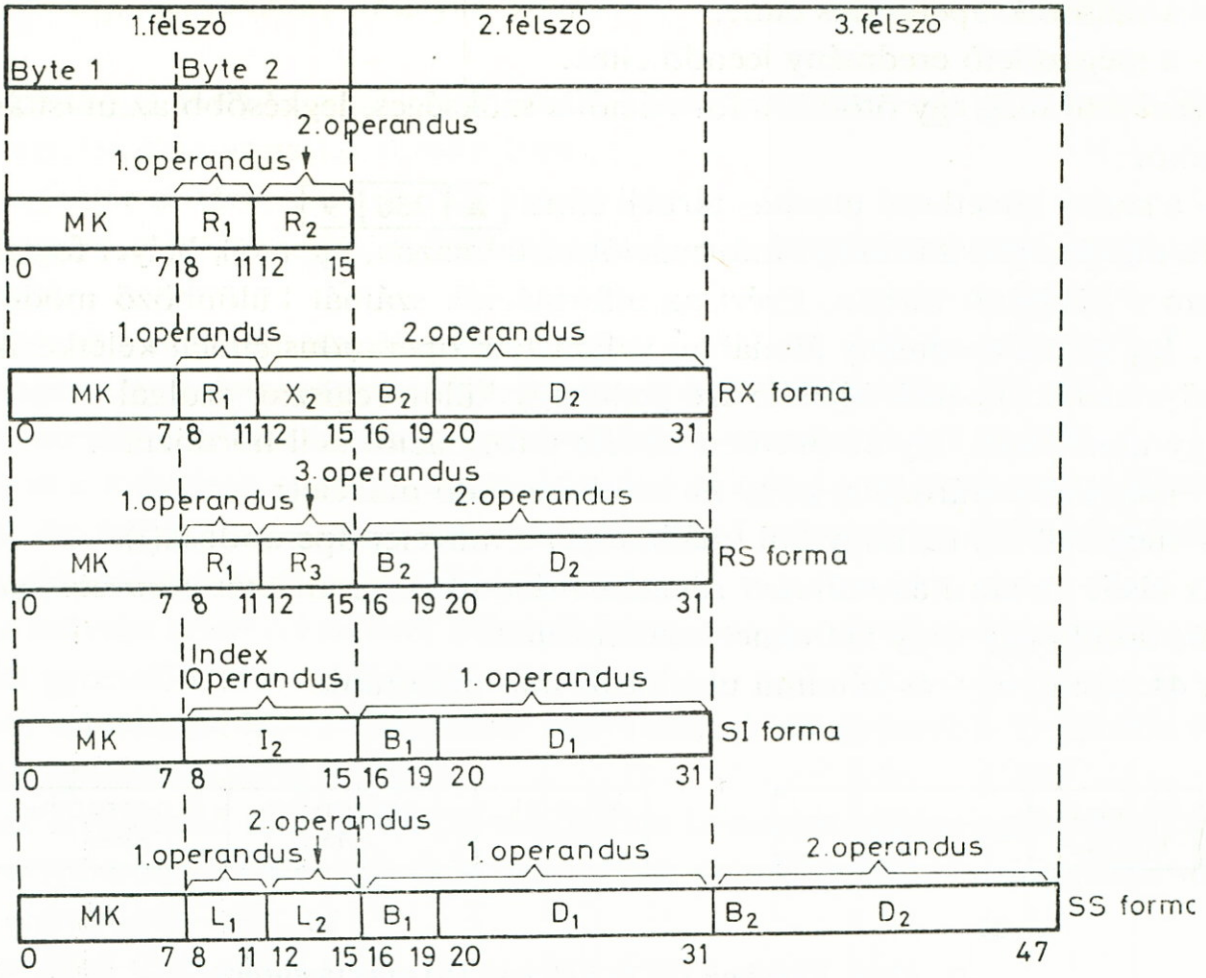
01 két félszó hosszú utasítás (RX típusú utasítás);

10 két félszó hosszú utasítás (RS vagy SI típusú utasítás);

11 három félszó hosszú utasítás (SS típusú utasítás).

Az előbbiekből következik, hogy a műveleti kódot alkotó két hexadecimális érték közül az első lehetséges értékei a következőképpen alakulhatnak:

RR típusú utasításnál 0, 1, 2, 3;



42. ábra. Utasításformák

RX típusú utasításnál 4, 5, 6, 7;

RS és SI típusú utasításnál 8, 9;

SS típusú utasításnál D, F. ▲ 257 10

Mit jelentenek az utasítástípusként megadott RR, RX, RS, SI, SS betűkombinációk?

Vizsgáljuk meg ezt a kérdést közelebbről. A bájt szervezésű számítógépek öt utasítástípusa a következők:

Az **RR típusú utasítás** két általános célú regiszter tartalmával végez műveletet. A regiszterek címét a második bájt tartalmazza. Mivel 16 általános célú regiszter áll a programozó rendelkezésére, ezek bármelyike négy biten megcímezhető. Az eredmény az első operandus helyére kerül.

RR típusú utasítások között találunk fixpontos és lebegőpontos aritmetikai, logikai és adatmozgató utasításokat.

Az **RX típusú utasításban** az egyik operandus az R_1-gyel jelzett regiszterben található, a másik a központi tárban. A tárcím indexelhetően érhető el, az X_2, B_2, D_2 mezőkben levő értékek segítségével. A tárbeli cím kiszámítási algoritmusát az utasítástípusok bemutatása után tárgyaljuk. A művelet eredménye a regiszterben képződik.

Az X_2, B_2, D_2 tartalmával megadott tárbeli cím az adat kezdőbájtjának címe. Az adat ettől a címtől kezdődően található meg a tárban. Az adat hossza (kettő, négy

vagy nyolc bájt) a kijelölt műveletből következik. Pl. a lebegőpontos műveletek négy vagy nyolc, a fixpontosak kettő vagy négy bájtosak.

Az **RS típusú utasítások** három operandussal dolgoznak. (Vannak utasítások, amikor az R_3 mező üres.) A műveletben szereplő első és harmadik operandus az R_1 és az R_3 regiszterben van. A második operandus a B_2, D_2 értékekkel megadott tárcímen található.

Ilyen utasítás pl. a 98 műveleti kódú „Load Multiply" utasítás, amely a B_2 D_2 által megadott címtől kezdődő szavakat tölt át egymás mellett elhelyezkedő regiszterekbe. Az első és az utolsó regiszter címét R_1 és R_3 adja meg.

Az **SI típusú utasításban** az I2 jelzésű bájt tartalma egy konstans érték, amelyet a programozó ad meg. Ennek a konstansnak az értéke a művelet végzése során nem változik meg. Ezt a címzési módot literális módnak nevezik.

Ilyen utasítás pl. a „Move Immediate", 92-es kódú utasítás, amelynek hatására a B1 D1 által meghatározott tárcímbe beíródik az I2-ben levő egy bájt méretű konstans érték.

Az **SS típusú utasítások** mindkét operandusa az operatív tárban található. A decimális aritmetikájú műveletek — tehát a zónázott operandusokkal végzett műveletek — SS típusúak.

Az első operandus kezdőcíme a B1 D1 által megcímzett bájt. Az operandus hosszát az L1 mező adja (az operandus hossza = L1 tartalma + 1 bájt, így az operandus maximális hossza 16 bájt lehet). A második operandus kezdőcíme a B2 D2 által megcímzett bájt. A második operandus hosszát az L2 mező adja (operandus hossza = L2 tartalma + 1 bájt). Ha a két operandus hossza megegyezik, ez a közös hossz 256 bájt lehet.

A tárcím generálása, abszolút és relatív címzés

Tanulmányozva a különböző típusú utasításokat, észrevehetjük, hogy minden olyan esetben, amikor nem regisztereket, hanem tárcímet kellett megadni, nem a tár egy konkrét, meghatározott címét adtuk meg. A legegyszerűbb nyilván az lenne, ha az utasítás címrészében annak a tárterületnek a kezdőcíme szerepelne, ahol a szükséges operandus elhelyezkedik. Pl. 32728 jelentené a tár 32728. rekeszével (bájtjával) kezdődő szót. Az ilyen címet nevezzük **abszolút címnek.** Az abszolút címzés azonban nem szerencsés és nem is realizálható. Miért? Nyilvánvaló, hogy minél nagyobb egy tár mérete, annál nagyobb sorszámmal adható meg a legnagyobb című rekesz abszolút címe. Ez azt jelenti, hogy a gép tárméretének növekedésével az utasítás címrészeiben levő bitek számának is növekednie kell. (Nagyobb szám leírásához több hely kell.) 256 Kbájtos tár esetén 18 bitre van szükség, 1 Mbájt esetén már 20 bitre. | ● | 261 | 11 | Az utasítás hosszának a tármérettől való függése azonban nem kívánatos. | ▲ | 257 | 12 | Más oldalról, ha a programban minden operandus tárbeli helye abszolút címben lenne megadva, a programot mindig csak egyazon tárterületre lehetne

betölteni futtatás céljából. Márpedig a modern számítógépeknél a program tárbeli helye nemcsak két végrehajtás során nem azonos, hanem egyetlen futás során is változhat, amint azt a későbbiekben látni fogjuk.

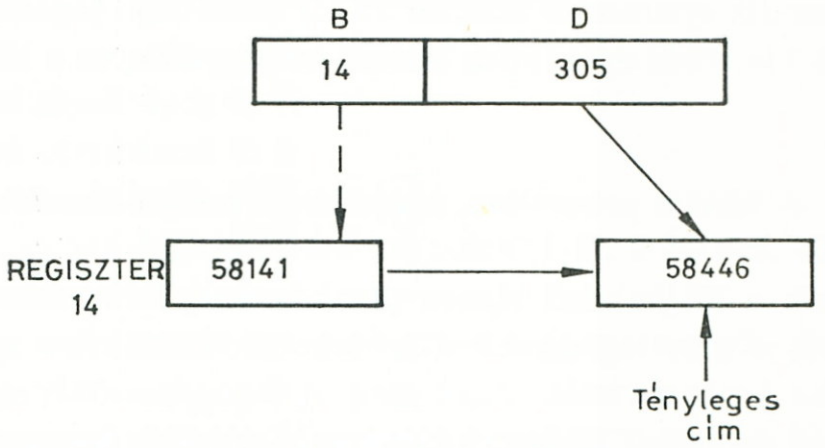
Az abszolút címzés helyett a szükséges tárterületekre való hivatkozást — vagyis a tényleges cím meghatározását — **relatív címzéssel** kell megvalósítani.

A tényleges cím kiszámításához RX típusú utasításoknál három, RS és SS típusú utasításoknál két mező tartalma nyújt segítséget. A következőkben az RX típusú utasítás esetén való címképzést mutatjuk be.

A tárbeli operandus címének előállítási mechanizmusa a következők szerint foglalható össze:

A B jelű, 4 bit hosszúságú mezőben a 16 általános célú regiszter egyikének a sorszáma van. (Ez 0 és 15 közötti bináris szám.) Ebben az ún. bázisregiszterben korábban elhelyezésre került a központi tár egy címe *(báziscím)*.

A D jelű, 12 bit hosszúságú területen egy *eltolási érték* található. Ez a bináris számérték azt mondja meg, hogy a megcímezni kívánt rekesz a bázisregiszterben levő címhez képest milyen messze van. Ha a megcímezni kívánt rekesz egy tömb valamely eleme, az X jelű 4 bites mezőben megnevezett regiszter is szerephez jut. A B és D együttesen ui. csak a tömb kezdőcímét adja meg, a keresett tömbelemnek a tömb kezdőcímétől való távolságát az X regiszter tartalmazza. |⊙|263|13| Az elmondottakat szemlélteti a 43. ábra.



43. ábra. A tényleges cím kialakítása

Ezzel a technikával a központi tár felső (nagyobb sorszámú) rekeszei ugyanúgy címezhetők, mint a kis sorszámúak, hiszen minden azon múlik, hogy mekkora érték kerül a B bázisregiszterbe. Egy bázisregiszterrel 4 Kbájtnyi terület címezhető végig (a D 12 bites). Az egy program által igénybevehető területet tehát az befolyásolja, hogy

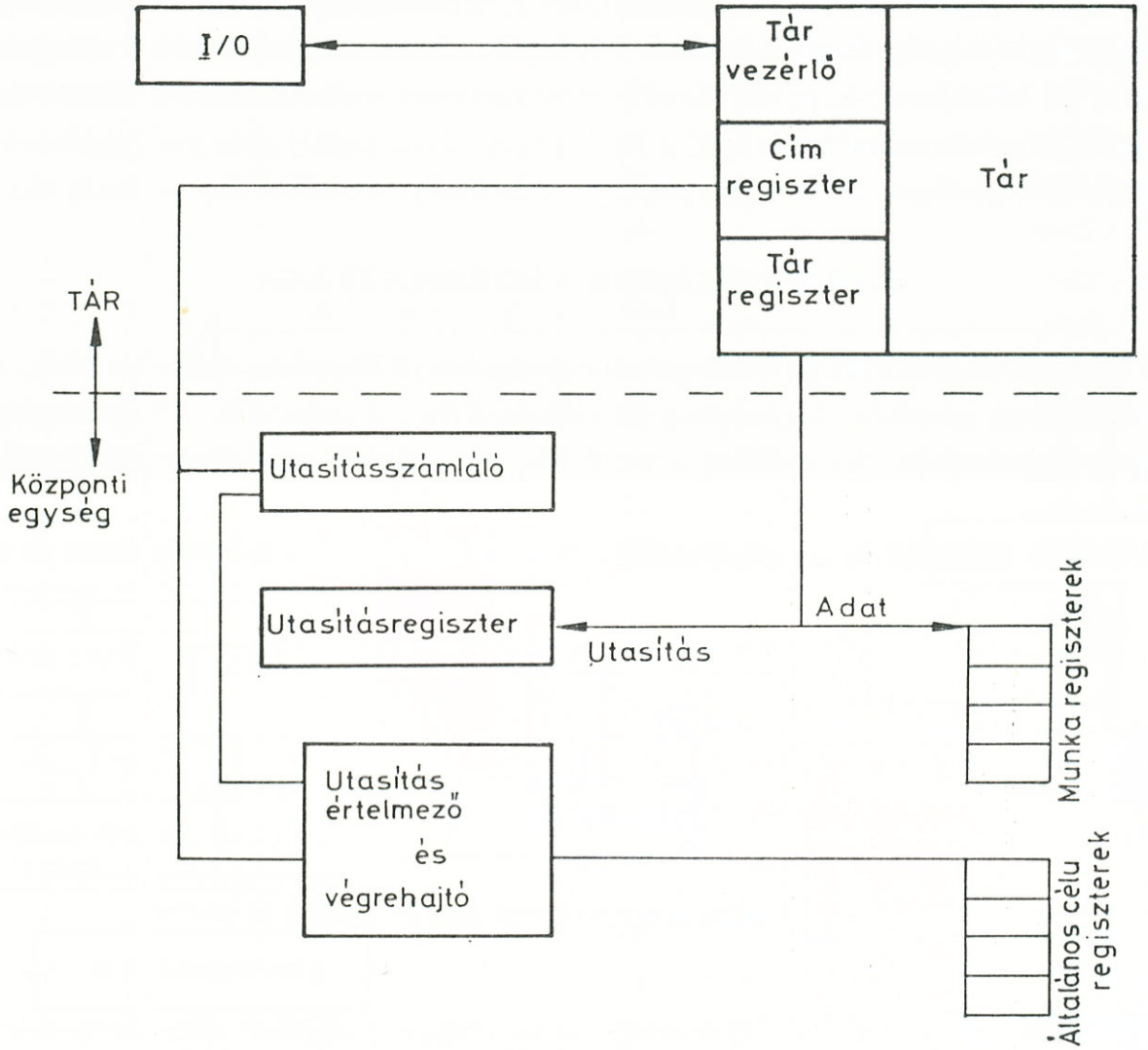
— hány bázisregisztert használhat a program,

— hogyan tudja ugyanannak a bázisregiszternek a tartalmát menetközben változtatni.

A programnak a táron belüli áthelyezése pedig mindössze a bázisregiszter tartal-

mának megváltoztatásával lehetséges, hiszen ilyenkor a tényleges címek „máshová" számítódnak ki.

A félvezetős tárak megjelenésével a probléma nem az, hogy mekkora központi tárat lehet egy számítógéphez illeszteni, hanem az, hogy mekkora területet lehet a központi tárból egy programon belül címezni. A központi egység és a központi tár logikai felépítését a 44. ábra szemlélteti.



44. ábra. A központi egység és a központi tár logikai felépítése

VIRTUÁLIS TÁRAK

A félvezetős tárak megjelenése előtt, amikor egy-egy számítógép központi tárméretében az 1 Mbájt jelentette a felső határt, felmerült az igény, hogy nagyobb helyigénnyel fellépő feladatok is futtathatók legyenek. E probléma megoldásaként jött létre az a technikai megoldás, amelyet virtuális tárnak nevezünk. (A „virtuális" szó látszólagost jelent.) Ennek lényege, hogy a felhasználó programja a végrehajtás ideje alatt nincs állandóan teljes terjedelmével a központi tárban, hanem egy része van a tárban, más része (amelyre éppen nincs szükség) háttértárolón (mágneslemezen) he-

lyezkedik el. A felhasználó persze ezt nem veheti észre, vagyis az éppen aktív programrésznek mindig bent kell lennie a valóságos tárban. Ennek az elvnek a megvalósítása részben hardver, részben szoftver változtatásokat, ill. fejlesztéseket igényelt a tárkezelés megvalósításában.

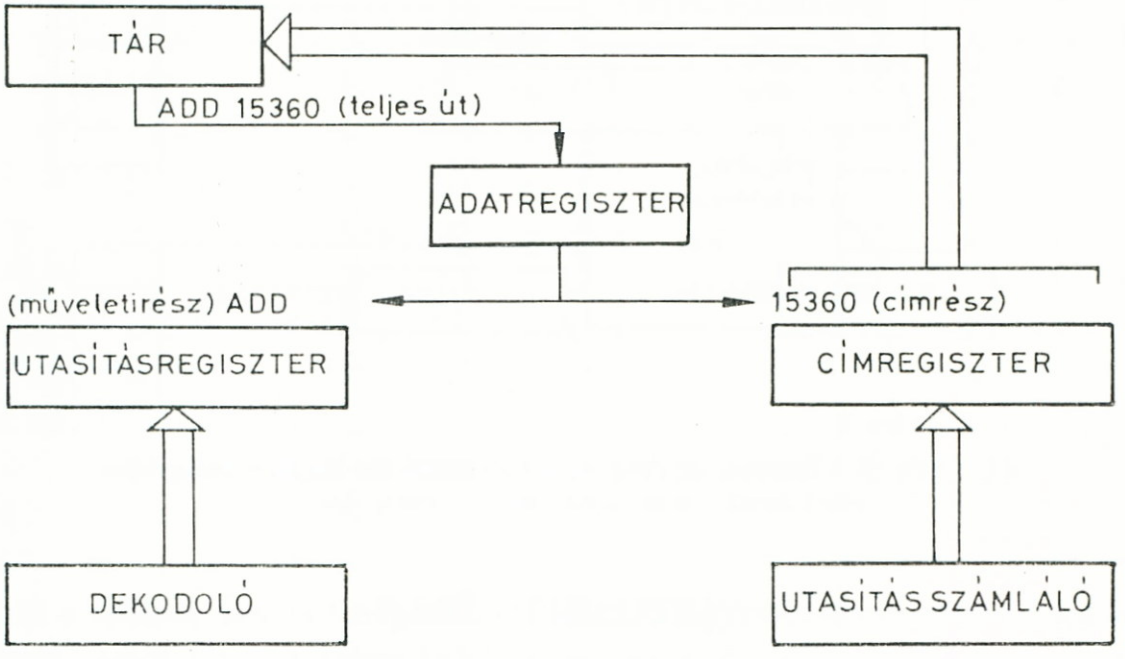
A legfontosabb feltételekre és azok megvalósításának elvére az operációs rendszereknél (IX. fejezet) még visszatérünk, mert jelenlegi ismeretanyagunk a megértéshez nem elégséges.

A virtuális társzervezéssel a számítógép a felhasználó számára a valóságos tárméret többszörösét tudja biztosítani. A félvezetős tárak megjelenésével a virtuális tárkezelés már 64 Mbájtos központi tárat (természetesen virtuális tárat) biztosít minden felhasználói program számára.

AZ UTASÍTÁSOK VÉGREHAJTÁSA

Egy utasítás végrehajtását a központi egység *két fő fázisban* valósítja meg. Az első, az *utasításfázis* az utasítás kiolvasása és előkészítése; a második, a *végrehajtási fázis* a tényleges végrehajtás. Az előbbit a vezérlő-, az utóbbit az aritmetikai-logikai egység végzi.

A 45. ábra szemlélteti az utasításfázisban elvégzett tevékenységeket. A vezérlő:



45. ábra. Az utasításfázis

— az utasítást a tárbeli helyéről az adatregiszterbe hozza;

— az utasítás műveleti kódját leválasztja és az utasítás regiszterbe viszi, amelyben dekódolás után vezérlőjelekké alakul át;

— az operandus címét a címregiszterbe helyezi. Ez fogja megmutatni, hogy a műveletben szereplő operandust hol keresse;

— a soron következő utasítás helyének (címének) meghatározását elvégzi.

A második — végrehajtási — fázisban történik a kijelölt művelet tényleges végrehajtása az aritmetikai-logikai egységben:

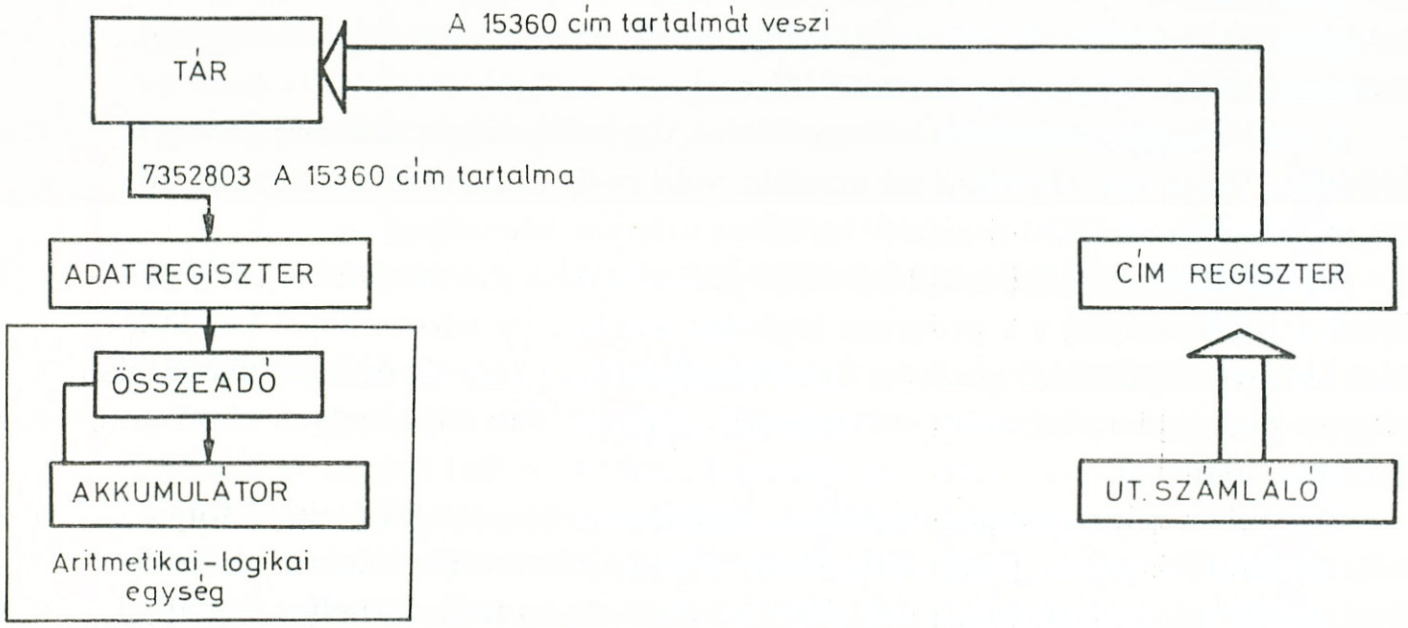
— a címregiszterben levő tárcím alapján a kért operandus kiválasztása, s az adatregiszterbe vitele;

— a vezérlőimpulzusok hatására a kijelölt művelet eredményének képzése.

A végrehajtási fázist a 46. ábra mutatja. (Az ábrán szemléltetett példában az elvégzendő művelet egy összeadás, egyik operandus a 15360 címen található, a másik egy „akkumulátor" nevű regiszterben. Az eredmény is itt képződik.)

A tárgyalt kétcímes utasításrendszernél a műveleti eredmények mindig az első operandus címén keletkeznek.

A vezérlőegység működése során bonyolult irányítási és összehangolási feladatot lát el. Ezek pontos, jól időzített elvégzését mikroprogramok segítségével oldja meg.



46. ábra. A végrehajtás fázisa

A *mikroprogram* tulajdonképpen „hardveresített" szoftver. Vagyis eredetileg egy program (tehát szoftver), amelyet azonban áramkörökkel (tehát hardverrel) valósítottak meg egy chipen (lapkán). A mikroprogramokat egy külön tárban, az ún. fixtárban helyezik el. Az elnevezésben a fix szó arra utal, hogy ebből a tárból csak kiolvasni lehet, oda információt bevinni nem. A fixtárakban a programokat gyárilag „égetik be", ami azt jelenti, hogy a mikroprogram egy adott technológiai folyamat eredményeképpen áll elő. ⊙ | 264 | 14 Az integráltsági fok növekedésével a modern számítógépekbe egyre nagyobb kapacitású fixtárakat építenek be. A fixtárban levő mikroprogram végrehajtása ui. hardver sebességű (vagyis a villamos áram sebességének a nagyságrendjébe esik), így a külső szemlélő számára olyan, mintha ténylegesen a hardver hajtaná végre a műveleteket.

A növekvő fixtárak újabb és újabb mikroprogramjai egyre bonyolultabb vezérlési feladatok gyors megvalósítását biztosítják.

Az utasítások sorrendjének meghatározása

Az eddigiekben nyomon követtük, hogy a központi egység hogyan hajtja végre az éppen soron következő utasítást. Nem válaszoltuk meg azonban azt a kérdést, hogy az utasítások milyen sorrendben követik egymást a végrehajtás folyamatában, s honnan tudja a vezérlőegység, hogy mikor, melyik utasítás a „soron következő". Az utasításoknak alapvetően kétféle sorrendje létezik minden programban. Az egyik a rögzítés *fizikai* sorrendje, ahogy a programozó a programutasításokat egymás után írta, ill. ahogy az utasítások egymást követik a listán. A másik a végrehajtás során kialakuló *logikai* sorrend, amely a programban levő feltételes és feltétel nélküli vezérlésátadások miatt nem azonos a fizikai egymásutánisággal. ▲ 258 15 Ez a kétféle sorrend természetesen független a program nyelvétől, gépi kódú állapotban ugyanúgy létezik, mint a program más nyelvű változatában.

A kétcímű utasításrendszer nem teszi lehetővé, hogy minden utasítás magában hordozza a logikailag őt követő utasítás címét. Erre a célra egy külön regiszter, az **utasításszámláló** vagy programszámláló **regiszter** szolgál.

A számítógép az utasításokat egyenként, egyiket a másik után hajtja végre. Annak eldöntése, hogy egy végrehajtott utasítás után melyik utasítás tekintendő következőnek, az utasításszámláló regiszter tartalma alapján lehetséges.

Az utasítások a központi tárban helyezkednek el. Az utasítások csak páros című bájton kezdődhetnek, s a program logikája szerint egy adott címtől kezdődően egymást követő — növekvő című — tárterületeket foglalnak el. Az utasításszámláló tartalma a program indulásakor megegyezik azzal a címmel, amelyen a program első utasítása található.

Miközben ennek az utasításnak a végrehajtásra való előkészítése folyik, az utasításszámláló tartalma is változik. Ha a végrehajtásra előkészítés alatt levő utasítás félszónyi tárhelyet foglalt el, a számláló kettővel nő, ha több tárhelyet foglalt el az utasítás (pl. hat bájtnyit), a számláló értéke hattal nő. Emlékezzünk vissza, hogy az utasítás műveleti kódjának első két bitje tartalmazza az utasítás hosszát, amely egy, két vagy három félszó lehet. Így minden esetben könnyen kiszámítható, hogy a következő utasítás mely címen kezdődik. Az utasításszámláló mindenkori értéke azt a címet mutatja, amelyen a soron következő, végrehajtandó utasítás található.

A programban levő vezérlésátadási funkciók megvalósításához szükség van arra, hogy ezt a szigorú sorrendiséget megváltoztatva, a vezérlést a program egy más részére adjuk át. Ezt a feladatot a vezérlésátadó (ugró) utasítások valósítják meg. Ezeknél az utasításoknál az egyik operandus maga az utasításszámláló regiszter. Feltétel nélküli vezérlésátadás esetén utasításszámláló regiszterbe íródik a soron következő utasítás kezdő címe. Ha feltételes vezérlésátadásról van szó, az utasításszámlálóban csak akkor áll elő az ugrási cím, ha az előírt feltétel teljesült. Ha ez a feltétel nem áll fenn, a fizikailag következő utasítás kerül végrehajtásra. ▲ 258 16

A különböző feltételek vizsgálatára és az ugrások megvalósítására külön gépi kódú utasítások léteznek.

Az utasításszámláló vázolt működése valósítja meg a számítógép emberi beavatkozás nélküli működését.

ADATMOZGATÁS A SZÁMÍTÓGÉPEN BELÜL

A központi egység, működése során, adatokat mozgat, hogy velük a programban előírt műveleteket végrehajtsa. Az adatok a központi tárban vannak, a művelet-végrehajtás az aritmetikai-logikai egységben zajlik, az eredmények ismét a központi tárba kerülnek. Hogyan történik az adatok mozgatása?

Azok az adatok, amelyek a központi tárból átkerülnek a központi egységbe, regiszterekben kerülnek tárolásra. A regiszterek közötti adatáramlás az **adatsínen** valósul meg. Az adatsín az egész gépet átfogó adatátvivő hálózat, amely megfelelő erősítőkkel van ellátva. Minden regiszter az adatsínhez kapcsolódik. A vezérlőegység szabályozza, hogy mikor melyik regiszter tartalma kerülhet a sínre. A sín „szélessége" az egyidejűleg mozgatható bitek száma. A leggyorsabb — és legköltségesebb —, amikor a teljes regisztertartalmat egyidejűleg képes a sín átvinni az előírt helyre.

ADATMOZGATÁS A SZÁMÍTÓGÉP ÉS A PERIFÉRIÁK KÖZÖTT

A központi tár és a központi egység működési sebessége másodpercenként több százezer művelet végrehajtását biztosítja. Ilyen sebesség mellett a gépbe óránként több száz Mbájtnyi adatot kell bejuttatni. A perifériák azonban — mivel mechanikus mozgást igényelnek — képtelenek kiszolgálni ezt az „adatéhséget".

A probléma szemléltetésére tekintsük a következő példát. Kártyaolvasóról kívánunk adatokat beolvasni a központi tárba. A műveletnek három résztvevője van: a lyukkártyaleolvasó berendezés, amely a kártyára külső kódban rögzített adatokat érzékeli és átalakítja; a központi tár, ahová az adatok kerülnek és a vezérlőegység, amely az adatátvitel vezérlését látja el. Tegyük fel, hogy viszonylag gyors, 1200 kártya/min sebességű kártyaolvasónk van, és a központi tár ciklusideje 1,25 μs. A kártyaolvasó az adott sebességgel egy kártyát 50 ms, azaz 50 000 μs alatt képes beolvasni. Válasszuk meg a rekordképet úgy, hogy a lehető legtöbb önálló adat forduljon elő. Ez nyilván akkor valósul meg, ha a lyukkártya minden oszlopát külön adatnak tekintjük, vagyis ha 80 adatot feltételezünk egyetlen lyukkártyán. (A valóságban ilyen eset ritkán akad, de ez a rekordkép okozza a legtöbb munkát a központi egységnek.) A vezérlő minden olyan adatot, amely külön címre íródik, egy-egy külön ciklussal ír be a tárba. A 80 adat beírásához tehát $80 \times 1{,}25$ μs, vagyis 100 μs szükséges.

Egy kártya beolvasásakor tehát — amely 50 000 μs ideig tart — a központi egység mindössze 100 μs ideig dolgozik.

Az első generációs számítógépeknél a központi egység várakozása valóság volt:

az input-output műveletek végrehajtásának ideje alatt a gépben semmi nem történt, a gép várt a művelet-végrehajtás befejezésére.

A számítógépek második generációjának megjelenésével megjelent az a hardver megoldás is, amely a hatékonyság növelése érdekében megoldja a sebesség eltérésből adódó problémát. A perifériák a központi egységhez **csatornákon** keresztül kapcsolódnak.

A csatornák a központi egységgel való kétirányú adatátvitel önálló lebonyolítására alkalmas berendezések. Az a tény, hogy a csatorna önállóan képes az adatátvitel lebonyolítására, vagyis mivel saját perifériavezérlője van, tehermentesíti a központi egység vezérlőjét az adatátvitellel kapcsolatos vezérlési feladatok ellátása alól.

A vezérlőegység feladatát csak a csatornák aktivizálása jelenti, a csatornák — mint speciális célú önálló egységek — lebonyolítják a perifériákra vonatkozó adatátvitelt — majd ennek befejezését visszajelzik a központi egységnek.

A vezérlőegység külön csatornaprogramokat használ a csatornák kezeléséhez.

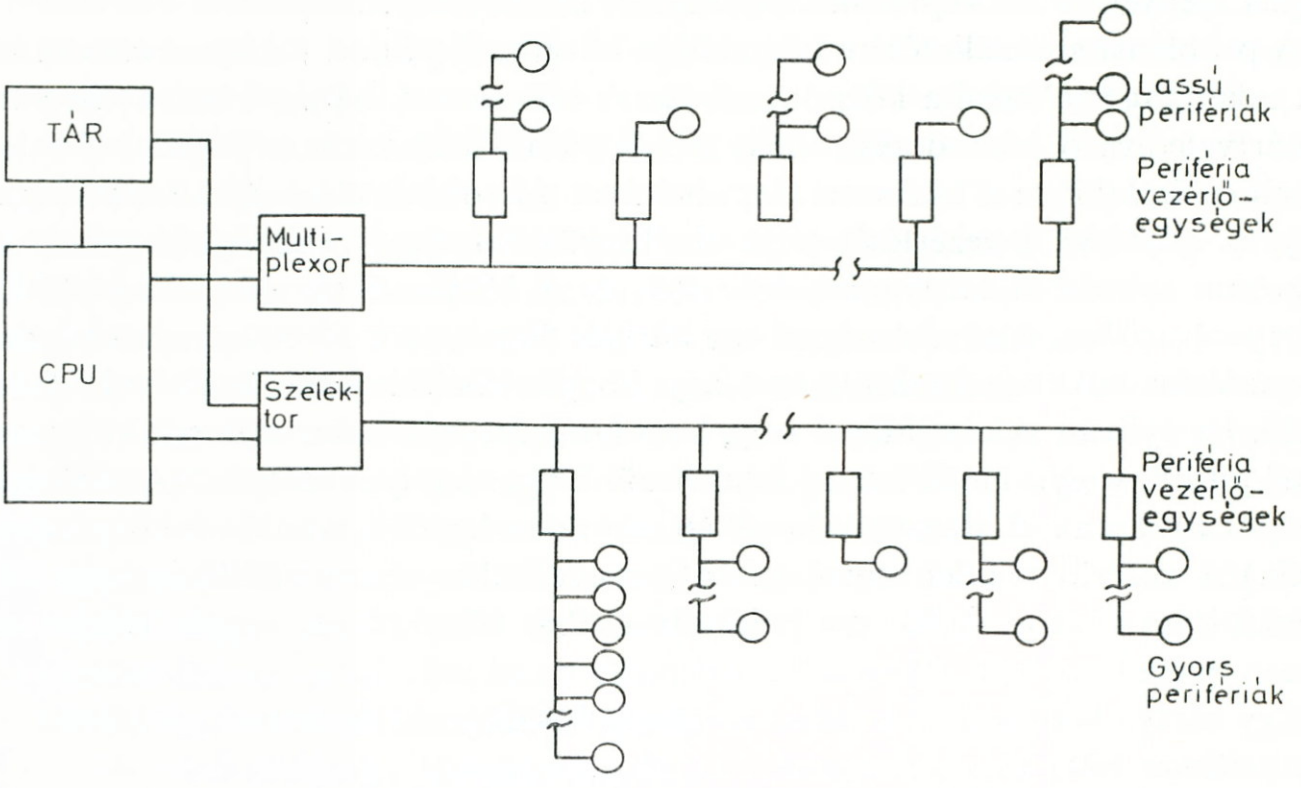
A csatornaelv megvalósítása más problémák megoldását is jelentette. Ezek a problémák a következők:

— a periféria, ill. a központi tár által egyidejűleg kezelhető adatmennyiség különböző;

— a perifériáknak — struktúrájuk és feladatuk eltérő volta miatt — különböző utasításokat kellene küldeni a vezérlőből, ha a teljes vezérlés onnan történne;

— egyidejűleg több periféria nem tudja közvetlenül igénybe venni a tárat;

— nincsen biztosítva, hogy a tárat vagy a perifériát éppen akkor kívánjuk igénybevenni, amikor azok szabadok.



47. ábra. A központi egység és a perifériák kapcsolódása

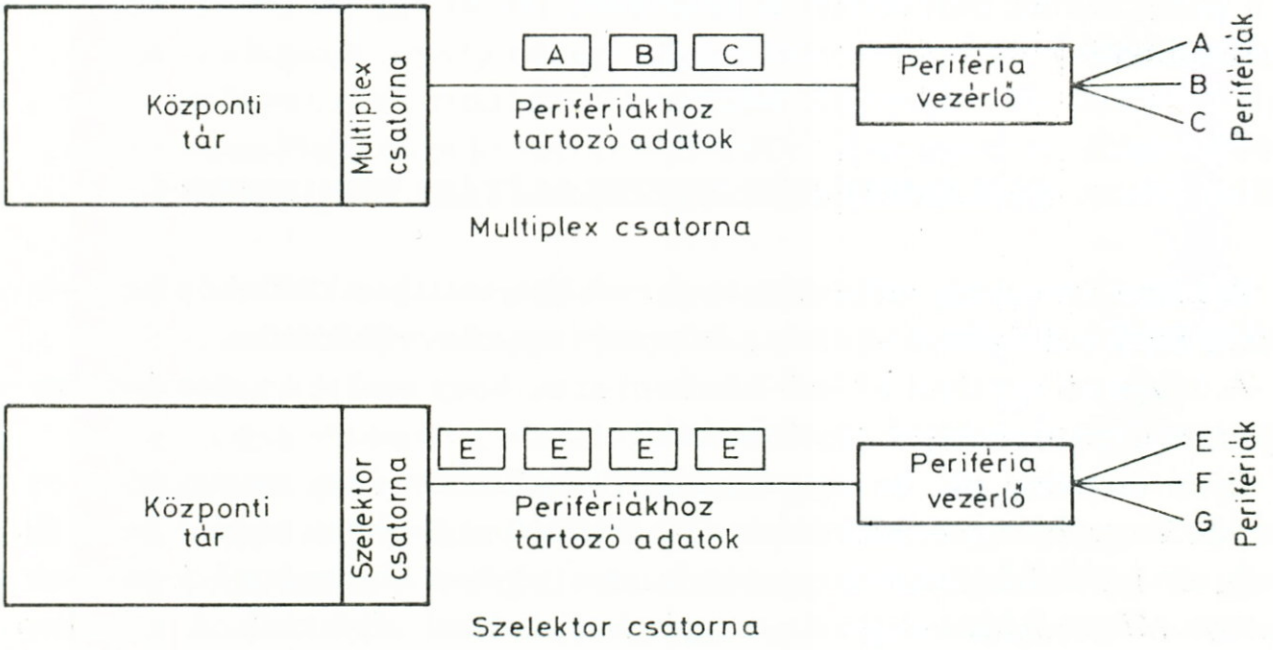
A 47. ábra a központi egység és a perifériák közötti adatátvitel hardver megoldását szemlélteti.

A perifériák a perifériavezérlő berendezéseken keresztül kapcsolódnak a csatornához. Egy-egy csatornához általában 8 vezérlőegység kapcsolódhat, és minden vezérlőegységhez 8-8 periféria. Egy csatorna tehát 64 perifériával tud kapcsolatot teremteni, de egyidejűleg mindig csak eggyel.

Kétféle csatornát szokás megkülönböztetni: a *szelektor csatornát* és a *multiplex csatornát*. A kétféle csatorna működési módjában tér el egymástól.

A gyors perifériákat, vagyis a háttértárakat szelektor csatornán keresztül csatlakoztatják. A szelektor csatorna ui. egyidejűleg nagyobb mennyiségű adatot — több Kbájtnyit — képes átvinni egy perifériából, ill. egy perifériára, s ez a tulajdonsága megfelel a mágneslemez és a mágnesszalag átviteli sebességének. Az alatt az idő alatt, amíg ez az adatmennyiség a csatornán áthalad, a csatornát az adott periféria teljesen le is foglalja. Ezzel szemben a multiplex csatorna egyszerre több input-output egységet (lassú perifériát) tud kiszolgálni. Ez úgy lehetséges, hogy a csatorna nem várja meg egy-egy lassú perifériánál az adatátvitel befejezését — hiszen ez a mechanikus mozgások miatt sok időt vesz igénybe —, hanem a különböző lassú perifériákat rövid időközönként sorban kezelve mindenhonnan néhány bájtnyi információt visz át. Így a saját sebességével szimultán módon több lassú perifériát képes kezelni. Egy közepes, nagy teljesítményű számítógéphez két (esetleg három) szelektor csatornát és egy multiplex csatornát csatlakoztatnak.

A szelektor és a multiplex csatorna működési elvét szemlélteti a 48. ábra.



48. ábra. A multiplex és a szelektor csatorna működési elve

ELLENŐRZÉSI FUNKCIÓK A SZÁMÍTÓGÉP MŰKÖDÉSE SORÁN

A központi tár és a központi egység működése során előállhatnak különböző hibaokok. Ezek egy része abból adódhat, hogy a számítógépen belüli adatforgalom során bitek „elvesznek", más részük abból, hogy a vezérlés során valami nem várt esemény lép fel. E hibalehetőségek kiküszöbölésére különböző védelmi elemek találhatók a számítógépben.

Az adatforgalom hibamentes biztosítását szolgálja az ún. *paritásellenőrzés.*

Azt tanultuk, hogy a bájt nyolc bitből áll. Ez a felhasználó szempontjából így is van, 1 bájt nyolc bitnyi információt képes tárolni. A központi tárban azonban minden bájthoz járul még egy kilencedik bit is. Ezt a bitet paritásbitnek nevezzük. A paritásbit értékét mindig a rendszer számítja ki oly módon, hogy a nyolc adatbitben levő 1 értékű bitek számát páratlan számúra egészíti ki. Ha tehát a bájtban levő 1 értékű bitek száma páratlan, akkor a paritásbit értéke 0 lesz, mint pl. a következő bájttartalom esetén: 11000010 (a B betű kódja). Ha a bájtban levő 1 értékű bitek száma páros, a paritásbit értéke 1 lesz. Pl. az 11000011 (a C betű kódja) esetében. Ily módon a kilenc biten levő 1 értékű bitek száma mindig páratlan.

Adatátvitel esetén az átvitt nyolc bit tartalmából egy új paritásbit képződik. Ezt a rendszer összeveti az eredeti paritásbittel. Ha a kettő megegyezik, az átvitel helyes volt, ha nem egyezik meg, az átvitel során bittévesztés lépett fel. Ilyenkor újra megtörténik az adatátvitel. (Persze a dolog mégsem ilyen egyszerű, mert ha pl. páros számú bit váltott értéket, a paritásbit egyezőség továbbra is fennáll. Anélkül, hogy ebbe a problémakörbe mélyebben belemennénk, megemlítjük, hogy megfelelő megoldásokkal a páros számú bittévesztés is kivédhető, sőt az egy bit tévesztésének a helye is megállapítható.)

A SZÁMÍTÓGÉP MEGSZAKÍTÁSI RENDSZERE

A központi egység működése során a környezetében különböző események történnek. Ezek befolyással vannak a központi egység működésére.

A központi egységet fel kell készíteni arra, hogy ezekre az eseményekre reagálni tudjon. Milyen események történhetnek?

Ilyen esemény pl., ha egy decimális utasításban nem megengedett operandus szerepel. Vagy, ha egy osztásnál az osztó értéke 0 vagy ahhoz nagyon közel álló mennyiség, és így a hányados nagyobb, mint a gépben ábrázolható legnagyobb szám. Ilyenkor *túlcsordulás* keletkezik, vagyis értékes bitek elvesznek. A műveletsort meg kell szakítani, hiszen a túlcsordult értékkel nem szabad tovább dolgozni. Más jellegű problémát vet fel, ha olyan input adattal kellene műveletet végezni, amely még nincs benn a tárban, tudniillik az olvasási művelet nem fejeződött be. Az ilyen esemény ellen a program működésének *megszakításával* lehet védekezni. A megszakítás addig tartson, amíg a kért adatátviteli utasítást a csatorna nem hajtotta végre. Nem túl

gyakran, de előfordulhat, hogy a központi egység működését valamilyen váratlanul fellépő géphiba (hardver hiba) akadályozza. Ilyenkor sincs más lehetőség, mint a futó program megszakítása. ◎ | 264 | 17

Az ilyen típusú eseményekre a közös megoldás: a *futó program működésének felfüggesztése* vagy más szóval *megszakítása.* Mit jelent egy program megszakítása?

Egy program megszakítása a program végrehajtásának időleges abbahagyása, felfüggesztése oly módon, hogy a későbbiekben a program a megszakítási helytől „sértetlenül" tovább folytatható legyen. A „sértetlenül" azt jelenti, hogy a program folytatása úgy valósul meg, mintha a felfüggesztés létre sem jött volna. A megszakításokra azért van szükség, hogy a központi egység reagálni tudjon a megszakítást kiváltó eseményre. (A korábbi példákban: kivédje a nulla osztásból adódó túlcsordulást, megoldja az input-output lassúságának kérdését stb.)

A megszakításkezelést részben hardver elemek, részben programok végzik. Itt most csak a hardver körébe tartozó problémákat tárgyaljuk. Ebből a szempontból legfontosabb kérdés, hogy miképpen lehet a megszakított program későbbi továbbfutását biztosítani. Nyilvánvalóan úgy, hogy minden olyan információt, amely az adott programnak a megszakítás pillanatában levő állapotára utal, meg kell jegyezni (tárolni kell), mert ezek alapján állítható vissza a megszakításkor észlelt állapot. Miután a program állapotára vonatkozó információk általában regiszterekben vannak (gondoljunk csak az utasításszámláló regiszter feladatára), általában regiszter-tartalmak tárolása a feladat. Sok számítógépnél egy külön tárterület az ún. *program-állapot szó* (PSW = program status word) 64 bitje tartalmazza mindazon információt, amely a program folytatásához szükséges. Így a feladat a PSW tartalmának kimentése és tárolása.

Amikor tehát a központi egységhez egy megszakítást kiváltó jelzés érkezik, megtörténik a programállapot szó tartalmának kimentése. És tovább mi történik? Nyilvánvaló, hogy a megszakítási okot ki kell küszöbölni. Láttuk azonban, hogy többféle okból következhet be megszakítás. Ezért a továbblépés menete a következő módon adható meg:

meg kell állapítani a megszakítási okot, gondoskodni kell a megszakítási ok megszüntetéséről és arról, hogy mi történjék, amíg a megszakítási ok kiküszöbölése folyamatban van.

A megszakítás kiküszöbölését a rendszerszoftver erre a célra szolgáló programjai végzik. A IX. fejezetben azt is látni fogjuk, hogy hogyan.

Annak a kérdésnek a megválaszolása, hogy „mi történjék, amíg a megszakítási ok kiküszöbölése folyamatban van", szintén túlmutat a hardver kérdéskörén. Ezzel a problémával a későbbiekben a multiprogramozás tárgyalásakor találkozunk ismét. ● | 261 | 18

Kérdések

1. Mi a bájt, és mi a rekesz?
2. Mit értünk egy rekesz címén?
3. Mit értünk egy rekesz tartalmán?
4. Mi a regiszterek feladata?
5. Mi a címregiszter és az adatregiszter feladata?
6. Mi a ciklusidő?
7. Milyen részekből áll a központi egység?
8. Mit az aritmetikai-logikai egység feladata?
9. Mi a vezérlőegység feladata?
10. Mit nevezünk gépi kódnak?
11. Hogyan kell kiszámítani a tényleges tárcímet?
12. Hogyan történik egy gépi kódú utasítás végrehajtása?
13. Mi a mikroprogram?
14. Mi az utasításszámláló regiszterek feladata?
15. Mi az adatsín szerepe?
16. Mi a csatornák funkciója, miért van szükség csatornákra?
17. Milyen csatornákat ismersz?

Feladatok

1. Készíts algoritmust két, egyenként ötjegyű szám összeadására, ha a két szám a következő módon van megadva:

Az egyik szám számjegyei az $a_1, a_2, \ldots, a_5$ tömbelemekben, a másik szám számjegyei a $b_1, b_2, \ldots, b_5$ tömbelemekben (egy tömbelemben egy számjegy). A kapott eredményt nyomtasd ki!

A központi tárnak azt a tulajdonságát, hogy a benne tárolt tömb elemei tetszés szerinti sorrendben elérhetők, felhasználhatjuk a következő feladatok megoldásakor.

2. Készíts algoritmust, amely egy n × n méretű kétdimenziós tömbről eldönti, hogy szimmetrikus-e. Egy kétdimenziós tömböt akkor nevezünk szimmetrikusnak, ha $a_{i,j} = a_{j,i}$ ($i = 1, 2, \ldots, n, j = 1, 2, \ldots, n$).

Tervezd meg az algoritmus inputját és outputját is!

3. Egy n × n méretű kétdimenziós tömb alsó háromszög része azoknak az elemeknek a halmaza, amelyekre igaz, hogy a sorindex (i) nagyobb, mint az oszlopindex (j).

Készíts algoritmust, amely kiszámítja, hogy egy n × n-es egész típusú tömb alsó háromszögében hány olyan elem van, amelyik páros!

Alprogramként alkalmazz egy olyan algoritmust, amely magát a párosságvizsgálatot végzi el!

Tervezd meg a feladat inputját és outputját!

4. Egy n × n méretű kétdimenziós tömb felső háromszög része azoknak az elemeknek a halmaza, amelyekre igaz, hogy a sorindex (i) kisebb, mint az oszlopindex (j).

Készíts algoritmust, amely kiszámítja, hogy egy n × n-es valós típusú tömb egyes soraiban mennyi a felső háromszögbe eső elemek összege!

Tervezd meg a feladat inputját és outputját!

5. Egy üres sakktáblára egy huszárt helyezünk. Készíts algoritmust, amely meghatározza azokat a mezőket, ahová a huszár léphet! (A sakktáblát egy 8×8-as tömbként kezeld, mindkét irányba számokkal azonosítva a mezőket! Az induló mező indexeit az m és n változók tartalmazzák.)

A feladat inputja az induló mezőhöz tartozó indexek, outputja a lehetséges mezőket azonosító indexpárok.

6. Kis elemsűrűségű tömb feltöltése. Egy 10×10 méretű tömb mintegy 20—25 elemet kivéve nullákból áll. Inputként csak a nem nulla elemek vannak megadva oly módon, hogy egy-egy rekordban az elem sorindexe, oszlopindexe és értéke van. Az utolsó rekordot fájl vége jel követi.

Készíts algoritmust, amely előállítja a teljes tömböt!

7. Oldd meg az 5. feladatot azzal az eltéréssel, hogy a tábla nem üres, hanem azon részben „ellenséges", részben saját bábuk vannak. Azoknak a mezőknek az indexeit, amelyeken bábu áll, inputként kell kezelni.

Ha az indexpár után 1 áll, akkor saját figuráról van szó, ha 2, akkor ellenséges figuráról. (A saját figurát nem szabad ütni, az ellenfelét igen.)

Tervezd meg a feladat inputját is!

VI. FEJEZET

Programozási nyelvek, fordítóprogramok

A központi tár és a központi egység működését megismerve választ kaptunk arra a kérdésünkre, hogy miképpen hajt végre a számítógép egy programot.

Még ezt megelőzően megtanultuk, hogy egy számítógépes program két részből tevődik össze: az adatdefiníciós részből és a vezérlésleíró részből. Azt is tudjuk, hogy a vezérlésleíró rész algoritmusok formájában tervezhető meg. Problémánkat a következőképpen fogalmazhatjuk meg:

A hardver, „amin a program fut", csak 0 és 1 bitek kombinációjából álló programot képes végrehajtani. Az ember, amikor egy problémát számítógéppel akar megoldani, megtervezi, majd elkészíti a programot. A tervezéshez — ha a probléma bonyolult — pl. a funkcionális lebontás módszerét használhatja. Amikor mind a programban szereplő adatok pontos definiálása, mind a teljes vezérlési szerkezet kialakult, szükségessé válik a program elkészítése. **De milyen nyelven?**

A programozó nem tudja a gép anyanyelvét, a gépi kódot (nem is vágyik rá, hogy megtanulja). A gép nem tudja a programozó anyanyelvét, a köznapi beszélt nyelvet (és egyelőre még várni kell egy darabig, hogy erre alkalmas legyen). Ez nyilvánvaló ellentmondás. További ismeretekre van szükségünk a programozás területén. Ebben a fejezetben ezt az ellentmondást fogjuk feloldani. Megismerkedünk a gépi kódnál magasabb szintű programozási lehetőségekkel és mindazzal a szoftverrel, ami ahhoz kell, hogy ez a kommunikáció mindenki számára elfogadható formában valósuljon meg. Tisztázni fogjuk a szoftver általános fogalmát, megvizsgáljuk a szoftver körét alkotó programok tagozódását, az egyes csoportba tartozó programok funkcióit. Megismerkedünk a programozási nyelvek közül néhány fontosabb nyelv sajátosságaival, alkalmazási lehetőségeivel. Miért fontos mindez számunkra?

Arról van szó, hogy a hardver „butaságát" kell elrejteni a felhasználó elől. Ez az elrejtés a gép szoftver úton való intelligensebbé tételével lehetséges. A szoftver erőforrások szerepét és súlyát éppen ez a tény határozza meg.

Egy mai értelemben vett számítógép már képtelen lenne működni mindazon szoftver erőforrások hiányában, amelyek körülveszik a hardvert. Amikor a felhasználó egy feladat megoldása kapcsán igénybeveszi a számítógépet, végeredményben már nem is tudja, hogy a megoldásban milyen súllyal vesz részt a hardver és mekkorával a szoftver. Ily módon a felhasználó egy látszólagos (virtuális) berendezéssel áll szemben. Ez a látszólagos gép a felhasználhatóság szempontjából jobb (hatékonyabb),

mint maga a hardver. A látszólagos berendezés a hardverből és az őt körülvevő szoftver rétegek sorozatából áll. Azért mondjuk „látszólagosnak", mert ez a szoftver felszereltség nem látható tulajdonsága egy számítógépnek. Kezdjük vizsgálódásainkat a szoftver fogalmának körüljárásával. Idézzük fel az I. fejezetben megfogalmazott szoftvermeghatározást: A szoftver a számítógépes programok és programrendszerek összefoglaló neve. Ezek közvetlenül a hardverre épülve, vagy már meglevő szoftveren keresztül a számítógép használatának lehetőségeit bővítik.

A szoftver körébe tartozó programokat és programrendszereket attól függően csoportosítjuk, hogy a számítógép használatának lehetőségeit milyen irányba és milyen jelleggel bővítik. Ebben a megközelítésben két fő csoportot különböztetünk meg. Az egyikbe azok a programok tartoznak, amelyek alapvetően a számítógép működésének hatékonyságát hivatottak biztosítani. Ezek képezik a **rendszer-szoftver** körét, és tovább bonthatók, most már attól függően, hogy a hatékonyságnövelés elsősorban kinek a munkáját teszi könnyebbé:

— a felhasználó (programozó) munkáját elsősorban a **feldolgozó programok** könnyítik meg, a programozási munka kényelmesebbé tételével és az adatállományok és programok kezelésének egyszerűsítésével;

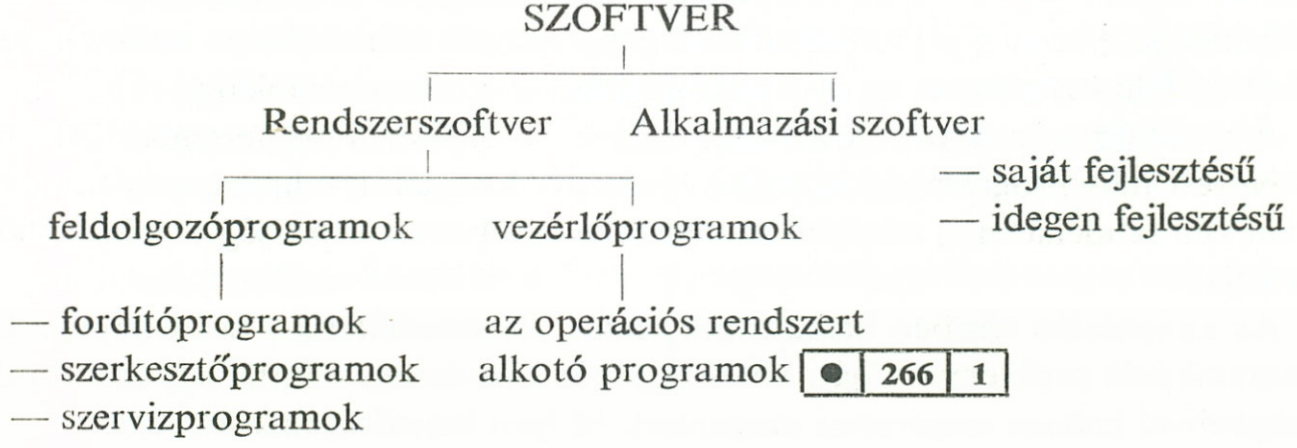
— az üzemeltető és az operátor munkáját elsősorban a **vezérlőprogramok** egyszerűsítik le, a számítógép erőforrásaival való gazdálkodás megvalósításával. Ezekből az előnyökből természetesen a programozó is részesül.

A szoftver másik csoportjába az **alkalmazási programok** tartoznak. Az alkalmazási programok különböző konkrét feladatok vagy feladatcsoportok gépi megoldásait tartalmazzák. Ezeknek a programoknak a végrehajtása mindig azért történik, hogy a felhasználó információk birtokába kerüljön. (Ezeket az információkat az adatfeldolgozás során keletkezett eredményadatok hordozzák.)

Az alkalmazási programok egy részét a felhasználók maguk készítik, más részüket készen veszik át vagy veszik meg szoftverkészítéssel foglalkozó cégektől.

A teljes hardver és a rendszer-szoftver azért van, hogy az alkalmazási programok végrehajtása gyors, gazdaságos és hibamentes lehessen.

A szoftver tagozódását a következő táblázat foglalja össze:



A PROGRAMOZÁSI NYELVEK

A világon létező programozási nyelvek száma több százra tehető. Ez rettentően nagy szám, de az igazsághoz az is hozzátartozik, hogy ezek közül a széles körben elterjedt nyelvek száma nem sok, legfeljebb néhány tíz.

E nyelvek jelentőségét akkor értjük meg, ha visszakanyarodunk a hardver adta lehetőségekhez, és megnézzük, hogyan volt működtethető a számítógép e nyelvek megszületése előtt.

Az előző fejezetben tanultak alapján tudjuk, hogy a számítógép alapvetően csak a saját gépi kódját képes megérteni és végrehajtani. Egyéb lehetőség hiányában egyetlen lehetőség adódik: gépi kódban kell elkészíteni a feladatmegoldó programot. [●] [267] [2] A gépi kódról rendelkezésre álló ismereteink alapján azonban megállapíthatjuk, hogy a gépi kód által nyújtott lehetőségek körülményessé és nehézkessé teszik a programozási munkát.

A gépi kódú programozás főbb korlátait a következőkben összegezhetjük:

— a műveleti kódokat numerikus (általában hexadecimális) formában kell megadni;

— a címeket abszolút tárcímben kell írni, vagyis a program készítésekor ismerni kell a tár vagy egy tárrész pontos felosztását;

— a program csak akkor hajtható végre, ha az abszolút címek által kijelölt tárterületre történik a program betöltése;

— minden műveletet elemi szinten kell megadni, vagyis olyan algoritmust kell szerkeszteni — a nyelv struktúrája miatt —, amelynél egy-egy utasításban csak egyetlen hardver utasítás szerepel;

— már korábban elkészített programrészek csak akkor kapcsolhatók össze újabb részekkel, ha az eredetileg kijelölt tárterületeket kívánjuk használni. Az új programrész kialakításakor figyelemmel kell lenni arra, hogy lefoglalt tárcímek ne kerüljenek ismételt felhasználásra;

— a megírt program jóságára vonatkozóan semminemű információ nem áll rendelkezésre mindaddig, amíg a program le nem fut vagy a végrehajtás során el nem akad. Ez utóbbi esetben sincs információ a hiba jellegére vonatkozóan.

Láthatjuk, hogy a gépi kódban való programozás nehézkessége mellett e programoknak a felhasználása és az újra felhasználása is igen sok problémát rejt.

A gépi kód helyett más nyelvi megoldás lehetőségét kell megteremteni. Ha a gépi kód helyett más programozási nyelvet kívánunk használni, ennek olyan tulajdonságokkal kell rendelkeznie, amelyek az előzőekben felsorolt nehézségek kiküszöbölését biztosítják.

Az annotációs részben bemutatott példák jól szemléltetik a számítógéppel való kommunikáció problematikáját. Kissé úgy néz ki a dolog, mintha egy távoli, idegen országbelivel kellene megértetni magunkat. Milyen lehetőségeink lennének:

Az egyik, hogy megtanítanánk egy olyan nyelvre, amit mi magunk ismerünk. (Nem okvetlenül magyarra.)

A másik lehetőség, hogy mi magunk megtanulnánk egy olyan nyelvet, amit ő ismer.

Harmadik lehetőség, hogy kerítenénk egy tolmácsot, aki mindkettőnk nyelvét ismeri. (Ha vendégünk elég ritka nyelvet beszél, talán nem is találunk ilyet.)

Utolsó lehetőségként adódik, hogy mindketten megtanulnánk egy olyan nyelvet, amelyet még egyikünk sem ismer.

Ha lehetőségeinket alaposan végiggondoljuk, hamar rájöhetünk, hogy a számítógéppel kapcsolatban egyedül az utolsónak említett lehetőség a kivitelezhető, ha a kapcsolatot mindenki számára biztosítani akarjuk. ▲ 265 3

A következő probléma, hogy vajon ez a mindkét fél által megtanulandó nyelv hol helyezkedik el (vagy hol helyezkedjék el) azon a skálán, amely a beszélt emberi nyelv és a gépi kód között van. (Láttuk, hogy ez a távolság meglehetősen nagy.)

Megalkotandó közbülső nyelvünkkel minél közelebb maradunk a számítógép hardverjének színvonalához, annál nehezebb marad e nyelv elsajátítása és használata az ember számára. És fordítva, minél közelebb visszük e nyelvet az emberhez, annál nagyobb energiát kell fektetni a gépet „okosító" szoftver előállítására. Igen ám, de ezt az energiát csak egyszer kell befektetni, s ettől kezdve az elkészült szoftver segítségét már csak sokszorosítani kell. Az előző esetben pedig mindenkinek külön-külön kell megharcolnia a maga tudásáért.

Ebből a gondolatsorból arra a következtetésre juthatnánk, hogy ezek szerint a programozási nyelvek nagyon is emberközeliek. Ez azonban csak részben igaz. Miért?

Először is a programozási nyelvek fejlődése időben lejátszódó folyamat, amelynek korai szakaszában még költség és egyéb (pl. hardver) okokból csak olyan nyelvek készültek, amelyek nem távolodtak el túlságosan a gép kód színvonaláról, de akkor mindenképpen előrelépést jelentettek. Másrészt egy nyelv minél közelebb van a gép kódszintjéhez, annál inkább alkalmas arra, hogy optimálisan használja ki a hardver erőforrásokat. S amíg ez döntő volt — az erőforrások szűkös volta miatt — sok program készült ilyen nyelven. De a számítógépek egyre tömegesebb elterjedésével, különösen a nyolcvanas években, amikor a mikrogépek mindenhol megjelennek, elengedhetetlen, hogy minél emberközelibb nyelvek alakuljanak ki.

Amint arra az előbb már utaltunk, a „közbülső nyelvek" kialakítása több lépésben történt. E fejlődési folyamat eredményeképpen sok programozási nyelv jött létre. Ezek közül azok, amelyek túlélték az újabbakat, két nagy csoportba sorolhatók:

— a géphez közelebb álló (és általában géptípushoz kötött) *assembly (esszembli)* szintű nyelvek;

— a programozóhoz közelebb álló (és géptípustól független) *magas szintű programozási* nyelvek.

Az assembly szintű nyelv a gépi kód alapvető problémáit oldja fel:

— A műveleti kód nem számjegykombináció, hanem betűkombinációból álló (mnemonikus = megjegyezhető) kód. A betűkombináció általában a műveletet leíró ige rövidítése.

— Az operandus címek nem a tár abszolút címei, hanem szimbolikus nevek. Egy programon belül egy adott szimbólum mindig ugyanazt a tárcímet képviseli.

— Az egyes utasításokra való hivatkozás nem az utasítás tárbeli címével történik, hanem egy szabadon választható címke (szimbolikus név, akár egy változónév) segítségével, amelyet a megcímkézni kívánt utasítás elé kell írni. Erre az utasításra a program bármely pontjáról a címkével lehet hivatkozni (pl. vezérlésátadás esetén).

Az assembly szintű programozásról elmondottakat jól nyomon kísérhetjük a következő példán. A feladat az első n természetes szám szorzatának előállítása. Assembly programozási nyelvünk legyen fiktív, a gépi kódú utasítások a következő mnemonikus kódokkal egyeznek meg: (A jelentés után zárójelben az a szó, aminek rövidítéséből a megjegyezhető kódot képeztük.)

Gépi kódú műveleti kód	Mnemonikus	Jelentés
01	PLU	összeadás (plusz)
02	MIN	kivonás (mínusz)
03	SZR	szorzás (szor)
04	PER	osztás (per)
05	TLT	töltés első címről a másodikra (töltés)
06	UGR	feltétel nélküli ugrás (ugrás)
07	UHA	feltételes ugrás nem negatív eredmény esetén (ugrás ha)
10	INP	olvasási utasítás (input)
11	OUT	íratási utasítás (output)
12	STP	stop — program megállítása

(Ezek az utasítások azonosak az annotációs részben található fiktív gépi kódnál bevezetett utasításokkal.)

És még egy utasítás, amelyre szükségünk van:

TLK konstans érték töltése a második címre.

Szimbolikus nevekként azokat használjuk, amelyeket a gépi kódú változat ma-

gyarázó részében is alkalmaztunk a fejezet [●|267|2] annotációjában. Ezek után programunk a következő:

```
...
        INP     N
        TLK     1       I
        TLK     1       M
ODA     SZR     I       M
        PLU     1       I
        TLT     I       W
        MIN     N       W
        UHA     ODA
        OUT     M
        STP
```

Megjegyzés: A program elején levő ... azt jelzi, hogy bizonyos adatdefiníciós tevékenység még kívánkozik a program elejére, ennek hiánya azonban a program szemléltető jellegét nem befolyásolja.

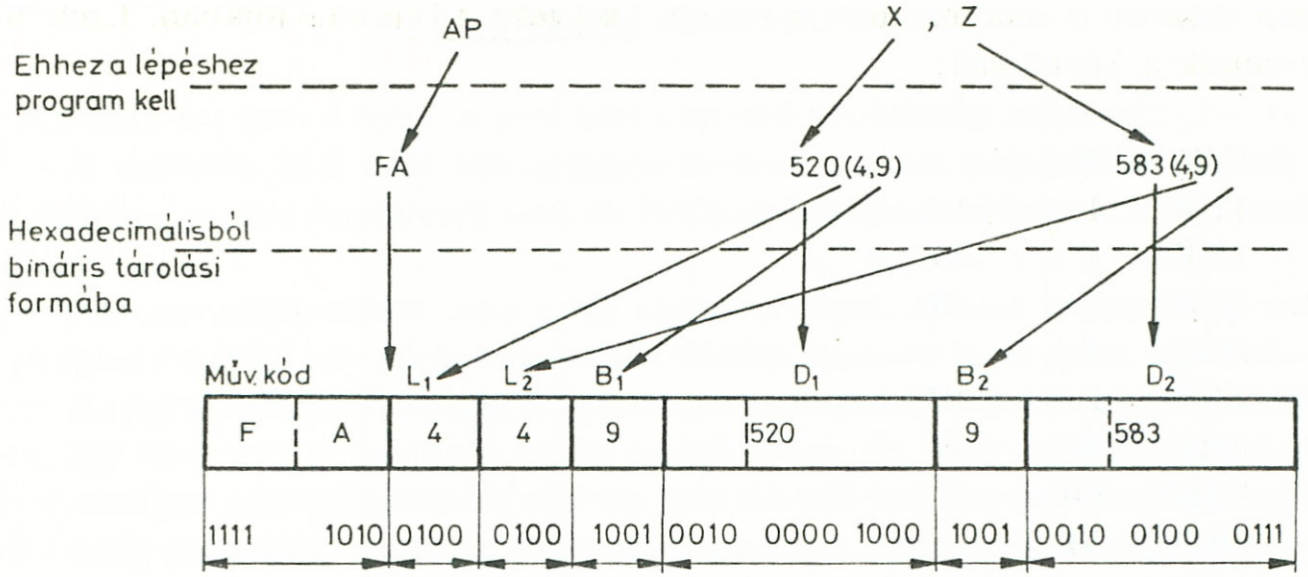
Ez a példa fiktív nyelven mutatta meg a megoldást. Hogyan néz ki mindez a valóságban?

Ennek szemléltetésére bemutatunk egy valóságos gépi kódhoz tartozó néhány utasítást, hexadecimális és mnemonikus kóddal. (E gépi kód az előző fejezetben is tárgyalt bájt szervezésű gépeknél található meg.)

Hexadecimális kód	Mnemonikus kód	Jelentés	Típus
58	L	betöltés regiszterbe	RX
18	LR	regiszter áttöltés	RR
1A	AR	regiszterek összeadása	RR
4B	SH	félszavak közötti kivonás	RX
46	BCT	ugrás a ciklusszámláló figyelésével	RX
FA	AP	decimális összeadás	SS

Megjegyzés: Első látásra talán a „megjegyezhető" kód is épp oly értelmetlennek tűnik, mint a numerikus, de némi angol tudással kiderül, hogy értelmes angol szavak vagy kifejezések rövidítéséről van szó. Pl.: LR = Load Register stb.

A 49. ábrán végigkísérhetjük, hogy egy assembler utasítás miképpen felel meg a gépi kódú változatnak:



49. ábra. Az utasítás assembler, hexadecimális és bináris formája

A számítógép ezt a nyelvet — természetesen — nem érti. A műveleti kódot, az utasítások és operandusok helyét vissza kell alakítani megfelelő számjegy-kombinációkká. Ez azonban egy olyan tevékenység, amelyet maga a számítógép is képes lenne elvégezni, ha a szükséges lépéseket egy program előírja.

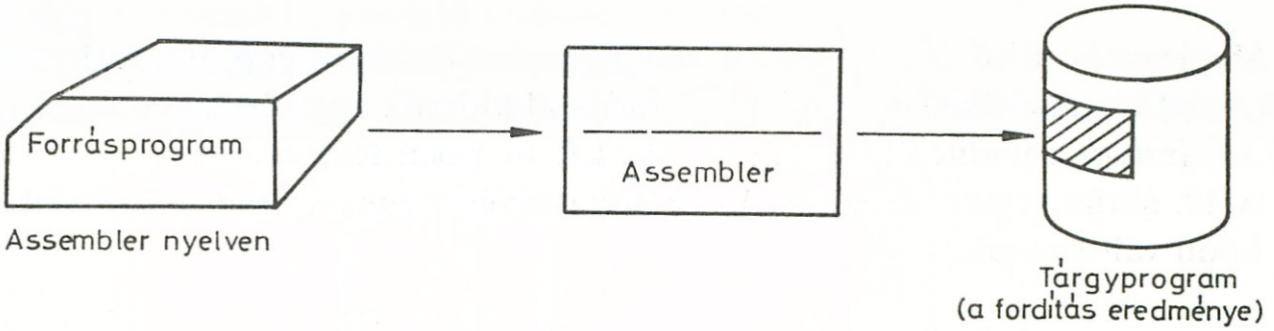
Anélkül, hogy a részletekbe bocsátkoznánk, pl. a mnemonikus műveleti kódok visszaalakításához mindössze egy szótár szükséges, amelyben az egyik oszlopban a mnemonikus kódok, a másikban a megfelelő numerikus kódok találhatók. A program az átalakítandó betűkombinációt keresi az első oszlopban, s ha megtalálja, a második oszlopból kiolvassa azt a numerikus kódot, amelyet a mnemonikus helyére kell írni.

Az eljárásnak megvan az az előnye is, hogy amennyiben egy hibás mnemonikumot nem talál meg (mert nem is létezik), hibaüzenetet küld, s így már a futás előtt kiderül, hogy melyik utasítás hibás (ebből a szempontból).

Azokat a programokat, amelyek assembly szintű programokat visszaalakítanak gépi kódra, **assemblereknek** nevezzük. (Miután a különböző gépek különböző gépi kódjaihoz különböző assemblerek szükségesek, az „assembler" egy gyűjtőfogalom. A bájt szervezésű gépeknél e program neve is Assembler. Ha a továbbiakban erről a programról lesz szó, a nevet nagy kezdőbetűvel írjuk.)

Az Assembler inputját **forrásprogramnak,** outputját **tárgyprogramnak** nevezzük. Ezt szemlélteti az 50. ábra.

A forrásprogram és a tárgyprogram ugyanazon feladat megoldása, az eltérés az



50. ábra. Forrás- és tárgyprogram

adatdefiníciót és az algoritmust rögzítő programozási nyelvben van. A továbbiakban a programozó által írott nem gépi kódú programot mindig forrásprogramnak, a gép által készített fordítást tárgyprogramnak nevezzük.

Az Assembler eddig megismert tevékenységéből következik, hogy az általa létrehozott tárgyprogram ugyannyi utasításból áll, mint a forrásprogram. Ez kezdetben igaz is volt.

A programozás során azonban sokszor fordult elő, hogy egyes utasításcsoportok rendre megismétlődtek. Egy-egy ilyen utasításcsoport valamilyen elemi programlogikai funkciót lát el. Ilyen pl. a regiszterek tartalmának kimentése.

Célszerűnek látszott, hogy ezeket az utasításokat ne kelljen állandóan megismételni, hanem egyszeri leírás után hivatkozni rájuk.

Így jött létre a *makroutasítás* (vagy röviden makro), ami nem más, mint egy utasításcsoportra való hivatkozás. ⊙ | 279 | 4

Megjelenésével lehetségessé vált, hogy a programozó egyetlen utasítással (ún. makroutasítással) adjon meg olyan tevékenységcsoportot, amely a végrehajtás szintjén egy egész utasításcsoportot jelent.

Az assembly szinten való programozás, bár a gépi kódhoz közelebb áll, mint a magasszintű programozási nyelvek, a makrók adta lehetőségekkel már alkalmas arra, hogy bizonyos esetekben a programozási munka eszköze legyen. Az Assembler-nek ugyanis — sok hátránya mellett — van egy előnye: éppen gépközelsége miatt hatékonyabban képes az erőforrások mozgatására. Ez a hatékonyság — elsősorban a kisebb teljesítményű és lassúbb gépek esetében — a tár jobb kihasználásában, a program rövidebb futási idejében mutatkozik meg.

E meggondolásból pl. magukat a rendszerprogramokat sokáig kizárólag Assembler-ben írták. Az elmúlt években bekövetkezett sebesség- és kapacitásnövekedés megváltoztatta a programozásnak ezt a szemléletét. Ma már egy olyan program, amely jól áttekinthető struktúrákból épül fel, s ezért könnyen érthető, javítható és változtatható, többet ér, mint egy másik, amelyik néhány tucattal kevesebb helyet foglal el a tárban, vagy pár másodperccel hamarabb fut le, de forrásnyelvi változata nehezebben kezelhető és elkészítése is hosszabb időt igényel. (A harmadik fejezetben részletesen tárgyaltuk ezt a problémakört.)

A rendszer méreteinek növekedésével egyre inkább a kezelhetőségi szempontok kerülnek előtérbe. Így azután ma már természetes (egy évtizede még elképzelhetetlen volt), hogy rendszer-szoftvert magas szintű programozási nyelven készítenek el. ⊙ | 280 | 5

Magas szintű programozási nyelvek

A makrók adta lehetőségek az assembly szinten való programozást lényegesen megkönnyítik. A széles körű elterjedésnek azonban továbbra is akadálya volt, hogy a nyelv konkrét géptípushoz kötődött, s ezért használata meglehetősen sok gépi ismeretet kíván.

A magas szintű programozási nyelvek kialakítása éppen ezeknek a problémáknak a megoldását célozta. Ezeknek a nyelveknek a közös jellemzője a gépfüggetlenség, abban az értelemben, hogy a programozónak nem kell tudnia, hogy programja milyen gépen fog futni. A forrásprogramot mindegyik gép a saját gépi kódjára fordítja le.

A magas szintű programozási nyelvek másik közös tulajdonsága, hogy utasításkészletük igyekszik az emberhez igazodni. Pl. a matematikai műveleti jelek a matematikában megszokott formában használhatók. Az egyes utasításszavak önmagukban is értelmes (általában angol) szavakból vagy kifejezésekből állnak stb.

Jól szemlélteti az elmondottakat a FORTRAN programozási nyelv, amely időrendben az első volt e nyelvek sorában. Pályafutását assembly szintű nyelvként kezdte, de több fejlesztés eredményeképpen három évtizede jól tartja előkelő helyét a magas szintű nyelvek között. (A hatvanas években úgy tűnt, hogy kiszorul a széles körű hazai alkalmazásból, a mikroszámítógépek megjelenése azonban ismét kedvező fordulatot hozott a FORTRAN számára: relatíve kis tárigénye miatt sok mikrogépen használható.)

Ha A, B és C egy háromszög oldalait jelenti, e háromszög területének kiszámítása a Heron-képlettel FORTRAN-ban így írható fel:

```
S = (A+B+C)/2.
TERUL = SQRT (S * (S—A) * (S—B) * (S—C))
```

Ha tudjuk, hogy **SQRT** a négyzetgyökvonást, * a szorzást, / az osztást, — a kivonást jelenti, a programrészletet pontosan el tudjuk olvasni és meg tudjuk érteni.

Más programozási nyelvek más meggondolás alapján igyekeztek közelíteni az algoritmus leírást az ember által használt nyelvi szerkezetekhez. Pl. a COBOL, amely a gazdasági (üzleti) élet legkülönbözőbb területén tevékenykedő személyek munkájának megkönnyítésére készült, megengedi, hogy még a matematikai műveleteket is szövegesen adjuk meg, hogy matematikai jártasság nélkül is olvasható legyen a program. Pl. COBOL-ban az ADD PENZ TO OSSZEG utasítás az OSSZEG: = OSSZEG + PENZ értékadásnak felel meg.

A FORTRAN és a COBOL összehasonlítása azért is érdekes, mert a magas szintű programozási nyelvek között a két szélsőséget képviselik. A FORTRAN a „tudományos számítások" programozására létrehozott nyelv, a COBOL az „adatfeldolgozásra" orientált nyelv, amely elsősorban az üzleti élet számára készült. (A szűkebb értelemben vett adatfeldolgozásra kell most gondolnunk.)

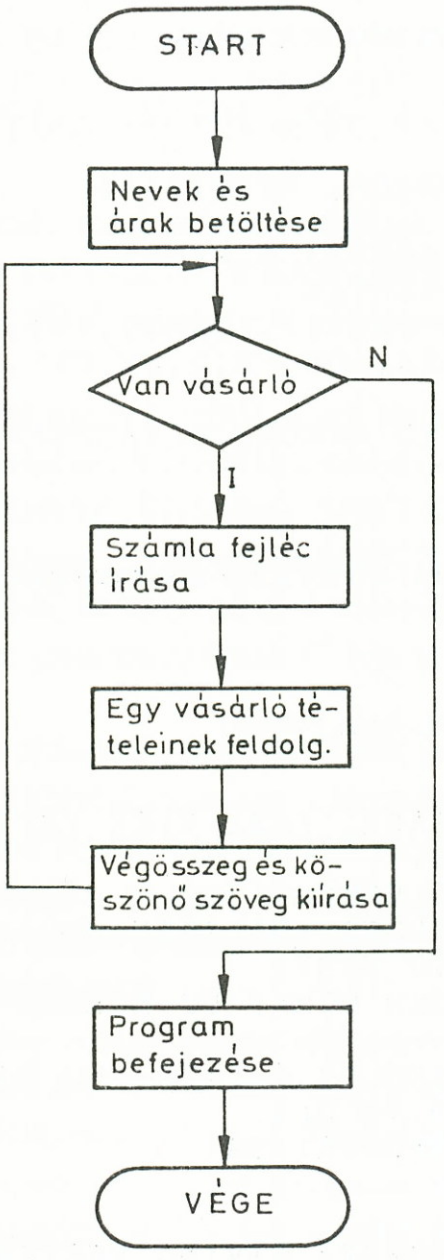
A két nyelv összehasonlítása és ezen keresztül a magas szintű programozási nyelvek néhány fontosabb tulajdonságának vizsgálata érdekében bemutatjuk ugyanannak a feladatnak a megoldását FORTRAN-ban és COBOL-ban.

A feladat egy egyszerűsített számlázóprogram elkészítése. A program először külső tárolóról betölti a központi tárba a lehetséges áruk megnevezését és egységárát. Száz lehetséges áru van. A név legfeljebb 24 karakter hosszúságú (karakter típusú), az egységár valós típusú ötszámjegyű szám, két tizedes pontossággal. Ezután a program egymásután olvassa a vásárlási tételeket, amelyek rekordképe: a vásárolt áru kód-

száma (1 és 100 közötti szám, ami megegyezik az árunak a külső tárolón levő sorszámával), a vásárolt mennyiség (valós adat).

Egy-egy vásárló utolsó tételét egy-egy —1 követi. Az utolsó vásárló —1 jelét fájl vége jel követi. A program minden vásárló számára számlát készít, tételesen felsorolva a vásárolt áruk nevét, a vásárolt mennyiségeket, az áruk értékét és legvégül, hogy mennyit kell fizetni az egészért. A számla tetején fejléc, végén „Köszönjük a vásárlást" szöveg található.

A megoldás menete első közelítésben:



51. ábra. Első közelítésű funkciómeghatározás

Az „egy vásárló tételeinek feldolgozása" rész FORTRAN-ban a következő:

```
C
C  VASARLASI REKORD OLVASASA, FAJL VEGE ESETEN
C  UGRAS A PROGRAM VEGERE
C
 5     READ (5,10,END=100) SORSZ,MENNY
 10    FORMAT (I4,F7.2)
C
C  VASARLASI REKORDOK VEGE? HA IGEN UGRAS AZ 50 CIMRE
C
       IF (SORSZ.EQ.-1) GO TO 50
C
C  HIBAS ARUAZONOSITO? HA IGEN UGRAS A 20 CIMRE
C
       IF (SORSZ.LT.1 .OR. SORSZ.GT.100) GO TO 20
C
C  REKORD FELDOLGOZASA, NYOMTATAS
C
       SERTEK = AR(SORSZ) * MENNY
       SUM = SUM + SERTEK
       WRITE (6,15) (NEV(J,SORSZ),J=1,6),MENNY,SERTEK
 15    FORMAT(11X,6A4,10X,F8.2,8X,F11.2)
C
C  UGRAS A KOVETKEZO REKORD OLVASASARA
C
       GO TO 5
```

52. ábra. FORTRAN programrészlet

Ugyanez a programrész COBOL-ban pedig:

```
 OLV.
*       VASARLASI REKORD OLVASASA, FAJL VEGE ESETEN UGRAS A
*            VEGE PARAGRAFUSRA.
     READ  KARTYA AT END GO TO VEGE.
*        VASARLASI REKORDNAK VEGE?, HA IGEN UGRAS A KESZ PARAGRAFUSRA
     IF SORSZ = -1    THEN GO TO KESZ.
*       HIBAS AZONOSITO? HA IGEN UGRAS UJ REKORD OLVASASRA

     IF NOT  JOSORSZ  THEN  DISPLAY 'ILYEN ARU NINCS' SORSZ
                            GO TO OLV.
*       REKORD FELDOLGOZASA, NYOMTATASA
     MOVE NEV (SORSZ) TO LNEV
     MOVE  MENNY TO LMENNY
     MULTIPLY MENNY  BY  AR (SORSZ) GIVING SERTEK
     ADD  SERTEK TO WERTEK
     MOVE SERTEK TO LERTEK
     WRITE LREK AFTER ANDVANCING 1
     ADD 1 TO TETEL.
     IF TETEL > 60 THEN PERFORM FEJIRAS.
     GO TO OLV.
```

53. ábra. COBOL programrészlet

Keressük meg először a hasonlóságokat a két programrészletben. Rövid tanulmányozás után mindkét programrészletben felfedezhetjük a magas szintű programozási nyelvek alapelemeit:

— a konstansokat: pl. 1, 100, a tízes számrendszer szabályai szerint;

— a változókat: az azonos tartalmú változókat mindkét nyelven azonos névvel neveztük el. Ez egyben azt is jelenti, hogy egy programban a változóneveket a programozó (a nyelv szabályain belül) szabadon választhatja. Ilyen változónevek pl.: SERTEK, MENNY, AR stb.;

— a címkéket: a címkék a vezérlésátadáshoz szükséges információk. A program adott pontjáról a vezérlés nem a soron következő, hanem a címkével azonosított utasításra adódhat. A COBOL címkék (eljárásnevek) betűkombinációkból is állnak, mint a változónevek pl.: OLV. A FORTRAN címkéi (ott utasításszám a nevük) csak numerikusak lehetnek, pl.: 5. Közös tulajdonságuk, hogy ezeket is szabadon választhatja a programozó;

— a nyelvi kulcsszavakat: ezek között találunk azonos formájúakat, pl.: **IF**, (ha... feltételes ugrás), **GO TO** (ugorj), **READ** (olvass), ezek ugyanazt az utasítástartalmat is hordozzák. A COBOL nyelvi programrészletben azonban szemlátomást több kulcsszó fordul elő.

Ezzel el is érkeztünk a két nyelv közötti egyik lényeges különbséghez: a COBOL minden adatmozgató és műveletleíró utasítása szabályos mondatszerkezetű: „MULTIPLY MENNY BY AR (SORSZ) GIVING SERTEK.", amit némi bőbeszédűséggel így lehetne magyarra fordítani: „Szorozd a MENNY változó értékét az AR tömb SORSZ-al azonosított elemével, és helyezd el a szorzatot SERTEK-ben."

A FORTRAN ugyanezt „matematikusan" írja le: SERTEK = AR(SORSZ)*MENNY

Egy másik szembetűnő különbség, amelyet a két programrészlet tanulmányozása során észreveszünk, hogy a COBOL-ban egy változónak lehet külön input területe (változóneve), ahová beolvasni lehet; lehet külön munkaterülete, ahol műveletet végezhetünk e változóval; és külön output területe, ahonnan az értékét kiírhatjuk. Az egyik területről a másikra külön utasítással kell átmozgatni a változók tartalmát, pl.: MOVE MENNY TO LMENNY

A FORTRAN változatban a beolvasott változóval lehet műveletet végezni vagy kiíratni, egy adathoz tehát csak egy tárterület (szó) tartozik.

Előnyös vonása a COBOL-nak, hogy a feladatmegoldáshoz szükséges rekordszerkezetek a program elején leírhatók, s a program további részében e rekordszerkezet elemeire vagy a teljes rekordra így is, úgy is lehet hivatkozni.

A WRITE LREC kezdetű utasítás az LREC névvel definiált teljes rekordot mozgatja, amelybe korábban szavanként már betöltésre került (MOVE utasítással) a LNEV, a LMENNY és a LERTEK változók tartalma. A feladat két nyelven elkészített teljes megoldása is tanulmányozást érdemel. | ● | 273 | 6 | | ● | 275 | 7 |

Az eddigiekben a két programozási nyelv egyes elemeit hasonlítottuk össze. Még nagyobb különbséget tapasztalhatunk, ha a FORTRAN és a COBOL nyelven írott programok szerkezetét vetjük össze.

A FORTRAN programok esetében tulajdonképpen szigorúan vett programszerkezetről nem is beszélhetünk. Ilyen vonatkozásban egyetlen szabály van, az adatdefiniáló utasításoknak a program elején kell állniok. Így a tömböket definiáló DIMENSION utasítás, a valós adatokat definiáló REAL, az egészeket definiáló INTEGER, a logikai típusú adatokat definiáló LOGICAL stb. utasításoknak meg kell előzniök a program első végrehajtó utasítását. A program utolsó utasítása pedig kötelezően END.

A COBOL esetében egy nagyon szigorú programszerkezet betartása kötelező. (Többek között ezért is hosszabbak a COBOL programok.)

E programszerkezet a következő séma szerint épül fel:

IDENTIFICATION DIVISION — azonosító főrész
PROGRAM ID.
(programazonosító)
ENVIRONMENT DIVISION — környezetleíró főrész
INPUT-OUTPUT SECTION.
FILE CONTROL.
(a programbeli logikai fájlnevek és a fizikai perifériák egymáshoz rendelése)
DATA DIVISION — adatleíró főrész
FILE SECTION.
(a programban szereplő fájlok belső szerkezetének (rekordképének) leírása)
WORKING STORAGE SECTION.
(a számítások során használt munkaváltozók tételes felsorolása)
PROCEDURE DIVISION — eljárási főrész
(a program vezérlőszerkezete)
STOP RUN.

Anélkül, hogy a vizsgálódásainkat folytatnánk, megállapíthatjuk, hogy a magas szintű programozási nyelvek — a létrehozásuk eltérő célja miatt — sok eltérést mutatnak, de az azonos funkció ▲ 265 8 miatt sok vonatkozásban hasonlítanak is egymáshoz. Egy közös vonásuk az eddig elmondottakból is nyilvánvaló: forrásprogramjuk lényegesen nehezebben fordítható vissza gépi kódra, mint ahogyan azt az assembly szintű nyelvek esetében láttuk.

Hogyan képes a fordítóprogram magas szintű forrásnyelven megírt programot visszaalakítani gépi közelségű tárgyprogrammá? E kérdésre akkor válaszolhatunk, ha megvizsgáljuk, hogy milyen feladatokat old meg a fordítóprogram a fordítás során.

A fordítás során az első feladat a programban levő alapelemek felismerése. Egy magas szintű nyelv alapelemei, amint arról már szó volt:

— a változók,
— a konstansok,
— a címkék és a
— kulcsszavak.

Pl. a következő programrészletben:

```
      IP=0
      IZ=0
      IN=0
      DO 2 I=1,N
      IF (ITO (I)) 10, 11, 12
10    IN = IN+1
      GO TO 2
11    IZ = IZ+1
      GO TO 2
12    IP = IP+1
 2    CONTINUE
```

a következő alapelemeket kell felismerni:

változók: N, I, ITO, IP, IZ, IN
konstansok: 0, 1
címkék: 2, 10, 11, 12
kulcsszavak: DO, IF, GO TO, CONTINUE

Az alapelemeket egymástól szóközök, műveleti jelek, speciális szimbólumok (pl. zárójel) választják el.

Az alapelemek felismerését és rendszerezését a fordítás *első* fázisa, a *lexikális analízis* végzi. A lexikális analízis eredményeképpen a programnak egy olyan állapota jön létre, amely a további feldolgozás alapja. A belső alapelemekre való lebontás után a *második* feladat a program *önálló szintaktikai egységeinek* felismerése és értelmezése. Pl.: az előbbi programban az

$$IZ = IZ+1$$

utasítás önálló szintaktikai egység, és egy helyes értékadó utasításként kerül értelmezésre.

Ebben a lépésben fedezi fel a fordítóprogram a szintaktikai hibák legnagyobb részét. Ezeket valamilyen lista formájában közli a programozóval.

A szintaktikai egységek értelmezését követően, *harmadik* feladatként a fordítóprogram előállítja a *forrásprogramnak egy közbülső formáját*, amelyben mind az aritmetikai utasítások, mind a vezérlésátadó utasítások, mind pedig a nem végrehajtható utasítások [▲ | 265 | 9] gépi kóddá generálható állapotban vannak. A fordítóprogramok legnagyobb része kisebb-nagyobb optimalizálást is végrehajt a programon. [● | 276 | 10]

A következő, *negyedik* feladat a program által igényelt *tárterület lefoglalása*, ill. kijelölése. Ez a tárfoglalás a programban szereplő változók és tömbök számára történik. A korábban létrehozott közbülső formában minden olyan információ összegyűjtve megvan, amelyből megállapítható a változó típusa, hossza, a tömbök mérete, egyszóval a szükséges szavak száma. Az egyes változó, ill. tömbnevekhez egymásutáni helyet foglal le, egy relatív 0 címtől kezdődően.

A fordítóprogram *ötödik* feladata a *gépi kód generálása.* A kódgenerálás makrók segítségével történik. A kódgeneráláson kívül *még egy feladat* van, a *címkék definiálása* és az *összes hivatkozás feloldása.*

Bár a fordítás fontosabb lépéseit egy FORTRAN programon keresztül mutattuk be, ez a hat lépés nyelvtől függetlenül, mindig így zajlik le. Mit lát mindebből a felhasználó, milyen információk alapján tudja megítélni, hogy a fordítás sikeres volt-e vagy pedig nem? Ha a fordítás sikeres volt, hol található a lefordított tárgyprogram? Ha a fordítás hibákat talált, honnan tudjuk, hogy mely utasítások voltak hibásak? Ezekre a kérdéseinkre a fordítás outputjai adják meg a választ.

A fordítóprogram végrehajtása során *többféle output* keletkezik, amelyeket a felhasználó kérése szerint a rendszer rendelkezésre bocsát. Ezen outputok közül az első a *forrásprogram kinyomtatása sornyomtatón.* A visszaírt program teljesen megegyezik a lyukkártyákon vagy terminálról bevitt forrásprogrammal, így tartalmazza az esetleges hibákat is. A rendszer minden utasítást (ill. minden nyomtatásra kerülő rekordot) egy-egy sorszámmal lát el. Az 54. ábrán egy FORTRAN nyelvű program kézzel írott változatát láthatjuk. [● | 277 | 11]

Ez a program lyukkártyára való rögzítés után a számítógépbe kerül. Ha a programfelvitel terminálon keresztül történik, természetesen előtte nem kell kódlapra leírni. A kódlap az adatrögzítést végző munkáját könnyíti meg.

A FORTRAN fordítóprogram az 55. ábrán látható listát szolgáltatja.

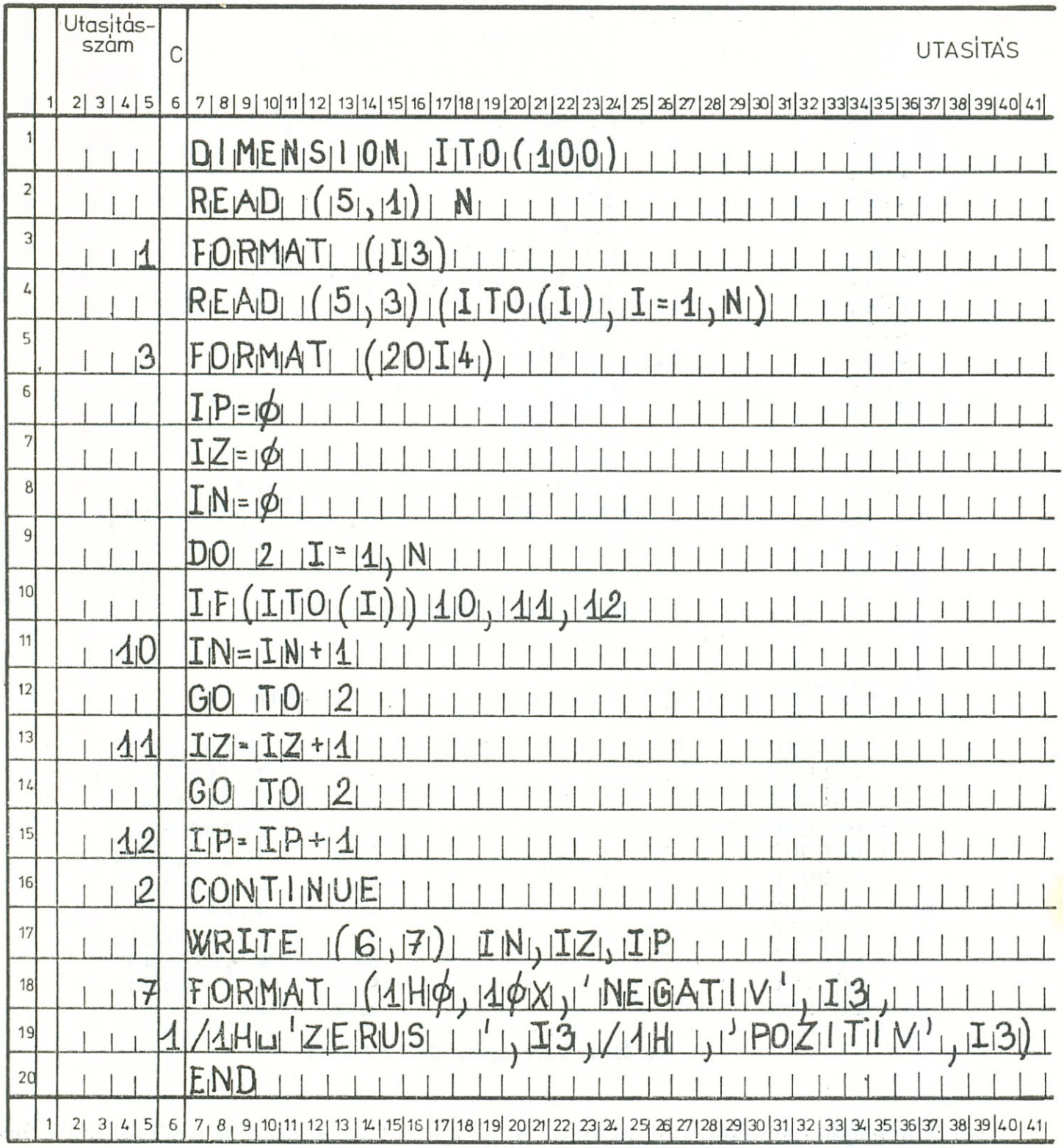
Kívánságra a fordítóprogram *listát* közöl a *tárgyprogramról* is. (E listától függetlenül, a tárgyprogram a további felhasználás céljaira egy mágneslemez területen is tárolásra kerül. A lista csupán a felhasználó tájékoztatására szolgál.) A tárgyprogram egy látszólagos assembly nyelven kerül nyomtatásra, vagyis nem gépi kódban, hogy a programozó szükség esetén könnyebben eligazodjék benne, de ugyanakkor nem is valódi assembly nyelven, hiszen a tárgyprogram ennél is alacsonyabb szintű. A lista oszlopok formájában rendezett, ahogyan azt az 56. ábra szemlélteti. Az egyes oszlopok jelentését a fejrovatok magyarázzák. (Az ábra nem tartalmazza a teljes tárgyprogramot, annak terjedelme miatt.) [● | 278 | 12]

A tárgyprogramnak egy másik megjelenítési formája lehet a lyukkártyákra való lyukasztás. Szerepe ma már kevésbé jelentős, mivel minden szempontból praktikusabb a háttértárolón való megőrzés. [● | 278 | 13]

A felhasználó megkaphatja a tárgyprogram előállítása során végrehajtott tárfelosztás eredményét, a *tárkivonatot*, vagy más néven memóriatérképet is. Ez gyakorlatilag egy táblázat azokról a szimbolikus nevekről, amelyeket a programban a programozó használt. A szimbólumok mellett megkapjuk azok relatív címét, tizenhatos számrendszerben. [● | 279 | 14]

Az 54. ábra programjának tárkivonatából az 57. ábrán láthatunk egy részletet.

A fordítás során a fordítóprogram felfedi a forrásprogramban levő formai hibákat. Ezekről, és a fordítás menetéről diagnosztikai üzenetek készülnek. A **diagnosztikai üzenetek** között legfontosabbak a **hibaüzenetek,** amelyek a formai hibák helyére és azok valószínű okára utalnak úgy, ahogyan azt az ábra szemlélteti. Ezek ismeretében



54. ábra. FORTRAN program kódlap

ben egyszerűbbé válik a hiba kijavítása. (Az 58. ábrán a 7. ábrán levő program egy korábbi (hibás) állapotát látjuk.)

Az ábrán levő hibajelzések tehát a program formai hibáit jelzik. Ezeket idegen szóval **szintaktikai hibáknak** nevezzük. A fordítóprogram akkor jelez szintaktikai hibát, ha olyan utasítást talál a fordítás valamelyik lépésében, amely ellentmond a nyelv szabályainak, vagy a korábban már leírtaknak.

Az ábrán levő első hiba azért keletkezett, mert a FORTRAN-ban a tömböket a

```
FORTRAN IV G LEVEL  21
 0001                DIMENSION ITO(100)
 0002                READ(5,1) N
 0003              1 FORMAT(I3)
 0004                READ(5,3)(ITO(I),I=1,N)
 0005              3 FORMAT(2014)
 0006                IP=0
 0007                IZ=0
 0008                IN=0
 0009                DO 2 I=1,N
 0010                IF(ITO(I))10,11,12
 0011             10 IN=IN+1
 0012                GO TO 2
 0013             11 IZ=IZ+1
 0014                GO TO 2
 0015              2 CONTINUE
 0016             12 IP=IP+1
 0017                WRITE(6,7) IN,IZ,IP
 0018              7 FORMAT(1HO,10X,'NEGATIV',I3,
                    1/1H 'ZERUS  ',I3,/1H ,'POZITIV',I3)
 0019                STOP
 0020                END
```

55. ábra. Lista a forrásprogramról

```
FORTRAN  IV G LEVEL  21                    MAIN

LOCATION    STA NUM      LABEL  OP      OPERAND
 000000                         BC      15,12(0,15)
 000004                         DC      06D4C1C9
 000008                         DC      D5404040
 00000C                         STM     14,12,12(13)
 000010                         LM      2,3,40(15)
 000014                         LR      4,13
 000016                         L       13,36(0,15)
 00001A                         ST      13,8(0,4)
 00001E                         STM     3,4,0(13)
 000022                         BCR     15,2
```

56. ábra. Tárgynyelvi lista (részlet)

program elején definiálni kell. A másik hiba oka: a 2 utasításszám 1-nél többször fordul elő.

Az egyes hibák nem azonos súlyúak. Vannak olyanok, amelyek kizárják a tárgyprogram további feldolgozását, és vannak olyanok, amelyek csak figyelmeztető jellegűek. Előfordulásuk esetén a további végrehajtás még sikeres lehet. A nem figyelmeztető jellegű hibák lehetetlenné teszik a tárgyprogram további feldolgozását. Ezért biztosítani kell, hogy erre a további feldolgozásra ne is kerülhessen sor, ha a forrásprogram hibás. ⊙ | 280 | 15

Igen ám, de mit jelent az, hogy a „tárgyprogram további feldolgozása"? Ez a kifejezés azt sugallja, hogy a tárgyprogramot a számítógép nem tudja végrehajtani,

```
                              SUBPROGRAMS CALLED
SYMBOL     LOCATION      SYMBOL     LOCATION      SYMBOL     LOCATION
IBCOM#          B8

                              SCALAR MAP
SYMBOL     LOCATION      SYMBOL     LOCATION      SYMBOL     LOCATION
N               BC       I               CO       IP              C4
IB              DO

                              ARRAY MAP
SYMBOL     LOCATION      SYMBOL     LOCATION      SYMBOL     LOCATION
ITO             D4

                           FORMAT STATEMENT MAP
SYMBOL     LOCATION      SYMBOL     LOCATION      SYMBOL     LOCATION
     1       264              3       268              7       26E


  *OPTIONS IN EFFECT    ID,EBCDIC,OURCE,NOLIST,NODECK,LOAD,MAP
  *OPTIONS IN EFFECT    NAME = MAIN    , LINECNT =        50
  *STATISTICS     SOURCE STATEMENTS =       20,PROGRAM SIZE =        1050
  *STATISTICS   NO DIAGNOSTICS GENERATED
```

57. ábra. Tárkivonat (részlet)

```
    0001                READ(5,1) N
    0002           1    FORMAT(I3)
    0003                READ(5,3)(ITO(I),I=1,N)
***********01)  IEY0111 UNDIMENSIONED***************
    0004           3    FORMAT(2014)
    0005                IP=0
    0006                IZ=0
    0007                IN=0
    0008                DO 2 I=1,N
    0009                IF(ITO(I))10,11,12
    0010          10    IN=IN+1
    0011                GO TO 2
    0012          11    IZ=IZ+1
    0013                GO TO 2
    0014           2    IP=IP+1
    0015           2    CONTINUE
                        $
***********01) IEY0061 DUPLICATELABEL**************
    0016                WRITE(6,7) IN,IZ,IP
```

58. ábra. Szintaktikai hibaüzenetek

azzal ezt megelőzően még valamit csinálni kell. Miért nem hajtható végre a tárgyprogram, és mit kell a tárgyprogrammal csinálni, hogy futtatható program álljon elő? E kérdésre akkor tudunk válaszolni, ha szemügyre vesszük azokat a feltételeket, amelyek egy program futtatásához szükségesek.

Fordítás, szerkesztés

E feltételek a következő gondolatmenet alapján gyűjthetők össze.

— Tudjuk, hogy a program — amikor végrehajtásra kerül — a tárban van. Az első feltétel tehát, hogy a tárban ki legyen jelölve a program által igényelt terület.

— A program működése során adatokkal foglalkozik, amelyeknek szintén a tárban kell lenniök. A második feltétel, hogy az adatok számára is legyen tárterület, s ennek pontos helye a programban is rögzítve legyen.

— A program tartalmazhat (és tartalmaz is) ún. külső hivatkozásokat. Külső hivatkozásról beszélünk, ha pl. a programban egy olyan makro, függvény vagy alprogram nevet használunk, amelynek leírása (kifejtése) nem található meg magában a programban. A végrehajtás előtt ezeket a külső hivatkozásokat „fel kell oldani", vagyis a programhoz hozzá kell szerkeszteni a külső hivatkozások kifejtését.

— Végül az összeszerkesztett programot be kell tölteni a tárba.

A tárgyprogram nem rendelkezik mindezen tulajdonságokkal:

— A fordítóprogram a tárgyprogram és a hozzá tartozó adatok számára nem tényleges tárcímeket jelöl ki, hanem csak relatív címeket. A relatív címekből egy meghatározott konstans hozzáadásával lehet abszolút címeket képezni. Ezt a konstans értéket valamelyik regiszterből lehet kiolvasni. 

— A fordítóprogram a forrásként megadott programot és alprogramokat önálló szintaktikai egységekként fordítja.

Azt a szintaktikai egységet, amelyet a fordításkor egyetlen önálló logikai egységként kezel a rendszer, a továbbiakban **szegmensek** nevezzük. A külső hivatkozások feloldása tehát nem más, mint a hivatkozott szegmensek összefűzése egyetlen logikai egységgé.

— A fordítóprogram nem tölti be a tárba a tárgyprogramot.

Ha meggondoljuk, hogy egy-egy magas szintű nyelv fordítóprogramja meghaladja a 100 Kbájtnyi tárigényt, könnyen magyarázatot találunk arra, hogy ezeket a feladatokat miért nem a fordítóprogrammal végeztetik el. Ha ui. a fordítóprogram végezné pl. a betöltést is, a futtatandó programot a saját maga által lefoglalt 100 Kbájtos területen kívül tudná csak elhelyezni. Ez azt jelentené, hogy pl. 200...300 Kbájtnyi tárigénnyel lépne fel egy olyan feladat, amely egyidőben sohasem foglal el ennyi helyet (fordításkor 100 Kbájtot, és futáskor 100...200 Kbájtot). Ez az egyik fő oka, hogy a fordítás befejezése után a tárgyprogrammal más rendszerprogramok foglalkoznak. Ezek a rendszerprogramok megoldják mindazokat a problémákat, amelyek miatt a tárgyprogram nem végrehajtható.

Azokat a rendszerprogramokat, amelyek a tárgyprogramból végrehajtható programot állítanak elő, **szerkesztőprogramoknak** nevezzük.

Kétféle szerkesztési funkciót ellátó program használatos. Az egyiket *kapcsolatszerkesztőnek*, a másikat *betöltőprogramnak* nevezik.

Mi a kapcsolatszerkesztő feladata, hogyan működik? E kérdés megválaszolásához vegyük szemügyre a kapcsolatszerkesztő működéséhez szükséges inputokat és az általa előállított outputokat.

A kapcsolatszerkesztő a következő input állományokat használja:

— *elsődleges* inputként egy (esetleg több) tárgyprogramot, amelyet magas szintű nyelvi fordító állított elő, és mágneslemezen helyezett el speciális, ún. könyvtári fájlban, röviden könyvtárban;

— *másodlagos* inputként egy automatikusan meghívásra kerülő könyvtárat, amely az elsődleges inputnak megfelelő magas szintű programozási nyelvhez tartozó, lefordított rutinokat tartalmaz. (Pl. a FORTRAN nyelven megírt programot a SYS1. FORTLIB nevű könyvtár rutinjai egészítik ki, a COBOL nyelvét a SYS1. COBLIB);

— szintén másodlagos inputként a programozó további könyvtárakat is előírhat, az elsődleges inputban ui. létezhetnek olyan hivatkozások (elsősorban alprogramokra vonatkozóan), amelyek valamilyen külön könyvtárban találhatók meg. | ● | 279 | 17 |

Outputként két állomány keletkezik:

— a *betölthető* (futtatható) *program*, amelyet a kapcsolatszerkesztő mindig egy könyvtárban helyez el, s az onnan való betöltést egy külön program végzi;

— egy *üzenetállomány*, amelyben a szerkesztés során a rendszer által generált (informatív jellegű) üzenetek és — hibás szerkesztés esetén — hibaüzenetek találhatók. | ● | 279 | 18 |

A kapcsolatszerkesztő program működése során — egyebek mellett — a következő tevékenységeket végzi el:

— feloldja az egyes szegmensekben levő külső hivatkozásokat, és ezzel „összefűzi" egyetlen logikai egységgé az egyes önálló tárgymodulokat.

A hivatkozások feloldása során a kapcsolatszerkesztő a programozó által összeállított input szegmenseket és a már korábbiakban említett könyvtárban található szegmenseket használja fel.

A szerkesztéshez az egyes szegmensek tartalmán kívül a kapcsolatszerkesztőnek *egyéb információk is* szükségesek, így pl.:

— ismernie kell a szegmens hosszát;

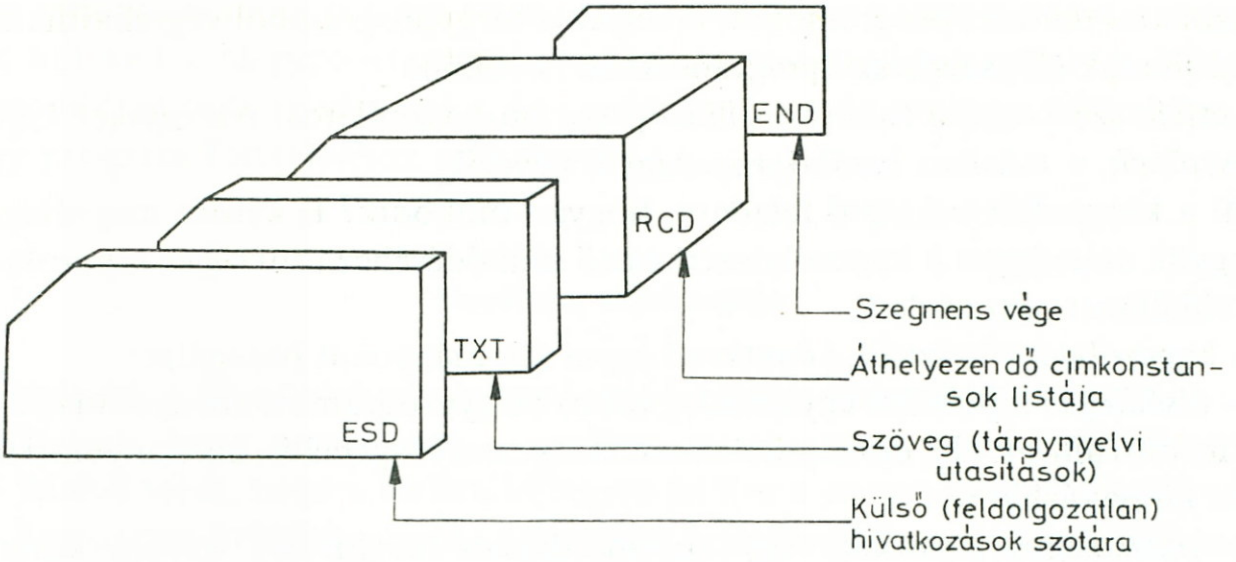
— tudnia kell, hogy melyek a szegmensben levő feloldatlan hivatkozások;

— ismernie kell azokat a címkonstansokat, amelyek megváltoznak, ha a program áthelyezésre (más címtől való betöltésre) kerül.

Honnan veszi ezeket az információkat a kapcsolatszerkesztő?

A fordítóprogram által előállított tárgyprogram tartalmazza ezeket az információkat is. Egy tárgyprogram szerkezetét az 59. ábra szemlélteti.

A kapcsolatszerkesztő a külső hivatkozások feloldásakor a címeket az első szeg-



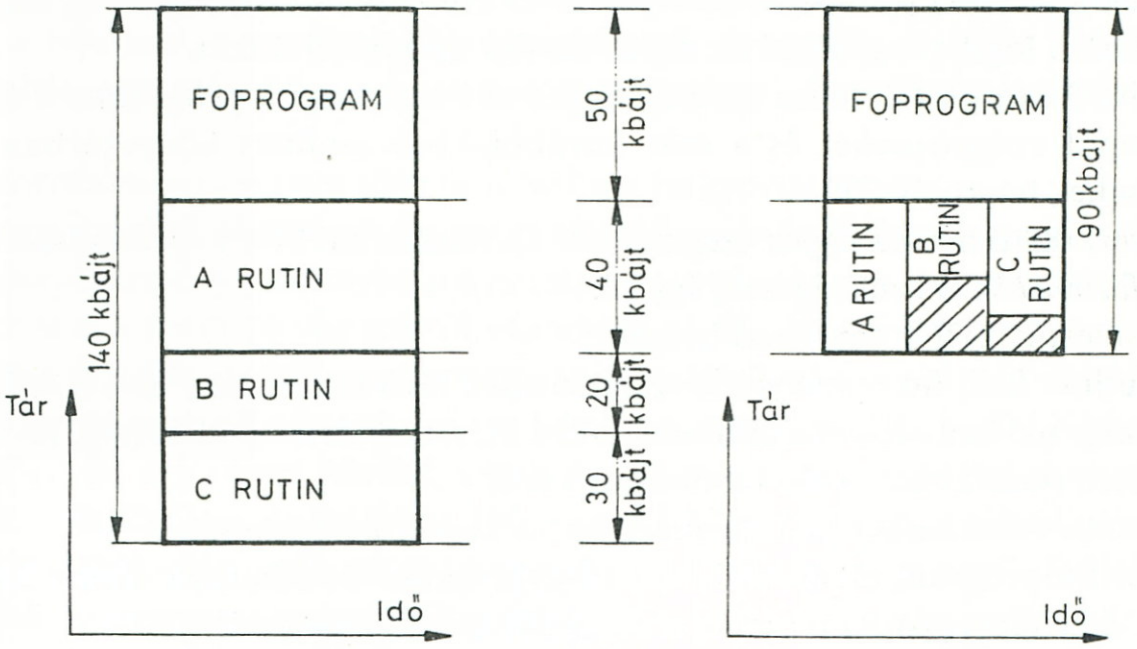
59. ábra. A tárgyprogram szerkezete

mens elejéhez viszonyítva jelöli ki. (Az első szegmens a főprogram, az a program, amelyet egyetlen más program sem hív. Mindig kell egy ilyennek lennie.)

Az eddigiekből láthattuk, hogy a kapcsolatszerkesztő a futtatandó programot általában több szegmensből állítja össze. Ez a tulajdonsága teszi lehetővé a programozó számára, hogy a programtervezés és -kivitelezés során bátran éljen a moduláris és strukturált programtervezés és programírás minden felkínált előnyével.

Nem tettünk említést a kapcsolatszerkesztő sokoldalú szolgáltatásairól, amelyekkel a programozó munkáját támogatja. Ezek közül csupán egyet emelünk ki, azt, amelyet angol szakkifejezéssel overlay-nek neveznek. A szó „átfedést" jelent, és lényegét a 60. ábra szemlélteti.

Az ábrából láthatjuk, hogy az átfedéses technika alkalmazásának eredményeképpen egy programnak az összes lefoglalt tárterülete kisebbé válhat. Hogyan valósul ez meg?



60. ábra. Felhasznált tárterület átfedés nélkül és átfedés alkalmazásával

A programnak vannak olyan részei (szegmensei), amelyeknek a teljes végrehajtási idő alatt a tárban kell lenniök. Ilyen pl. az ábra szerinti FŐPROGRAM. Más szegmensekre csak a végrehajtás bizonyos fázisaiban, bizonyos ideig van szükség. Ilyenek a példabeli A, B és C rutinok. A kapcsolatszerkesztő erre a tényre nincs figyelemmel, s az ábra bal oldalán látható tárfoglalást valósítja meg. Az átfedéses technika alkalmazásával a kapcsolatszerkesztő a tárat az ábra jobb oldali változatában osztja fel, az időben egymást követő rutinok számára ugyanazt a fizikai helyet jelöli ki. Természetesen ezt akkor tudja megvalósítani, ha a programozó pontosan leírja az átfedési struktúrát. E technika alkalmazása elsősorban akkor szükséges, ha a program méretei és a rendelkezésre bocsátható tárméret nincs összhangban egymással.

A *betöltőprogram* szűkebb lehetőségekkel rendelkezik, mint a kapcsolatszerkesztő. Tevékenysége során előállít egy futtatható programot (vagyis elvégzi az input szegmensek összeszerkesztését), és az eredményül kapott futtatható programot azonnal be is tölti a tárba. Ez azt jelenti, hogy a betöltővel szerkesztett programot, ha a programozó újra akarja futtatni, újból kell szerkeszteni. A kapcsolatszerkesztő esetében ez azért nem áll fenn, mert a kapcsolatszerkesztő által létrehozott futtatható program egy mágneslemez területen kerül elhelyezésre. Mindez azt jelenti, hogy azokat a programokat, amelyeket időről időre újra akarunk futtatni, a kapcsolatszerkesztővel kell könyvtárazni, s minden futtatást ebből a könyvtárból indítani. Ilyen feladat lehet pl. egy vállalatnál a készletekkel vagy a bérekkel kapcsolatos — havonta ismétlődő — programrendszer futtatása.

A MIKROSZÁMÍTÓGÉPEK PROGRAMNYELVEI

A személyi számítógépek megjelenésével új igény fogalmazódott meg a programírásra és a programfuttatásra vonatkozóan. A nagygépes „menetrend" több vonatkozásban nem illett bele a mikrogépek alkalmazási céljaiba. Ebből a szempontból az egyik problémát az jelentette, hogy hosszadalmas folyamat a program fordítása, a hibák javítása, újrafordítása, majd szerkesztése, futtatása. Ráadásul a tömegfogyasztásra szánt személyi számítógépek nem is rendelkeznek olyan olcsó kiírószerkezettel, amelyen a forrásprogramot meg lehetne jelentetni. Ha képernyőn meg is lehet jelentetni a fordításból adódó hibalistát, a javítást nem ezen kell elvégezni, hanem a forráslistában. A két lista nem tud egyidejűleg megjelenni a képernyőn.

Mindezen okok miatt a személyi számítógépek egyszerűbb fajtáinál más programozhatósági lehetőséget igényelnek a felhasználók. Ezeknek a lehetőségeknek a fontosabb vonásai a következők:

— a programot egy erre a célra jól használható nyelven kell megírni;

— az elkészült program azonnal futtatható, és mindaddig fut is, amíg hibás utasításba nem ütközik;

— hibás utasítás előfordulása esetén kiírja ennek a sorszámát;

— a hibás sorszámú utasítást ki kell javítani (a hiba okára vagy valószínű okára utaló jelzést a programozó megkapja);

— mindaddig újra kell próbálkozni, amíg a program végül is le nem fut.

Ezzel a módszerrel még akkor is lehet dolgozni, ha a személyi számítógéphez nincs képernyő, csak egy egysoros folyékony kristályos megjelenítő. Milyen szoftver háttér kell az ilyen típusú munkához? Mindenekelőtt egy olyan programozási nyelv, amely ezt a módszert jól támogatja, könnyen elsajátítható, valamint ehhez a nyelvhez kapcsolódóan egy olyan lehetőség, amely a fordítási és végrehajtási funkciót egy menetben látja el. E feltételeknek eleget tevő programozási nyelvet nem kellett kitalálni. Még 1965-ben egy amerikai egyetemen létrehozták a BASIC (bézik) programozási nyelvet, amelyet elsősorban olyan helyeken használtak, ahol a felhasználók terminálon keresztül vették igénybe a számítógép szolgáltatásait. | ● | 279 | 19 |

A BASIC jól használható azok számára, akik különösebb számítástechnikai előképzettség nélkül akarnak programokat készíteni. Mindössze azt kellett kitalálni, hogy miképp biztosítható a külön fordítás nélküli programvégrehajtás. Így született meg a BASIC interpreter változata.

A *BASIC interpreter* egy olyan program, amely a számítógép csak olvasásra használható (ún. ROM) tárából egy parancs segítségével vehető elő, és fogadja a program utasításait, majd hozzálát a program végrehajtásához.

Egy személyi számítógép BASIC-ben való programozásához egyrészt ismerni kell az adott gépre érvényes BASIC nyelvi változatot, másrészt azokat a BASIC *parancsokat*, amelyekkel a programozó közölheti kívánságait a géppel. A program utasításait billentyűzetről kell bevinni, minden utasítás előtt egy sorszámot alkalmazva. Ügyesen választott sorszámozással (pl. 10-től kezdve tízes lépésközzel) biztosítható, hogy a kifelejtett utasítások utólag is bevihetők legyenek egy közbülső sorszámmal. A gép a programot mindig növekvő sorszám szerint értelmezi. A program utolsó utasítása **END**. Ez után lehet parancsokat adni a gépnek. A gépnek szóló parancsok különbözők lehetnek. Néhány parancs a HT 1080Z számítógép BASIC változatából:

RUN — futtatni a programot;

LIST — kilistázni a munkaterületen levő programot (a lista képernyőn jelenik meg);

CSAVE — a munkaterületen levő program tárolása (feltéve, ha van valamilyen mágneses háttértár);

CLOAD — a parancsban megadott program betöltése háttértárról a munkaterületre.

Bár a BASIC-re is létezik szabvány előírás, a helyzet mégis az, hogy nagyon sokféle BASIC változat (inplementáció) van forgalomban. Talán idő kérdése, hogy más nyelvekhez hasonlóan, nagyobb egységesítésre sor kerüljön.

Miközben a BASIC segítségével a személyi számítógépek jól programozható, emberközelségű eszközökké „szelídültek", megjelentek a piacon a professzionális személyi számítógépnek nevezett berendezések. Ezek központi tárméretükkel, nyomtatási lehetőségükkel és a háttértáruk méretével nagyon komoly teljesítményre alkalmas

(és lényegesen drágább) készülékek. Sok fajtájuknál megjelentek a nagygépeken használt nyelvek e célra átdolgozott fordítóprogramjai. Így elsősorban a FORTRAN és a PASCAL. Ezek a gépek tehát — miközben egy íróasztalon elférnek — nagyszámítógépek módjára programozhatók, az ott megszokott és bevált programozási nyelveken.

A BASIC nyelv alapelemei

Említettük már, hogy a BASIC nyelvnek sokféle változata van forgalomban. Az itt következő részben néhány fontosabb elemét mutatjuk be a nyelvnek. Ezek segítségével a korábban blokkdiagramként elkészített algoritmusok egy részét BASIC nyelvű program formájában is elkészíthetjük. Sem mód, sem lehetőség nincs arra, hogy e nyelv további lehetőségeit is tárgyaljuk.

Olyan esetekben, ahol a különböző BASIC változatok eltérnek egymástól, a HT 1080Z iskolaszámítógépen létező változatokra támaszkodunk.

A nyelv alapelemei azok a betűk, számjegyek és speciális jelek, amelyek a programozáshoz használhatók. Az iskolaszámítógép esetében ezek az alapelemek az ASCII kódtáblázat 32—95 sorszámozott karakterei, vagyis az angol abc nagybetűi, a tízes számrendszer számjegyei és 28 speciális jel.

A konstansok lehetnek egész, valós, duplapontos és karakter típusúak. A karakter típusú változóban max. 255 karakter hosszú szöveg vihető be.

Példa konstansokra:

7274 —45 egész típusú (max. 32767),

32,44 —0,31 valós típusú (max. 6 jegyű),

3,141592653589 dupla pontos (max. 16 jegyű),

„EZ EGY BASIC SZOVEG" karakter típusú.

A változók betűvel kezdődő nevek, amelyek tetszőleges hosszúságúak lehetnek, de a rendszer csak az első két karaktert kezeli. Így a különböző változóneveknél fontos, hogy az első két karakterük ne egyezzen meg.

A változók a konstansokhoz hasonlóan lehetnek egész, valós, duplapontos és karakter típusúak. Egy változó adott típusra való beállítása típusdeklarálással vagy megfelelő névválasztással lehetséges. A típusdeklaráló utasítások:

DEFINT az itt felsorolt változók egész típusúak lesznek,

DEFSNG az itt felsorolt változók valós típusúak lesznek,

DEFDBL az itt felsorolt változók duplapontos típusúak lesznek,

DEFSTR az itt felsorolt változók karakter típusúak lesznek.

A változónév második karakterét meg lehet adni úgy, hogy ezzel típust deklarálunk az előbbi utasítások használata nélkül is: % egész, ! valós, # duplapontos, $ karakter típusú változót definiál.

Pl. A% egész típusú lesz, B# duplapontos lesz stb.

Az egész, valós és duplapontos változókkal aritmetikai műveletek végezhetők. Így jutunk aritmetikai kifejezéshez.

A következő aritmetikai műveletek léteznek:

összeadás	+
kivonás	—
szorzás	* (csillag)
osztás	/
hatványozás	[

Ezek a műveletek a következő sorrendben kerülnek végrehajtásra: hatványozás, szorzás, osztás, összeadás, kivonás.

Ha több azonos (vagy azonos helyre sorolt) művelet fordul elő, ezeket balról jobbra való haladás előfordulási sorrendjében értékeli ki.

Aritmetikai kifejezések között relációkat értelmezhetünk. A következő relációs műveletek definiáltak a BASIC-ben:

kisebb	<
kisebb egyenlő (nem nagyobb)	<=
egyenlő	=
nem egyenlő	<>
nagyobb egyenlő (nem kisebb)	>=
nagyobb	>

A relációjelekkel összekapcsolt aritmetikai kifejezések logikai értéket adnak. Pl.:

(A+BB)*ALFA < BETA—2

(ez adott változó értékek mellett vagy igaz, vagy hamis!)

Az előbbi módon létrejött logikai értékekre értelmezve vannak a következő logikai műveletek:

negáció	NOT
konjunkció	AND
diszjunkció	OR

A gép egy olyan kifejezést, amelyben aritmetikai műveletek, relációk és logikai műveletek egyaránt előfordulnak, a következő sorrendben értékel ki:

— először az aritmetikai műveleteket a már megismert sorrendben;

— másodiknak a relációkat balról jobbra az előfordulás sorrendjében;

— harmadiknak a logikai műveleteket:

NOT AND OR.

A BASIC nyelv *utasításszavai* — mint minden olyan nyelvé, amely a számítógéppel való „párbeszédes" munkavégzést lehetővé teszi — két jól elkülönülő csoportba tartoznak:

A *parancsok*, amelyek segítségével a programozó a gép funkcióit és szolgáltatásait befolyásolja. Ilyen parancsszavakkal lehet pl. a tár tartalmát megjeleníteni a képernyőn (LIST), a programvégrehajtást elindítani (RUN), a háttértárakkal adatátvitelt bonyolítani (CSAVE, CLOAD) stb. Ezekkel a parancsokkal nem foglalkozunk.

Az *utasítások*, amelyek segítségével mód nyílik arra, hogy programot készítsünk.

Az utasítások általános alakja:

sorszám utasítás,

ahol a sorszám egy egész számot jelent. Az utasítások a sorszámok növekvő sorrendjében (és nem a bevitel sorrendjében) kerülnek végrehajtásra.

Egyszerűbb programok készítéséhez szükséges utasítások:

Értékadás:

LET változónév = kifejezés a változónév és a kifejezés típusa megegyező legyen

Pl.:

```
LET A% = 22
LET B! = 3,59
LET P# = 3,1425926535
LET R$ = "A SZOVEG IDEZOJELEK KOZOTT SZEREPELJEN"
```

Vezérlésátadás:

GO TO sorszám — feltétel nélküli vezérlésátadás a megadott sorszámú utasításra

IF kifejezés **THEN** művelet **ELSE** művelet — feltételes vezérlésátadás:

ha a reláció kifejezés igaz, a THEN után levő műveletet hajtja végre, egyébként az ELSE után levőt.

(Az IF utasításnak lehetnek egyszerűbb alakjai, ezek közül csak azt említjük meg, hogy az ELSE ág elmaradhat.) Pl.:

```
IF A = B THEN GOTO 60 ELSE GOTO 20
```

Ciklusszervezés:

FOR számláló-kezdőértéke **TO** végérték **STEP** lépésköz — a számláló induláskor felveszi a kezdőértéket, majd minden ciklusmenet után megváltozik a lépésköz értékével. Addig történik a ciklusmag ismétlése, amíg a számláló el nem éri vagy túl nem haladja a végértéket. A lépésköz lehet negatív szám is. Ha nem adtuk meg a lépésközt, akkor az alapértelmezés szerint az 1 lesz.

NEXT számláló — a ciklusmag eddig az utasításig tart. Pl.:

```
FOR I = 1 TO 10 STEP 3
        utasítások
NEXT I
```

Tömbök deklarálása:

DIM tömbnév (méret 1, méret 2, . . . méret n) A tömb dimenziószáma nincs korlátozva.

Néhány további utasítás:

GOSUB sorszám — egy alprogram a sorszámmal megadott soron kezdődik, ez az utasítás ide adja át a vezérlést.

RETURN — az alprogramból a vezérlés az alprogramot hívó GOSUB utasítás mögé adja vissza.

END — a program végét jelző utasítás

REM — ennek az utasításnak a sorába magyarázó szöveg írható.

Input-output utasítások:

PRINT lista — a listában szereplő változók tartalma a képernyőn megjelenítésre kerül. A két " közé tett szöveg is megjelenik.

INPUT lista — a listában szereplő változókra értéket vár a billentyűzeten keresztül.

Ezeknek az utasításoknak az ismeretében tekintsünk meg néhány példát (csak magát a programot mutatjuk be).

A bemutatásra kerülő programokhoz tartozó algoritmusokkal már a korábbi fejezetek végén, a megoldandó feladatok között találkoztunk.

1. Egy háromszög három oldalának mérőszáma A, B, C. Egy másik háromszög három oldalának mérőszáma D, E, F. Egybevágó-e a két háromszög, ha a megfelelő oldalak megadási sorrendje megegyezik?

```
10 REM HAROMSZOGNEK EGYBEVAGOSAGA
20 INPUT A, B, C
30 INPUT D, E, F
40 IF A=D AND B=E AND C=F THEN IGEN=1 ELSE IGEN=—1
50 PRINT IGEN
60 END
```

2. Mértani közép számítása. Számkettesek alkotnak egy-egy rekordot. Számítsuk ki és nyomtassuk a számkettesek mértani közepét. Az inputban az utolsó számkettest egy dupla 0-t tartalmazó zárórekord követi.

```
10 REM MERTANI KOZEP SZAMITASA
20 PRINT „KEREK KET ADATOT"
30 INPUT A, B
40 IF A=0 AND B=0 THEN GO TO 90
50 REM A FAJLNAK NINCS VEGE
60 Q=(A*B)[0,5
70 PRINT „A MERTANI KOZEP", Q
80 GO TO 20
90 PRINT „VEGE"
100 END
```

3. Készítsünk algoritmust, amely egy n × n méretű kétdimenziós tömbről eldönti, hogy szimmetrikus-e. A program inputja a tömb és annak mérete, outputja vagy a SZIMMETRIKUS vagy pedig a NEM SZIMMETRIKUS szöveg.

```
10 REM SZIMMETRIA VIZSGALAT
20 DIM T (100, 100)
30 INPUT N
40 FOR I=1 TO N
50 FOR J=1 TO N
60 INPUT T (I, J)
70 NEXT J
80 NEXT I
90 L$ = „SZIMMETRIKUS"
```

```
100 REM A CIKLUS INDEXEI CSAK AZ ALSÓ HAROMSZÓG RÉSZBEN
MOZOGNAK
110 FOR I=2 TO N
120 FOR J=1 TO I—1
130 IF T (I, J <>T (J, I) L$ = „NEM SZIMMETRIKUS"
140 IF L$ = „NEM SZIMMETRIKUS" THEN GO TO 170
150 NEXT J
160 NEXT I
170 PRINT L$
180 END
```

Kérdések

1. Mit értünk a szoftver fogalmán?
2. Mit nevezünk gépi kódnak?
3. Milyen korlátai vannak a gépi kódban való programozásnak?
4. Mi tette szükségessé az assembly szintű programozás bevezetését?
5. Mi az assembler?
6. Mi jellemzi a magas szintű programozási nyelveket?
7. Melyek a magas szintű programozási nyelvek alapelemei?
8. Mi a programfordítás, és mikor szükséges?
9. Mi a fordítóprogram feladata?
10. Miért nem futtatható a fordítóprogram outputja?
11. Mi a kapcsolatszerkesztő program feladata?
12. Mit nevezünk elsődleges és másodlagos inputnak a szerkesztési fázisban?
13. Mi történik az összeszerkesztett programmal?
14. Mi jellemzi az interpretert a fordítóprogrammal szemben?
15. Melyek a BASIC nyelv parancsainak funkciói?
16. Melyek a BASIC nyelv utasításainak funkciói?

Feladatok

Készítsd el a BASIC programját a következő, korábban már blokkdiagrammal megoldott feladatoknak:

3. fejezet utáni 1. feladat
3. feladat
9. feladat

5. fejezet utáni 3. feladat
4. feladat
5. feladat
6. feladat

VII. FEJEZET

Input-output egységek, háttértárak

A számítógépek felépítésének és működésének megismerése során egy nagyon fontos és kritikus ponthoz jutottunk el. Eddigi ismereteinket a következő módon foglalhatjuk össze:

— a számítógépet használni kívánó programozó egy (a megoldandó feladat szempontjából kiválasztott) programozási nyelv segítségével, algoritmizálási tudásának felhasználásával elkészíti a felmerült probléma megoldásához szükséges számítógépes programot;

— ezt a programot az adott nyelv fordítóprogramja tárgyprogrammá fordítja, majd a kapcsolatszerkesztő program futtatható programot állít elő;

— a futtatható program a központi tárba helyezve a központi egységet működésbe hozza, s az ott levő regiszterek és áramkörök „realizálják" a programot.

Mindez azonban feltételezi, hogy:

— a forrásnyelven megírt program valamilyen úton-módon *bekerül* a számítógépbe;

— a fordítóprogram, amelyre csak a fordítás ideje alatt van szükség, *valahonnan bekerül* a központi tárba;

— a tárgyprogram és a betölthető program, *valahol* ideiglenesen tárolásra kerül a fordítás, ill. a szerkesztés után;

— a futtatható program *valahonnan* kapja a feldolgozandó adatmennyiséget (amely akár több tízezer rekord is lehet);

— a feldolgozás eredményei *valahová* kikerülnek a számítógépes környezetből, és az ember számára olvasható, felhasználható alakot öltenek.

Az alapvető probléma tehát, amelyet meg akarunk oldani, **milyen a központi tár és a központi egység kapcsolata, kommunikációja a külvilággal.** Nagyon fontosnak és egyben kritikusnak neveztük ezt a kérdést. Fontosságát nem szükséges részletezni, elég, ha arra gondolunk, hogy pl. mit érne egy nagyon okos és nagyon tanult ember rengeteg gondolata és ötlete, ha egy toronyszobában ülne egyedül egész nap, senkivel nem beszélne, gondolatait nem írná le.

De miért neveztük a kommunikáció kérdését kritikusnak is egyben? Talán az előbbi „toronylakóval" ezt is érzékeltetni lehet. Tegyük fel, hogy ennek az embernek minden nap annyi közölnivalója van a külvilág számára, ami száz oldalnyi papirost

igényel. Naponta viszont csak tíz oldalnyi papírt képes teleírni. Minél nagyobb a veszteség az elmaradt mondanivaló miatt, annál kritikusabb a helyzet.

A számítógépek külvilággal való kapcsolattartásában a kritikus elem tehát a központi egység és a külvilággal kapcsolatot tartó egységek sebessége közötti különbségben van. Másképpen fogalmazva: a számítógépes feldolgozások „szűk keresztmetszete" a kommunikáció sebessége. S mint ahogyan egy acéllánc ereje megegyezik a láncban levő leggyengébb láncszem erejével, ugyanúgy egy rendszer átbocsátóképességét a leglassúbb rész áteresztőképessége határozza meg.

Megválaszolandó kérdéseink tehát a következők:

— **Milyen eszközök** segítségével tud a központi egység és a központi tár kapcsolatot tartani a külvilággal?

— **Milyen célokat** szolgálnak ezek az eszközök, mi a funkciójuk?

— **Hogyan működnek** ezek az eszközök?

Ez utóbbi kérdés azért is fontos, mert a rosszul megszervezett és végrehajtott feldolgozások semmit sem érnek. Ha viszont nem ismerjük kellően azokat az eszközöket, amelyek az ember és a számítógép információcseréjét bonyolítják, nagyon könnyen kerülhetünk olyan helyzetbe, hogy a jó elgondolásainkat rosszul valósítjuk meg.

A számítógép központi egysége a környezettel a **perifériákon** keresztül tartja a kapcsolatot. Már az ötödik fejezetben megismerkedtünk a csatornákkal. Ezek a csatornák biztosítják a fizikai kapcsolatot a központi egység és a perifériák között. Szerepük a sebességkülönbségből adódó problémák áthidalása.

De melyek is azok a perifériák?

Azokat a készülékeket, amelyeket a számítógép központi egységéhez kapcsolnak a kétirányú adatáramlás biztosítása érdekében, periférikus egységeknek, vagy röviden perifériáknak nevezzük.

Milyen elvek szerint csoportosíthatjuk a perifériákat?

A perifériákat csoportosíthatjuk az adatáramlás iránya szerint. Ebből a szempontból beszélhetünk bemeneti (input) és kimeneti (output) egységekről.

Input egységek pl.: *kártyaolvasók, terminálok, bizonylatolvasók.*

Output egységek pl.: *sornyomtatók, grafikus megjelenítők, mikrofilmes megjelenítők.*

Input/output — vagyis mindkét irányú adatmozgatást biztosító berendezések: *mágnesszalag egységek, mágneslemez egységek.*

Egy másik — a központi egység szempontjából történő — csoportosítás az adatátvitel sebessége szerint lehetséges. Ebben az értelemben szokás megkülönböztetni: *lassú perifériákat* (kártyaolvasó, sornyomtató stb.), *gyors perifériákat* (mágnesszalag, mágneslemez vagyis a háttértárak).

Ez utóbbi megkülönböztetés a központi egység számára oly módon realizálódik, hogy a lassú perifériák a multiplex csatornán, míg a gyors perifériák a szelektor csatornán keresztül kapcsolódnak a központi egységhez.

A mikroszámítógépek megjelenésével új típusú perifériák is megjelentek, ilyenek pl. a kazetta, a hajlékony lemez.

A következőkben először az input, majd az output egységekkel ismerkedünk meg. Ezt követően a háttértárak által nyújtott lehetőségeket mutatjuk be. Végül szót ejtünk a mikrogépekkel megjelent új perifériákról.

INPUT-OUTPUT EGYSÉGEK

A perifériák közül először a környezettel közvetlen kapcsolatot biztosító perifériákról lesz szó.

Azoknál a számítógépeknél, amelyeknél nincs lehetőség arra, hogy a programozók terminálóknál ülve végezzék munkájukat, a programbevitel fontos és gyakorlatilag egyetlen eszköze a *kártyaolvasó berendezés*. ▲ | 282 | 1

A kártyaolvasó

Feladata a lyukkártyákra rögzített információk (programok, adatok, vezérlő utasítások) érzékelése, és kódátalakítás után a központi egység felé való továbbítása.

A kártyaolvasó berendezés főbb részei:

— kártyaadagoló,
— továbbító mechanizmus,
— olvasóállomások,
— kártyalerakó,
— puffertároló.

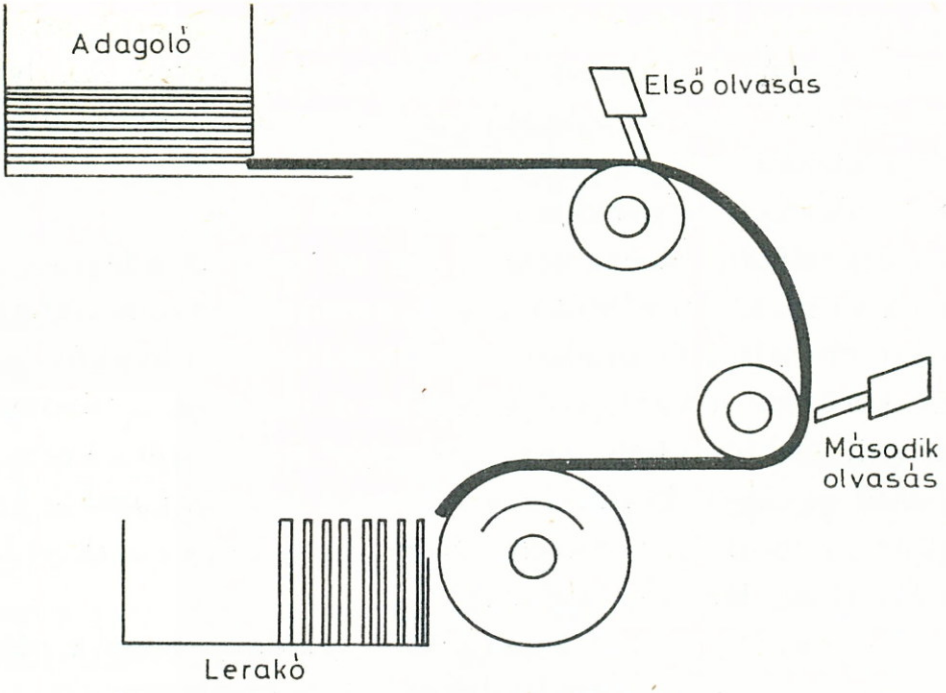
A beolvasni kívánt kártyákat (kártyakötegeket) a kártyaadagolóban helyezi el az operátor. A kártyák fizikai sorrendje megfelel a megkívánt olvasási sorrendnek.

A továbbító mechanizmus a soron következő kártyát az első olvasóállomáshoz továbbítja, amely — modern berendezésnél fotoelektromos úton — a lyukkombinációkat érzékeli. Az érzékelés általában oszlopként történik.

Az érzékelt lyukkombinációhoz — amennyiben az nem egy tiltott kombináció — tartozik egy EBCDIC kódbeli 8 bites karakterkép, amelyet a kártyaolvasó elhelyez egy 80 bájt méretű átmeneti, ún. puffertárolóban. A 80 oszlop végigolvasásával ez a puffertár megtelik. A továbbhaladó kártyát egy újabb olvasóállomáson ismételten leolvassa a berendezés, és az olvasási műveletet csak akkor fogadja el, ha a két olvasás eredménye megegyezett. Ez esetben a puffertároló tartalma továbbítható a központi egységhez. (Ez a továbbítási funkció már nem a kártyaolvasó berendezés feladata.) ● | 283 | 2

A leolvasott kártyák a kártyalerakóba kerülnek. Az olvasás sebessége 600...1200 kártya percenként. Az elmondottakat a 61. ábra szemlélteti.

A kártyaolvasó berendezés működését a berendezésen elhelyezkedő lámpákkal és nyomógombokkal lehet nyomon követni, ill. befolyásolni. (Itt található a bekapcsoló és kikapcsoló gomb, egyes típusoknál a bennmaradt kártya kiürítését biztosító gomb.



61. ábra. A kártyaolvasó működési elve

A lámpák egy része a berendezés állapotára utalnak, pl.: a bekapcsolt állapotot jelzi, más lámpák a fellépő hibákat jelzik, pl. ha beragad egy kártya. Ilyenkor az operátornak kell közbeavatkoznia.) A különböző típusú kártyaolvasóknál különböző módon vannak ezek a jelző és beavatkozást biztosító funkciók megvalósítva. Egyes kártyaolvasó berendezések összeépítettek olyan kártyalyukasztóval, amelyen keresztül — a programban előírt módon — lyukasztott kártya outputot lehet előállítani. Ezek a lyukasztó berendezések a mágneses háttértárolók megjelenésével sokat veszítettek jelentőségükből, de néha még ma is használatosak. Elsősorban olyan programoknál, amelyeket sok évvel ezelőtt készítettek, de még ma is használatban vannak és a programozó — az akkori legegyszerűbb lehetőségként — lyukasztott outputot írt elő.

A program futásakor keletkezett kártyára lyukasztott output állomány a következő futás inputjának részét képezi. Ezzel az új programfutáskor a számítógép korábbi eseményeket, döntéseket is figyelembe tud venni. | ● | 284 | 3 |

Optikai olvasóberendezések

A korábbiakban | ▲ | 282 | 4 | már volt szó arról, hogy azoknál a programoknál, amelyeknek nagytömegű adatigényük van, a másodlagos adathordozók (lyukkártyák) előállítása sok munkát és energiát igénylő feladat. Ráadásul minél nagyobb tömegű adathalmaz rögzítéséről van szó, annál nagyobb (és természetesen költségesebb), valamint időigényesebb munkafolyamat az input adattömeg lyukkártyán való előállítása. Több adatrögzítő soknapos munkájának eredményét a számítógép egy óránál rövidebb idő alatt feldolgozza és az outputokat kinyomtatja. Ennek az ellentmondásnak a feloldására sokféle kísérlet volt. Ezek közé tartozik annak az ötletnek a megvalósítása, amely így fogalmazható meg: legyen olyan adathordozó, amelyet az ember

és a számítógép egyaránt képes elolvasni. Az ilyen bizonylatok számítógéppel való olvasásához természetesen különleges perifériára van szükség. Az ilyen feladatot ellátó perifériákat *optikai úton való jelfelismerő berendezéseknek*, angol elnevezésük alapján *OCR* berendezéseknek nevezik. [★ | 282 | 5]

Az OCR berendezések alapvetően négy részből állnak.

— *A bizonylattovábbító mechanizmus:* ide tartozik az adagoló, ahol az elolvasandó bizonylatok vannak, a továbbítógörgők, amelyek a nyomdákban használt technikához hasonlóan elviszik a bizonylatot a leolvasóhoz és a lerakók, ahová az elolvasott bizonylatok kerülnek. (Általában külön lerakóba azok a bizonylatok, amelyek olvasása sikeres volt, és külön helyre azok, amelyeké nem volt sikeres.)

— *Képfelbontó egység:* a korábbi képfelbontók egy-egy karakterképet több lépésben „tapogattak le", a modernebb berendezésekben egy-egy karakter képének letapogatása fotódiódákkal egy lépésben történik.

— *Felismerő egység:* ennek az egységnek a feladata, hogy a már érzékelt jelet összevesse a szótárban levő lehetséges jelalakokkal, és kiválassza azt, amellyel az olvasott megegyezik.

— *Kimeneti egység:* az azonosított jelhez tartozó belső kódot képzi és továbbítja. A továbbított belső kód pl. mágnesszalagon kerül elhelyezésre.

Az OCR olvasók használatának igen sok szervezési és előkészítési feltétele van, így elsősorban azoknak a személyeknek a kiképzése, akik a bizonylatokat kitöltik. Meg kell szervezni ugyanakkor azoknak a bizonylatoknak a felvitelét, amelyeket az optikai olvasó nem tudott elolvasni. Ezeket a bizonylatokat általában lyukkártyára lyukasztják és úgy olvastatják be, vagy terminálokról közvetlenül gépelik be. Akár így, akár úgy, mindenképpen külön munkamenet szükséges. A kísérletek és alkalmazások azt bizonyították, hogy az optikai úton való olvastatás csak akkor kifizetődő, ha a számítógépbe bevinni szánt adattömeg nagyság meghaladja az 50 000 lyukkártyányi méretet. Éppen ezért rendszeres (kizárólagos) alkalmazása nem is gazdaságos, hiszen még olyan helyeken is, ahol nagytömegű adatokat visznek fel, nem biztosítható, hogy minden feldolgozás ekkora adattömegű legyen. [● | 284 | 6]

Hazánkban 1972-ben a Központi Statisztikai Hivatal számítóközpontjában alkalmazták először az optikai úton olvasó berendezést. A hetvenes évek elején világszerte nagy várakozással tekintettek az adatbevitelnek ezen formája felé. Az azóta eltelt idő ezt a nagy várakozást nem teljes mértékben igazolta. Igen sok helyen inkább a közvetlenül mágnesszalagra való adatrögzítést valósították meg, annak kisebb költségigénye és biztonságosabb volta miatt. [● | 284 | 7]

Nyomtató berendezések

A feldolgozási eredmények hagyományos megjelenítési formája a nyomtatott kép. [● | 284 | 8] A nyomtatott kép előállításának legnagyobb problémája a megfelelő sebesség biztosítása. Az teljesen nyilvánvaló, hogy a hagyományos írógépszerű nyom-

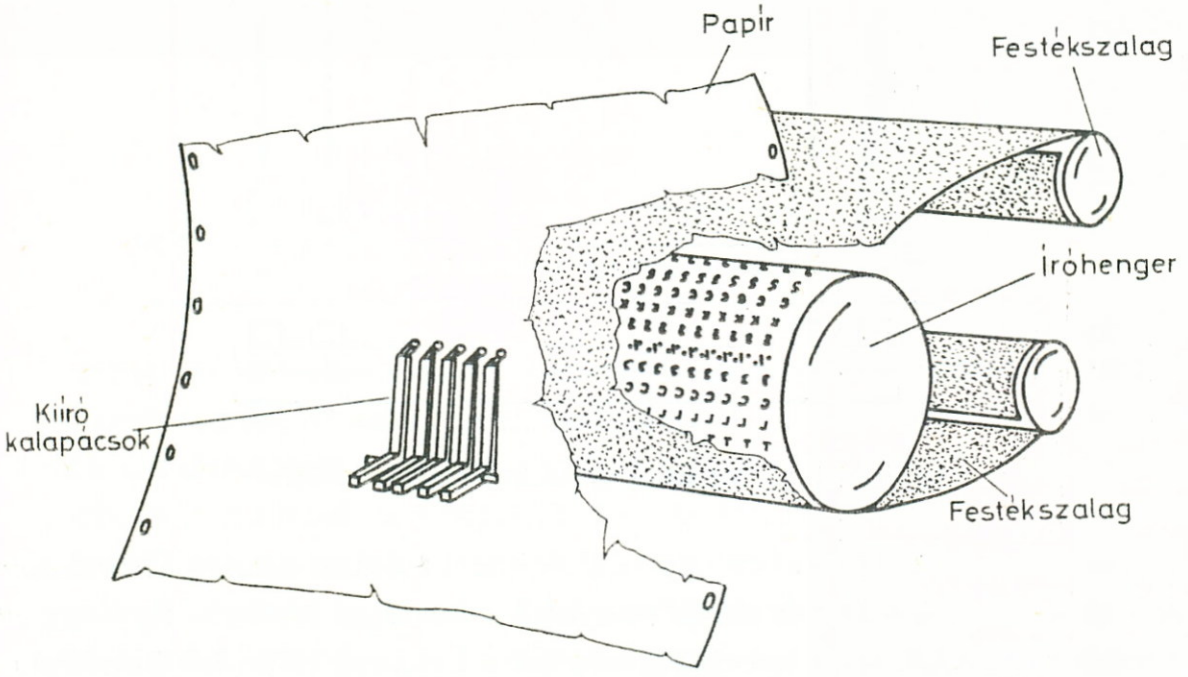
tatás, amely percenként néhány száz karakternél nem tud többet nyomtatni, nem lehet megoldás. Ezért a nyomtatóberendezések egyik típusa nem karaktereket, hanem sorokat nyomtat egyszerre, ami jelentős gyorsítást jelent.

Attól függően, hogy e sornyomtatást hogyan valósítják meg, megkülönböztetünk:

— íróhengeres,
— írórudas,
— íróláncos

sornyomtató berendezéseket.

A teljes sor nyomtatásának előfeltétele egy puffertár, amelyben nyomtatás előtt a teljes sor (fizikai rekord) nyomtatandó képe rendelkezésre áll. Ennek a puffernek a megléte ugyanakkor a központi egységből a sornyomtatóra való adattovábbítást is felgyorsítja, hiszen egy teljes fizikai rekordot egyszerre lehet átküldeni. A pufferban minden nyomtatandó karakter EBCDIC kódban van. A „sor" nyomtatás logikája jól nyomon kísérhető az *íróhengeres nyomtatás* elvének áttekintésekor, amelyet a 62. ábra szemléltet.



62. ábra. Az íróhengeres nyomtató működési elve

Ha az ábrán látható henger palástját gondolatban kiterítenénk, egy olyan lapot kapnánk, amelyben n sor található (n értéke a jelkészletben levő karakterek számával egyezik meg. A sornyomtatók jelkészlete általában 64 jelből áll.) Egy-egy karakter annyiszor ismétlődik egy sorban, ahány pozícióra a sornyomtatón nyomtatni lehet. (Ez az érték legfeljebb 160.) A henger körbe forog. Egyik érintő síkjában halad el a papír, amelyre a nyomtatás történik. A papírlap és a henger között található a festékszalag.

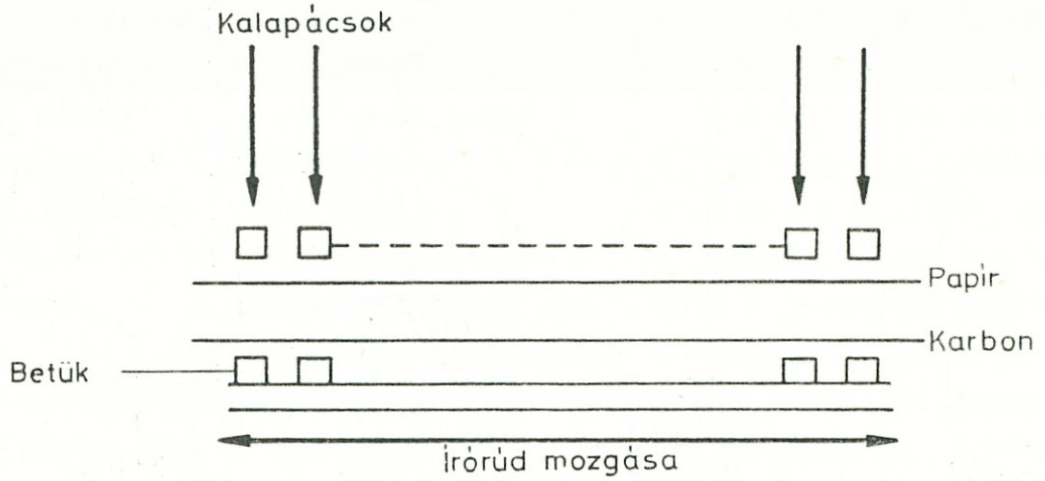
Minden egyes nyomtatási pozícióhoz egy-egy „kalapács" tartozik, amelyek a papírnak a hengerrel ellentétes oldalán helyezkednek el.

A sor nyomtatása úgy történik, hogy a mozgó hengerre a kalapácsok — vezérlőimpulzusok hatására — mindig akkor ütnek rá, amikor a nyomtatni kívánt betű képe elhalad a kalapácsok alatt. A sort tehát nem egyszerre írja ki, hanem szakaszosan. Először pl. az összes A betűt, majd a henger forgásával a többit. Legkésőbb a henger egyszeri körülforgásakor a nyomtatandó sor előállt.

Ha meggondoljuk, hogy az ilyen rendszerű eljárással percenként 300...1400 sort lehet nyomtatni, akkor nyilvánvaló, hogy ezt a szakaszosságot sem füllel, sem szemmel nem lehet érzékelni.

Az *írórudas sornyomtatóknál* a jelkészlet egy ún. írórúdon található. Egy-egy jel képe többször is előfordul. A rúd a papír mozgására merőleges irányban mozog, s az írókalapácsok mindig akkor nyomják rá a rudat a papírra, amikor a kívánt betűkép alatta elhalad.

Az egyes karakterek „fésűfog"-szerűen vannak a rúdhoz erősítve, így azok egymástól függetlenül nyomódnak a papírra. A rúd és a papír között ebben az esetben is festékszalag található. Az írórudas nyomtatás elvét a 63. ábra szemlélteti.



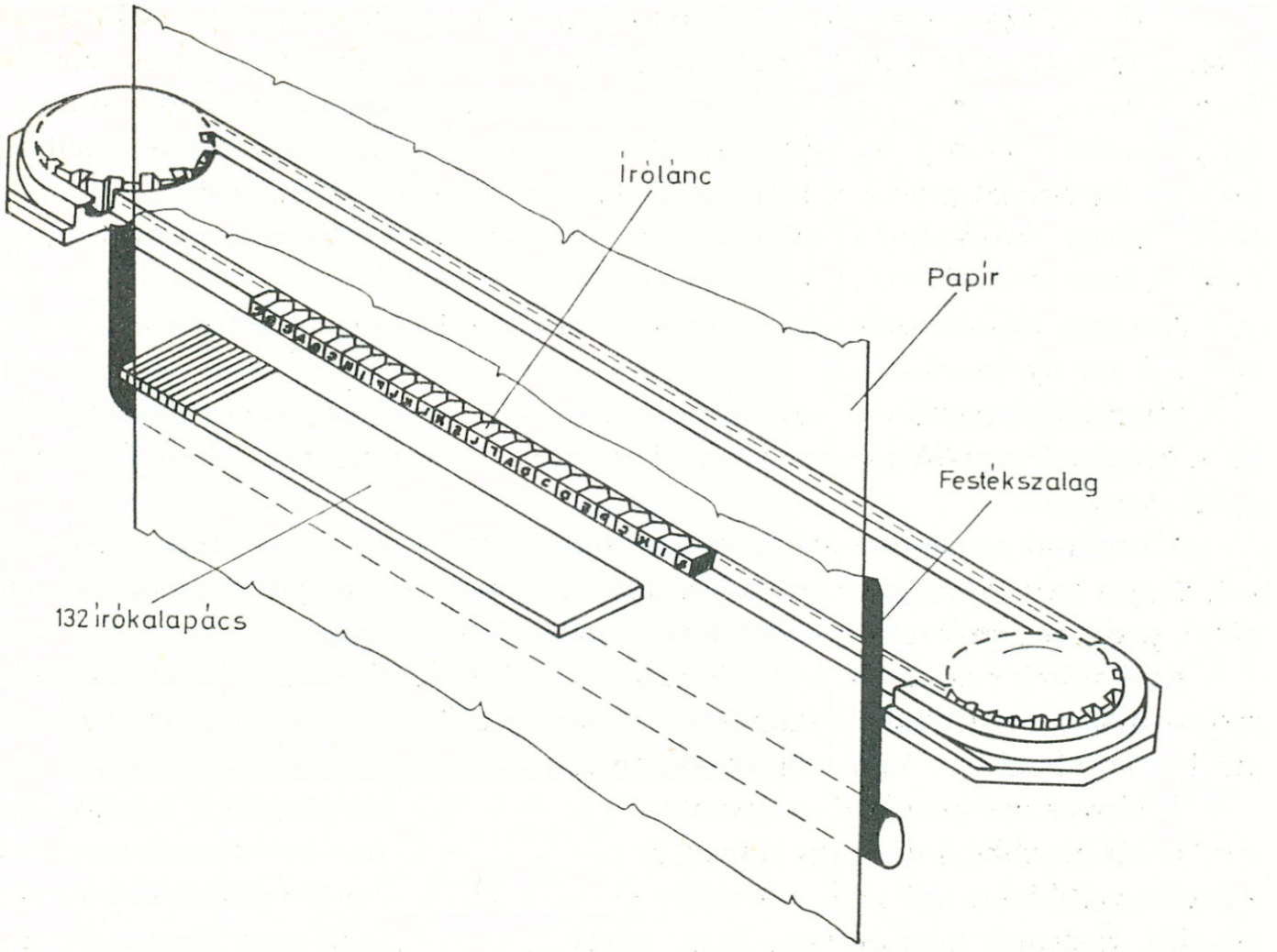
63. ábra. Az írórudas nyomtató működési elve

Az *íróláncos nyomtató* esetében egy végtelenített írólánc mozog körbe, egy irányban. Az egyirányú mozgás az írórúdnál nagyobb sebességet biztosít. Egy-egy jel a láncon többször is előfordul, így a nyomtatandó sor a lánc egy teljes körülfordulása előtt is elkészülhet. Ennek elvét a 64. ábra mutatja be.

A láncot motor hajtja körbe, miközben a megfelelő nyomtatási pozícióban egy elektromágnessel vezérelt kalapács a papírt és a festékszalagot hozzányomja a lánc kívánt karakteréhez.

Ez azt jelenti, hogy a nyomtatás „hátulról" történik, vagyis nem a karakterképet tartalmazó láncelemet kell a papírhoz nyomni, hanem a papírt nyomja a kalapács a lánchoz.

Mind az írórudas, mind az íróláncos megoldásnál lehetőség van arra, hogy szűkített — tehát nem 64, hanem ennél kevesebb jelből álló — rúd, ill. lánc feltételével a sebességet növeljék. Ilyenkor az elhagyott speciális jeleket a nyomtató nem képes nyomtatni. ● | 284 | 9



64. ábra. Az íróláncos nyomtató működési elve

A különböző nyomtatási megoldások több szempontból eltérő nyomtatóberendezéseket eredményeznek. E szempontok közül egyik fontos tényező a berendezés ára. (A nyomtatók a számítógép többi egységéhez viszonyítva drágák.) Másik tényező a nyomtatási sebesség. Az íróláncos berendezések között pl. található olyan is, amelynek sebessége eléri a 2000 sor/min-ot.

A tárgyalt módszerek valamelyikével elért nagy nyomtatási sebesség a nyomtatásra szolgáló papírral szemben is követelményeket támaszt. Egyrészt magát a papírt kell biztonságosan továbbítani, másrészt meg kell oldani, hogy egy-egy oldal teleírása után ne kelljen új lapot befűzni, hiszen az sok időt venne igénybe. E két problémát a *széllyukasztott leporelló* oldja meg. A leporellószerűen csak perforálással szétválasztott papírlapok közül a legelsőt indulásnál be kell fűzni. A két szélén levő lyukakba belekapaszkodó továbbító műanyag tüskék azután megfelelően továbbmozgatják a lapokat. Így az egyik lap húzza maga után a másikat, mivel nincsenek szétválasztva. Az írás meghatározott sorközökkel történik. Ez géptípusonként adott vagy szabályozható. Az egész rendszernek fontos része a továbbító szerkezet (carriage), amely általában egy végtelenített lyukszalag vezérlése alatt áll. Ez a lyukszalag szinkronban mozog a leporelló lappal. A szalag hossza megegyezik a leporelló egy lapjának a hosszával. A továbbító szerkezet általában 8 csatornás lyukszalaggal működik.

A különböző csatornákon levő lyukak hatására történnek a technikailag nyomtatási képet kialakító különböző lapmozgatások. Pl. amikor a carriage-ben elhelyezett fotódióda vagy kefe a „lap vége" lyukasztást érzékeli, elkezd sort emelni, amíg a „lap teteje" lyukasztás meg nem állítja az új lap megfelelő helyén (amit előre ki kell számítani). A lap függőleges mozgását a puffertároló bal oldali bájtja vezérli. Ez azt jelenti, hogy a programozó által a puffer bal oldali bájtjába töltött karakter nem kerül nyomtatásra, hanem vezérlési célokra a sornyomtató „elnyeli".

A sornyomtatók szélessége leggyakrabban 120, 132 vagy 160 karakter. Egy lapra 60...65 sor nyomtatható.

Megfelelő leporelló befűzésével lehetőség van 2 és 3 példányos output előállítására. Maga a leporelló dobozolva kerül forgalomba. Egy-egy dobozban 1500 lap (egypéldányos) van.

A bemutatott nyomtatási technikák mindegyike rendelkezik azzal a tulajdonsággal, hogy a szöveg megjelenítése sok mechanikus mozgást igényel. A mechanikus mozgások pedig nagymértékben csökkentik a működési sebességet.

Különböző eljárásokkal kísérleteznek annak érdekében, hogy a nyomtatásnál a mechanikus mozgásokat a minimálisra lehessen csökkenteni. Az egyik ilyen megoldás, amely sikert hozott, a *lézer* felhasználásán alapul.

A *lézeres nyomtatóknál* a nyomtatott jel helyett a lézersugárral a papírra „égetett" jelről beszélhetünk. Nagy előnye, hogy egyetlen mechanikus mozgás maradt meg, a papír továbbítása. Az írási sebességet ezzel a technikával 15 000 sor/min-re sikerült növelni. Érdemes megemlíteni, hogy hazánkban is készült már lézeres nyomtató. Széles körű elterjedésére egyelőre a nagy előállítási költség miatt nem igen kerülhet sor. Várható azonban, hogy a jövőben elfogadható árú lézernyomtatók jelennek meg a piacon. ● 285 10

Mikrofilmes output

A számítási eredmények sornyomtatón való megjelenítése — a már említett viszonylagos lassúságon túlmenően — további hátrányokat is jelenthet. Ilyenek pl.:

— ha nagytömegű outputról van szó, nehézkes a feldolgozási táblák közötti keresés és eligazodás;

— ezeknek a tábláknak a tárolása nagy helyigényű;

— korlátozott a példányszám, sokszorozásuk újranyomtatással vagy fénymásolással lehetséges. Mindkettő drága.

Egyebek mellett e problémák leküzdésére született meg a gondolat: az eredmények mikrofilmen jelenjenek meg. A mikrofilmtechnika nagy könyvtáraknál már sikeresen alkalmazott megoldás volt ekkor is.

A számítógépes outputok mikrofilmen való előállításának két módja kínálkozott. A „hagyományos", amikor a papíron megjelenített eredménytáblákat fényképezik és kicsinyítik (a könyvtári alkalmazások mintájára), ill. egy olyan megoldás, amely a papíron való megjelenítést kihagyja a folyamatból. Ez utóbbi a *COM* (Computer Output

Microfilm = *mikrofilmes számítógép output*) névvel azonosított eljárás. Alkalmazásával a már említett hátrányok megoldásán túl a következő előnyök is adódnak:

— nagyobb biztonság érhető el titkos anyagok ügykezelésénél;

— illetéktelen javítások, hamisítások ellen biztosítékot nyújt;

— megsemmisítése egyszerűbb, mint a papíron levő outputoké;

— az üzemeltetési költségei alacsonyabbak a nyomtatásos papír + nyomtatási idő költségénél.

A mikrofilmes output előállításához önálló berendezésre van szükség. Tekintsük át röviden egy ilyen berendezés működését.

A számítógép az output adatokat sornyomtató helyett mágnesszalagra írja ki — formailag ugyanúgy, mintha az sornyomtató lenne, de lényegesen nagyobb sebességgel. A mikrofilm előállítása a mágnesszalagról külön történik, ez a munkafolyamat tehát már nem terheli a számítógépet. A mikrofilm lehet 16 mm széles film vagy 105×148 mm méretű filmlap. Egy ilyen filmlapra mintegy 270 leporelló lapnyi információ vihető fel.

Hazánkban is működő mikrofilmes berendezésekből a Datagraphix 4560 típusút mutatjuk be. Működése független a számítógéptől, az adatokat hordozó mágnesszalagra a következő kikötések vannak:

— megengedett adatsűrűség: 556, 800, 1600 bit/inch;

— csatornaszám: 7 vagy 9;

— kódkészlet: EBCDIC kód (ettől eltérő kódhoz konvertáló program kell).

A berendezés sebessége 20 000 sor/min. A kicsinyítés 24-szeres vagy 48-szoros lehet. ⊙ | 286 | 11

A gép a mágnesszalagról olvasott adatokat — különböző, e célra készített programok segítségével — egy nagyteljesítményű katódsugárcsövön jeleníti meg, ezüsthalogén filmen. A filmet — többnyire előhívás után — filmlapokra vágják.

Egy-egy ilyen filmlap felépítése a következő:

— *Fejléc:* szabad szemmel is olvasható módon tartalmazza a filmlapra vonatkozó legfontosabb információkat (pl. adatállomány megnevezése, dátum stb.).

— *Adatoldal:* a filmlapon belül képek vannak egymástól elkülönítve, amelyeken egy-egy A/3-as nagyságú leporelló adatai helyezkednek el úgy, ahogyan nyomtatásban megjelentek volna.

— *Index tábla:* a tartalomjegyzék szerepét tölti be, a „lapozást" segíti elő, a visszakeresési szempontoktól függően 1 vagy 2 oldalas lehet, a filmlap jobb alsó sarkában található.

A gép tulajdonképpeni programozásaként paraméter adatokkal kell megadni az input adatállományra vonatkozó információkat (rekordformátum, rekordméret, kódkészlet stb.), a fejléc tartalmát. Paraméterezni kell továbbá az adatoldalt tartalmazó képek felépítését is. (Egy sorban hány pozíció legyen a képen, hány soros legyen a kép stb.) Az index tábla tartalmát filmezés alatt az egyik program egy munkaterületre viszi, és ott kerül feltöltésre. A teljes filmlap elkészülte után kerül a fejléccel együtt a filmlapra (utolsó képként).

Az adatok visszakereséséhez, olvasásához egy olvasóberendezésre van szükség, amely a leggyakrabban egy képernyős nagyító (egy speciális fényforrásból, optikából, tükörrendszerből és egy matt üvegű képernyőből áll).

Bár üzemeltetési költségek szempontjából a COM technika kifizetődőbb a hagyományos nyomtatásnál, beszerzési költségei és a külön beszerzést igénylő olvasókészülékek miatt hazánkban széles körben még nem terjedt el, speciális alkalmazásával azonban már találkozhatunk. ● | 285 | 12 | ⊙ | 287 | 13

Terminálok

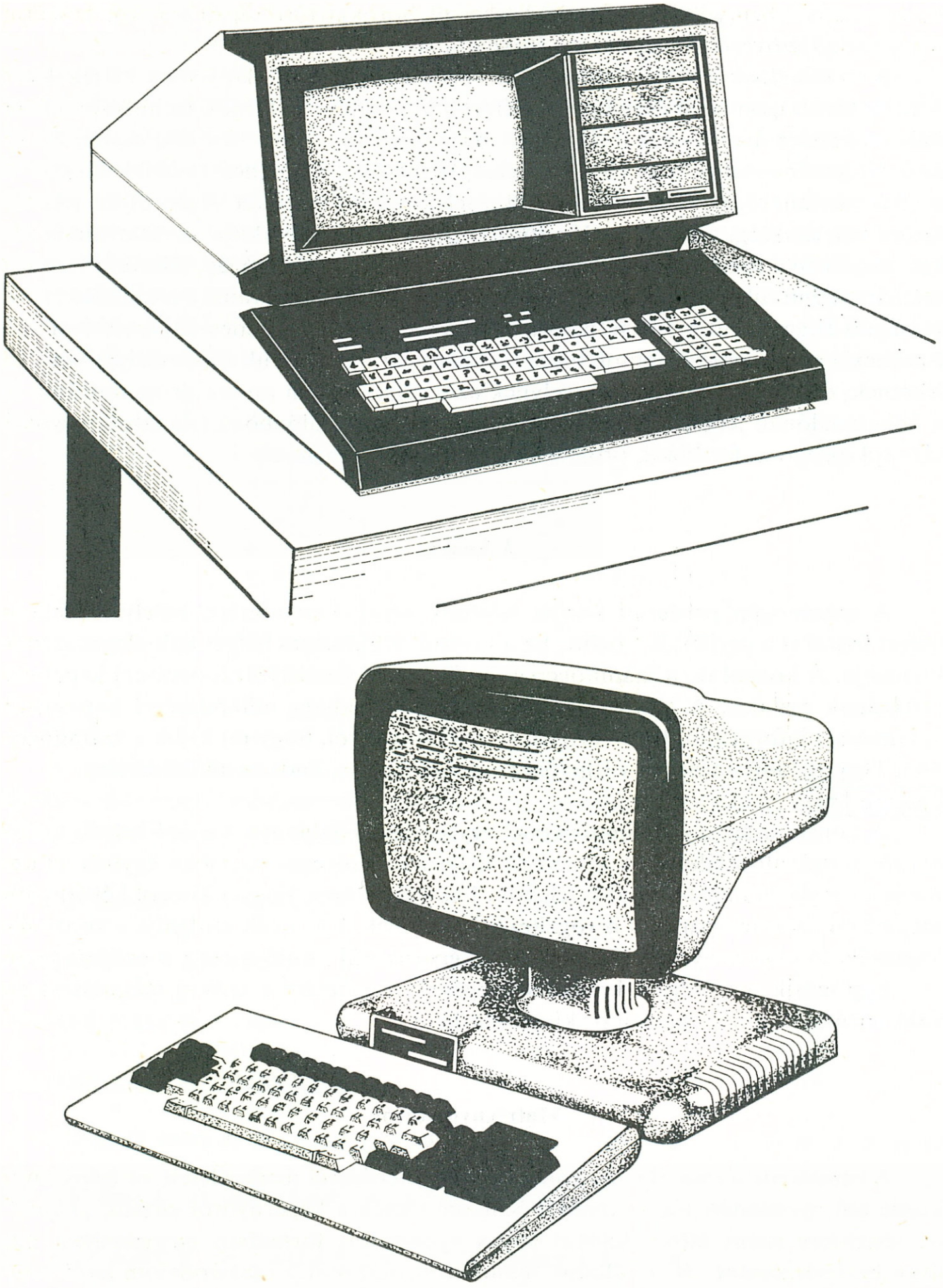
A számítógépek alkalmazásának korai szakaszában, amikor egy viszonylag szűkkörű, zömmel mérnökökből, fizikusokból, matematikusokból álló gárda használta a számítógépet, természetes volt, hogy a program írója jelen legyen a program futásakor. Nem is lehetett volna másképpen, hiszen pl. a fordítóprogramok megjelenése előtt a hiba előfordulását csak a futás során lehetett észlelni, mert sem a hiba helyéről, sem annak okáról a programozó nem kapott tájékoztatást a géptől. Később a fejlődés és a széles körű elterjedés miatt a programozók kiszorultak a gépteremből. Ez megkönnyítette az üzemeltetők dolgát, de kényelmetlenebbé tette és lelassította a programkipróbálás menetét.

A hatvanas évek közepétől a technikai fejlődés következtében ismét lehetőség nyílott arra, hogy a programozó „közel" kerüljön — nem magához a számítógéphez — a számítógép erőforrásaihoz. Természetesen nem a géptermek ajtajai nyíltak meg, hanem új hardver eszközök és szoftver lehetőségek születtek.

Mik ezek az új hardver eszközök? Ezek az új eszközök a számítógép erőforrásaihoz való kényelmes hozzáférést biztosító perifériák: a **terminálok**. A terminálok funkciója tehát az, hogy a programozó e célszerűen kialakított készülék segítségével kapcsolatba tudjon lépni a számítógéppel, s a megoldandó feladathoz a szükséges hardver és szoftver erőforrásokat igénybe tudja venni. Mindezt olyan körülmények között teheti meg, hogy munkájával nem zavar másokat, beleértve a számítógépet üzemeltető személyzetet.

A terminál leggyakoribb megjelenési formája egy katódsugárcsővel (tv. képernyőszerűséggel) összekapcsolt billentyűzet, amelyhez még az energiaellátást biztosító egység és a kétirányú kommunikációt biztosító egység tartozik. A 65. ábrán VIDEOTON terminálok láthatók.

A legegyszerűbb esetben a terminál csak arra képes, hogy a billentyűzeten beírtakat karakterenként továbbítsa a számítógéphez, amely azt megjegyzi és egyben visszza is küldi a képernyőre. A terminálok a bennük elhelyezett áramköri lapok segítségével, kényelmesebb használatúvá, „okosabbá" tehetők. A terminálokat kiszolgáló szoftveren is sok múlik, hogy a felhasználó kényelmesen tudjon dolgozni. Ilyen kényelmet jelent, ha a terminál pl. egy teljes képernyőnyi szöveget képes tárolni úgy, hogy ezen a szövegen a programozó változtatni is tud, ha szükséges. A billentyűzeten levő



65. ábra. VIDEOTON terminálok

funkcionális billentyűk használatával lehet pl. sorokat törölni, új sorokat beszúrni, több sornyi szöveget mozgatni a képernyőn stb.

A terminálnak a számítógéppel való fizikai kapcsolata alapvetően kétféle lehet. Kisebb távolságon belül (néhány tíz méter) a kapcsolatot a híradástechnikában használatos kéteres árnyékolt kábellel lehet megvalósítani. Nagy távolság esetén a kapcsolatot telefon- vagy telexvonalon lehet létrehozni, de ehhez további hardver és szoftver is szükséges. Hardver oldalról a telefonvonal mindkét végén olyan jelátalakítóra van szükség, amely vonalon továbbítható formára alakítja az elektromos jeleket, ill. a vonal másik oldalán visszaalakítja azokat a számítógép, illetve a terminál számára. Ezen túlmenően ezt a tevékenységet kiszolgáló programok is kellenek. A távolságtól függetlenül a felhasználó a számítógéppel párbeszédes kapcsolatban van. Ezeknek a párbeszédes kapcsolatoknak sajátos parancsnyelvük van, amelyek sokfélék lehetnek, de mindegyik parancsnyelvnek az a célja, hogy a puszta programíráson túl a felhasználónak lehetősége legyen a számítógépeket különböző feladatok megoldására (pl. program fordítása, program futtatása stb.) utasítani.

A konzol

A számítógép perifériái között találunk olyan berendezést, amely különleges helyet foglal el a perifériák között. Ez a konzol. Különleges helyét különleges szerepe biztosítja. A konzol ui. a számítógép és az őt kezelő személyzet (operátor) kapcsolattartásának eszköze. A konzolnál jelenik meg a rendszer működésével kapcsolatos valamennyi információ. Innen értesül az operátor arról, hogy mi zajlik a számítógépben. Ugyanakkor innen van lehetőség beavatkozni a rendszer működésébe.

▲ 283 14

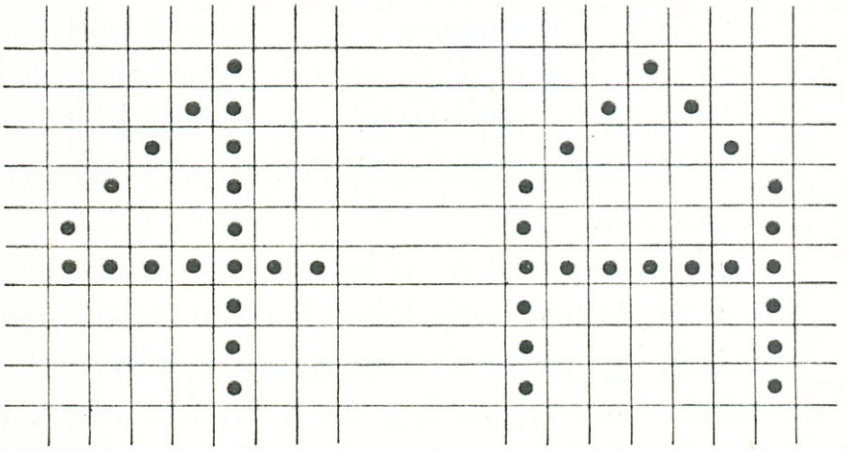
A konzol a legtöbb számítógépnél egy írógép. Újabban a konzol helyén is képernyős terminált találunk. A konzolon megjelenő szövegek azonban egyben az üzemeltetés egyik dokumentumát is jelentik. Ezért szükséges, hogy a konzol képernyőjén megjelenő üzenetek papíron is rögzítésre kerüljenek. Ezt a célt szolgálja a konzolhoz kapcsolható mátrix-nyomtató, amelyen a képernyőről kifutó szöveg rögzítésre kerül.

Egy másik megoldás a konzol szövegének rögzítésére a szöveg mágnesszalagra való kiírása, ahonnan bármikor kinyomtatható.

Mátrixnyomtatók

A képernyős terminálok és konzolok megjelenésével megszületett az igény olyan kisméretű nyomtatók iránt, amelyek szükség esetén a képernyőről eltűnő, „kifutó", de rögzítésre szánt információkat képes nyomtatott formában megjeleníteni. (Az ilyen berendezéseket, ill. az általuk nyomtatott outputot a szakirodalom hard copynak (hard kopi) „kemény másolat"-nak nevezi.)

A professzionális személyi számítógépek megjelenésével az ilyen nyomtatók iránti igény ugrásszerűen megnőtt. Ez utóbbiak esetében a hordozhatóság (kis súly, kis térfogat) igénye is felléphet. A terminálok, ill. a személyi számítógépekhez kapcsolható, a fenti kívánalmaknak megfelelő nyomtató berendezések a mátrixnyomtatók. Nevüket a nyomtatandó jel kialakításának módjából kapták. Ezeknél a nyomtató berendezéseknél a nyomtatott jel nem a hagyományos módon születik meg. Hagyományos módnak tekintjük, amikor a nyomtatandó jel fém képe valamilyen festéket tartalmazó anyagon keresztül a papírra nyomódik. A mátrixnyomtatóknál ezzel szemben a jelek pontok sorozatából állnak össze. Általában a jelek magassága 9 pontnyi, szélességük 7 pontnyi. Egy ilyen 9×7 táblázatban (a matematikában az ilyet nevezik mátrixnak) a létező, ill. a hiányzó pontok alakítják ki a nyomtatási képet. A 66. ábrán egy betű és egy számjegy stilizált alakját mutatjuk be, amely a leírt módon készült.



66. ábra. A mátrixnyomtató betűképzése

A nyomtatási képet 9 tű hozza létre, természetesen a papíron elmozdulva, több fázisban. Bár ezzel a módszerrel lassúbb nyomtatási sebesség érhető el, mint a hagyományos nagy nyomtatókkal, de ugyanakkor miután nincs kötött jelkészlet, gyakorla-

Programok = adatok + algoritmus

Programok = adatok + algoritmus

Programok = adatok + algoritmus

Programok = adatok + algoritmus

Programok = adatok + algoritmus

Programok = adatok + algoritmus

Programok = adatok + algoritmus

67. ábra. Mátrixnyomtatóval előállított szövegek

tilag bármilyen jel nyomtatása lehetséges. Ez nem csak a kisbetűk és magyar hosszú magánhangzókat jelenti, hanem szükség esetén pl. a japán vagy a kínai írás jeleit. A programozó még a nyomtatandó betűk nagyságát is befolyásolni tudja. A 67. ábrán a mátrixnyomtató által írt szöveget láthatunk. A nyomtatás sebessége 60...80 jel/s. [● | 285 | 15]

HÁTTÉRTÁRAK

A számítógépek térhódítása az adatfeldolgozás területén már a korai szakaszban azzal az igénnyel párosult, hogy lehetőség legyen nagytömegű adatok olcsó tárolására és viszonylag gyors elérésére. [● | 286 | 16] Ennek az igénynek a kielégítésére már a második generációs gépeknél megjelentek a mágnesszalagok, majd a mágneslemezek. Annak illusztrálására, hogy milyen mértékben nőtt az igény a háttértárak iránt, néhány számadat: az első mágneslemezes rendszereknél kb. 23 Mbájtnyi össz lemezterület létezett, ez az érték ma megközelíti az 1000 Mbájtot. A mágnesszalagon rendelkezésre álló háttérterület a kezdeti átlagos 47 Mbájttól 300 Mbájt fölé nőtt. Egy nagyteljesítményű számítógép teljes háttértárterülete eléri a központi tár kapacitásának 1500...1600-szorosát.

A háttértárakra való adatkivitel és az onnan való adatbevitel csak a központi egység szempontjából jelent „külső" hellyel való kommunikációt. Maguk a háttértárak is részei a számítógépnek, így ez az adatmozgatás a számítógépen belül zajlik le, azaz automatikussá tehető, ami felgyorsítja ezt a tevékenységet. Ráadásul a háttértárolókon lehetőség van arra, hogy az adatok belső (bináris) kódban kerüljenek tárolásra, így a kiíráskor és visszaolvasáskor megtakarítható az adatkonverzió ideje.

Hogyan csoportosíthatók a háttértárak?

A háttértárakat két nagy csoportra szokás osztani, attól függően, hogy a tárolt adat (rekord) milyen módon érhető el. Beszélünk *soros hozzáférésű* és *közvetlen hozzáférésű háttértárakról.* Hazánkban a soros hozzáférésű tárak közül a mágnesszalag, a közvetlen hozzáférést biztosító perifériák közül a mágneslemez terjedt el általánosan.

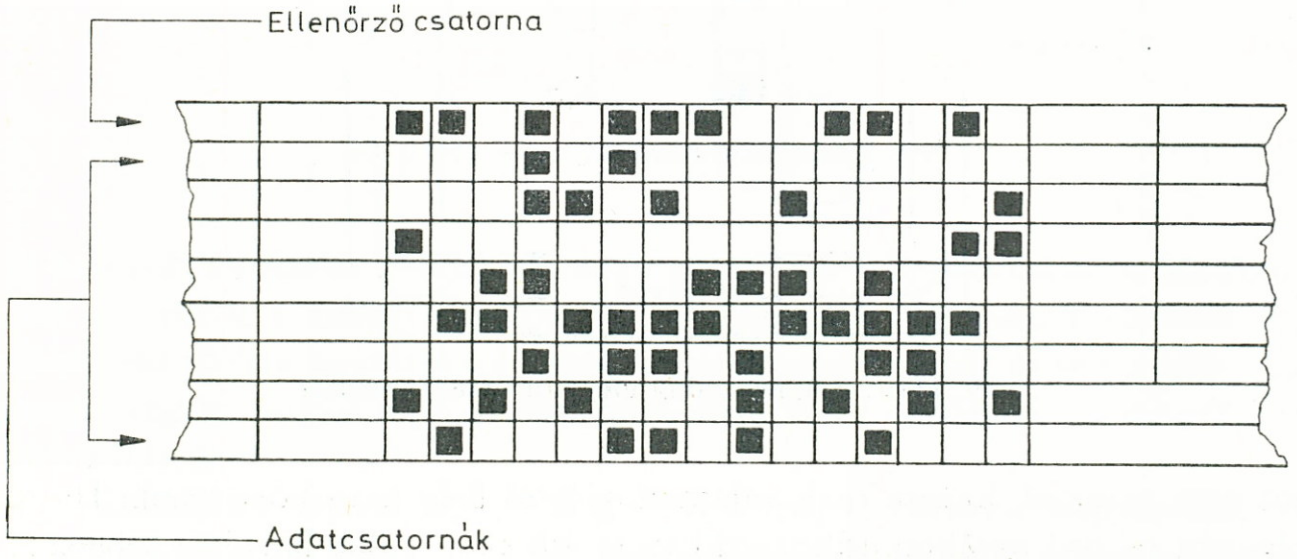
A mágnesszalag

Könnyű kezelhetősége és alacsony költsége miatt a legszélesebb körben elterjedt soros hozzáférésű adattároló a *mágnesszalag.* Mérete szabványosítva van: szélessége 0,5 inch (kb. 1,27 cm), hossza 600, 1200 vagy 2400 láb (ez utóbbi kb. 731 m). Anyaga plasztik, amelynek egyik oldala mágnesezhető anyaggal van bevonva. Szine hasonlít a szokványos magnetofonszalagok színére. Szabványos formájú csévéken hozzák forgalomba.

Azok a számítógépek, amelyeknek tára bájt szervezésű, 9 csatornás mágnesszalagot használnak. [★ | 282 | 17]

Hogyan történik e csatornákon az adattárolás?

A 9 csatornás mágnesszalagon való adatelhelyezkedés módját a 68. ábra szemlélteti. Az ábrán is látható, hogy a 9 csatorna közül 8 csatornán egy bájt található, a kilencedik csatornán pedig a bájthoz tartozó paritásbit. Szerepe és funkcionálása megegyezik a központi tárnál tanultakkal, azzal az értelemszerű különbséggel, hogy ebben az esetben a központi egységen és a központi táron kívüli adatmozgatás jóságát segít ellenőrizni, míg ott csak a központi részek közötti adatmozgatás hibátlanságát (vagy hibáját) jelezte. ● | 286 | 18



68. ábra. A 9 csatornás mágnesszalag csatornái

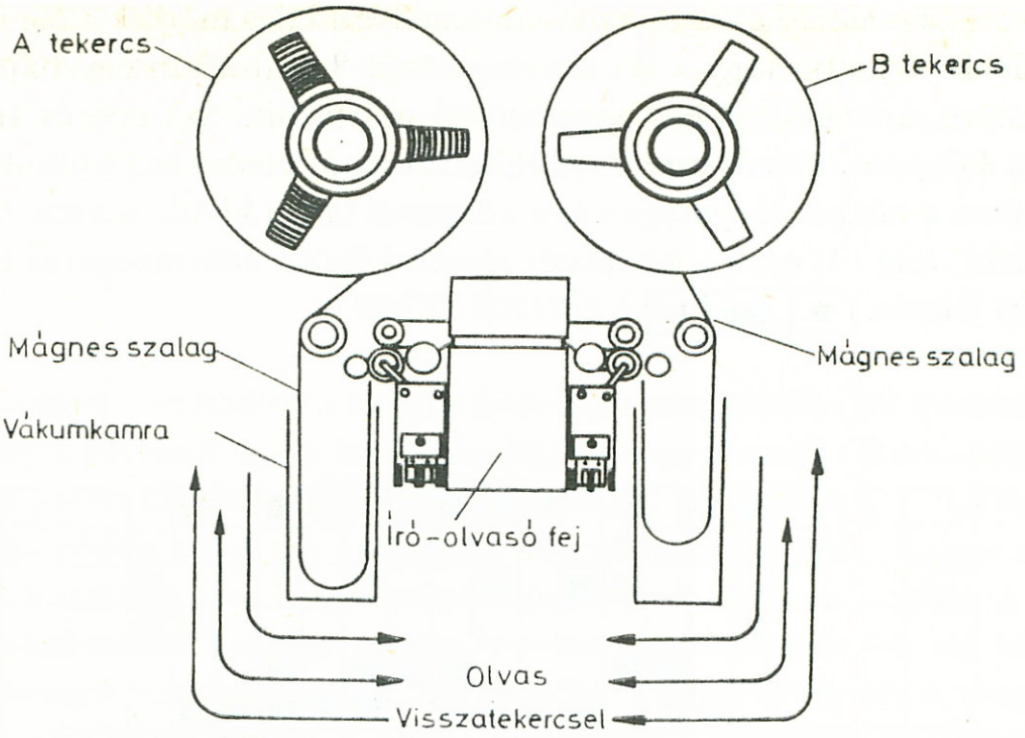
Egy következő kérdés, hogy a csatornákon milyen sűrűn helyezkednek el a bitek, azaz milyen az adatfelírás sűrűsége?

Az adatfelírás sűrűsége különböző lehet, leggyakrabban: 800 bit/inch (315 bit/cm), 1600 bit/inch (630 bit/cm), nagyteljesítményű modern rendszereknél 3200... 6400 bit/inch; természetesen minden csatornán.

A számítógép a mágnesszalagot a mágnesszalag meghajtó mechanizmuson keresztül tudja kezelni. Ennek működését szemlélteti a 69. ábra.

A szalag — miközben az egyik csévéről a másikra tekercselődik át — elhalad az író-olvasó fejek előtt, amelyek a szalagra való írást vagy az onnan való olvasást hajtják végre. Mivel a szalag viszonylag nagy sebességgel (120...150 cm/s) mozog, biztosítani kell, hogy el ne szakadjon, a két cséve — technikailag megoldhatatlan — szinkron mozgásának hiányában. Ezt a célt szolgálják a vákuumkamrák. Az ide belógó szalagrésznek sebességkiegyenlítő szerepe van. ▲ | 283 | 19

A sebességkiegyenlítésre azért van szükség, mert a szalagról való olvasás két ütemben valósul meg. Az első ütemben — ekkor a szalag teljes sebességgel mozog — történik a rekord olvasása, a másodikban a leolvasott rekord a szelektorcsatornán keresztül a központi tárba kerül. Amíg ez a második ütem zajlik, nincs mód újabb rekord olvasására. Ez egyben azt is jelenti, hogy a továbbítás idejére a szalagot vagy meg kellene állítani (ez lehetetlen), vagy minden rekord után szünetet kell hagyni a mágnesszalagon. (A „szünet" azt jelenti, hogy a szalagon a rekord után a következő



69. ábra. Mágnesszalag meghajtó mechanizmus

rekord nem azonnal, hanem csak valamennyi üres hely után következik. Ha ismert a leolvasott rekord továbbításához szükséges idő és a szalag haladási sebessége, kiszámítható ennek a szünetnek a hossza.)

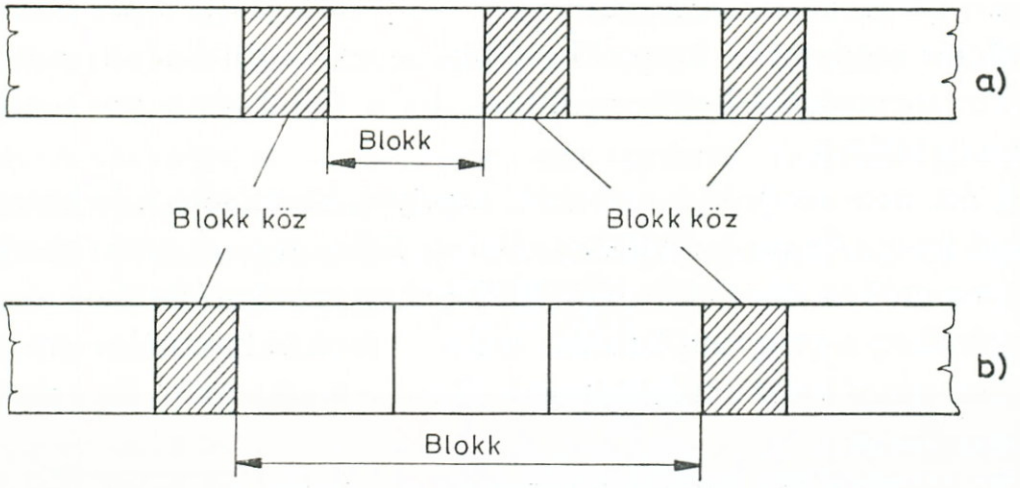
Ha a szalagot a szükséges időre lelassítjuk (majd ismét felgyorsítjuk), a kisebb sebesség miatt a szünet hossza is kisebb lehet. Minden olyan esetben, amikor a rekord mérete kicsi (néhány lyukkártyányi), a szalag kihasználása igen rossz, mert hosszabbak lesznek a szünetek, mint az értékes (rekordokat tartalmazó) területek.

Hogyan lehetne növelni a hasznos területeket a mágnesszalagon? Nyilvánvalóan úgy, hogy egy ütemben az eddiginél nagyobb adatmennyiséget próbálunk átvinni, mondjuk nem egy rekordnyit, hanem több (pl. 10 vagy 100) rekordnyit. Ennek megvalósításával elvált egymástól a program által egy egységnek kezelt rekord és a gép által egy átviteli egységnek kezelt adatmennyiség, amelyet **blokknak** nevezünk.

Egy blokk tehát több (általában több tíz) rekordból álló adatmennyiség, amelyet a mágnesszalag-meghajtó olvasó mechanizmusa egy egységnek kezelve olvas le és továbbít a szelektorcsatornára. A blokkok közötti szüneteket ***blokkközöknek*** nevezzük. Az elmondottakat a 70. ábra szemlélteti.

A 70*a* ábrán nincs blokkolás, minden rekord után kihasználatlan hely, blokköz van, a 70*b* ábrán a rekordok hármasával vannak blokkolva. (A 70*a* ábrán levő esetet szokás blokkolatlan ábrázolásnak nevezni.) A blokkok közötti köz mérete 800 bit/inch sűrűség mellett 480 bájt, és ez a méret független az előtte levő blokk hosszától, csak az írássűrűségtől függ.

Az elmondottakból az is következik, hogy egy mágnesszalagra sohasem fér fel annyi adat, mint amennyi a szalag névleges kapacitása. A ténylegesen felírható adatmennyiség az írási sűrű égen kívül a blokkmérettől is függ. ⊙ | 287 | 20



70. ábra. Adatábrázolás mágnesszalagon

A blokkolt ábrázolás velejárója, hogy a szalagra való felíráskor is blokkokban érkeznek az adatok a szelektrocsatornáról. A rekordok blokkolását (felíráskor), ill. a blokkok rekordokra bontását (olvasáskor) az operációs rendszer erre a célra készült programjai végzik, vagyis a programozónak nem kell foglalkoznia ezzel a kérdéssel a program írásakor.

A továbbiakban azt vizsgáljuk meg, hogy a mágnesszalagon hogyan helyezkednek el, és hogyan alkotnak összetartozó adatállományt a blokkokba szervezett rekordok.

A szalag elején — kb. 4 m-re a szalag fizikai elejétől — a nem mágnesezett oldalon egy kis fémlapocska található. Ez a *szalagkezdet jel*. A szalagkezdet jel előtti rész a befűzéshez szükséges. A meghajtó mechanizmus fotoelektromos úton tudja a szalagot a kezdőpontra pozicionálni. Ugyanilyen ún. *szalagvég jel* található a szalag fizikai vége előtt mintegy 7 m-rel. Szerepe annak biztosítása, hogy a szalag ne tudjon leszaladni a csévéről. ▲ | 283 | 21

A mágnesszalagon tárolni kívánt blokkok tehát a szalagkezdet jel és a szalagvég jel közötti részre írhatók fel. Az egymás után felvitt blokkok alkotják a mágnesszalagra írt fájlt.

Hogyan lehet eligazodni a különböző mágnesszalag tekercsek és a rajtuk levő fájlok között? Az eligazodást, a tájékozódást az erre a célra szolgáló *címkék* biztosítják. A címke speciális rekord, amelynek tartalma a mágnesszalag, ill. a rajta levő fájl vagy fájlok azonosítását végzi. Háromféle címkét szokás megkülönböztetni. Ezek:

— az *adathordozói címke*, amely a mágnesszalag tekercs azonosítását végzi, és a szalagkezdet jel után található;

— a *fájl bevezető címke*, amely a címkézett fájl elején helyezkedik el és a fájl azonosítását végzi;

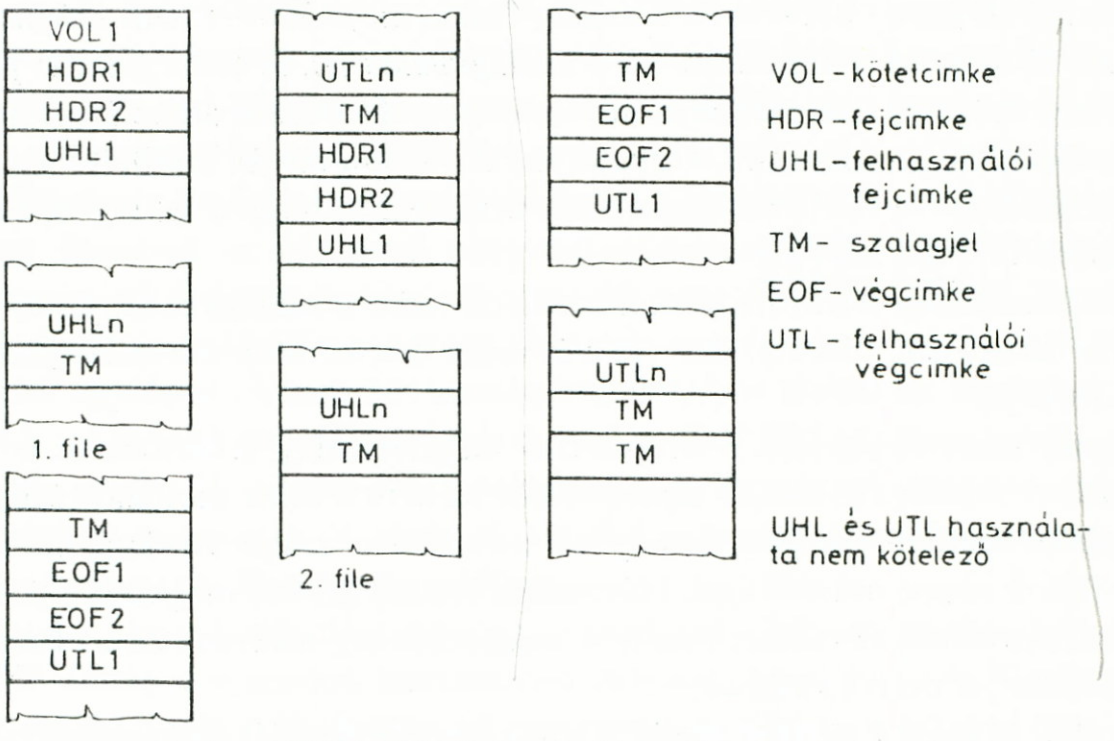
— a *fájl zárócímke*, amely a fájl utolsó rekordja (blokkja) után helyezkedik el. ⊙ | 288 | 22 Ezek az ún. *szabványos címkék*, előnyük, hogy a programozónak nem kell különösebb gondot fordítania a kezelésükre, az operációs rendszer a felírásukat, olvasásukat automatikusan elvégzi, ha ezt a programozó előírja.

A szabványos címkék alkalmazása tehát kényelmet jelent a programozó számára. Azonos operációs rendszerrel üzemelő számítógépek közötti adatcserét mágnesszalagokon problémamentesen lehet bonyolítani, ha a fájlok és a tekercsek szabványos címkével vannak ellátva.

Léteznek ún. *nem szabványos címkék*, amelyek kezelésére a felhasználónak saját programot kell írnia. Az ilyen címkével ellátott fájlokat csak ezeknek a kezelőprogramoknak az ismeretében lehet elolvasni. Elsősorban munkafájlokhoz — amelyek nem kerülnek megőrzésre a program lefutása után — szokás használni címkézetlen szalagokat is. Ezeken nincs semmi információ a fájlra vonatkozóan. Egy programon belül célszerű a használatuk.

Hány fájlt célszerű tárolni egy tekercsen? A kérdést az indokolja, hogy sokszor akkora fájlokkal dolgozik egy-egy program, amely egy-egy tekercsnek csak töredék részét foglalja el.

Vannak számítóközpontok, amelyekben előírt szabály, hogy egy tekercsen csak egy fájl lehet. (Igen nagyméretű fájlok persze csak 2—3 tekercsen férnek el.) Ennek oka elsősorban a szalagok technikai kezelésében jelent előnyt, viszont lényegesen több szalagot kell tárolni, hiszen a rövid fájlok mögött sok üres hely marad egy-egy szalagon. Amennyiben egy tekercs több fájlt is tartalmaz, a mágnesszalagon minden fájl előtt és mögött fel kell írni az adott fájl bevezető és zárócímkéjét. Egy ilyen állapotot mutat a 71. ábra.



71. ábra. Több fájl elhelyezkedése mágnesszalagon

Nem tettünk még említést az ábrán látható *szalagjelről*. Ez egy különleges karakter, amely elválasztja a címkék csoportját a tényleges fájltól. A szalag logikai végét (tehát a szalagon levő utolsó fájl végét) két szalagjel jelzi, az ábrán is látható helyen.

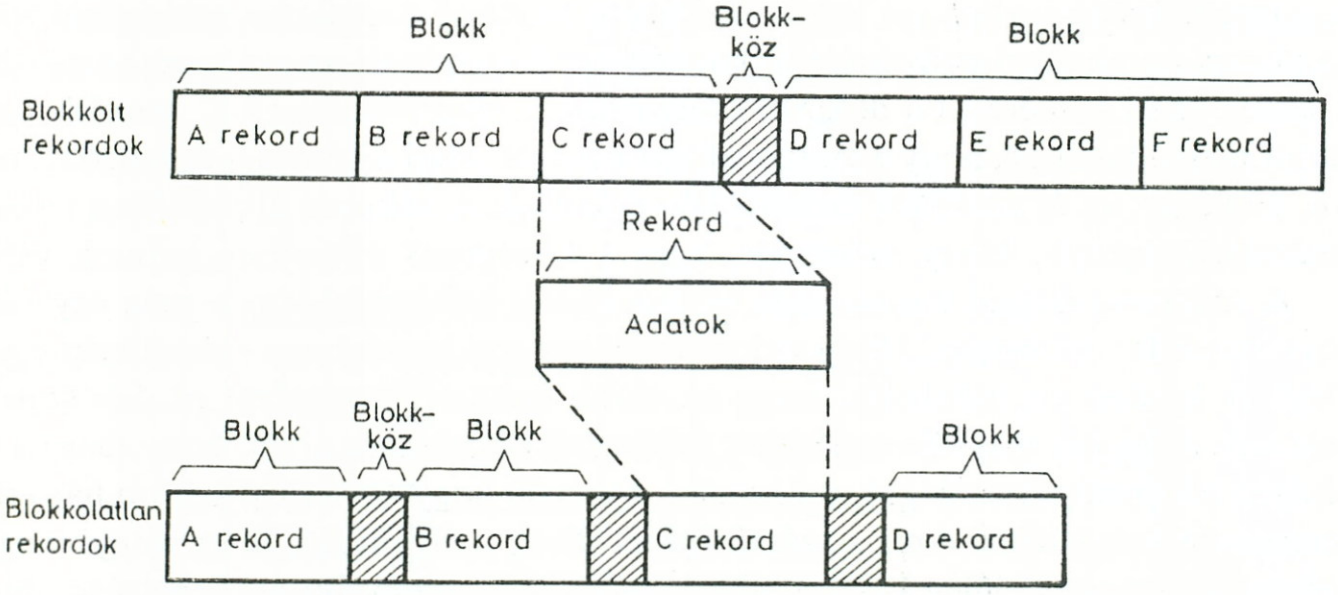
A korábbiakban már volt szó arról, hogy a fájlban levő rekordok lehetnek blokkolatlanok és blokkoltak. Egy másik szempont szerint a rekordok lehetnek:

— *fix hosszúságúak* (minden rekord egyforma hosszú);

— *változó hosszúságúak* (a rekordok nem egyforma hosszúak);

— *meghatározatlan hosszúságúak.*

A 72. ábrán bemutatjuk a leggyakrabban használatos fix hosszúságú rekord blokkolatlan és blokkolt változatát. ⊙ 288 23



72. ábra. Fix hosszúságú blokkolt és blokkolatlan rekordok

A mágnesszalagos állományok kezelése

Soros feldolgozások esetén a mágnesszalagok használata igen előnyös. A kb. 40 cm átmérőjű, műanyag dobozba vagy zárógyűrűvel védve viszonylag kis helyen tárolhatóak a szalagok. A mágnesszalagokon levő adatok védelmét fizikai eszközök is segítik. Ilyen eszköz az írást megengedő gyűrű, vagy röviden *írógyűrű.* Alkalmazásának megértéséhez tudnunk kell, hogy egy mágnesszalagos fájl egy program működése során vagy csak inputként vagy csak outputként szerepelhet. Az előző esetben csak olvasás történik a szalagról, a második esetben csak írás történik a szalagra. Amikor az operátor felteszi a mágnesszalagot a meghajtóra, már tudnia kell, hogy a szalag milyen „szerepet" fog játszani. Írási művelethez a szalagorsóban levő kör alakú mélyedésbe be kell helyeznie egy műanyag gyűrűt. Ennek hiányában a rendszer nem hajlandó írni a feltett szalagra. A gyűrű hiánya megvédi a szalagot a véletlen és nem kívánt törléstől vagy felülírástól.

A mágnesszalagos állományok védelmét szolgálja a tartalék vagy archív másolatok készítése is. Nagyobb feldolgozási rendszereknél, ahol meghatározott időközönként (naponta, hetente, havonta stb.) ismétlődő feldolgozások vannak (pl. az OTP átutalási betét rendszere) a rendszer biztonsága szempontjából meg kell őrizni két

feldolgozásra visszamenően az alapfájlokat. Ez lehetővé teszi, hogy ha pl. hardver hiba miatt egy feldolgozás során a törzsfájl tönkremegy, az előző havi törzsfájl felhasználásával, egy havi feldolgozást megismételve ismét előálljon a megsemmisült törzsfájl. Ahol ennek különösebb jelentősége van, előírás lehet, hogy minden fájlról készüljön másolat, amelyet rendszerbiztonsági okokból megőriznek.

Itt kell felhívni a figyelmet a mágnesszalagok egyik tulajdonságára, amely szintén növeli a feldolgozási rendszerben használt szalagok számát. Említettük már, hogy egy szalagos fájl egy program futása alatt vagy csak input vagy csak output lehet. Ez a megállapítás egyben azt is jelenti, hogy nincs mód a szalagba való belejavításra. Ha pl. egy munkaügyi nyilvántartásban néhány dolgozónak megváltozott az órabére, és ezeket az új órabéreket a dolgozók rekordjában rögzíteni akarjuk, nem lehet a feladatot megoldani úgy, hogy a szalagnak azt a helyét, ahol a kérdéses dolgozó rekordja van, felülírjuk az új adatokat tartalmazó rekorddal. A változás átvezetése a teljes fájl lemásolását igényli. Hogy miképpen, arra a következő fejezetben kapunk választ.

A mágnesszalaggal kapcsolatos vizsgálataink befejezéseképpen még egy kérdés megválaszolása szükséges. Hány mágnesszalagot tud kezelni egy számítógép egyidejűleg? A kérdést az indokolja, hogy az előbb említett feladatban pl. két szalagról esett szó. A kérdés megválaszolásakor külön kell beszélnünk arról, hogy maga a számítógép hány mágnesszalagot tud kezelni, és arról, hogy egy-egy program hány mágnesszalagot használhat. Ami a számítógépet illeti, nyilvánvaló, hogy annyi mágnesszalagot képes egyidejűleg kezelni, ahány meghajtó üzembeállítása lehetséges, hiszen minden szalaghoz egy önálló meghajtó kell.

A számítógépekhez kapcsolható meghajtók számát a szelektorcsatornák száma határozza meg. Minden szelektorcsatornához nyolc alcsatorna, és minden alcsatornához nyolc periféria csatolható. Ha tehát háttértárként csak mágnesszalagokat kívánunk egy számítógéphez csatolni (lemezeket nem), akkor egy szelektorcsatornán keresztül 64 meghajtó lehetséges, de ezek közül egyidejűleg csak egy lehet aktív, hiszen a szelektorcsatorna egyidejűleg egy perifériát szolgál ki. A valóság ezzel szemben az, hogy egy-egy számítógéphez ennél jóval kevesebb mágnesszalag egységet kapcsolnak. Számukat az elvégzendő munkák befolyásolják. Átlagos számuk — hazai gépeinken — 4 és 8 között változik.

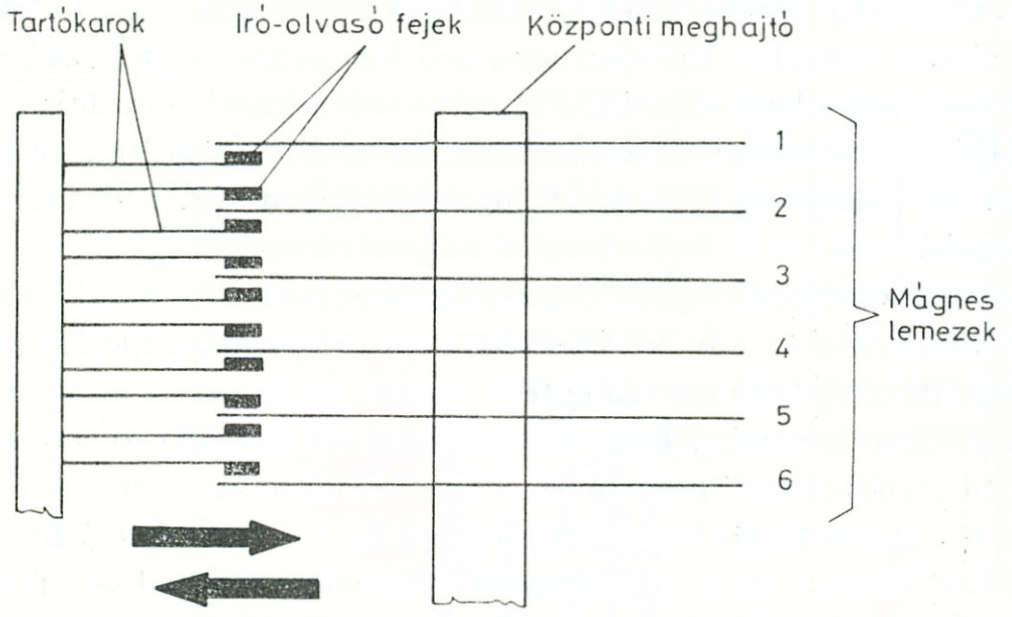
Azt viszont, hogy egy program ezek közül a készülékek közül hányat vehet igénybe, a számítóközpontok általában korlátozzák. Így nem szokás megengedni háromnál több egység igénybevételét, ill. ha ez szükséges, akkor külön kell kérni pl. a diszpécser szolgálatnál. Ennek alapvető oka, hogy — mint majd a későbbiekben látni fogjuk — a számítógép egy időben több programmal foglalkozik, így a többi meghajtót más programok használják. Más oldalról a mágnesszalag egységek esetleges meghibásodásakor nem futtatható olyan program, amely az összes meghajtót használni akarja.

A mágneslemez

A közvetlen elérésű adattárolók közül legelterjedtebb a *mágneslemez*.

Azonos elven működő, de különböző kapacitású mágneslemez egységek és lemezcsomagok vannak forgalomban. Kisebb teljesítményű gépek 7,25 Mbájt és 29 Mbájt névleges kapacitású lemezcsomagokat (és a hozzájuk tartozó meghajtókat) használnak. A nagyteljesítményű gépek (1 Mbájt körüli és e feletti központi kapacitással) már 100 Mbáj t névleges kapacitású lemezeket használnak.

A mágneslemez működési elvét a 7,25 Mbájtos lemez fizikai paramétereivel mutatjuk be. A 73. ábra egy lemezcsomag és a meghajtó mechanizmus metszeti képét mutatja.



73. ábra. A mágneslemez működési elve

A 12 lehetséges lemezfelületből a legfelsőt és a legalsót nem használják. A 10 felület mindegyikén 203—203 *sáv* — koncentrikus kör — helyezkedik el, amelynek mentén az adatok felírhatók. Az adatrögzítést a felületek mágnesezhetősége biztosítja. A különböző felületek azonos sorszámú sávjait *cilindernek* nevezzük.

A 203 sáv közül egyidejűleg mindig 200 használható, a fennmaradó 3 tartalék célokat szolgál arra az esetre, ha egy sáv megsérülne.

A mágneslemez csomagot a lemezmeghajtó egységre kell feltenni. A meghajtó mechanizmus általában 40 fordulat/s sebességgel forgatja a lemezcsomagot egy függőleges tengely körül. A lemezoldalak közé öt karon tíz író-olvasó fej nyúlik be. A kiválasztott író-olvasó fejre adott impulzus hatására történhet egy-egy felületről az olvasás vagy az oda való írás.

Az író-olvasó fejeket mozgató karok csak együtt képesek mozogni. A cilinder jelentősége éppen abban van, hogy egy karállással tíz sávot (egy cilindert) lehet végigírni vagy -olvasni. A sok időt ui. a mechanikus mozgás emészti fel. (A nagyobb kapacitású lemezcsomagoknál nem tíz, hanem húsz felület van, és az írási sűrűség is nagyobb.)

Miután a mágneslemezeken mód van arra, hogy a felírt rekordokat közvetlenül is elérhetjük, felmerül a kérdés, hogy mennyi ideig tart egy-egy rekord megtalálása, vagyis hogy mekkora a mágneslemezek *hozzáférési ideje.*

A közvetlen hozzáférés a központi tár esetében minden tárcímre vonatkozóan azonos ideig tart, függetlenül a cím fizikai helyétől (vagyis attól, hogy kisebb vagy nagyobb sorszámú rekeszekről van-e szó). Mágneslemez esetén a hozzáférési idő függ attól, hogy a keresett adat hol helyezkedik el a lemezen. Ha áttekintjük, hogy milyen tényezők befolyásolják az elérési időt, könnyen beláthatjuk ennek az okát is.

Az *első tényező*, amely az elérés idejére hatással van, a *fejmozgás ideje.* Fejmozgási időnek nevezzük azt az időszükségletet, amely az író-olvasó fejeknek a kívánt cilinderre való pozicionálásához (mozgatásához) szükséges. A fejmozgás ideje alapvetően attól függ, hogy mekkora utat kell a fejeknek megtenniük. Lehet, hogy éppen a kívánt cilinderen vannak — ilyenkor nem kell fejmozgás — | • | 286 | 24 | lehet, hogy egy távolabbi (esetleg a legtávolabbi) cilinderre kell eljutni. Ha a fejeknek a pillanatnyi helyükről el kell mozdulni, ennek időszükséglete minimum 10...30 ms, maximum 50...165 ms. Így átlagosan (a lemez típusától függően) 25...90 ms időszükséglete van a fej mozgásának.

A kívánt cilinderre való ráállás után elektronikusan történik a megfelelő író-olvasó fej kiválasztása (a sáv kiválasztása). Miután ehhez mechanikus mozgásra nincs szükség, időigénye elhanyagolható.

Gondoljuk azonban meg, hogy a kijelölt sávra megérkező író-olvasó fej és a keresett rekord „találkozása" hogyan jöhet létre. Szerencsés esetben a keresett rekord éppen akkor érkezik a forgó lemezfelületen, amikor az író-olvasó fej beáll a sávra. Ilyenkor azonnal lehet olvasni. „Szerencsétlen" esetben az író-olvasó fej éppen lekéste ezt a találkozást, egy körülfordulásnyi időt várakoznia kell. (Persze vannak közbülső esetek is, amikor nem egy teljes körülfordulásnyi időt kell várni.) Ezt a várakozási időt *forgási időnek* nevezzük, és *ez a másik tényező*, amely befolyásolja az elérési időt. A forgási idő időtartama a lemez forgási sebességétől függ. Ez általában 25 ms, bár vannak ennél nagyobb sebességgel forgó lemeztípusok. Az átlagos forgási idő 2,5... 12,5 ms.

Az előbbiek alapján a mágneslemezek átlagos elérési ideje típustól függően *30...100 ms* közötti értékek. A 100 Mbájt méretű lemezeknél pl. — a technikai fejlesztések következtében — az átlagos hozzáférési időt 30 ms-ra sikerült csökkenteni.

A rekordok közvetlen eléréséhez ezt megvalósító mechanizmuson — a fejmozgás lehetőségén — kívül még valamire szükség van. Hasonlóan a központi tárhoz, a mágneslemezen is szükséges egy olyan *címzési lehetőség*, amelynek segítségével megadható, hogy az író-olvasó fejek a lemez mely részére álljanak a direkt kereséskor. Ezt a címzési lehetőséget a sávok szerkezete adja meg. Mágneslemezek esetén a címzés a következő módon történik:

— Mindenekelőtt meg kell adni, hogy a keresett rekord *melyik cilinderen* található. Ezt mutatja a *cilinderszám*, amely 0...199 között lehet (a tartaléksávok a 200, 201, 202 számot viselik).

— A cilinderen belül tudni kell, hogy melyik felületről, vagyis *melyik sávról* van szó. Ez megegyezik a sávhoz tartozó író-olvasó fej sorszámával. Ezt nevezik *fejszámnak*.

A mágneslemez valamennyi sávjának elején ▲ | 283 | 25 az adott sávra vonatkozó fenti két azonosító megtalálható. A sávon belül azonban általában nem egy adatrekord vagy adatblokk helyezkedik el, mivel a sáv kapacitása ennél többet tesz lehetővé. Szükség van tehát a sávon belül további azonosításra. Ez viszont azt jelenti, hogy a sávon az adatrekordokon kívül más jellegű információknak is el kell helyezkedniök. Ezért egy *lemezbeli rekord* ★ | 282 | 26 általában két részből áll: az első rész, amely az *azonosítási célokat* szolgálja (cilinderszám — fejszám — sávon belüli sorszám), a másik részen pedig a *felhasználói adatok* tárolása lehetséges. Ezek a felhasználói adatok blokkoltan vagy blokkolatlanul helyezkednek el, a mágnesszalagnál már megismert mód szerint.

Fontos megemlíteni, hogy minden sáv első rekordjában a sávra vonatkozó információk kerülnek tárolásra (pl. van-e szabad hely a sávon stb.). ◉ | 290 | 27

A lemezsávok fizikai szerkezetének vázlatos megismerése után arra vagyunk kíváncsiak, hogy a mágneslemezen hogyan helyezkednek el az adatállományok. A kérdés azért is indokolt, mert már a mágnesszalagnál láttuk, hogy különböző rendszerinformációk (címkék) is szükségesek a szalagon.

A mágneslemezen való tájékozódás — fájlok, rekordok fizikai helyének megtalálása — lényegesen bonyolultabb feladat, mint a mágnesszalag esetében volt. Ennek főbb okai:

— a lemezeken mindig több (több tucat) fájl kerül elhelyezésre;

— egy-egy fájl fizikailag távoleső lemezterületen (ún. tartományokban) is elhelyezkedhet;

— a közvetlen elérést biztosító címzési rendszeren keresztül nemcsak a fájlok kezdetét kell azonosítani, hanem a fájlon belül is tájékozódni kell.

Ezeket a mágnesszalaghoz képest bonyolultabb feladatokat a lemezen levő — a mágnesszalagéhoz hasonlítva bonyolultabb — információk biztosítják.

A lemezen való eligazodásban az operációs rendszert a *mágneslemez címkék* segítik. A szabványos mágneslemez címkék a következő csoportokba sorolhatók:

— *adathordozói címke* — amellyel a lemezköteg azonosítható;

— *az operációs rendszer által használt fájl címkék* — amelyek segítségével a lemezen való tájékozódás lehetséges;

— *felhasználói fájl címkék* — amelyeket a felhasználó helyezhet el a fájlok elején.

Az adathordozói címke (kezdeti köteg címke) 80 bájt méretű, feladata a lemezcsomag azonosítása. Tartalmazza továbbá annak a különleges fájlnak a kezdőcímét, amelyben a rendszer által használt fájlokra vonatkozó információk foglalnak helyet. Ez a különleges fájl funkcióját tekintve a lemez *tartalomjegyzékének* szerepét játssza. Az angol „Volume Table of Contents" elnevezés rövidítéséből meghonosodott a magyar szakirodalomban is a *VTOC* elnevezés.

A kezdeti köteg címke a lemez elején, a 0. cilinder 0. sávjában van. Használata kötelező.

A VTOC minden fájlról, amely a lemezen van tartalmazza egyebek mellett a fájl nevét (azonosítóját), a fájl belső szerkezetének leírását (rekordméret, blokkméret, a szervezettség típusa), a fájl fizikai kezdőcímét a lemezen. Ezen kívül a VTOC külön nyilvántartást tartalmaz a lemezen levő üres (felhasználható) területekről. A VTOC ezen kívül a lemezen levő fájlokra vonatkozó további bejegyzéseket is tartalmaz. ⊙ 291 28 A VTOC-ba az operációs rendszer programjai végzik el a bejegyzéseket, a rendszerprogramozó speciális programok segítségével tud a VTOC-hoz hozzáférni.

A MIKROSZÁMÍTÓGÉPEK PERIFÉRIÁI

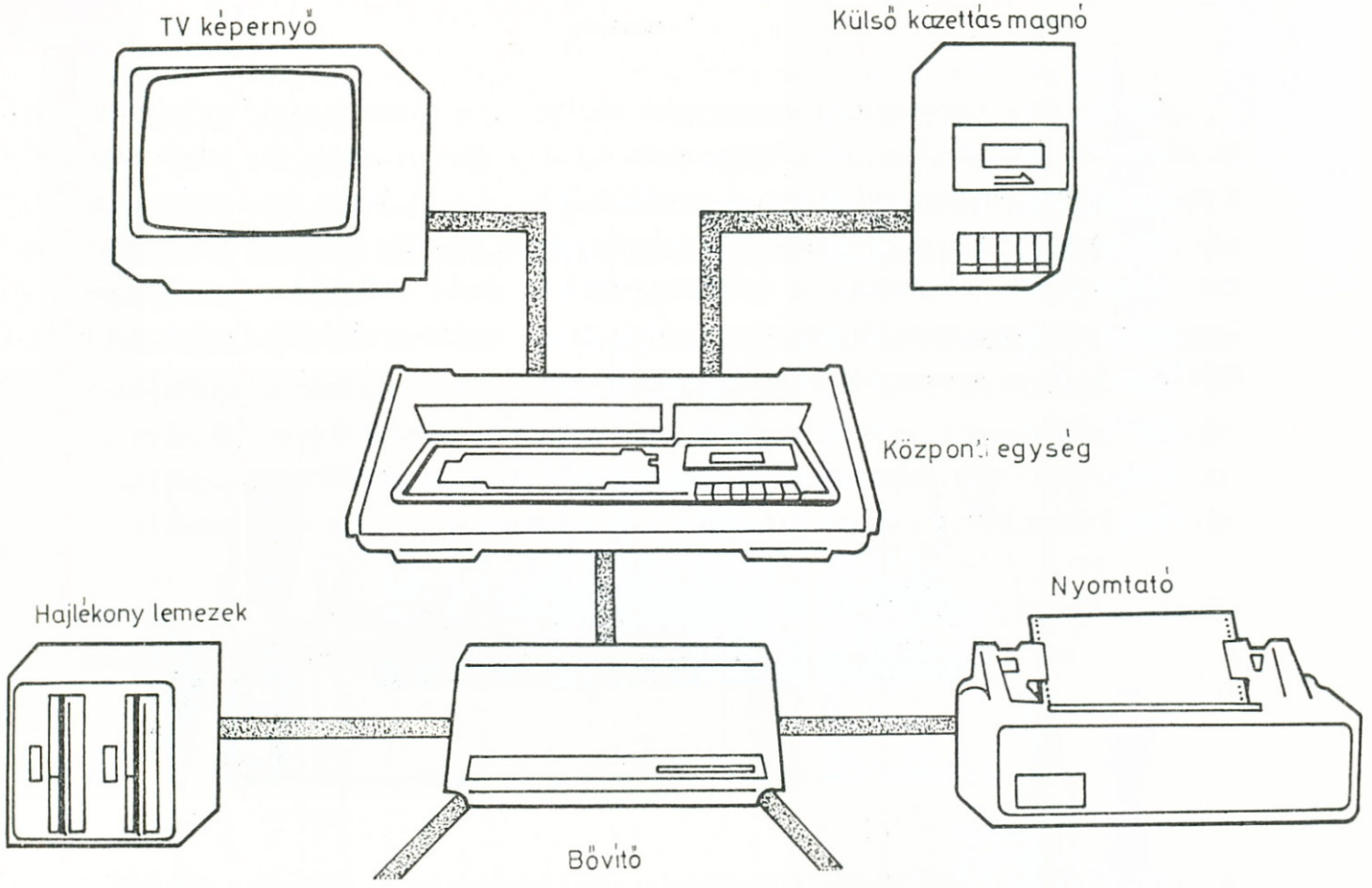
A mikroszámítógép elnevezésben a mikro szó azt jelenti, hogy nagyon kicsi. Az már nem egyértelmű, hogy ezeknél a gépeknél mi az, ami „nagyon kicsi". Általában kicsi a gépek helyigénye: aktatáskában vagy hasonló kis helyen elférnek. Nagyjából kicsi a gépek ára (természetesen egy hagyományos számítógéphez viszonyítva), de itt már lehetnek lényeges eltérések is. Mindenesetre maga a „mikro" szó semmit sem árul el ezeknek a gépeknek a teljesítményéről, a képességekről. Itt is széles a skála, hiszen az egyszerűbb video játékoktól kezdve a hagyományos értelemben vett „nagygépek" nyújtotta lehetőségeket felülmúló teljesítmények lehetségesek a mikrogépek között is. Éppen azért, mert ilyen széles tartományban mozoghatnak a mikroszámítógépek, a továbbiakban a gépek azon kategóriájával ismerkedünk, amely a középiskolás számítástechnika, programozás területén használható típust jelenthet. Egy ilyen kiépítettséget szemléltet a 74. ábra. Egy mikroszámítógép a következő perifériákon keresztül tart kapcsolatot a környezetével:

Billentyűzet

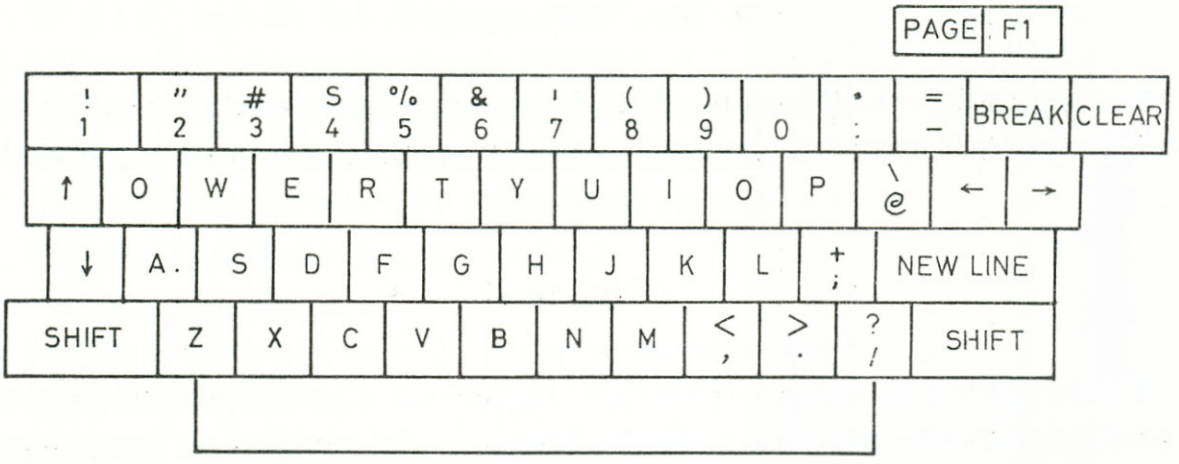
A billentyűzeten keresztül van lehetősége a programozónak arra, hogy közlendőit a gépbe bevigye. A billentyűzet tehát egy input egysége a gépnek. Betűk, számok billentyűi mellett ún. funkciós billentyűk is előfordulnak. A 75. ábrán a HT 1080Z iskolaszámítógép billentyűzetét láthatjuk.

A billentyűzettel kapcsolatban meg kell említeni, hogy a nagyszámítógépekkel ellentétben a mikrogépeknél reális igény a kisbetűk alkalmazása, hiszen itt az adatokon és programokon kívül sokszor szövegek kialakítása és tárolása is lehet a cél. Külön probléma a magyar abc ékezetes betűinek a biztosítása. Ez nagyon fontos, hiszen tudjuk, hogy az írógépekről hiányzó hosszú magánhangzók mennyit rontanak helyesírásunkon.

A hazai gyártású PRIMO személyi számítógép billentyűzetén ezt a problémát megoldották. Így nincs akadálya, hogy e számítógépek segítségével a magyar nyelv helyesírási szabályainak mindenben megfelelő szöveget lehessen tárolni és nyomtatni.



74. ábra. Mikroszámítógép kiépítettség



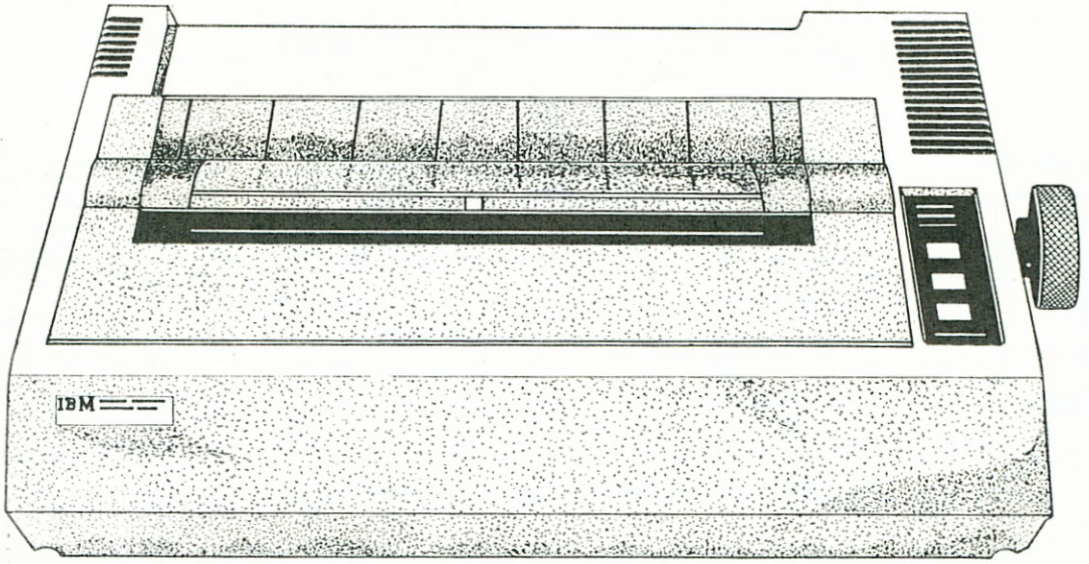
75. ábra. A HT 1080Z típusú számítógép billentyűzete

Képernyő

Már a terminálóknál is láttuk, hogy a megjelenítés fontos eszköze a katódsugárcső. A számítógépek egy része saját képernyővel rendelkezik, más részüknél biztosítva van közönséges TV készülékek csatlakoztatása. Ez a TV készülék lehet fekete-fehér vagy színes. Egyre több mikrogép képes arra, hogy kezelni tudja a különböző színek megjelenítését a színes képernyőn.

Nyomtatók

Az első mikrogépeknél a legnagyobb problémát a nyomtatási lehetőség biztosítása jelentette. Viszonylag gyors berendezésre volt szükség, amely méretben illeszkedik a mikrogéphez, és nem túl drága. A problémát az okozta, hogy sok mechanikus mozgás van nyomtatáskor, ami lassítja a teljesítményt. Sokféle nyomtatómegoldás született, alapvetően valamennyi a mátrixnyomtatás elvén működik. A mikrogépekhez kapcsolt mátrixnyomtatók egy része (a már tanult módon) többféle betűtípust tudnak előállítani. Ilyen nyomtatót szemléltet a 76. ábra.



76. ábra. Mátrixnyomtató

A nyolcvanas évek elején a nyomtatók ára még aránytalanul magas. Egy megbízható teljesítményű nyomtató ára megegyezik egy jóminőségű mikrogép árával.

Háttértárak

A mikrogépeknél, miután az alapvető beviteli egység a billentyűzet, nagyon fontos kérdés a háttértár megoldása. Ennek hiánya ui. azt jelentené, hogy egy-egy programot minden használat előtt ismét és ismét kézzel kell bevinni, ami értelmetlen és felesleges. A háttértárak egyik csoportja a nagygépes mágnesszalagok mintájára a *kazettás magnetofon*. Hatalmas előny, hogy az így kialakított mikrogéphez a forgalomban levő bármilyen kazettás magnetofonkészülék csatlakoztatható, és az ugyancsak bolti forgalomban levő (jobb minőségű) kazetta használható.

A kazettáról való programbetöltéskor több típusnál a programozónak nagyon oda kell figyelni, hogy honnan kezdi a betöltést (olvasást) a kazettáról, mivel az erre vonatkozó ellenőrzési funkció nincs minden mikrogépbe beépítve. Ahol ilyen létezik, elegendő megadni a keresett program nevét és a keresés automatikusan lezajlik. A kazettás magnetofonról az adatok vagy programok betöltése hosszadalmas, egyrészt

a szekvenciális keresés, másrészt a kazetta sebessége miatt. Ezért szükségessé vált olyan háttértár kidolgozása, amely közvetlen elérést is képes biztosítani. Így jöttek létre a különböző *hajlékony lemezegységek*. Méretben, írássűrűségben és a felhasznált oldalak szerint többféle hajlékony lemez és meghajtó van forgalomban.

A hajlékony mágneslemezen az információ koncentrikus körök mentén helyezhető el. Ezeket itt is sávoknak nevezzük. A sávok kisebb fizikai egységekre, ún. szektorokra oszlanak.

Az adat címzése a sáv és a szektor címének megadásával lehetséges. Többfelületes meghajtóknál a címben meg kell adni a felület (vagyis a hozzá tartozó író-olvasó fej) sorszámát. Egy-egy felületen 35 vagy 77 sáv helyezkedhet el. Ezeken kívül általában két tartaléksáv is található, és a lemez elején a legelső sáv külön szerepet kap. Ez az ún. indexsáv, és ezen helyezkednek el a lemez tartalmára vonatkozó információk.

A professzionális mikrogépek további térhódításának fontos feltétele a hajlékony lemeztárak kapacitásának növelése és az elérési idő csökkentése.

Kérdések

1. Milyen eszközök segítségével tud a központi egység kapcsolatot tartani a külvilággal?
2. Hogyan csoportosíthatók a perifériák?
3. Mi a kártyaolvasó feladata?
4. Mi az optikai bizonylatolvasás lényege?
5. Milyen elven működő sornyomtatókat ismersz?
6. Milyen előnyei vannak a mikrofilmes outputnak?
7. Milyen hátrányai vannak a mikrofilmes outputnak?
8. Milyen célt szolgálnak a terminálok?
9. Mi a konzol funkciója?
10. Milyen célokra használhatók a rajzolóberendezések?
11. Milyen háttértárakat ismersz?
12. Milyen írássűrűségek lehetségesek mágnesszalagoknál?
13. Mitől függ a mágnesszalagra felvihető bájtok mennyisége?
14. Mi a blokk?
15. Mi a blokkolt adattárolás előnye a blokkolatlanhoz képest?
16. Mit tartalmaz az adathordozói címke?
17. Mit tartalmaz a fájl bevezető címke?
18. Hogyan épül fel a mágneslemez?
19. Mi a sáv, és mi a cilinder?
20. Hogyan épül fel egy sáv?
21. Milyen címkéket ismersz a mágneslemezeken?
22. Milyen perifériák találhatók a mikrogépeknél?

A háttértárolók fontos szerepet játszanak a nagytömegű adattárolásban. Ezeknek az adatoknak a feldolgozása előtt sokszor szükséges az adatok helyességének ellenőrzése. (Az adatok hibája adódhat a rögzítés vagy az adatfelvétel hibájából.)

Készítsetek adatellenőrzési algoritmusokat a feladatok alapján.

1. Egy feldolgozáskor a következő rekordfelépítésű adatállományt kell ellenőrizni. Az állomány a 21. fájlban van.

Azonosító (5 jegyű szám)
Születési év
Nem (1 — férfi, 2 — nő)
Legmagasabb iskolai végzettség:
1 — nyolc általánosnál kevesebb
2 — nyolc általános iskola
3 — középiskola
4 — szakmunkásképző
5 — főiskola, egyetem
Családi állapot:
1 — nőtlen
2 — hajadon
3 — házas
4 — elvált
5 — özvegy

Készíts algoritmust, amely ellenőrzi, hogy a rekordokban levő szavak nem hibásak-e. A jó rekordokat írd fel a 12 fájlba, a hibásakat nyomtasd ki sornyomtatón!

2. Oldd meg az 1. feladatot úgy, hogy a nyomtatóra kiírásra kerülő hibás rekordokhoz tartozzon információ a hiba jellegére vagy helyére vonatkozóan!

3. Oldd meg az 1. feladatot úgy, hogy nem csak az egyes adatok jóságát ellenőrzöd, hanem megvizsgálod, hogy az adatok ne legyenek ellentmondásosak.

A következő eseteket vizsgáld:

a) nem — családi állapot

(ha a vizsgált személy férfi, nem lehet hajadon, ha a vizsgált személy nő, nem lehet nőtlen)

b) életkor — családi állapot

16 évnél fiatalabb fiú és 14 évnél fiatalabb lány nem lehet házas, elvált, özvegy

c) életkor — iskolai végzettség

14 évesnél fiatalabb csak 1 lehet, 18 évesnél fiatalabb csak 1, 2, 4 lehet
22 évesnél fiatalabb nem lehet 5

A hibás adatrekordok kerüljenek a 12 fájlba.

4. Egy gyár termékeiről a következő adatokat gyűjtötték össze:

Termékkód
Anyagköltség

Bérköltség
Egyéb költségek
Költségek összesen
Gyártott mennyiség
Eladási ár (egységnyi mennyiségre)
Haszonkulcs%

Egy hónap adatait a leírt rekordképpel a 20. adatállományba vitték fel. Ellenőrizni kell a rekordok hibátlanságát. Szempontok:

költségek összesen = anyagköltség + bérköltség + egyéb költség

$5\% \leq \text{haszonkulcs} \leq 15\%$

$$\text{haszonkulcs} = \frac{\text{eladási ár} \times \text{gyártott mennyiség}}{\text{összes költség}}$$

$\pm 0{,}5\%$ eltérés megengedett a számított és a megadott haszonkulcs között.

Készíts algoritmust, amely a hibás rekordokat kilistázza sornyomtatóra, a hibátlanokat felírja a 10. fájlba.

5. Oldd meg a 4. feladatot oly módon, hogy a hibalistában legyen információ a hiba jellegére vonatkozóan is. Ha egy rekord több szempontból hibás, ez is derüljön ki.

VIII. FEJEZET

Fájlszerkezetek

A háttértárak megjelenésével újabb lehetőségek nyíltak meg a számítógépek alkalmazásának területén. Lehetővé vált nagy tömegű adatoknak a rendszeren belüli tárolása. A háttértárakban elhelyezett adatok visszakeresése is meggyorsult: az adatokat, adatfájlokat hordozó mágnesszalagnak vagy mágneslemeznek a meghajtóra való felhelyezése után a rendszer nagy gyorsasággal és automatikusan képes megtalálni a programban kért fájlt vagy fájlokat.

Az eddigiek során megismerkedtünk a háttértárak fizikai felépítésével, a mágnesszalagon és mágneslemezen való adatelhelyezéssel. Tudjuk, hogy míg a mágnesszalag csak arra alkalmas, hogy az oda felírt rekordokat meghatározott sorrendben olvassuk vissza, addig a mágneslemez olyan hardver mechanizmussal és címzési rendszerrel rendelkezik, amely közvetlen elérést is képes biztosítani.

Nem beszéltünk azonban arról, hogy a különböző tárolók fizikai szerkezete hogyan befolyásolja a tárolni kívánt rekordok, fájlok belső szerkezetét. Nem vizsgáltuk meg, hogy a hardver tárolók által nyújtott elérési lehetőségek kihasználásához milyen esetleges további információk tárolása szükséges az alapadatokon túlmenően egy-egy fájlban. És fordítva: egy, a feldolgozás szempontjából kívánatos logikai fájlszerkezet hogyan realizálható a fizikai hordozóeszközökön? Alapvető problémánk tehát az, hogy **a különböző adatfeldolgozási feladatok által kívánt fájlszerkezetek és a rendelkezésre álló háttértároló szerkezetek hogyan képezhetők le egymásra.** Ez a kérdés azért is fontos, mert amennyiben ezt a leképzést nem sikerül megteremteni, akkor bizonyos feladatmegoldások lehetetlenné válnak, vagy csak nagyon rossz hatásfokkal (pl. nagyon nagy gépidőráfordítással) oldhatók meg.

Gondoljunk csak arra, hogy nyilvánvalóan másképpen kell megvalósítani annak a fájlnak a belső szerkezetét, amelyben programokat kívánunk tárolni, és szükség esetén egy egész programot egyszerre elővenni, mint egy olyan fájlét, amelyből a rekordokat direkt módon kívánjuk elérni, miközben mindkét fájl mágneslemezen helyezkedik el. Ismét más módon kell megvalósítani egy olyan fájlt, amelynek rekordjait egymás után kívánjuk feldolgozni.

A különböző feldolgozási céloknak eleget tevő fájlszerkezetek fizikai tárolókon való elhelyezésének kérdéseivel a **fájlszervezés foglalkozik**. A fájlszervezés a számítástudománynak az a területe, amely a következő két alapvető kérdés megválaszolásával foglalkozik:

Hogyan lehet egy konkrét fájlszerkezetet fizikai tárolón megvalósítani (tárolni)?
Hogyan lehet egy ilyen fájl rekordjait elérni (feldolgozás céljából)?

Az eddigiekből jól látható, hogy egész gondolatmenetünk a fájlszerkezet fogalma körül mozog. Mindenekelőtt tehát azt kell tisztáznunk, hogy milyen fájlszerkezetek léteznek. Ezt követően néhány jellemző fájlszerkezettel ismerkedünk meg.

A fájlszerkezetek két nagy csoportot alkotnak:

— egyszerű (vagy alapvető) fájlszerkezetek,

— komplex fájlszerkezetek.

Az *egyszerű fájlszerkezetek* esetében a fájlban való keresés magából a fájl adataiból (azonosító) megoldható.

A *komplex fájlszerkezeteknél* külön információkra (indexekre, mutatókra) van szükség ahhoz, hogy egy-egy rekord helye, ill. maga a rekord megtalálható legyen.

Az egyszerű fájlszerkezetek közül a szekvenciális, a komplex fájlszerkezetek közül az indexelt szekvenciális fájl kerül bemutatásra. Megismerkedünk a particionált (könyvtári) fájlokkal, amelyek a programok tárolásánál játszanak fontos szerepet.

⊙ 297 1

A SZEKVENCIÁLIS FÁJL

Egy szekvenciálisan szervezett adatállományban a rekordok azonosítóra (kulcsra) rendezetten követik egymást. A rendezés általában növekvő sorrendű rendezést jelent (az állomány elején a legkisebb, az állomány végén a legnagyobb kulcsú rekord található). Elképzelhető fordított sorrendű (kulcs szerint csökkenő) rendezettség is. A továbbiakban mindig feltételezzük, hogy *a rekordok kulcs szerint növekvő sorrendben* helyezkednek el.

A szekvenciális állománynak ebből a tulajdonságából következik, hogy bármelyik rekord azonosítójának ismeretében megmondható, hogy egy másik azonosítójú rekord az állományban hol keresendő. Ha pl. $REK_A < REK_B$ (ahol REK_A az A rekord azonosítója, REK_B pedig a B rekord azonosítója) és az állományban az A rekordnál tart a keresés (vagy feldolgozás), biztos, hogy a B rekord az állományban hátrább fog előfordulni (feltéve, hogy létezik).

Az elmondottakat úgy is megfogalmazhatjuk, hogy a *rekord kulcsa* és a *rekord* állományban *levő helye között logikai kapcsolat*, objektív összefüggés van, ami így foglalható össze:

egy rekord helyét a megelőző és az őt követő rekord határozza meg. Pl. a 197-es azonosítójú rekord a 196-os mögött és a 198-as előtt kell, hogy legyen (feltéve, hogy van 196, ill. 198 azonosítójú rekord).

A szekvenciális állományok feldolgozásánál éppen ezt a tulajdonságot használjuk ki, amint azt a következő példa is szemlélteti.

A feladat legyen a következő:

Egy mágnesszalagon levő fájl valamely iskola tanulóiról tartalmaz kimutatást.

Minden tanulóról egy rekord van a fájlban a következő rekordképpel:

— a tanuló azonosítója (5 jegyű szám);

— a tanuló neve;

— a tanuló családjában az egy főre eső havi jövedelem Ft-ban;

— a tanuló születési adatai (év, hó, nap).

Az állomány az előző tanévben érvényes adatokat tartalmazza. A rekordok az azonosító növekvő sorrendjében követik egymást. Miután nem tudjuk, hogy hány tanuló (rekord) van, ezért az utolsó rekordot „fájl vége" jel követi. | ▲ | 293 | 2 |

Az új tanév elején aktualizálni kell a nyilvántartást. Egyrészt lehetnek tanulók, akik más iskolában folytatják tanulmányaikat, ezeket törölni kell a nyilvántartásból. Másrészt jöhettek új tanulók, ezeket fel kell venni a nyilvántartásba. Végül biztosan voltak olyanok, akiknél megváltozott az egy főre eső havi jövedelem, azt át kell vezetni a nyilvántartásban.

A változásokat egy kártyafájl tartalmazza a következő rekordképpel:

— a tanuló azonosítója (új tanuló esetében az azonosító nem egyezhet meg már meglevő azonosítóval);

— a tanuló neve;

— a tanuló családjában az egy főre jutó havi jövedelem Ft-ban:

(ha a tanuló más iskolába ment, akkor ez és a következő adatmező üres,

ha „régi" tanulónál bekövetkezett változásról van szó, itt az új jövedelem összege szerepel);

— a tanuló születési adatai (csak akkor van kitöltve, ha új tanulóról van szó).

A kártyafájlban a rekordok a tanuló azonosítója szerint növekvő sorrendben követik egymást.

Készítsük el azt a nyilvántartást, amely a változások utáni állapotot rögzíti.

A feladat megoldásához célszerű végiggondolni néhány dolgot.

A feladatnak két input fájlja van. Az egyik a tavalyi állapotot tükröző (ezt nevezzük *törzsfájlnak*), a másik a változásokat tartalmazó (ezt nevezzük *módosító fájlnak*). A feldolgozás eredménye az *új törzsfájl* lesz. Az egyes állományokra való hivatkozások: legyen a törzsfájl azonosítója 34, az új törzsfájlé 35. A kártyaolvasót az 5 azonosítja. (Lehetne az egyes fájloknak változónév formájú azonosítója is, ezt a tényleges programozási nyelv lehetőségei döntik el. Számunkra ebben a fázisban egy számmal való azonosítás is megfelelő.) A feladat megoldása a következő:

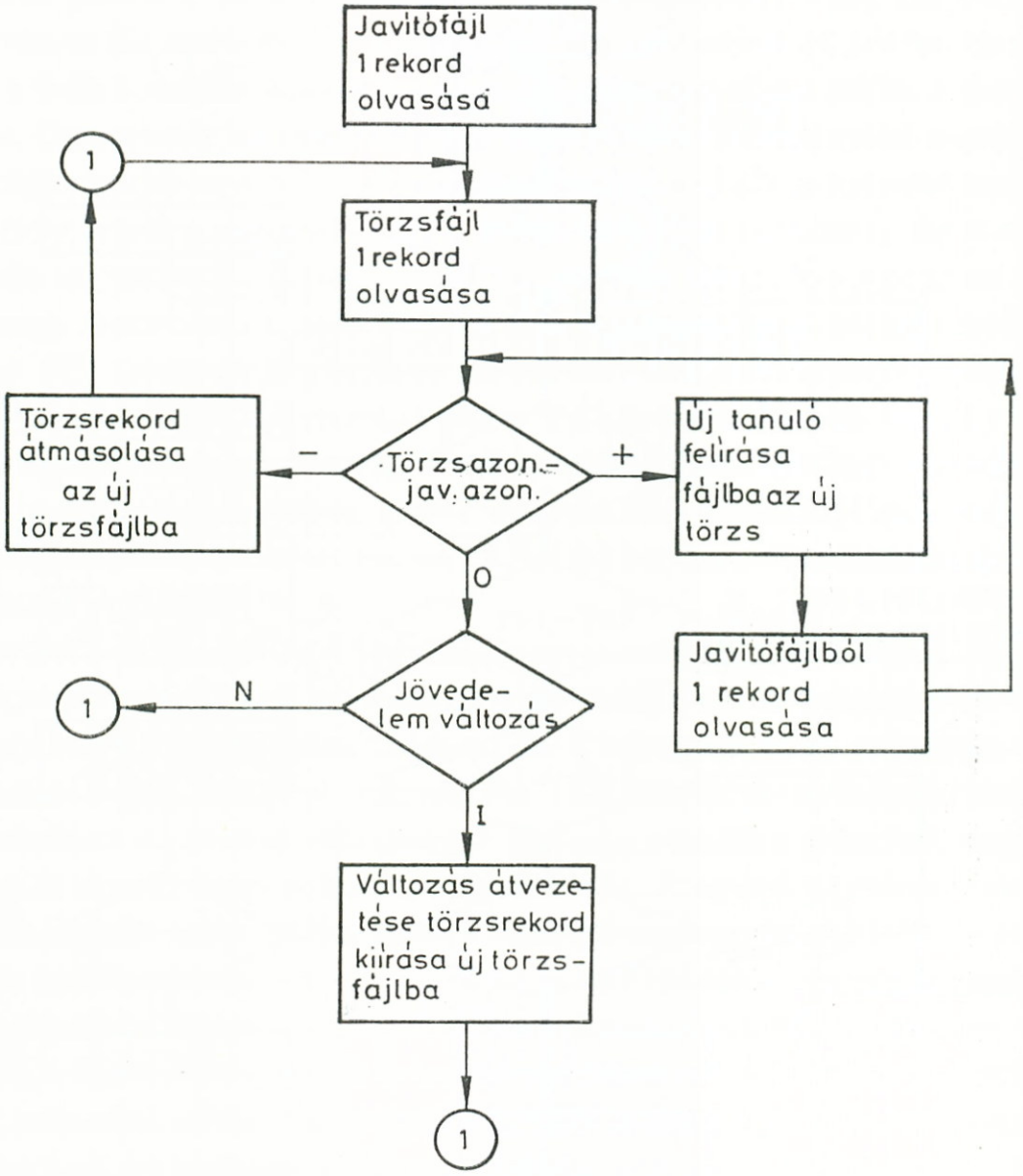
A módosító fájlból beolvasunk egy rekordot. A törzsfájl, miután rendezettek a rekordjai, mindaddig változtatás nélkül másolható, | ▲ | 293 | 3 | amíg kisebb azonosító érkezik onnan, mint a módosító fájlbeli rekord azonosítója. | ▲ | 293 | 4 |

Ha találunk a törzsfájlban egy olyan azonosítójú rekordot, mint a bentlevő módosító rekord azonosítója, el kell dönteni, hogy eltávozott tanulóról vagy pedig jövedelemváltozásról van-e szó. (Ez pl. úgy lehetséges, hogy az eltávozott tanulónál a jövedelem adat értéke 0.)

Ha a törzsfájlból olyan rekord érkezne, amelynek azonosítója nagyobb, mint a bentlevő módosító rekord azonosítója, ez azt jelenti, hogy a „módosító" rekord egy

új tanuló rekordja, akit most kell felírni az új törzsfájlba, mert ekkor fog sorszám szerint a helyére kerülni.

A 77. ábrán — egyelőre még szövegesen fogalmazva — a megoldás közelítő algoritmusát adjuk meg. (A feladat jellegéből következik, hogy minden tanulóhoz legfeljebb egy módosító rekord tartozhat. Ezt a tényt az algoritmus elkészítésekor ki is használjuk.)



77. ábra. A feladat közelítő megoldása

A közelítő megoldásból nyomon követhető a vezérlési szerkezet, de még sok minden hiányzik belőle. Így pl. nem derül ki, hogy az egész eljárás mikor ér véget.

Az új törzsfájl nyilvánvalóan akkor van kész, ha már sem a régi törzs-, sem a módosító fájlban nincsen több feldolgozandó rekord. Ez viszont egy másik problémát vet fel.

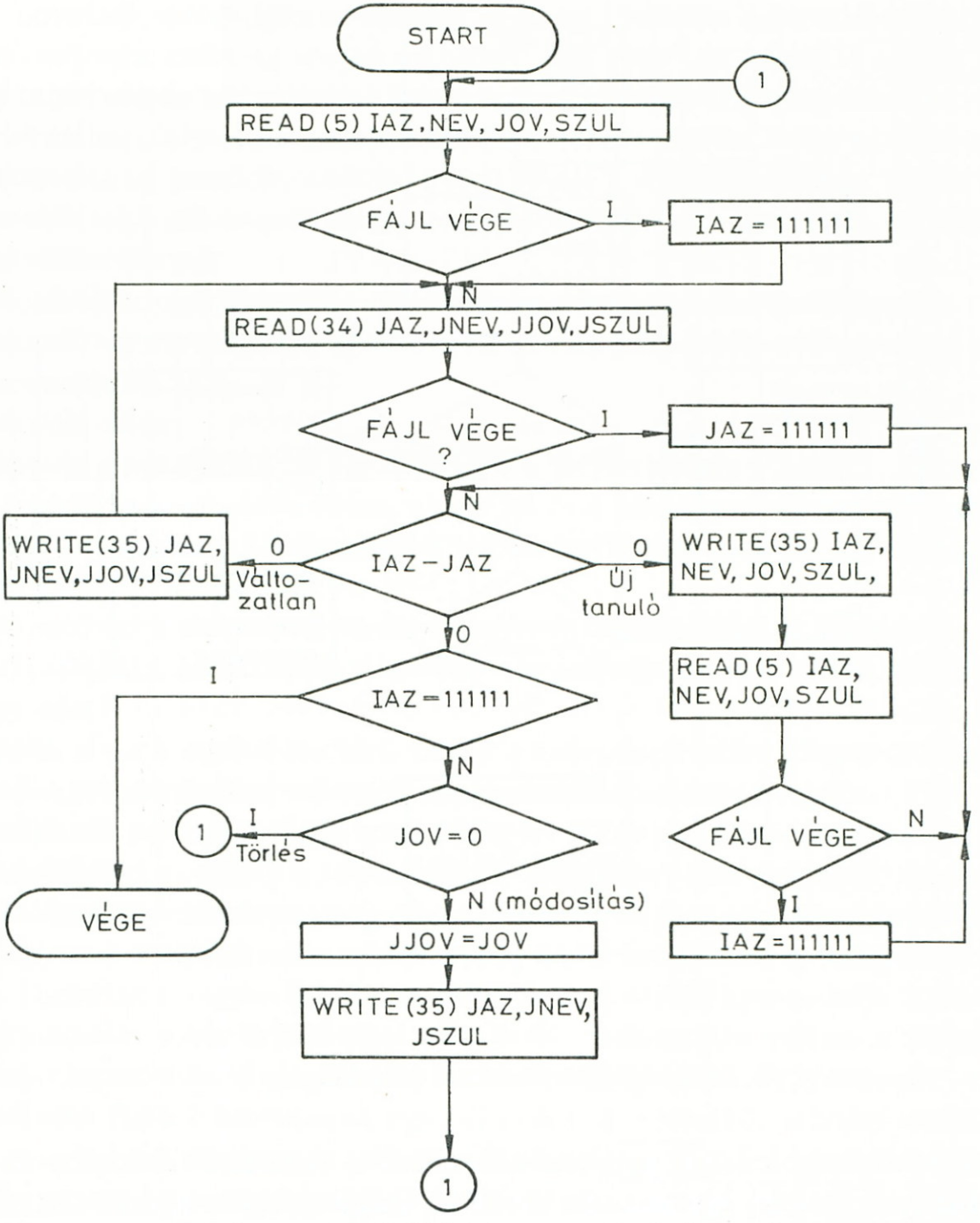
Mi a teendő akkor, amikor az egyik fájlnak a végére értünk, de a másiknak még nem? Könnyen beláthatjuk ui., hogy előfordulhatnak ilyen esetek.

Ha a módosító fájl fogyott el, ez azt jelenti, hogy a régi törzsfájl megmaradt re-

kordjaiban nincs változás. Ha a törzsfájl fogyott el, akkor a módosításként érkező további rekordok már mind csak új tanulókra vonatkozhatnak.

Bármelyik eset áll is fenn, a még meglevő fájl maradék rekordjait át kell másolni az új törzsfájlba.

A megoldás blokkdiagramját a 78. ábra mutatja be.



78. ábra. Fájl felújítási program blokkdiagramja

Amikor bármelyik fájl elfogyott (észleltük a fájl vége jelet), a hozzá tartozó azonosító értékét egy hatjegyű számra állítjuk, amely biztosan nagyobb, mint a másik fájlban még hátralevő azonosítók bármelyike (hiszen azok ötjegyűek). Ezzel biztosítjuk, hogy a két azonosító hasonlításakor mindig arra az ágra adódjék a vezérlés,

amelyen a még létező fájlhoz tartozó rekord feldolgozása történik. Amikor mindkét azonosító ezzel a hatjegyű értékkel rendelkezik, az jelzi, hogy mindkét fájl elfogyott, be lehet fejezni a feldolgozást.

A bemutatott példa a *fájlfelújítás.*

Minden olyan szekvenciális feldolgozásnál, amelynél ismétlődés van, a feldolgozás első lépéseként elő kell állítani az új törzsfájlt. (Ezt megelőzően sok esetben külön munkamenetet jelent a módosító fájlban a kívánt rendezettség megvalósítása. Egy raktári nyilvántartás esetében pl. a módosító fájl rekordjainak sorrendje a keletkezési idő, vagyis a 3-án kiszállított mennyiséget rögzítő rekord megelőzi a 6-án kiszállított mennyiséget, függetlenül az áru kódjától, amely a rekordazonosító szerepét fogja játszani. Feldolgozás előtt a módosító fájlt tehát rendezni kell, azonosító szerint növekvő sorrendre. Erre a célra ugyan léteznek előre megírt programok, de a rendezési idő — különösen nagyméretű fájlok esetén — jelentős lehet. Ha azonban a módosítás a törzsfájl nagy részét érinti, még mindig ez a leggyorsabb és legegyszerűbb feldolgozási módszer. (Pl. tanév elején, amikor minden érettségizett tanulót törölni kell a nyilvántartásból, az új elsősosztályosokat pedig nyilvántartásba kell venni.)

A szekvenciális szerkezet a tárolóval szemben nem támaszt különleges követelményeket — a méretet leszámítva. Ezért szekvenciális állományt akár mágneslemezen, akár mágnesszalagon létre lehet hozni. (A lyukkártyán levő adatokat is szekvenciális fájlként kezeli a rendszer.)

Bemutatott példánkból jól látható, hogy a szekvenciális fájlok felújítása során azokat a rekordokat is át kell másolni az új törzsfájlba, amelyeknél nem volt változás. Ez nem jelent nagyobb problémát akkor, ha a rekordok nagy százaléka változik. De gondoljuk csak végig, hogy pl. egy iskola tanulóinak a nyilvántartásában milyen arányt képviselnek az évközi változások? Nyilván nem az a jellemző, hogy tanév közben sok tanuló elmegy vagy sok új tanuló érkezik. A nyilvántartásnak viszont mindig naprakésznek kell lennie. Előfordulhat tehát, hogy egy vagy két rekord változása miatt a teljes (több száz rekordból álló) fájlt át kell másolni. Ez viszont már korántsem szerencsés megoldás. Ebben és a hasonló esetekben olyan fájlszerkezetre van szükség, amely egyrészt biztosítja a rekordok szekvenciális elérését, másrészt lehetővé teszi, hogy egyes rekordok direkt módon is hozzáférhetők legyenek. Az ilyen igények kielégítésére szolgál az indexelt szekvenciális szervezettségű fájltípus.

Az indexelt szekvenciális fájl egyben jó példa arra, hogy miképpen lehet a fájl belső logikai szerkezetét összehangolni az adathordozó (esetünkben a mágneslemez) fizikai szerkezetével (cilinder-, sávszerkezettel).

AZ INDEXELT SZEKVENCIÁLIS FÁJL

Az **indexelt szekvenciális állomány** (IS állomány) a komplex szerkezetű állományok közé tartozik. Alapvetően két részből áll, egy *indextáblából* és egy **szekvenciális fájból.** Az indextábla tartalmazza azokat az információkat, amelyek az egyes (vagy

csak bizonyos) rekordok fizikai (lemezen levő) címét adják meg. Ez az oka, hogy az IS állományok csak mágneslemezen hozhatók létre. A mágneslemezes IS állomány kialakításakor jól kihasználható az a tény, hogy az azonos cilinderhez tartozó sávok mindegyike hozzáférhető az író-olvasó fejek mozgatása nélkül.

Az állományban a rekordok logikailag megőrzik rendezettségüket, bár nem biztos, hogy a rekordok fizikai sorrendje is megegyezik ezzel.

Mielőtt megismerkedünk az indexelt szekvenciális fájl felépítésével, vizsgáljuk meg e fájlszerkezet logikáját.

Mindaddig, amíg a fájlban csak azok a rekordok vannak, amelyek már a fájl létrehozásakor is léteztek, a rekordok kulcs szerint növekvő sorrendben követik egymást a cilindereken, ill. a sávokon. Ekkor tehát látszólag egy szekvenciális fájllal van dolgunk. A helyzet akkor kezd bonyolultabb lenni, ha új rekordokat kívánunk beiktatni a fájlba. A szekvenciális sorrend megtartása érdekében ezeket az új rekordokat a „helyükre" kell beírni, vagyis oda, ahová a kulcs szerint tartoznak.

Ha az eredeti fájlt úgy hoztuk létre, hogy egy-egy sávnak csak pl. a kétharmadát töltöttük fel, vagyis minden sávon van üres hely, az új rekord bevitele abból áll, hogy a megfelelő sávon az új rekordnál nagyobb kulcsú rekordokat „el kell tolni" és az így keletkezett helyre az új rekordot be kell írni. Ez mindaddig lehetséges megoldás, amíg a hátrább kerülő rekordok még elférnek a sávon. Az első probléma akkor adódik, amikor olyan sávra kell felírni új rekordot, amely tele van. Ilyenkor ui. a sáv legmagasabb kulcsú rekordja az eltolás miatt nem fér fel a sávra: *túlcsordul.* Az indexelt szekvenciális fájl logikájából következően ez a túlcsordult rekord nem kerülhet a következő sávra, hanem egy külön e célra fenntartott területre íródik, amelyet *cilinder-túlcsordulási területnek nevezünk.* Ez a túlcsordulási terület a sávhoz tartozó cilinderen van, de természetesen külön sávon (vagy sávokon). Annak érdekében, hogy a túlcsordult rekordokat is megtaláljuk, amikor olvasunk a fájlból, szükség van olyan információra, amely megmondja, hogy egy sávhoz tartozik túlcsordult rekord. Ez az információ az ún. *sávindexen* található. A sávindex is külön helyen, a cilinder első sávján található. Mindaddig, amíg a cilinder-túlcsordulási területen van hely, újabb probléma nem adódik.

Elképzelhető azonban, hogy egy adott kulcstartományban sok új rekordot akarunk felvinni. Ilyenkor az adott cilinder saját túlcsordulási területe is betelik, vagyis adódnak olyan rekordok, amelyek innen is túlcsordulnak. A különböző cilinderről, ill. azok cilinder-túlcsordulási területeiről kiszoruló rekordok közös „gyűjtőhelye" a *független túlcsordulási terület.* Ide tehát a fájl bármely részéről kerülhet rekord. Természetesen ezeket is valami módon nyilván kell tartani.

A különböző túlcsordulási területeken levő rekordok számának növekedésével lelassul a fájlban történő keresés. Ilyenkor szükségessé válik a fájl *újraszervezése,* ami azt jelenti, hogy a teljes fájl újbóli elhelyezésre kerül, oly módon, hogy ismét minden rekord fizikailag is a helyére kerül, s az egyes sávok csak mintegy kétharmad részig vannak feltöltve. Ezek után nézzük meg, hogy mindez hogyan valósul meg a gyakorlatban.

Egy IS állomány három részből áll. Ezek:

— elsődleges adatterület,

— cilinderindex-terület,

— független túlcsordulási terület.

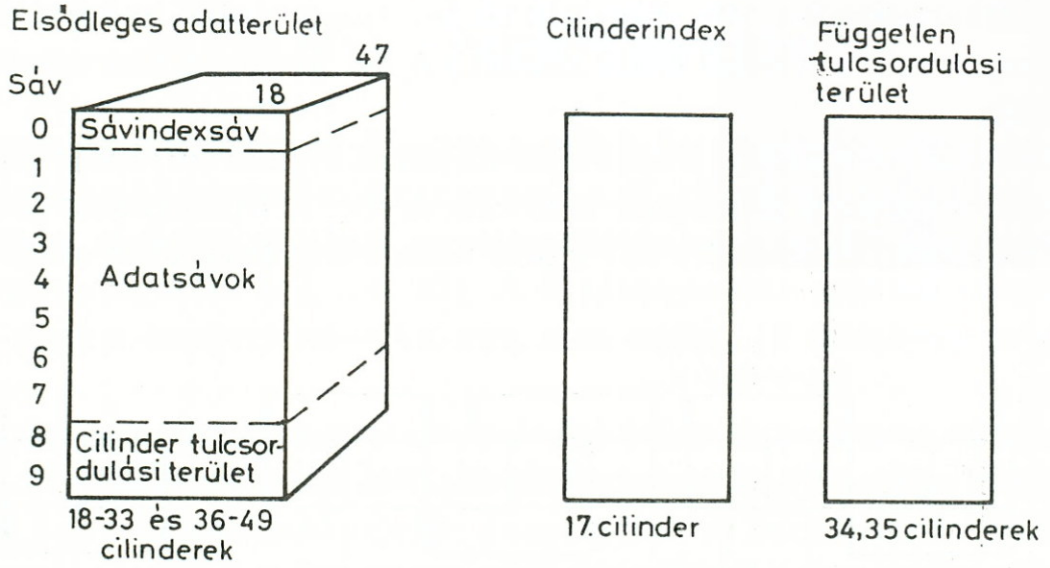
Az elsődleges adatterület valamennyi cilinderén a következő elrendezésűek a sávok:

— adatsávok,

— sávindexsáv,

— cilinder-túlcsordulási terület sávja(i).

A 79. ábra egy olyan IS állomány előbb tárgyalt struktúráját szemlélteti, amely a lemez 17—49 cilinderein helyezkedik el. (A cilindereket téglalapok szimbolizálják.)



79. ábra. IS állomány elhelyezése mágneslemezen

A cilinderindex a 17. cilinderen van, az elsődleges adatterületek a 18.—33. és a 36.—49. cilindereken van. A független túlcsordulási terület a 34. és 35. cilinderen van. ▲ | 293 | 5

Milyen hozzáférési mechanizmust tesz lehetővé az ilyen szerkezet? Erre a kérdésre akkor tudunk válaszolni, ha megvizsgáljuk az egyes területek tartalmát és funkcióját.

Kezdjük az adatsávokkal. Az állomány létrehozásakor ide kerülnek a felhasználói adat rekordok. Fontos feltétel, hogy ezeket a rekordokat előzőleg rendezni kell, kulcs (azonosító) szerint növekvő sorrendben. (Az IS állomány létrehozásának inputja tehát egy szekvenciális állomány.) Az elsődleges adatterület adatsávjaiban a rekordok úgy kerülnek fel, hogy a rekordazonosító, vagyis a kulcs egy külön ún. *kulcsterületre* íródik. ⊙ | 298 | 6

Az IS állományt létrehozó program a kulcsra (azonosítóra) növekvően rendezett rekordokat az egyes sávok mentén írja fel, általában úgy, hogy egy-egy sávra annak kapacitásánál kevesebb rekord kerüljön. ⊙ | 298 | 7 (Amint láttuk, ez később az állománybővítésénél lesz fontos.)

Amikor egy adatsáv írása befejeződik, egy *sávindex rekord* kerül kitöltésre a sávindex sávon (ennek tartalmát később ismerjük meg). A cilinder betelte esetén egy *cilinderindex rekord* kerül rögzítésre a cilinderindex területén.

Az indexterületek a következő formájúak.

Az elsődleges adatterület valamennyi cilinderének 0. sávján (79. ábra) kerül kialakításra egy-egy *sávindex sáv*. A sávindex sávon találhatók a sávindex rekordok.

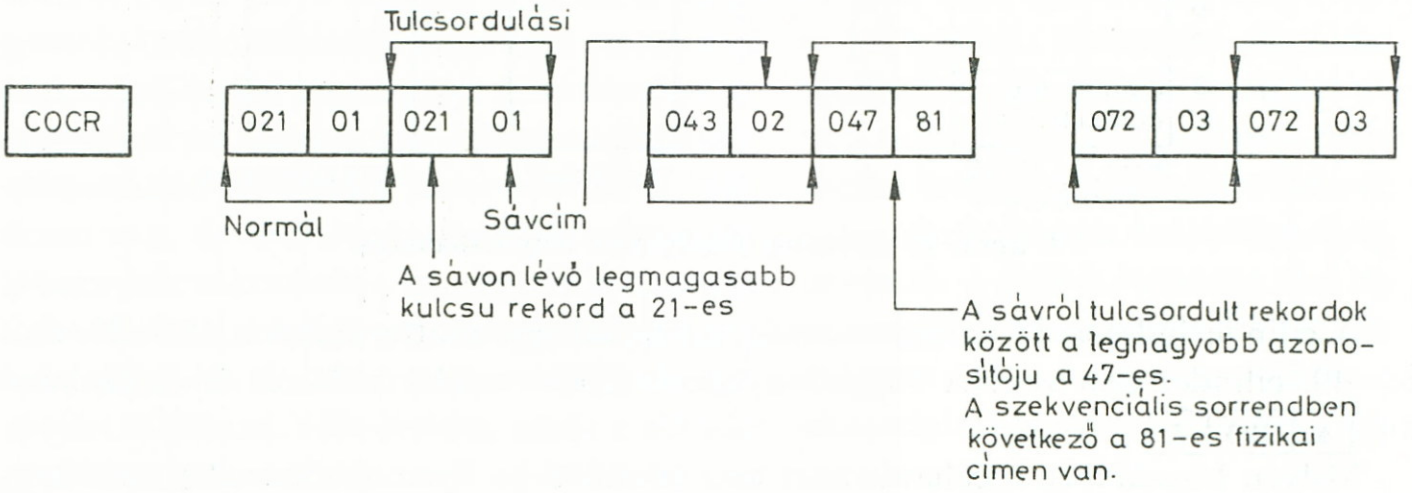
Egy sávra vonatkozóan a sávindex rekord két részből áll:

— a normál index tételből és

— a túlcsordulási index tételből.

A *normál tétel* a sáv címét és a sávon levő legmagasabb kulcsú rekord kulcsát tartalmazza. A *túlcsordulási index tétel* a sávhoz tartozó legmagasabb kulcsú túlcsordult rekord kulcsát, és a sávhoz tartozó legkisebb túlcsordult rekord címét (lemezen levő helyét) tartalmazza. (Ha még nem történt túlcsordulás, akkor a túlcsordulási index tétel megegyezik a normál index tétellel.) A 80. ábrán az elmondottak jól nyomonkövethetők.

A túlcsordulás létrejöttével a rekordok fizikai és logikai sorrendje (növekvő azonosítók szerinti sorrendje) többé már nem esik egybe. A túlcsordulás kezelését a későbbiekben láthatjuk. A sávindex sáv előbb bemutatott felépítését a 80. ábra szemlélteti.

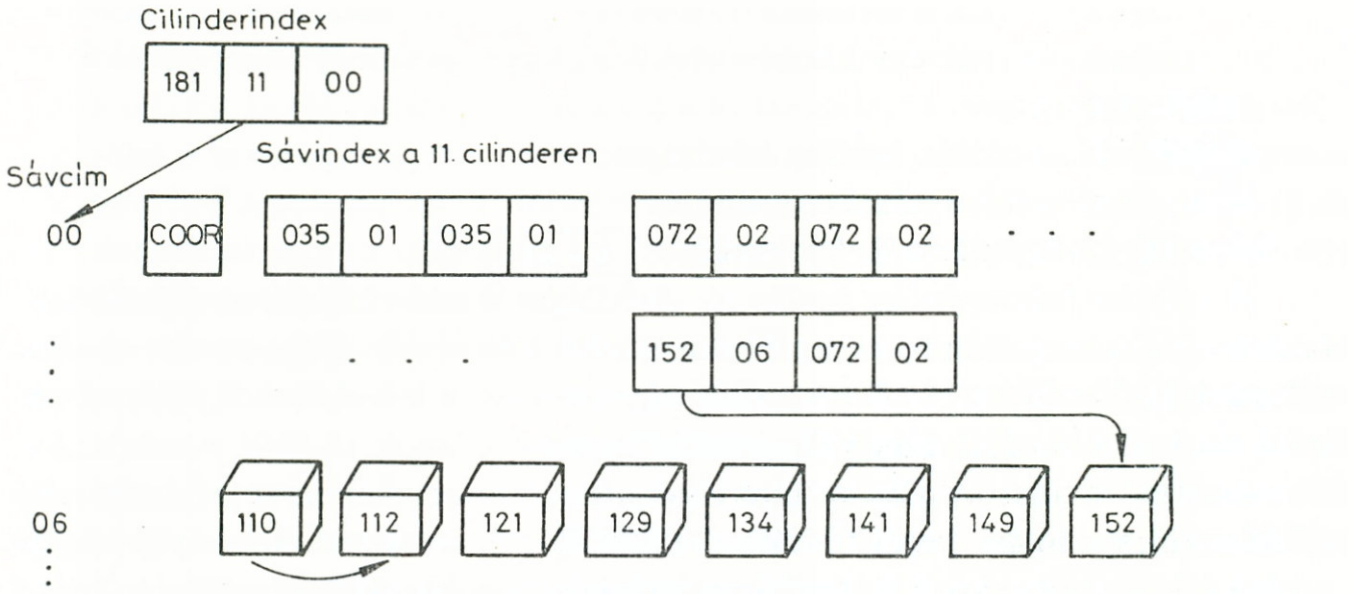


80. ábra. Sávindexsáv

A *cilinderindex* az IS állományhoz tartozó elsődleges adatterületek cilindereiről tartalmaz információkat. Minden cilinderhez, amelyen vannak adatok, tartozik egy cilinderindex. Felépítése hasonló a sávindex normál indextételéhez: rögzíti a cilinderhez tartozó sávindex sáv címét és a cilinderen levő legmagasabb kulcsú rekord azonosítóját.

Az eddig bemutatott struktúra működési mechanizmusát a 81. ábrán követhetjük nyomon, ahol blokkolatlan rekordokból áll az állomány.

A 112 azonosítójú rekordot kívánjuk visszakeresni. A cilinderindex alapján meghatározható, hogy az elsődleges adatterület által elfoglalt cilinderek közül a 11 sorszámucilinderen a legmagasabb kulcs 181. Az ábra nem tünteti fel, de legyen a feltöl-



81. ábra. Rekordok keresése IS állományban

tés olyan, hogy a 10. cilinderen levő legmagasabb kulcs értéke 100. Ily módon (a szekvencialitás miatt) a 112 azonosítójú rekordnak a 11. cilinderen kell lennie. Az így kiválasztott cilinder sávindex sávjából megállapítható, hogy az első olyan sáv, amelyen a kulcs már nagyobb, mint 112, a 6. sáv. A 6. sávot ezután már szekvenciálisan kell végigolvasni, amíg a keresett rekordot meg nem találja. (Esetünkben ez sorrendben a második.)

Feldolgozáskor a cilinderindex tartalmát betöltik a tárba, így az ebben való keresés igen gyors. A cilinderindexből kiolvasott információ alapján tudni lehet, hogy melyik cilinderre kell az író-olvasó fejeket diszponálni. A cilinderen belüli keresés sebessége már csak a lemez forgási sebességétől függ. Ha a keresett rekord nem a független túlcsordulási területen van, akkor csak egyszer kell a lemezhez fordulni.

A túlcsordulási területek és azok szerepe a következők alapján érthető meg.

A 81. ábrán láthatjuk, hogy egy sávon belül a rekordok „levegősen" vannak felírva, vagyis kevesebb rekord van egy-egy sávon, mint amennyit a sáv kapacitása megengedne. Ugyanakkor azt is láthatjuk, hogy nem minden azonosítójú rekord létezik az állományban (pl. a 116 azonosítójú rekord nem létezik). Az állomány azonban gyarapodhat újonnan belépő rekordokkal. (Persze csak olyanokkal, amelyek azonosítója még nem fordult elő az állományban.)

Egy újonnan belépő rekordot a rendszer először megpróbál arra a sávra elhelyezni, ahová a szekvencia sorrend alapján tartozik. Ha ezen a sávon van még hely, beilleszti a rekordot, és a többit — a sávon levő magasabb kulcsú rekordokat — hátrább tolja. (Ezért jó, ha az állomány létrehozásakor nincsenek a sávok teljesen feltöltve.) Ha a sávon nincs már hely, a beillesztés és az eltolás megtörténik, és a legmagasabb kulcsú rekord, amely ily módon túlcsordul, a túlcsordulási területen kerül elhelyezésre.

Vegyük észre, hogy a sávon belül továbbra is növekvő sorrendűek maradnak az ott levő rekordok. A túlcsordulással egyidejűleg korrigálásra kerül az adott sávra vo-

natkozó sávindex rekord tartalma is: megváltozik a normál tételekben a legmagasabb kulcs (hiszen most már egy kisebb értékű a „legmagasabb") és módosul a túlcsordulási tétel tartalma is.

Kétféle túlcsordulási terület lehetséges:

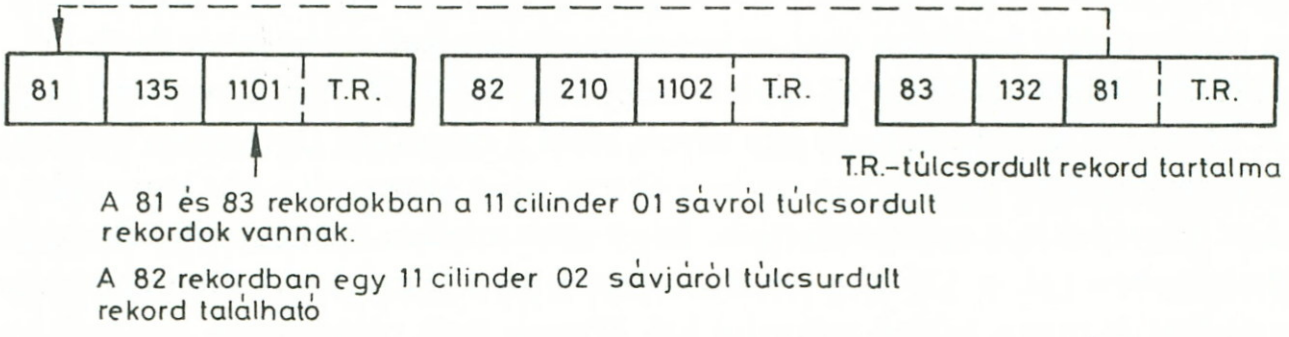
— a cilinder-túlcsordulási terület és

— a független túlcsordulási terület. | ▲ | 294 | 8 |

A *cilinder-túlcsordulási terület* az elsődleges adatterület része (79. ábra). Általában 1—2 sáv terjedelmű, és a cilinder utolsó két sávján helyezkedik el. A cilinder-túlcsordulási területen levő rekordok ugyanazzal az író-olvasó fejállással elérhetők, mint azok a rekordok, amelyek „között" a helyük lenne.

A cilinder-túlcsordulási területre vonatkozó információkat a sávindex elején levő cilinder-túlcsordulást vezérlő rekord (COCR) tartalmazza. Itt kerül rögzítésre az utolsó túlcsordult rekord fizikai címe, valamint a túlcsordulási területen még fel nem használt bájtok száma.

A *független túlcsordulási terület* független az elsődleges adatterülettől. (Fizikailag is más cilinderen található.) Ide bármelyik cilinderről kerülhetnek rekordok. Mivel e terület eléréséhez külön fejmozgás szükséges, célszerű, ha helyét nem az elsődleges adatterületek cilinderei után, hanem valahol közben jelölik ki. Pl. ha az elsődleges adatterület a 18—49. cilindereken van, a független túlcsordulási területet a 34—35. cilinderek tájékán érdemes kijelölni, mert ide az állomány bármely részéről átlagosan rövidebb fejmozgási úttal lehet elérni. (A 79. ábrán is ezért van így.) A túlcsordulási területeken a rekordok egymáshoz láncolva helyezkednek el, ahogyan azt cilinder-túlcsordulási területen levő rekordok esetén a 82. ábra szemlélteti.



82. ábra. Rekordok a túlcsordulási területen

A számláló a fizikai címet adja. Az azonosító a túlcsordult rekord kulcsa. A láncmező az azonos sávról előzőleg túlcsordult rekord számlálójára mutat. (A túlcsordulási területre a belépés a túlcsordulási index tételben levő címen, vagyis a sávról utoljára túlcsordult rekord címén lehetséges. Innen a rekordok „visszafelé" vannak láncolva, így minden rekord láncmezőjében a nagyságrendileg nála nagyobb kulcsú túlcsordult rekord címe található. A legelső rekord láncmezője arra a sávra mutat vissza, amelyről a rekord túlcsordult. (A cilinder-túlcsordulási területen annyi lánc alakulhat ki, ahány elsődleges adatsáv létezik.)

Mindaddig, amíg az újonnan belépő rekordok nem csordulnak túl, ill. amíg csak a cilinder-túlcsordulási területre töltődnek, a keresett rekordok direkt elérése gyors. Minél több rekord kerül azonban a túlcsordulási területekre, annál gyakrabban válik bonyolultabbá a keresés, vagyis a rekordok elérése lelassul. Ezért az IS állományokat időnként *újra kell szervezni*. Ez a művelet a túlcsordult rekordokat visszaviszi az elsődleges adatterületre, újra elkészítve az IS állományt. Ezzel a művelettel ismét egy gyorsan feldolgozható állomány áll elő.

Indexelt szekvenciális állományt általában akkor érdemes használni, ha a direkt módon elérni kívánt rekordok száma nem haladja meg a teljes állományban levő rekordok számának 10%-át, és az állományt szekvenciálisan is fel kell dolgozni. A már korábban bemutatott iskolai nyilvántartás pl. jól megvalósítható indexelt szekvenciális módon, mert az elmondott feltételek teljesülnek. Ahol nagy tömegű közvetlen elérést kell biztosítani, ott más fájlszerkezet kialakítása szükséges. | ● | 294 | 9 |

A KÖNYVTÁRI FÁJL

Az eddig vizsgált fájlszerkezetek közös jellemzője, hogy rekordjaikban adatok kerülnek elhelyezésre. Tudjuk azonban, hogy a számítógépben nemcsak adatokat, hanem programokat is kell tárolni, amikor végrehajtási céllal nincsenek a központi tárban. Emlékezzünk csak vissza a fordítóprogram működésére. Magát a fordítóprogramot is be kell tölteni a működéshez egy tárolóhelyről, és a fordítás outputját — a tárgyprogramot — is tárolni kell mindaddig, amíg a kapcsolatszerkesztőnek nincs szüksége rá. A programok tárolására kijelölt helyeket **könyvtáraknak** nevezzük. A könyvtár egy speciális elérési lehetőséggel rendelkező állomány, éppen ezért csak mágneslemezen hozható létre. Milyen tárolási szerkezet célszerű akkor, amikor programokat akarunk tárolni és visszakeresni?

Programok tárolására olyan szerkezet célszerű, amely

— biztosítja, hogy egy-egy program eleje (első rekordja) közvetlenül elérhető legyen;

— a program rekordjai (utasításai) ettől kezdődően szekvenciálisan legyenek elérhetők (a rögzítés sorrendjében).

A könyvtári állományok ilyen szervezettségűek. Egy könyvtári állomány *két részből* áll:

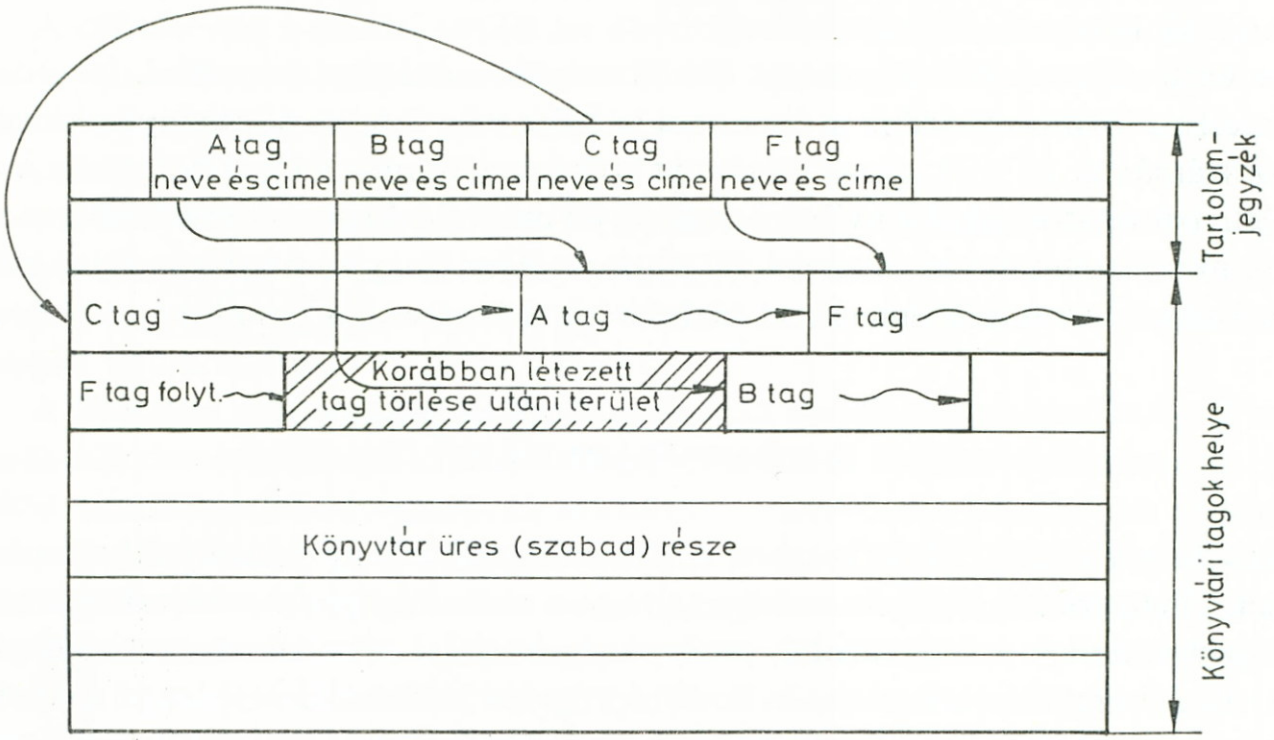
— tartalomjegyzékből és

— sorosan szervezett tagokból.

A *tagok* szekvenciális szerkezetű elemei a könyvtári állománynak. Egy-egy tag egy-egy program tárolására alkalmas. Az egyes tagoknak az állományon belüli helyéről és méretéről a *tartalomjegyzék* tartalmaz információkat. A tartalomjegyzékben (néhány más információn kívül) megtalálható a tag neve (legfeljebb 8 karakter) az első rekordjának relatív címe, a tag mérete félszavakban mérve. | ▲ | 294 | 10 | A cím

SSR formájú, ahol SS a tag első rekordját tartalmazó sáv sorszáma az állomány elejétől számítva, R a tag első rekodjának sorszáma a sávon belül.

A tartalomjegyzék 256 bájtos adatmezőkből épül fel, az egyes tagok abc sorrendben szerepelnek a tartalomjegyzékben. A 83. ábra egy könyvtári állomány szerkezetét szemlélteti.



83. ábra. Könyvtári állomány szerkezete

A könyvtári fájl szerkezetének megismerése után felmerül a kérdés, hogyan csoportosíthatók a könyvtárak és milyen célokra használhatók?

Egy számítógépes rendszerben levő könyvtárak többféle szempontból csoportosíthatók. A *könyvtárak hatóköre* szempontjából megkülönböztetünk:

— rendszerkönyvtárakat,
— felhasználói könyvtárakat,
— ideiglenes könyvtárakat.

A *rendszerkönyvtárak* az operációs rendszerhez tartozó programok tárolására szolgálnak. (Nevük általában SYS karaktersorozattal kezdődik.) Ezeket a könyvtárakat a felhasználói programok is igénybe vehetik. Pl. a FORTRAN nyelvű programokhoz kiegészítő rutinok szükségesek a SYS1.FORTLIB nevű könyvtárból. A feldolgozó programok (fordítóprogramok, szerkesztők stb.) a SYS1.LINKLIB nevű könyvtárban találhatók.

A rendszerkönyvtárak között található a SYSCTLG nevű könyvtár is. Ez az ún. *rendszerkatalógus*. A rendszerkatalógusba egy állomány katalogizálása során bejegyzésre kerül az állomány neve és annak a mágneslemeznek az azonosítója, amelyen az állomány elhelyezkedik. A katalogizált állományokra nevük megadásával elég hivatkozni. A rendszer a megtaláláshoz szükséges többi információt a katalógusból

veszi ki. (A katalogizálást erre a célra szolgáló program végzi.) A rendszerkönyvtárakat bármelyik felhasználó olvasási szándékkal igénybe veheti. (A rendszerkatalógusba való írást általában az üzemeltetők korlátozzák.)

A *felhasználói könyvtárak* olyan könyvtári állományok, amelyeket a felhasználó saját programjainak tárolása céljából hozhat létre. Ebből következően használati köre szűkebb, a hozzáféréséhez szükséges információk csak az érintett felhasználóknak állnak rendelkezésére.

Korszerű, terminálos rendszereknél a felhasználói könyvtárak alapvető szerepet játszanak: a felhasználónak minden programja könyvtárban van, onnan van mód elővenni, javítani, elküldeni feldolgozásra. A könyvtárból elővett programok (egy munkaterületről, amely szintén mágneslemezen van) megjeleníthetők a terminál képernyőjén.

Az ideiglenes könyvtár egy-egy program (vagy néhány, futáskor összekapcsolt program) rendelkezésére álló, annak futási ideje alatt létező könyvtár. Így csak az érintett programok számára férhető hozzá. A program(ok) lefutása után a könyvtár törlésre kerül. Elsősorban munkaterület jellegű, olyan állományok számára, amelyek igénylik, hogy könyvtári állományként létezzenek. (Pl. a kapcsolatszerkesztő outputja.)

Egy *másik felosztás* alapján beszélhetünk:

— forrásnyelvi programokat,

— tárgynyelvi programokat,

— futtatható (betölthető) programokat

tartalmazó könyvtárakról.

Az ilyen megkülönböztetést az teszi szükségessé, hogy a háromféle könyvtár a felhasználótól nem azonos kezelést kíván.

A *forrásnyelvi könyvtárakban* levő programok valamilyen magas szintű programozási nyelven kerültek megírásra. Ilyen könyvtárakat elsősorban programfejlesztési célokból célszerű létrehozni. Jól szervezett számítóközpontokban minden fontosabb programnak rendelkezésére áll a forrásnyelvi változata. Ez a biztosítéka annak, hogy szükség esetén bármikor javításokat vagy változtatásokat lehessen végezni egy programon. Terminálos rendszereknél a felhasználók könyvtári forrásnyelvi könyvtárak.

A *tárgynyelvi* (vagy más néven előszerkesztett) könyvtárakban lefordított, de futásra össze nem szerkesztett alprogramok találhatók. Az ilyen programok létjogosultságát az adja, hogy a felhasználók sokszor alkalmaznak programjaikban már elkészített és kipróbált alprogramokat, amelyek egy-egy logikailag önálló algoritmusrész megoldását szolgáltatják. Az ilyen alprogramnak csak forrásnyelven való tárolása állandó újrafordítást kíván, ami sok felesleges gépidőt visz el. A már lefordított alprogramok a futtatás előkészítésének második fázisában (a szerkesztés folyamatában) kapcsolódnak a hívó programhoz. (Természetesen célszerű, ha a tárgyprogramok forrásnyelvi változata megvan egy másik könyvtárban. Erre utaltunk az előbb, amikor a programfejlesztésről volt szó.) ⊙ | 298 | 11

A *betölthető programok* könyvtára futásra kész, gépi kód szintű programokat

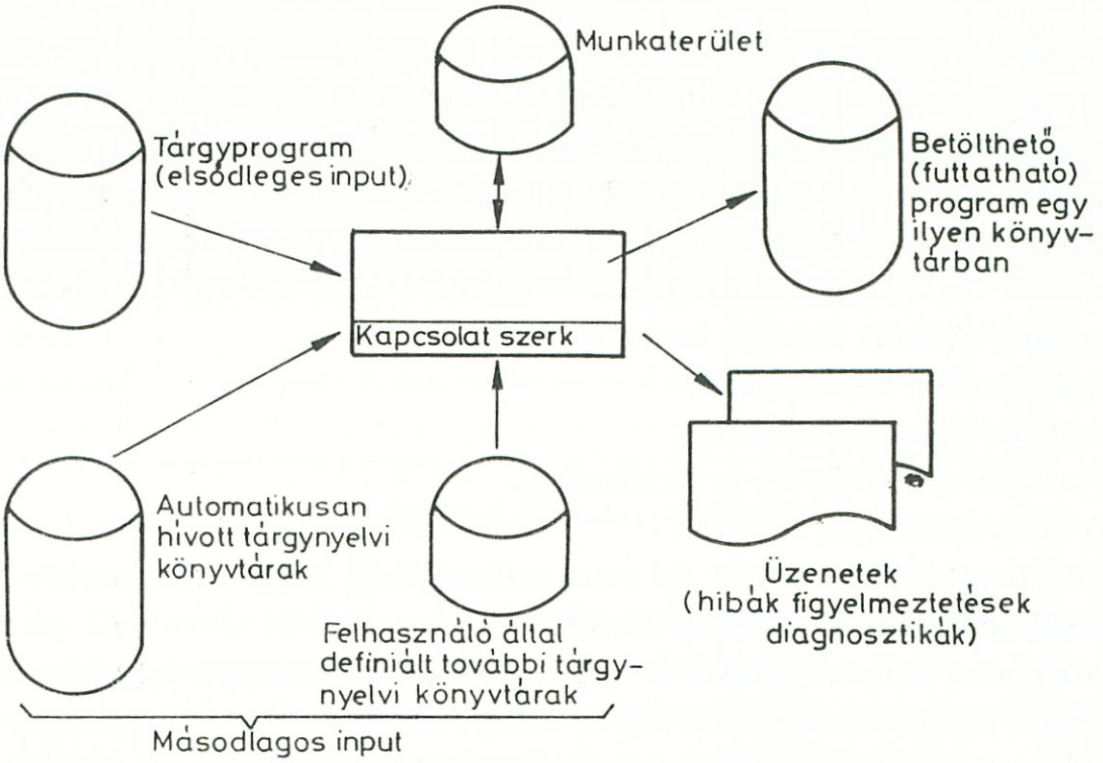
tartalmaz. Ha a felhasználó ilyen könyvtárból akar programot futtatni, nincs más tennivalója, mint

— elő kell készítenie a feldolgozandó adatait (olyan formában, ahogy ezt a program kéri);

— megfelelő módon meg kell nyitni a könyvtárat, és futásra meg kell hívni a kívánt programot. ⊙ | 298 | 12

Az ilyen könyvtárak használata a gyakran ismétlődő programok esetében igen hatékony gépidő-felhasználást eredményezhet.

A könyvtárakról elmondottak illusztrálására a 84. ábrán bemutatjuk a kapcsolatszerkesztő program működése során használt könyvtárakat, és azok logikai kapcsolatát.



84. ábra. A kapcsolatszerkesztő input-output könyvtárai

Az eddigiek során a fájlszerkezetek közül néhányat kiválasztva a fizikai tárolón való megvalósítás kérdéseit vizsgáltuk. Nem foglalkoztunk azonban azzal a kérdéssel, hogy *hogyan lehet a fájlok rekordjait feldolgozás céljából elérni.*

A kérdés megfogalmazása (feldolgozandó rekord) adatfájlokra vonatkozik, ezért a könyvtári állományokat hagyjuk figyelmen kívül.

A szekvenciális fájlok esetében való rekordelérést (olvasást) a szekvenciális fájlnál bemutatott példán láthattuk. Bármely programozási nyelvből lehetőség van arra, hogy egy szekvenciális adatfájlt kezelni tudjunk.

Az indexelt szekvenciális fájlból való olvasás már nem ilyen egyszerű, hiszen gondoljunk csak arra, hogy a rekord lehet az adatsávon, a cilinder-túlcsordulási területen vagy akár a független túlcsordulási területen. Ennek a kérésnek a programozása igen nagy feladat. Éppen ezért indexelt szekvenciális fájlt nem is lehet bármely

programnyelvből kezelni. Ezek kezeléséhez ui. *külön szoftver támogatás szükséges.* A magas szintű programozási nyelvek közül pl. a COBOL és a PL/1 nyelvekben ez a lehetőség megvan, ezért ezeken a nyelveken írott programokból az IS fájlok kezelése könnyen megoldható.

A következő fejezetben látni fogjuk, hogy tulajdonképpen *bármilyen típusú fájlról* is van szó, az ahhoz való *hozzáférés mindig külön szoftver támogatást igényel.* Ezt a szoftver támogatást az operációs rendszer *adatkezelő programjai* végzik el. Az ún. *elérési módok* biztosítják az egyes fájlszerkezetek használatát. ● 295 13

Kérdések

1. Mi jellemzi a szekvenciális fájlt?
2. Milyen adathordozón hozható létre szekvenciális fájl?
3. Mi a lényegi különbség az egyszerű és a komplex fájlszerkezetek között?
4. Hogyan lehet mágnesszalagon levő szekvenciális fájlt karbantartani?
5. Milyen elérést biztosít az indexelt szekvenciális fájl?
6. Milyen adathordozón hozható létre indexelt szekvenciális fájl?
7. Mi az indexelt szekvenciális szerkezet lényege?
8. Milyen területek léteznek egy indexelt szekvenciális fájl esetén?
9. Hányféle indexterületet ismersz?
10. Milyen információkat tartalmaz a sávindex sáv?
11. Mikor keletkezik túlcsordulás?
12. Milyen túlcsordulási területeket ismersz?
13. Hol célszerű elhelyezni a független túlcsordulási területet?
14. Milyen célt szolgál a könyvtári állomány?
15. Milyen adathordozón hozható létre könyvtári állomány?
16. Mit nevezünk tagnak?
17. Milyen információkat tartalmaz a könyvtári állomány tartalomjegyzéke?

Feladatok

A szekvenciális fájlok feldolgozásához kapcsolódnak a következő feladatok:

1. Egy vállalat által gyártásra tervezett termékekről a következő rekordfelépítésű szekvenciális fájl áll rendelkezésre:
   — Azonosító
   — Termékegységre jutó anyagköltség
   — Termékegységre jutó bérköltség
   — Termékegységre jutó egyéb költség
   — Tervezett termelés
   — Tervezett eladási ár

A fájl rendezett, végén fájl vége rekord van. A gyártás megkezdése előtt kiderült, hogy az eredeti tervhez képest változások vannak: egyes termékeket nem lehet gyártásba venni, másoknál megváltozik a költség vagy az ár, esetleg a tervezett termelés.

A változásokat egy kártyafájl tartalmazza, amelynek rekordfelépítése a következő:

— Azonosító
— Változást igénylő adatok száma
— A változtatni kívánt első adat sorszáma a rekordon belül
— Az új érték
— A változtatni kívánt második adat sorszáma a rekordon belül
— Az új érték
...

(Legfeljebb öt adat változhat, hiszen hateleműű a rekord és az azonosító nem változhat.)

Ha olyan rekordról van szó, amelyet törölni kell, az azonosítót követő szó tartalma 0.

A kártyafájl azonosítóra rendezett, végén fájl vége jel van.

Készítsd el az új törzsfájlt előállító algoritmust.

2. Az iskola számítástechnikai szakkörének tagjait számítógéppel tartják nyilván. A nyilvántartást tartalmazó szekvenciális fájl rekordképe a következő:

— Azonosító
— Diák neve
— Belépés dátuma
— Lakcíme (város, utca, házszám)

A tanév elején a nyilvántartást fel kell újítani. A kilépetteket törölni kell, az új belépőket nyilvántartásba kell venni.

A módosító fájl rekordképe megegyezik a törzsfájl rekordképével, ha valaki kilépett, csak az azonosítója szerepel.

A módosító fájl lyukkártyán van, rendezett és végét fájl vége jel jelzi.

Készítsd el a törzsfájl felújító algoritmusát.

3. Készíts algoritmust, amely alkalmas arra, hogy egy szekvenciális fájlban levő munkaügyi adatokból előállítsa a kifizethető bérek listáját. A fájl rekordfelépítése a következő:

— Azonosító
— Bruttó bér
— Összes levonandó összeg (gyermektartás, törlesztés stb.)
— Összes hozzáadandó összeg (családi pótlék stb.)

A nyugdíjjárulék mértékét a következő táblázat tartalmazza:

...2 100	3%
2 101...2 600	4%
2 601...3 300	5%
2 301...4 300	6%

4 301... 5 300	7%
5 301... 6 300	8%
6 301... 7 300	9%
7 301... 8 300	11%
8 301...10 300	12%
10 301...12 300	13%
12 301...14 300	14%
14 301-től	15%

Az algoritmus a következők szerint működjék:

A bruttó bérhez a pillanatnyilag érvényes szabályozóknak megfelelően hozzáad 310 forintot. Az így kapott értékből kiszámítja a nyugdíjjárulékot és a nyugdíjjárulékkal csökkentett bért, ez utóbbiból még levonja a levonandó összeget és hozzáadja a növelő tételt. Sornyomtatóra írja az azonosítót, a bruttó bért, a nyugdíjjárulékot, a levonások és növelések (előjeles) egyenlegét és a nettó bért.

Az elmondottakat minden rekordra el kell végezni. Legvégül írassuk ki azt is, hogy összesen hány forint kerül kifizetésre (mennyi a nettó bérek összege).

4. Egy nagyvállalatnak két gyára van. Mindkét gyárból egy-egy mágnesszalagon elküldték a dolgozóikról a következő rekordfelépítésű fájlt a központba:

— Dolgozó azonosítója
— Neve
— Beosztása
— Fizetése
— Munkábaállásának kezdete
— A gyárba való belépés dátuma

Mindkét fájl rendezett és fájl vége jellel zárul. Ugyanaz az azonosító mindkét fájlban nem fordulhat elő.

Készítsünk algoritmust, amely a két input fájlból előállítja a teljes vállalati nyilvántartást. Az output fájl is legyen azonosítóra rendezett.

5. Egy gyárnak két telephelye van, mindkettőn folytatnak termelést. Vannak termékek, amelyeket csak az egyik telephelyen gyártanak, vannak, amelyeket mindkettőn. Mindkét telepen a leltár után előállították az ott gyártott termékek fájlját a következő rekordfelépítéssel:

— Azonosító
— Megnevezés
— Raktáron levő mennyiség a dec. 31. leltár szerint
— Egységár

Készítsünk algoritmust, amely előállítja a gyár összevont nyilvántartását. Az input fájlok rendezettek, fájl vége jellel zárulnak. Az output fájl is ilyen legyen. Ha előfordul olyan termék, amelyet mindkét telephelyen gyártanak, a két raktári mennyiséget össze kell adni. Ilyen esetekben biztosítva van, hogy a mindkét telephelyen gyártott termék ára azonos.

6. Egy szövetkezet termékeiről a következő rekordfelépítésű fájl áll rendelkezésre:

— Azonosító
— Termelt elsőosztályú mennyiség
— Termelt másodosztályú mennyiség
— Termelt harmadosztályú mennyiség
— Termelt selejtmennyiség
— Egységár elsőosztályú áron

A másodosztályú terméket 20%-kal, a harmadosztályút 25%-kal olcsóbban lehet csak értékesíteni.

Készítsünk algoritmust, amely egy output fájlba kiírja a termékenkénti várható árbevétel adatait a következő rekordképpel:

— Azonosító
— Várható árbevétel az elsőosztályú termék értékesítéséből
— Várható árbevétel a másodosztályú termék értékesítéséből
— Várható árbevétel a harmadosztályú termék értékesítéséből

Sornyomtatóra írja ki a veszteség értékét, mégpedig külön az első/másodosztályból adódó összes veszteséget, külön az első/harmadosztályból adódó összes veszteséget, külön a selejttermelés miatt adódó összes veszteséget (ez utóbbit oly módon számítva, hogy a selejt mennyiségét az elsőosztályú árral szorozzuk).

IX. FEJEZET

Operációs rendszer

A számítógépekről és azok programozásáról szerzett ismereteink mennyisége már indokolja, hogy az ismétlés és a rendszerezés igényével feltegyük a kérdést: Mit tudunk a számítógépekről?

Ismerjük a hardver felépítését: megismertük a központi tár, a központi egység funkcióit, a különböző perifériák és háttértárak működésének fontosabb elemeit.

Megismerkedtünk az algoritmusok készítésének lényegesebb mozzanataival, és az adatstruktúrák közül a fontosabbakkal. Tudjuk, hogyan lehet programot tervezni, sőt egy egyszerűbb feladat programját el is tudjuk készíteni. Ismereteket szereztünk a programozási nyelvekről, tudjuk, hogy a magas szintű programozási nyelveknek milyen előnyei vannak.

Megismertük a számítógépes szoftver erőforrások közül a fordítóprogramok és a szerkesztők feladatait, és működésük főbb jellemzőit. Találkoztunk néhány gyakori fájlszerkezettel és a szekvenciális fájllal kapcsolatos feldolgozási példákkal.

Mindez — úgy gondolhatjuk — elegendő, hogy befejezzük a számítástechnikával kapcsolatos ismeretgyűjtésünket. Pedig fontos dolgokról még nem volt szó!

Az a helyzet ui., hogy a korszerű számítógépek hardverjének bonyolultsági foka túllépte azt a szintet, amely ezeknek az erőforrásoknak a felhasználói programból való (gazdaságos) igénybevételét biztosítaná. Ugyanakkor a felhasználók köre olyan mértékben megnőtt és kiszélesedett, hogy ilyen jellegű feladatok megoldását már nem is szabad megkívánni a felhasználótól.

E gondolatmenet alapján rögtön felmerül néhány probléma. Eddigi ismereteink alapján nem tudjuk a választ pl. arra, hogy

— honnan tájékozódik a számítógép arról, hogy egy program milyen erőforrásigénnyel lép fel;

— hogyan kapja meg egy program a végrehajtásához szükséges erőforrásokat;

— mi történik egy programmal, amikor bekerül a számítógépbe;

— hogyan automatizálódik a számítógép működése, miközben több felhasználó sokféle munkáját dolgozza fel.

Ahhoz, hogy ezekre a kérdésekre válaszolhassunk, meg kell ismerkednünk egy új fogalommal. Ez az operációs rendszer fogalma.

Az *operációs rendszert* a számítógép szoftverjének vezérlőprogramjai alkotják. E programoknak a következő főbb feladatokat kell megvalósítaniok:

— a hardver erőforrásoknak (így mindenekelőtt a tárnak és a perifériáknak) a szétosztását a felhasználói programok között;

— a felhasználói programok futásának figyelemmel kísérését (felügyeletét);

— a feldolgozóprogramok működtetését;

— az adatkezelések leegyszerűsítését;

— ellenőrzési funkciók ellátását;

— a felhasználóval és a gépet kezelő operátorral való kapcsolattartás leegyszerűsítését;

— a fellépő hibák kezelésének megkönnyítését. [● | 301 | 1]

Az operációs rendszer főbb feladatait látva megfogalmazhatjuk azokat a kérdéseinket, amelyek megválaszolása nélkül ismereteink hiányosak maradnak. Miután pedig a felhasználók is és az operátor is csak az operációs rendszeren keresztül tud a számítógéppel kapcsolatot teremteni, ill. tartani, ismereteinket ebben a témakörben okvetlenül bővítenünk kell. Alapvető problémánk tehát a következők megválaszolása:

Melyek az operációs rendszerek, hogyan csoportosíthatók, milyen funkciókat látnak el? Hogyan tud a felhasználó az operációs rendszerrel — és ezen keresztül a számítógéppel — kapcsolatot teremteni? Hogyan valósulnak meg az operációs rendszer funkciói?

Ebben a fejezetben ezekre a kérdésekre keressük a választ. A kérdések megválaszolása során újabb fogalmakkal is találkozunk.

Az operációs rendszerek kialakítása, hasonlóan a programozási nyelvek kialakulásához, hosszabb folyamat során ment végbe.

Az első generációs számítógépek nem rendelkeztek semminemű vezérlési funkciót ellátó programmal. A későbbiek során először a különböző nyelvi fordítók jelentek meg, [▲ | 299 | 2] és a gép működésének szinte minden mozzanata az operátorra hárult. [● | 302 | 3]

A számítógépnek az ilyen kézi kiszolgálása hallatlan mértékben lelassította a berendezés működési sebességét és átbocsátóképességét. [▲ | 299 | 4] A számítógép ideje legnagyobb részében az operátorra várt. Már a második generációs gépeknél megjelentek azok a programok, amelyek e tevékenységek valamilyen módon való automatizálását szolgálták. Ezek az ún. monitorprogramok tekinthetők a mai operációs rendszerek „őseinek", bár csak bizonyos funkciókat valósítottak meg. [● | 302 | 5]

A mai értelemben operációs rendszernek nevezhető egységes programegyüttes, amely a számítógép optimális működtetésével kapcsolatos valamennyi feladatot ellátja, a harmadik generációs gépekkel jelent meg. Egy számítógép működtetésének azonban többféle optimuma is lehetséges, ezért maguk az operációs rendszerek is többfélék lehetnek. Az operációs rendszerek csoportosításának alapvető szempontja éppen az, hogy a rendszerrel *milyen célkitűzés optimalizálását* kívánjuk megvalósítani.

Az *egyik ilyen cél* az lehet, hogy a számítógép egy adott időszakban (pl. egy óra alatt) *minél több hasznos munkát* végezzen (minél több programot dolgozzon fel).

Egy *másik cél* lehet, hogy a felhasználó a feladat megoldóprogramját *minél rövidebb idő alatt* tudja hibamentesen elkészíteni és a számítógépen feldolgozni.

Egy *harmadik fajta* célkitűzés az lehet, hogy valamilyen körülhatárolt témakörben a *felhasználó azonnal* — tehát gyakorlatilag várakozás nélkül — választ tudjon kapni a kérdéseire.

Azok az operációs rendszerek, amelyek nem különleges igények kielégítésére készült speciális rendszerek, az előbbi célok valamelyikének érdekében működnek. Ennek megfelelően a következő hármas csoportosítást tehetjük:

— kötegelt feldolgozást biztosító rendszerek,
— többfelhasználós időosztásos rendszerek,
— valós idejű rendszerek.

Kötegelt feldolgozásról akkor beszélünk, amikor a különböző felhasználók munkáit összegyűjtik és „kötegekben" (több munkát együtt) olvastatják be a számítógépbe. A számítógép ezeket a programokat egymás után (vagy időben átlapolva) dolgozza fel. A programozónak ebben az esetben nincs semmi kapcsolata a géppel. Hibajavítás úgy lehetséges, hogy a futási eredményeket visszakapva elvégzi a szükséges javításokat, majd ismét leadja a programot. A programozási munka és a program kipróbálása tehát időben egymástól elkülönülő tevékenység.

A **többfelhasználós időosztásos rendszer** alkalmas arra, hogy több terminált szolgáljon ki. Ilyen esetben a felhasználók a terminálóknál ülve a programkészítés és kipróbálás (futás) során állandó kapcsolatban lehetnek a számítógéppel, annak minden erőforrását igénybe tudják venni. Az egyes terminálóknál ülő felhasználók mindegyike úgy érzi, hogy a számítógép csak vele foglalkozik. A terminálok akár közvetlenül, akár telefonvonalakon keresztül kapcsolódhatnak a számítógéphez.

A **valós idejű rendszerek** valamilyen előre meghatározott feladathoz igazodnak. A felhasználók (általában szintén terminálon keresztül) e feladathoz kapcsolódóan vehetik igénybe a számítógépet. Ilyen feladat lehet pl. valamilyen folyamatnak a számítógépes irányítása (űrhajó vezérlése, zárt gyártási folyamat vezérlése) vagy többfelhasználós formában valamilyen adatbázisban való tájékozódás (pl. repülőgépes helyfoglalási rendszer). A felhasználó csak olyan számítógépes erőforrásokat vehet igénybe, amelyek e feladat céljait szolgálják. Léteznek olyan operációs rendszerek is, amelyek „kevert" célokat szolgálnak: pl. kötegelt feldolgozást és időosztásos terminálhasználatot egyszerre tesznek lehetővé. Ilyen esetekben azonban a „tiszta" esetekhez képest lassabb rendszerszolgáltatásokat kapunk.

A továbbiakban figyelmünket a kötegelt feldolgozást biztosító rendszerekre fordítjuk. ● 303 6

Rögtön az első kérdés, amelyet meg kell válaszolnunk, így hangzik: Hogyan képes az operációs rendszer megállapítani, hogy a számítógépnek mit kell csinálnia egy programmal, amelyet pl. kártyaolvasóról olvasott be vagy terminálról küldte egy felhasználó.

Nyilvánvaló, hogy ehhez a döntéshez *információkra* van szükség. Azokat az adatokat, amelyek a szükséges információkat hordozzák, a felhasználónak kell az operá-

ciós rendszer számára biztosítani. Ezek az adatok az elvégzendő munka erőforrásigényét írják le. Ily módon a számítógép számára a következőket kell átadni, ha a program feldolgozását kívánjuk:

— **vezérlő utasításokat,** amelyek a program hardver és szoftver erőforrásigényét közlik a rendszerrel;

— a **programot,** amely a megoldó algoritmus és a hozzá tartozó adatstruktúrák leírását tartalmazza;

— azokat az **adatokat,** amelyekkel a program műveleteket végez.

A JOB (MUNKA)

Ez a háromféle információ megfelelően rögzített és összeállított állapotban a számítógép számára *egy munkát, idegen szóval: jobot* (dzsob) *határoz meg.*

A munka egy vagy több, általában logikailag összetartozó programot magában foglaló, a számítógép számára összeállított, végrehajtásra előkészített műveletsor. Legtöbb esetben kisebb egységekből (job lépésekből) épül fel. A job lépés általában egy önállóan végrehajtható programot jelent a jobon belül.

A job a felhasználó által, a számítógép szempontjából kívülről meghatározott munka, amelyet az operációs rendszer *feladatokká* bont le.

A TASK (FELADAT)

A task (feladat) futás céljából a központi tárba betöltött program és a hozzá tartozó adatok. A számítógép ezeket a feladatokat dolgozza fel. Általában egy job lépés alkot egy feladatot.

Egy munkát alkot pl. a következő műveletsor:

— egy FORTRAN nyelvű program lefordítása;

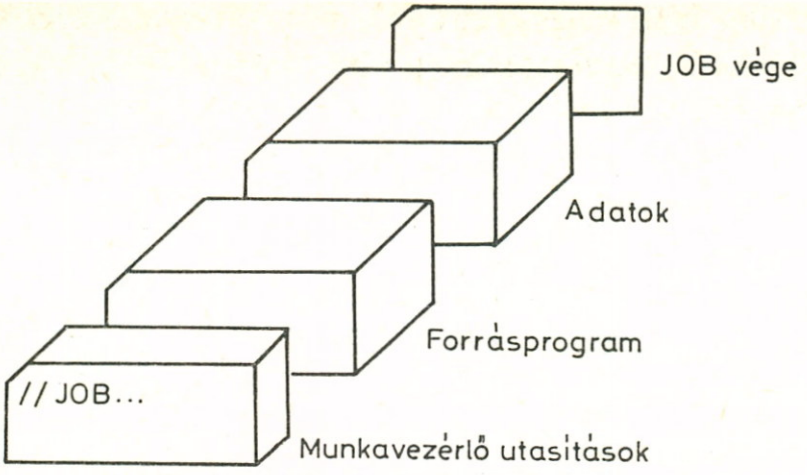
— a tárgynyelvi program szerkesztése és betöltése a tárba;

— a program futtatása.

Ez a munka három job lépésből áll. Az első lépésben a fordítóprogram működik, a másodikban a kapcsolatszerkesztő, a harmadik lépésben a felhasználói program.

A 85. ábra egy job általános szerkezetét mutatja, minden információt lyukkártyán szemléltetve. Természetesen jobot terminálról is lehet összeállítani és indítani.

Ha egy olyan programot akarunk futtatni, amely korábban már szerkesztve volt és könyvtárban található, s ez a program egy olyan fájlt dolgoz fel, amely pl. mágneslemezen vagy mágnesszalagon van, akkor — feltéve, hogy a futtatás kötegelt módon történik — a kártyaolvasóba csak a munkavezérlő utasítások kerülnek. Ezek feladata, hogy pontosan leírják a futtatandó program tárolási helyét és a feldolgozandó fájl azonosításához szükséges információkat. ▲ 299 7



85. ábra. Munka-struktúra lyukkártyán szemléltetve

MUNKAVEZÉRLÉS

A munkavezérlés funkcióit az ún. **munkavezérlő nyelvvel** lehet megvalósítani. A munkavezérlő nyelv segítségével lehetővé válik, hogy az egyes munkák erőforrásigényét az operációs rendszer elemezze, és külső beavatkozás nélkül valamilyen elv szerint kielégítse. Ezzel lehetővé teszi a munka elvégzését: a programok feldolgozását.

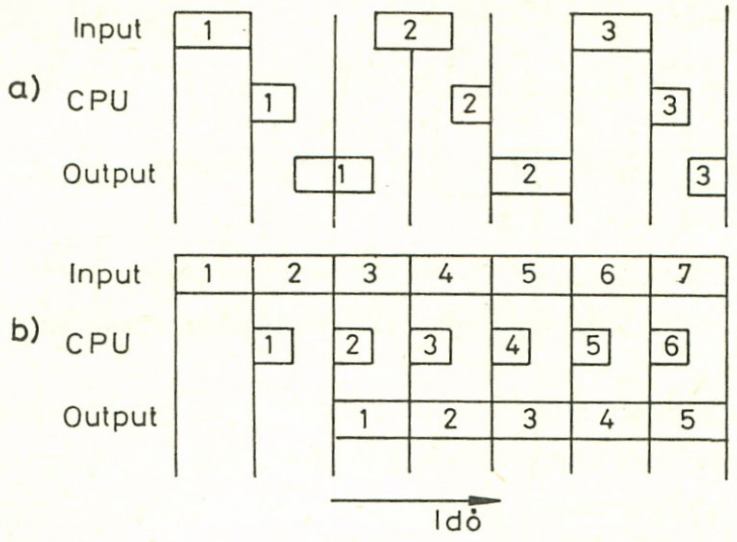
Kötegelt feldolgozás esetén így kiküszöbölhető a rendszer működési sebességét fékező állandó operátori beavatkozás. A kötegelt feldolgozás lényege ui. az, hogy a munkák egymás után érkeznek, és azokat a számítógép úgy dolgozza fel, hogy minél nagyobb átbocsátóképessége legyen a rendszernek. Kisebb számítógépek esetén [•] [303] [8] a feldolgozás sorosan történik, vagyis egy munkát akkor kezd el végrehajtani, ha az előzővel végzett. Nagy számítógépek esetén azonban ez a megoldás nem célszerű: egyetlen program képtelen kihasználni az összes erőforrást. Ezeknek a gépeknek az operációs rendszere biztosítani tudja, hogy a számítógép az egyes munkákat nem egymás után, hanem párhuzamosan, *egymás mellett* dolgozza fel. Hogyan lehetséges ez?

Ennek megértéséhez ismételjünk át néhány dolgot, amit a számítógép működéséről korábban megismertünk.

A csatornák megjelenése után többé már nem a központi egység feladata az adatátvitel felügyelete és bonyolítása. Számításokat a központi egység azonban csak azokkal az adatokkal tud elvégezni, amelyek már a központi tárban vannak.

Mivel azonban a számítógépnek a központi egység a leggyorsabb része, gyakran előfordult, hogy várakoznia kellett a számításokhoz szükséges adatok megérkezéséig, az adatátvitel lebonyolításáig. Ennek az lett a következménye, hogy egy feladat — bár számítási időigénye csupán néhány századmásodperc volt — több percig is lefoglalta a gépet, és lefoglalta az erőforrásokat. A perifériák a csatornákon keresztül állandóan szolgáltatták az adatokat (rekordokat), de a központi egység mindig gyorsabban dolgozta fel ezeket, mint amíg a következő input rekord megérkezett. A központi egység

várakozási idejének alakulása megfigyelhető a 86. ábrán, amelyen az időszakok jelölése nem méretarányos. A központi egység által valóságosan elhasznált idő az ábrázoltnál jóval kevesebb.



86. ábra. A központi egység időbeli terhelése

A MULTIPROGRAMOZÁS

A központi egység ilyen „rossz" kihasználásának felszámolása, vagyis az állandó várakozások megszüntetése csak úgy lehetséges, ha a számítógép egy időszakban nem egy, hanem több programot kezel. Ezzel megszületett a *multiprogramozás* gondolata. A multiprogramozás a programok időben egymásba ágyazott végrehajtása. Multiprogramozáskor a számítógépben tehát egyidejűleg több végrehajtásra kész állapotban levő program található, s a számítógép ezen programok közül egy meghatározott módszer szerint válogat. Ha nincs multiprogramozás, az egyetlen végrehajtás alatt levő program birtokolja a teljes számítógépet, annak összes erőforrásával együtt, amelyeket nem képes hatékonyan lekötni. Multiprogramozás esetén **több programnak osztozkodnia kell** az erőforrásokon. Vannak erőforrások, amelyek megoszthatók a programok között. Ilyenek pl. a központi tár, a mágneslemezek. Más erőforrásokat (pl. egy mágnesszalagot) egy program vehet igénybe. Magát a központi egységet pedig csak időben felváltva használhatja több program.

Mi a feltétele annak, hogy egy programot végrehajthatónak tekintsünk?

Egy program végrehajtásának szükséges, de nem elégséges feltétele, hogy az igényelt erőforrások rendelkezésre álljanak. Az operációs rendszer egy különleges programjának feladata, hogy a kielégített erőforrásigénnyel fellépő program számára a futást lehetővé tegye. Mivel egy adott időpillanatban egynél több ilyen is létezhet, valamilyen módszer szerint kell ezek közül választani.

A multiprogramozás megvalósítása több megoldandó problémát vetett fel. Ezek közül a legfontosabb kettőt mutatjuk be.

Az egyik a **központi tár kezelése,** a másik a **programok „váltása".**

A tár kezelése azért fontos, mert a központi tárnak mint a számítógép egyik legfontosabb részének a „közös" használatát szabályozza. A programok váltása esetén pedig arról van szó, hogy hogyan tudnak a programok megosztozni a számítógép másik fontos részén: a központi egységen.

A központi tár kezelése

Multiprogramozás esetén a központi tár több program számára való biztosítása többféle módon lehetséges. Adott számítógép esetén a rendelkezésre álló hardver struktúra és operációs rendszer dönti el, hogy melyik használható.

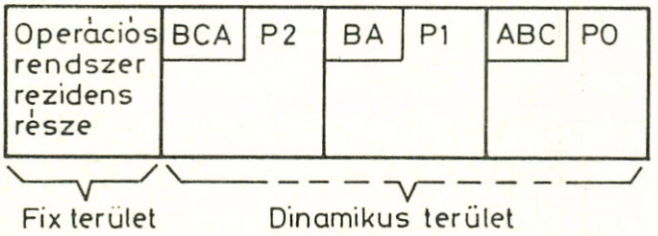
A legfontosabb lehetőségek a következők:

— partíciókra osztott tár,

— laptechnikán alapuló tár.

A továbbiakban a **partíciós tárfelosztásnak** azt a változatát mutatjuk be, amelyben előre meghatározott számú feladat között osztja fel a tárat az operációs rendszer.

A központi tár két alapvető részre tagozódik. Az egyik az ún. fix terület, a másik a dinamikus terület. A felosztás elvét a 87. ábra szemlélteti.



87. ábra. A tár szerkezete OS/MFT esetén

A fix területen helyezkedik el az operációs rendszernek az a része, amelynek állandóan a tárban kell lennie. Ezt nevezzük a rendszer *rezidens* részének. A dinamikus terület a logikailag önálló részekre, partíciókra van felosztva. A partíciófelosztást az operátor a mindenkori igénynek megfelelően változtatni tudja. ⊙ 309 9

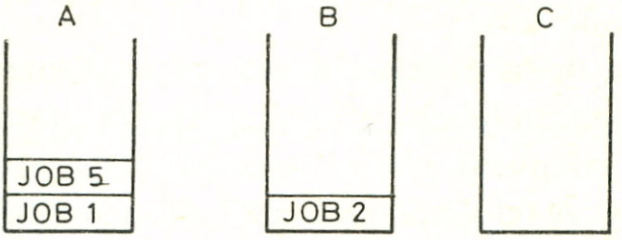
Az egyes partíciók egy-egy *sorszámmal* rendelkeznek, amelyek egymáshoz viszonyított elsőbbségi sorrendet jelentenek: a P0 partíció előbbsége nagyobb (nagyobb prioritású), mint P1 partícióé stb. Az egyes partíciókhoz ugyanakkor *job osztályok* is tartoznak. A munkákat — job-okat — szintén osztályba kell sorolni. Ha ezt a programozó nem teszi meg, a rendszer automatikusan hozzárendel a munkához egy job osztályt. A job osztályok A-tól O-ig az abc egy betűjét kaphatják. Egy-egy partícióba csak olyan munka kerülhet be, amelynek osztálya megegyezik a partícióhoz tartozó legfeljebb három osztály valamelyikével.

A „legjobb" osztályok nyilvánvalóan azok, amelyek a jobban privilegizált partícióhoz vannak hozzárendelve. Általában ezt a hozzárendelést úgy végzik, hogy az abc-ben előrébb álló betű előbbre való job osztályt jelentsen, mint a hátrább álló betű által jelzett.

A 87. ábrán látható felosztás szerint pl. P0 partícióban A, B, C osztályú; P1 partícióban B, C, A osztályú jobok kerülhetnek be. A partíción belüli megadási sorrend egyben elsőbbségi sorrendet is jelent.

Honnan kerülnek be a jobok egy adott partícióba?

Minden job osztályhoz tartozik a rendszerben egy-egy *várakozó sor*. Ezek a várakozó sorok mágneslemezen kerülnek kialakításra. A rendszerbe érkező munkák mindegyikéről egy *munkavezérlő táblázat* készül. Ebben a táblázatban rögzítésre kerülnek a munka feldolgozásához szükséges információk. A munkavezérlő táblázat (angol nyelvű rövidítése = JCT) a job osztálynak megfelelő várakozó sorban kerül tárolásra. A várakozó sorokban az érkezés sorrendjében történik a várakozás. Ez alól csak azok a munkák kivételek, amelyek külön prioritással rendelkeznek. A 88. ábra a korábban bemutatott tárfelosztásnak megfelelő várakozó sorokat szemléltet.



Ha a várakozó sorokban ezek a JOB-ok vannak, a tárba kerülés az alábbi lesz:

JOB1 - P0 partícióba
JOB2 - P1 partícióba
JOB5 - P2 partícióba

88. ábra. Várakozó sorok

Ha egy partíció üres, mód van arra, hogy abban egy munka ütemezésre kerüljön Az ütemezést végző program (ez a program is része az operációs rendszernek) először a partícióhoz tartozó első job osztály várakozó sorában keres munkát. Ha ez a várakozó sor üres, a sorrendben másodikban keres, és így tovább. ● | 304 | 10

A multiprogramozás a partíciók között zajlik le, az operációs rendszer mindig azt a partíciót aktivizálja, amelyik a legkisebb sorszámú (legnagyobb prioritású), és benne kielégített erőforrásigényű feladat található. A fix feladatokra való tárfelosztásnak több hátrányos jellemzője is van, így elsősorban az, hogy a mindenkori partícióméretek nem tudnak automatikusan igazodni az éppen sorra kerülő munkák tárigényéhez. Ugyanakkor adott határok között gazdaságos alkalmazást képes biztosítani.

Fejlettebb operációs rendszerek az említett merevséget is fel tudják oldani. ⊙ | 309 | 11

A programok váltása

A multiprogramozás során a task-ok „versenyeznek" a számítógép erőforrásaiért, köztük a másik fontos erőforrásért, a központi egység birtoklásáért. Ebből a verseny-

ből mindig a legmagasabb prioritású, futásra kész állapotban levő task kerül ki győztesen.

Futásra kész állapotban levőnek tekintünk egy task-ot, ha a pillanatnyi erőforrásigénye ki van elégítve, kivéve a központi egységet.

Miután a **task-ok erőforrásigénye időről-időre változik,** mindig más-más task lesz az, amelynek a „versenyből" győztesen kell kikerülnie. Az operációs rendszer legfontosabb programjának, a **felügyelő programnak** a feladata, hogy a megfelelő task aktivizálásáról gondoskodjék. A felügyelő programot szokás *supervisornak* (szupervájzor) is nevezni az angol elnevezés átvételével.

A felügyelőprogram a partíciókra osztott tár fix területén helyezkedik el.

Miért mondjuk azt, hogy a felügyelőprogram az operációs rendszer legfontosabb része? Azért, mert a felügyelőprogramnak *kitüntetett szerepe* van minden más programmal szemben. Ez a kitüntetett jelleg legszemléletesebben abból látható, hogy a számítógép működésének két állapota van:

— a *supervisor állapot*, amikor a felügyelőprogram működik;

— a *program állapot*, amikor bármilyen más program működhet.

Vannak utasítások, amelyek csak supervisor állapotban adhatók ki, vagyis csak a felügyelőprogram adhatja ki ezeket az utasításokat. Ilyen utasítás pl. a tényleges adatátvitelt lebonyolító csatornautasítás kiadása. A felhasználói programnak tehát abban az esetben, ha adatátvitelt kér, először a supervisorhoz kell fordulnia, mert önmaga nem adhat ki csatornautasítást. Ilyenkor jön létre a programmegszakítás. A megszakítás teszi lehetővé, hogy amikor egy éppen futó task (az ún. aktív task) új erőforrás-igénnyel lép fel, annak kielégítéséig másik task aktivizálódhassék.

Már a központi egység tárgyalásakor volt szó arról, hogy a modern számítógépek működése során a programok különböző okokból megszakadhatnak. Ugyanitt megismertük e problémakör legfontosabb hardver vonatkozásait.

Megszakítást a *megszakítási okok* valamelyike válthat ki. A legfontosabb megszakítási okok:

— az aktív task-nak fizikai adatátvitelre van szüksége, s ezért a supervisorhoz fordul (SCV = supervisor call megszakítás);

— egy várakozó task részére a korábban kért adatátvitel befejeződött, ezért a task vissza kívánja szerezni a futás jogát (input-output megszakítás);

— az operátor a konzolról „kilövi" a futó task-ot (külső megszakítás);

— programhiba vagy géphiba (hardver hiba) miatt megszakad a futó task.

A megszakítás felléptekor a következő teendők adódnak:

— a megszakított task pillanatnyi állapotát meg kell őrizni;

— ki kell értékelni a megszakítási okot;

— gondoskodni kell a megszakítási ok kiküszöböléséről;

— egy másik task-ot aktivizálni kell.

A megszakításkor fennálló *állapot megőrzése* hardver úton, a programállapot szó (PSW) segítségével történik. Minden megszakítási okhoz külön PSW tartozik, a supervisor ide helyezi el azokat az információkat (pl. az utasításszámláló regiszter tar-

talmát), amelyek segítségével a megszakított folyamat ott lesz folytatható, ahol abbamaradt.

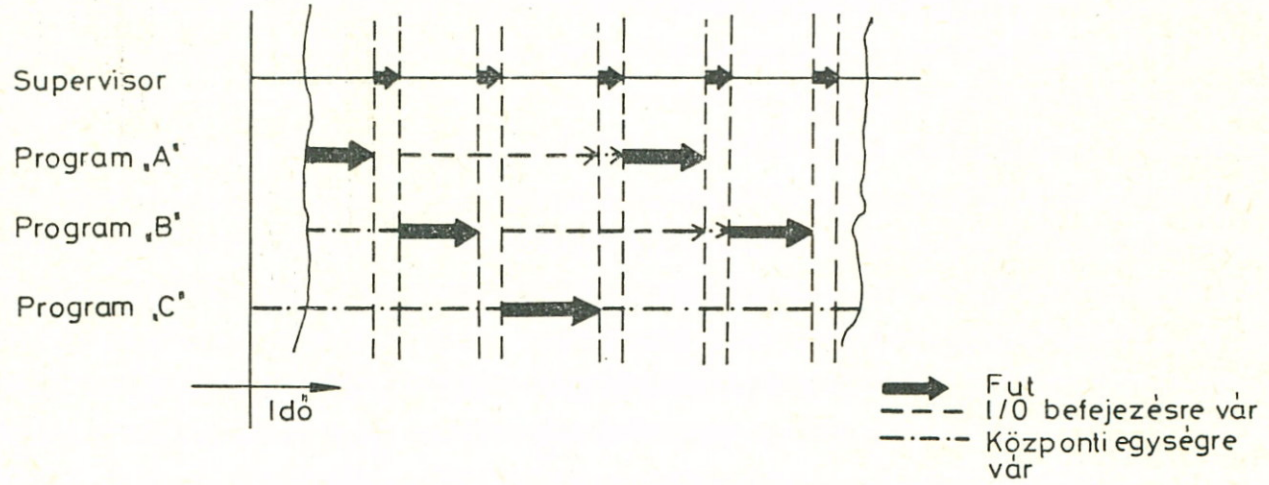
A *megszakítási okok értékelését* a megszakításkezelő rutinok végzik. Minden megszakítási okhoz egy (vagy több) megszakításkezelő program tartozik. Ezek a programok gondoskodnak a megszakítási ok kiküszöbölésének kezdeményezéséről is. (Elképzelhető persze az is, hogy a megszakítási ok kiküszöbölhetetlen állapotot teremt. Az operátori kilövéssel szemben pl. a rendszer már nem tud mit tenni. A megszakításkor fennállt állapot legfontosabb paramétereit kiírja, és egy, a megszakítási okra utaló hibakóddal befejezi a teljes job feldolgozását. Egy SVC megszakításkor viszont a kezelőprogramok aktivizálják a kért csatornát a szükséges adatátvitel elindítására.)

Az elmondott folyamat eredményeképpen a központi egység terhelése a 89. ábrán bemutatott szerkezetű lesz. Az ábrán azt is láthatjuk, hogy a programokat magukba foglaló task-ok állapotai a következők lehetnek:

— aktív (éppen fut),

— központi egységre vár (minden rendelkezésre áll a futáshoz, de egy nála nagyobb prioritású task az aktív);

— input-outputra vár (egy nála alacsonyabb prioritású task aktív, de neki adatátvitelre kell várakoznia).



89. ábra. A központi egység terhelése multiprogramozás esetén

AZ OPERÁCIÓS RENDSZER FOGALMA

Az eddig megismertek alapján megfogalmazhatjuk, hogy mit is nevezünk operációs rendszernek.

Az operációs rendszer a rendszerprogramok azon összessége, amelynek működése a számítógép erőforrásainak adott cél szerinti optimális kihasználását, a programozói munka megkönnyítését és az operátori munka egyszerűsítését eredményezi. A számítógépben folyó valamennyi folyamat vezérlését és ellenőrzését az operációs rendszer látja el.

Hogyan tud a felhasználó az operációs rendszerrel *kapcsolatot teremteni?*

Amint azt a munka (job) meghatározásakor láthattuk, a felhasználó az operációs rendszer számára vezérlőutasításokban tudja előírni programjának erőforrásigényét. Ezek a vezérlőutasítások alkotják a *munkavezérlő nyelvet*, amelyet szokás az elnevezés angol változatának rövidítéséből JCL-nek is nevezni.

A MUNKAVEZÉRLŐ NYELV

A munkavezérlő utasítások az operációs rendszer számára rendkívül fontosak. Egy jobon belül az operációs rendszer számára csak munkavezérlő és nem munkavezérlő utasítások léteznek. Ezért a munkavezérlő nyelvi utasítások olyan jelzéssel kezdődnek, amely azonnal jelzi az operációs rendszer számára, hogy az érkező rekordban vezérlő információk vannak. Esetünkben ez a felismerő jel a rekord első két bájtjában (a kártya első két oszlopán) levő // kombináció. (A / — ferde vonal vagy per jel a EBCDIC kódrendszer egyik speciális karaktere.)

A munkavezérlő nyelv operációs rendszerenként eltérő logikájú és felépítésű, vannak azonban olyan funkciók, amelyek minden operációs rendszerhez tartozó munkavezérlésnél megtalálhatók.

Melyek a munkavezérlő nyelvek *legfontosabb funkciói?*

Három olyan funkció van, amely fontosságban kiemelkedik. Ezek:

— az elvégzendő munka azonosítása,

— a végrehajtandó program azonosítása,

— a feldolgozás során használni kívánt fájlok azonosítása.

A következőkben bemutatjuk, hogy ez a három funkció hogyan valósul meg az *OS operációs rendszer* munkavezérlő nyelvében.

Az *elvégzendő munka azonosítását a* **JOB** utasítás valósítja meg.

Az azonosítás a következőket jelenti: A JOB utasítás megjelenése azt jelzi, hogy egy új job kezdődik. A JOB utasításban levő azonosító (legfeljebb nyolc karakteres név) teszi lehetővé, hogy minden, adott nevű munkával kapcsolatos információ, esemény hozzárendelhető legyen a megfelelő munkához. A JOB utasításban levő elszámolási információ teszi lehetővé, hogy a job által elhasznált erőforrások ellenértékét el lehessen számolni. ▲ 300 13

A JOB utasításban azonosítható a programozó személye is.

Példa: //TPELDA JOB 223344, 'MACI LACI', CLASS = A, PRTY = 9

Magyarázat:

TPELDA Jobnév, legalább 1 és legfeljebb 8 karakter lehet. Csak alfanumerikus jeleket tartalmazhat. A 3. pozición kell kezdődnie (közvetlenül a // után), és az első karaktere nem lehet számjegy. Az egyes számítóközpontokban a job nevével kapcsolatban különböző megszorításokat szoktak alkalmazni.

223344, elszámolási információ, a „job" utasítás első pozicionális paramétere. A job által elhasznált erőforrások automatikus számlázását teszi lehetővé. Használata

talmát), amelyek segítségével a megszakított folyamat ott lesz folytatható, ahol abbamaradt.

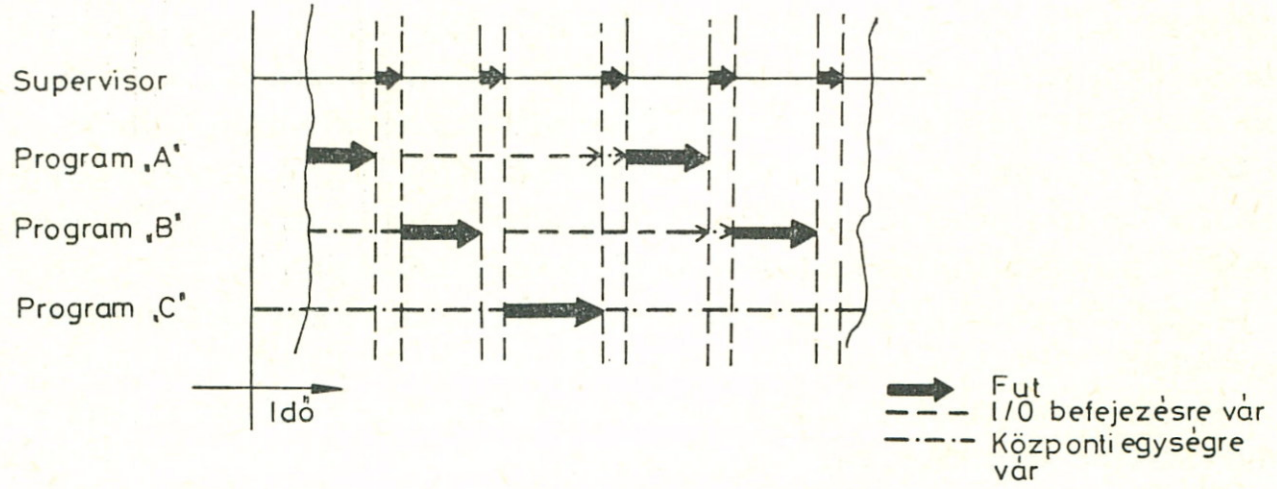
A *megszakítási okok értékelését* a megszakításkezelő rutinok végzik. Minden megszakítási okhoz egy (vagy több) megszakításkezelő program tartozik. Ezek a programok gondoskodnak a megszakítási ok kiküszöbölésének kezdeményezéséről is. (Elképzelhető persze az is, hogy a megszakítási ok kiküszöbölhetetlen állapotot teremt. Az operátori kilövéssel szemben pl. a rendszer már nem tud mit tenni. A megszakításkor fennállt állapot legfontosabb paramétereit kiírja, és egy, a megszakítási okra utaló hibakóddal befejezi a teljes job feldolgozását. Egy SVC megszakításkor viszont a kezelőprogramok aktivizálják a kért csatornát a szükséges adatátvitel elindítására.)

Az elmondott folyamat eredményeképpen a központi egység terhelése a 89. ábrán bemutatott szerkezetű lesz. Az ábrán azt is láthatjuk, hogy a programokat magukba foglaló task-ok állapotai a következők lehetnek:

— aktív (éppen fut),

— központi egységre vár (minden rendelkezésre áll a futáshoz, de egy nála nagyobb prioritású task az aktív);

— input-outputra vár (egy nála alacsonyabb prioritású task aktív, de neki adatátvitelre kell várakoznia).



89. ábra. A központi egység terhelése multiprogramozás esetén

AZ OPERÁCIÓS RENDSZER FOGALMA

Az eddig megismertek alapján megfogalmazhatjuk, hogy mit is nevezünk operációs rendszernek.

Az operációs rendszer a rendszerprogramok azon összessége, amelynek működése a számítógép erőforrásainak adott cél szerinti optimális kihasználását, a programozói munka megkönnyítését és az operátori munka egyszerűsítését eredményezi. A számítógépben folyó valamennyi folyamat vezérlését és ellenőrzését az operációs rendszer látja el.

Hogyan tud a felhasználó az operációs rendszerrel *kapcsolatot teremteni?*

Amint azt a munka (job) meghatározásakor láthattuk, a felhasználó az operációs rendszer számára vezérlőutasításokban tudja előírni programjának erőforrásigényét. Ezek a vezérlőutasítások alkotják a *munkavezérlő nyelvet*, amelyet szokás az elnevezés angol változatának rövidítéséből JCL-nek is nevezni.

A MUNKAVEZÉRLŐ NYELV

A munkavezérlő utasítások az operációs rendszer számára rendkívül fontosak. Egy jobon belül az operációs rendszer számára csak munkavezérlő és nem munkavezérlő utasítások léteznek. Ezért a munkavezérlő nyelvi utasítások olyan jelzéssel kezdődnek, amely azonnal jelzi az operációs rendszer számára, hogy az érkező rekordban vezérlő információk vannak. Esetünkben ez a felismerő jel a rekord első két bájtjában (a kártya első két oszlopán) levő // kombináció. (A / — ferde vonal vagy per jel a EBCDIC kódrendszer egyik speciális karaktere.)

A munkavezérlő nyelv operációs rendszerenként eltérő logikájú és felépítésű, vannak azonban olyan funkciók, amelyek minden operációs rendszerhez tartozó munkavezérlésnél megtalálhatók.

Melyek a munkavezérlő nyelvek *legfontosabb funkciói?*

Három olyan funkció van, amely fontosságban kiemelkedik. Ezek:

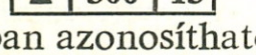
— az elvégzendő munka azonosítása,

— a végrehajtandó program azonosítása,

— a feldolgozás során használni kívánt fájlok azonosítása.

A következőkben bemutatjuk, hogy ez a három funkció hogyan valósul meg az *OS operációs rendszer* munkavezérlő nyelvében.

Az *elvégzendő munka azonosítását a* **JOB** utasítás valósítja meg.

Az azonosítás a következőket jelenti: A JOB utasítás megjelenése azt jelzi, hogy egy új job kezdődik. A JOB utasításban levő azonosító (legfeljebb nyolc karakteres név) teszi lehetővé, hogy minden, adott nevű munkával kapcsolatos információ, esemény hozzárendelhető legyen a megfelelő munkához. A JOB utasításban levő elszámolási információ teszi lehetővé, hogy a job által elhasznált erőforrások ellenértékét el lehessen számolni. 

A JOB utasításban azonosítható a programozó személye is.

Példa: //TPELDA JOB 223344, 'MACI LACI', CLASS = A, PRTY = 9

Magyarázat:

TPELDA Jobnév, legalább 1 és legfeljebb 8 karakter lehet. Csak alfanumerikus jeleket tartalmazhat. A 3. pozición kell kezdődnie (közvetlenül a // után), és az első karaktere nem lehet számjegy. Az egyes számítóközpontokban a job nevével kapcsolatban különböző megszorításokat szoktak alkalmazni.

223344, elszámolási információ, a „job" utasítás első pozicionális paramétere. A job által elhasznált erőforrások automatikus számlázását teszi lehetővé. Használata

akkor kötelező, ha a gépben automatikus számlázóprogram működik. A számítóközpont adja meg az értékét.

MACI LACI, programozó neve, a „job" utasítás másik pozicionális paramétere. Legfeljebb 20 karakterből állhat, ha szóköz szerepel benne, mint példánkban is, a nevet felső vesszők közé kell tenni.

A „job" utasításnak ún. kulcsszavas paraméterei is lehetnek. Ezek közül példánkban kettő szerepel.

CLASS = A, a job osztály megadása, a multiprogramozáskor ettől függ, hogy mely partíciókba kerülhet be a job.

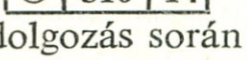
PRTY = 9, sürgősségi szám, amelynek hatására az A várakozó sorban megelőzi azokat a jobokat, amelyek PRTY paramétere kisebb, mint 9.

A *végrehajtandó program azonosítása az* adott job lépésben végrehajtandó program nevének megadását jelenti. A végrehajtandó program azonosítása az **EXEC** *utasítás segítségével* történik. (Nevét az angol execute = végrehajt szó rövidítéséből kapta.) Ebben az utasításban egy lefordított, összeszerkesztett (betölthető) program nevét kell megadni. Ennek a programnak a következő könyvtárak valamelyikében kell lennie:

— a rendszer egy állandó könyvtárban, amelyet betölthető programok számára hoztak létre. (Ilyen könyvtár lehet rendszer szintű, vagyis bárki által használható, és felhasználói szintű, vagyis egy felhasználóhoz rendelt.);

— ideiglenes könyvtárban, amelyet a kapcsolatszerkesztő futáskor hoz létre a rendszer, és a kapcsolatszerkesztő által létrehozott futtatható program ideiglenes tárolására szolgál. Ilyenkor a programot a job valamelyik soron következő job lépésében kell végrehajtatni, hiszen a könyvtár ideiglenes jellege azt jelenti, hogy a job befejezésével törlésre kerül. Pl.: // EXEC PGM=IEYFORT

Magyarázat:

PGM = ... az egyenlőségjel után megadott program kerüljön végrehajtásra (esetünkben a FORTRAN fordítóprogram). Ha a végrehajtásra szánt program nem a rendszer saját könyvtárainak valamelyikében van, hanem a felhasználó saját könyvtárában, ezt a könyvtárat egy külön utasítással meg kell adni (l.: a DD utasításnál a 3. példát.) 

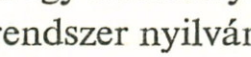
A feldolgozás során különböző állományokra van szüksége a programozónak. Milyen állományokról van szó?

Minden olyan információ (program vagy adat), amely valamilyen tárolóközegen van vagy a job feldolgozása során tárolóközegre kerül, a munkavezérlés során azonosítandó állományt jelent. Így pl. ha a forrásprogram lyukkártyán került rögzítésre, a fordítási job-lépésben a kártyaolvasón egy input állomány jelentkezik. Ha a program futása során egy mágnesszalagon levő fájl rekordjait dolgozza fel, és a feldolgozás eredményeit sornyomtatón jeleníti meg, a mágnesszalagon levő fájl a végrehajtás input állományaként, a sornyomtatót pedig output állományként meg kell adni.

A job feldolgozás során előforduló *állományok azonosítását a* **DD** *utasítás végzi.* (Nevét az angol Data Definition = adatmeghatározás rövidítéséből kapta.)

Önálló DD utasítással kell azonosítani minden job lépésben:

— az adott job lépés inputját,
— a job lépés outputját,
— a job lépés során esetleg előforduló munkaállományokat.

Hogyan történik egy állomány azonosítása? Mindenekelőtt meg kell adni az állomány nevét. Mivel azonban egy állománynak általában más neve van a programban, és más neve az operációs rendszer nyilvántartásában,  mindkét nevet meg kell adni, sőt e két nevet egymáshoz kell rendelni. Ezenkívül — ha nem a sornyomtatóról, ill. a kártyaolvasóról van szó — meg kell mondani, hogy az állomány milyen adathordozón (mágnesszalag, mágneslemez) található és mi a neve (azonosítója) annak a szalag- vagy lemezkötegnek, amelyen megtalálható. A gyakran használt állományok esetében az állomány helyére vonatkozó információk beírhatók egy speciális fájlba, a *rendszerkatalógusba.* Ilyenkor elegendő az állomány nevét megadni. Egy job lépés output állománya esetében, tehát amikor egy új állomány kerül létrehozásra, az állomány további jellemzőit is elő kell írni. Így pl. meg kell adni a rekordméretet, a blokkméretet, a szervezettséget (szekvenciális, index-szekvenciális, direkt stb.). Ilyenkor kell megadni mágnesszalagos állományok esetében az írási sűrűséget stb.

Példa 1: //FORT.SYSIN DD *

Magyarázat:

FORT.SYSIN — a FORT nevű job lépésben szereplő DD utasítás neve, a SYSIN a rendszer inputot azonosítja.

* — Azt jelenti, hogy a DD-ben definiált állomány lyukkártyán helyezkedik el, mégpedig közvetlenül a DD utasítást követően. (Ezeken a kártyákon egy FORTRAN nyelven megírt program található) a FORT nevű job lépés nevet ui. a FORTRAN fordítóprogram job lépésének szokás adni.

Példa 2: // GO.UJFILE DD DSN=FUJ5, UNIT=2314, VOL=SER=SZTEK2,
// DISP=(NEW, KEEP, DELETE),
// DCB=(LRECL=120, BLKSIZE=600, RECFM=FB),
// SPACE=(600, (200, 30), RLSE)

Magyarázat:

GO.UJFILE — az állomány programbeli azonosítója, amelyet a GO nevű job lépésben használunk

DSN=FUJ5 — az UJFILE állomány neve a lemezen FUJ5

UNIT=2314 — az állomány 29 Mbájt kapacitású mágneslemezen helyezkedik el

VOL=SER=SZTEK2 — az állomány az SZTEK2 nevű lemezen van

DISP=(NEW, KEEP, DELETE) — Rendelkezés az állományról

NEW — új, most kerül létrehozásra

KEEP — a job lépés sikeres befejezése után az állomány kerüljön megőrzésre (keep = megtartani)

DELETE — ha a job lépés befejezése nem sikeres, az állomány kerüljön törlésre (delete = törölni).

DCB — adatvezérlő blokk (Data Control Block), amelyben a létrehozandó állomány belső szerkezetét kell leírni.
LRECL logikai rekordhossz most 120 bájt
BLKSIZE blokkméret, most 600 bájt, vagyis 5 rekord = 1 blokk
RECFM rekordforma most FB vagyis fix hosszúságú blokkolt.

SPACE — Ha az új állományt mágneslemezen kívánjuk létrehozni, helyet kell kijelölni a számára.
600: ilyen hosszú blokkokat írunk fel (bájtban)
200: ennyi blokk felírását tervezzük (e két adatból a rendszer kiszámítja, hány sáv szükséges)
30: ha a felírandó blokkok száma több, mint 200, a rendszer 15 alkalommal még 30 blokknyi helyet biztosít (példánkban tehát összesen 650 blokkot képes felírni, többet nem)
RLSE ha az állomány felírása után még a 200 blokknyi hely sem került kihasználásra, a fel nem használt területet a rendszer visszaveszi.

Példa 3: //JOBLIB DD DSN=MIKI50, DISP=SHR

Magyarázat:

JOBLIB — felhasználói könyvtár megnyitását előíró utasítást definiál. Ez az egyetlen olyan DD utasítás, amely nem egy job lépésen belül helyezkedik el, hanem közvetlenül a JOB utasítás után, az első EXEC előtt. A könyvtár megnyitása a job idejére vonatkozik.

MIKI50 — a megnyitott könyvtár neve

DISP=SHR — az SHR (megosztani) paraméter hatására a könyvtárhoz egyidejűleg több program is hozzáférhet. (Ha pl. a P1 partícióban futó program használja a könyvtárat — erőforrásként hozzá van rendelve, — a vele párhuzamosan a P2 partícióban futó másik program egyidejűleg használhatja ugyanezt a könyvtárat. | ⊙ | 311 | 16 |

Szembetűnő, hogy a könyvtárat definiáló DD utasításban sem az adathordozó típusára (UNIT), sem az adathordozó azonosítására (VOL=SER) nem került sor.

Hogyan tudja akkor az operációs rendszer azonosítani és megtalálni ezt a könyvtárat? Ahhoz, hogy a kérdést meg tudjuk válaszolni, meg kell ismerkednünk a **rendszerkatalógus fogalmával.** A rendszerkatalógus (a SYSCTLG nevű állomány) egy speciális fájl a rendszerben. Feladata a gyakrabban használt állományok azonosításának segítése. Ahhoz, hogy ez létrejöhessen, az szükséges, hogy az állomány katalogizálásra kerüljön. Ez azt jelenti, hogy az állományt tartalmazó köteg neve és típusa (vagyis a UNIT és a VOL paraméter) bejegyzésre kerül a rendszerkatalógusba. Ha ezek után egy ilyen katalogizált állományra úgy hivatkozunk, hogy nem adjuk meg e paramétereket, az operációs rendszer automatikusan a rendszerkatalógusban keresi a hiányzó információkat. Mivel az állomány katalogizálva van, az információk rendelkezésre állnak, s az állomány elérhető. | ● | 304 | 17 |

A munkavezérlő nyelv segítségével egy teljes job összes erőforrásigénye meg-

adható. Vannak azonban olyan erőforrásigények, amelyek több jobnál állandóan ismétlődnek. Pl., ha egy magasszintű programozási nyelven megírt programot fordítani, szerkeszteni és futtatni akarunk, s a futtatáskor nincs a sornyomtatón és a kártyaolvasón kívül más fájlra szükség, akkor bármely, azonos nyelven megírt program ugyanazzal az erőforrásigénnyel lép fel, vagyis ugyanazt a munkavezérlő nyelvi utasítássorozatot kell megadni.

A 90. ábra példaképpen egy FORTRAN nyelvű program futtatásához szükséges három job lépést írja le.

```
//          EXEC PGM=IEYFORT,REGION=100K
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSLIN   DD DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//            SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//LKED  EXEC PGM=IEWL,REGION=132K,PARM='XREF,LET,LIST,SIZE=(132K,60K)',
//      COND=(4,LT,FORT)
//SYSLIB   DD DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),DCB=BLKSIZE=1024,
//            UNIT=(SYSDA,SEP=(SYSLIB)),SPACE=(1024,(35,14,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(100,10),RLSE),DCB=BLKSIZE=1024,
//            DSNAME=&SYSUT1
//SYSLIN   DD DSNAME=&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//FT05F001 DD DDNAME=SYSIN
//FT06F001 DD SYSOUT=A
//FT07F001 DD SYSOUT=B
```

90. ábra. A FORTGCLG eljárás munkavezérlő utasításai

Nyilvánvaló, hogy ennek a 15 munkavezérlő utasításnak gyakori használata kényelmetlen. Szükséges, hogy ez a használat valahogy egyszerűbb legyen. Ezt a célt szolgálják a **katalogizált eljárások.**

A katalogizált eljárás egy munkavezérlő nyelvi utasítássorozat, amely gyakran ismétlődő munkák erőforrásigényét írja le. Ez az utasítássorozat egy erre a célra szolgáló könyvtárban kerül elhelyezésre, ahonnan egy eljáráshívó utasítással kimásolható, aktivizálható. Ez az aktivizáló utasítás egy EXEC utasítás, amelyben azonban nem a program nevét, hanem az eljárás nevét adjuk meg. A 90. ábrán levő eljárás a következő utasítások valamelyikével hívható:

// EXEC PROC=FORTGCLG vagy // EXEC FORTGCLG

Mielőtt az elmondottakat egy példával szemléltetnénk, ismerkedjünk meg még két munkavezérlő nyelvi utasítással.

A /* (per csillag) utasítás

Feladata: A job struktúrában adatok vagy program végét jelöli. Ilyen utasítás található tehát mind a forrásprogramban, mind a feldolgozandó adatok végén. (A /* a kártyán levő állományok esetén a fájl vége jel.)

A // (per per) utasítás

Feladata: a job végét jelöli.

A 91. ábrán bemutatjuk, hogy a katalogizált eljárások használata mennyire leegyszerűsíti a munkavezérlő nyelv használatát. Az eljárás meghívásán kívül csak a program és az adatok helyére (adathordozójára) való DD hivatkozást kell külön megadni. (Tudjuk, hogy példánk esetén a * miatt mindkettő lyukkártyán található.) Ha azonban a program és az adatok is fent lennének egy-egy fájlban, ez a hivatkozás is része lehetne a katalogizált eljárásnak, mivel abban csak *-t tartalmazó DD kártya nem szerepelhet.

```
 //TPELDA22 JOB 123456,'PICI ORSI',CLASS=B
//FORTRAN EXEC FORTGCLG
//FORT.SYSIN DD *

      IDE KERUL A FORRASPROGRAM

/*
//GO.SYSIN DD *

      IDE KERULNEK AZ ADATOK

/*
//
```

91. ábra. FORTRAN „job" munkavezérlő utasításai

A katalogizált eljárásra való hivatkozás hatására az operációs rendszer az adott katalogizált eljárást *kimásolja* az eljáráskönyvtárból és az utasításokat a job-ban megadott további munkavezérlő nyelvi utasításokkal együtt úgy helyezi el az input várakozó sorban, hogy minden vezérlőutasítás a végrehajtás logikája szerinti helyre kerül. A 92. ábrán a XX-jellel kezdődő utasításokat az eljáráskönyvtárból, a //-el kezdődőeket a job leírásból vette a rendszer.

AZ OPERÁCIÓS RENDSZER A FELHASZNÁLÓ SZEMPONTJÁBÓL

Az eddigiekben választ kaptunk arra, hogy milyen fontosabb funkciói vannak az operációs rendszernek, s hogy miképpen tud a programozó kapcsolatot teremteni az operációs rendszerrel. Meg kell még vizsgálnunk, hogy miképpen valósulnak meg az operációs rendszer funkciói. Célszerű kiindulópont, ha mindezt a felhasználó szemszögéből tesszük, vagyis az operációs rendszer funkcióit aszerint csoportosítjuk, ahogyan azok a felhasználó számára jelentkeznek.

A *felhasználó szemszögéből vizsgálva, az* OS operációs rendszer három, jól körülhatárolható funkciót lát el, ezért programjai három csoportba sorolhatók. A három funkció: job kezelés; task kezelés és adatkezelés. Ennek megfelelően a felhasználó számára az OS

```
//          EXEC FORTGCLG,TIME.GO=(5,0)
XXFORT      EXEC PGM=IEYFORT,REGION=100K
XXSYSPRINT DD  SYSOUT=A
XXSYSPUNCH DD  SYSOUT=B
XXSYSLIN    DD  DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
XX              SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//FORT.SYSIN DD *
XXLKED EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LET,LIST),COND=(4,LT,FORT)
XXSYSLIB    DD  DSNAME=SYS1.FORTLIB,DISP=SHR
XXSYSLMOD   DD  DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
XX              SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
XXSYSPRINT DD  SYSOUT=A
XXSYSUT1    DD  UNIT=SYSDA,SPACE=(1024,(100,10),RLSE),DCB=BLKSIZE=1024,
XX              DSNAME=&SYSUT1
XXSYSLIN    DD  DSNAME=&LOADSET,DISP=(OLD,DELETE)
XX          DD  DDNAME=SYSIN
XXGO EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
XXFT05F001 DD  DDNAME=SYSIN
XXFT06F001 DD  SYSOUT=A
XXFT07F001 DD  SYSOUT=B
//GO.SYSIN DD *
```

92. ábra. A kifejtett eljárás listája

— a job-kezelő programok,
— a task-kezelő programok,
— az adatkezelő programok köréből tevődik össze.
A továbbiakban ilyen részletezésben tekintjük át az OS működését.

Job-kezelő programok

A következő táblázatban összefoglaljuk a legfontosabb job-kezelő programokat és azok feladatait.

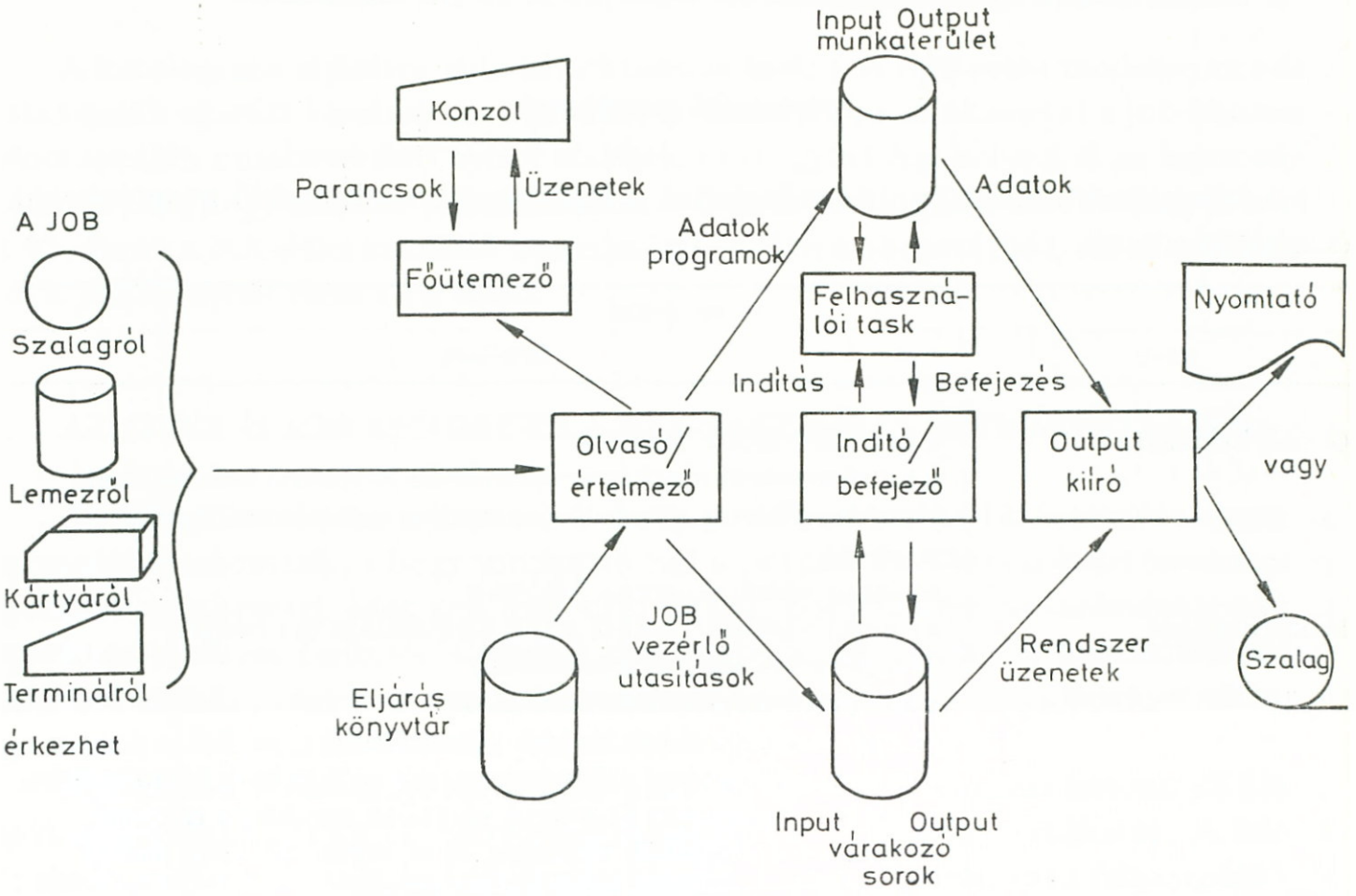
A program	
neve	feladata
Olvasó/Értelmező	a beérkező jobok fogadása a munkavezérlő utasítások felismerése és formai (szintaktikai) elemzése a hívott eljárások bemásolása a rendszerkönyvtárból a job-ba, az input várakozó sorok feltöltése
Indító/Befejező	az input várakozó sorból a job kiválasztása (az osztály és a prioritás szerinti sorrendben) a job perifériaigényének biztosítása az egymást követő job lépések végrehajtása, job lépés befejeztével a feleslegessé vált erőforrások felszabadítása a job befejezése után az output várakozó sorokba a rendszer output információinak betöltése, és az input várakozó sorból a munkára vonatkozó információk törlése

A program	
neve	feladata
Output kiíró	az output várakozó sorokból a rendszerüzenetek és a SYSOUT állományokból a felhasználó outputjainak kiírása a kívánt perifériára
E három program együttesen alkotja a *job-ütemezőt*.	
Főütemező	a job-ütemező programjainak indítása és megállítása, az operátorral való kapcsolattartás: az operátori parancsok feldolgozása és továbbítása a rendszer által az operátornak küldött üzenetek kezelése és továbbítása a konzolra input és output várakozó sorok létrehozása és kezelése

A job kezelés programjainak működését, a jobok feldolgozásának menetét a 93. ábrán kísérhetjük nyomon.

Az ábrán láthatjuk, hogy a beérkező jobot az olvasó/értelmező fogadja. Leválasztja a munkavezérlő utasításokat, amelyeket az első két bájton levő // jelről ismer fel. Valamennyit ellenőrzi szintaktikailag, s amennyiben hibás utasítást talál a job, a további feldolgozás elmarad, a sornyomtatóra „JCL hiba" kerül kiírásra.

Amennyiben a munkavezérlő utasítások formailag helyesek voltak, az ezekből összeállított táblázatokat elhelyezi a job osztályának megfelelő várakozás sorba.



93. ábra. A jobok feldolgozási folyamata

A job-ban levő programot és adatokat egy átmeneti tárolóhelyen helyezi el, s ennek címét szintén rögzíti a job várakozó sor egyik táblázatában.

Az olvasó/értelmező működéséhez tárterületre van szükség. Tudjuk, hogy a programok számára a tárterület partíciók formájában állhat rendelkezésre. Az olvasó/értelmező működtetéséhez alapvetően kétféle módon lehet biztosítani a szükséges tárterületet.

Az első esetben az olvasó/értelmező önálló partícióban működik. Ilyenkor *rezidens olvasó/értelmezőről* beszélünk. Előnye, hogy aktív állapotában újbóli betöltés nélkül több jobot is képes beolvasni és értelmezni. Aktivizálása a multiprogramozás ismert lefolyása szerint a partíció aktivizálásával történik.

A másik esetben az olvasó/értelmező nem rezidens, tehát nem foglal le egy önálló partíciót. Betöltésére akkor kerül sor, ha szükség van rá. Ilyen esetben működhet a rendszerhez rendelve, vagyis amíg van a méretének megfelelő szabad partíció, abban tevékenykedik. Működhet felhasználói partícióhoz rendelve, vagyis működése után átadja helyét a felhasználói jobnak. Nyilvánvaló, hogy ez utóbbi esetben, bár területből kevesebb kell, hosszabb ideig tart működése, hiszen a betöltésnek is van időigénye.

Amennyiben a központi tár 512 Kbájtnál nagyobb, célszerű, ha az olvasó/értelmező program rezidens módon működik. Ha egy várakozó sorba az olvasó/értelmező már elhelyezett job-információkat, lehetővé válik — akár az olvasó/értelmezővel párhuzamosan — e várakozó sorhoz tartozó partícióban a job indítása.

Ehhez az *indító/befejező* program működésére van szükség. Az indító/befejező aktivizálása után megvizsgálja, hogy az egyes job osztályokhoz tartozó várakozó sorokban található-e végrehajtásra váró job. Az egyes várakozó sorok lekérdezése a job osztályok fontossági sorrendjében történik.

Amikor az indító/befejező kiválaszt egy job-ot, igyekszik megteremteni azt az erőforrás környezetet, amely a job futásához szükséges. Ennek érdekében megvizsgálja, hogy a job által kért input állományok és eszközök rendelkezésre állnak-e, ill. az output számára szükséges terület biztosítható-e. Ha az erőforrások között olyan fordul elő, amelyet az adott pillanatban nem tud hozzárendelni, mert egy másik job számára foglalt (és nem osztható meg), az operátor számára üzenetet küld. Az operátor ekkor általában várakoztatja az előkészítést.

▲	300	18

Az erőforrások sikeres biztosítása után a felügyelőprogram az indító/befejező helyére — vagyis a felhasználói partícióba — betölti az első job lépésben definiált programot, amely a vezérlést is megkapja.

Egy-egy job lépés befejezése után ismét az indító/befejező kapja meg a vezérlést, rendelkezik az adatállományokról a munkavezérlésben előírtak alapján.

●	305	19

Elvégzi a lekötött perifériák felszabadítását, feldolgozza az állapotkódot.

▲	300	20

Amennyiben nemcsak a job lépés fejeződött be, hanem maga a job is (vagyis a job utolsó lépése ért véget), az output várakozó sorokba táblázatokat helyez el. Az egyik táblázat a rendszerüzenet blokk, amely a job futása során keletkezett valamennyi

rendszerüzenetet tartalmazza, a másik az adatállomány blokk, amelyben a job-ban SYSOUT utasítással megadott állományok találhatók.

Multiprogramozás során a keletkező outputok nem kerülnek közvetlenül nyomtató berendezésre, hanem ún. SYSOUT osztályokban — mágneslemez területeken — kerülnek elhelyezésre. Ennek alapvetően az a célja, hogy az időbeli fedéssel létrejövő output rekordok ne keveredjenek össze a sornyomtatón, ill. az egyetlen job miatt ne kelljen a többinek a nyomtatóra várnia. ▲ 301 21 ● 305 22

A SYSOUT osztályokban levő adatállomány blokkok és a rendszerüzenet blokkok nyomtatóra vitelét az *output kiíró* valósítja meg. Az output kiíró is működhet rezidens módon, önálló partícióban, és mód van arra, hogy csak szükség esetén indítsa el az operátor felhasználói partícióból.

Ha az operátor egy output kiírót aktivizál, az megvizsgálja, hogy a hozzá rendelt output osztályban várakozik-e nyomtatási output.

Az output kiíró az indító/befejező által készített blokkok alapján tájékozódik. A kiírás befejezése után az output kiíró megszünteti az output sort, s ezzel a kiírt jobról a rendszerben csak az elszámoláshoz szükséges információk maradnak.

Task-kezelő programok (Supervisor)

Amikor az eddigiek során a felügyelőprogram, ill. a supervisor kifejezések előfordultak, tulajdonképpen azokról a programokról volt szó, amelyek a task-kezelés feladatát látják el. E három fogalmat a továbbiakban is mint egymás szinonímáit fogjuk használni. A felügyelőprogram legfontosabb feladatai a következők:

— a megszakítások kezelése,
— a központi tár kezelése,
— a központi tár tartalmának nyilvántartása,
— a feladatok (task-ok) kezelése,
— a rendszer órájának kezelése.

A *megszakításokról* már a korábbiakban szó volt. Tudjuk, hogy egy megszakítás akkor lép fel, amikor egy futó (aktív) program további működése elé akadály gördül. Ilyen „akadály" létrejöttét a futó program környezetében beállt változás okozza, amennyiben e változás kihatással van a futó program erőforrásigényére. A megszakítás teszi lehetővé a felügyelőprogram számára, hogy reagáljon ezekre a változásokra.

Ismerjük már a legfontosabb megszakítási okokat is:

— SVC megszakítás (a futó program adatátvitelt kér),
— input-output megszakítás (egy korábban kért adatátvitel befejeződött),
— program- vagy géphibából adódó megszakítás,
— külső megszakítás (az operátor konzolról beavatkozik).

Külső megszakításnak számít a rendszer órájáról érkező megszakítás is. ⊙ 312 23

A központi tár kezelése a következőket jelenti:
— a tárterületek kijelölése,
— a foglalt területek felszabadítása,
— a szabad területek nyilvántartása.

A központi tár tartalmának kezelése a programok betöltését és a betöltött programok nyilvántartását jelenti.

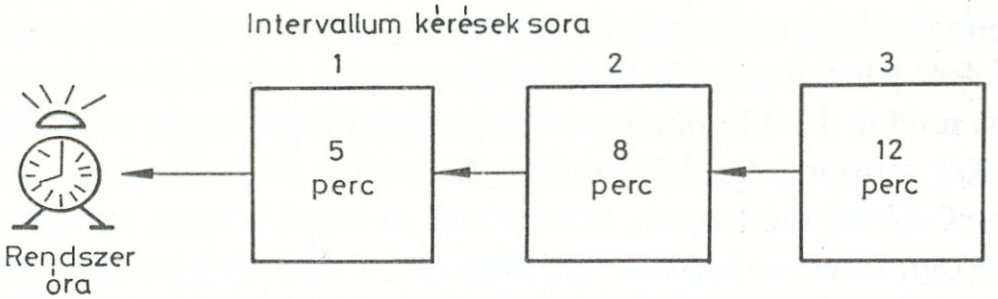
A tárban levő betöltött modulok nyilvántartása ún. igény-blokkok segítségével történik. Az igényblokkok egymással és az aktuális task-kal kapcsolatban vannak.

A **task-kezelés** során történik az erőforrásért versengő taskok kezelése. Az egyes taskokat a rendszerben egy-egy blokk, az ún. task vezérlési blokk képviseli. Ezek a blokkok a belső futási prioritás szerint követik egymást. Mindig az a task aktív, amelyik a legmagasabb prioritási számmal rendelkezik.

A rendszer órájának kezelése biztosítja a mindenkori hozzáférést a napi dátumhoz és időhöz, valamint lehetővé teszi, hogy az egyes programokhoz időlimitet lehessen hozzárendelni. ● 306 24

A pillanatnyi idő lekérdezése a TIME makro hívásával lehetséges, amely századmásodperc pontossággal adja meg a pontos időt a 0. általános célú regiszterbe. A napi dátumot (a nap sorszámát a tárgyéven belül) az operátornak kell reggel betöltenie.

Az intervallumok kérése a STIMER makróval történhet. Miután egyidőben több task is kérhet az órától intervallum szolgáltatást, ezeket a kéréseket sorba kell állítani. A sor elején az a kérés áll, amelyhez tartozó intervallum leghamarabb jár le, s így minden igény kielégíthető. Ezt a sorbanállást szemlélteti a 94. ábra. ◉ 313 25



94. ábra. Várakozás a rendszer óraszolgáltatásaira

Adatkezelő programok

A felhasználói programokban levő adatátviteli utasítások hatására nem jön létre tényleges (fizikai) adatátvitel. Az állományokkal kapcsolatos fizikai tevékenységeket az operációs rendszer adatkezelő programjai végzik el.

A következő oldalon lévő táblázatból láthatjuk az adatkezelő programok tagozódását és feladataikat.

Az input-output felügyelőprogram bonyolítja a fizikai szintű input-output műveleteket. Feladata az input-output hiba miatt megszakadt műveletek újraindítása is és a hibák bejegyzése az erre a célra kijelölt rendszer fájlba. Szolgáltatásait a magas

A program	
neve	feladata
Input-output felügyelőprogram	adatátvitel fizikai bonyolítása input-output hiba miatt megszakadt művelet újraindítása
Elérési módok	a különböző szervezettségű állományok kezelése
OPEN, CLOSE	állományok megnyitása, lezárása
DADSM	mágneslemez területek kezelése
Katalóguskezelő	a katalógus (SYSCTLG) kezelése

szintű programozási nyelvben kiadott adatátviteli utasítások kezdeményezésére az SVC megszakítások hatására végzi el. Az adatátvitelhez szükséges információkat a fordítás során felépülő táblázatok, valamint a rendszerben kialakított vezérlési blokkok biztosítják. Az adatátvitel lebonyolításánál fontos szerepet játszanak a **csatornaprogramok,** amelyek a csatornák működését vezérlik.

A csatornaprogramok csatornautasítás szavakból (CCW) állnak. A csatornautasítás-szóban levő fontosabb információk:

— utasításkód (mit kell elvégeznie a csatornának),

— adatcím (az átvitelre kerülő adat tárcíme),

— bájtszámláló (az átvitelre kerülő adat hossza).

A csatorna, amikor megkapja az adatátvitel megindítására vonatkozó utasítást, a központi tár egy meghatározott címéről kiolvassa a csatornaprogram kezdőcímét. Ezt a kezdőcímet az input-output felügyelőprogram helyezi el, és ennek a feladata a csatorna indítását jelentő utasítás kiadása is.

Az **elérési módok** a különböző szervezettségű állományok programból való kezelését segítik. Két csoportjukat különböztetjük meg:

— *egyszerű* elérési módszerek, amelyeknél az input/output művelet befejezéséig a felhasználói program visszakapja a vezérlést. A művelet befejezését a programozónak kell ellenőriznie. Csak blokkokat tudnak mozgatni, azok rekordokká bontása szintén a programozó feladata;

— a *bővített* elérési módszerek megvárják az input/output művelet befejezését, és képesek a blokkokban levő rekordok kezelésére is.

A következő táblázatban összefoglaljuk az elérési módszereket.

	Egyszerű	Bővített
Szekvenciális	BSAM*	QSAM*
Particionált (könyvtári)	BPAM*	—
Index szekvenciális	BISAM	QISAM
Direkt	BDAM	—

* a rendszerprogramok is használják, ezért rezidensnek kell lenniök.

A távadatfeldolgozást a ● 306 26 következő elérési módszerek támogatják: BTAM, QTAM, TCAM.

Az OPEN és CLOSE programoknak fő feladata az adatállományok megnyitása és lezárása. Amikor egy adatállomány egy task-hoz erőforrásként hozzá van rendelve, az állományt az első hozzányúlás előtt (első olvasás vagy írás előtt) meg kell nyitni, és a task befejezése után — mielőtt az állományt a befejezőprogram elengedi — le kell zárni.

Az állomány megnyitásakor az állomány címkéit kell ellenőrizni. Az ellenőrzéshez szükséges információk forrása az állományra vonatkozó DD utasítás, amelyet a job fájl vezérlőblokkban őriz a rendszer. Az állomány nevének azonosítása után a megőrzési időt ellenőrzi az OPEN program. Azok az állományok, amelyeknek a megőrzési ideje még nem járt le, csak az operátor engedélyével írhatók felül.

Az új állományok esetén az OPEN program írja fel az adatállomány címkéket. Az ehhez szükséges információkat alapvetően a programból és az állományra vonatkozó DD utasításból veszi.

A CLOSE programmal történik az állományok lezárása, azok utolsó használata után. Input állományok esetében ismételten ellenőrzésre kerülnek a címkék (ez esetben a végcímkék), output állományoknál pedig sor kerül ezek felírására. A szükséges információk a job fájl vezérlőblokkban állnak rendelkezésre.

A mágneslemez területeket kezelő program *(DADSM)* végzi el azokat a tevékenységeket, amelyeket a SPACE paraméterrel írunk elő új állományok lemezen való létrehozásakor a DD utasításban. Így:

— az elsődleges terület kijelölése;

— a másodlagos területigény biztosítása (legfeljebb 15 alkalommal);

— az üresen maradt terület felszabadítása.

Ezeken kívül ez a program végzi el:

— az állományok törlését (a VTOC-ból az állományra vonatkozó bejegyzések törlését);

— állományok átnevezését (a VTOC-ban az állományra vonatkozó névbejegyzés megváltoztatását);

— a lemezen levő szabad területek nyilvántartását.

Az operációs rendszert alkotó programok, amelyeket eddig tárgyaltunk, a vezérlőprogramok körébe tartoznak. A feldolgozó programok körében van egy csoport, amelyről ezideig nem volt szó. E programok funkcióikat tekintve operációs rendszer közelségű feladatokat látnak el. Ezek a **szervizprogramok.**

A SZERVIZPROGRAMOK

Szolgáltatásaik elsősorban a számítógépet üzemeltetők számára jelentenek segítséget, de sokszor válik szükségessé a felhasználó számára is, hogy a szervizprogramok különböző szolgáltatásait igénybe vegyék.

A teljesség igénye nélkül bemutatunk néhány olyan feladatot, amely szervizprogramok segítségével megoldható.

Mágnesszalagokra és mágneslemezekre való adatrögzítés előtt gondoskodni kell arról, hogy a szalag-, ill. a lemezkötegek elejére *felkerüljenek a rendszer számára szükséges információk.* Mágnesszalagok *címkézését* és mágneslemezek *inicializálását* a megfelelő szervizprogramokkal lehet elvégeztetni.

Ha egy mágneslemez csomag valamely felületén *egy sáv megsérül*, a *tartalék sávok* közül ki kell jelölni egy új sávot, és a hibás sávot ki kell iktatni. Erre a célra is megfelelő program áll rendelkezésre.

Igen sokszor van szükség arra, hogy különböző *mágnesszalagok vagy mágneslemezek tartalmát kilistázhassuk*, fájlokat másoljunk egy háttértárolóról egy másikra. Hasonlóan gyakori igény a különböző könyvtárakban levő rekordok megváltoztatása (programok felújítása). Mindezen feladatok elvégzésére is szervizprogramok állnak rendelkezésre.

A szervizprogramokat két nagy csoportba szokás sorolni. Ezek:

— a **rendszerfüggetlen** szervizprogramok, amelyek akkor is működtethetők, ha nincs betöltve operációs rendszer;

— a **rendszertől függő** szervizprogramok, amelyek az operációs rendszer felügyelete alatt futnak.

A rendszertől függő szervizprogramok egyik csoportját képezik azok, amelyek a rendszer saját adatait és adathordozóit kezelik. A másik csoportba a felhasználók adatainak kezelését biztosító szervizprogramok tartoznak.

Az elmondottakat két példával szemléltetjük. A rendszer saját adatai közé tartozik pl. a mágneslemezek elején található VTOC. Ennek tartalma — a lemezen levő állományok neve és azok jellemzői — sokszor szükségesek az üzemeltető számára.

A VTOC listázását biztosítja az IEHLIST szervizprogram. A felhasználó számára — ha nem akarja, hogy a mágneslemezen levő könyvtárai beteljenek — azok karbantartása fontos feladat. *Könyvtárak karbantartására* (feleslegessé vált programok törlése, a könyvtárba „szétszórtan" levő programok [tagok] egymás mögé tömörítése) az IEBCOPY szervizprogrammal kényelmesen megvalósítható.

Annak illusztrálására, hogy egy-egy szervizprogramot hogyan lehet használatba venni, nézzük meg részletesebben a már említett IEHLIST használatát.

A programnak három funkciója van:

— a VTOC tartalmának kiírása,

— a könyvtár tartalomjegyzékének kiíratása,

— a katalógus tartalmának kiíratása.

A három funkcióból a szükséges kiválasztása a szervizprogram vezérlőutasításaiból való választással történik. Lemeztartalomjegyzék kiíratása a LISTVTOC vezérlőutasítás alkalmazásával történik. (A könyvtár tartalomjegyzékét a LISTPDS utasítással kérhető.)

Az egyes vezérlőutasításoknak további paraméterei vannak, pl. a LISTVTOC utasítás mellé meg kell adni annak a kötetnek a nevét, amelynek VTOC-ját listáztatni

kívánjuk. Kérhetjük, hogy ne teljes listát kapjunk, ill. kiválaszthatjuk, hogy milyen formában történjék a listázás (ún. szerkesztett vagy nem szerkesztett, hexadecimális formában).

A program aktivizálása — hasonlóan bármely más program aktivizálásához — a munkavezérlő nyelvi utasításokkal történik: a programhívás EXEC utasítással, a szükséges állományok DD utasítással. ● 306 28 ⊙ 313 27

A MIKROSZÁMÍTÓGÉPEK OPERÁCIÓS RENDSZEREIRŐL

A mikroszámítógépek szoftverjét, pontosabban szoftverrel való ellátottságát vizsgálva nagyon sokszínű képet láthatunk. Vannak berendezések — elsősorban az olcsóbb, leginkább csak a programozás tanulására alkalmas gépek —, amelyek szoftver ellátottsága gyakorlatilag a BASIC interpreterből áll. A sor másik végén azok a professzionálisnak nevezett személyi számítógépek állnak, amelyek nem csak hardver kiépítettségükkel (központi tárméret, háttértár kapacitás), hanem szoftver ellátottságukkal is alkalmasak arra, hogy a szó valódi értelmében számítógépnek tekintsük őket.

A mikrogépek szoftverjét alkotó programok, feladataik szerint nagyjából ugyanúgy csoportosíthatók, mint a nagy számítógépek szoftverje: megtalálható az operációs rendszer, a felhasználót támogató szervizprogramok, valamint a fordítóprogramok, ill. az interpreterek. Ez utóbbi kettővel a VI. fejezetben foglalkoztunk, ezért csak az operációs rendszer és a szervizprogramok funkcióival ismerkedünk meg.

A mikroszámítógépek általában egyfelhasználós operációs rendszerrel rendelkeznek, ami azt jelenti, hogy egy mikrogépen egyidejűleg csak egyvalaki tud dolgozni. (Az utóbbi időben megjelentek a többfelhasználós mikrogépek is.)

Maga az operációs rendszer általában két alapvető részből áll. Az egyik a **parancsértelmező**, a másik a **fájl rendszer**. A parancsértelmező feladata a felhasználó által bevitt parancsok értelmezése és végrehajtása. Ezek a parancsok hasonló szerepet játszanak a mikrogépeknél, mint a munkavezérlő nyelvi utasítások játszanak a nagygépes operációs rendszereknél: segítségükkel tart kapcsolatot a felhasználó az operációs rendszerrel.

A parancsok egy részét maga a parancsértelmező végre tudja hajtani. Ilyen parancsok pl.: egy fájl (program) törlése, egy fájl (program) átnevezése, a fájl katalógus kezelése stb. Más parancsokat közvetlenül nem tud végrehajtani. Ezek a parancsok általában egy program nevével azonosak (pl. fordítóprogram vagy interpreter hívás). Ebben az esetben a parancsértelmező megkeresi az adott nevű programot, és aktivizálja.

Az operációs rendszer másik része a fájl rendszer. A fájl rendszer a hozzá tartozó input-output kezelő rendszer segítségével a hajlékony mágneslemezen levő könyvtárban, ill. annak katalógusában (tartalomjegyzékben) tud műveletet végezni. Az operációs rendszert a szervizprogramok szolgáltatásai teszik teljessé.

A mikroszámítógépek szervizprogramjai nagyon fontos szerepet játszanak a gép kényelmes használatában. A következőkben összefoglalt feladatok csupán a legfontosabbak azok között, amelyeket egy mikroszámítógép szervizprogramjai biztosítanak a felhasználó számára.

— *Szövegszerkesztés.* A szövegszerkesztők a képernyőn megjelenített bármilyen szöveg (nem csak program) javítását, átszerkesztését, változtatását lehetővé teszik. A szövegszerkesztők segítségével lehet a szöveg vagy a program billentyűzeten át való bevitelét is elvégezni (begépelni a szöveget).

— *Szövegnyomtatás.* A végső formára hozott szöveget a nyomtatóprogramok segítségével lehet a mátrixnyomtatón megjeleníteni. Előírható a nyomtatott betűk nagysága, típusa.

— *Másolás.* Kazettáról hajlékony lemezre vagy hajlékony lemezről másik hajlékony lemezre való program-, ill. szövegmásolást végeznek a másoló programok.

— *Nyomkövetés.* A program futása közben a nyomkövető és a hibakereső programok segítségével válik lehetővé, hogy a tár és a regiszterek tartalma megjelenjen a képernyőn. Ezzel az eljárással a programozó nyomon tudja követni a program futását, ami a programkipróbálás során nagyon nagy segítséget jelent.

A mikroszámítógép operációs rendszere a gép bekapcsolásával automatikusan aktivizálódik. Ezt az automatikus aktivizálódást többféle módon oldották meg. Egyik jellemző megoldási mód, hogy a bekapcsolás után a mágneslemez első szektorának tartalma bemásolódik a gépbe. Ezen az első szektoron egy betöltőprogram van elhelyezve. Ez a betöltő képes azután arra, hogy a teljes operációs rendszert betöltse a tárba.

Azok a mikrogépek, amelyek az előbbiekben vázolt szoftver lehetőséggel rendelkeznek — megfelelő hardver kiépítettséggel — már felveszik a versenyt a hagyományos értelemben „nagyszámítógépnek" nevezett berendezések több kategóriájával.

Kérdések

1. Mi az operációs rendszer?
2. Hogyan csoportosíthatók az operációs rendszerek?
3. Mi jellemzi a kötegelt feldolgozást biztosító rendszereket?
4. Mi jellemzi a többletfelhasználós időosztásos rendszereket?
5. Mi a job (munka)?
6. Mi a task (feladat)?
7. Mikor beszélünk multiprogramozásról?
8. Mit nevezünk partíciónak, mi a szerepe multiprogramozáskor?
9. Mi a job osztály?
10. Mi a megszakítás, mi történik egy megszakítás felléptekor?
11. Milyen fontosabb megszakítási okokat ismerünk?
12. Hogyan tud a felhasználó kapcsolatot teremteni az operációs rendszerrel?

13. Milyen munkavezérlő utasításokat ismerünk?
14. Mi a JOB utasítás funkciója?
15. Mi az EXEC utasítás funkciója?
16. Mi a DD utasítás funkciója?

Feladatok

1. Írd fel azt a job utasítást, amely az MTKFTC22 nevű jobhoz tartozik, elszámolási száma 123456.

2. Írd fel azt az EXEC utasítást, amely az IEWL nevű programot indítja, akkor ha a FORT nevű job lépés visszatérítési kódja legfeljebb 4.

3. Írd fel azt a DD utasítást, amely a következő állomány leírásához szükséges: A KARAK2 nevű, 2314 típusú mágneslemezen levő VAHUR nevű állomány, amelynek programbeli neve: UNKAOO. A fájl már létezik, feldolgozás után is megtartjuk.

4. A programban OUTPUT nevű állományban 2000 darab, egyenként 40 szó (!) hosszúságú rekordot írunk fel. Azt akarjuk, hogy 10 szó alkosson egy blokkot a VUKKAG nevű lemezen. Az állományt sikeres felírás esetén megőrizzük, sikertelen felírás esetén eltöröljük.

X. FEJEZET

Számítóközpont

Kilenc fejezeten át ismerkedtünk a számítástechnika alapjaival: a számítógépek hardverjének felépítésével, a szoftver funkcióival, a programtervezés és a programozás legfontosabb elemeivel. Kilenc fejezeten át próbáltunk behatolni abba a világba, amelynek eredményei alapvetően befolyásolják (vagy rövidesen befolyásolni fogják) életünket, munkánkat, szabadidőnket.

Van-e még olyan ismeretanyag, amely részét képezi az alapvető számítástechnikai ismereteknek, és az eddigiek során nem találkoztunk vele? Bizonyára van, hiszen pl. nem ismerkedtünk meg mélyebben egyetlen programozási nyelvvel sem. Talán szerencsésebb az előző kérdés, ha hozzátesszük: „az érvényben levő tantervi célkitűzések keretei között".

Az eddigiekben a számítógépet mint egy *önálló* berendezést vizsgáltuk. Megismertük hardver és szoftver összetevőit, programozásának egyes kérdéseit. Az operációs rendszerrel kapcsolatosan szó volt a fontosabb működési módokról.

A számítógépnek azonban — mint minden dolognak a világon — *környezete* is van. A jól működő számítógép pedig megfelelő formában *illeszkedik* ebbe a környezetbe.

Manapság, amikor jelentős teljesítményű professzionális személyi számítógépek találhatók az íróasztalokon vagy az irodák sarkaiban egy-egy kisasztalon, s a gépnél helyet foglaló felhasználó saját környezetében tud dolgozni, furcsának tűnik, hogy sok számítógép üzemeltetéséhez speciális feltételeket kell teremteni.

A nagyobb számítógépek mindegyike ún. számítógép teremben vagy röviden *gépteremben* van elhelyezve, s ez a gépterem része egy, a gép működtetésével foglalkozó szervezeti egységnek. Ezt a szervezeti egységet *számítóközpontnak* nevezzük. **Mi a számítóközpont?**

A számítóközpont a nagy számítógépek üzemeltetéséhez szükséges helyiségek, berendezések összessége, amelyet megfelelő felkészültségű személyzet használ, ill. üzemeltet. Legtágabb értelemben a számítóközpont feladatkörébe a gép tényleges üzemeltetésén kívül beletartoznak a számítógépes szervezéssel, programozással, rendszerfejlesztéssel kapcsolatos feladatok is. Másrészt léteznek olyan szervezetek — és ezeket is sokszor számítóközpontnak nevezik —, amelyek kizárólag a számítógép üzemeltetésével kapcsolatos feladatokat látják el. ● | 314 | 1

A számítóközpont szintén kapcsolódik a környezetéhez. Ennek a kapcsolódásnak a megjelenési formája nagyon sokféle lehet, attól függően, hogy *milyen céllal*, milyen *feladatok ellátásának* érdekében hozták létre a számítóközpontot.

A SZÁMÍTÓKÖZPONTOK CSOPORTOSÍTÁSA

A számítóközpontok csoportosításában többféle szempont lehetséges, így pl.:

— a számítóközpont célja szerint (oktatás, vállalati termelésirányítás, statisztikai feldolgozás, bérmunka stb.);

— az üzemeltetett gép vagy gépek nagysága, típusa szerint (kis-, közepes, nagygép, ESZR bázisú, nem ESZR bázisú stb.);

— a számítóközpontot üzemeltető szervezet jellege szerint (vállalat, oktatási intézmény, szervezőintézet, kutatóintézet, kórház stb.);

— a számítóközpontban dolgozók létszáma szerint (10—15 főstől a 100—150 fős intézményig).

Ezek a szempontok persze egymásra is hatással vannak, hiszen nyilvánvaló, hogy pl. egy bérmunka jellegű számítóközpontnak nagyteljesítményű géppel kell rendelkeznie, létszáma pedig megközelíti egy középvállalat létszámát.

Hazánkban a jellegzetes számítóközpont-típusok közé tartozik:

— a vállalati számítóközpont;

— a számítástechnikai bérmunkát végző, szervező vállalat számítóközpontja;

— az oktatási intézmény (pl. egyetem) számítóközpontja.

A vállalati számítóközpont

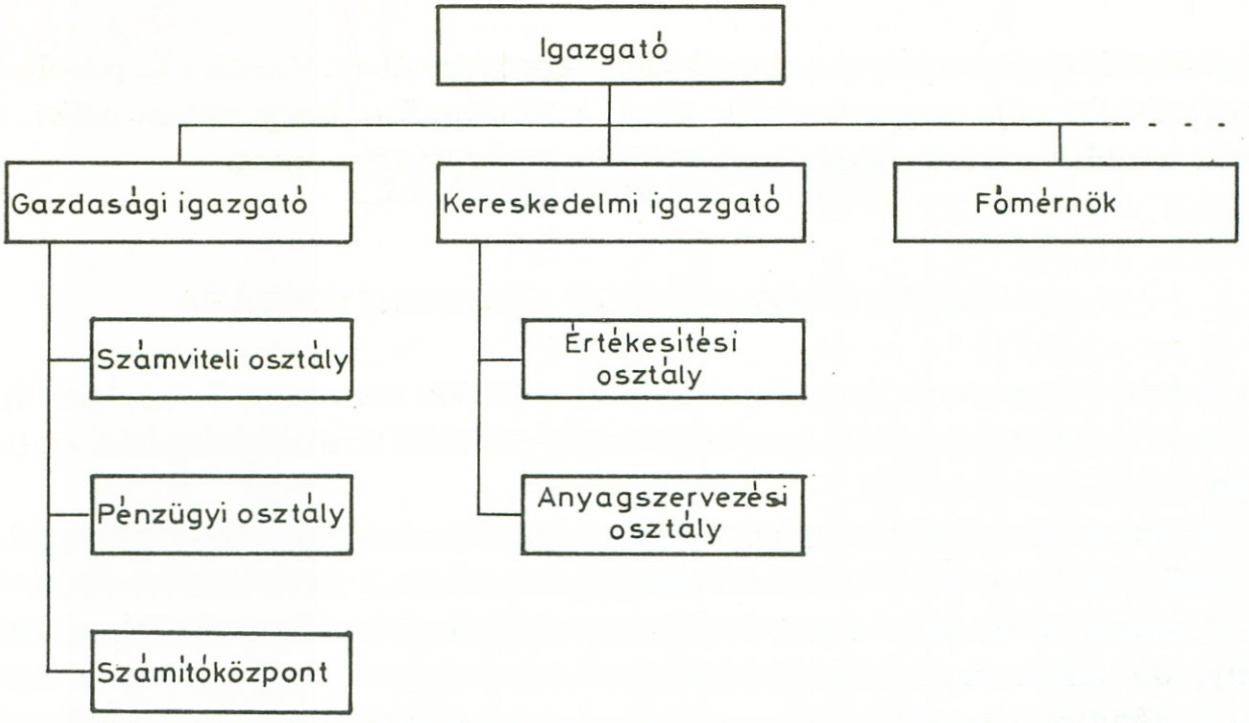
A számítástechnika gazdasági életben való térhódításának szükségszerű velejárója, hogy egyre több vállalatnak van saját számítógépe. A gépeket üzemeltető szervezetek (számítóközpontok) különböző módon és formában illeszkednek a vállalati szervezetbe.

Az egyik gyakori forma, amikor a számítóközpontot a fő felhasználók egyikének a hatáskörében helyezik el, értelemszerűen annál, amelyiknek a feldolgozási igénye a legnagyobb. Így a számítóközpont kerülhet pl. a gazdasági igazgató felügyelete alá vagy pl. a kereskedelmi igazgatóhelyettes alá stb. ▲ 314 2

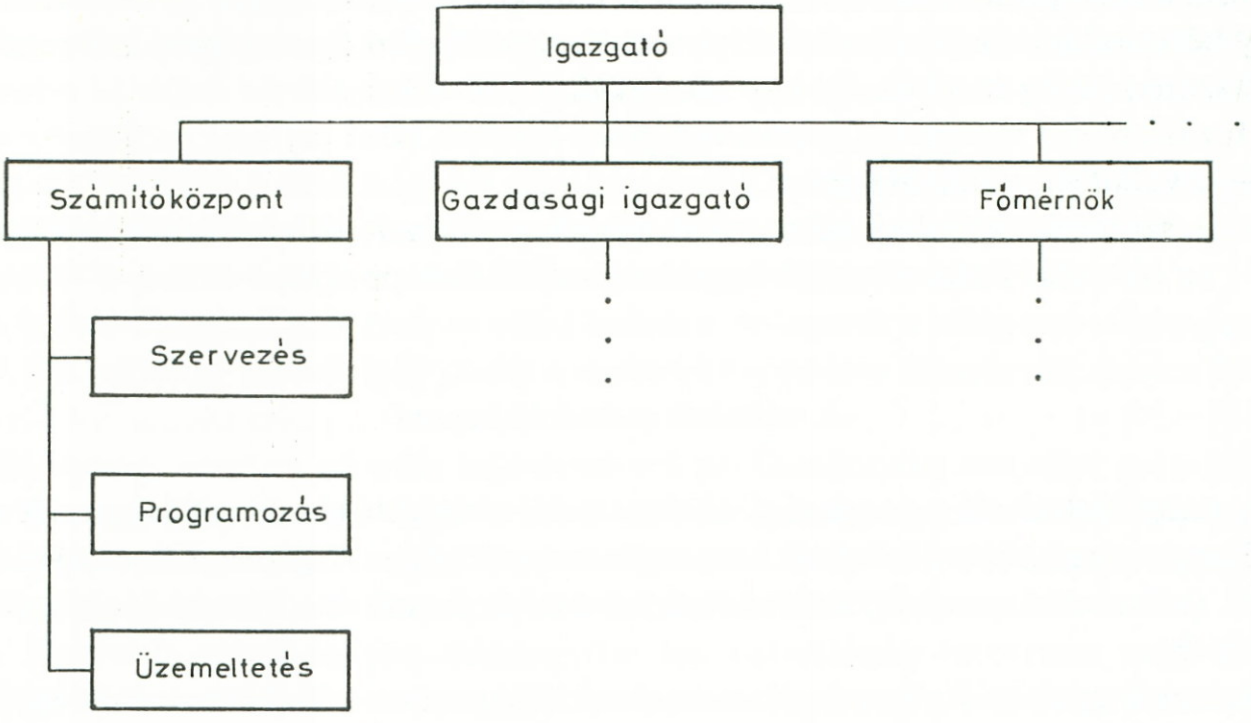
Egy ilyen vállalat szervezeti felépítését szemlélteti a 95. ábra.

Ez persze sok ellentmondásnak és viszálynak lehet a forrása, elsősorban akkor, amikor a vállalat más részlegei is „felnőnek" a számítógép használatához. Ezekben az esetekben — meg minden olyan esetben, amikor a számítógép által végzett munka közvetlen hatással van a vállalat életére — a számítóközpontot mint önálló szervezeti egységet közvetlenül az igazgató hatáskörébe rendelik.

Ilyen szervezeti felépítést szemléltet a 96. ábra.



95. ábra. Számítóközpont a legfontosabb felhasználó hatáskörébe rendelve



96. ábra. Számítóközpont az igazgató hatáskörébe rendelve

A vállalati számítóközpontok ***belső felépítése*** szintén sokban függ attól, hogy a számítóközpont milyen szerepet játszik a vállalat életében. Pl. egy olyan központ, amelyet középnagyságúnak tekinthetünk és közvetlenül az igazgató alá van rendelve, a következő főbb részlegekre tagozódhat:

— számítóközpont vezető;
— adminisztráció, titkárság;
— üzemeltetési vezető;
— hardver karbantartók;

— számítógép-kezelők (operátorok);
— diszpécserek;
— rendszerfelelősök;
— adatelőkészítők, adatrögzítők;
— fejlesztési vezető (programozók, rendszerprogramozók, szervezők).

A vállalati számítóközpontok közös jellemzője, hogy működésükkel a vállalati termelő folyamatot, és az azt kísérő információs folyamatot (pénzügyek, könyvelés) támogatják. ◎ | 317 | 3

A számítástechnikai bérmunkát végző, szervező vállalatok számítóközpontja

Ezek a vállalatok abból a célból létesültek és működnek, hogy más vállalatok számára számítástechnikai szolgáltatásokat végezzenek. Az ilyen bérmunka vállalatok „termelése", tehát magának a számítástechnikai munkának a végzése. Ez azt jelenti, hogy a megbízó részére a rendszer kialakításától kezdve a rendszeresen ismétlődő feldolgozásig mindenféle számítástechnikai szolgáltatással a megrendelők rendelkezésére áll, természetesen pénzbeli ellenszolgáltatásért. Ebből a bevételből fedezi kiadásait és képezi a nyereségét. Az ilyen típusú vállalatoknál a tulajdonképpeni üzem vagy műhely — tehát az a hely, ahol a „termék" készül — maga a gépterem. A vállalat egész tevékenysége ezt készíti elő (programozás) vagy ennek munkáját regisztrálja (könyvelés). ● | 315 | 4

Oktatási intézmény számítóközpontja

Az oktatási intézmények — ha az oktatás színvonala és a tanulók létszáma ezt megköveteli — önálló számítóközponttal rendelkezhetnek. S bár a személyi számítógépek térhódítása értelemszerűen az iskolákban a legnagyobb, hazánkban is található nem egy oktatási intézmény — elsősorban egyetem — ahol nagyszámítógépet üzemeltető önálló számítóközpont van. ● | 315 | 5

Induljunk el — ha csak képzeletben is — és látogassunk meg egy ilyen számítóközpontot. Ismerkedjünk meg az ott folyó munkával, s vizsgálódásaink, nézelődéseink során fedezzük fel újra mindazt, amit a tanév során tanultunk. Látogatásunk legyen egyben alkalom a tanultak ismétlésére is.

LÁTOGATÁS EGY SZÁMÍTÓKÖZPONTBAN

Kísérőnk a számítóközpont vezetője, aki először a feladataikról tájékoztat bennünket.

— „Egyetemi számítóközpont vagyunk, ezért elsődleges feladataink közé tarto-

zik az egyetemi hallgatók számítástechnikai és szakmai tanulmányaihoz a számítógépes háttér biztosítása.

Számítástechnikai képzésben az egész első évfolyam, majdnem 300 hallgató részesül, akik elsősorban programozási technikát és programozási nyelvet tanulnak. Ezeknek a hallgatóknak a kiszolgálása terminálokon keresztül, többfelhasználós időosztásos rendszerben történik."

A számítóközpont vezetője elmondja, hogy a különböző szakmai tárgyak keretében egyre több hallgató veszi igénybe a számítógépet, a legkülönbözőbb feladatok és problémák megoldása érdekében. Ezeknek a hallgatóknak egy része szintén terminálokat használ, más részük a hagyományos, kötegelt feldolgozási módot választja, bizonyára azért, mert pár éve számítástechnikából még csak ezt tanulták.

— Meg tudná valamelyikőtök mondani, hogy mi az alapvető különbség a kötegelt és a többfelhasználós, időosztásos működésmód között?

— Szerintem a következő: a tisztán kötegelt üzemmódnál az a cél, hogy a számítógép átbocsátóképessége minél nagyobb legyen, a másik esetben pedig, hogy a felhasználók kényelmesen és közvetlenül használhassák a számítógépet annak termináljain keresztül.

— Igen, így van. Ebben a számítóközpontban olyan kevert megoldást találunk, amelynél a terminálnál ülő felhasználók „mögött" a háttérben kötegelt feldolgozás is lehetséges. Ilyen esetekben mindig a terminálról érkező munkáknak kell a nagyobb prioritást biztosítani.

— „A hallgatókkal való foglalkozás mellett más feladataink is vannak — folytatja kísérőnk. Így mindenek előtt az oktatásszervezéssel és az oktatásadminisztrációval kapcsolatos rendszerek üzemeltetése, ill. az újabbak szervezése és programozása. Feladatunknak tekintjük továbbá a tanszéki kutatások számítógéppel való támogatását. Talán nem érdemtelen megjegyezni, hogy a személyi számítógépek — elsősorban a professzionálisnak nevezett gépek — megjelenésével sok tanszékünk úgy ítéli meg, hogy ezek beszerzése és tanszéki használata kényelmesebb számukra, mint a mi szolgáltatásaink, amelyeket csak itt a helyszínen tudnak igénybevenni, mivel a terminálok tanszékekre való kihelyezése pillanatnyilag — technikai okokból — nem megoldható."

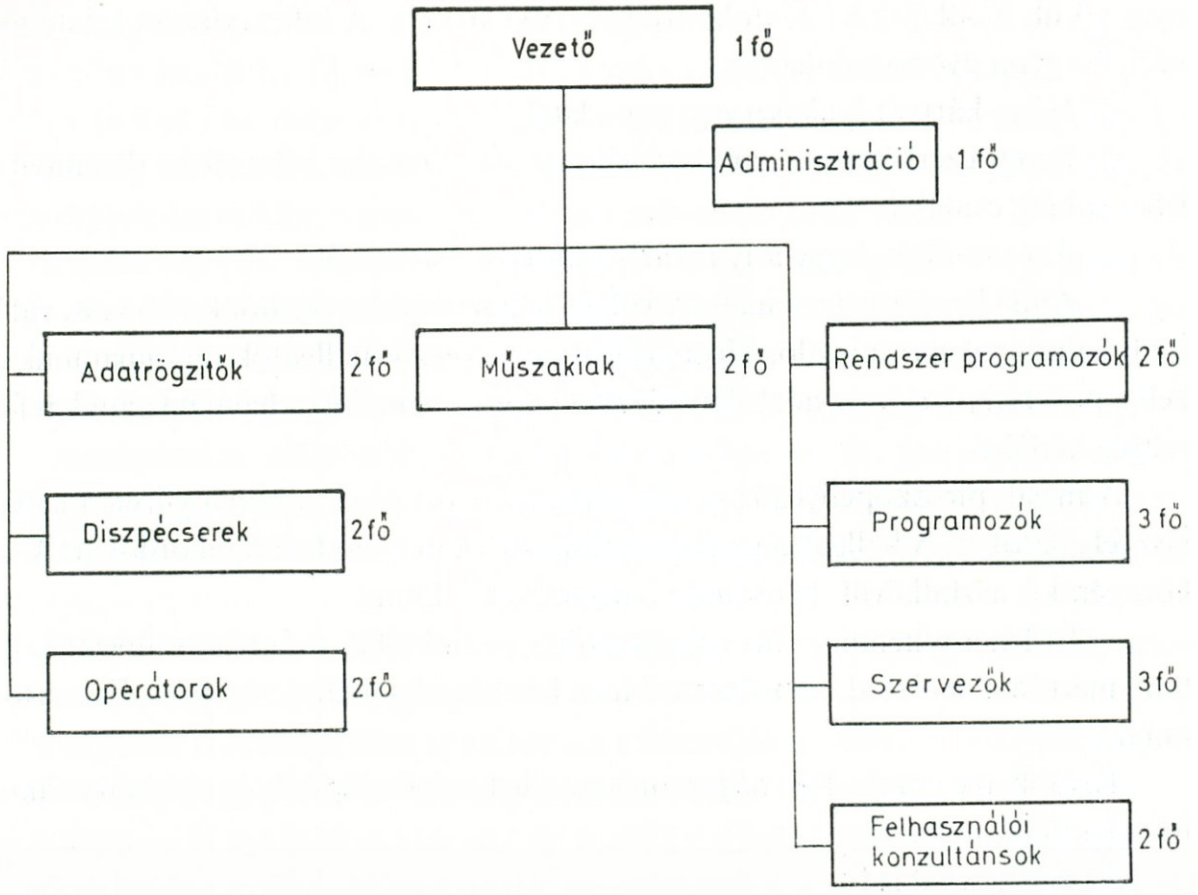
A számítóközpont feladatainak ismeretében tesszük fel a kérdést: Milyen a számítóközpont szervezeti felépítése?

A válaszból megtudjuk, hogy a 21 fős számítóközpont közvetlenül a rektorhelyettes hatáskörébe tartozik. A munkakörök és az ott dolgozók száma a szervezeti felépítésen látható (97. ábra).

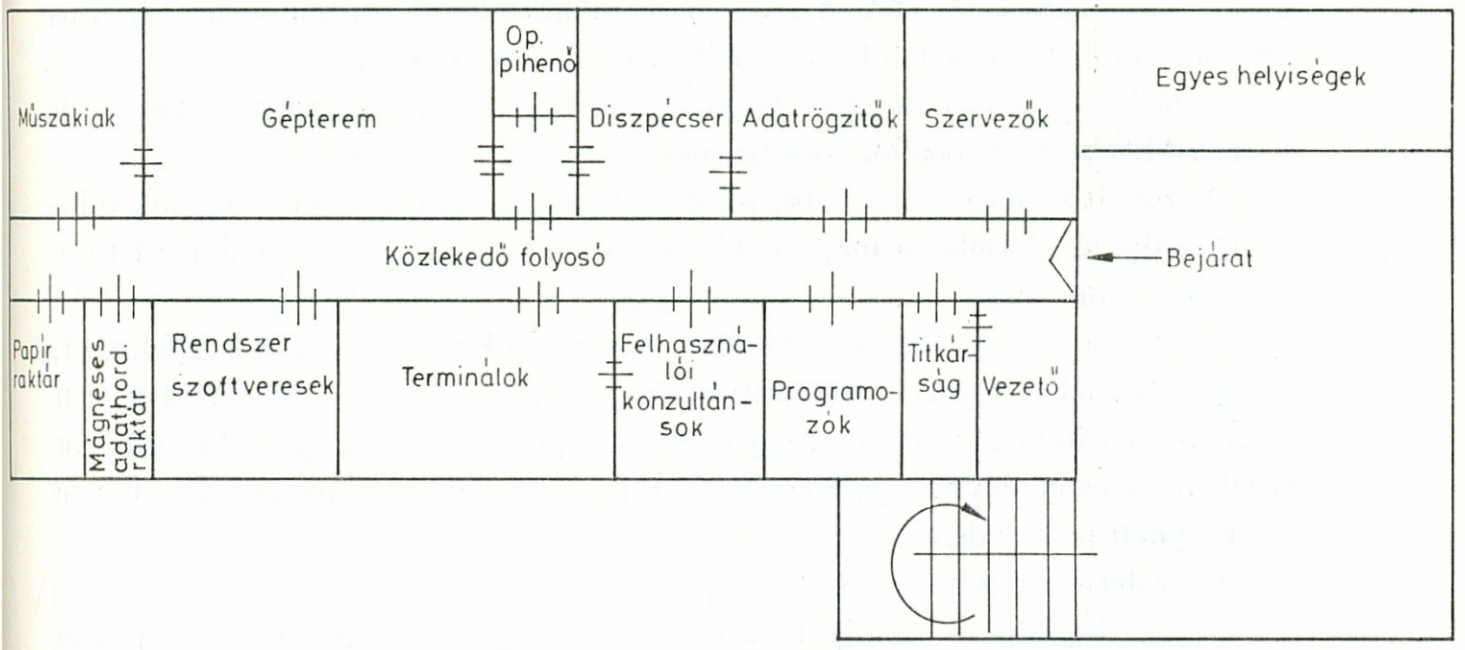
Kis tájékoztató táblát pillantunk meg a bejárat mellett, amelyiken a számítóközpont alaprajza látható, térképként segítve a tájékozódást. Vessünk rá egy pillantást, hiszen mindenhová be akarunk nézni (98. ábra).

Induljunk! — hallatszik vezetőnk hangja, és kulccsal nyitja a bejárati ajtót. Néhány hallgató jön velünk szemben, most fejezték be (vagy csak abbahagyták) munkájukat.

Elsőnek az „Adatrögzítők" feliratú ajtón lépünk be.



97. ábra. Kisebb méretű oktatási számítóközpont szervezeti felépítése



98. ábra. A „meglátogatott" számítóközpont funkcionális alaprajza

(Álljunk meg egy pillanatra a küszöbön, és gondoljuk végig mindazt, amit a másodlagos adathordozókról, az adatrögzítésről tanultunk. Ha felelevenítettétek a tanultakat, nézzünk körül.)

A tágas, világos szobában két lyukkártyalyukasztó gépet és egy négymunkahelyes csoportos adatrögzítő berendezést találunk. Az egyik kártyalyukasztónál pirosköpe-

nyes nő ül. Kódlapra írt adatokat rögzít lyukkártyára. A billentyűzetet szinte nem is nézi. Nagyon gyorsan dolgozik.

— Hány kártyát lyukaszt egy nap alatt?

— Ezret, ezerkétszázat, amikor nagyon sok a munka kétezret is, de annyit nem lehet sokáig csinálni.

— Azt tanultuk, hogy a lyukkártya kimegy a divatból.

— Amíg kevesebb terminálunk volt és mágneses adatrögzítőnk sem volt, valóban több volt a lyukasztani való. Most elsősorban a végzős hallgatók diplomamunkáihoz kell a programokat és az adatokat rögzíteni. De ez nem divat, hanem technikai felszereltség kérdése.

A másik pirosköpenyes lány egy képernyő előtt ül, és a billentyűzeten keresztül visz fel adatokat. A hallgatói nyilvántartáshoz rögzíti a módosító rekordokat. A szoba közepén kis asztalkörül kényelmes karosszékeket látunk.

— Itt lehet pihenni — mondja vezetőnk — tudniillik óránként szünetet kell tartani, mert sem derékkal, sem szemmel nem lehet bírni sokáig a rögzítést. Fárasztó ez a munka.

Tovább megyünk. Pár négyzetméteres helyiségbe lépünk be, ahonnan minden irányba ajtók nyílnak.

Az egyiken tábla:

„Gépterem. Idegeneknek belépni tilos!" Kísérőnk az ezzel szemben levő ajtón nyit be.

— „A diszpécserek — mutat maga elé —, ők készítik elő a számítógépes feldolgozásokat, és tőlük veszik át a felhasználók az eredménytablókat."

(Emlékeztek, a számítógépes eredményközlés egyik fontos formája, a nyomtatott szöveg. Milyen egyéb eredménymegjelenítési lehetőségeket ismertek?)

A szobában világossárga köpenyes fiatalember éppen a gépteremből kihozott listákat szedi szét. Munkáját megkönnyíti, hogy a kezdőlapon nagy betűkkel fel van írva a job azonosítója.

— „Itt vannak a terminálról indított job outputok is" — mondja a fiatalember, és egy köteg félbehajtott leporellóval elindul a folyosóval határos fal felé. Beépített fehér szekrényhez lép. Kinyitja az egyik ajtót és a polcra beteszi a leporellókat. Ekkor látjuk, hogy a szekrény hátlapja hiányzik. A folyosóról nézve ez nem is szekrény, hanem egy beépített polcrendszer.

Visszatérünk a szűk előszobába.

— Ez a légzsilip — mondja kísérőnk —, azért van rá szükség, mert a gépterem klimatizált és pormentesített, s ezért nem nyílhat közvetlenül a folyosóra. A gépterembe nyíló ajtó egyben tűzszakasz határoló is. Itt pedig — mutat az egyik ajtóra — az operátori pihenő található, de most éppen üres. ▲ 314 6

Belépünk a gépterembe. Hűvös, száraz levegő csap meg. Kellemes világítás, tompa, de nem bántó zaj. „Ez lenne tehát az a hely, ahová egyszerű halandó nem juthat be soha?"

A gépterem rendjét írásban szabályozták, és ez jól látható helyen van elhelyezve a

falon. Itt találjuk a tilalmakat és a kötelezettségeket felsorolva (pl.: tilos a dohányzás, a gépteremben legalább 2 főnek kell tartózkodnia — áramütés miatt — stb).

Amíg becsukódik mögöttünk a gépterem ajtaja, rendszerezzétek emlékezetetekben — szükség esetén az V. és a VII. fejezet átismétlésével —, hogy a számítógép hardverje miképpen tagozódik, s melyik egységnek milyen funkciói vannak.

Emelkedő rámpán teszünk néhány métert. A gépterem padlószintje legalább negyven centivel magasabban van, mint a légzsilipé, ahonnan beléptünk. „Ez az álpadló, ami alatt vannak a különböző egységek számára áramot biztosító, valamint az adatátvitelt szolgáló huzalok és kábelek, s ide fújja be a klímagép a friss levegőt."

A mennyezeten süllyesztett lámpák, levegőelszívók és füstjelzők vannak. ▲ 314 7 A gépteremben éppen nagy a nyüzsgés. A másik diszpécser egy kis szállítókocsit gördít az operátorokhoz. A kocsin mágneslemezek, két mágnesszalag, kevés lyukkártya.

— Délután esedékes a havi feldolgozások egyike — magyarázza kísérőnk, amikor észreveszi, hogy a kocsit figyeljük.

A diszpécser most adja át az operátornak a futtatáshoz szükséges mágneses adathordozókat és a feldolgozáshoz szükséges munkavezérlő kártyákat. A programok — természetesen — könyvtárazva vannak. Az operátor a futtatási lapot tanulmányozza, és ellenőrzi, hogy a szalag- és lemeznevek megegyeznek-e a lapon levőkkel. A másik operátor két barnaköpenyes férfival beszélget.

— Főnök, a hármas lemezegység megint nyavalyog! — szól az egyik barnaköpenyes, a hardver mérnök. Az egyik lemezmeghajtóról leszerelték a borítólemezeket, a teteje is nyitva van, szemlátomást „nyavalyog".

Később megtudtuk, hogy az író-olvasó fejekkel van baj. Hiába tesznek fel lemezt és indítják el, a lemez felgyorsul, de az író-olvasó fejek nem akarnak megmozdulni. Műszerrel kell bemérni, hogy melyik csatolókártyán van a hiba.

Hirtelen erős zaj hallatszik. Az egyik sornyomtató meglehetősen nagy zajjal nyomtatni kezdett. A hardveresekkel beszélgető operátor odalép a nyomtatóhoz, ellenőrzi, hogy a papír jól van-e beállítva, majd figyeli a kijövő leporelló lapokat.

A négy mágnesszalag-egység közül kettő van bekapcsolva, az egyiken a szalag szép tempósan halad előre. Időnként lelassul, majd ismét felgyorsul. Jó nagy lehet a blokkméret — gondoljuk magunkban, visszaemlékezve a tanultakra.

(Elevenítsétek fel a mágnesszalagon való adattárolásról tanultakat. Mi módon állapítható meg egy működő mágnesszalag-egységet nézve, hogy kicsi vagy nagyméretű blokkok vannak a szalagon?)

— Terminál is van a gépteremben? — mutat rá egyikünk egy képernyős berendezésre.

— Ez a konzol, amely formájában valóban hasonlít a terminálokhoz, de más a funkciója.

— Ki tudja megmondani, mi a konzol funkciója, miért van rá szükség?

— A konzol biztosítja a kapcsolatot a számítógépet irányító operációs rendszer és a számítógépet kezelő ember között. Az operációs rendszer a konzolra küldi az

üzeneteit, az operátor a konzolon keresztül küldhet parancsokat az operációs rendszernek vagy innen üzenhet a terminálóknál ülő felhasználóknak.

A sornyomtató elhallgat. Az operátor odaül a konzol elé és a billentyűzeten keresztül valamit közöl a géppel. Az egyik job korábban olyan erőforrást kért, amelyet a most befejeződött program kötött le. Az operátor a várakozásból oldja fel a jobot.

— Ez itt a központi egység, ugye? — mutat a konzolt fel nem ismerő fiú a szemben levő, jelzőlámpácskáit villogtató fémszekrényre.

— Igen, ez a központi egység, és itt van az 1 Mbájtos központi tár is.

Meg tudná mondani valaki, hogy a központi tár milyen hardver elemekből épül fel?

— Félvezetős tárnak kell lennie, mert ha ferrites volna, elfoglalná a fél géptermet.

— Valóban félvezetős, az 1 Mbájt annak a fémszekrénynek a felső részében található.

Az egyik lány, akinek az apja tűzoltó, az oltókészülékek iránt érdeklődik. Édesapjától hallotta, hogy néhány éve egy számítóközpontban tűz ütött ki, és az oltáshoz porral oltót használtak. A por több kárt tett a gépben, mint a tűz.

— A számítóközpontok tűzvédelme nem csak azért fontos, mert több millió forintos értéket képviselnek ezek a gépek, hanem azért is, mert a számítógépben őrzött adatok, információk is nagy értéket jelenthetnek. A számítóközpontokban speciális gázzal, halonnal töltött tűzoltókészülékek találhatók. Nézzétek, ott a bejárat mellett és ott a gépterem közepén a falon vannak elhelyezve az oltókészülékek. Az operátorok megfelelő kiképzésben részesültek. A legfontosabb persze itt is a megelőzés. | ● | 315 | 8 |

Indulunk kifelé.

Elbúcsúzunk az operátoroktól, a hardveresek időközben a másik ajtón kimentek a saját birodalmukba a méréshez szükséges műszerekért.

— Szóljatok Józsinak, hogy hozzon be a raktárból délutánra legalább négy doboz papírt — szól oda vezetőnk az egyik operátornak.

Kifelé jövet észreveszünk egy nagy üvegablakot a gépterem falán. Mögötte az operátori pihenő, ugyanolyan kényelmes karosszékekkel, mint az adatrögzítőknél. | ● | 316 | 9 |

— Mi csak akváriumnak nevezzük — nevet kísérőnk, az operátorok meg dühösek, mert szerintük a gépterem az akvárium.

A folyosó másik oldalán a „Rendszerprogramozók" feliratú szobába kopogtatunk be. A szoba közepén egymással szemben két íróasztal, a falnál fémlábú asztalok, két terminállal és egy matrixnyomtatóval. Az egyik rendszerprogramozó a terminálnál ül, a másik egy listából — természetesen számítógépes listából — diktál. Rövid odafigyeléssel rájövünk, hogy éppen könyvtárak karbantartása történik. Most állítják össze a képernyőn azt a jobot, amelyben az IEBCOPY segédprogrammal tömörítik a SZTLIB lemezen levő könyvtárakat.

(Amíg a job végrehajtódik, ismételjétek át a könyvtári állományról tanultakat, és mondjátok meg, miért kell a számítógépben levő könyvtárakat időről időre karbantartani?)

— Mi van a hármas lemezegységgel? — kérdezik kísérőnket.

— Már javítják! — válaszolja, majd hozzáteszi — a heti gépidő-elszámolást tegyétek az asztalomra, reggel át akarom nézni.

A programozók ajtaját zárva találjuk. Az ajtó nyílásába dugott lyukkártya tájékoztat, hogy a szervezőknél értekeznek. Mi is benyitunk a szervezők szobájába.

A munkaértekezlet utolsó percei zajlanak. A témavezető éppen összefoglalja a megbeszélés eredményét:

— „A törzsfájl tehát indexelt szekvenciális szervezettségű lesz, hogy ezeket a két időszak közötti speciális igényeket is ki tudjuk elégíteni. Az adatellenőrző és -felvivő programokat ti csináljátok. A háromféle feldolgozóprogram pedig úgy készüljön, ahogy megbeszéltük, a négy közös alprogramot ne írjátok meg külön-külön. Találkozunk jövő kedden, addigra legyen kész az első fázishoz tartozó valamennyi program. Jó munkát, szervusztok."

(A témavezető összefoglaló mondatai jó alkalmat adnak nekünk is, hogy átismételjük a fájlszerkezetekről tanultakat, összefoglaljunk mindent, amit az adatellenőrzésről tanultunk. Tegyétek meg!)

A szoba kiürül, mindenki megy a dolgára.

A témavezető szervezővel még beszélgetünk munkamódszerükről. Elmondja, hogy a kisebb feladatokat általában egy ember önállóan oldja meg, a nagyobb és bonyolultabb feladatokra meg alkalmi csoportokat hoznak létre.

— Nálunk a szervezők és a programozók egyaránt tudnak programozni is meg szervezni is, az elnevezésük csak a besorolásukat jelenti.

— Kevesen vagyunk, sietni kell, hogy minél hamarabb kész legyünk, mert különben körmünkre égnek a feladatok. A felhasználói konzultánsokra csak ritkán lehet számítani, mert őket a hallgatók állandóan ellepik mindenféle problémával.

Nézzük meg a terminálokat is — invitál vezetőnk és indul előre, hogy mutassa az utat. A gépteremmel szemben a folyosó másik oldalán található a kiírás: „Terminálok". Gépterem nagyságú helyiségbe nyitunk be. A teremben egymás mellett ötösével terminálok sorakoznak. Húsz egyforma terminál s hátrább a terem túlsó végében más típusú berendezések. Majd mindegyiknél ül valaki.

— Húsz terminál mellett tíz professzionális személyi számítógép is a hallgatók rendelkezésére áll. A számítástechnikai képzést biztosító szoftvert mind a nagygépre, mind a mikrogépekre megcsináltuk. Így az mind a terminálokról, mind a mikrogépeknél igénybe vehető. Egyelőre mátrixnyomtatónk nincs elegendő, ezért a mikrogépekről való listázás problematikus. Húsznál több terminál nagyon lelassítja a rendszert, így is sokszor kell várakozni, ha a munkák torlódnak.

Megállunk az egyik hallgató háta mögött. Éppen programot ír, vagy ahogy itt mondják, editál. A készülő program utasításait egymás után írja fel a képernyőre. Ezzel egyidejűleg a bevitt sorok rögzítődnek egy mágneslemez munkaterületen. Majd, ha kész lesz, az egészet egyszerre bemásolhatja a saját könyvtárába.

Tovább lépünk, egy másik hallgató a már elkészült programjába javít. Kijavítja egy rosszul begépelt változó nevét, majd két új utasítást illeszt be a programba.

— Talán most jó lesz! — sóhajt és az egész programot visszaíratja a könyvtárába.

(Írjátok le, hogy mindaz, amit a tanuló a terminálnál megcsinált, hogyan nézne ki, ha ezt a programot lyukkártyáról, kötegelt üzemmódban kellene tesztelni.)

A harmadik képernyőn figyelemmel kísérhetjük egy program futását.

Velünk együtt figyeli az egyik felhasználói konzultáns, akinek a feladatkörébe tartozik, hogy időnként végigmenjen a terminálok között, és segítsen a rászorulóknak.

A „szerző" ügyes megoldást alkalmazott, mert mielőtt a program adatot kérne a terminál billentyűzetéről, megjelenik a képernyőn az adatkérő szöveg.

— Az interaktív környezetben futó programok szerkezetét másképpen kell felépíteni, mint a kötegelt futásra szánt programokét — magyarázza csendben a szoftveres, hogy közben ne zavarja a diákokat. Ha pl. ezek az adatkérő kiírások nem jelennének meg, a gép az adatra várna, a tanuló meg a gépre, s ha a tanuló nem kap időben észhez, akár órákig is elüldögélhet itt.

Időközben a képernyőn megjelenik egy táblázat: a feldolgozás eredménye. A hallgató az előre kiszámított eredményekkel veti össze a képernyőn levőt. Az arcán látszik, hogy a tesztelés eredménye jó.

Haladunk tovább.

— Most olyasmit mutatok, amit még nem sok helyen lehet látni — szólal meg ismét kísérőnk. Ez a tanuló programjának a kipróbálását egy különleges szoftver segítségével végzi. Ez egy nyomkövető és hibakereső program, amely lehetővé teszi, hogy futás közben a programot előre meghatározott pontokon megállítsuk, és a különböző változók értékét lekérdezzük. Hallatlan nagy segítséget nyújt, elsősorban nagyobb méretű, sok alprogramból épülő programok esetében. Olyan hibákra lehet viszonylag hamar rájönni, amilyeneket hagyományos módszerekkel az ember napokig keres. Arra is van lehetőség, hogy bármely változó értékét futás közben megváltoztassuk, felülírjuk.

— Miért mondta, hogy nem sok helyen látható ilyen szoftver?

— A hibakeresésnek ez a programmal való támogatása az ESZR gépeknél újabb keletű. Ma elsősorban olyan gépeknél található meg, amelyekhez Pascal fordítóprogram is van. Érdekes módon ezek elsősorban professzionális személyi számítógépek, bár egyre több nagygépre is telepítenek már Pascalt. A mi programunk érdekessége, hogy nem Pascalban írott programok esetében is használható. Kollégáim készítették, igen jól szolgálja az oktatást.

Az egyik terminálnál valaki éppen befejezi a munkáját. Elköszön a géptől, a gép nyugtázza a kilépést, kiírja az elhasznált központi egység időt, és a kapcsolati időt.

— Tanár úr — szólal meg egyikőtök —, mi már tisztában vagyunk a számítástechnika alapjaival, leülhetnénk egy kicsit dolgozni ezzel a géppel?

— Elvi akadálya nincs, de ehhez az szükséges, hogy legyen felhasználói azonosítód meg jelszód, meg legyenek könyvtáraid a jelenlegi operációs rendszerben. Ha komolyan gondolod, a rendszerprogramozók holnapra biztosítják számodra mindezt.

Addig is olvasd el ezt a kis füzetet, ebből kiderülnek a terminál kezelésével kapcsolatos legfontosabb technikai tennivalók.

Elbúcsúzunk. Siessetek haza, hogy készülhessetek a holnapra és a Holnapra.

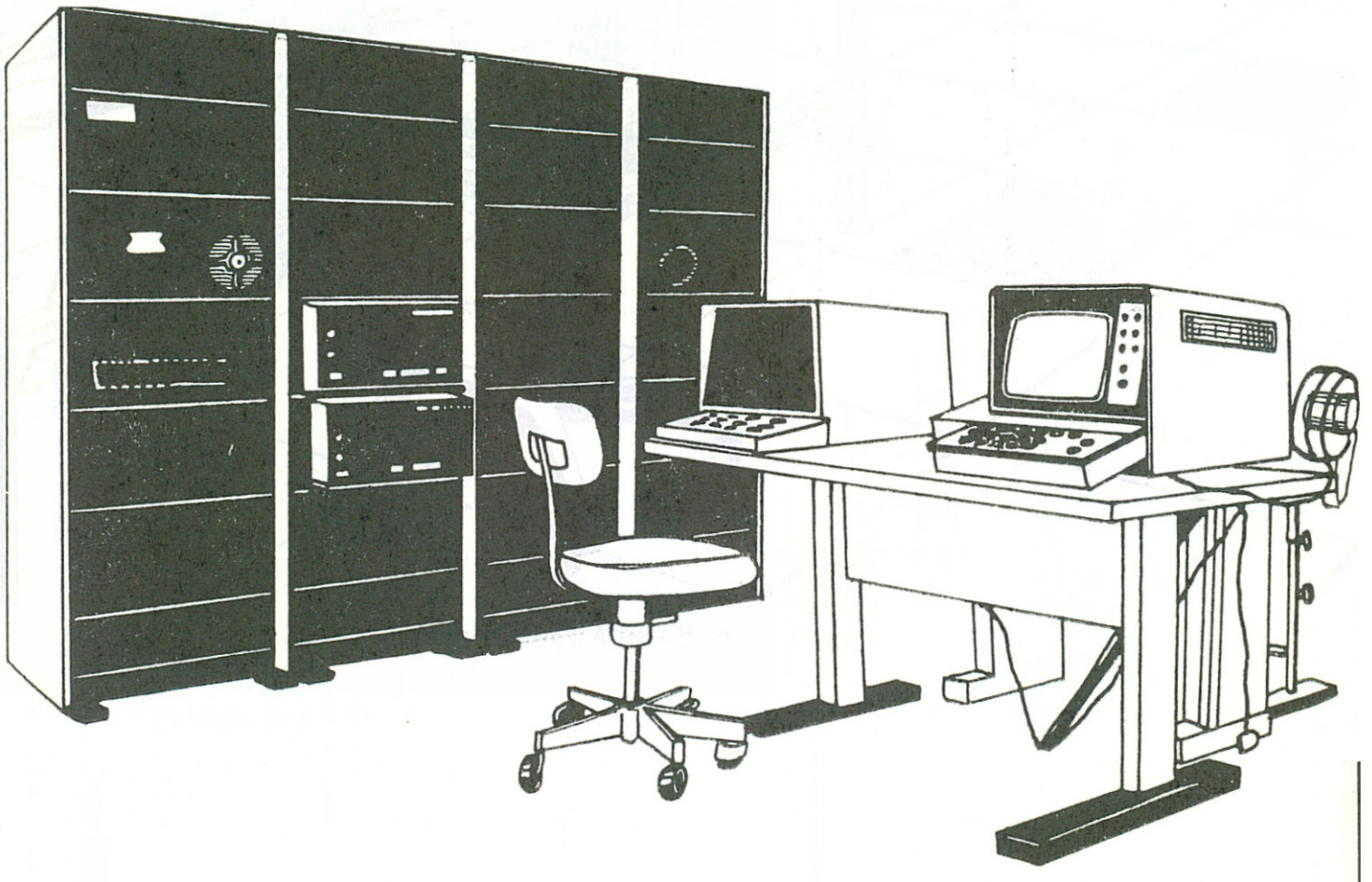
Kérdések

1. Mi a gépterem?
2. Mi a számítóközpont?
3. Milyen céllal üzemelhet a számítóközpont?
4. Mi a géptermi álpadló és álmennyezet funkciója?
5. Miért van szükség operátori pihenőre?

Feladatok

1. Ha van rá lehetőség, szervezzetek kirándulást valamelyik környéken levő számítóközpontba!

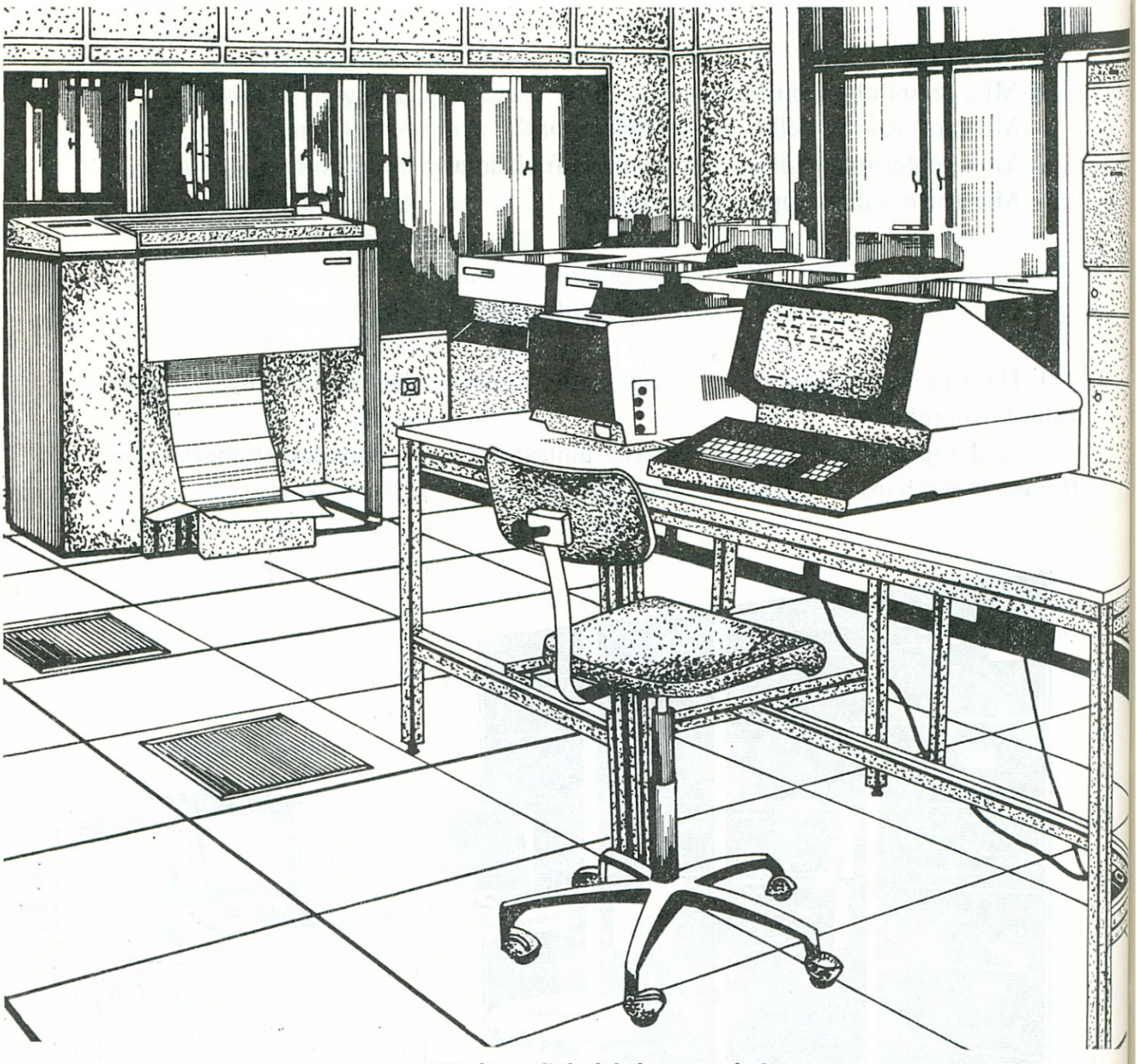
2. A „látogatás" ismételt elolvasásával állítsátok össze a képzeletben meglátogatott számítógép hardver kiépítettségének listáját!



99. ábra. SzM—4 miniszámítógép

3. Válaszoljátok meg az összes kérdést, amely a fejezetben felmerült, de nem került megválaszolásra!

4. Ismételjétek át a tananyagnak azokat a részeit is, amelyeket a „látogatás" során nem érintettünk!



100. ábra. Számítógépterem részlet

ANNOTÁCIÓS RÉSZ

I. fejezet

■ 5

Neumann János (1903—1957) magyar származású fizikus, matematikus nevével a II. osztályos Technika c. tárgy keretében találkoztatok. Jelentős szerepet játszott az első számítógépek megépítésében. Két ötlete — a kettes számrendszerben való műveletvégzés és a számítógép „belső vezérlése", vagyis hogy a gép a saját működését vezérelje — ma is használt elgondolások.

Ez utóbbit szokás úgy megfogalmazni, hogy a számítógép belső programozású vagy tárolt programú berendezés.

■ 10

A II. osztályos Technika c. könyv 100. ábráján (147. oldal) jól nyomon követhető a mikroelektronika fejlődése.

A kis mértékben integrált áramkörök (SSI) 1969-ben jelentek meg, ezt követte a közepes integráltság (MSI), majd a nagymértékben integrált áramkörök (LSI) és végül a nyolcvanas évek elején a VLSI.

★ 1

A számítógéppel a II. osztályos Technika c. tárgy keretében találkoztatok. A tankönyv a 7. fejezetben így írta le a számítógép fogalmát: „A modern számítógép olyan összetett rendszer, amelyben gépi berendezések és programok egymástól elválaszthatatlan együttest alkotnak..."

Ebben a tankönyvben a számítógépnek ugyanezt a berendezést tekintjük, a későbbiekben azonban egy pontosabb fogalommeghatározással találkoztok.

★ 2

Az információ fogalmával a második fejezetben találkozunk. Egyelőre az információfeldolgozást tekintsük egy olyan tevékenységnek, amelynek során a számítógéppel műveleteket, számításokat végeznek és az eredményeket valamilyen célra hasznosítják.

★ 12

A mindennapi életben, és sokszor a számítástechnikában is *adatfeldolgozásnak* nevezik azokat a munkákat, amelyeknél nem bonyolult matematikai apparátussal nagytömegű adat feldolgozása történik. Ilyen munka pl. egy népszámlálás kérdőíveinek különböző szempontok szerinti összesítése. Összeadáson és osztáson kívül talán más matematikai műveletet nem is igényel egy ilyen munka, de ezt több tízmillió adattal kell elvégezni. (Ha pl. 200 ember napi nyolc órán keresztül, percenként 30 kérdést értékelne ki, legalább hat évre lenne szükség csak a legfontosabb feldolgozási eredmények előállításához. Ráadásul ezek az eredmények — figyelembe véve az ember hibázási gyakoriságát — nem is lennének hibátlanok. Ugyanezeknek a kérdőíveknek a kiértékelése — megfelelő előkészítés után — a számítógép számára néhány órai munkát jelent. Az időeltérés az eredmények felhasználhatóságában jelent minőségi különbséget, hiszen pl. kit érdekel, hogy hat évvel korábban hány iskolaköteles gyermek volt.)

Az ilyen jellegű munkákkal szemben beszélnek az ún. számításigényes feladatokról, ahol viszonylag kevés adattal kell bonyolult vagy hosszadalmas, sokszor ismétlődő műveletsort végezni.

Egy rakéta esetében a szükséges pályakorrekció kiszámításához nem kell sok bemenő adat, de bonyolult számításokat olyan gyorsan kell végrehajtani, hogy azok még használhatók legyenek a pályakorrekció végrehajtásához, hiszen a rakéta már néhány perc múlva is teljesen más jellemzőkkel rendelkezik.

A látszólagos ellentmondás az „adatfeldolgozás" kifejezés értelmezésében van.

Ennek a kettősségnek a feloldása érdekében vannak szerzők (mint pl. a II. osztályos Technika c. könyv írója is), akik inkább az „információ feldolgozás" kifejezést használják.

Ebben a tankönyvben az „adatfeldolgozás" számítógépes feldolgozást jelent, az információ és az adat fogalmának tisztázásakor kiderül, hogy miért.

▲ 7

A matematikában **operandusnak** nevezik a műveletekben résztvevő értékeket (pl. az összeadás tagjait, a szorzás tényezőit stb.). A számítástechnikában minden olyan értéket operandusnak nevezünk, amelyet a gép működése során bármilyen tárolóhelyről elő kell venni, vagy ott rendelkezésre áll a művelet elvégzéséhez.

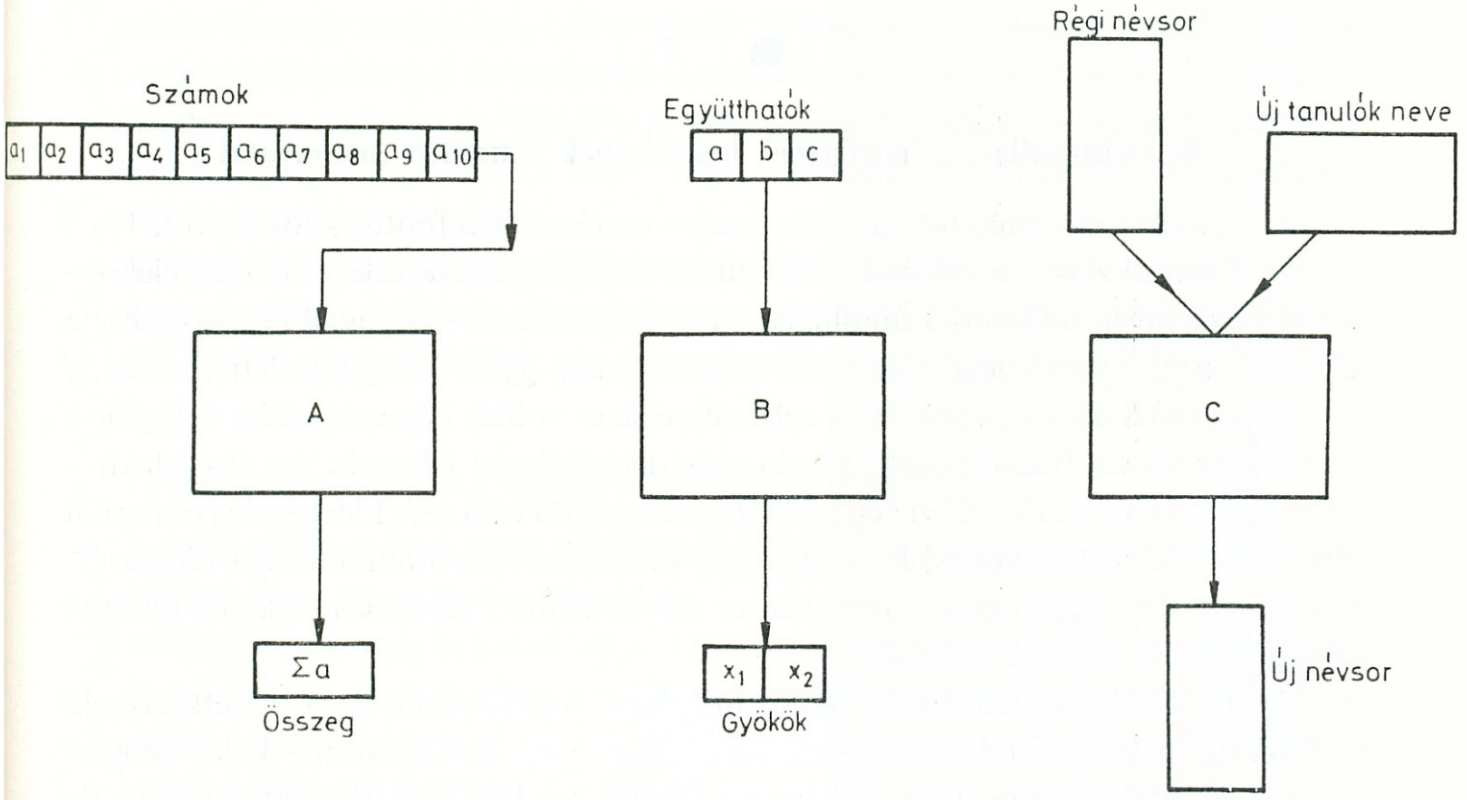
▲ 11

Válasszunk ki találomra három feladatot, amelyek látszólag nincsenek kapcsolatban egymással.

A. feladat: Ki kell számítani tíz megadott szám összegét.

B. feladat: Az $x^2+x-6=0$ másodfokú egyenlet valós gyökeit kell kiszámítani.

C. feladat: Négy tanuló nevét be kell illeszteni az osztály abc szerinti névsorába.



A-1. ábra. Bevitel—Feldolgozás—Eredményközlés

A három sematikus ábra a három feladatban rejlő közös vonásra hívja fel a figyelmünket:

— a feladatok megoldásához bizonyos induló értékek (számok, nevek; összefoglaló névvel adatok) állnak rendelkezésre, ezeket be kell juttatni a számítógépbe;

— a kezdeti adatokkal a számítógép feldolgozást végez (műveletet végez velük, új sorrendbe helyezi el őket stb.);

— a feldolgozás végén vagy esetleg közben is az eredményeket, vagy más kifejezéssel eredményadatokat az ember (felhasználó) által kezelhető alakban megjeleníti.

● 3

A számítógép-tudománnyal kapcsolatban meg kell említenünk a *kibernetikát*. Norbert Wiener századunk negyvenes éveinek a végén publikálta „Cybernetica" c. könyvét, amelyben „az élő szervezetben és a gépben történő irányítás és kommunikáció elméletének" tekintette a kibernetikát. Arról van tehát szó, hogy komplex tudományos igénnyel próbálta elemezni a bonyolult rendszerek működési mechanizmusát, keresve azokat az ismérveket, amelyek e rendszerek konkrét anyagi megvalósulási formáitól függetlenül fellelhetők.

A kibernetikai gondolkodásmód nagy hatással volt a számítógép tudomány különböző területeire is.

Maga a „kibernetika" szó egyébként görög eredetű, eredeti értelme „a kormányzás művészete" vagy a „kalauzolás művészete".

Wiener nyilvánvalóan átvitt értelemben alkalmazta a már említett művében.

● 4

Az elektronikus számítógépek fejlődésének fontosabb mozzanatai

A számolás gépesítésének sok évszázados történetében fontos állomás volt 1941, amikor Konrad Zuse vezetésével elkészült az első programvezérlésű de még elektromechanikus elven működő számológép. A kutatásokat a II. világháború gyorsította fel, amikor is nagytömegű hadászati célú számítás gyors elvégzése lett a feladat. Az elektromechanikus gépek ilyen sebességre nem voltak képesek, ezért a figyelem az elektroncsövek (rádiócsövek) alkalmazási lehetőségére irányult. Az első elektronikus (rádiócsöves) működésű gép az 1946-ra elkészülő ENIAC (Electronic Numerical Integrator and Calculator) volt, de ez a gép külső vezérléssel működött. Érdekességképpen: 18 000 elektroncső, több tízezer kondenzátor, ellenállás, tekercs alkotta, energiafelvétele 150 kW óránként.

Az ENIAC híre felkeltette Neumann János érdeklődését, és a következő gép az EDVAC már az ő közreműködésével született meg. Ez a gép már belső programozású és elektronikus volt. (Neve az Electronic Discrete Variable Automatic Computer elnevezés rövidítéséből ered.)

Ebben a korai szakaszban a számítógépek egyedi darabként készültek és általában maguk az építők voltak a későbbi használók is. Programozásuk gépi kódban volt csak lehetséges.

1951-ben kezdődik meg a számítógépek sorozatgyártása, az első sorozatban gyártott gép az UNIVAC 1. Ennek a gépnek már mágnesszalagos perifériái is voltak, mintegy 1000 szám tárolására volt képes.

Az ötvenes évek elején elkészült az első szovjet számítógép a BESZM, amely 7000...8000 műveletet végzett másodpercenként. Még az ötvenes évek elején meg-

született a tranzisztor és néhány év múlva (az évtized közepén) az első tranzisztorbázisú számítógép. 1958—59-ben elkészül Magyarországon az M3, amelyet szovjet dokumentációk alapján építettek. A hatvanas évek elején megjelennek az első magasszintű programozási nyelvek, az ALGOL, a FORTRAN és a COBOL.

Az IBM cég — a világ egyik legnagyobb számítógépgyártó cége — ekkortájt jelenti be a 360-as gépcsaládot az első harmadik generációs rendszert. (Az integrált áramkörök szilícium lapja ekkor mintegy öt áramköri elemet — tranzisztort, kondenzátort, ellenállást — tudott megvalósítani.)

Az IBM 360-as rendszerrel megjelentek az első korszerűnek számító operációs rendszerek és az első olyan magasszintű programozási nyelv (a PL/1), amelyet már az operációs rendszerhez fejlesztettek. A hatvanas évek közepére — végére megjelentek a számítógép hálózatok: több számítógép összekapcsolása. Kialakulnak a távadatfeldolgozási rendszerek. A hatvanas évek végén a szocialista országok számítástechnikai fejlesztéseik koordinálása érdekében létrehozzák az Egységes Számítógép Rendszert (ESZR).

A hetvenes évek az integrált áramköri elemek fejlődése jegyében kezdődtek. Megjelennek a nagy integráltságú áramkörök. 1971-ben elkészül az első mikroprocesszor, a számítógép központi egységének miniatürizált változata. A hetvenes évek elején jelennek meg a piacon az IBM 370-es gépcsaládjának tagjai. 1978 a nagysebességű és nagykapacitású miniatürizált tárak megszületésének éve. Megjelennek a lézerrel működő nyomtatóberendezések.

A nyolcvanas évek kétségtelenül a mikroszámítógépeké.

Az integráltság fokának jellemzésére egy adat 1983-ból: 260 ezer jel tárolását és $120 \cdot 10^{-9}$ s alatt történő elérését egy 68 mm² területű szilícium lapocskán megvalósították.

Az Egységes Számítógép Rendszer

A szocialista országok 1969 óta közösen elfogadott program alapján összehangolt számítástechnikai fejlesztéseket végeznek.

Az ESZR számítógépcsalád több, különböző nagyságrendű, de egységes technológia szerint épített és egységes szoftverrel működő számítógéprendszer. A gyártásszakosítás eredményeképpen hazánk elsősorban perifériákat állít elő (VIDEOTON Elektronikai Vállalat, ORION stb).

Az ESZR berendezések mellett a szocialista országok kisebb számítógépek összehangolt gyártásával is foglalkoznak. Ezeket összefoglalóan MSZR (Mini Számítógép Rendszer) névvel illetik. Ebbe a kategóriába tartozó gépet gyárt Magyarországon pl. a VIDEOTON és a KFKI (Központi Fizikai Kutató Intézet) is.

● 9

A fogalmakat angol kifejezésekkel jelöljük. A „hardware" magyarul „kemény árut", a számítógép esetében a technikai bázist; a „software" pedig „puha árut", papíron megjelenő szellemi terméket jelent. A magyar szakirodalom éppen ezért vette át az eredeti neveket, mert a magyar fordítás csak áttételesen és pontatlanul fedi a fogalmak lényegét. Az angolból átvett neveket fonetikusan írjuk.

Egy számítógép teljesítőképességét a hardver és a szoftver együttesen határozza meg.

● 13

Számítógép az áruházban
(Olvasmány)

Melyik háziasszony nem bosszankodott még egy-egy bevásárlás után, amikor észrevette, hogy megint néhány forinttal többet számoltak a pénztárnál. És melyik pénztáros nem bosszankodott még, amikor a vevő kosarában levő italos üvegen vagy húskonzerven nem volt ár, és a régire hiába emlékezett, mert épp a minap változott meg.

Segíthet-e itt a technika? A számítástechnika?

Magyarország 1984. január 1-e óta tagja az európai vonalkód rendszernek. Ez azt jelenti, hogy néhány éven belül a hazai üzletekben is megjelennek azok a termékek, amelyeknek csomagolásán egy vastagabb-vékonyabb vonalakból álló csíkozás található. Ahhoz hasonló, mint amit egy külföldi csokoládé borítójáról az A-3. ábrán bemutatunk.

A-3. ábra. Vonalkódos címazonosító

A vevő az önkiszolgáló üzletben a polcokról összeszedi mindazt, amire szüksége van. A pénztárnál a pénztáros az egyes csomagokat elhúzza egy fényforrás előtt (esetleg egy fényt kibocsátó „ceruzát" húz el a csomag vonalkódja előtt). A pénztárgépnek „álcázott" mikroszámítógép azonnal kiírja egy listára az áru nevét és árát. Az utolsó tétel után pedig a fizetendő összeget. A pénztárosnak csak a pénz átvételére kell figyelnie. Mi történt? A fénynyalábos érzékelés alapján a mikroszámítógép „felismerte" a vonalkódhoz tartozó árut és memóriájából kikereste annak nevét és árát.

Mindkettőt kinyomtatta a vevő blokkjára. Tévedés nem lehetséges, mert a mikrogépben mindig az érvényes árak vannak. De más is történt, amit a vevő nem láthat, de ami nélkül az egész túl drága mulatság lenne.

A mikroszámítógéppel felszerelt pénztárgépek egy központi számítógéppel is kapcsolatban vannak. Ehhez továbbítják az adatokat, hogy az egyes áruféleségekből mennyi fogyott az egyes vásárlások során. Az így egybegyűlt adatokból azután a központi számítógép — akár percre pontos — de gyakorlatilag naprakészen meg tudja határozni

— a napi forgalom összegét áruféleségenként;

— az egyes áruféleségekből raktáron, ill. az eladótérben maradt mennyiségeket;

— megadja azoknak az áruknak a nevét, amelyekből az eladóteret fel kell tölteni, és természetesen azt is megadja, hogy miből mennyit kell átszállítani, hogy a polcok megint tele legyenek;

— megadja azoknak az áruknak a nevét, amelyekből rendelni kell, mert a raktáron levő mennyiség csak addig elég, amíg az új szállítmány megérkezik. (Természetesen itt is megadja a rendelendő mennyiségeket.) Az is természetes, hogy külön listán közli a számítógép a különböző helyekről szükséges beszerzéseket.

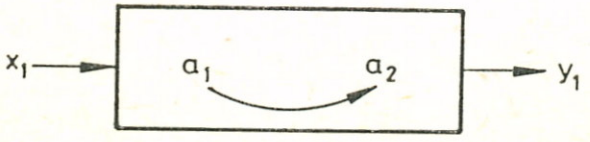
Az áruház irodai munkája leegyszerűsödött, a nyilvántartások pontosabbá váltak, jó szállító partnerek esetében nem lép fel áruhiány, a pénztárnál gyorsabban halad a sor. Mindez más minőségű munkát eredményez.

⊙ 6

A matematika külön területe (az algebrán belül) az *automataelmélet*. Az automatára adott verbális (szöveges) meghatározásnál egzaktabb definíciót az automataelméletből vehetünk át.

Ennek értelmében az automata egy olyan rendszer, amely a külvilágból egy olvasón keresztül jeleket észlel. (A bemenőjelek halmazát jelöljük X szimbólummal.) Az automata egy jel hatására saját állapotát megváltoztatja (az állapotok halmazát jelölje az A szimbólum), és egy írófejen keresztül valamilyen kimenőjelet ad (a kimenőjelek halmazát jelöljük Y szimbólummal).

$$A = \langle a_1, a_2, a_3, \ldots, a_k \rangle$$
$$X = \langle x_1, x_2, x_3, \ldots, x_k \rangle$$
$$Y = \langle y_1, y_2, y_3, \ldots, y_k \rangle$$



A-2. ábra. Automata állapotváltozása

Az A-2. ábra szerinti állapotban az automata az x_1 jelet érzékelve a_1 állapotból a_2 állapotba kerül és az y_1 kimenőjelet adja. Ha ez az automata pl. az $x_1x_2x_3 \ldots x_k$ bemenőjelsorozatot kapja, hatására a következő folyamat zajlik le:

$$\begin{array}{ccccccccc} & x_1 & & x_2 & & x_3 & & x_4 \ldots\ldots x_k & \\ a_1 & \rightarrow & a_2 & \rightarrow & a_3 & \rightarrow & a_4 & \rightarrow \ldots \rightarrow & a_{k+1} \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ & y_1 & & y_2 & & y_3 & & y_k & \end{array}$$

Más bemenőjelre persze másképpen viselkedik.

Az ilyen rendszer azután algebrai formulákkal pontosan leírható.

A számítógép ennél bonyolultabban, de alapvetően ezen az elven működik.

Ma már talán tudománytörténeti érdekességnek hat, hogy a „számítógép" elnevezés megszületését a magyar szakirodalomban milyen hosszú vajúdás előzte meg. Jónevű szakemberek szakfolyóiratokban vitáztak azon, hogy egy-egy névvariáns mennyire kifejező arra a teljesítőképességre, amelyet ez az új berendezés produkál. Mindenféle jelzőkkel díszítették a főnevet, hogy az elnevezés minél inkább megkülönböztesse a fogalmat a korábbi — minőségileg valóban kisebb képességű — elődeitől.

Ilyen nevek születtek: „digitális, univerzális, programvezérlésű, elektronikus számoló (vagy számító) gép". Idővel ez a „névháború" elcsitult, és ma már csak a „számítógép" esetleg az „elektronikus számítógép" elnevezést használjuk. (Újabban a személyi számítógépek és a professzionális személyi számítógépek megjelenésével ismét színesedik az elnevezések palettája.)

II. fejezet

6

A matematikai logika néhány alapfogalmáról már volt szó az elsős matematikában (állítás fogalma és logikai értéke: 169. oldal, diszjunkció és konjunkció fogalma: 214. oldal). A továbbiakban összefoglaljuk a legfontosabb fogalmakat, amelyekre a későbbiekben szükségünk lehet.

1. Alapfogalmak

A mindennapi életben állításainkat mondatokban fejezzük ki. Ezek lehetnek egyszerű és összetett mondatok, attól függően, hogy egy vagy több állítmány szerepel-e bennük. Ha közelebbről megvizsgálunk egy egyszerű kijelentő vagy tagadó mondatot, amelyben csak egy állítmány szerepel, akkor — elvonatkoztatva az állítás konkrét tartalmától — minden esetben azt mondhatjuk, hogy az adott állításunk *igaz* vagy nem igaz, azaz *hamis*. Ilyenkor már nem a mondatról beszélünk, hanem annak logikai formájáról.

A matematikai logika ezekkel a tartalmi jelentésüktől megfosztott állításokkal foglalkozik úgy, hogy az állításokból azt veszi figyelembe, hogy azoknak mi az igazságtartalma. Az ilyen állításokat kijelentésnek (más terminológia szerint állításnak) nevezzük.

Egy kijelentő mondatot, ha eltekintünk annak tartalmától, nyelvi formájától és csak azt a tulajdonságát vesszük figyelembe, hogy a benne foglalt állítás igaz-e, vagy hamis, *kijelentésnek* nevezzük.

Az igaz és a hamis tulajdonságokat *logikai értékeknek* nevezzük. A logikai értékek tehát a matematikai logikában a konstans — a logikai konstans — szerepét töltik be. Ebben a rendszerben két konstans szerepel, hasonlóan a kettes számrendszerhez. Azt is megállapíthatjuk, hogy ez a két érték egymással kizáró viszonyban van, vagyis nem lehet valami egyidejűleg igaz is és hamis is. Az olyan logikai rendszereket, amelyeknek logikai értéktartománya két, egymást kizáró logikai értékből áll, *kétértékű logikai rendszereknek* nevezzük.

Megjegyezzük, hogy léteznek többértékű logikai rendszerek is, ezekkel azonban nem foglalkozunk.

A logikai értékeknek többféle jelölési módja ismeretes. A továbbiakban az igaz logikai értéket (s így az igaz logikai konstanst is) az I betűvel, a hamis logikai értéket (és így a hamis logikai konstanst) a H betűvel jelöljük. Léteznek egyéb jelölések is.

Ha bizonyos kijelentések igazságtartalmát nem ismerjük, nem helyettesíthetők az I vagy a H értékkel. Ezeket a kijelentéseket csak olyan *változókkal* reprezentálhatjuk, amelyeknek két lehetséges értéke létezik, az I és a H. Ezeket a változókat *logikai változóknak* nevezzük.

A logikai változókat általában a p, q, r, s ... kisbetűkkel jelöljük. Minden logikai változó értékkészletét két lehetséges érték, az I és a H alkotja.

Az elemi kijelentés fogalma. Egy kijelentést akkor tekintünk elemi kijelentésnek, ha nem tudjuk vagy nem akarjuk valamilyen művelettel más kijelentésből előállítani. Az elemi kijelentés logikai értékét adottnak, más kijelentések logikai értékétől függetlennek tekintjük.

A logikai művelet fogalma. Azok a műveletek, amelyeket elemi kijelentéseken hajtunk végre és eredményül ismét kijelentéseket kapunk, miközben az eredmény logikai értéke csak azon értékek logikai értékétől függ, amelyeken a műveletet végrehajtottuk. (Vagyis logikai értékekből logikai értékhez jutunk el.)

A logikai műveletek végezhetők logikai változókkal is, mivel minden egyes elemi kijelentés helyettesíthető egy megfelelő igazságértékű logikai változóval.

2. Logikai műveletek

A továbbiakban olyan logikai műveleteket tárgyalunk, amelyekben egy, ill. két logikai változó szerepel.

Negáció

Negációnak nevezzük azt a logikai műveletet, amelyet a „nem" szóval fejezünk ki. A műveletben egy változó szerepel.

Jelölése: $\neg$

Értelmezése: Ha $p=I$, akkor $\neg p=H$, ill.
ha $p=H$, akkor $\neg p=I$.

A negáció tehát az a nyelvi NEM szóval (vagy annak szinonimájával) jelölt művelet, amely bármely logikai értékből indulva a vele ellentétes másik logikai értékhez jut el.

A negáció értelmezéséből következik, hogy
$\neg\neg p=p$, p bármely igazságértéke mellett, vagyis a kettős negálás eredményeképpen azt a logikai értéket kapjuk, amelyet kétszer negáltunk.

Konjunkció

Konjunkciónak nevezzük azt a logikai műveletet, amelyben ítéleteket a nyelvi ÉS kötőszóval kapcsolunk össze.

Jelölése: $\wedge$

Értelmezése (logikai értékmegoszlása):

p \ q	I	H
I	I	H
H	H	H

A konjunkció tehát a nyelvi ÉS szóval (vagy annak szinonimájával) jelölt logikai művelet, amely csak akkor ad igaz eredményt, ha mindkét tényezője igaz, minden más esetben hamis eredményt ad.

Diszjunkció

Diszjunkciónak nevezzük azt a logikai műveletet, amelyben ítéleteket a megengedő VAGY szóval kapcsolunk össze.

Jelölése: $\vee$

Értelmezése (logikai értékmegoszlása):

p \ q	I	H
I	I	I
H	I	H

A diszjunkció tehát az a nyelvi megengedő VAGY szóval (vagy annak szinonimájával) jelölt logikai művelet, amely csak akkor ad hamis eredményt, ha a művelet mindkét tényezője hamis, minden más esetben az eredmény igaz.

Léteznek további logikai műveletek is, ezek tárgyalására azonban nem térünk ki.

★ 1

A II. osztályos Technika könyv a 134. oldalon az információra a következő meghatározást tartalmazza: „Mind a közvetlen tapasztalat, mind a tanulás útján szerzett olyan ismereteket, amelyek valamilyen — korábban meglevő — bizonytalanságot (ismeretlenséget) oszlatnak el, információnak nevezzük. Az „informatio" ugyan szó szerint közleményt jelent, de az olyan közlés, amelyben a fogadó számára semmi új sincs, nem információ."

A kétféleképpen megfogalmazott definíció ugyanazt a fogalmat jelenti. Az idézet az információnak a bizonytalanságot megszüntető jellegét emeli ki, tankönyvünk megfogalmazása az új ismeretre helyezi a hangsúlyt.

▲ 2

Tekintsünk egy egyszerű példát az információ fogalmának megértésére. A közismert barkochba játék során valakinek ki kell találnia egy személyt. Amikor arra a kérdésre, hogy az illető férfi-e, azt a választ kapja, hogy „igen", információ birtokába

jutott, mert megszűnt a kitalálandó személy nemére vonatkozó „tudatlansága", azaz határozatlansága. A példából is kitűnik, hogy a kérdező számára teljesen közömbös, hogy a kérdésre az „igen" választ milyen formában kapja meg: kimondva, leírva, fejbólintással stb. Számára ezen formák valamelyike csupán az információ hordozójaként szerepelt.

▲ 3

Nézzünk ismét egy példát az elmondottakra. Valamilyen esemény (pl. egy utcai baleset) időpontja 16 óra 27 perc. A rendőrség számára a baleset időpontjául szolgáló információt a következő adat jelenti: 16.27. Ha ebből a négy számjegyből akár csak egyet is elhagynánk, a megmaradt három számjegy már nem képes a baleset pontos időpontját meghatározó információ megjelenítésére, az információ tehát megcsonkul, eltorzul.

▲ 4

Alkalmazzuk az elmondottakat egy konkrét feladatra. Nyilvántartást kell készíteni az iskola tanulóiról. A nyilvántartásba a tanulók következő adatait kívánjuk szerepeltetni:

A tanuló neve
Születési dátuma
Lakhelye
Eltartójának foglalkozása
Testvérei száma
Középiskolai tanulmányainak kezdő éve
Tanulmányi átlaga az előző tanév végén

Az adatfelvételt a tanulókkal végeztetjük el úgy, hogy egy táblázat egy-egy sorába íratjuk be adataikat. Az egyik osztályban kitöltött táblázat eleje így nézhetne ki:

Név	Születési év, hó, nap	Lakhely	Eltartó fogl.	Testv. száma	Kezdő év	Átlag
Balla Béla	1969. I. 22.	II. Kacsa u. 8.	eladó	2	1983	4,1
Gőz Ede	1969. V. 7.	V. Ó u. 77.	marós	1	1983	4,3
Pék Pál	1967. X. 34.	I. Fő u. 6.	meós	0	1982	2,3

Az osztályban 32 tanuló van, tehát a teljes táblázat 32 sorból fog állni. Látjuk, hogy a sorban az osztálybeli egyedek (most tanulók), az oszlopokban pedig a korábban kiválasztott tulajdonságok jelennek meg.

▲ 5

Annak illusztrálására, hogy az adat típusa hogyan határozza meg a vele végezhető műveletek körét, nézzünk egy egyszerű példát.

Péter minden nap kétszer körülfutja a háztömböt. Legutóbb az első kört 1 perc 45 mp alatt tette meg, a másodikat 1 perc 56 mp alatt. Mennyi ideig futott Péter? A választ gyorsan ki tudjuk számítani: 1 perc 45 mp + 1 perc 56 mp = 3 perc 41 mp. Kinek jutott volna eszébe azt mondani, hogy 2 perc 101 mp, vagy olyat, hogy 3 perc 1 mp?

Megtanultuk valamikor, hogy melyek azok az időegységek, ahol a váltószám hatvan.

Ezt másképpen úgy is fogalmazhatjuk, hogy az idő típusú adatnál az összeadás másképpen „viselkedik", gépi műveletvégzés esetén egy olyan összeadást kell megvalósítani, amelyik „ismeri" a hatvanas váltószámot. (Ilyen művelet persze nem létezik, de program segítségével előállítható.)

▲ 7

Pl. egy feldolgozásban a következő fájlok szerepelhetnek:

Termék törzsfájl:

minden termékről tartalmazza a gyártásához szükséges technológiai adatokat. Egy rekordja egy termékre vonatkozik.

Rendelésfájl:

tartalmazza egy időszakra vonatkozóan a megrendeléseket. Egy rekordjában az adott termékre vonatkozó rendelési mennyiséget, a vevők kódszámát.

Vevő törzsfájl:

a vevők adatait (név, cím, bankszámlaszám) tartalmazza.

Valamennyi fájl közös jellemzője, hogy a funkciójához tartozó összes rekordot tartalmazza. (Ha 100 féle terméket gyárthatnak, akkor az első fájlban 100 rekord van, ha 600 vevője van a vállalatnak, akkor a harmadik fájlnak 600 rekordja kell, hogy legyen stb.)

▲ 8

Gyerekek kedvenc „titkosírása", amikor egy szövegben minden betű helyére az abc-ben kettővel vagy hárommal rákövetkező betűt írják. Az így kódolt szöveget csak az tudja elolvasni, aki ismeri ezt a szabályt. Ha a szöveg idegen kezébe kerül, és sikerül megfejtenie a szöveg tartalmát, akkor a szabályra sikerült rájönnie.

Amióta háborúk léteznek — és sajnos nagyon régóta léteznek —, a hadban álló felek számára mindig fontos volt, hogy üzeneteiket úgy juttassák el saját csapataik-

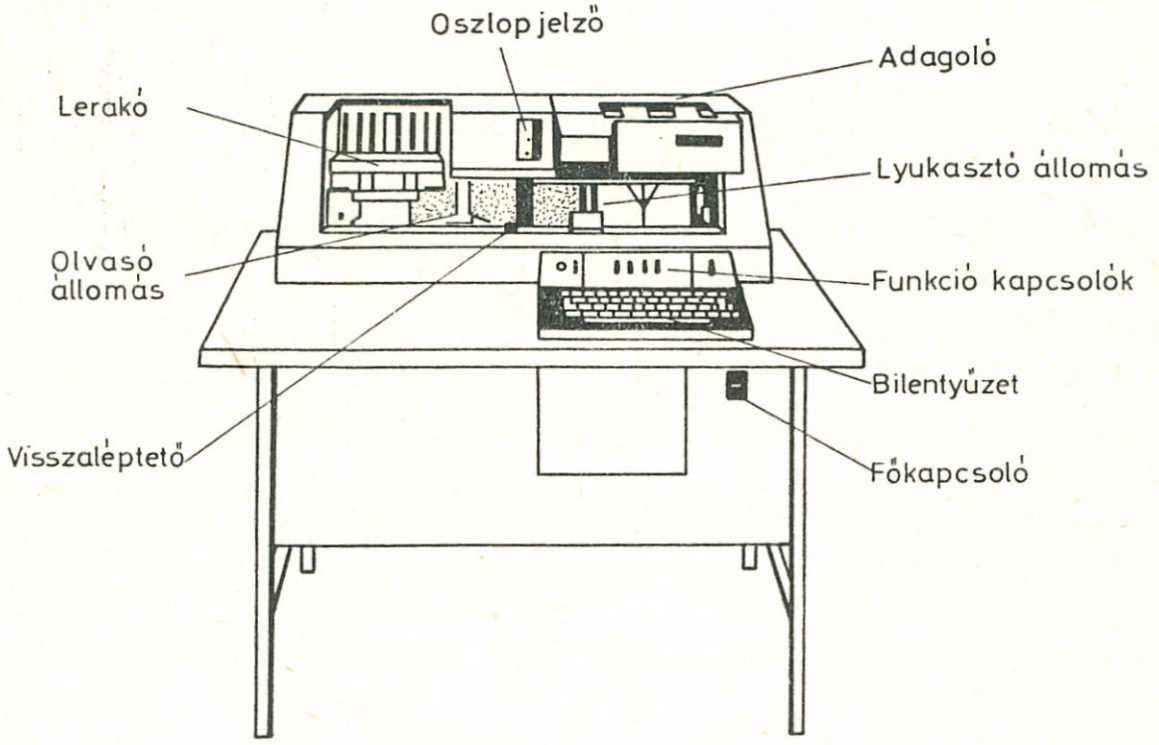
hoz, hogy azt az ellenség — ha birtokába jut — ne tudja elolvasni. Ezért aztán nem csodálható, ha a hadászatban a kódolás és a dekódolás, a nehezen megfejthető kódolási szabályok kitalálása külön területté vált.

9

Lyukkártyalyukasztó berendezések

Ha a szervezési munka során így döntenek, a kialakított kártyaterv alapján az elsődleges adathordozókon rögzített adatokat lyukkártyára kell rögzíteni. Ezt a munkafolyamatot a lyukkártyalyukasztó berendezések segítségével az adatrögzítők végzik el. Sokféle kártyalyukasztó van forgalomban hazánkban is. Bár alapfunkciója mindegyik típusnak a kártyára való lyukasztás, az ennek érdekében nyújtott egyéb szolgáltatásaik nagyban eltérhetnek.

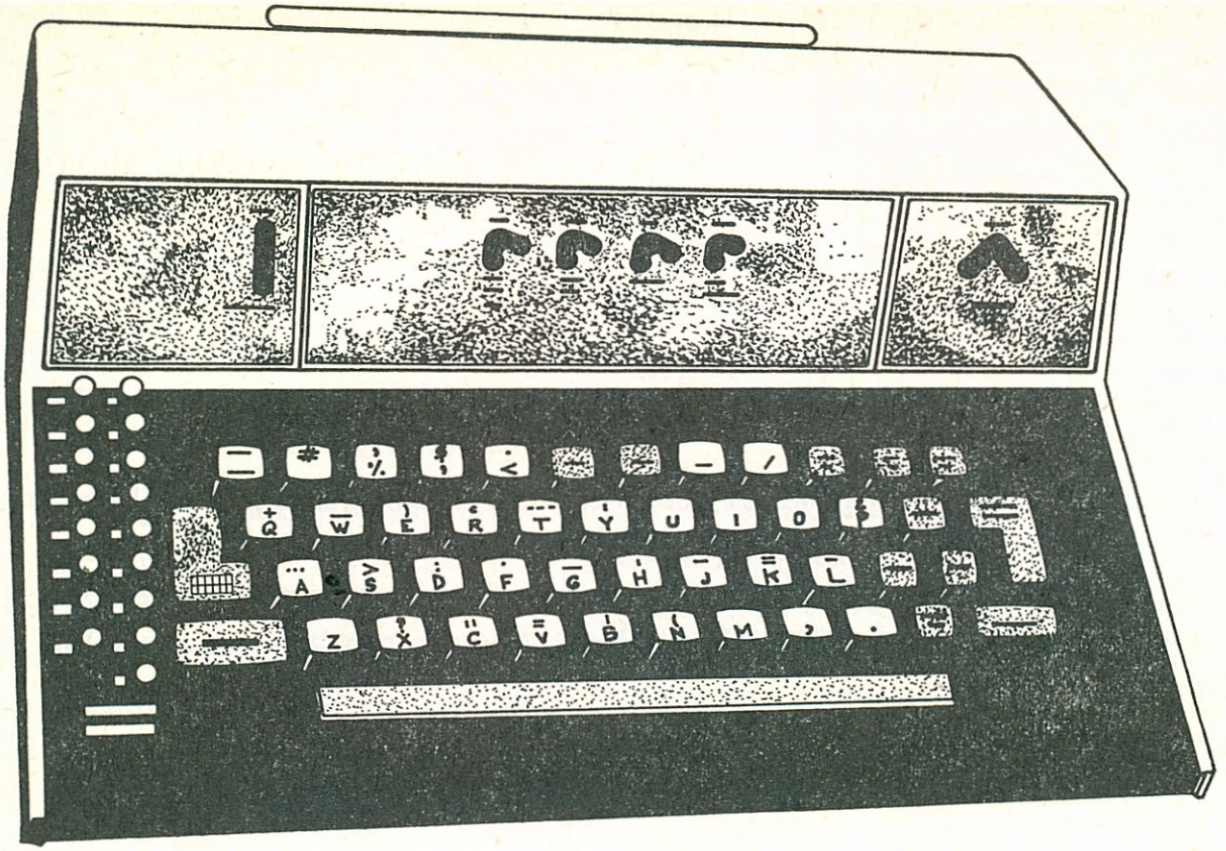
Lássuk, milyen főbb szolgáltatásokat nyújt pl. az IBM 029 lyukasztógép.



A-4. ábra. IBM 029 kártyalyukasztó

A berendezés a következő fő funkcionális részekből áll:

— Billentyűzet. A kártyalyukasztó billentyűzete numerikus és alfanumerikus karaktereket tartalmaz, valamint olyan „vezérlő billentyűket", amelyek a kártyák kezelésére (mozgatás, másolás) vonatkozó tevékenységeket biztosítanak. A számjegyek billentyűi a jobb kéz hatósugarában vannak összpontosítva. Az adatrögzítő így nagyobb sebességgel tudja a csak numerikus adatokat lyukasztani. Ez a berendezés a betűknek csak a nagybetűs alakját tartalmazza.



A-5. ábra. Billentyűzet

— *Kártyatartó.* Az ide helyezett üres kártyákból a vezérlőbillentyűzet használatával lehet kártyát mozgatni a lyukasztóhoz.

— *Lyukasztó állomás.* Itt található a lyukakat előállító 12 kés (minden sorhoz egy), valamint az a mechanizmus, amely a kártya tetejére a lyukkombinációnak megfelelő karaktert felírja.

— *Érzékelő állomás.* Itt érzékelhetők egy már lyukasztott kártya lyukkombinációi. Szerepe akkor van, ha egy kártyát teljes egészében vagy részleteiben kell másolni. (Ennek segítségével lehetséges pl., hogy egy hibásan lyukasztott kártyát nem kell teljes egészében újralyukasztani.)

— *Kártyalerakó.* Itt kerül gyűjtésre és innen lehet kivenni a kész kártyákat.

— *Programdob.* Valójában egy henger, amelynek palástjára lehet felrakni a programkártyát. A programkapcsoló bekapcsolásával az előre programozott tevékenységeket automatikusan elvégzi minden kártyánál. A programdob szinkronban mozog a lyukasztóállomáson áthaladó kártyával. A programdob alján figyelemmel kísérhető, hogy a kártya hányadik oszlopa van lyukasztási pozícióban, mivel ez közvetlenül nem látható (a lyukasztókések miatt takarva van).

— *Visszaléptető.* A lyukasztó és érzékelő állomás között levő gomb lenyomásával a kártya egy oszlopnyit visszafelé mozog.

A vezérlőbillentyűkön kívül a kártyalyukasztón néhány funkciókapcsoló is befolyásolja a gép működését. Pl.:

— lehetőség van a feliratozás letiltására;

— lehetőség van új kártya leemelésének a letiltására;

— minden kártyát ki lehet üríteni a gépből stb.

A gép bekapcsolása után két kártyát le kell hívni (pl. a FEED gomb lenyomásával). Ezt követően egy-egy kártya lyukasztása után automatikusan érkezik az új kártya (feltéve, hogy nem tiltjuk le). A lyukasztott kártya az érzékelő állomáshoz kerül, míg a korábban ott levő a lerakóba.

Elsősorban adatok rögzítésénél sokszor előfordul olyan tevékenység, amelyet minden rekord esetében el kell végezni. Pl.:

— adatmezők — vagy azok egy része — ugyanazzal az értékkel ismétlődik (ilyen eset fordul elő, ha a vállalat vagy az üzemegység nevét vagy kódját minden kártyán fel kell tüntetni);

— meghatározott oszlopokban nem kell lyukasztani (üresen kell hagyni);

— a numerikus rekord egy-két mezőjébe alfabetikus lyukasztás kerül.

A programkártya, amely a lyukasztandó kártyával szinkronban mozog olyan vezérlőlyukat tartalmazhat, amelyek a felsorolt tevékenységek automatikus elvégzését indítják el.

III. fejezet

■ 11

Blokkdiagramokkal találkoztatok már a II. osztályos Technika keretében. Ott azt a megfogalmazást tanultátok, hogy a blokkdiagram lényegében egy olyan szemléletes írásmód, amely még a legbonyolultabb eljárások áttekintését is megkönnyíti. Lényege, hogy a számítás egyes önálló részeit (ún. blokkokat) keretbe foglaljuk, és nyilakkal jelöljük a blokkok közötti sorrendet.

Tananyagunkban a blokkdiagramoknál használt művelet szót (azonos tartalommal) a *tevékenység* szóval helyettesítjük, mert itt a művelet kifejezést más célokra használtuk fel (pl. művelet adatsorral).

★ 5

Az **algoritmus** fogalma már megjelent a II. osztályos Technika c. könyvben, bár ott maga a fogalom definiálásra nem került. „A számítások egymásutáni lépéseit (az ún. algoritmust) hosszadalmas lenne mindig szavakkal leírni." — olvashattátok a 142. oldalon. Most ugyanezzel a fogalommal ismerkedünk meg részletesebben

---

▲ 1

El akarunk menni kerékpárral az Őrségbe (Vas megye). Mivel egy nap alatt nem tudjuk megtenni az utat, több napra tervezzük, gondosan kiválasztva az éjszakai szálláshelyeket. Minden szálláshelyre való megérkezés egy új „állapotot" jelent, és minden „állapot" közelebb visz az úticélhoz.

▲ 2

Egy valóságos programban ilyen funkciót valósít meg pl. egy feltételes vezérlésátadó utasítás.

Erre jó példa a másodfokú egyenlet valós gyökeinek meghatározása során az a pont, amikor el kell dönteni, hogy a diszkrimináns (b^2—4ac) negatív-e vagy nem. (Negatív diszkrimináns esetén ui. nincs valós gyök.) Máshol folytatódik a megoldás folyamata, ha a gyökvonás elvégezhető és megint máshol, ha a gyökvonás nem végezhető el.

▲ 4

A **program** döntő fontosságú szerepét akkor érthetjük meg, ha meggondoljuk, hogy egy számítógépnek nem lehet feladatot adni, mert a gépnek nincs emberi értelemben vett intelligenciája, amellyel a feladatot analizálni tudná, s a megoldásra rá tudna találni. A számítógéppel csupán programok közölhetők, s a gép a közölt algoritmus részlépéseit szolgai módon végrehajtja.

Neumann János azt írja egy helyen ezzel kapcsolatban, hogy a számítógépeknek „sajátosságuk, hogy mindazt, amit velük közöltek, abszolút komolyan és szó szerint veszik, azaz valóban azt végzik el, amit előírtak nekik, beleértve a sajtó- és gondolkodási hibákat is."

A számítógép működésének helyessége ilyen értelemben kizárólag a program jóságától függ. A számítógép alkalmazási korlátai még sokáig nem magában a gépben, hanem az őt alkalmazó emberben lesznek.

▲ 8

A kör területét meghatározó matematikai képletben

$$T = r^2\pi,$$

ahol T a terület, r a kör sugara, és nyomon követhető a konstans és a változó. Az r helyére mindig az aktuális sugarat írjuk be, a π helyére mindig a 3,14 értéket.

▲ 10

A 9. ábra azt szemlélteti, hogy egy valóságos programozási nyelv alkalmazásakor mi történik, ha *adatbeviteli művelet* végrehajtására kerül sor. (Az ábrán megadottak csak a történés logikáját mutatják, a valóságban ugyanez áttételesebben érvényesül.)

A READ utasítás három változót — az ABA, a BABA és a CICA szimbólumokkal azonosítottakat — kíván értékkel feltölteni. Az olvasási pozícióban levő lyukkártya három mezőjében rendre a következő konstans értékek szerepelnek: 125 337 25. Az utasítás hatására a változók — a felsorolás sorrendjében — felveszik a három konstans értékét, s a számítógép központi tárának három — egymástól független — fizikai helyén a három érték tárolásra kerül. A korszerű programozási nyelvek esetében a programozó nem tudja, hogy ezek a konkrét helyek hol vannak a tárban, de a megfelelő névvel a program során hivatkozhat rájuk, s a hivatkozás hatására a megfelelő érték rendelkezésre áll (pl. egy értékadás jobb oldalán).

▲ 12

A képletben az a, b és c a háromszög oldalának mérőszáma, az s a háromszög kerületének a felét jelenti.

▲ 13

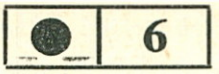
A háromszög egyenlőtlenség értelmében egy háromszög bármelyik két oldalának összege nagyobb a harmadik oldalnál.

▲ 20

Bútorvásárláskor előbb azt döntjük el, hogy milyen funkciójú bútort akarunk. Mondjuk egy fekhelyet, két fotelt és egy szekrényt. Utána gondolkozunk azon, hogy milyen anyagú, formájú, színezetű legyen a háromféle bútor. Arra is figyelünk, hogy egymáshoz és a szoba már meglevő berendezési tárgyaihoz illeszkedjenek.

● 3

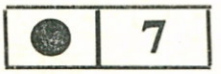
A különböző programozási nyelvek eltérő módon és eltérő mértékben kívánják meg az adatdefiníciók meghatározását. Van olyan nyelv — pl. a FORTRAN —, amely rendelkezik automatikus típusdeklarálással. Ez azt jelenti, hogy a változó kezdőbetűje egyben megadhatja az adott változó típusát (csak egész, ill. valós típus esetén). Ez első hallásra kényelmesnek tűnik, hiszen programozási munka takarítható meg. Más oldalról viszont az is belátható, hogy ha egy programban minden definíció tételesen benne van, ez nagyban segíti a program megértését, a későbbiek során való változtatást, fejlesztést.



Meg lehet pl. fogalmazni a fagylaltvásárlás és a fagylaltevés algoritmusát valahogy így:

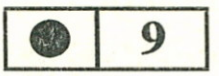
1. Ha van pénzed és nincs nagyon hideg, keress egy cukrászdát.
2. Állj be a sorba.
3. Haladj a sorral.
4. Ha rád kerül a sor, mondd meg, hogy milyen fagylaltot kérsz.
5. Fizess és vedd el a tölcsért.
6. Óvatosan edd, hogy ne csurogjon a ruhádra és ne fázzon meg a torkod.
7. Ha vége, töröld meg a szádat.

De nyilván senkinek sem jutna eszébe, hogy a fagylaltot a számítógéppel etesse meg, még akkor sem, ha a számítógép erre képes lenne.



A különböző programozási nyelvekben többféle adatbeviteli és adatkiviteli utasítás létezik. Így pl. a kölcsönzött READ és WRITE utasításokon kívül a GET (behozatal) és PUT (kivitel) is lehetséges utasítások.

A konkrét programozási nyelvek pontosan definiálják az adatmozgató utasításaikat.



Néhány példán kövessük nyomon az elmondottakat.

```
x:=5
```

Az x nevű változó a továbbiakban 5-tel lesz egyenlő.

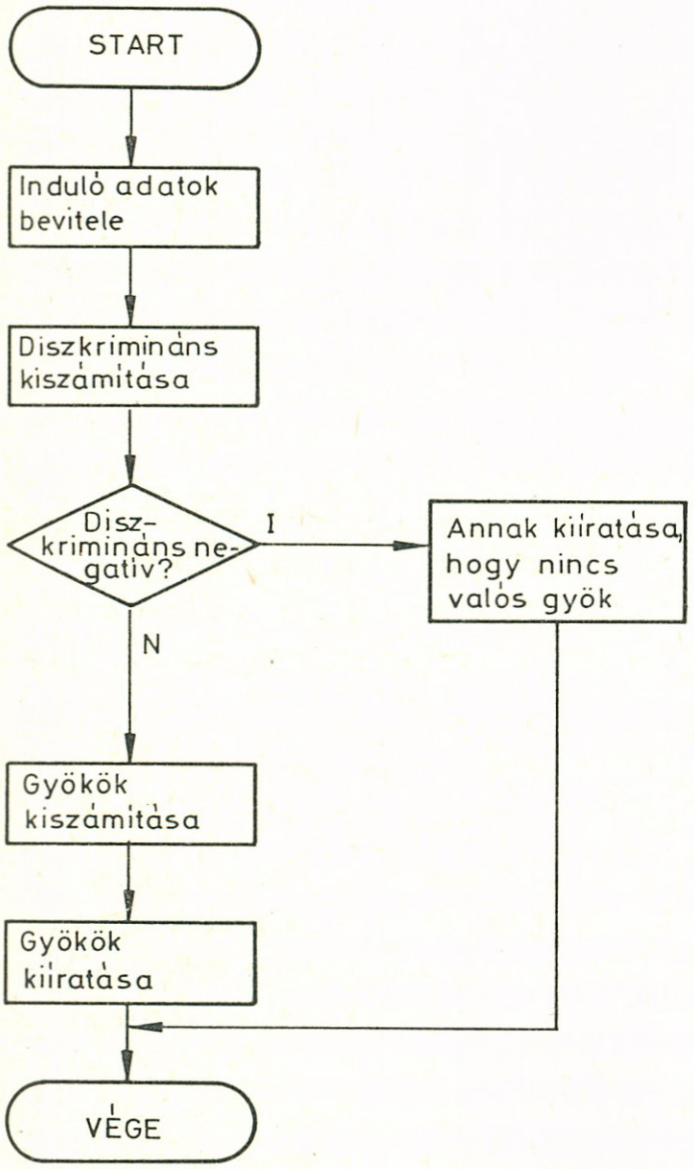
```
g:=4
b:=3
g:=g+b
```

Az első két művelet g-nek és b-nek értéket ad, majd a g nevű változó felveszi a 4+3 művelet eredményét, a 7-et, és g korábbi értéke (a 4) már nem létezik tovább (felülírásra került).

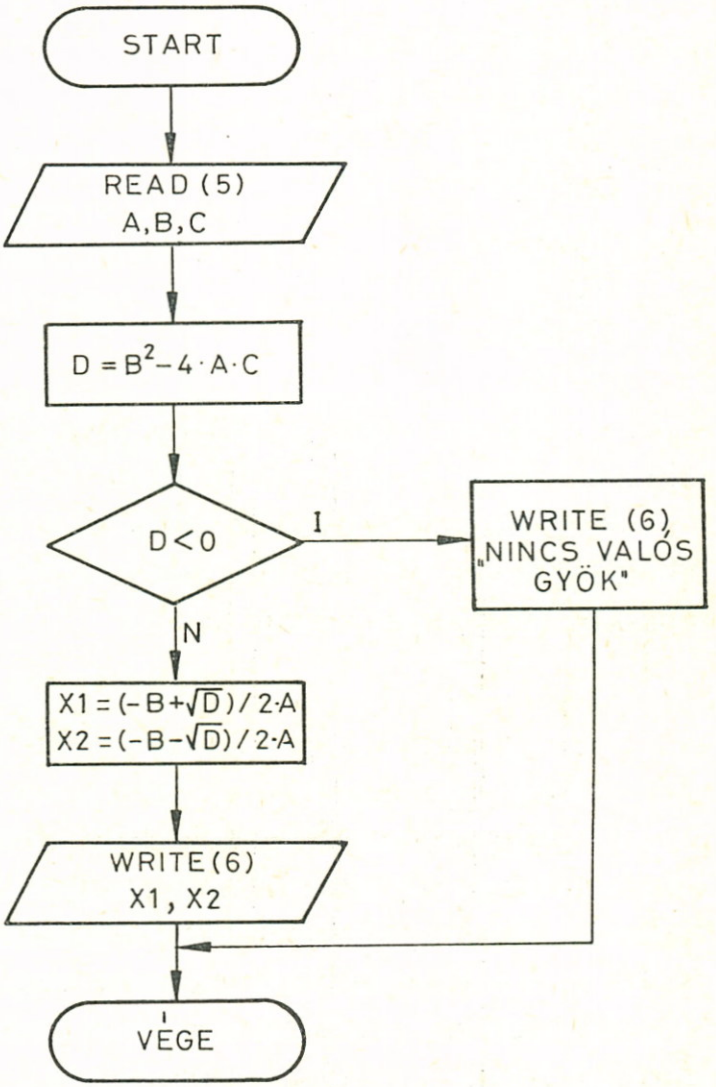
```
y:=2
x:=y—z
```

x értéke meghatározhatatlan, mert z értéke is az, tehát az ilyen alkalmazás tilos.

Maga a blokkdiagram jelölésrendszere nem ír elő semminemű szabályt arra vonatkozóan, hogy az egyes jelekbe miképpen rögzítsük az esedékes algoritmus lépést. Ennek illusztrálására tekintsük meg az A-6. és az A-7. ábrát, amely ugyanannak a feladatnak, a másodfokú egyenlethez tartozó gyökök meghatározásának leírását adja.



A-6. ábra. Másodfokú egyenlet megoldása 1.



A-7. ábra. Másodfokú egyenlet megoldása 2.

Láthatjuk, hogy mindkettő eleget tesz mind a blokkdiagram jelölésrendszerének, mind az algoritmusra adott megfogalmazásunknak. A kettő között mégis nagy különbség van. Míg az első felbontja ugyan részlépések egymásutánjára a feladatot, de a matematikai hátteret az algoritmus felhasználójának ismernie kell. Ezt az algoritmust csak olyan személy tudja programozni, aki tudja pl. hogy mit jelent matematikailag a „gyökök kiszámítása". A második megoldás matematikailag is korrekt, bárki programozhatja, aki a szükséges nyelvet ismeri. Matematikai felkészültség nem szükséges a program megírásához.

● 15

A magas szintű programozási nyelvekben a **ciklusszervezést** automatikusan megvalósító utasítások találhatók. Itt tehát (ismerve a ciklus működési elvét) a programozónak nem kell törődnie a ciklusszervezés technikájával.

Pl. BASIC-ben

```
FOR I=1 TO 15
ciklusmag
NEXT I
```

vagy FORTRAN-ban

```
DO 12 I=1,15
ciklusmag
12 CONTINUE
```

vagy PL/1-ben

```
DO I=1 TO 15
ciklusmag
END;
```

Ezek a ciklusszervezési utasítások arra a legegyszerűbb esetre vonatkoznak, amikor a ciklusnak adott kezdőértéktől adott végértékig 1 lépésközzel kell működnie.

A magas szintű nyelvek közül több esetében ennél bonyolultabb ciklusszervezési lehetőség is van. (Pl. a ciklus addig működik, amíg valamilyen feltétel fennáll, vagy a lépésköz lehet negatív szám is stb.)

● 17

Kezdő programozók egyik legtöbbször elkövetett hibája a „végtelen ciklusok" létrehozása. A végtelen ciklus egy olyan utasítássorozat, amelyet egy vezérlésátadás hatására állandóan ismételni kell. A programozási hiba az esetek nagy részében a ciklusból való kilépés feltételének rossz megfogalmazásából ered.

A következő példa jól szemlélteti az ilyen hibás feltételmegadást, bár ennél sokkal durvább hibák is adódnak a kezdőknél.

A feladat a következő: egy függvény helyettesítési értékeit kell kiszámítani és nyomtatni a [0,1] intervallumban 0,2 lépésközönként.

A programozó a ciklust úgy szervezte meg, hogy egy számláló 0-ról indulva mindig 0,2-del növekszik. A ciklusból való kilépés feltétele ez volt: $x=1{,}0$, vagyis akkor lépjen ki, ha x már egyenlő 1-gyel.

Látszólag semmi probléma nincs. A baj az, hogy a 0,2 a kettes számrendszerben egy végtelen szakaszos tört, így egzakt pontossággal nem ábrázolható. Következésképpen az ilyen „majdnem" 0,2-ek összege sohasem lesz egzakt pontosságú 1. A feltétel tehát soha sem teljesül.

Úgy kellett volna „ha $x \geq 1$", fejezze be a ciklust.

● 18

A fájl vége jel, ami tehát az elmondottak szerint az adatállomány végén van, fizikailag is megjelenik az adathordozón, amelyre az állományt elhelyezzük. Lyukkártyás állományok esetében pl. a /* karakterkombináció jelzi a fájl végét. Ugyanígy egy önálló rekord az EOF a mágnesszalagon és a mágneslemezen is.

● 19

A számítástechnika korai szakaszában a programozást *mesterségnek*, a programozót mesterembernek tekintették. Az a nézet járta, hogy — akár egy cipőt — egy programot is az tud jobban elkészíteni, aki gyakorlottabb, aki ügyesebb.

Később a programozási munkára úgy tekintettek, mint *művészetre*. Sok cikk jelent meg a programozási stílusokról, arról cikkeztek, hogy a programozási munkának esztétikai oldalai vannak.

Az elmúlt évtizedben kialakult az a nézet, hogy a programozás *tudomány*, s egyre több olyan könyv jelenik meg, amelyek megteremtették a modern programozás egzakt elméleti alapjait.

● 22

Mindenható a számítógép?

(Olvasmány)

A mesebeli király nem akarta lányát a mese hőséhez feleségül adni, inkább feladatokat adott neki. Az egyik így szólt: reggelre számold meg hány csillag van az égen! A király biztosan nem tanult sem matematikát, sem csillagászatot, de irigysége ösztönösen megsúgta neki, hogy ezt a feladatot nem lehet megoldani. És ezen a tényen az sem sokat változtatott volna, ha hősünknek lett volna egy számítógépe.

Persze mi már azt is meg tudjuk mondani, hogy ez a feladat miért nem oldható meg: mert nem véges számú lépésből állna az algoritmusa.

Sok olyan kérdést lehetne feltenni, amelyre még számítógéppel sem lehet választ adni. A következő heti lottó számokat pl. nem lehet előre kiszámítani. (Azt meg lehet mondani, hogy hány lottószelvényt kellene kitölteni ahhoz, hogy biztosan legyen öttalálatos, de ez többe kerülne, mint amennyit nyerni lehet.)

Sikertelen kísérletekre mondják a fizikusok, hogy a természet csak a jól feltett kérdésekre válaszol. És milyen kérdésekben ad választ a számítógép? Adható-e általános válasz e problémára?

Bármily furcsán hangzik, a kérdést több, mint egy évtizeddel az első számítógép tényleges megépítése előtt Turing angol matematikus már megválaszolta. Ő fogalmazta meg elsőnek, hogy egy számoló automatával minden olyan probléma megold-

ható, amelyre „effektív eljárás", mai szóhasználattal élve algoritmus adható. Az algoritmizálható feladatok tehát megoldhatók számítógéppel.

Több matematikus foglalkozott az algoritmizálhatóság feltételeinek vizsgálatával. Az elméleti matematika területén sikerült is találni néhány olyan problémát, amely bizonyíthatóan nem algoritmizálható.

Az algoritmus megléte egy adott problémakörre, még csak a megoldhatóság elvi kérdéseit tisztázta. Ahhoz, hogy a probléma a valóságban is számítógéppel megoldásra kerüljön a megfelelő kapacitással rendelkező gépen kívül, még valamire szükség van. Megfelelő adatokra. A számítógép munkájának minősége ui. a program és az adatok minőségétől függ. Tökéletes program esetén (tegyük fel, hogy ez fennáll), minden az adatokon múlik. Ha „szemetet" viszünk be a gépbe, „szemét" lesz az eredmény is, amit a számítástechnikai szakzsargon az angol „Garbage in — garbage out" kifejezésből GIGO-nak rövidít.

⊙ 16

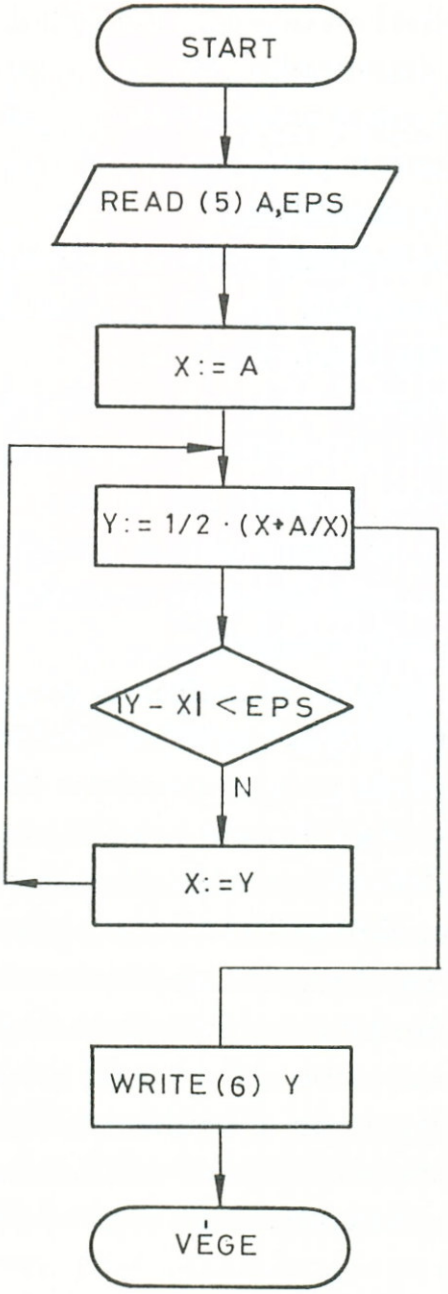
Vannak ciklusok, amelyeknek menetszáma előre nem számítható ki. Ezeknél a kilépés ténye pl. egy, a számolásban résztvevő mennyiség nagyságrendjének alakulásától függ. Jól példázza az ilyen ciklusokat a $\sqrt{A}$ kiszámításának algoritmusa. Anélkül, hogy a probléma matematikai bizonyítását adnánk, megmutatjuk, hogy miképpen lehet kiszámítani egy pozitív szám négyzetgyökét a négy alapművelet segítségével. Legyen A egy pozitív szám, amelynek négyzetgyökét keressük, és X_0 a $\sqrt{A}$ első közelítő értéke. Ekkor kiszámítjuk a $X_{i+1} = 1/2(X_i + A/X_i)$ $(i=0, 1, 2, \ldots)$ mennyiségeket.

Ezek az egymást követő lépések $\sqrt{A}$ értékét közelítik. Ha A négyzetszám volt, néhány lépés múlva megkapjuk a gyököt, s ettől kezdve minden lépésben a gyököt kapjuk. Tehát, ha a befejezés feltételeként azt adnánk, hogy legyen vége az eljárásnak, ha két egymást követő közelítés megegyezik, az iteráció véget érne. Ha azonban A nem négyzetszám, akkor a $\sqrt{A}$ irracionális szám, vagyis elvileg a végtelenségig lehetne folytatni az eljárást, s így a $\sqrt{A}$ egyre nagyobb pontossággal áll elő.

Ezért meg kell adni egy olyan értéket, amely a számolás pontosságát szabályozza. (Pl. ha két tizedes pontosságra kívánjuk a gyököt egy EPS nevű változó értékét 0,01-re választjuk, s az iterációt addig folytatjuk, amíg két egymást követő közelítő gyök különbsége nem lesz kisebb, mint EPS. EPS értékét csökkentve az iterációk száma növekszik. De nem tudjuk előre megmondani, hogy adott EPS mellett hányszor hajtódik végre a ciklusmag.

Az A-8. ábrán bemutatjuk az elmondott algoritmust.

Az algoritmusban (a képlettől eltérően) nem használtunk tömböt, mert felesleges. Induló értéknek magát A-t választottuk.



A-8. ábra. Négyzetgyökvonás iterációval

A monolitikus programozástól a strukturált programozásig

A világban létező különböző bonyolultságú szoftver termékek száma robbanásszerűen növekszik. Ezzel egyidőben növekszik a bonyolultabb szoftverek részaránya is. (Gondoljunk csak pl. az űrkutatást támogató számítógépes programrendszerekre.)

A szoftverek azonban továbbra is a hagyományos módon készültek. Erre a „hagyományos" módszerre alapvetően az jellemző, hogy az általában blokkdiagram formában megtervezett programot először „szőröstől-bőröstől" beprogramozták, majd elkezdték tesztelni, a programozáskor elkövetett hibákat kiküszöbölni. Hogy ezzel a munkastílussal hogyan alakult a ráfordított idő, arra legyen példa annak az operációs

rendszernek egy változata, amellyel a későbbiekben találkozni fogunk. A OS/360 operációs rendszer tervezésére az összmunka 33%-át, a programozásra az összmunka 17%-át, míg tesztelésére a teljes munkamennyiség 50%-át (!) fordították.

Az ilyen módszerrel létrehozott programoknak ugyanakkor több negatív tulajdonsága volt. Ezek közül a legfontosabbak:

— a nehézkesség — a tesztelhetőség szempontjából (lásd a korábbi példát),

— módosíthatóság szempontjából (a programba annak bonyolultsága miatt nehéz változtatási igénnyel utólag belenyúlni);

— megbízhatatlanság — miután a program nincs tesztelve az input adatok összes lehetséges előfordulására, még ún. „kitesztelt" programok esetén is előfordulhat, hogy bizonyos adatelőfordulások esetén hibásan működik;

— udvariatlanság — merev az input adatok összeállítási szabálya, az outputok esetében nincs mód szelektálásra, csak bizonyos eredmények lekérésére, a hibajelzések nem egyértelműek, sokszor semmitmondóak.

A programozásnak ezt a módját **monolitikus** programozásnak nevezi a szakirodalom. A monolitikus programozás „csődjét" annak a felismerésnek a hiánya okozta, hogy egy nagy számítógépes programrendszer létrehozása egy kicsihez képest nem csak mennyiségben jelent többet, hanem minőségileg is más feladat.

Talán jó példa ennek szemléltetésére a következő:

Egy kisebb település telefon előfizetőinek száma pár tucat. Ismerünk egy telefonszámot, és van egy előfizetők neve szerint abc-be rendezett telefonkönyvünk. Meg akarjuk nézni, hogy az adott szám kinek a hívószáma. Érezhető, hogy ez a feladat, egy telefonszám kikeresése száznál kevesebb szám közül, némi idő ráfordításával könnyen megoldható. Ha ugyanez a helyzet, de a kérdéses előfizető Budapesten van, akkor most már három vastag kötetben kell keresni, a feladat nemcsak mennyiségileg, időben több, hanem minőségileg is. Az emberi agy és szem fáradékonyságát is belekalkulálva, talán nem is megoldható.

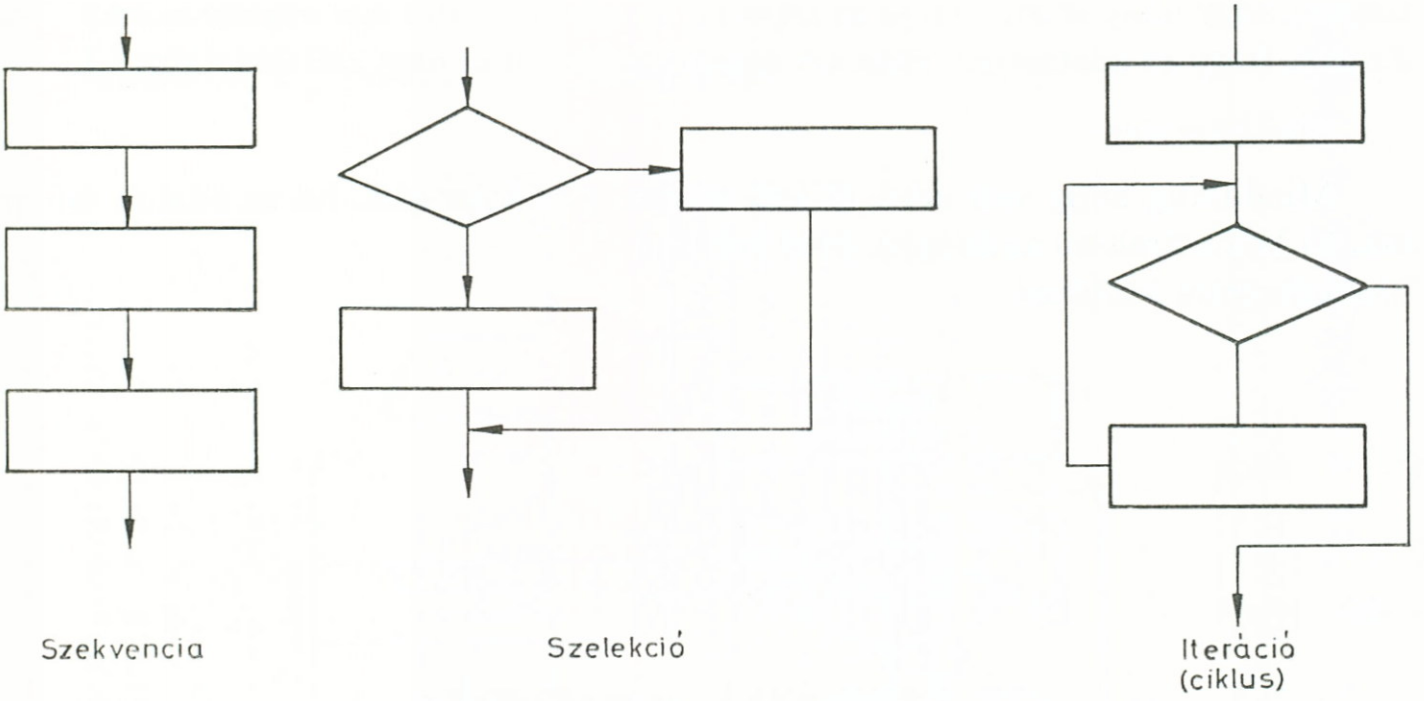
Ugyanez a helyzet a programozásban is. Amikor erre a felismerésre rájöttek, megkezdődött a kivezető út keresése.

Az egyik előrelépés a **moduláris programozás** megjelenése volt. A modulárisan megírt program részprogramok jól definiált kapcsolatrendszeréből áll. (Ennek következetes végigvitele a funkcionális lebontáson alapuló programtervezéshez vezet.)

Jelentős áttörést eredményezett a **strukturált programozás** elméleti alapjainak kidolgozása. Bebizonyították, hogy bármilyen probléma algoritmizálható három algoritmuselem segítségével. E három algoritmuselem a szekvencia, a szelekció és az iteráció. (Jelentésük jól leolvasható az A-9. ábráról.)

Később a kényelmesebb használhatóság kedvéért további három elemet is bevezettek.

Látható az ábrán, hogy minden algoritmuselemnek egy belépési és egy kilépési pontja van. Ez nagymértékben megkönnyíti az áttekinthetőséget, és alkalmat ad a **lépésenkénti finomítás** alkalmazására. Ez az elv nagy vonalaiban a következőket jelenti:



A-9. ábra. Alapvető struktúrák

A hangsúly a programtervezésen van, szemben a hagyományos eljárással, ahol — mint láttuk — a programtesztelés viszi el a legtöbb időt.

A program több absztrakciós szinten készül lépésről lépésre. Az egyes szinteknek megfeleltetett programot csak valamilyen elképzelt gép tudná végrehajtani. A feladat éppen az, hogy végül is egy olyan nyelvű programhoz jussunk, amelyhez már valóságos gép is hozzárendelhető.

A program a legmagasabb absztrakciós szinten könnyű beláthatósággal jól működik. Az absztrakt programból a „kevésbé" absztraktba úgy kell átlépni, hogy ezzel a program jósága ne károsodjék. Így végül a végrehajtható program is hibátlan lesz.

Egy olyan feladat esetében, ahol a feldolgozás az adatbevitel—adatfeldolgozás—eredményközlés hármas műveletsorral oldható meg, a legabsztraktabb program a következő formájú:

„input",
„feldolgozás",
„output".

Ha lenne olyan gépünk, amely ezt a három szót utasításként tudná értelmezni, és végre tudná hajtani, már kész is volna a program. De ilyen valóságos gép természetesen nincsen. Ezért ezeket az „utasításokat" tovább kell bontani, olyan programelemekkel és módszerekkel, amelyek közelítenek egy valóságos gép adottságaihoz.

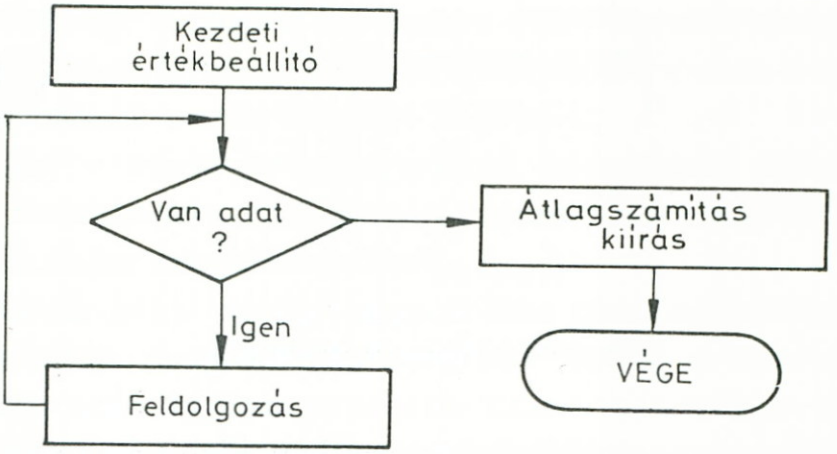
Az elmondottakat ismét egy példával szemléltetjük. (E példa kapcsán ismételten utalunk rá, hogy az ilyen tervezési mód előnyei nagy, nehezen áttekinthető problémák megoldásakor jelentkeznek. Ugyanakkor itt — érthető okokból — csak kisméretű feladatot lehet bemutatni.) A feladat a következő:

Pozitív valós számokat olvassunk egyenként, és számítsuk ki ezek átlagát. Nem

tudjuk, hogy hány szám alkotja az inputot, az utolsó adatot egy negatív szám követi. Ez jelzi, hogy az adatsornak vége. (A negatív számot már nem kell feldolgozni.)

Első közelítés:

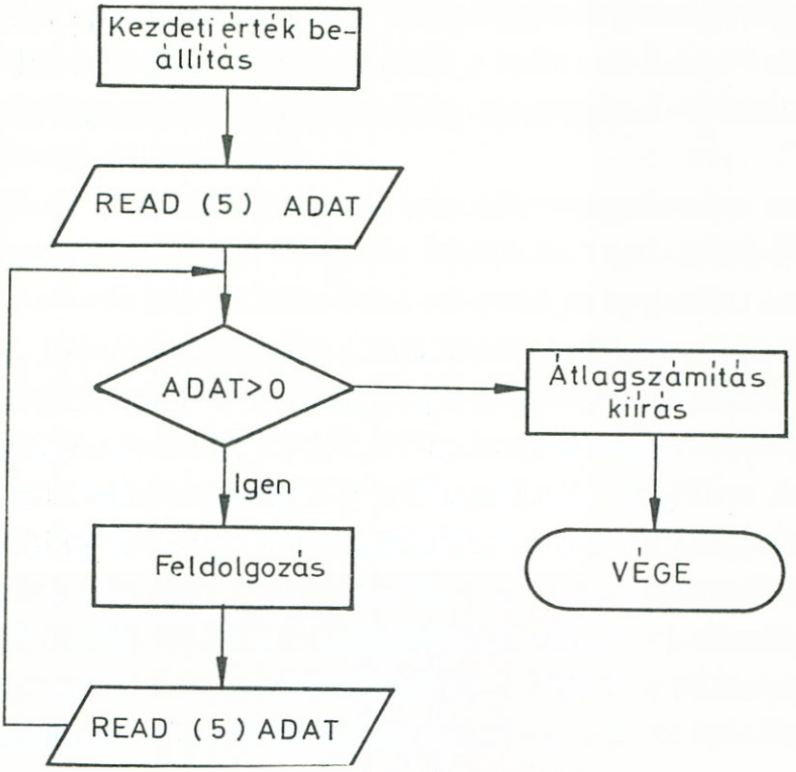
Mindaddig amíg van adat, el kell végezni a feldolgozást, ha az adatok elfogytak, ki kell számítani az átlagot, és ki kell írni.
Blokkdiagram jelöléssel:



A-10. ábra. Első közelítés

Második közelítés:

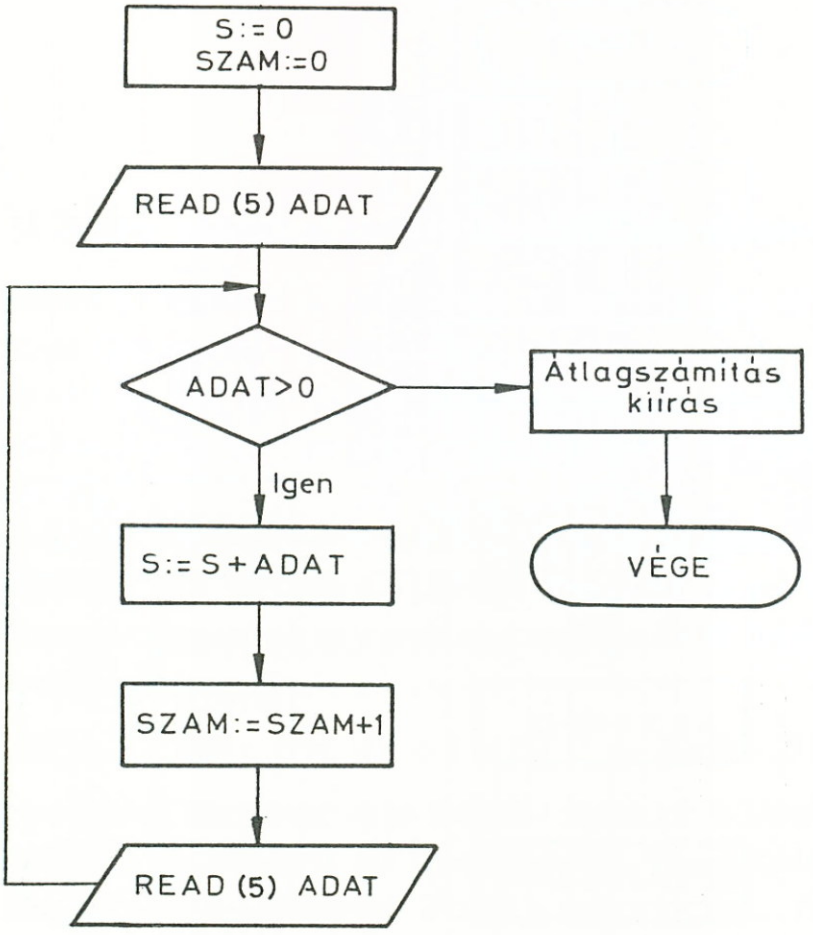
Finomítjuk a „van adat" problémakört és beépítjük az olvasási pontokat.



A -11. ábra. Második közelítés

Harmadik közelítés:

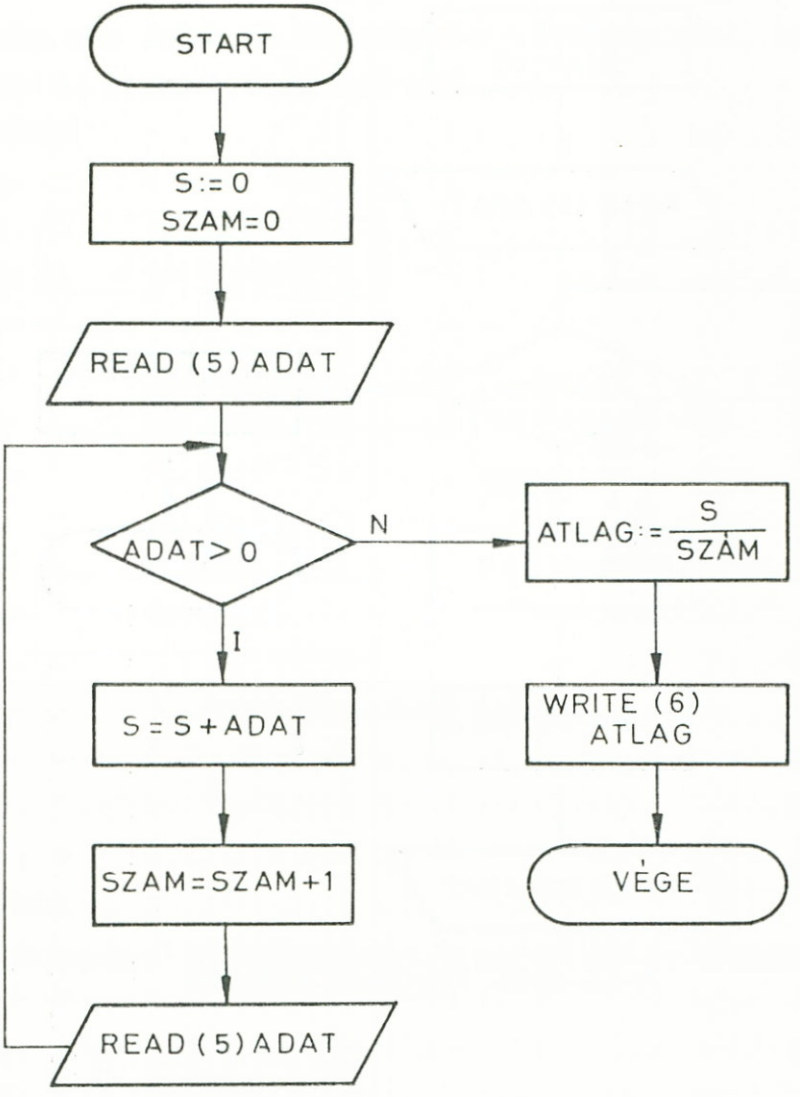
Finomítjuk a „feldolgozás" problémakört.



A-12. ábra. Harmadik közelítés

Negyedik közelítés:

Finomítjuk az „átlagszámítás és kiírás" problémakört.



A-13. ábra. Negyedik közelítés

A finomítás során mindig csak egy problémakörre koncentrálunk. Bonyolultabb feladatoknál elképzelhető, hogy egy-egy ilyen finomítás több közelítést is igényel.

IV. fejezet

1

A kettes számrendszerről tanultak összefoglalása

A köznapi életben a számokat általában tízes rendszerben ábrázoljuk. A tízes számrendszer pozíciós (helyértékes) rendszer.

Egy számrendszert akkor nevezünk pozíciósnak, ha a benne használt számjegy értéke változik aszerint, hogy a számjegy valamely számot előállító számjegysorozatban hol helyezkedik el.

A tízes számrendszer esetében egy adott helyérték minden egysége a nálánál közvetlenül kisebb helyérték tíz egységét foglalja magába. Egy tízes számrendszerbeli szám (pl. a 3785,1) nem más, mint egy polinom alak rövid formája. (Esetünkben ez a polinom alak a következő:

$$3\times10^3+7\times10^2+8\times10^1+5\times10^0+1\times10^{-1}=3000+700+80+5+0{,}1).$$

A bemutatott polinom alakban való felírási móddal könnyen felírhatunk számokat más számrendszerben is. Mielőtt ezt megtennénk, vegyük észre, hogy a tízes alapú számrendszerben tíz jel van értelmezve. Ezek a számjegyek. A tíz értelmezett számjegy: 0, 1, 2, 3, ..., 9. Általában is igaz, hogy p alapú számrendszerben p számjegy van értelmezve, és a legnagyobb értékű számjegy mindig a $p-1$.

Kettes számrendszerben az alapszám a 2, itt tehát két számjegy van értelmezve: a 0 és az 1.

A kettes számrendszerben az egyes helyértékeket elfoglaló bináris számjegyek (bitek) tényleges értékét úgy kapjuk meg, hogy az alaki értéket megszorozzuk kettőnek (a számrendszer alapszámának) a helyértékből adódó hatványával.

A 1100110,1 bitsorozat értékét, tízes számrendszerbeli megfelelőjét a következőképpen kell értelmeznünk az elmondottak szerint:

$$1\times2^6+1\times2^5+0\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0+1\times2^{-1}$$

vagyis

$$64+32+44+2+0{,}1=102{,}5.$$

A leírt egyszerű esetből is kitűnik, hogy valamely szám bináris alakja több helyértéket használ fel (hosszabb a jegysorozat), mint decimális megfelelője.

Egy kis megfontolás hozzásegít bennünket ahhoz, hogy a bináris és a decimális rendszerbeli helyértékszükségletre pontos arányt kapjunk.

Nyilvánvaló, hogy N számú bináris jeggyel ki tudjuk fejezni mindazokat a számokat, amelyek nagyságrendileg a 0 és 2^N—1 zárt intervallumba esnek.

Ugyanígy n decimális jeggyel mindazokat, amelyek 0 és 10^n-1 ugyancsak zárt intervallumban találhatók. Ha most az N bináris jeggyel ábrázolható legnagyobb szám pontosan akkora, mint az n decimális számjeggyel leírható legnagyobb, akkor írhatjuk, hogy

$$2^N-1=10^n-1.$$

Ebből pedig

$$N \cdot \lg 2=n,$$

ahonnan

$$N=\frac{n}{\lg 2}=\frac{n}{0{,}301}\sim 3{,}3\ n.$$

Ez azt jelenti, ugyanazon szám leírásához a bináris rendszerben átlagban 3,3-szor több jegy (helyérték) szükséges, mint a decimális rendszerben.

Bemutatunk két táblázatot. Az egyik 2 egész kitevős hatványait tartalmazza 0 és 20 között, a másik néhány tízes számrendszerbeli szám kettes számrendszerbeli megfelelőjét.

n	2^n	Decimális	Bináris
0	0	0	0
1	2	1	1
2	4	2	10
3	8	3	11
4	16	4	100
5	32	5	101
6	64	6	110
7	128	7	111
8	256	8	1 000
9	512	9	1 001
10	1 024	10	1 010
11	2 048	11	1 011
12	4 096	12	1 100
13	8 192	13	1 101
14	16 384	14	1 110
15	32 768	15	1 111
16	65 536	16	10 000
17	131 072	17	10 001
18	262 144	18	10 010
19	524 288	19	10 011
20	1 048 576	20	10 100

A táblázatból látszik, hogy a tízes számrendszerbeli egész számok kettes rendszerbeli megfelelői szintén egész számok.

Általában is igaz — anélkül, hogy matematikailag bizonyítanánk —, hogy bár-

mely számrendszerbeli egész szám egy másik számrendszerben pontosan ábrázolható. Egy törtszám esetében azonban ez nem áll fenn, törtszámok más számrendszerben való ábrázolása általában csak közelítő pontossággal lehetséges.

Műveletvégzés kettes számrendszerben

A műveletek itt is számjegyeken végzett műveletek. A szabályok azonban igen egyszerűek.

Mindenek előtt adjunk meg a műveletekhez szükséges számolási szabályokat:

Összeadás	Kivonás	Szorzás
$1+0=1$	$0-0=0$	$1\cdot 0=0$
$0+1=1$	$1-0=1$	$0\cdot 1=0$
$0+0=0$	$1-1=0$	$0\cdot 0=0$
$1+1=10$		
(1 átvitel)	$10-1=1$	$1\cdot 1=1$

Az ezek alapján végezhető műveleteket legegyszerűbb, ha olyan példákon keresztül mutatjuk be, ahol a művelet tízes számrendszerbeli megfelelőjét is láthatjuk:

a) Összeadás

17	10 001
+23	+10 111
40	101 000

Jobbról balra haladva $1+1$ egyenlő kettővel (10-val). Leírjuk a 0-át, maradt 1 (átvitel), amihez 1-et adva újra kettőt (10-et) kapunk; leírjuk a 0-t, megint lesz egy átvitel, amihez 1-et adva ismét 10-et és így egy átvitelt kapunk, amihez 0-át adva 1-et kapunk. A legnagyobb helyértékű két jegy összege $1+1$ újra 10-et ad, amit leírunk. A bináris összeg:

$$101\,000_2 = 1\cdot 2^5+0\cdot 2^4+1\cdot 2^3 = 40.$$

Tört esetén ugyanez az eljárás.

b) Kivonás

40	101 000
−23	−10 111
17	10 001

Ismét jobbról balra haladva: a 0-ból 1-et úgy vonunk ki, hogy a magasabb helyértékről vett egy egységgel a 0-át 10-re egészítjük ki. Így 10-ből (kettőből) 1-et levonva, az eredmény 1, és marad egy kivonandó átvitel, amit a kivonandó következő helyértékhez hozzáadunk, és az így kapott számot vonjuk ki a kisebbítendő soron következő jegyéből. És így tovább.

A kivonás még egyszerűbb, ha az ún. komplemens képzést alkalmazzuk. Ezáltal

kivonás helyett összeadást végezhetünk. Egy adott szám komplemense valamely számrendszerben a szóban forgó szám egyes jegyeit a számrendszer legnagyobb értékű jegyére kiegészítő jegyek sorozataként előállító szám. Ezek szerint pl.

3547 komplemense $9999-3547 = 6452$,
1001_2 komplemense $1111-1001 = 0110$.

Végezzük most el a

$$8765-3428$$

kivonást a következő módon:

Állítsuk elő a 3428-nak, tehát a kivonandónak a komplemensét. Ez 6571. Adjuk hozzá a kisebbítendőhöz:

$$\begin{array}{rr} & 8765 \\ & +6571 \\ \hline & 5336 \\ + & 1\ \ 1 \\ \hline & 5373 \end{array}$$

Az összeget adó szám legmagasabb helyiértékén keletkező (esetünkben bekeretezett) jegyet „vegyük el", és adjuk hozzá a legalacsonyabb helyértéken álló jegyhez. Az így kapott szám a kivonás eredménye.

Azt, hogy eljárásunk helyes eredményre vezet, könnyen beláthatjuk, ha végiggondoljuk, mit is csináltunk tulajdonképpen?

A kisebbítendőhöz hozzáadtuk a kivonandó komplemensét, majd a legnagyobb helyértékű jegyet elvettük (ezzel esetünkben nyilván 10 000-et vontunk le), és hozzáadtunk 1-et az így előállított számhoz. Azaz

$$8765+(9999-3428)-10\,000+1 = 8765+9999-3428-9999 = 8765-3428.$$

Nézzünk még egy kettes számrendszerbeli kivonást az előbbi komplemensképző módszerrel. Képezzük a

$$10\,110-10\,011$$

különbséget.

A 10 011-nek a komplemense $11\,111-10\,011=01100$. Írhatjuk tehát, hogy

$$\begin{array}{rr} & 10\,110 \\ & +01\,100 \\ \hline & 100\,010 \\ + & 1\ \ 1 \\ \hline & 11 \end{array}$$

a keresett különbség. Ugyanis:

$$10\,110+(11\,111-10\,011)-100\,000+1=$$
$$=10\,110+11\,111+10\,011-11\,111=10\,110-1\,0011.$$

Könnyen észrevehető, hogy a kettes számrendszerben valamely szám komplemense úgy képezhető, hogy a szóban forgó számot alkotó számjegy sorozatban az 1-esek helyett 0-kat, a 0-k helyett pedig 1-eseket írunk.

c) Szorzás

$5 \cdot 6 = 30$ a bináris rendszerben:

$$\begin{array}{r} 101 \cdot 110 \\ \hline 101 \\ 101 \\ 000 \\ \hline 11\,110 \\ \underbrace{} \\ 30 \end{array}$$

d) Osztás

$$\begin{array}{l|l} 221 : 23 = 9 & 11\,011\,101 : 10\,111 = \underbrace{1001}_{9} \\ 207 & -10\,111 \\ \hline 14 & 100\,101 \\ & -10\,111 \\ & \hline & 1\,110 \\ & \underbrace{} \\ & 14 \end{array}$$

Leválasztjuk az osztandó egy olyan részét, hogy az osztást el tudjuk kezdeni. Legegyszerűbb, ha levonjuk az osztót, a maradékot leírjuk, és a hányadosba 1-est írunk. A maradékhoz hozzávesszük az osztandó következő jegyét. Ha ebben az osztó megvan, tehát az osztó kivonható, kivonjuk, a maradékot leírjuk és a hányadosba ismét 1-est írunk. Ha nincs meg, a hányados soron következő helyértékére 0-t írunk és levesszük az osztandó soron következő jegyét stb.

Természetesen az itteni kivonások elvégzésénél is alkalmazhatjuk a komplemensképzés módszerét.

▲ 4

Az egész számok minden számrendszerben egzakt pontosságú egészekként jelennek meg.

▲ 5

A normált mantissza nagyságrendjét egy tízes számrendszerbeli számra mutatjuk be (a jobb megértés érdekében)

ha a mantissza értéke: 0,0386 — ez nem normált alak, mert a 10^{-1} helyértéken 0 áll;

ha a mantissza értéke: 0,513 — ez normált alak.

A legkisebb mantissza értéke nyilván a 0,100 ... forma vagyis éppen 1/10. A logaritmustáblázatban levő mantisszák „féllogaritmikus normált formában" vannak megadva, ott a kikötés az, hogy a mantissza 1-nél nagyobb és nem érheti el a 10-et.

● 2

Mindaddig, amíg a számítógépeket csak számolásra — tehát tisztán matematikai műveletekre visszavezethető feladatok megoldására — használták, a szót alkotó bitek száma számítógéptípusonként nagyon eltérő volt. Építettek olyan gépeket, ahol a szó 24 bites, de olyat is, ahol a szónak 40 bites tárolóelem felelt meg.

Az adatfeldolgozás megjelenésével kialakult és széles körben elterjedt az a társzerkezet, amely 8 bites, tehát bájtos alapelemekből áll. Az ilyen gépeket *bájt szervezésűnek* szokás nevezni. A továbbiakban csak ilyen felépítésű gépekkel foglalkozunk.

● 3

A kettes számrendszert összefoglaló annotációs részben (249—253. oldal) bemutatott komplemens az ún. egyes komplemens, ahol a „megfordított" értékekhez a +1 hozzáadása nem történik meg.

Az egyes komplemenssel végzett kivonásnál ezért a túlcsorduló bitet még hozzáadtuk a keletkezett eredményhez.

A kettes komplemensben való ábrázolás előnye éppen az, hogy ez már eleve hozzá van adva.

● 6

Ha pl. egy rekordban a születési évvel nem kívánunk matematikai műveletet végezni, csak azért tároljuk, hogy szükség esetén kiírásra kerüljön, célszerű karakter típusúnak kezelni.

● 7

Ha nyolc bit helyett négy bitet használunk minden számjegynél, ez azt jelentené, hogy pontosan feleannyi helyre lenne szükség tömörítés után. Ez azonban azért nem igaz, mert az előjel ábrázolásához továbbra is szükség van négy bitre, továbbá a tömörített formában ábrázolt számot is bájtokból felépülő szavak tárolják, így elképzelhető, hogy fél bájtnyi hely „kihasználatlan" marad.

Pénzforgalom — pénz nélkül
(Olvasmány)

„Az a tied, ami a zsebedben van" — tartja a mondás, arra utalva, hogy az ember csak azt a pénzt érezheti sajátjának, amit a kezében (zsebében) tarthat.

Talán ennek a mondásnak a túlságosan komolyan vétele okozta, hogy hazánkban az OTP által bevezetett lakossági csekk az első években a vártnál kisebb sikert aratott.

Úgy látszik szeretjük látni a pénzt, és idegenkedünk a pénzkímélő lehetőségektől. Ezért talán hihetetlennek is tűnik ez a történet, pedig már valóság.

Bérfizetéshez készülődnek egy több tízezer alkalmazottat foglalkoztató vállalatnál. A munka azonban nem a bérszámfejtésen folyik, hanem a számítóközpontban. Másnap a dolgozók rövid kimutatást kapnak, de pénzt nem. A kimutatás alapján nyomon követik, hogy a havi fizetésükből az adón kívül még mit vontak le (felvett hitelek törlesztése). Ezeket a pénzeket természetesen már át is utalták a megfelelő helyre. A megmaradt nettó összeg pedig a dolgozó bankszámláján került jóváírásra, annak a mágnesszalagnak az alapján, amelyet a vállalat a bankhoz átküldött. Ugyanekkor a bank a vállalat számláját megterhelte a kifizetett bérek összegével.

Hazafelé menet az egyik dolgozó megáll egy utcai automata előtt. Tárcájából kis mágneses lapot vesz elő, amit az automata nyílásába bedug, majd néhány gombot megnyom. Az automatából pénz esik ki. A mágneses kártyán levő azonosító és a gombnyomással közölt titkos jelszó alapján az automatában levő számítógép azonosította az ügyfelet, majd telefonvonalon lekérdezte a banki géptől, hogy van-e fedezet a kifizetésre. Ezt követően a bankszámlából levonta a beütött összeget, a pénzt pedig kiadta.

Egy másik dolgozó hazafelé menet betért a közeli élelmiszeráruházba, történetesen éppen abba, amelyről az első fejezetben már szó volt. A pénztárnál csekket vesz elő, kitölti és átadja a pénztárosnak. A csekken mágnesesen felvitt számsor látható, ennek segítségével a pénztáros röviden ellenőrizheti, hogy a csekknek van-e fedezete.

A mágneses jelekkel helyettesített pénzt túl azon, hogy nem hamisítható, nagyban lecsökkenti a forgalom lebonyolításához szükséges pénz mennyiségét. Minden állam sokat költ a bankjegyek elhasználódás miatti újranyomásra.

V. fejezet

▲ 4

Képzeljetek el egy lakótelepi házat, amelyben száz lakás van. A lakásokat sorszámmal látták el, van 1-es lakás, 15-ös lakás, 100-as lakás. Az egyes lakásokat tehát egy-egy sorszám azonosítja. Egy új postás — aki nem tudja, hogy melyik lakásban ki lakik — a lakásszám alapján találja meg a címzettet. A központi tár címei is sorszámok, de kettes számrendszerben. Tizenhatos számrendszerbe való átszámításuk — a IV. fejezetben tanultak szerint — nagyon egyszerű.

▲ 5

Ha az előbbi ház 93-as lakásából valaki elköltözik, és új lakó érkezik, a lakás „tartalma" megváltozik. Eddig „Kovácsék" volt a tartalom, mostantól „Gáspárék" az új tartalom. A lakás sorszáma azonban — függetlenül a bent lakóktól — továbbra is 93.

Tegyük fel, hogy most egy levél érkezik, amire az utca nevén és a házszámon kívül ez áll: 93. lakás. Név nincs a borítékon. A postás a levelet kézbesíteni fogja, miközben nem tudja (nem is érdekli), hogy a valódi címzett vajon Kovács vagy Gáspár. A levél megtalálja a címet.

▲ 6

Egy bájtnyi tároláskor a hossz 1, félszóban való tároláskor a hossz 2, szóban való tároláskor a hossz 4, duplaszóban való ábrázoláskor a hossz 8.

▲ 9

Ha blokkdiagramban felírjuk C:=A+B értékadást, ebben a műveletben a következő információk vannak:

1. Összeadást kell végezni.
2. Az összeadás első tagja (operandusa) az A szimbolikus címen található.

3. A második tag a B szimbolikus címen található.

4. Az összeget (eredményt) a C szimbolikus címre kell elhelyezni.

5. Miután ez a művelet egy program része, ezért vannak előzményei és következményei.

Az utasításoknak ezt az egymásutániságát biztosítja az, hogy mindig tudni kell a soron következő utasítás helyét (címét) is.

▲ 10

A 0, 1, 2, 3 bináris alakja
0000
0001
0010
0011, tehát az első két bit 00.

A 4, 5, 6, 7 bináris alakja
0100
0101
0110
0111, tehát az első két bit 01.

A 8 és 9 bináris alakja
1000
1001, tehát az első két bit 10.

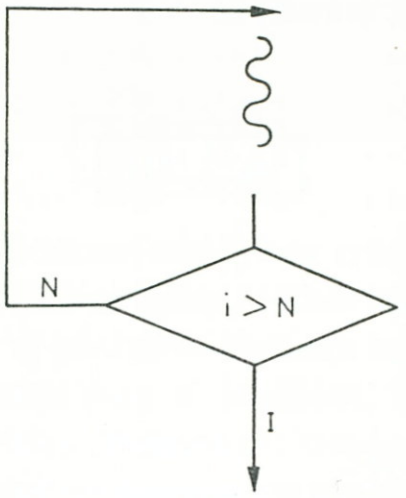
A D és F bináris alakja
1101
1111, tehát az első két bit 11.

▲ 12

Ha azt a megoldást választanánk, hogy az utasításrendszert a központi tár méretétől tennénk függővé, lehetetlen állapot állna elő, mondjuk olyan esetben, amikor egy gépnél tárbővítést akarnak megvalósítani. A tárbővítés során ui. a meglevő tárhoz újabb tárolóelemeket illesztenek, s ezzel a meglevő rendszert pl. 64 vagy 128 Kbájttal megnövelik. Gyakori megoldás ez, pl. azért, mert a gép vásárlásakor a vállalatnak nem volt elég pénze, ugyanakkor a munkák mennyisége nem is kívánta meg a nagy tárméretet. Később a vállalat „kinőtte" a gépet, bővíteni kellett.

▲ 15

Tekintsük a következő blokkdiagram részletet, amely egy ciklus végét szemlélteti.



A-14. ábra. Vezérlésátadás blokkdiagramban

Abban az esetben, amikor a döntés hamis értéket ad, tehát a feltétel nem teljesül, a vezérlés visszakerül a ciklus elejére, és nem a soron következő utasításra. Ilyenkor programvégrehajtásnál vezérlésátadás történik.

▲ 16

Az elmondottak szemléltetésére vegyük a következő példát. Egy ciklusban a ciklusmag végrehajtása után a ciklusváltozó értéke megnövekedett a lépésközzel. Most egy feltételes vezérlésátadás következik. Ha a ciklusváltozó új értéke meghaladja a ciklus végértékét, be kell fejezni a ciklust (tovább kell lépni a soron következő utasításra). Ha azonban az $i \leq n$ feltétel még teljesül, vissza kell ugrani egy korábbi tárcímre, oda ahol a ciklusmag kezdődik. Az utasításszámláló tartalma tehát az első esetben másképpen változik, mint a második esetben. Az első esetben az épp végrehajtott utasítás hosszának megfelelő értékkel nő, a másik esetben egy kiszámított cím kerül bele.

● 1

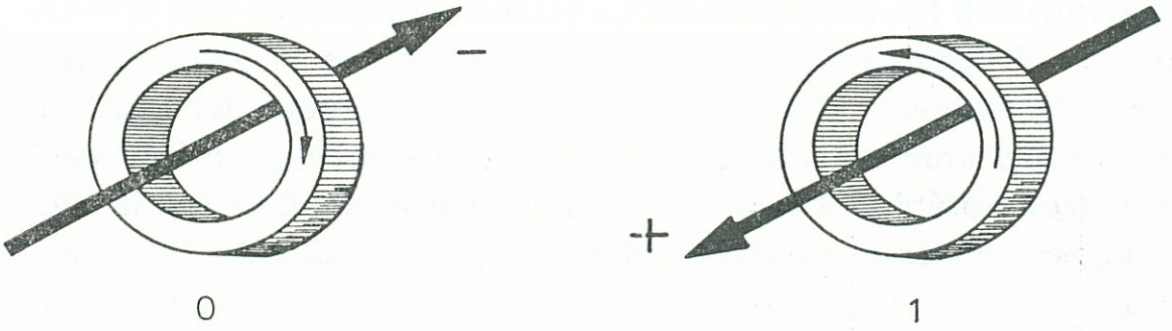
A számítástechnika négy évtizedes fejlődésének voltak látványos és kevésbé látványos eredményei. Kétségtelen, hogy a külső szemlélő számára a leglátványosabb a hardver terjedelmének rohamos csökkenése volt.

A generációkat egymástól megkülönböztető ismérvek között is első helyen áll a hardver megvalósítás technológiája. Ugyanakkor megállapítható, hogy az új hardver technológiák megjelenése (pl. az integrált áramkör) megteremtették a lehetőségét annak, hogy újabb hardver elemek (pl. a csatorna) jelenhettek meg. Ezt a bonyolult egymásrahatást tovább bonyolítja, hogy maga a hardver és a mindenkori szoftver is állandó kölcsönhatásban van egymással.

Nem járjuk végig a hardver fejlődés állomásait, hanem egy mai értelemben vett korszerű számítógép hardverjével ismerkedünk meg.

2

A bináris számok fizikai tárolásának megvalósítása szemléletesen látszik a *ferritgyűrűs tárak* esetében. A ferritgyűrűs (ferritmagos) tárolás elve azon alapszik, hogy egy ferritből (vasoxid és kerámiaanyag keverékéből) készült gyűrű az elektromos áram hatására kétféle irányban mágnesezhető.



A-15. ábra. A ferritgyűrű mágnesezettségi állapotai

A kétféle mágnesezettségi állapot a gyűrűn átvezetett áram irányától függ. A ferritgyűrű mágnesezésével kapcsolatban a következő követelmények merültek fel:

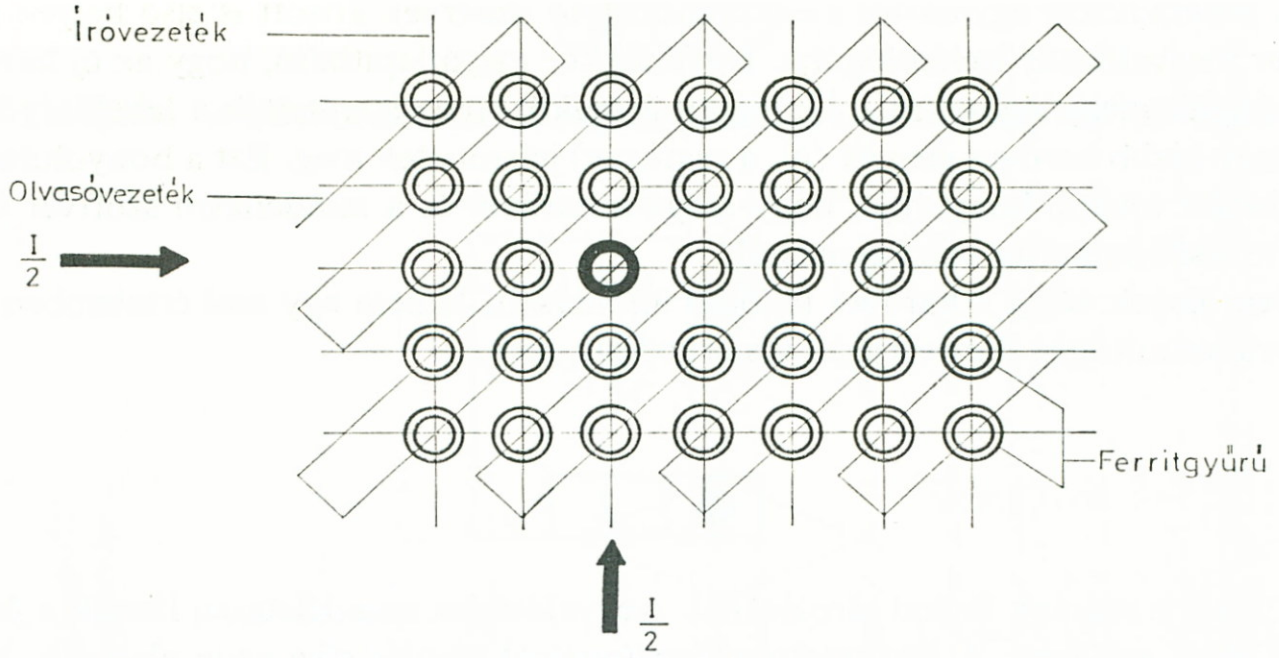
— a mágnesezettségi állapot csak meghatározott nagyságú I áram hatására változzon;

— tetszőlegesen sokszor legyen mágnesezhető;

— a tárolt bit kiolvasásához elégséges remanens mágnesességgel rendelkezzen.

A ferritgyűrűs tároló a felsorolt követelményeket kielégítő, 1-2 mm átmérőjű gyűrűkből tevődik össze. A kívánt mágnesezettségi állapot előállíthatósága végett a ferritgyűrűk egy vezetékrendszerből álló rács rácspontjain helyezkedtek el. Ily módon minden gyűrűn egy „függőleges" és egy „vízszintes" vezeték halad át. (A függőleges és a vízszintes kifejezés itt szemléltető jellegű, relatív irányokat jelent.)

Ahhoz, hogy egy gyűrű mágneses állapotba kerüljön, legyen szükséges I erősségű áram. Ha a kiválasztott ferritgyűrűhöz tartozó két vezetékre egyenként I/2 erősségű áramot adunk, a két vezeték találkozási pontjában (a kívánt gyűrűnél) I erősségű áram fog áthaladni. Így az áthaladó áram hatására ez az egy gyűrű 1-es állapotba kerül (A-16. ábra). Ily módon egy bitet tárolunk.



A-16. ábra. A ferritgyűrűs tároló tárolósíkja

Azért, hogy a ferritgyűrű tartalmát el is lehessen olvasni, egy harmadik vezeték is haladt a ferritgyűrűk között. Ezt neveztük olvasó vezetéknek.

Az olvasás elvileg úgy történt, hogy a két írővezetékre „negatív irányú" és egyenként I/2 erősségű áramot adunk. Ennek hatására a kérdéses ferritgyűrűn „negatív irányú" I erősségű áram haladt át, amely a gyűrű mágnesezettségét megfordította, amennyiben azon korábban I áram haladt át (vagyis a gyűrű az 1-et tárolta). A megfordított mágnesezettség az olvasó vezetékben áramot indukált. Ha ez az áram egy meghatározott értéknél nagyobb, ez azt jelentette, hogy a ferritgyűrűben az 1 volt tárolva, amennyiben ez az áram kisebb, ez azt jelenti, hogy nem fordult meg a mágnesezettség iránya, tehát korábban, a beíráskor is negatív irányú I áram haladt át, vagyis a tárolt bit 0 volt.

Az elmondottakból következik, hogy a tár tartalmának „olvasása" elrontotta a bitek értékét. Mivel ez megengedhetetlen (hiszen ez olyan, mintha egy magnetofonszalag lejátszása egyben a szalag törlését is jelentené) a ferritgyűrűs tárat regenerálni kellett, vagyis eredeti állapotába vissza kellett állítani.

Az ilyen tárakat *destruktív tároknak* nevezték. A ferritgyűrűs tárak először a második generációs gépeknél jelentek meg, és a harmadik generációs gépek első sorozata is ferrites tárakkal épült.

1978 óta van lehetőség arra, hogy a tárakat félvezetőkből építsék. Ez lényegesen csökkentette a központi tár fizikai terjedelmét és növelte a megbízhatóságot.

● 3

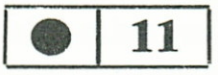
A félvezetős technológia megjelenése lehetővé tette, hogy a számítógépek méretcsökkenése a központi tárra vonatkozóan is bekövetkezhessék. A ferritgyűrűs tárak

esetében pl. 512 Kbájtnyi tár elhelyezésére egy kb. 170 cm magas, 160 cm hosszú, és 75 cm széles szekrényre volt szükség. Ugyanekkora tárkapacitás az újabb technológia szerint egyetlen áramköri lapon elfér, amelynek mérete 30×10 cm körül van.

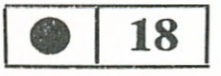
A központi tár és a központi egység, amely korábban több nagyméretű szekrényből állt, egy vagy két karcsú szekrényben elfér, miközben a szekrényeknek a fele nincs is kitöltve, későbbi fejlesztések számára általában üresen hagyják.

A BCD kóddal működő, alapvetően tudományos számításokra épített gépek *szószervezésűek*. A szóhossz általában 24 bit.

Ezeknél a gépeknél a legkisebb címezhető táregység a 24 bites szó.



Persze azt is mondhatná valaki, hogy akkor a központi tár méretétől függetlenül a maximális tárméretthez igazodjék az utasítás címrészeinek hossza. Előző példánk szerint mindig 20 bit legyen, legfeljebb nincs rá szükség. Ez azonban azt jelentené, hogy kisebb kiépítettség esetén a program — azért, mert minden utasítása felesleges (üres) biteket tartalmaz — lényegesen több helyet foglal el a tárban, mint szükséges lenne. Ez pedig pazarlás az amúgy is szűk erőforrással a központi tárral.



A számítógép az orvosi diagnosztikában
(Olvasmány)

A betegek diagnosztizálása és kezelése az egészségügyi ellátás fő funkciója. Az orvostudomány mai állása szerint kb. 30 ezer betegségből kell az orvosnak kiválasztani azt az egyet vagy egynéhányat, amely a vizsgált betegnél megtalálható. A klinikai döntés legsarkalatosabb pontja, hogy az orvosnak nagyon sok adatot kell értékelnie. A kórelőzmény adatai, a tünetek és jelek, a rutinszerűen kivitelezett laboratóriumi és egyéb vizsgálatok eredményei által meghatározott számos lehetőség közül kell választania és kialakítania a beteg állapotára vonatkozó hipotéziseket. Sok száz kórtörténet kiértékelése alapján kiderült, hogy a tünetek száma egy-egy betegségnél 3...40, de a 80...100-at is elérheti.

Világszerte folynak a kísérletek azzal a céllal, hogy a számítógép lehetőségeit a diagnosztizálás, a gyógykezelés területén minél hatásosabban tudja az orvos közvetlenül a beteg érdekében hasznosítani.

Az orvosi döntéshozatal számítógépes támogatásának területén az utóbbi években több, egymástól alapvetően eltérő kezdeményezés bontakozott ki. Ezek közül mutatunk be néhányat.

Az egyik módszer az orvosi döntést úgy segíti, hogy korlátozott számú, ismert, jól definiált betegség közül a vizsgált betegnél megállapított betegségtünetek alapján kiválasztja az objektíve legvalószínűbb diagnózist. A módszer feltételezi, hogy a tünetek valószínűségi paramétere, a betegségek gyakorisága ismert, hogy a tünet-valószínűségek függetlenek egymástól, valamint azt, hogy az adott beteg csak számítógépben nyilvántartott betegségek egyikében szenved.

Az előző módszerhez igen közel áll az a programrendszer, amely mint megbízható konzultáns segíti az orvost bármilyen diagnosztikai vagy terápiai döntésben, a diagnosztikai következtetések folyamatát és a terápia stratégiák értékelését becslési, számítási feladattá redukálva. Az ismert tünetek, vizsgálati eredmények és a kórelőzmények alapján a program kiválasztja a szóbajöhető betegségeket és valószínűségi megoszlásuk alapján rangsorolja őket, javaslatot tesz esetleges további vizsgálatokra, ill. a megfelelő gyógykezelésre.

Egy harmadik módszer úgy igyekszik segíteni az orvos önálló diagnosztikai munkáját, hogy az adott betegségre vonatkozó jellemző kórképet bocsátja az orvos rendelkezésére. Ez a módszer főként olyan betegségek esetén hatékony, amelyek tünetei nem feltétlenül ismertek az orvos előtt.

Példaként egy gyakorlati megvalósításra, ismertetjük a MEDIAC elnevezésű orvosdiagnosztikai számítógépes rendszer működését:

A rendszer egy központi számítógépen alapszik. Az orvos munkahelyéről telefonon kérhet speciális orvosi szempontok szerinti diagnózist.

A megfelelő telefonszámot tárcsázva az orvos közvetlenül összeköttetésbe kerül a géppel. A beteg tüneteit, a vizsgálati eredményeket és minden egyéb információt telefontárcsa segítségével, szám- és betűkódok formájában továbbít a géphez. A telefon fülhallgatóján keresztül a gép hallható beszédformában visszaolvassa a betárcsázott adatokat, így az orvos ellenőrizheti, hogy a tárcsa útján közölt adatokat valóban jól küldte-e el a számítógépnek. Bármely tévedést az orvos újratárcsázással korrigálhat. Az eredmény néhány valószínűsíthető diagnózis, azok valószínűségi sorrendjében és egyben további kiegészítő vizsgálatok ajánlása. Ezeket az eredményeket 30 másodpercen belül kapja meg az orvos, mégpedig élőszóban, a telefon fülhallgatóján át. A szöveg ismétlése mindaddig tart, amíg az orvos le nem teszi a telefont.

Az orvos és a gép között egy speciális input-output egység közvetít. Feladata, hogy az orvos tárcsázott üzenetét a gép nyelvére kódolja, ill. a gép válaszait hangokká alakítsa. A feldolgozó egység értékeli a beteg tüneteit, és ezek alapján az adott esetben legvalószínűbb öt betegséget határozza meg, azok valószínűségi sorrendjében, valamint jelzi a további vizsgálati ajánlatokat.

Az orvos a telefontárcsával először egy speciális kódjelet tárcsáz, amellyel jelzi, hogy megkezdi a tünetek közlését. Minden tünetet háromjegyű kód képvisel, és egy újabb speciális kódjel tárcsázandó a befejezés után. A gép 128 tünetet képes kezelni.

A feldolgozó egység összehasonlítja az adott, betáplált tünetegyüttest mind a 2560 általa tárolt tünetprofillal, és kiválasztja az öt legvalószínűbb betegséget, azon az alapon, hogy azok tünetprofilja mennyire hasonlít az általa tároltakhoz, egyúttal számításba veszi az egyes tünetcsoportok mérlegelésével kapcsolatos szempontokat is.

A kimenő üzenet úgy fogalmazódik meg, hogy az előre, végtelenített magnószalagra felvett szavak közül a megfelelőeket a gép kiválasztja és a kívánt sorrendben átjátssza azokat egy output magnószalagra. Ezt követően a szalagon levő szöveget mindaddig ismétli a telefonba, amíg a kagylót le nem teszik.

A számítógép tárkapacitása lehetővé teszi, hogy az ismert betegségek összes jellemzőikkel együtt tárolhatók legyenek és egy konkrét esetben a beteg problémáit értékelve, a szóba jöhető betegségek mindegyikét az orvos rendelkezésére bocsássa.

A számítógépes és a hagyományos diagnosztika alapjaikban különböznek egymástól. Ha a számítógép felhasználásra is kerül a klinikai diagnosztikában, akkor sem helyettesítheti az orvost, mindössze segítheti a gyógyító munkát. Egyes jövőkutatók véleménye szerint a számítógép belátható időn belül az orvosi munkának ugyanolyan megszokott segédeszközévé válik, mint manapság a sztetoszkóp.

⊙	7

Neumann János a tárolt program elvének előnyeként említette, hogy a programozónak lehetősége van a program utasításait adatként kezelni, és ily módon az utasításokat megváltoztatni. Ez a ciklus végrehajtás kezdeti megoldása volt.

Az utasítás megváltoztatása azonban egyben a program változását is jelentette, ami nagyon sok veszélyt is rejtett magában.

Ezért a programozási nyelvek kialakításakor már olyan ciklusszervezési módokat találtak ki, amelyek magát a programot nem változtatják meg.

Az ilyen programokat *tiszta programnak* nevezik, szemben a *nem tiszta programokkal*, amelyek önmagukat képesek megváltoztatni.

⊙	13

A tárbeli cím kiszámítása a valóságban a következő módon történik. A B-ben megadott bázisregiszter hátsó 24 bitjén levő, pozitívnak tekintett számhoz hozzáadódik az X-ben meghatározott regiszter hátsó 24 bitje (ha erre szükség van, egyébként ez az összeadás elmarad). Az így kapott érték hátsó 12 bitjéhez a D-ben levő érték kerül hozzáadásra. A tényleges tárcím az így kapott érték hátsó 24 bitje.

⊙	14

A fix tárakat szokás *ROM* táraknak nevezni, az angol Read Only Memory *(csak olvasható tár)* kifejezés rövidítéséből.

Ma már különböző fix tárak léteznek. A már említett típusú ROM táron kívül (vagyis amelybe a program úgy kerül beégetésre, hogy azon változtatni már nem lehet) megjelentek a *programozható fix tárak (PROM)*, amelyek lényege, hogy gyártáskor üres, és (pl. a felhasználó igénye szerint) mód van rá, hogy abba egy programot utólag égessenek be.

Az *EPROM* tárak *(törölhető és újraprogramozható fixtárak)* esetében speciális módon, ultraibolya fénnyel a beégetett program törölhető és helyére másik program égethető be.

A ROM tárakkal szemben a hagyományosan használható tárakat (pl. a központi tárat) szokás *RAM* tárnak nevezni, amely elnevezés a Random Access Memory *(véletlen elérésű tár)* kifejezés rövidítéséből ered.

⊙	17

Bizonyos szituációban a programozó szeretné, ha egy esemény bekövetkezése nem okozna megszakítást.

Ilyen eset pl., amikor egy program kipróbálás alatt áll, és a programozónak nincs kellő számú adata a program kipróbálásához, így bizonyos osztások esetében az osztó 0 értékkel rendelkezik. A programozó szeretné, ha ilyen esetekben a 0-val való osztás nem okozna megszakítást. Erre lehetőség is adódik. Az ilyet úgy szokás mondani, hogy maszkolják (célszerűen választott bitekkel felülírják) azt a területet, amelyből a megszakítás kiderülne.

Természetesen nem szabad minden okot maszkolni, mert ez nagyobb hibák forrása lehet.

VI. fejezet

▲ 3

Az első eset azért esik ki, mert a számítógépet valamilyen emberi nyelvre (magyarra, oroszra, angolra) megtanítani az elmúlt évtizedekben nem lehetett. Bár jelenleg ilyen irányú kísérletek már folynak, eredmény csak korlátozottan várható.

A második eset éppen a gépi kódban való programozás lenne, ez az a mód, amit ki akarunk küszöbölni.

A tolmácsolás olyan áttételeket jelent, amelyet a negyedik módszer ki tud küszöbölni.

A számítógép „megtanítása" egy általa nem tudott nyelvi struktúrára, ilyen célú programokkal lehetséges, erről szól a fejezet.

▲ 8

Azonos funkció = valamennyi programozási nyelv funkciója, hogy megkönnyítse a számítógép programozását.

▲ 9

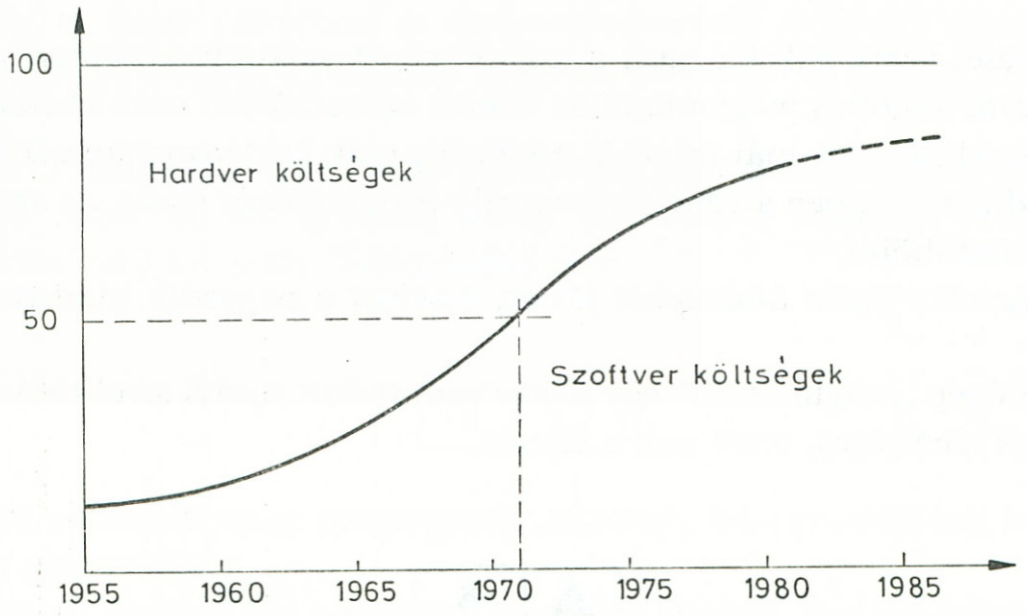
Nem végrehajtható utasítás pl. a tömbök méretének kijelölését végző utasítás, vagy a változók típusát megadó utasítás. Ezek a fordításkor játszanak szerepet, informálják a fordítóprogramot. A végrehajtáskor már nincs szerepük. Innen az elnevezésük is.

▲ 16

A fordítóprogram azért nem adhat rögtön végleges címeket (azért nem tölthet értéket a bázisregiszterekbe), mert a fordítás idején nem lehet előre tudni, hogy a futtatáskor a tár mely részei állnak rendelkezésre.

A szoftver aránya a számítógépen belül

Az elmúlt évtizedekben figyelemreméltó változáson ment keresztül a hardver és a szoftver egymáshoz viszonyított aránya, amely a számítógépre vonatkoztatott költségek nagyságrendjén keresztül vizsgálható. Az A-17. ábra a számítógép összes költségeit 100-nak tekintve a hardver és szoftver költségek részarányának alakulását szemlélteti az elmúlt évtizedekben.



A-17. ábra. Hardver és szoftver költségek arányváltozása

A költségarányok megfordulását két tényező alakította. Egy oldalról a tranzisztorok, félvezetők, integrált áramkörök megjelenése és beépülése a hardverbe csökkentette a hardver árát, miközben a teljesítőképesség és a megbízhatóság növekedett. Más oldalról abszolút értelemben is megnőtt a géphez tartozó szoftver mennyisége. Így állt elő az a furcsa állapot, hogy a számítógéppel — mint hardverrel — szállított „ingyen" szoftver (rendszerprogramok és egyes esetekben alkalmazási programok is) a vételár legalább 50%-át jelentik.

Emellett az egyes számítóközpontok jelentős összegeket fordítanak az új alkalmazási programok kidolgozására, és ugyancsak nem csekély összegekbe kerül a meglevő szoftver karbantartása (felfedezett hibák korrigálása).

A szoftver abszolút növekedését a következő példa is szemlélteti. Az IBM által a hatvanas évek legelején fejlesztett 704-es gép teljes szoftverjét 40 000 utasításra becsülték. Az évtized közepén (1964-ben) bejelentett 7090 típus szoftverje mintegy 600 000 utasításból állt, míg a 360-as gépcsalád, a hatvanas évtized végén már kb. 6 millió utasításos szoftvert tartalmazott.

A példában a szoftver mennyiségének növekedése egy évtized alatt mintegy 150-szeres. Ekkora, és ehhez hasonló méretű növekedések tartalmukat tekintve négy jól elhatárolható és nem is egy időben lezajló irányt jelentenek.

Az első időben is legkorábbi fejlődési irány a magas szintű programozási nyelvek kialakítása. Viszonylag hamar kiderült, hogy amennyiben a számítógépeket széles körűen akarják alkalmazni, túl kell lépni a gépi kódban való programozáson, ami speciális felkészültséget kíván meg a programozótól. Így alakultak ki a magas szintű programozási nyelvek. Ezzel közel egy időben felmerült az igény arra, hogy a számítógép munkájának felügyeletét és koordinálását automatizálják, azaz magával a számítógéppel végeztessék el. Az állandó emberi beavatkozás ui. nagyon lecsökkentette a gép teljesítőképességét.

Ebből az igényből jött létre a szoftver fejlesztés másik iránya, ami végeredményben a vezérlőprogramok kialakulásához vezetett.

A harmadik irány a tipizált alkalmazási programok fejlesztése. Időben később, kb. akkor, amikor a vezérlőprogramokkal és nyelvekkel felszerelt gépekre az egyes felhasználók nem tudtak nagyméretű, de ugyanakkor hatékony rendszereket készíteni. Ennek a szoftver fejlesztési iránynak az eredményeképpen jöttek létre a legkülönbözőbb célú programcsomagok. Egy idő múlva kiderült, hogy a programcsomagok nem „mindenhatóak". Alkalmazásuk sokszor akadályokba ütközött, mert egy-egy konkrét területre csak jelentős változtatásokkal alkalmazhatók. Ennek is szerepe volt abban, hogy a szoftver fejlesztők már nem kész programot kívántak a felhasználónak nyújtani, hanem olyan eszközöket (nyelveket), amelyek segítségével egy szakterület vagy gazdasági terület számítógépes modellezése hatékonyan megoldható. Így születtek meg a speciális célú nyelvek. Ezek a nyelvek — a magas szintű programozási nyelvekkel ellentétben — nem általános célúak, tehát alkalmazási körük lényegesen szűkebb, de az adott területen nagy hatékonysággal alkalmazhatók.

A szoftver fejlesztés legújabb iránya az automatizált programozást irányozza, vagyis olyan rendszerkomponensek kialakítását, amelyek az input és output adatstruktúrákból és néhány szükséges ismeretből automatikusan elkészítik a megoldóprogramot.

2

Számítógép — „lélek nélkül"

A címbeli „léleknélküliség" kifejezés arra kíván utalni, hogy kiinduló példánk csak „testből", vagyis hardverből álló gépet feltételez. Természetesen ma már a gyakorlatban (szerencsére) nincs üres hardver, ezért mind a feltételezett gép, mind a bemutatott gépi kód kitalált.

A fiktív kód, amelyet bemutatunk, kétcímű. Tudjuk, hogy az ilyen utasítások általános alakja a következő:

műveleti kód	1 operandus címe	2 operandus címe

Egy-egy ilyen szerkezetű utasítás a központi tár egy-egy szavában egy-egy címen helyezkedik el. Ugyanitt helyezkednek el az adatok is, amelyek a műveletek operandusai lesznek.

A következő utasításokat értelmezzük:

01 összeadás	06 feltétel nélküli ugrás
02 kivonás	07 feltételes ugrás
03 szorzás	10 bevitel
04 osztás	11 nyomtatás
05 átvitel	12 megállás

(A műveleti kódokat nyolcas számrendszerben adtuk meg.)
Az egyes műveletek értelmezése legyen a következő:

01 A B hatására az A cím és a B cím tartalma *összeadódik*, és az eredmény a B címen képződik.

02 A B hatására az A cím tartalmából *kivonja* a B cím tartalmát, és a különbség a B címre íródik be.

03 A B hatására az A cím tartalmát *megszorozza* a B cím tartalmával, és a szorzat a B címen képződik

04 A B hatására az A cím tartalmát *osztja* a B cím tartalmával, és a hányados a B címen képződik.

05 A B hatására az A cím tartalmát *átírja* a B címre. (Ez az utasítás látja el az értékadás funkcióját is.)

06 A B hatására a *vezérlés* az A címen található utasításra *adódik át*. (Ennél az utasításnál teljesen mindegy, hogy a második címre mit írunk az utasítás végrehajtását nem befolyásolja.)

07 A B *Amennyiben* az előző művelet eredménye *nem negatív* volt, ennek az utasításnak a hatására a *vezérlés* az A címen található utasításra *adódik át*. Ellenkező esetben a vezérlés a B címen található utasítást kapja.

10 A O hatására az olvasóban soron következő adatot *beolvassa*, és az A címen helyezi el.

11 O B hatására a B című rekesz tartalma *nyomtatásra* kerül.

12 O O hatására a gép *befejezi* a program végrehajtását.

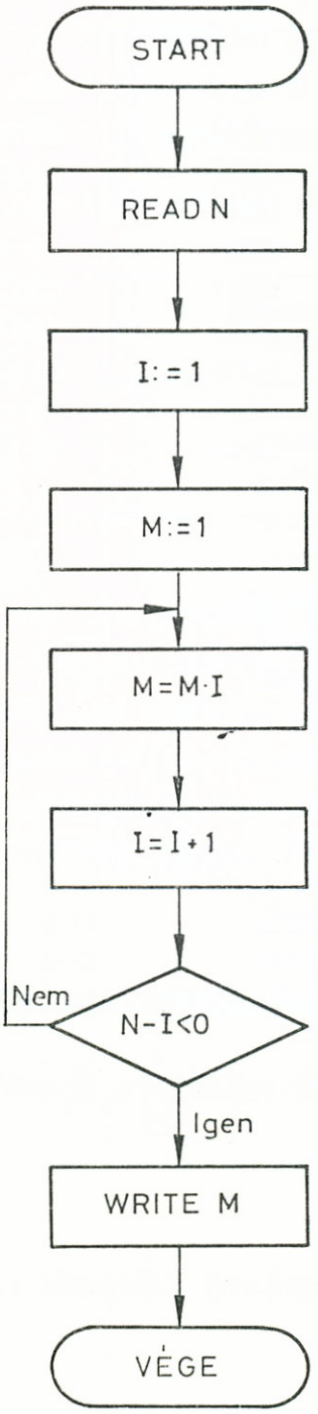
A tiszta gépi kód nem ismeri a szimbolikus neveket, az egyes adatokra csak tárcímeken keresztül lehet hivatkozni. Ezért szükséges, hogy a programozó elkészítse a tároló felosztását, azaz meghatározza, hogy mely adatai mely címeken kerüljenek elhelyezésre.

Ugyanebből a meggondolásból előre rögzíteni kell, hogy maga a program melyik címtől kezdődően kerüljön betöltésre.

Az elmondottakat kísérjük nyomon a következő példán.

1. példa. Készítsünk gépi kódú programot, amely előállítja az M=n! értéket. (Az n! jelölés az első n természetes szám szorzatát jelenti.)

A feladat input adata: n.
A feladat output adata: M.
A megoldásban szerepel egy konstans: 1, valamint egy munkarekesz is. A program áttekintése előtt vizsgáljuk meg az A-18. ábrán levő blokkdiagramot, amely a megoldás vezérlési szerkezetét írja le.



A-18. ábra. Az n! kiszámításának blokkdiagramja

Tárbeosztás: (A fordítóprogram egymás utáni címeken helyezi el a változókat és a konstansokat. Itt ettől eltérő beosztás következik.)

Konstans a 0220 címen 1
A változók helye: a 0200 címen n
a 0201 címen i

a 0300 címen M (eredmény)
a 0202 címen egy munkaterület.

A programot a 0100 címtől kezdődően helyezzük el.

Utasítás címe	Műveleti kód	Első cím	Második cím	Megjegyzés
0100	10	0200	0000	n beolvasása
0101	05	0220	0201	i: = 1
0102	05	0220	0300	M: = 1
0103	03	0201	0300	M: = i · M
0104	01	0220	0201	i = i+1
0105	05	0201	0202	i munkarekeszbe
0106	02	0200	0202	n−i < 0 ?
0107	07	0103	0110	feltételes ugrás
0110	11	0000	0300	M kiírása
0112	12	0000	0000	STOP

A munkarekeszre azért van szükségünk, mert abban képződik a különbség, amelynek alapján eldöntjük, hogy i értéke túllépte-e már n értékét. Ha a 0201 rekeszt használnánk fel közvetlenül (ebben tároljuk i-t), akkor — mivel a kivonás eredménye a második címen képződik — i aktuális értéke állandóan elromlana.

A 0107 rekeszben levő utasítás jól példázza a címke szerepét. Az előző művelet eredményétől függően vagy a 0110 címkéjű utasításra kell ugrani, vagy a 0103 címkéjűre.

2. példa. Készítsünk gépi kódú programot, amely előállítja és kinyomtatja az

$$S=\sum_{i=1}^{4} a_i b_i$$

szorzatösszeget.

(A $\sum_{i=1}^{4} a_i b_i$ szorzatösszeg a következő kifejezés rövidebb formában való felírása:

$a_1b_1+a_2b_2+a_3b_3+a_4b_4$.)

A feladat input adatai: a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4,
valamint a következő konstansok: 0, 1, 4.
A feladat output adata: S.
(Az előző feladattal szemben itt a megoldáshoz szükséges konstansokat is külön vigyük be a tárolóba. Az adatok a következő sorrendben legyenek rögzítve: 0, 1, 4, a_1, b_1, ... a_4, b_4.)

Tárbeosztás:

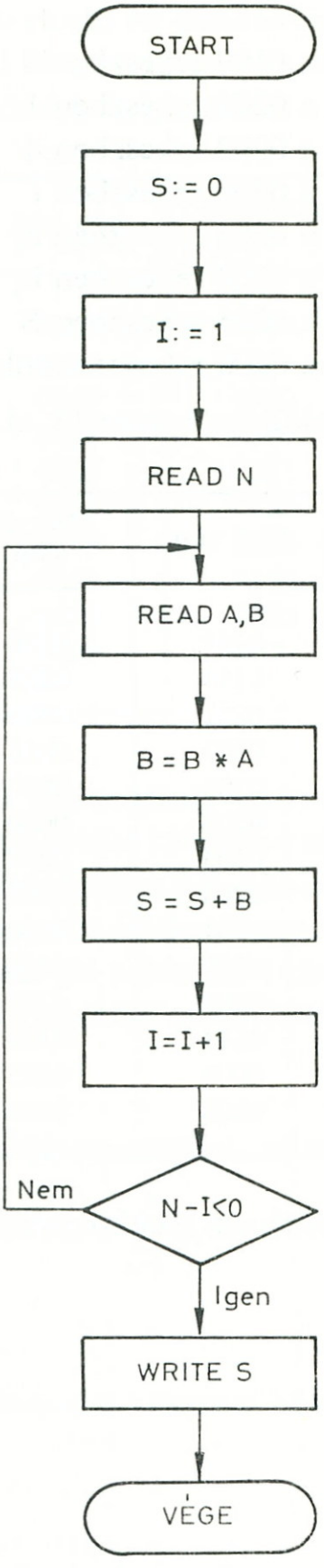
Állandók (konstansok):
- a 0199 rekeszben 0
- a 0201 rekeszben 1
- a 0202 rekeszben 4

A változók elosztása:
- a 0200 rekeszben i
- a 0203 rekeszben a_i
- a 0204 rekeszben b_i
- a 0205 rekeszben S
- a 0206 rekesz: munkarekesz

A programot a 0100 címtől kezdődően helyezzük el.

Utasítás címe	Műveleti kód	Első cím	Második cím	Megjegyzés
0100	10	0199	0000	200 címre 0
0101	05	0199	0205	205 címre 0
0102	10	0201	0000	201 címre 1
0103	10	0202	0000	202 címre 4
0104	10	0203	0000	203 címre a
0105	10	0204	0000	204 címre b
0106	03	0203	0204	szorzás
0107	01	0204	0205	összegzés
0110	01	0201	0200	i növelése
0111	05	0200	0206	i munkarekeszbe
0112	02	0202	0206	$n-i < 0$?
0113	07	0114	0104	feltételes ugrás
0114	11	0000	0205	S kiírása
0115	12	0000	0000	STOP

A programműködés elemzésének megkönnyítésére az A-19. ábra a megoldás blokkdiagrammját mutatja.



A-19. ábra. A a_ib_i szorzatösszeg blokkdiagramja

```
C  ADATOK DEKLARALASA
C
      REAL NEV
      INTEGER SORSZ
      DIMENSION NEV(6,100),AR(100)
      REAL MENNY
      N=100
C
C  SORNYOMTATO LAP TETEJERE VALO ALLITASA
C
      WRITE (6,55)
  55  FORMAT (1H1)
C
C  ARUNEVEK ES EGYSEGARAK BETOLTESE ALPROGRAM SEGITSEGEVEL
C
      CALL TORZS(NEV,AR,N)
C
C  FEJLEC IRO ALPROGRAM HIVASA
C
 1    CALL FEJLEC
      SUM = 0.0
C
C  VASARLASI REKORD OLVASASA, FAJL VEGE ESETEN
C  UGRAS A PROGRAM VEGERE
C
 5    READ (5,10,END=100) SORSZ,MENNY
 10   FORMAT (I4,F7.2)
C
C  VASARLASI REKORDOK VEGE? HA IGEN UGRAS AZ 50 CIMRE
C
      IF (SORSZ.EQ.-1) GO TO 50
C
C  HIBAS ARUAZONOSITO? HA IGEN UGRAS A 20 CIMRE
C
      IF (SORSZ.LT.1 .OR. SORSZ.GT.100) GO TO 20
C
C  REKORD FELDOLGOZASA, NYOMTATAS
C
      SERTEK = AR(SORSZ) * MENNY
      SUM = SUM + SERTEK
      WRITE (6,15) (NEV(J,SORSZ),J=1,6),MENNY,SERTEK
 15   FORMAT(11X,6A4,10X,F8.2,8X,F11.2)
C
C  UGRAS A KOVETKEZO REKORD OLVASASARA
C
      GO TO 5
C
C  HIBAS ARUAZONOSITO KEZELESE
C
 20   WRITE (6,25) SORSZ
 25   FORMAT (5X,I4,' ILYEN ARU NINCS')
      GO TO 5
C
C  VEVO ALTAL FIZETENDO OSSZEG NYOMTATASA
C
 50   WRITE (6,60) SUM
 60   FORMAT (11X,'FIZETENDO OSSZEG',32X,F11.2,//25X,
     1'KOSZONJUK A VASARLAST '/1H1)
C
C  UGRAS UJ VEVO REKORDJANAK OLVASASARA
C
      GO TO 1
 100  CONTINUE
      END
```

```
      SUBROUTINE TORZS(NEV,AR,N)
C
C   AZ ARUK NEVENEK ES EGYSEGARANAK BETOLTESE
C
      DIMENSION NEV(6,1),AR(1)
       REAL NEV
      DO 10 I = 1,N
         READ (1,20,END=30) (NEV(J,I),J=1,6),AR(I)
 20      FORMAT (6A4,F5.2,51X)
 10   CONTINUE
 30   RETURN
      END




      SUBROUTINE FEJLEC
C
C    A SZAMLA FEJLECET KESZITO ALAPROGRAM
C
       WRITE (6,20)
 20    FORMAT (1H1,10X,'MEGNEVEZES',14X,'VASAROLT MENNYISEG',
     1          14X,'ERTEK',/)
       RETURN
       END
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. VASARLAS.
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT  TORZS    ASSIGN TO U1.
     SELECT  KARTYA   ASSIGN TO LS.
     SELECT  LISTA    ASSIGN TO EDIT.
 DATA DIVISION.
 FILE SECTION.
 FD  TORZS  LABEL RECORD STANDARD  DATA RECORD TREK.
 01  TREK.
   02 TNEV         PIC X(24).
   02 TAR          PIC 9(3)99.
   02 FILLER       PIC X(51).
 FD  KARTYA LABEL RECORD OMITTED DATA RECORD KREK.
 01  KREK.
   02 SORSZ        PIC S9(3)  SIGN LEADING SEPARATE.
 88   JOSORSZ            VALUE  1 THRU 100.
   02  MENNY       PIC 9(5)V99.
 FD  LISTA   LABEL RECORD OMITTED DATA RECORD LREK.
 01  LREK.
   02  VK          PIC X.
   02  SOR.
     03  FILL      PIC X(10).
     03  LNEV      PIC X(24)B(10).
     03  LMENNY    PIC Z(4)9.998(18).
     03  LERTEK    PIC Z(8)9.99.
     03  FILLER    PIC X(39).
 WORKING-STORAGE SECTION.
 01  TART.
   02 TARSOR   OCCURS 100.
     03  NEV       PIC X(24).
     03  AR        PIC 9(3)V99.
 77  FEJLEC  PIC X(120) VALUE '           MEGNEVEZESE
-    '    VASAROLT MENNYISEG            ERTEK
-    '                          ',
 77  SERTEK        PIC 9(8)V99.
 77  WERTEK        PIC 9(8)V99  VALUE 0.
 77  I             PIC   9(3).
 77  TETEL         PIC  9(2).
 77  VEG           PIC  X       VALUE LOW-VALUE.
 PROCEDURE DIVISION.
 KEZD.
*       FAJLOK MEGNYITASA
     OPEN INPUT TORZS KARTYA  OUTPUT LISTA
     READ  TORZS AT END MOVE HIGH-VALUE TO VEG.
*      AZ ARU NEVENEK ES ARANAK BETOLTESET HIVO CIKLUS
     PERFORM TOLT VARYING I FROM 1 BY I UNTIL I > 100  OR VEG =
     HIGH-VALUE
     CLOSE TORZS.
*       FEJLECET IRO PARAGRAFUS
 FEJIRAS.
     MOVE FEJLEC TO SOR
     WRITE LREK AFTER ADVANCING PAGE
     MOVE 4 TO TETEL.
     MOVE SPACES TO SOR.
```

```
 OLV.
*        VASARLASI REKORD OLVASASA, FAJL VEGE ESETEN UGRAS A
*              VEGE PARAGRAFUSRA.
      READ  KARTYA AT END GO TO VEGE.
*          VASARLASI REKORDNAK VEGE?,
*          HA IGEN UGRAS A KESZ PARAGRAFUSRA
      IF SORSZ = -1    THEN GO TO KESZ.
*        HIBAS AZONOSITO? HA IGEN UGRAS UJ REKORD OLVASASRA
      IF NOT  JOSORSZ  THEN  DISPLAY 'ILYEN ARU NINCS'SORSZ
                             GO TO OLV.
*        REKORD FELDOLGOZASA, NYOMTATASA
      MOVE NEV (SORSZ) TO LNEV
      MOVE  MENNY TO LMENNY
      MULTIPLY MENNY  BY  AR  SORSZ  GIVING SERTEK
      ADD  SERTEK TO WERTEK
      MOVE SERTEK TO LERTEK
      WRITE LREK AFTER ADVANCING  1
      ADD 1 TO TETEL.
      IF TETEL > 60 THEN PERFORM FEJIRAS.
      GO TO OLV.
*           UGRAS A KOVETKEZO REKORD OLVASASARA
 TOLT.
*        ARUNAK ES EGYSEGARANAK BETOLTESE
      MOVE  TNEV  TO NEV (I)
      MOVE  TAR    TO AR (I).
      READ TORZS AT END MOVE HIGH-VALUE TO VEG.
 KESZ.
*        VEVO ALTAL FIZETENDO OSSZEG NYOMTATASA
      MOVE SPACES TO LREK
      MOVE 'FIZETENDO' TO LNEV
      MOVE WERTEK TO LERTEK
      WRITE LREK  AFTER ADVANCING 2.
      MOVE SPACES TO LREK
      MOVE 'KOSZONJUK A VASARLAST' TO LNEV
      WRITE LREK AFTER ADVANCING 2.
*        UGRAS UJ VEVO REKORDJANAK OLVASASARA
      GO TO OLV.
 VEGE.
      CLOSE KARTYA  LISTA.
      STOP RUN.
```

A **programoptimalizálás** azt jelenti, hogy a programban levő, és az optimalizáló lépésben észrevett felesleges dolgok kiküszöbölésre kerülnek. Pl. vezérlésátadás történik egy olyan címre, amelyről további, feltétel nélküli vezérlésátadás van. Ilyenkor azonnal lehet a második címre irányítani a vezérlést.

A program *kódlapra* készült. A kódlap egy sora 80 karakter szélességű, azaz 1 lyukkártyányi. Látható, hogy az utasítások a 7. oszlopból kezdődően találhatók. Az 1—5. oszlopokon helyezkednek el az utasításszámok (címkék). A 6. oszlop jelzi a folytatósort. Ha egy sorban a 6. oszlopon valamilyen jelzés van, ez azt jelenti, hogy ez az utasítás az előző sorban (kártyán) levőnek a folytatása.

Minden programozási nyelvnek meghatározott felosztású kódlapja van. A kódlapokról történik a program lyukasztása, ha a feldolgozás ezt igényli.

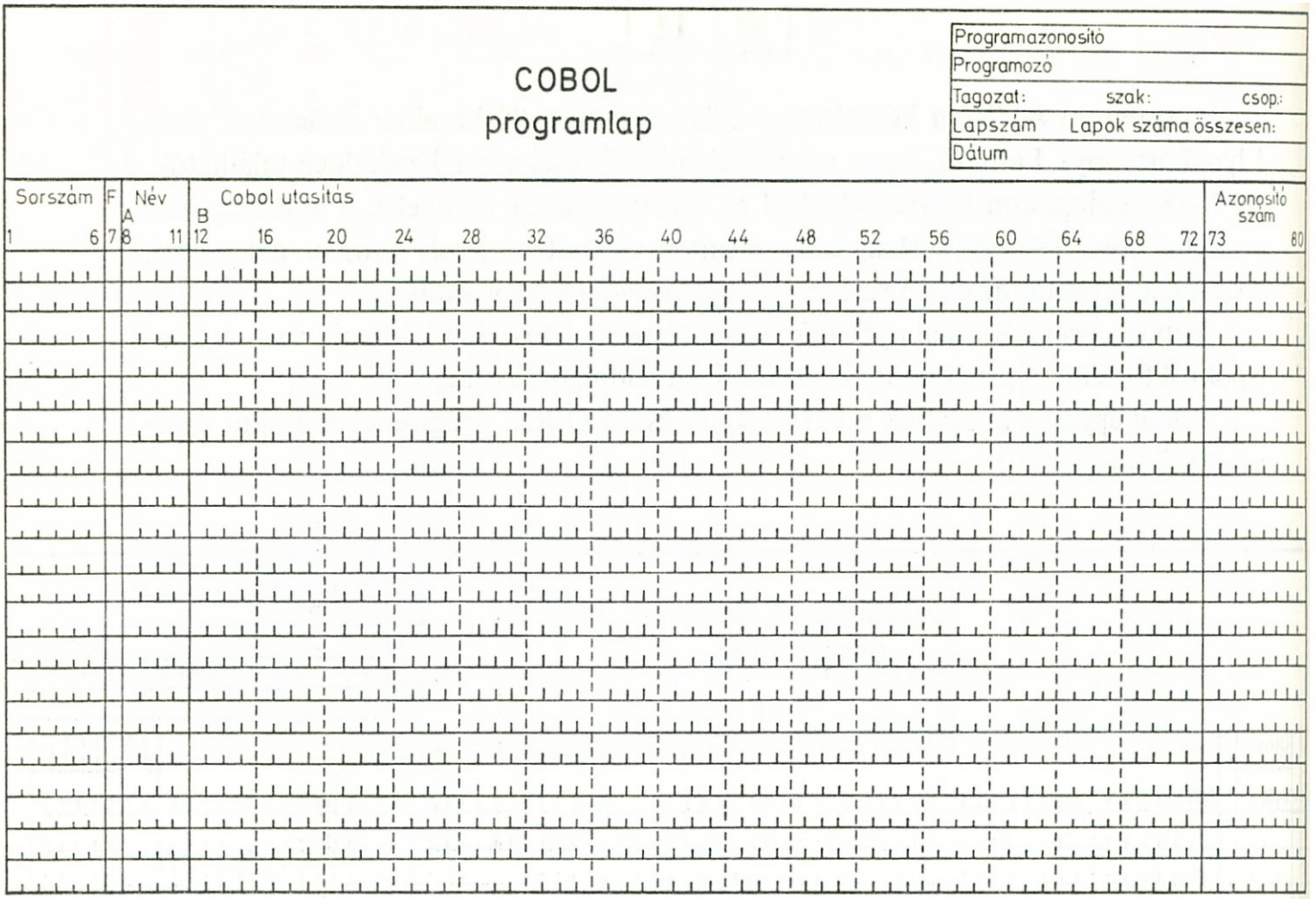
Példaképpen bemutatjuk a FORTRAN és a COBOL programok írásakor használatos kódlapot.

FORTRAN
programlap

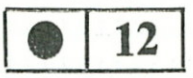
Programazonosító	
Programozó	
Tagozat:	szak: csop.:
Lapszám	Lapok száma összesen
Dátum:	

	Utasítás-szám	6	UTASÍTÁS	
1 2 3 4 5		7 8 9 10		80
1				1
2				2
3				3
4				4
5				5
6				6
7				7
8				8
9				9
10				10
11				11
12				12
13				13
14				14
15				15
16				16
17				17
18				18
19				19
20				20
1 1				80

A-20. ábra. FORTRAN kódlap



A-21. ábra. COBOL kódlap



A tárgyprogramra a felhasználónak általában nincs szüksége. Néha azonban előfordul, hogy valamilyen programhibát a forrásprogram alapján a programozó nem talál meg. Ilyenkor felmerül annak a lehetősége, hogy esetleg a lefordított programba — valamilyen okból — hiba került, vagyis a tárgyprogram és a forrásprogram között algoritmikus eltérés van. Ilyenkor kerül sor a tárgyprogram „megfejtésére", a tárgykód részletesebb vizsgálatára. Ez nagyon időigényes munka, szerencsére viszonylag ritkán van rá szükség.

A lyukkártyán megjelenített tárgyprogramnak akkor volt jelentősége, amikor még nem léteztek mágneses háttértárak. A programfordítás és -végrehajtás akkor úgy zajlott, hogy a gép által lyukasztott tárgyprogramot kézzel be kellett helyezni a kártyaolvasó berendezésbe, újra be kellett olvastatni, és így lehetett továbblépni a feldol-

gozásban. Ez nem csak azért volt rossz, mert lelassította a gép működését, hanem azért is, mert a kártyaolvasó is hibázhatott, s a visszaolvasott tárgyprogram ilyenkor nem volt azonos a kártyán levővel. A mágneses háttértárakkal ez a probléma megoldódott.

● 14

A relatív cím azt jelenti, hogy a fordítóprogram a változó nevekhez hozzárendelt tárterületeket (szavakat) 0-tól kezdve sorszámozza.

● 17

Ha valaki pl. olyan programot ír, amelyben már meglevő, mások által elkészített alprogramokat akar használni, ezeket az alprogramokat lefordított állapotban egy külön könyvtárban célszerű tárolni. Ezzel megtakarítható ezen alprogramok állandó (és felesleges) újrafordítása.

● 18

Hibásan fejeződik be a szerkesztés pl. abban az esetben, ha a programban egy olyan alprogramot hívunk meg, amely nem létezik, ill. a kapcsolatszerkesztő a számára előírt input könyvtárakban nem találja.

Az ilyen program éppen ezért nem is futtatható. A kapcsolatszerkesztő diagnosztikai üzeneteiből az ilyen hiba kiderül.

● 19

A terminálokról a következő fejezetben lesz szó.

1965-ben egy terminál gyakorlatilag egy számítógéphez illesztett írógép formájában jelent meg. Billentyűzetén üzent a programozó, s a betett papírra jött vissza a számítógép válasza.

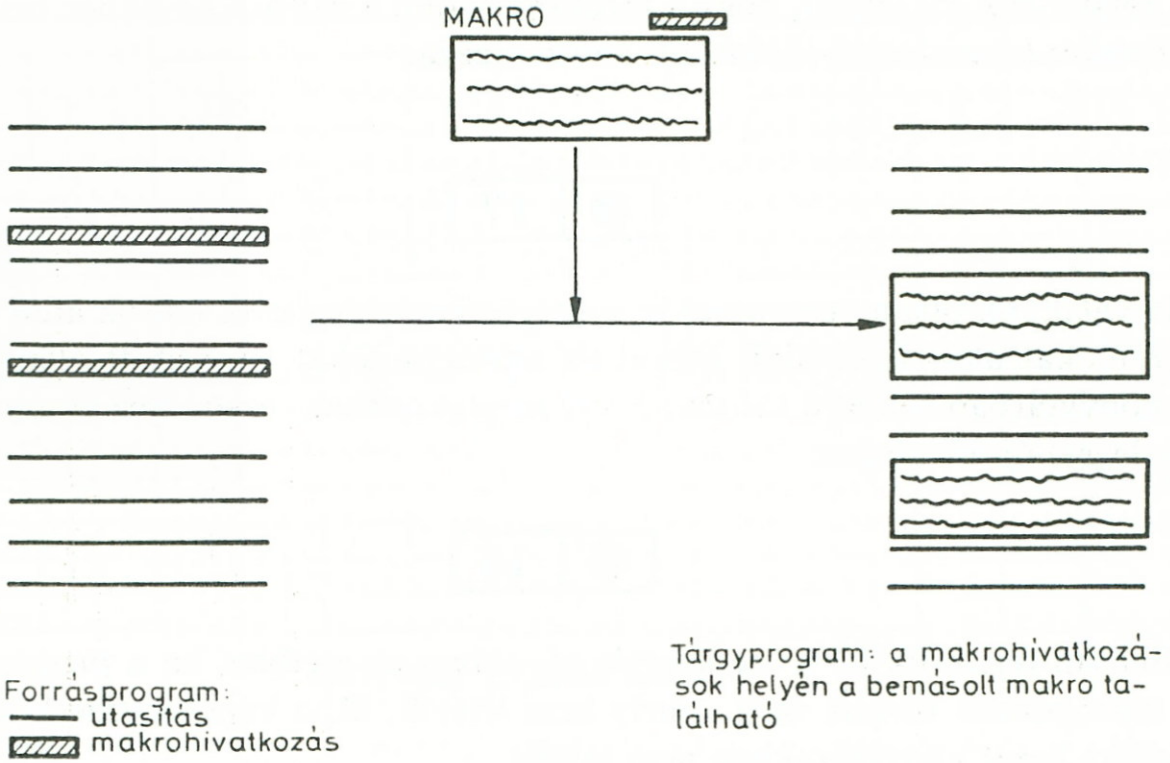
⊙ 4

A *makroutasítás* programban való használatához az szükséges, hogy a rogramozó (vagy korábban valaki más) definálja a makrót, amelyre ezek után a definiálás

során adott névvel lehet hivatkozni. Természetesen az Assemblernek ez egy újabb feladatot jelent: képesnek kell lennie arra, hogy a makrohivatkozást felismerje, és a hivatkozás helyére bemásolja a makroban definiált utasítássorozatot.

A programozó a megoldandó feladattól függően definiálhat különböző makrokat, s ezzel lényegesen könnyebbé teheti a programozás, a tesztelés és a későbbi módosítás munkáját. Ugyanakkor a program is áttekinthetőbbé válik.

A makrok beépülését a tárgyprogramba az A-20. ábra szemlélteti.



A-22. ábra. Makrók beépülése a tárgyprogramba

Pl. a PASCAL programozási nyelv, amelyet a hetvenes évek közepén dolgoztak ki, kiválóan alkalmas is rendszerszoftver jellegű programok megírására.

Annak biztosítására, hogy a feldolgozóprogram mágneslemezen elhelyezett outputjai ne kerülhessenek továbbfeldolgozásra, ha azok hibásak, többféle megoldás létezik.

Az egyik legelterjedtebb megoldás lényege, hogy minden feldolgozóprogram, így a fordítóprogramok is egy ún. *visszatérési kódot* generálnak a futás befejezésekor. A visszatérési kódból megállapítható, hogy a futás milyen eredménnyel fejeződött be.

Pl. a fordítóprogramok visszatérési kódjai és azok jelentése a következő:

Visszatérési kód értéke	Jelentése
0	hibátlan befejezés
4	olyan hibák vannak a forrásprogramban, amelyektől a futás még jó lehet (pl. deklarált egy tömböt, de nem használta)
8	olyan hibákat talált a forrásprogramban, amelyek miatt a továbbfeldolgozás is hibás lesz A teljes programot fordítja A kapcsolatszerkesztő nem indul el automatikusan, de a programozó előírhatja, hogy induljon el a kapcsolatszerkesztő működése
12	olyan hibákat talált a fordítás során, amelyek a végrehajtást lehetetlenné teszik
16	a fordítóprogram működése megszakadt a fordítás során

A visszatérési kódok ismeretében a továbbfeldolgozás megengedése vagy letiltása automatizálható.

VII. fejezet

★	5

Az OCR betűrövidítéssel már találkoztunk a II. fejezetben. Ott egy írási szabályról volt szó, amelyet OCR-A-nak neveznek. Itt most azokról a perifériákról lesz szó, amelyek az OCR szabvány szerint írt szövegeket el tudják olvasni.

Az OCR betűrövidítés az angol Optical Character Recognition = optikai úton való jelfelismerés kifejezésből ered.

★	17

A csatorna szó a korábbiakban azt a hardvert jelentette, amely lehetővé teszi a központi egység és a perifériák összekapcsolását (szelektorcsatorna, multiplex csatorna). Ebben az összefüggésben a csatorna szó mást jelent, ahogy az a továbbiakból is kiderül.

★	26

A „rekord" szó eddigi szóhasználatunkban adatstruktúrát jelentett: szavak egymásutánját pl. egy input-output listában. Mágneslemeznél *szintén rekordnak nevezik* (és R betűvel jelölik) azt az együttest, amely az ún. leíróterületből és a felhasználói adatterületből áll. A leíró területen a cím-információk, az adatterületen a felhasználói rekordok vagy blokkok találhatók.

▲	1

A kártyaolvasó berendezés lyukkártyákon levő adatokat és programokat juttat be a központi tárba. A lyukkártya felépítéséről és a rajta való adatelhelyezésről a II. fejezetben tanultatok.

▲	4

A másodlagos adathordozók előállításával kapcsolatos problémákat a II. fejezetben tárgyaltuk.

▲ 14

A rendszer működésébe való beavatkozás pl. a rendszer elindítása reggel üzemkezdéskor. Előfordulhat olyan eset, hogy egy program futását a konzolról kell megszakítani, mert olyan erőforrást köt le hosszú ideig, ami megbénítja a rendszer működését.

Ugyancsak a konzolnál kell aktivizálni az egyes perifériákat.

Konzolra menő üzenet a rendszer részéről pl. egy mágnesszalag vagy mágneslemez iránti kérés, ha ez a szalag vagy lemez nincs feltéve.

Ugyancsak a konzolról értesül az operátor, ha pl. a kártyaolvasóba beszorul egy kártya, vagy valamilyen input-output hiba lép fel olvasáskor vagy nyomtatáskor.

▲ 19

A két csévét nevezzük A-nak és B-nek. Ha a mágnesszalag az A-ról a B-re tekercselődik át, és B valamilyen okból lelassul, az A-hoz tartozó vákuumkamrában megnő az oda befutó szalaghossz. Ha B valamilyen okból (pl. az induláskor) jobban gyorsul, mint A, a B-hez tartozó vákuumkamrában lecsökken az oda befutó szalag hossza. A vákuumkamrában levő szalagrésznek tehát kiegyenlítő szerepe van.

▲ 21

Gondoljunk csak egy magnetofonszalagra, annak az elején is van egy ún. befűző szalag, amely ahhoz kell, hogy a fogadó orsóra már rátekeredjék a szalag eleje, miközben a lejátszófej előtt még nem haladt el „értékes" szalagrész.

Ugyanígy a szalag végén van egy kifutó rész, amely ahhoz kell, hogy a felvétel végéig feszes legyen a szalag.

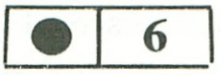
▲ 25

A sáv egy kör, így a kezdőpont definíció kérdése. A lemezen található egy olyan hardver jel, amely meghatározza valamennyi sáv kezdőpontját a körön.

● 2

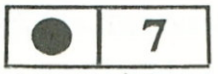
Egyszerűbb (és olcsóbb) olvasóberendezéseknél csak egyszeri olvasás történik.

Az ilyen programok átírása oly módon, hogy ezek az outputok pl. mágnesszalagra vagy mágneslemezre kerüljenek sokszor nem oldható meg, mert a forrásnyelvi változata a programnak nincs is meg.

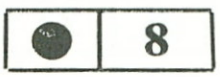


Ugyanakkor léteznek olyan speciális területek, ahol az OCR technika tért tudott hódítani. Ilyen a már korábban is említett banki alkalmazás, ahol a csekkek valódiságát ilyen berendezésekkel egyszerűen lehet ellenőrizni.

Speciális bizonylatolvasónak tekinthetjük azt a „fényceruzát" is, amely a vonalkódokat tudja elolvasni és az árukat azonosítani.

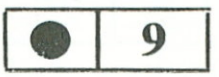


A Magyar Posta keretei között működő berendezés, amely az irányítószámok alapján a postai küldeményeket osztályozza, bár nem számítógép, de szintén optikai olvasóberendezésnek tekinthető.



A nyomtatott eredménymegjelenítés idősebb, mint a számítógép.

Az elektromechanikus elven működő középgépek feldolgozási eredményeit is táblázógépek írták ki. Ezek működése azonban nagyon lassú volt, bár akkor megfelelt a középgépek feldolgozási sebességének. (Ui. a feldolgozás is lassú — elektromechanikus — volt.)



A nyomtató berendezések jelkészletének egyik problémája, hogy betűkészletük általában csak a nagybetűket tartalmazza, és hiányoznak a magyar abc ékezetes magánhangzói.

Ez utóbbiakra sok helyen nagy szükség van, pl. ahol az OTP átutalási betétszámlák feldolgozása vagy a közüzemi számlák készülnek. Az állampolgárok nevét ui. pontosan kell írni. Ilyen esetekben a legegyszerűbb megoldás, hogy bizonyos speciális jelek helyére a nyomtatón az ékezetes magánhangzók képét szerelik fel (pl. a $ helyére az Ö betűt).

Ma már azonban léteznek olyan nyomtatók, amelyek a magyar abc kis- és nagybetűit egyaránt tudják nyomtatni.

● 10

Rajzolóberendezés

A számítógépeknek a műszaki élet területén való felhasználásakor gyakran felmerül az igény arra, hogy a számítások eredménye ne számok, hanem valamilyen rajz vagy ábra formájában jelenjék meg. Ilyen ábra pl. egy autópálya vonalvezetése során megtervezendő kanyar képe vagy egy új autókarosszéria ábrája. Az ilyen jellegű igényeket a különböző rajzolóberendezések elégítik ki. A rajzolóberendezés tehát egy olyan output egysége a számítógépnek, amely arra képes, hogy rajzokat, ábrákat jelenítsen meg papíron. Természetesen ezt akkor tudja megvalósítani, ha erre programozva van. A programozónak olyan programot kell készítenie, amely előre megadja azt, hogy az adatok függvényében hogyan alakuljanak a kirajzolt vonalak.

Ez tehát bizonyos speciális szoftver meglétét igényli. Minél inkább használnak rajzolóberendezést egy számítógépnél, annál nagyobb hangsúlyt fektetnek az üzemeltetők arra, hogy e munkát kényelmessé tevő programok álljanak rendelkezésre.

Maga a hardver berendezés többféle megoldású lehet. A leggyakrabban használt megoldás, hogy a sík egyik irányába való mozgást a papír mozgása, az erre merőleges mozgást pedig a rajzolótoll mozgása biztosítja. A toll kétféleképpen tud mozogni: nyomot hagyva, vagy úgy, hogy nem hagy nyomot. (A papírra ráhelyezett vagy arról felemelt módon.) Csak olyan helyeken használják, ahol erre igény van.

● 12

Ilyen speciális alkalmazás pl. a telefonkönyvek számítógépes előállítása és mikrofilmen való tárolása.

A telefon tudakozóban ezekről a mikrofilmekről olvasva tudnak felvilágosítást adni.

● 15

Az ismertetett mátrixnyomtatás ötlete régi. A lyukkártyákra a feliratok ugyanezzel a technikával készülnek.

● 16

Amíg a számítógép erre nem volt képes, a probléma megoldását az adatok másodlagos adathordozón (pl. lyukkártya) való tárolása jelentette. Ez persze rengeteg problémát okozott:

— nagy volt a tárolás helyigénye;
— lassú volt a hozzáférés (megkeresni, beolvastatni);
— maga a beolvasás ritkán volt hibamentes;
— minél többször lett egy kártya beolvasva, annál inkább elhasználódott.

Ezért nagyon hamar sor került jobb megoldások realizálására.

● 18

A nem bájt szervezésű gépek, amelyek zömmel 24 bites szavakból álló tárral rendelkeznek, 7 csatornás mágnesszalagokat használnak (4 db 6 bites BCD kód + a paritásbitek).

A bájt szervezésű és a nem bájt szervezésű gépek közötti adatcsere éppen ezért problematikus, hiszen a mágnesszalagok eltérő fizikai tulajdonságúak.

Ha a bájt szervezésű gép rendelkezik 7 csatornás mágnesszalag-olvasóval, ezen keresztül a szalag olvasható, és a BCD kód 8 bites EBCDIC kódra konvertálható.

● 24

Ha pl. egy fájl egyetlen cilinderen elfér, a teljes fájl olvasása egyetlen fejmozgást igényel, amikor az író-olvasó fejek beállnak az adott cilinderre.

Ez a példa is jól szemlélteti a cilinder mentén való írás nagy előnyét.

⊙ 11

Maga a berendezés a következő részekből áll:

— 32 Kbájtos központi egység,
— 2 db mágnesszalag egység,
— 1 db 10 Mbájtos mágneslemez egység,
— 2 db hajlékony lemezegység,
— 1 db katódsugárcsöves felvevő (kamera),
— 1 db konzol írógép,
— 1 db automatikus előhívó,
— 1 db másoló.

⊙	13

A számítástechnikában élenjáró országokban már kidolgozták a CIM (Computer Input Mikrofilm) technikát is, azaz a mikrofilmre vitt adatok egy későbbi feldolgozás inputjául szolgálhatnak minden átalakítás nélkül.

⊙	20

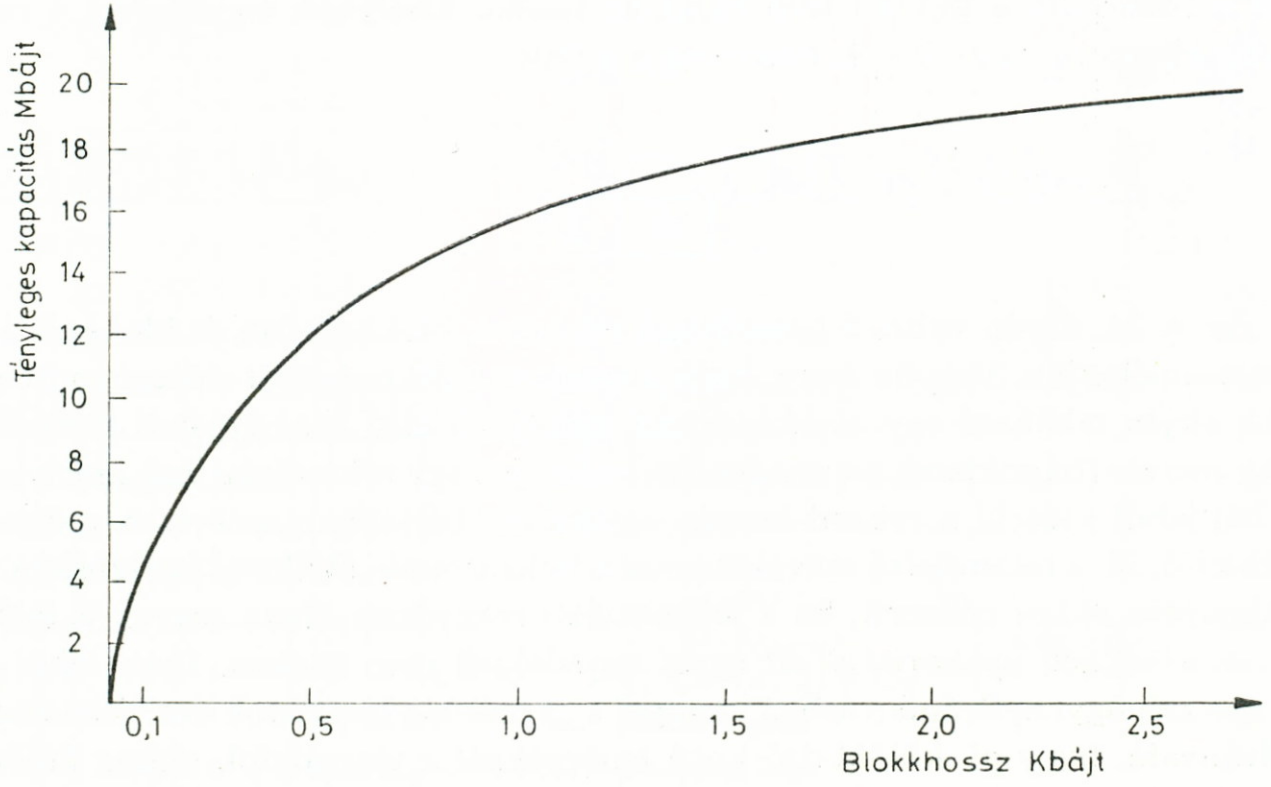
Jelöljük a blokk bájtokban mért hosszát N-nel. Egy 730 m-es mágnesszalag tényleges kapacitása:

$$Kt = \frac{730\,N}{15{,}2 + 0{,}0317\,N},$$

ahol Kt a tényleges kapacitás; 15,2 a blokkhoz tartozó blokk-köz hossza mm-ben; 0,0317 1 bájt helyigénye mm-ben.

Az eredményt Kbájtban kapjuk. Pl.: ha kártyaképi formában visszük fel a rekordokat, blokkolatlan formában (N=80), a tényleges kapacitás 9,3 Mbájt, vagyis a 22 Mbájt-os névleges kapacitás hetedrésze.

A blokkméret növelésével a tényleges kapacitás eleinte rohamosan növekszik. Ez a növekedés azonban a 2 Kbájtos blokkméretnél lelassul, s a blokkméret további növelése már nem célszerű, mert a szükséges input-output terület biztosítása már több hátránnyal jár, mint amennyi előny származik belőle. Az elmondottak az ábráról is



A-23. ábra. A mágnesszalag kapacitásának alakulása a blokkhossz függvényében

leolvashatók. Az írás és olvasás sebessége 120...150 cm/s. A visszacsévélés sebessége ennél nagyobb, típustól függően 5...10 m/s.

Az elmondottakat az A-23. ábra szemlélteti.

⊙	22

A rögzített információk azonosítására tehát háromféle címke szolgál:

— adathordozói címke, amellyel a mágnesszalag tekercs azonosítása történik;

— fájl bevezető címke, amellyel a fájl azonosítása történik (egy fájlhoz több is tartozhat);

— fájl zárócímke, amely a fájl végén található.

Ismerkedjünk meg közelebbről ezek tartalmával.

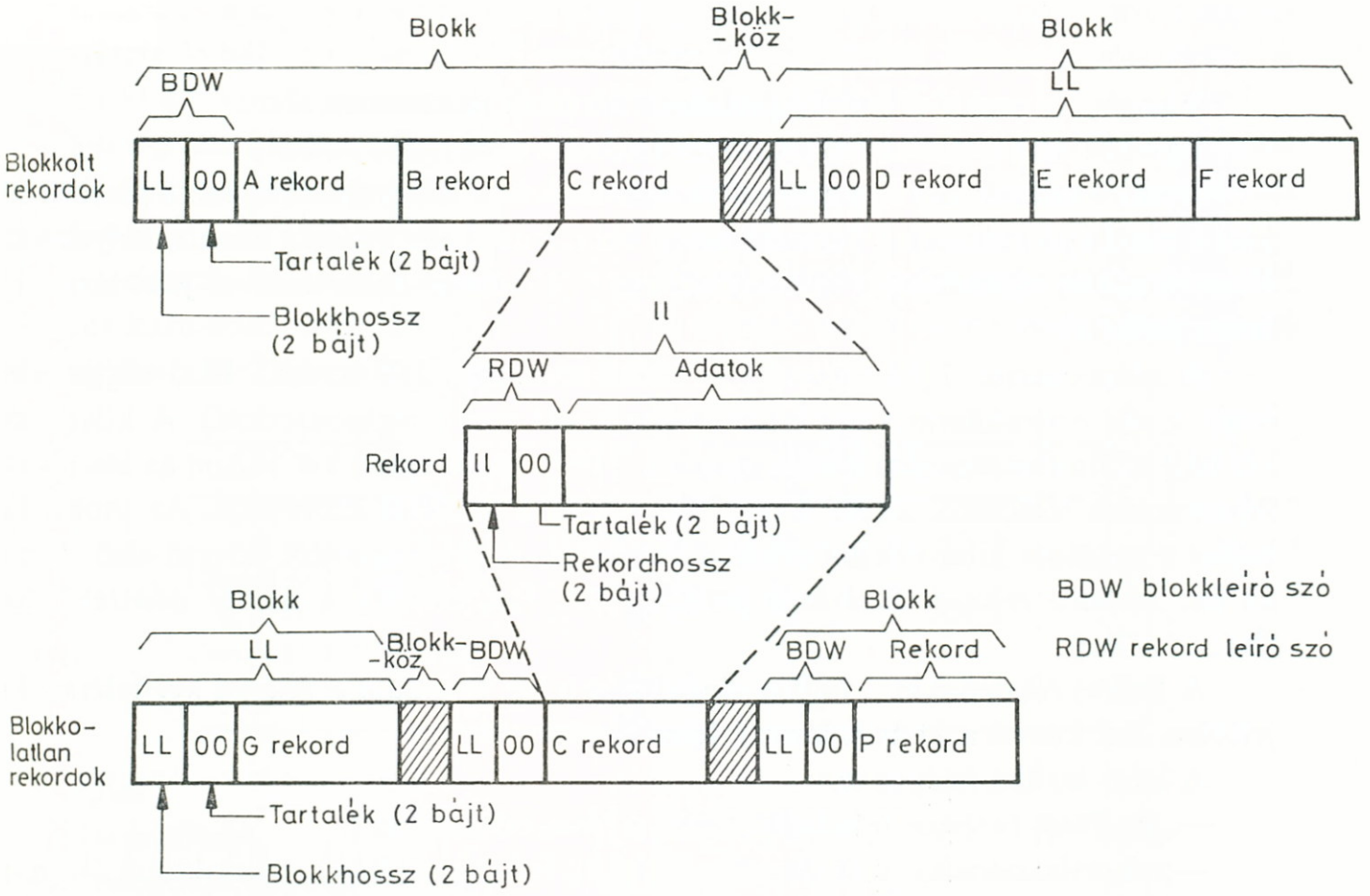
Az adathordozói címke (más szokásos megnevezése: kezdeti köteg címke) a mágnesszalag tekercs és a felhasználó azonosítását szolgálja. Mérete 80 bájt. Segítségével ellenőrzi pl. a rendszer, hogy az operátor által feltett tekercs megegyezik-e a program által kért tekerccsel.

A fájl bevezető címke a fájl azonosítására szolgál. Megtalálható benne a fájl neve, létrehozási ideje, rekordhossza, blokkmérete stb. E címkében lehet gondoskodni arról, hogy a szalag olvasása csak azok számára legyen lehetséges, akik ismerik a védőkulcsot. (Ez egy, a címkébe felírt karaktersorozat, amelyet a konzolról kell beírni, s a rendszer csak ekkor nyitja meg a fájlt.) Vannak operációs rendszerek, amelyek további bevezető címkéket is megkívánnak.

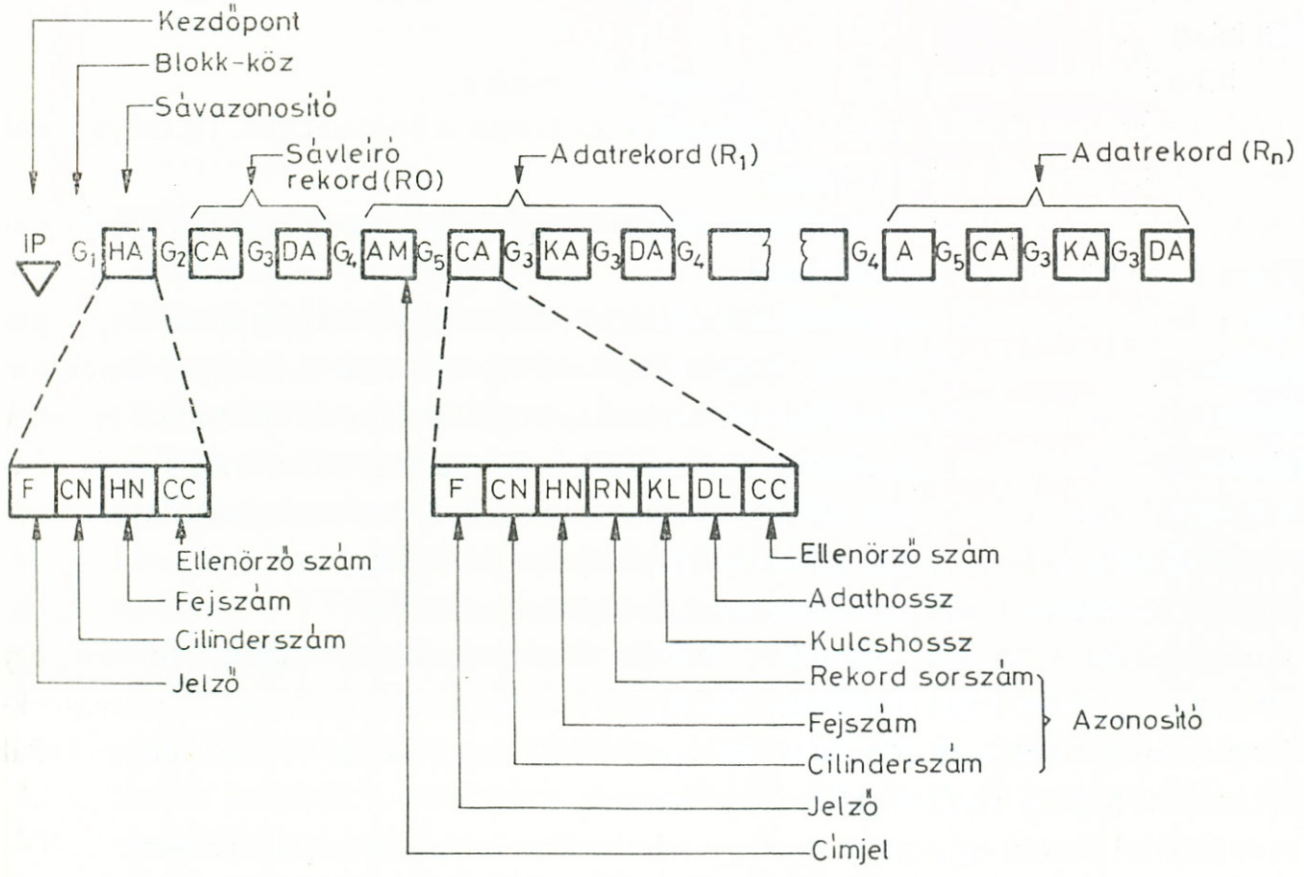
A fájl zárócímkéjében ugyanazok az információk találhatók, mint a fájl elején. Ezen túl felíródik a fájlban levő blokkok száma, amelynek segítségével a rendszer mindig ellenőrzi, hogy jól olvasta-e végig a fájlt.

⊙	23

Az A-24. ábrán változó hosszúságú rekordok blokkolatlan és blokkolt ábrázolását szemléltetjük. Vegyük észre, hogy a rendszer tájékozódását elősegítendő minden blokk elején található egy *blokkleíró szó*, amelynek első két bájtjából kiolvasható a blokk mérete (bájtokban), és minden rekord elején egy *rekordleíró szó*, amelynek első két bájtjából kiderül a rekord hossza ugyancsak bájtokban mérve. A méretekbe a blokkleíró, ill. a rekordleíró szavak hossza is beletartozik. (Változó hosszúságú rekord alkalmazása akkor célszerű, ha a felhasználói rekordban olyan szavak is előfordulnak, amelyeknek gyakorisága az egyes egyedeknél nem azonos. Ilyen lehet pl., ha egy egészségügyi nyilvántartásban szerepel a „korábbi vizsgálatok éve" jelentésű adat. Nyilvánvaló, hogy pl. különböző korú embereknél a vizsgálatok száma különböző. Van akinél 1 vagy 2, van akinél 10 vagy 20.)



A-24. ábra. Változó hosszúságú blokkolatlan és blokkolt rekord



A-25. ábra. Általános sáv struktúra mágneslemezen

Vizsgáljuk meg a sávok felépítésének kérdését részletesebben.

Egy sáv fizikai szerkezetét mutatja az A-25. ábra. (289. oldal)

A sáv kezdetét egy *hardver jel* (a gyártáskor a lemezen elhelyezett perforáció) jelzi a meghajtó számára. (Természetesen valamennyi sáv kezdete azonos helyen van, hiszen a sávok egymás alatt találhatók.) Az A-25. ábrán ezt a hardver jelet nevezzük index pontnak.

A *sávazonosító* 7 bájt hosszú és a sáv azonosítására szolgál. Első bájtja jelzi, hogy a sáv használható vagy használaton kívül van (meghibásodott). A következő két bájt a cilinder sorszámát adja meg (0...199), majd újabb két bájton az író-olvasó fej sorszáma található, amely a cilinderen belül a felületet azonosítja. Az utolsó két bájtot a rendszer hibavédelmi célokra használja. A sávazonosítót követő első rekord az ún. *sávleíró rekord*. Szokásos jelölése: R0. Ezt követik a fizikai adatrekordok (R1, ..., Rn).

A fizikai rekordokat 2 bájtos címjel vezeti be. Feladata a rekord kezdetének kijelölése. Ezt követi a 11 bájt hosszú leíró terület.

A leíró terület felépítése:

— jelzőbájt (azonos feladattal, mint a sávazonosítóban);

— rekordazonosító: cilinderszám (2 bájt), fejszám (2 bájt), rekordszám (1 bájt);

— kulcshossz (1 bájt): a keresési kulcsterület hosszát adja meg, értéke 0 és 255 közötti érték lehet (ha 0, nincs külön kulcsterület);

— adathossz (2 bájt): az adatterület bájtokban mért hossza. Értéke 0...65 535 között lehet;

— hibavédelmet szolgáló 2 bájt a rendszer céljaira.

Ha a kulcshossz értéke nem 0 volt, most következik a kulcsterület. Itt helyezkedik el a rekordhoz tartozó keresési kulcs.

A keresési kulcs a rekordnak egy olyan azonosítója, amely szerint a rekordot a fájlban kereshetjük, direkt eltéréssel.

Az a tény, hogy a kulcs a külön területen van, meggyorsítja a keresést, vagyis a rendszer rövidebb idő alatt meg tudja állapítani egy rekordról, hogy a keresett-e, mintha a teljes adatterületet is le kellene olvasnia, hogy az azonosítót onnan vegye ki. A kulcsterületről olvasott azonosítót ui. maga a lemezvezérlő ellenőrzi, s ha a keresett rekordról van szó, azonnal olvasni is lehet a rekordot, mert az olvasófej a hasonlítás ideje alatt a kulcsterületet követő blokk-közön halad át.

Az adatterületen található a felhasználó logikai rekordja.

Amennyiben a logikai rekordok felírása blokkolatlan formában történt, egy adatterületen egyetlen logikai rekord található. Amennyiben az adatok felvitele blokkolt formában történt, egy adatterületen egy blokk, vagyis a blokkolási tényező által meghatározott számú logikai rekord található.

Adatátvitel esetén egyszerre egy-egy adatterület tartalma kerül átvitelre.

Az egyes területeket blokk-közök választják el. Ezek különböző hosszúságú bit-

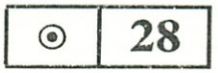
kombinációk, amelyeket a vezérlőegység használ. (Pl. az ábrán G_1-gyel jelölt blokk-köz mérete 36 bájt, a G_2-vel és a G_3-mal jelzett mérete 18 bájt. Az egyes adatrekordok közötti blokk-közök mérete már függ az adatmező hosszától.

Ha egy sáv hardver hibás lesz, helyére ún. *alternatív sávot kell kijelölni a tartalék-sávok közül.* Az alternatív sáv kijelölésének mechanizmusát egy példán keresztül mutatjuk be:

Tételezzük fel, hogy meghibásodott a 42. cilinder hatodik sávja. Helyette a 201. cilinder harmadik sávját kell aktivizálni. Az érintett sávok sávazonosítójában és R0 rekordjában levő azonosítók így alakulnak:

	Sávazonosító R0 leíró terület	
	J C F	J C F
Eredeti (42. cilinder 6. sáv)	2 42 6	2 201 3
Tartalék (201. cilinder 3. sáv)	1 201 3	1 42 6

Ahol:
J=jelzőbájt,
C=cilinderszám (1 — aktív; 2 — inaktív),
F=fejszám.
Az érintett sávok között tehát egy kereszthivatkozás jön létre.



A *VTOC* az operációs rendszer által használt *címkéket* tartalmaz. Ötféle címke létezik, valamennyi 140 bájt méretű.

— *Minden fájlhoz* kötelezően tartozik egy *Format 1* típusú címke. Tartalmazza a fájl azonosítására szolgáló nevet, a létrehozás időpontját, megőrzési idejét, a fájl leírását (rekordméret, rekordforma, blokkméret stb.), valamint a fájlt magában foglaló legfeljebb három tartomány kezdő- és végcímét. Ha a fájl háromnál kevesebb tartományban helyezkedik el, ezekből a mezőkből csak a szükségesek vannak feltöltve. Ha a fájl háromnál több tartományban helyezkedik el, egy mutatót is tartalmaz a címke, amely a VTOC egy másik címére mutat. Itt egy Format 3 típusú címkében megtalálható a további tartományok leírása.

— Az *indexelt szekvenciálisan* szervezett fájlokhoz *Format 2* típusú címke tartozik a VTOC-ban. Ebben a címkében a fájl belső elhelyezkedéséről (indexterület, túlcsordulási terület) is találhatók információk. (Az indexelt szekvenciális fájlról a VIII. fejezetben lesz szó.)

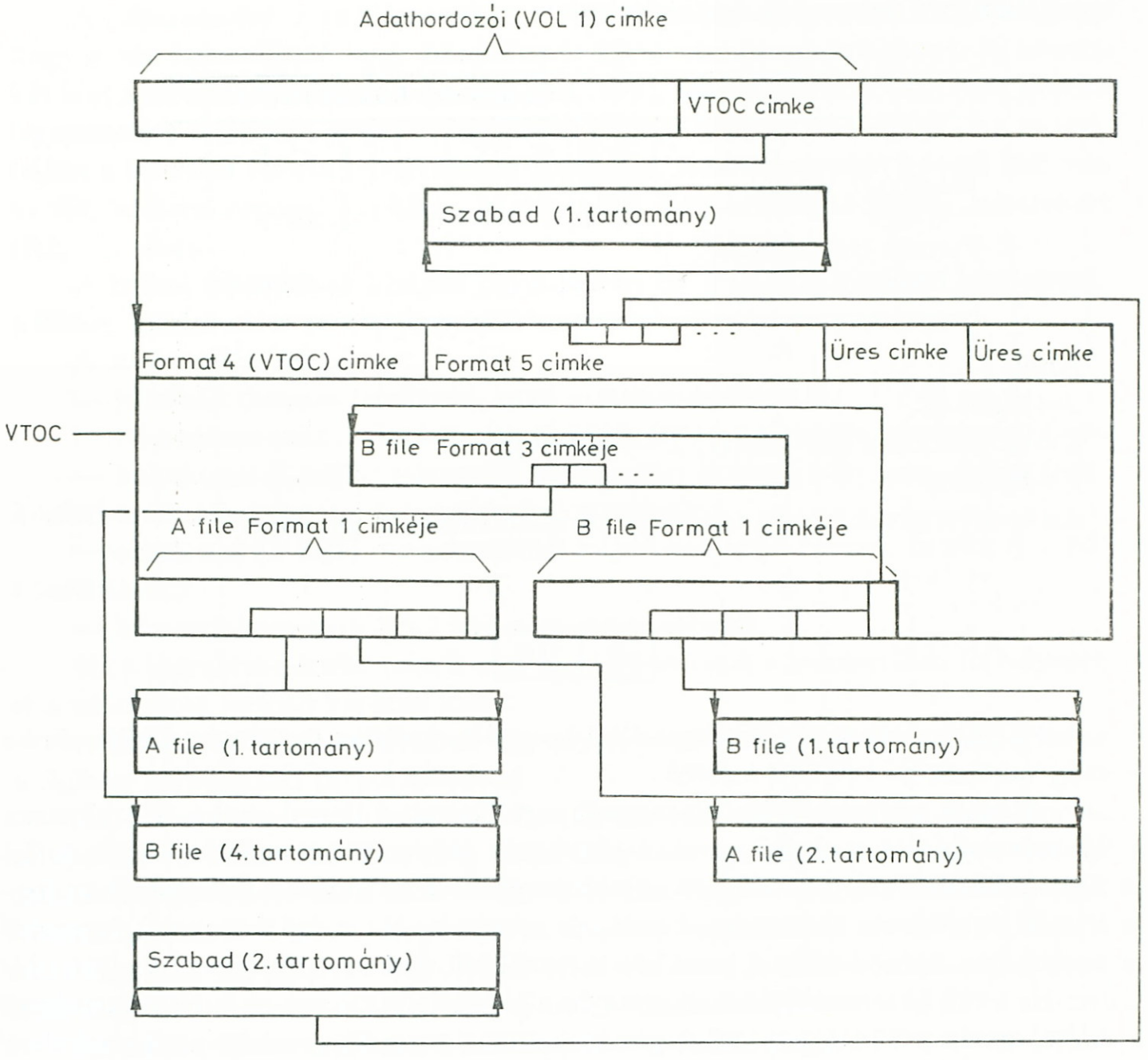
— A *Format 3* típusú címkére akkor van szükség, ha a *fájl háromnál több tarto-*

mányban kerül elhelyezésre. Egy Format 3 típusú címkében további 13 kezdő- és végcímke helyezhető el.

— A *Format 4 típusú* címke magát a *VTOC-t* írja le.

— A *Format 5* típusú címke a *lemezen levő üres helyeket tartja nyilván.* (Az üres helyeket mint speciális fájlokat kezeli a rendszer.)

Az elmondottakat szemlélteti az A-26. ábra, amely egy mágneslemez VTOC-jának és a fájlok elhelyezkedését szemlélteti sematikusan.



A-26. ábra. Szabványos címkék elhelyezkedése mágneslemezen

A felhasználói fájl címkék az egyes fájlok elején találhatók. Méretük 80 bájt. Használatuk nem kötelező, az operációs rendszer nem is gondoskodik feldolgozásukról. Alkalmazásuk esetén a felhasználónak kell megírnia a címkefeldolgozó programot is.

VIII. fejezet

▲	2

Amint azt már a III. fejezetben megtanultuk, a „fájl vége" jel egy önálló rekord a fájl utolsó adatrekordja mögött. Ebben a rekordban egy olyan karakterkombináció van, amelyről a rendszer észleli, hogy elfogyott a fájl. Szemmel is látható fájl vége jel van pl. a lyukkártyán levő adatfájl végén: egy önálló kártya, amelyben a /* lyukasztás található.

▲	3

Miután a törzsfájl mágnesszalagon van, ezért — amint azt a VII. fejezetben tanultuk — nem lehet magába a fájlba beleírni, így kénytelenek vagyunk az új törzsfájlt egy másik mágnesszalagon létrehozni.

▲	4

Itt kerül kihasználásra a fájlok rendezettsége. Ha az első javító fájlbeli rekord azonosítója 00172, ez azt jelenti, hogy ugyanebben a fájlban utána már csak ennél nagyobb azonosító érkezhet, következésképpen a törzsfájlban minden 00172-nél kisebb azonosítójú rekord változatlan marad.

▲	5

A későbbiekben látni fogjuk, hogy a független túlcsordulási területre (elvileg) bármelyik cilinder feldolgozása során szükség lehet. Ha a független túlcsordulási területet az állomány végén helyeznénk el (pl. a 48. és 49. cilinderen), ez azt jelentené, hogy az állomány elejéről (pl. a 18. cilinderről) nagyon messze lenne ez a terület, vagyis hosszabb lenne az az út, amelyet az író-olvasó fejeknek meg kell tenniök. Ez természetesen hosszabb időt is jelent. Az állomány közepén levő független túlcsordulási terület átlagosan kisebb utat jelent, ha sok cilinderről szükséges elérni.

▲ 8

Felmerül a kérdés, hogy miért van szükség kétféle túlcsordulási területre?

Ha az új belépő rekordok azonosítói (kulcsai) a lehetséges értelmezési tartományban egyenletesen oszlanak el, tulajdonképpen a független túlcsordulási terület el is hagyható. Ilyenkor ui. az a helyzet, hogy minden cilinder-túlcsordulási területen keletkeznek túlcsordult rekordok, de mire ezek is betelnének, a fájl újraszervezésre kerül.

Ha azonban az a helyzet, hogy van egy olyan része a kulcsok értelmezési tartományának, amelyben több új rekord várható, mint más részeknél, az egyik (vagy néhány) cilinder túlcsordulási területe már betelhet, miközben más cilinder túlcsordulási területek még üresek. Ilyenkor „jól jön" a független túlcsordulási terület. Ez utóbbi eset fordulhat elő pl. abban az esetben, ha egy indexelt szekvenciális fájlban termékek kerülnek nyilvántartásra, és egyszerre megjelenik egy új termékcsalád a választékban. Nyilván ennek az új termékcsaládnak (15—20-féle terméknek) a kódszáma (kulcsa) egymáshoz közelálló értékeket tartalmaz. Ilyenkor bizony előfordulhat, hogy a cilinder túlcsordulási terület kicsinek bizonyul.

Az elmondottakból az is következik, hogy amikor egy indexelt szekvenciális fájlt létrehoznak, ismerni kell a fájlnak a feldolgozásban játszott szerepét is.

▲ 10

A relatív cím azt jelenti, hogy a program kezdőcíme a könyvtári fájl elejéhez van viszonyítva. Ha ismerjük a könyvtár kezdeti helyét a lemezen, a program helye is kiszámítható.

● 9

Direkt-fájlszerkezet

A direkt fájlszerkezet az alapvető fájlszerkezetek közé tartozik. Ez azt jelenti, hogy a fájlbeli rekordok kezeléséhez nincsen szükség az adatrekordokon kívül további információkra. Elnevezéséből következik, hogy a direkt szervezésű fájl rekordjai közvetlenül is elérhetők. Ez utóbbi tényből pedig az következik, hogy ilyen szervezettségű fájlt csak közvetlenül címezhető tárolón (mágneslemezen) lehet létrehozni.

Milyen elven alapszik a direkt fájl szervezése? Az elv lényege, hogy a rekord logikai azonosítója és a rekord fizikai helye (lemezbeli címe) között kapcsolat teremthető. A kapcsolatteremtés azt jelenti, hogy megadható egy olyan leképzés, amely-

nek segítségével a rekordazonosítóból kiszámítható az a cím, ahol a rekord található. Ez általában egy relatív címet ad, vagyis a fájl elejéhez viszonyítva határozza meg a rekord helyét (cilinder, sáv, sávon belüli rekordsorszám). Attól függően, hogy a direkt fájl a lemez mely cilinderétől kerül elhelyezésre, az abszolút cím is meghatározható.

A cím kiszámításának többféle algoritmusa létezik, alkalmazásukat több tényező befolyásolja. Így elsősorban a logikai rekordazonosítók eloszlása. A legegyszerűbb eset az, amikor a rekordazonosítók többé-kevésbé egyenletesen oszlanak meg egy intervallumban, és számuk megegyezik a részükre kijelölt helyek számával. Ennek az esetnek az az előnye is megvan, hogy ilyenkor nem jön létre túlcsordulás.

Bonyolultabb esetekben, pl. amikor az azonosítók nagyobb intervallumban helyezkednek el, mint a fizikai helyek címei, más leképzési algoritmusokat kell használni. Ilyenkor túlcsordulás is keletkezhet, vagyis előfordulhat, hogy ugyanarra a fizikai címre több rekordazonosítóból is leképzés történik. A túlcsordulás kezelése ezt a problémát oldja fel.

13

Az adatbázisokról

A különböző fájlszerkezetek igen alkalmasak arra, hogy bizonyos feladatok megoldásához hatékony adatkezelést biztosítsanak. A már bemutatott indexelt szekvenciális állomány esetében pl. láthattuk, hogy minden esetben, amikor a feldolgozás általában szekvenciális, de a rekordok egy részét (kis részét) közvetlenül is el kell érni, jó hatékonysággal alkalmazható. Mindaddig, amíg a számítógépek háttértárainak kapacitása korlátozó tényező volt, a számítógépet egy-egy gazdasági vagy műszaki részterület problémáinak megoldására használták. Így jöttek létre pl. egy vállalaton belül

— a bérelszámolással,
— a munkaüggyel,
— a raktárnyilvántartással,
— a termelésprogramozással

kapcsolatos számítógépes alrendszerek, amelyek elkészítése sokszor időben sem esett egybe. Részben ezért, részben más okokból nem is kapcsolódtak egymáshoz. Ez elsősorban akkor okozott gondot, amikor ugyanarra az adatra több alrendszernek is szüksége volt, s ezeket az adatokat mindegyik alrendszer saját maga kezelte. Ezzel nagy mennyiségű adathalmaz duplán vagy akár többszörösen került tárolásra. Ez azt is jelentette, hogy ha ilyen adat változott, minden alrendszerben át kellett vezetni ezeket a változásokat.

A harmadik generációs számítógépekkel megjelenő nagy teljesítményű mágneslemeztárak megteremtették a lehetőségét annak, hogy létrejöhessenek e probléma megoldásaként az *adatbázisok*.

Mi az adatbázis, megjelenése miért jelent minőségi változást a korábbi alkalmazásokkal szemben, tehetjük fel rögtön a kérdést.

Az adatbázis olyan egymással kapcsolatban álló adatok összessége, amelyet a különböző felhasználók különböző szempontból, más-más célra használhatnak. Az adatok fizikai elhelyezése központilag történik, oly módon, hogy az azonos tartalmú adatok lehetőleg egyszer kerüljenek tárolásra. Az adatbázisban levő adatok változtatása, új adatok felvitele és az egész adatbázis védelme szintén központilag történik. Az adatbázishoz tartozó adatbázis-kezelő rendszer — amely egy speciális szoftver — biztosítja a különböző felhasználóknak az adatbázishoz való hozzáférését. A felhasználói programok a tényleges tárolási módtól függetlenek lehetnek.

Ebből a megfogalmazásból kitűnik, hogy az adatbázisnak milyen előnyei vannak a hagyományos fájlszervezéssel szemben:

— a felhasználók különféle igényeit is képes kielégíteni, míg egy-egy fájlszerkezet csak a létrehozáskor kiemelt szempont szerinti lekérdezéskor biztosít hatékony elérést;

— az adatbázisban levő adatok elhelyezkedése teljes mértékben független a felhasználói program adatigényétől, míg a fájlszerkezeteknél a feldolgozó programnak ismernie kell az adott fájl rekordszerkezetét;

— az adatbázis használatát segíti egy adatkezelő nyelv. Ennek a segítségével lehet műveleteket végezni az adatbázison;

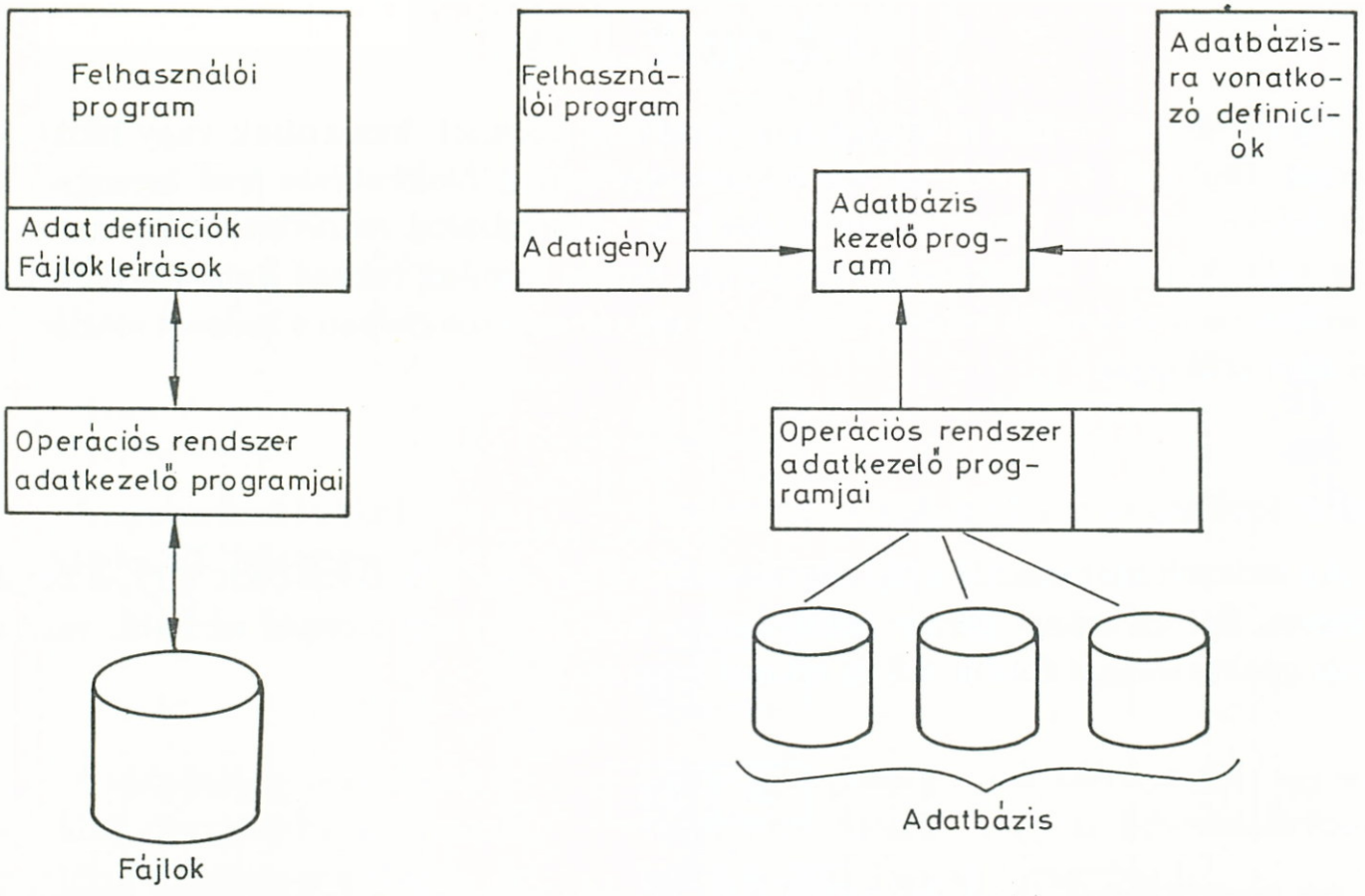
— az adatbázisban levő adatok védelme megoldott. A védelemnek több szintje van, vagyis bizonyos adathoz csak azok a felhasználók férhetnek hozzá, akiknek erre jogosultságuk van.

Az adatbázisok természetesen akkor használhatók fel sikeresen, ha azok a valóságnak megfelelő adatokat tartalmaznak. Ugyanakkor az adatbázisok meglehetősen bonyolult szerkezetek, s ezért nem célszerű sem létrehozásukat, sem naprakészen tartásukat magára a felhasználóra bízni. A már említett adatvédelem problematikája pedig egyenesen megkívánja, hogy egy külön „megbízható" és természetesen hozzáértő személy (vagy személyek csoportja) végezze el ezeket a tevékenységeket.

Azt a szakembert, aki az adatbázis kialakítását, adatokkal való feltöltését, az üzemeltetési szabályoknak megfelelő mentéseket, archiválásokat végzi, adatbázis felügyelőnek vagy adatbázis felelősnek nevezzük. Az ő feladatát képezi a felhasználók segítése az adatbázis használatában.

A ma létező adatbázisok és adatbázis kezelő rendszerek több csoportot alkotnak, elsősorban attól függően, hogy milyen logikai kapcsolatok léteznek az egyes adatok között. Ezekkel azonban nem foglalkozunk.

A hagyományos fájlszerkezetek és az adatbázis alkalmazásának összevetését szemlélteti az A—27. ábra, amely az elmondottakat némileg illusztrálja is.



A-27. ábra. A „hagyományos" fájl kezelési módszer és az adatbázis használatának sematikus vázlata

⊙ 1

Az egyszerű (vagy alapvető) fájlszerkezetek közé soroljuk a következő fájlszerkezeteket:

— soros vagy szeriális,
— szekvenciális,
— random (közvetett címzésű),
— direkt (közvetlen címzésű).

A komplex fájlszerkezetek közé tartozik:

— az indexelt szekvenciális fájl,
— a listaszerkezet,
— az invertált fájl,
— a particionált (könyvtári) fájl.

⊙	6

Az elsődleges terület adatsávjain a rekordok lehetnek blokkoltak vagy blokkolatlanok. Blokkolt ábrázolás esetén a kulcsmezőben a *blokkokban levő legnagyobb kulcs* értéke szerepel. Ugyanakkor a blokkon belül minden rekordnak tartalmaznia kell a saját kulcsát. Blokkolatlan ábrázolás esetén minden rekord kulcsa szerepel a rekordhoz tartozó (saját) kulcsterületen, így magában a rekordban a kulcsot megismételni nem szükséges.

⊙	7

Az indexelt szekvenciális fájl létrehozásakor egy-egy adatsávnak kb. 60%-a kerül kitöltésre. Ezt az arányt persze befolyásolja, hogy mekkora mozgás várható, vagyis milyen gyakorisággal fordul elő új rekord belépése.

⊙	11

Az algoritmusok tervezése során (III. fejezet) tanultuk a funkcionális lebontás elvét. Ennek az elvnek a programozáskor való következetes végigvitele azt eredményezi, hogy a feladatmegoldó programban minden funkciót önálló alprogrammal valósítunk meg. Ilyen megoldási technika mellett egy feladat megoldása egy főprogramból és sok (a főprogramból hívott) alprogramból áll. Amikor az így megírt rendszert futtatni akarjuk és könyvtárakat használunk, a megoldás menete a következő:

— az egyes alprogramokat külön-külön fordítjuk le, elhelyezzük a lefordított változatokat a tárgynyelvi könyvtárban;

— a főprogramot lefordítjuk és egyben szerkesztjük, megadva a tárgynyelvi könyvtár nevét a kapcsolatszerkesztő számára.

Ilyenkor a főprogram fordításra kerül, majd a szerkesztési fázisban a kapcsolatszerkesztő a megadott tárgynyelvi könyvtárból hozzászerkeszti a kért alprogramokat a főprogramhoz. Ezzel előáll a futtatható program.

Ez a munkamódszer elsősorban akkor célravezető, ha a programfejlesztési munkákat terminál mellől végezzük.

Amennyiben a tesztelés során valamelyik részben javítani kell, mindig csak az érintett alprogram és a főprogram újrafordítása szükséges.

⊙	12

Teljesen kitesztelt állapotban a programot egy betölthető programokat tartalmazó könyvtárba lehet tenni, s ettől kezdve csak a futási időigény merül fel, valahányszor végre akarjuk hajtatni.

IX. fejezet

▲ 2

A nyelvi fordítókkal, vagyis a fordítóprogramokkal, azok kialakulásával és funkcióikkal a VI. fejezetben már találkoztunk.

▲ 4

A számítógép *átbocsátóképessége* a teljesítőképesség egyik mérőszáma: egy adott idő alatt elvégzett összes munka mennyisége. Nyilvánvaló, hogy az átbocsátóképesség mértékét a leglassabb részek nagyban befolyásolják (negatív irányban).

A teljesítőképesség más mérőszáma lehet még:

— a *rendelkezésre állás*, vagyis a rendszer üzembiztossága. Hiába gyors ui. egy számítógép, ha állandóan javítani kell (vagy nagyon gyakran kell leállítani pl. hardver hibák miatt);

— *a válaszidő vagy fordulási idő*, ami azt az időt jelenti, amíg a felhasználónak a programjára várakozni kell.

▲ 7

A futtatandó program tárolási helye egy könyvtár, mégpedig egy futtatható programok számára létrehozott könyvtár.

Mint később látni fogjuk, meg kell adni a program nevét és (szükség esetén) a programot tartalmazó könyvtár nevét, valamint azt a lemeznevet, amelyen a könyvtár van.

Az adatfájl azonosításához meg kell adni a fájl nevét, adathordozóját (mágnesszalag, mágneslemez), szervezettségét (szekvenciális, index szekvenciális stb.), s még további információkat, amint a későbbiekben ezt is látni fogjuk.

▲ 12

Az SVC megszakítás megértéséhez tekintsük a következő helyzetet: három program fut a multiprogramozott környezetben: A, B és C. Legyen A elsőbbsége a

legnagyobb, C elsőbbsége a legkisebb stb. A vezérlés az A programnál van, és egy adott pillanatban inputra van szüksége. Adatátvitelt azonban csak a supervisor bonyolíthat. Ezért a felhasználói programban egy supervisorhívás, egy SVC megszakítás történik. Amíg az A adatigénye nincs kielégítve, A-t nem lehet végrehajthatónak tekinteni. Ezért a vezérlést B program kapja meg. B dolgozik, de közben (tegyük fel) az előző adatkérés végbement, a kért rekord bent van a tárban. Ekkor — mivel A végrehajtható, hiszen nincs erőforrás „hiánya" — A-nak vissza kellene kapni a vezérlést. Ez úgy történik, hogy a csatorna az adatátvitel befejezése után jelzést ad a supervisornak. E jelzés alapján egy input-output megszakítás megszakítja B futását, s a supervisor A-nak adja vissza a vezérlést. (Annak ellenére, hogy B futhatna tovább.)

▲ 13

Az elszámolási információknak ott van fontos szerepe, ahol gépidőeladás történik, vagyis ahol az elhasznált erőforrásokért fizetni kell. A számítógép egy erre a célra készített programja minden programfutásnál feljegyzi egy fájlba, hogy az adott program mely erőforrásból mennyit használt el, ill. mennyi ideig használta az adott erőforrást.

Ebből a fájlból időnként egy számlázóprogram a megrendelők számára számlát készít.

▲ 18

Ilyen eset fordulhat elő pl. akkor, ha a jobnak egy mágnesszalagra van szüksége, de ezt a szalagot egy másik partícióban futó program használja.

Hasonló eset, ha egy mágneslemezen levő könyvtárat egy job nem a DISP=SHR (megosztható) paraméterrel kötött le, hanem pl. DISP=OLD megadással. Ez utóbbi azt eredményezi, hogy mindaddig, amíg a DISP=OLD paramétert tartalmazó job le nem fut, másik job nem tud hozzáférni az adott könyvtárhoz.

▲ 20

Minden feldolgozóprogram futásakor keletkezik egy ún. *állapotkód*, amely a programot tartalmazó job lépés befejezéséről tájékoztat. A hibátlan befejezést a 0 értékű állapotkód jelzi. Minden feldolgozóprogramnál megadják a lehetséges állapotkódértékeket és azok jelentését.

A felhasználónak is van lehetősége, hogy saját programjaihoz állapotkódokat rendeljen.

▲ 21

„Az időbeni átfedéssel létrejövő outputok" problémáját a következő példából érthetjük meg. Multiprogramozásos környezetben a P1 és a P2 partícióban fut egy-egy program. Mindkettő nyomtatni akar a sornyomtatóra, s tételezzük fel, hogy a rendszer ezt lehetővé is teszi számára. Mindkét program egy-egy Petőfi-verset kíván kinyomtatni, de miután a multiprogramozás miatt hol az egyik, hol a másik működik, nyilván a két program által küldött kétféle szöveg is felváltva érkezik a nyomtatóra, s az eredmény valami ilyesmi lenne:

Reszket a bokor, mert
Falu végén kurta kocsma,
Madárka szállott rá,
Oda rúg ki a Szamosra,
Reszket a lelkem, mert
Meg is látná magát benne,
Eszembe jutottál,
Ha az éj nem közelegne.

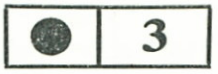
Hasonló problémák adódnának, ha a kártyaolvasót is egyidejűleg használhatná több program.

● 1

Az első fejezetben a szoftver fogalomkörébe tartozó programok csoportosításakor láttuk, hogy vannak rendszerprogramok és alkalmazási programok. A rendszerprogramokon belül megkülönböztettük a vezérlő- és a feldolgozóprogramokat.

Nem alakult ki egységes szemlélet arra vonatkozóan, hogy az operációs rendszer fogalmába csak a vezérlőprogramok tartoznak-e, vagy a teljes rendszerszoftver. (A kettő közötti különbség a fordítók és szervizprogramok köre.)

Ezért szokás „szűkebben értelmezett" operációs rendszerről beszélni, s ilyen értelemben csak a vezérlőprogramokat ide sorolni, és „tágabb értelemben vett" operációs rendszert emlegetni, amely a fordítóprogramok lehetőségeit is ide sorolja. A továbbiakban — miután a fordító- és szerkesztőprogramokkal már megismerkedtünk — a vezérlőprogramok szolgáltatásait tekintjük operációs rendszer funkcióknak.



Kísérjük nyomon egy egyszerű példán, hogy milyen tevékenységek adódtak a gépteremben egy program futtatása során, amikor *minden tennivaló az operátorra hárult.*

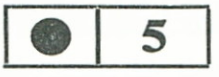
A felhasználó készített egy programot (pl. COBOL nyelven), és ezt lyukkártyára lyukasztotta. Az így keletkezett kártyacsomagot átadta az operátornak, aki megállapította, hogy a program COBOL-ban íródott, vagyis futtatásához a COBOL fordítóprogramra van szükség. Megkereste a fordítóprogramot (amely lyukkártyán vagy esetleg mágnesszalagon található), bekapcsolta a megfelelő perifériát (kártyaolvasót vagy szalagegységet) és a fordítóprogramot a tárba töltötte.

Ezt követően a kártyaolvasón be lehetett olvastatni a felhasználó programját, amelyet a fordítóprogram lefordított és mágnesszalagra írva vagy lyukkártyára lyukasztva visszaadta a tárgyprogramot. Az így visszakapott tárgyprogramot — feltéve, ha hibátlan, de ezt előzőleg ellenőrizni kellett — a betöltőprogramnak kellett feldolgoznia.

E célból most a betöltőprogram bevitele következett a fordítóprogram bevitelével hasonló módon. A tárgyprogramot is vissza kellett olvastatni a tárba. A betöltő a tárgyprogramot futásra alkalmassá tette (szerkesztés). A program futását is az operátor indította el, miután meggyőződött arról, hogy a szükséges perifériák be vannak kapcsolva, és a feldolgozandó adatok a kártyaolvasóban vannak.

A felsorolt tevékenységek kezdési és befejezési idejét a számítógép naplójába jegyezte be.

A bemutatott példa talán jól szemlélteti, hogy mennyire nehézkes és bonyolult tevékenységsorozat volt, amelyet az operátornak hibátlanul és főleg gyorsan kellett elvégezni ahhoz, hogy egyetlen programot a számítógép végrehajtson. Persze az operátor bármilyen gyorsan is dolgozott, az a számítógép számára nagyon is lassú volt, szinte minden tevékenységre várakoznia kellett a gépnek.



Vizsgáljuk meg, hogy az operációs rendszer megjelenése mennyiben változtatta meg az operátori feladatkört és milyen tennivalókat végez el az operációs rendszer az operátor helyett.

Az operátor a felhasználó programját — más programokkal együtt — elhelyezi a kártyaolvasóban, és azt bekapcsolja. Ellenőrzi, hogy a többi periféria is bekapcsolt állapotban van-e, majd a konzolról elindítja a már korábban betöltött operációs rendszer futását. Ezután figyelemmel kíséri a gép működését.

Az operációs rendszer a kártyaolvasón keresztül beérkező kártyafolyamban „észreveszi" azokat a speciális kártyákat, amelyek minden programra vonatkozóan vezérlő információkat tartalmaznak, s ezek alapján automatikusan meghívja a szükséges

fordítóprogramot. Elindítja a fordítást, és ha ennek során a forrásprogramról kiderül, hogy nem hibátlan, be is fejezi a tevékenységet. Gondoskodik arról, hogy a fordítás eredménye listára kerüljön, majd a soron következő programmal kezd el foglalkozni.

Hibátlan fordítás esetén lehetségessé válik a tárgyprogram további feldolgozása. A fordítóprogram hívásához hasonlóan aktivizálásra kerül a kapcsolatszerkesztő, amely a szintén automatikusan megnyitott könyvtárakból elvégzi a futtatható program szerkesztését. Hibátlan szerkesztés után kezdődhet a program végrehajtása.

Ellenőrzi, hogy a kért mágnesszalagok és mágneslemezek kerültek-e fel a meghajtókra, s tévedés esetén értesíti az operátort. (Ugyancsak üzenetet küld az operátornak, ha egy szükséges szalag vagy lemezcsomag nincs feltéve.)

Bizonyos hibákat megpróbál korrigálni (pl. 0 osztáskor), ha ez nem sikerül, olyan kódot generál, amely segíti a programozót a hiba megtalálásában.

Figyelemmel kíséri a géphez kapcsolt terminálokon folyó munkákat. A terminálról érkező jobot — hasonlóan a kártyaolvasóról érkezetthez — lépésenként feldolgoztatja a megfelelő fordítóprogrammal, ill. a kapcsolatszerkesztővel.

Segíti a terminálnál ülő felhasználót, hogy tájékozódhassék a rendszerben folyó munkáról.

Kapcsolatot tart az operátorral a konzolon keresztül. Ugyancsak innen küldhet az operátor parancsokat a rendszernek (pl. egy végtelen ciklusban levő program „kilövésére", futásának megszüntetésére).

● 6

Több olyan operációs rendszer létezik, amely az említett cél érdekében üzemel. Ezek sokszor egymástól eltérő módon valósítják meg azokat a feladatokat, amelyeket a számítógép hatékony működése érdekében el kell látniok. A hazai nagyszámítógépeken létező operációs rendszerek között legelterjedtebb a DOS (Disc Operating System=lemez operációs rendszer) és az OS különböző változatai: az OS/MFT, az OS/MVT és az OS/VS.

A továbbiakban, ha erre külön utalás nincs, az OS/MFT változatra vonatkoznak az elmondottak. (MFT=multiprogramozás fix számú feladattal, kötött méretű partíciókban.)

● 8

A számítástechnikában a „kis" és a „nagy" fogalmával vigyázni kell. Léteznek olyan számítógépek, amelyek többre képesek, mint tíz éve gyártott „nagy" berendezések.

A multiprogramozás lehetősége elsősorban (de nem kizárólag) operációs rendszer kérdése. Vannak olyan számítógépek (pl. R 11), amelyek 1 Mbájt központi tár mellett

sem tudnak kötegelt munkákat multiprogramozva feldolgozni. Mégis az a jellemző, hogy egy számítógép minél több erőforrással bír, s minél nagyobb felhasználói létszámot szolgál ki, annál valószínűbb, hogy multiprogramozható.

● 10

Tegyük fel, hogy a P1 partíció ürült ki, és ebbe a partícióba kell munkát keresni és ütemezni. A partícióhoz tartozó job osztályok legyenek B, C, A.

Az ütemező először a B osztályú jobok várakozó sorában keres munkát. Ha talál, a soron következőt ütemezi. Ha a B osztályú jobok várakozó sora üres, a C osztályú várakozó sorhoz fordul. Ha ebben sem talál munkát, az A osztályú várakozó sort vizsgálja. (Ha ez is üres lenne, a P1 partícióba mindaddig nem lehet munkát ütemezni, amíg kívülről — pl. a kártyaolvasóról — nem érkezik B, C vagy A osztályú job, ill. amíg az operátor meg nem változtatja a P1 partícióhoz tartozó job osztályokat.)

● 15

A kétféle elnevezésnek, hogy tudniillik egy állomány más nevet visel a programban, és más nevet a rendszerben, több oka van. Eleve vannak programozási nyelvek, amelyeknek az állományok elnevezésére vonatkozó szabálya nem egyezik meg az operációs rendszerbeli elnevezési szabályokkal. Pl. a FORTRAN csak numerikus tartalmú fájl neveket enged meg, míg az operációs rendszerbeli névnek hat karakteres alfanumerikus szimbólumnak kell lennie.

Másrészt e kétféle elnevezés biztosítja a programozó számára a rendszertől való függetlenséget: nem kell tudnia, hogy egy adott állomány melyik szalagon vagy lemezen helyezkedik el, az állomány azonosítását ettől függetlenül elvégezheti.

Harmadrészt — gondoljunk csak a már tanult fájlfelújítás esetére — a mindenkori törzsfájl konkrét adathordozója (szalagja) hónapról-hónapra változik, maga a fájl azonban a feladat logikája szerint mindig a törzsfájl.

● 17

A DSN=MIKI50 hivatkozás alapján — mivel a könyvtár helyére vonatkozó információ nincs a DD kártyán — a rendszer a katalógusban keresi a MIKI50 nevű fájlra vonatkozó információkat. (Ha nem találja, ezt hibaüzenetként jelzi és a végrehajtás abbamarad.) A katalógusból kiderül, hogy a keresett fájl pl. a BALI40 nevű lemezen van. A következő lépésben a BALI40 lemez VTOC-jából kell megkeresni a könyvtár lemezen belüli helyét. Utolsó lépésként a megtalált könyvtár tartalomjegyzékéből olvasható ki, hogy a futtatni kívánt program a könyvtáron belül hol van.

Az adatállományról való rendelkezés, mint arra példát is láttunk, a DISP paraméterrel történik a DD utasításban.

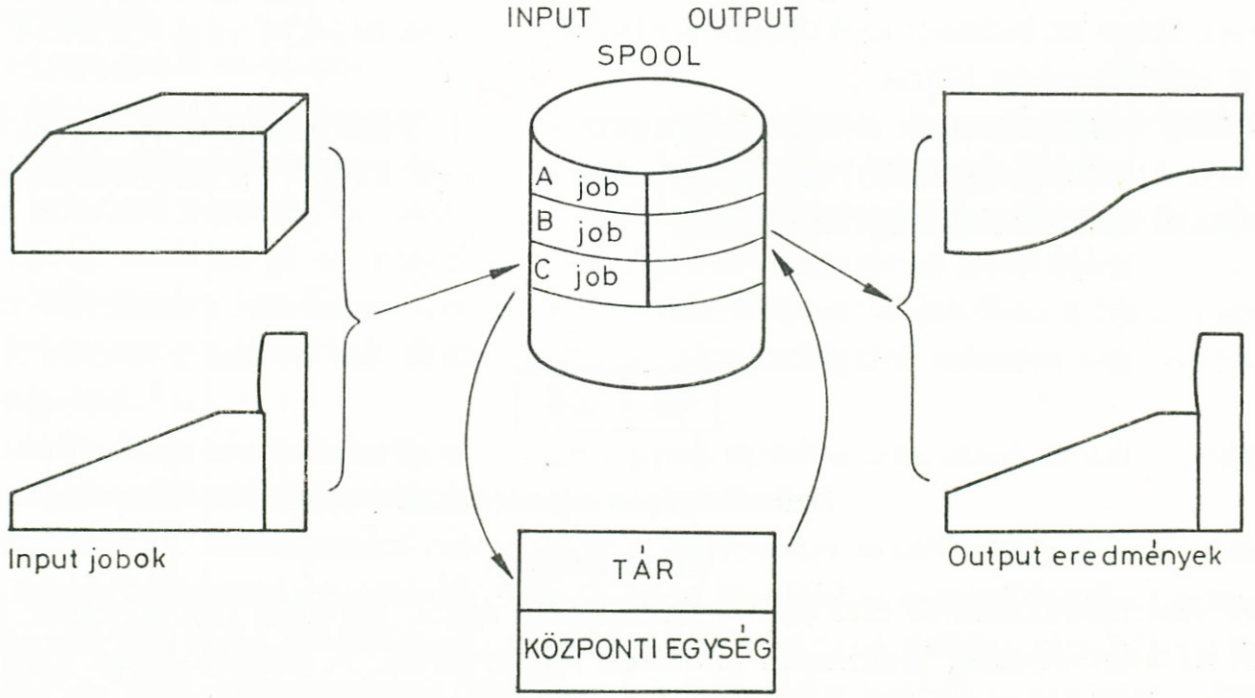
A fontosabb előírások:
törölni az állományt,
megőrizni az állományt,
továbbadni az állományt a következő job lépésnek.

Az input-output perifériák kezelésére kialakított egyik megoldás a *spool* (szpul) *módszer*. Nevét a „perifériák egymás melletti közvetlen használata" kifejezés angol rövidítéséből kapta.

A spool módszer lényege, hogy minden munka számára a rendszer mágneslemezen kijelöl egy input és egy output munkaterületet. Az input munkaterületre kerül a job tartalmából minden, kivéve a munkavezérlő utasításokat. (Tehát, ha pl. a job tartalmazza a forrásnyelvi programot és a program futásához szükséges adatokat, akkor mindkettő a job input munkaterületére kerül.) Az output munkaterületre íródnak azok a rekordok, amelyeket a felhasználó a sornyomtatóra szánt. Az input spoolba az olvasó értelmező írja fel a rekordokat, az output spool tartalmát pedig az output kiíró írja ki a program futása után.

A program futásakor a kártyaolvasóra vagy a terminál billentyűzetére vonatkozó olvasási utasítások az input spoolból veszik a rekordokat, a nyomtatásra vonatkozó utasítások hatására pedig a kiírandó rekord az output spoolba kerül.

Az elmondottakat az A-28. ábra szemlélteti.



A-28. ábra. A spool-módszer lényege

● 24

Az időlimit (időhatár) védi a rendszert a végtelen ciklusba kerülő programok „végtelen ideig" való futásától. Ha egy program számára pl. 2 perc központi egységhasználati idő van megengedve, ennek elteltével a futás abnormálisan befejeződik, és a felhasználó üzenetet kap az időtúllépésről.

Ilyenkor el kell dönteni, hogy a 2 perc valóban kevés volt-e a feladat megoldásához vagy pedig a program nem úgy működik, mint ahogyan azt a programozó elképzeli.

A végtelen ciklus egyik lehetséges esetét a III. fejezethez tartozó 17. annotációban bemutattuk (240. oldal). Itt felsorolunk néhány olyan okot, amely végtelen ciklust eredményezhet:

— ciklusból adott feltétel teljesülésekor visszaugrunk a ciklus elejére anélkül, hogy a ciklus befejezésére vonatkozó feltételt vizsgálnánk;

— olyan feltételt adunk a ciklusból való kiugrásra, ami sohasem teljesül;

— hibás hivatkozásokkal, pl. a tömbnek nem létező elemére való hivatkozással, ill. oda való értékadással illegálisan felülírjuk a programhoz tartozó valamelyik területet. Ez lehet egy adatterület, amely pl. a ciklusváltozónkat elrontja vagy lehet programterület, amitől a program vezérlési szerkezete megváltozik, és ez okoz végtelen ciklust.

● 26

Az elérési mód megléte a szoftver feltétele annak, hogy a hozzá tartozó fájlszerkezetet használni lehessen. Így pl. indexelt szekvenciális fájlok kezeléséhez nem elégséges az a tény, hogy a számítógép rendelkezik mágneslemez egységekkel, arra is szükség van, hogy az indexelt szekvenciális elérést biztosító BISAM vagy QISAM elérési kód is rendelkezésre álljon.

(Nem minden magas szintű programozási nyelv teszi lehetővé az összes elérési mód közvetlenül programból való alkalmazását. Így pl. a FORTRAN nem támogatja az indexelt szekvenciális fájl kezelését.)

● 28

Számítógépes adatvédelem

(Olvasmány)

A számítástechnikai eszközök alkalmazása egyik fejlődési fokozatának tekinthetjük az adatvédelmi, biztonsági kérdések felmerülését. A számítógépes bűnesetek, ill. visszaélések megakadályozásának, valamint a személyi információk védelmének

jogos igénye a fejlettebb számítástechnikai kultúrával rendelkező országokban a 60-as években, nálunk 1981-ben eredményezték az adat- és információvédelemmel, pontosabban a számítógépes rendszerek titok-, vagyon- és tűzvédelmével kapcsolatos szabályozások kibocsátását. Ennek szükségességét és időszerűségét számos példa igazolja. Egy becslés szerint a brit nemzetgazdaság a számítógépes visszaélések következtében évente annyit veszít, mint amennyi megfelel az évszázad négy legnagyobb rablótámadásának.

Amerikai forrásokból azt is megtudhatjuk, hogy az Egyesült Államokban 1976-ban az évi veszteségek 100 millió dollárt tettek ki, és évi 400%-os ütemben nőttek. De az anyagi haszonszerzéssel kapcsolatos visszaéléseken kívül számos példát ismerünk arra, hogy vállalati titkokat ipari kémek szereztek meg, személyekkel kapcsolatos bizalmas információk kerültek illetéktelenek kezébe, és szándékos rongálásokat is feljegyzett már a szakirodalom.

Az „*adatvédelem*" összefoglaló név igen nagy területet takar. Ebbe a fogalomkörbe tartozik a számítóközpontok biztonsági őrzése, az elemi csapásokra való felkészülés mellett a számítástechnikai rendszerek előkészítésétől a megvalósításig és működtetésig terjedő folyamat zárttá tétele: a titkos információk kiszivárgásának megakadályozása; és ide tartozik maga a tényleges adat- és információvédelem is. Az előbbiek főként csak a számítóközpontok vezetői számára érdekesek, míg az utóbbiak a számítástechnikát alkalmazók (szervezők, programozók, üzemeltetők stb.) széles körét is foglalkoztatják, ezért a témakörnek ezt a részét mutatjuk be.

A magyar szabályozás a számítógép-üzemeltetőket több kategóriába sorolja. Mindenhol a kategóriának megfelelő mértékű adatvédelmi intézkedéseket kell meghozni, ill. betartani.

Az adat- és titokvédelemmel kapcsolatosan más-más feladatok és lehetőségek vannak az adatrögzítés, a kötegelt- és távadatfeldolgozás, valamint a kiszállítás és a tárolás területein.

Az adatrögzítés területén:

a) a bizonylatok teljességének megőrzése (intézkedések bizonylatvesztés ellen), amire pl. a bizonylatok kötegelése, számozása, leszámolása lehet alkalmas;

b) a tartalmi teljesség és pontosság megőrzése (intézkedések adatvesztés, adattorzulás ellen), amelyet pl. az ellenőrző adatrögzítés segítségével lehet elérni;

c) az illetéktelen betekintés megakadályozása, aminek érdekében a bizonylatok személyek közötti átadását kell zárttá tenni, ill. az adatrögzítő terembe való belépést kell szabályozni.

Az adatfeldolgozás technológiai folyamatának minden szakaszán az adatvédelem következő szintjeit különböztethetjük meg:

a) Intézkedési, szabályozási szint: azok a biztonsági rendszabályok, előírások, amelyek betartása esetén megakadályozható vagy legalábbis megszűrhető az adatok illetéktelen elérése. Ilyen lehetőségek: a védendő állományok külön, zárt helyen tartása és csak az arra jogosultaknak kiadása; a gépterembe belépő személyek szűrése és ezek feljegyzése, amennyiben titkos állományokkal folyik a munka; a feldolgozási

fázisban a technológiai folyamat megszervezése oly módon, hogy a jobok beadásától az eredmények átvételéig a programok és az adatok útja szabályozottan nyomon követhető legyen: időosztásos rendszerekben a terminálok bejelentkezési idejének meghatározása stb.

b) Mechanikus védelmi szint: amelyet a hardver eszközök biztosítanak. Ezek gépenként különbözőek. Legegyszerűbb példái: a mágnesszalagok írógyűrűjének eltávolítása a szándékos vagy véletlen felülírások megakadályozására; hardver védelmi eszközök beépítése (pl. amely a központi tárat a job befejezése után automatikusan törli); távadatfeldolgozás esetén a fizikai hozzáférés ellenőrzése (valóban arról a terminálról jelentkeztek-e be), a kábelek ellenőrzése a lehallgatási jellemzők tekintetében; a vonali csatornákon az információk vagy azok egy részének titkosítása stb.

c) Programszintű védelem: amelynél a felhasználói programok biztosítják a védelmet. Ezek az adatvédelem három formájában nyilvánulhatnak meg:

1. Megelőző adatvédelem, amely meggátolja az adatállományhoz való illetéktelen hozzáférést. Jelszó vagy kulcs segítségével megállapítja, hogy ki és milyen mértékben, inputra, outputra vagy mindkettőre használhatja-e az állományt és illetéktelenség esetén a jobot kizárja (terminál esetén a vonalat bontja), munkaszámát, azonosítóját feljegyzi.

2. A regisztráló adatvédelem ugyan nem védi az adatállományt, de feljegyzi az állományhoz való hozzáfordulásokat (ki, milyen mértékben, milyen módon használta), és ezt a számítóközpont vezetése folyamatosan figyeli. Léteznek már ilyen regisztráló adatvédelmi lehetőségeket biztosító rendszerek, de házilag is beépíthetők a meglevő rendszerbe, ill. készíthetők ilyen programok.

3. A vegyes forma az előző kettő együttes alkalmazása.

d) A *rendszerszintű* adatvédelmi eszközöket az alkalmazott operációs rendszer tartalmazza. Ezeknek az eszközöknek a felosztása megegyezik a programszintű adatvédelmi eszközökével, és tulajdonképpen ugyanazokat a funkciókat látják el. A különbség csak annyi, hogy felhasználói programok helyett rendszerprogramok oldják meg az említett feladatokat. Jellemző példaként az adatbázis-kezelő rendszerek védelmi eszközei, ill. távadatfeldolgozásnál a terminálok jogosulatlan használatát megakadályozó, valamint a távadatátviteli vonalak lehallgatása elleni védelmet biztosító módszerek említhetők.

A feldolgozás befejezése után különösen fontos, hogy a védendő eredményeket tartalmazó listák ne kerüljenek illetéktelen kezekbe. Ezt legjobban az biztosítja, ha a diszpécser az eredménytáblákat személyesen adja át a felhasználónak vagy zárt, lepecsételt küldeményben továbbítják.

A feldolgozáshoz szorosan kapcsolódik az adatállományok archiválása, mentési rendszerének kidolgozása. Hogy egyáltalán melyik állományról, hány példányban, milyen gyakorisággal készüljön másolat, az az állomány fontosságától és rendszerben elfoglalt szerepétől függ. Adatvédelmi szempontból különösen érdekes az a kérdés, hogy a katasztrófaterv alapján háttér-adattárba (földrajzilag távoli helyre) elhelyezett állományok aktualizálása milyen gyakoriságú legyen. Az archív másolatok felhasz-

nálása csak rendkívül indokolt esetben, és csak szigorú biztonsági előírások betartása (pl. az archív kópia előzetes átmásolása) után lehetséges. A mentési, archiválási rendszer és az ehhez kapcsolódó kérdések egyébként külön tanulmány tárgyát képezhetik.

Az előbbiekben az adatvédelemmel kapcsolatos területeknek csak kis részét érintettük. Az adatvédelmi kérdések a számítástechnika minden területén minden fázisában jelen vannak. Természetesen nem mindenkit érintenek egyformán: a szabályozó intézkedéseket kinek-kinek a rá vonatkozó mértékben kell figyelembe vennie. Az alapintézkedés az 1/1981. (I. 27.) BM számú rendelet; ehhez kapcsolódnak a minisztériumok által készített végrehajtási utasítások, és jó, ha mindenki ismeri saját vállalata, intézete adatvédelmi előírásait.

⊙	9

A dinamikus terület partíciókra való felosztását a rendszer generálásakor kell elvégezni. Ez a rendszerprogramozók feladata. A partíciók generálásakor meg kell adni minden partíció méretét (Kbájtban), és a partícióhoz rendelt job osztályokat.

A legkisebb partícióméret 8 Kbájt, a legnagyobbat a dinamikus terület mérete befolyásolja, hiszen ezt kell felosztani.

Az operátornak — ha üres a tár, vagyis minden partíció üres — lehetősége van arra, hogy a partíciófelosztást megváltoztassa a konzolról bevitt paranccsal.

⊙	11

Virtuális tár

A virtuális társzervezés a laptechnikán alapuló társzervezés egyik fejlett változata. Az elnevezésben a „virtuális" (látszólagos) szó arra utal, hogy ilyen társzervezés mellett a felhasználó a valóságosan létező tárméretnél nagyobb (sokkal nagyobb) tárat feltételezhet és használhat a programjában. A felhasználói program ui. nincs teljes terjedelmével a tárban. A programot a rendszer ún. lapokra bontja szét (egy ilyen lap mérete 512—4096 bájt lehet), és a tárat ugyanilyen méretű blokkokra osztja. Bármely lap bármely blokkban elhelyezhető.

Az a lap, vagy azok a lapok a tárban tartózkodnak, amely (vagy amelyek) a programvégrehajtás adott pillanatában szükségesek. A többi lap lehet a tárban vagy mágneslemezen.

Ez a módszer természetesen több hardver problémát is felvet. Így pl. szükség van olyan táblázatra, amelynek segítségével a programbeli címhivatkozások tényleges fizikai címekké alakíthatók. Ehhez speciális regiszterekre és külön tárterületre van

szükség. Azok a lapok, amelyek nincsenek a tárban, mágneslemezen helyezkednek el. Nagy előnye a rendszernek, hogy a tárban levő lapok egymástól fizikailag távol levő blokkokban is elhelyezhetők, vagyis nem kell több blokknyi egybefüggő területnek rendelkezésre állnia.

Amikor egy olyan lapra van szükség, amely az adott pillanatban nincs a tárban, gondoskodni kell annak behozataláról. Ilyenkor egy speciális megszakítás lép fel, módot adva az operációs rendszernek a közbeavatkozásra. A virtuális tárakat támogató operációs rendszer (pl. az OS/VS supervisor programja) gondoskodik a szükséges új lap behozataláról.

Külön problémát jelent, hogy az új lap melyik blokkba, vagyis melyik bentlevő lap helyére kerüljön. Erre több megoldás létezik. Pl. az egyik esetben a rendszer figyeli, hogy a tárban levő lapok közül melyik hányszor volt használva, és a legritkábban használt lap helyére hozza be az új lapot. A virtuális tárkezeléssel megoldott multiprogramozás ily módon dinamikussá teszi a tár használatát, lényegesen megnövelve a rendszer átbocsátóképességét. Ugyanakkor — mint az elmondottakból is kitűnik — rendkívül bonyolult tevékenységeket követel meg a számítógéptől. Alkalmazása ezért csak nagyteljesítményű, gyors gépek esetén célravezető.

⊙ 14

A kapcsolatszerkesztő által létrehozott ideiglenes könyvtárból való programhívás:

//GO EXEC PGM=*LKED. SYSLMOD, COND=(4, LT, FORT), (4, LT, LKED)

Magyarázat:

GO a „job" lépés neve (futtatási fázis)

* .LKED. SYSLMOD az aktivizálandó program nem könyvtárból kerül aktivizálásra, hanem a „job" egy korábbi lépésében hoztuk létre. A munkaterületen levő állomány azonosítása

* .ugyanebben a job-ban, az LKED nevű job lépésben levő SYSLMOD nevű DD állomány.

Természetesen ez az állomány egy futtatható programot kell, hogy tartalmazzon.

COND az EXEC utasítás kulcsszavas paraméterei közül az egyik. Segítségével biztosítható, hogy egy job lépés indítása egy (vagy több) korábbi job lépés sikeres vagy sikertelen befejezésének függvényében megtörténjék vagy nem.

A COND paraméter értelmezéséhez tudnunk kell, hogy minden program befejezése után keletkezik egy állapotkód, amely a programot tartalmazó job lépés befejezéséről tájékoztat. A feldolgozandó programok esetében az állapotkód akkor 0 értékű, ha a job lépés hibátlanul befejeződött. Minél nagyobb az állapotkód értéke, annál komolyabb jellegű hiba keletkezett a futás során.

4, LT, FORT a GO nevű job lépés végrehajtása előtt a FORT nevű job lépésből származó állapotkódot hasonlítsa a 4-hez.

LT = kisebb, vagyis a hasonlítás „négy kisebb, mint a FORT állapotkódja". Ez a hasonlítás akkor igaz, ha a FORT állapotkódja 5-nél nagyobb, és akkor nem igaz, ha az állapotkód 4 vagy annál kisebb.

Ha a hasonlítás eredménye igaz, akkor a GO nevű job lépés nem kerül végrehajtásra. (Ez azt jelenti, hogy ha a fordítás során keletkezett hibák 4-nél nagyobb állapotkódot generáltak, nem kerülhet sor a futásra.)

4, LT, LKED az előbb elmondottak érvényesek az LKED nevű job lépés állapotkódjára is. Vagyis a futásra akkor sem kerülhet sor, ha a szerkesztés során 4-nél nagyobb értékű állapotkódot előállító hiba lépett fel.

⊙	16

Példa a sornyomtató definiálására

//GO. PRINT DD SYSOUT = A

Magyarázat:

SYSOUT Kulcsszavas paraméter, amely a GO nevű job lépésben az output osztályt definiálja.

Az A output osztály általában a sornyomtató, a B pedig a gép kártyalyukasztója. (Az operációs rendszer generálásakor a rendszerprogramozó által kerül beállításra.) Az egyes számítóközpontokban a további output osztályoknak (D, E, F stb.) más-más jelentése lehet. Pl. a kétpéldányos papírra szánt outputokat az E osztályba gyűjtik, majd egyszerre, amikor 2 példányos papír került befűzésre a nyomtatóba, íratják ki.

Példa FORTRAN-ból használt lemezállomány azonosítására

//GO. FT23F001 DD DSN = REGIF2, DISP = (OLD, KEEP, KEEP),
//UNIT = 2314,
//VOL = SER = KABOCA

Magyarázat:

FT23F001 FORTRAN-ban írt programok esetén a programban levő numerikus fájl azonosító (jelen esetben 23) és a rendszerbeli állománynév között ilyen konvencióval kell megteremteni a kapcsolatot. FORTRAN programok esetében tehát mindig ilyen formájú a DD név.

DSN = REGIF2 a hivatkozott állomány rendszerbeli neve REGIF2. Az állomány ezen a néven van nyilvántartva.

DISP = (OLD, KEEP, KEEP) Rendelkezés az állományról. OLD — az állomány már létezik (ha most történne a létrehozás, akkor NEW-t kellene írni).

KEEP — a job lépés hibátlan befejezése után az állományt továbbra is meg kell őrizni. KEEP a harmadik paraméter arról intézkedik, hogy mi történjék az állománynyal, ha a job lépés abnormálisan (hibásan) fejeződik be. Jelen esetben akkor is meg

kell őrizni. (A törlési igényt a DELETE paraméterrel a következő job lépésnek való átadási igényt a PASS paraméterrel lehet előírni.)

UNIT = 2314 A hivatkozott állomány 29 Mbájtos mágneslemezen található.

VOL = SER = KABOCA A hivatkozott állomány azon a mágneslemezen található, amelyiknek az azonosítója KABOCA.

Példa a FORTRAN-ból használt mágnesszalagos fájl azonosítására

//GO. FT26F002 DSN = MASODIK, UNIT = 2400, VOL = SER = BALAZS,
//DISP = (OLD, KEEP, KEEP), LABEL = (2, SL), DCB = DEN = 3

Magyarázat:

UNIT = 2400 A hivatkozott állomány 9 csatornás mágnesszalagon található.

FT26F002 A FORTRAN programbeli fájl azonosító mágnesszalagok esetén nem magát az állományt, hanem a hordozó szalag tekercset azonosítja. A 002 sorszám azt jelenti, hogy a szalagtekercsen levő második fájlról van szó.

DEN = 3 DCB paraméter, amely a szalagon levő írás sűrűségét adja meg (2 esetén 800 bit/inch, 3 esetén 1600 bit/inch az írás sűrűsége).

LABEL Cimkeinformációk leírására szolgáló paraméter. A mágnesszalagon levő fájl fejcímkéjére vonatkozik.

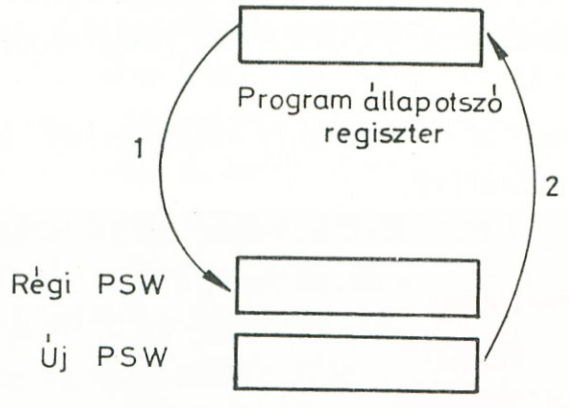
2 a szalagon ez a második fájl.

SL szabványos címkéről van szó. (Ezt a rendszer írja fel és ellenőrzi, a felhasználónak nincs gondja a kezelésével. Ezért célszerű mindig szabványos címkét használni.)

⊙	23

A megszakításokat a felügyelőprogram a programállapotszavak (PSW-k) segítségével kezeli az A-29. ábrán bemutatott elv szerint.

Valamennyi megszakítási okhoz két-két, a memória fix címein elhelyezkedő, és 64 bit (dupla szó) méretű terület tartozik, amelyek az esedékes PSW tárolására szolgálnak. Az egyik területen az éppen megszakított program állapotát tároljuk el, a másik a (tárban közvetlenül mellette levő) megszakítást szolgáló program PSW-jét tar-



A-29. ábra. PSW csere megszakítás felléptekor

talmazza. Az előbbit régi PSW-nek, az utóbbit új PSW-nek nevezik, s ilyen névvel használjuk az ábrán is.

Az ábra azt szemlélteti, hogy egy megszakítás felléptekor a futó program állapota az állapotszó regiszterből kimentésre kerül a régi PSW-be, a kezelőprogram új PSW-je pedig betöltésre kerül a regiszterbe. A régi PSW őrzi azokat az információkat, amelyekből egyrészt megállapítható a megszakítás konkrét oka, másrészt, amely alapján a megszakított program később tovább futtatható.

A kezelőprogram tárbeli címét az új PSW „utasításcím" része tartalmazza. Maga a felhasználói program „észre sem veszi", hogy meg volt szakítva, mert a megszakítási ok kiküszöbölése után úgy folytatódik, mintha „semmi sem történt volna".

A megszakítási okok mindegyikéhez külön kezelőprogram tartozik.

⊙	25

Maga a rendszer óra hardverbe épített eszköz. A központi tár 80-as címén levő 32 bites szó mindenkori tartalma szolgál e célra. Kihasználva az elektromos áram 50 Hz frekvenciáját, e szó 21. és 22. bit-jéről az 50 Hz-nek megfelelően (20 millimásodpercenként) levonásra kerül 1. A teljes szó 15,5 óra alatt lesz 0. Ekkor újra feltöltésre kerül. Ezt a tulajdonságot megfelelő technikával felhasználva lehet a rendszeren belül az időt mérni.

⊙	27

Az IEHLIST szervizprogram használatát a következő példán mutatjuk be.

Listát akarunk kapni a MKSZTO nevű lemez VTOC-jában levő bejegyzésekről. A listát szolgáltató job szerkezete:

```
//LISTA JOB
//EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=A
//DD2 DD UNIT=2314, DISP=OLD
//SYSIN DD*
  LISTVTOC FORMAT, VOL=2314=MKSZTO
/*
```

ahol: az EXEC utasítás hívja a szervizprogramot, a SYSPRINT nevű DD utasítás írja le a nyomtatás outputját: a sornyomtatót;

a DD2 nevű DD utasítás határozza meg magát a mágneslemezt mint állományt;

a SYSIN DD * egy input állományt ír le, ez esetünkben egyetlen utasításból áll, amely a * után következik;

LISTVTOC ennek hatására történik a VTOC listázás formátumozottan (ez az egyik megjelenítési mód). Itt kerül megadásra a lemez neve és típusa is.

X. fejezet

▲ 2

Olyan vállalatoknál, ahol elsősorban a termelést kísérő információs folyamatok (elsősorban a pénzügyi folyamatok) számítógépen vannak, de a termeléssel kapcsolatos feladatokhoz nem veszik igénybe a számítógépet, indokolt a gazdasági igazgatóhoz való tartozás. Ahol a vállalat vagy gyár nagy belföldi, esetleg külföldi forgalmat bonyolít, s a kereskedelemmel kapcsolatos tevékenységeket segíti a számítógép, elképzelhető a kereskedelmi igazgatóhelyetteshez való tartozás.

▲ 6

A számítógépeknél két okból van szükség a klimatizálásra. Először, mert ezek a gépek hőt termelnek és ezt el kell vezetni. Az integráltsági fok növekedésével, persze, csökken a hőtermelés is, manapság elsősorban a mágneses háttértárak hőtermelése a nagyobb. Másrészt — s ez a fontosabb — elsősorban a mágneslemezek számára pormentes környezetet kell biztosítani. A klimatizálás tehát pormentes, meghatározott hőmérsékletű és páratartalmú környezetet jelent.

▲ 7

A klímagép az álpadlóba nyomja be a megfelelően szűrt levegőt, és a mennyezeten levő elszívókon keresztül szívja el az elhasznált levegőt. Így a gépteremben a levegő mozog. A mozgásnak ott kell a legnagyobbnak lennie, ahol a legtöbb hő keletkezik. A beépített füstjelzők észlelik, ha füst, ill. tűz keletkezik.

● 1

A magyar szakirodalom és a számítástechnikai köznyelv a számítóközpont fogalmára több más elnevezést is ismer. Ezek részben azonos jelentésűek a számítóközponttal, részben árnyalják azt, éppen az említettek miatt.

Így pl. szokás számítógépközpontnak nevezni azokat a szervezeteket, amelyek csak gépet üzemeltetnek. De találkozhatunk még a „gépi adatfeldolgozó, elektronikus adatfeldolgozó" kifejezésekkel, amelyekhez aztán a központ, osztály főnév járul. Sok helyen a számítástechnikai osztály vagy számítástechnikai főosztály elnevezés divatos.

● 4

Ilyen vállalat pl. hazánkban a SzÜV, a Számítástechnikai Ügyvitelszervező Vállalat, amely a Központi Statisztikai Hivatal vállalataként, az ország majd minden megyéjében rendelkezik önálló számítóközponttal, és bérmunkával tartja el önmagát (pl.: népszámlálás, ipari vállalat részére bérszámfejtés stb.).

● 5

Az egyetemeken üzemelő nagyszámítógépek legtöbbjét hamarabb telepítették, mint ahogy a mikrogépek elterjedtek. Másrészt a mikroszámítógépek, bármennyire is professzionálisak és bármekkorát fejlődnek, nem tudnak megoldani minden felmerülő problémát. A nagyszámítógépekre tehát továbbra is szükség lesz. Ez pedig azt jelenti, hogy az ehhez való szakemberek képzése is szükséges marad.

● 8

Munka- és tűzvédelem a számítóközpontban
(Áttekintés)

Már a számítóközpontok tervezése, építése során figyelembe kell venni a munkavédelmi és tűzvédelmi előírásokat. Különösen vonatkozik ez magára a gépteremre és a különféle adathordozók tárolására szánt helyiségekre.

Munkavédelem

A gépterem klimatizált, a berendezések villamos árammal működnek, és egyes berendezések üzemeltetése (pl. sornyomtató) zajos. A gépterem dolgozóit az elszenvedhető ártalmak ellen védeni kell.

Csak megfelelő szakismerettel rendelkező és munkavédelmi oktatásban részesült dolgozó kezelheti a gépeket.

A gépteremben a zaj csökkentésére zajelnyelő falburkolatot építenek.

A gépteremben étkezni, cigarettázni nem szabad. A gépkezelők részére külön pihenőhelyiség áll rendelkezésre, 2 óránként 10 perc munkaközi szünetet (munkahelyi

torna) kell tartani. Az állandó légkondicionált levegőben dolgozókat meleg védőruha és munkaköpeny is megilleti.

A gépeket biztonságtechnikailag folyamatosan ellenőrizni kell.

A gépterembe csak az oda beosztott dolgozók léphetnek be.

A géptermek működési rendjét az adott körülmények figyelembevételével szabályozzák és a Géptermi Rendben rögzítik.

Tűzvédelem

A gépterem hő- és hangszigetelésére csak nem éghető anyag használható fel.

A gépterem ajtói önműködően kell hogy záródjanak. Nem éghető anyagból kell készülniük. Térelválasztásra csak fém—üveg kombinációjú fal használható.

A gépteremben csak a folyamatos üzemeltetéshez szükséges anyagmennyiség tárolható.

A számítóközpontban csak a kijelölt helyen — a gépteremben nem — szabad dohányozni. Tilos a dohányzás a klímateremben és az adathordozó raktárban is.

Az adathordozó raktárban a közlekedési utakat biztosítani kell. Tároláshoz csak nem éghető anyagból készült tárolószekrény vagy polc használható.

A villamos berendezéseket és a hálózatot tűzvédelmi szempontból 3 évenként felül kell vizsgálni.

A számítóközpontokban a kézi tűzjelző berendezés mellett automatikus tűzjelző berendezés is használatos.

A keletkező tűz oltására vagy automatikus (beépített) tűzoltóberendezést, vagy halonnal, ill. CO_2-vel oltó kézi tűzoltóberendezést kell üzemképes állapotban, készenlétben tartani.

A gépkezelőket és karbantartókat félévenként tűzvédelmi oktatásban kell részesíteni, amelyben ismertetni kell:

— a teendőket vészhelyzet esetén, így pl.:
  = a tűzoltókészülékek alkalmazását,
  = az anyagmentést,
  = a tűzoltók, mentők értesítését stb.;
— a tűzjelző berendezés működését;
— a számítóközpont tűzvédelmi utasítását;
— a karbantartók részére a tűzveszélyes folyadékok használatát.

● 9

Az operátori pihenőre azért van szükség, mert nem lehet napi nyolc órát a klimatizált gépteremben eltölteni. Általában óránként vagy kétóránként szükség van rövid kikapcsolódásra.

⊙ 3

A számítógépnek a vállalat életében elfoglalt helye és a vállalati „számítógépesítés" problémaköre külön területe a számítástudománynak. Általában „Számítógépes információrendszerek" elnevezéssel illetik. Külföldi és hazai példák tucatja igazolja, hogy csak azok a vállalatok tudják sikeresen alkalmazni a számítógépeket, amelyek kellő energiát fordítanak arra, hogy a vállalati folyamatokat az információrendszerek tervezésének elvei szerint alkalmassá tegyék a számítógép fogadására.

Az egyes fejezetek végén levő kérdésekre adott válaszok

I. FEJEZET

1. Az automatát az jellemzi, hogy a kívülről érkező jelek hatására belső állapotát megváltoztatja és valamilyen kimenőjelet hoz létre.

2. A tárolt program fogalma azt jelenti, hogy a számítógépben (annak is a központi tárában) a végrehajtandó program benne van. A számítógép ezáltal belső vezérlésű.

3. A számítógép az a műszakilag megvalósított rendszer, amely képes adatok tárolását, visszakeresését és feldolgozását emberi beavatkozás nélkül, a benne elhelyezett program működtetésével megvalósítani.

4. Első generáció: elektroncsövekből épülő hardver, gépi kódban való programozás. Második generáció: tranzisztorok a hardver elemek építőelemei, megjelennek a programozási nyelvek. Harmadik generáció: egyre nagyobb fokú integrált áramkörök a hardverben, korszerű rendszerprogramokkal rendelkező szoftver.

5. A számítógép alapvető erőforrásai a hardver és szoftver.

6. A hardver a számítógép műszakilag megalkotott, megépített technikai elemeinek összefoglaló neve.

7. A szoftver a programok és a programrendszerek összefoglaló neve. A számítógép használatának lehetőségeit bővíti.

8. A mikroszámítógépek alapvető jellemzői: kis méret, könnyű szállíthatóság, alacsony ár (a nagygépekhez viszonyítva), jelentős teljesítmény.

9. Foglalkozási csoportok: rendszerszervezők, rendszerprogramozók, alkalmazási programozók, operátorok, műszakiak, adatrögzítők.

10. Az operátor feladata a programok bevitelének előkészítése, a rendszer működtetése, a géptermi rend biztosítása, a szükséges nyilvántartások vezetése.

II. FEJEZET

1. Információnak nevezzük a számunkra új ismereteket hordozó jelek tartalmi jelentését.

2. Az elemi adat az információnak olyan megjelenített formája, amely kisebb

egységekre már nem osztható fel anélkül, hogy az általa hordozott információ ne torzulna.

3. Minden elemi adat valamilyen egyedre vonatkozik.

4. Az adat típusa meghatározza az adat által felvehető értékek és az adattal végezhető műveletek halmazát.

5. A tanult adattípusok: egész, valós, logikai, karakter típus.

6. A numerikus adat csak számjegyekből állhat (egész vagy valós típusú lehet), az alfanumerikus tartalmazhat betűket is (karakter típusú lehet).

7. A tanult adatstruktúrák: a tömb és a rekord.

8. A tömb azonos típusú adatok rendezett halmaza. Egy-egy tömbelem valamilyen sorszámmal azonosítható.

9. Tetszőleges típusú adatok egy logikai egysége képezi a rekordot.

10. Fájl keletkezik, ha egy feldolgozás céljaira adott szempont szerint valamennyi egyed rekordját összegyűjtjük.

11. A kódrendszer egy jelkészlet a hozzárendelési szabállyal kiegészítve.

12. Az adathordozókon történik az adatok rögzítése (ábrázolása).

III. FEJEZET

1. A program egy utasításcsoport, amely egy kijelölt input adatstruktúrából egy előre meghatározott eredménystruktúrába való átmenet lépéseit írja le, valamilyen programozási nyelven.

2. Az algoritmus a program vezérlésleíró része.

3. Általános műveletek: hasonlítás, értékadás, átvitel.

4. Logikai típusú adatokkal értelmezett műveletek: konjunkció, diszjunkció, negáció.

5. Az értékadásnál figyelembe kell venni a következőket:

— az értékadás jobb oldalán csak értékkel rendelkező adat lehet;

— a jobb oldalon megjelenő változó értéke megmarad;

— a bal oldalon megjelenő változó korábbi értéke elvész.

6. A blokkdiagram az algoritmus leírásának egyik lehetséges eszköze.

7. A tömbelemekre való hivatkozás: megadjuk a tömb nevét, és egydimenziós tömb esetében egy sorszámot, kétdimenziós tömb esetén két sorszámot. A két sorszám közül az első az aktuális sor, a második az aktuális oszlop számát határozza meg.

8. A ciklus az algoritmusban levő hurok, amelynek segítségével bizonyos utasítások — a ciklustörzs — többször is végrehajthatók.

9. Az alprogram formailag önálló programegység, amely általában egy konkrét funkciót valósít meg. Környezetével — a hívó programmal — paraméterlistában megadott változók segítségével tud kapcsolatot tartani.

10. A funkcionális lebontás módszerének lényege, hogy a megoldandó feladatot modulokra bontjuk. A modulok mindegyike a feladat egy-egy jól körülhatárolt részét (egy-egy funkciót) valósítja meg.

IV. FEJEZET

1. A kettes számrendszer lényege, hogy minden számot kettő hatványaként fejez ki.

2. A bit a kettes számrendszer egy jegye, ill. egy helyértéke.

3. A kettes számrendszer előnye, hogy két lehetséges állapottal kifejezhető (leírható) minden adat, s ezt technikailag könnyű megvalósítani.

4. A kettes számrendszer a tízessel szemben azért hátrányos, mert egy-egy szám leírásához (átlagosan 3,3-szor több) helyértékre van szükség.

5. A tizenhatos számrendszerben 16 számjegyet értelmezünk.

6. $A=10$, $B=11$, $C=12$, $D=13$, $E=14$, $F=15$.

7. Fixpontos ábrázolás esetén a törtpont egy előre meghatározott helyen kerül értelmezésre.

8. Fixpontosan csak egész számok ábrázolása lehetséges.

9. Fixpontosan egész típusú adatokat célszerű ábrázolni.

10. Lebegőpontos ábrázolás esetén minden szám mantissza karakterisztika formában (vagyis szorzat formában) kerül megadásra.

11—12. Egy szám normálalakjáról akkor beszélünk, amikor a mantissza eleget tesz a következő feltételnek: $1/p \leq |M| < 1$, ahol p a számrendszer alapszáma, M a mantissza.

13. A lebegőpontos ábrázolás esetén az ábrázolható számok halmaza nagyobb, mint fixpontos ábrázolás esetében.

14. A lebegőpontosan ábrázolt számokkal lassúbb a műveletvégzés, mint a fixpontosan ábrázoltakkal.

15. Tömörített ábrázolásról beszélünk, amikor egy bájtban két számjegy kerül elhelyezésre.

V. FEJEZET

1. A központi tárban megvalósított tárolás egyik alapegysége a bájt. 8 bit tárolására alkalmas. A rekesz a tárnak az a fizikai helye, amelyben 1 bájt tárolható.

2. Egy rekesz címe, a rekeszhez rendelt sorszám, amely a rekesz azonosítására szolgál.

3. Egy rekesz tartalma a benne tárolt bitkombináció aktuális értéke.

4. A regiszterek adatok átmeneti ideig való tárolására szolgáló hardver elemek.

5. A címregiszter és az adatregiszter a központi tárral való kapcsolattartáshoz szükségesek. A címregiszterbe kerül a keresett tárcím, az adatregiszterbe a kiolvasott (vagy tárolandó) tartalom.

6. A ciklusidő a tárhoz való kétszeri hozzáférés közötti legrövidebb idő.

7. A központi egység két részből áll, az aritmetikai-logikai egységből és a vezérlőegységből.

8. Az aritmetikai logikai egység végzi az utasítás végrehajtását.

9. A vezérlőegység a műveletvégzés előkészítését és a műveletek végrehajtásának irányítását végzi.

10. Gépi kódnak nevezzük azt a nyelvi struktúrát, amelyet a számítógép közvetlenül végre tud hajtani.

11. A tényleges tárcím kiszámítása legegyszerűbb esetben a bázisregiszter és az eltolási cím alapján történik.

12. Egy gépi kódú utasítás végrehajtása két lépésben történik, az előkészítő fázisban az utasítás megkeresése, kiolvasása és értelmezése történik meg, a második, a végrehajtási fázisban a tényleges végrehajtás.

13. Egy mikroprogram a fixtárban levő, hardveresített utasítássorozat, amely a központi egység működéséhez szükséges alapvető tevékenységek programjait tartalmazza.

14. Az utasításszámláló regiszter tartalma a soron következő utasítás címe.

15. Az adatsín olyan hardver elem, amelyen a regiszterekhez, ill. a regiszterektől való adatáramlás megvalósul.

16. A csatornákon történik a központi tár és a perifériák közötti adatmozgás. Elsősorban a sebességkülönbségből adódó problémák megoldását szolgálják.

17. Multiplexor csatorna, amelyre a lassú, és szelektorcsatorna, amelyre a gyors perifériák csatlakoznak.

VI. FEJEZET

1. A szoftver a programok és a programrendszerek az összefoglaló neve. A számítógép használatának lehetőségeit bővíti.

3. Gépi kódnak nevezzük azt a nyelvi struktúrát, amelyet a számítógép közvetlenül végre tud hajtani.

3. Főbb korlátok: numerikus műveleti kódok, numerikus és közvetlen tárcímek, kötött hely a tárban, elemi szintű algoritmus, hibajavítás nehézkessége.

4. A gépi kódban való programozásból adódó problémák megoldása.

5. Az assembly szintű programokat gépi kódra fordító program az assembler.

6. Főbb jellemzőik: gépfüggetlenség, az emberhez közelálló utasításszavak, könnyű elsajátíthatóság.

7. Alapelemek: konstansok, változók, címkék, nyelvi kulcsszavak.

8. A programfordítás az a folyamat, amelynek során a forrásnyelven megírt programból tárgyprogram készül. Minden esetben szükséges, ha egy program nem gépi kódban áll rendelkezésre és futtatni szeretnénk.

9. A fordítóprogram feladata, hogy forrásnyelvi programból tárgyprogramot állítson elő.

10. A fordítóprogram outputja azért nem futtatható, mert feloldatlan hivatkozásokat tartalmaz.

11. A kapcsolatszerkesztő feladata a tárgyprogramban levő külső hivatkozások feloldása és a futtatható program előállítása.

12. Elsődleges input: a tárgyprogram, amihez hozzá kell szerkeszteni. A másodlagos input: a külső hivatkozásokat tartalmazó könyvtárak, amelyből hozzá kell szerkeszteni az elsődleges inputhoz.

13. Az összeszerkesztett program egy ideiglenes vagy egy állandó könyvtárba kerül.

14. Az interpreter a forrásprogramot utasításonként fordítja, és rögtön végre is hajtja.

15. A BASIC parancsok segítségével a programozó a gép szolgáltatásait és funkcióit befolyásolja.

16. A BASIC nyelv utasításainak segítségével a programozó a megoldandó feladat programját készíti el.

VII. FEJEZET

1. A perifériákon keresztül tud a központi egység kapcsolatot tartani a külvilággal.

2. Az adatáramlás iránya szerint: input, output és input-output egységek, az átvitel sebessége alapján: lassú és gyors perifériák.

3. A kártyaolvasó feladata a lyukkártyára lyukasztott programok és adatok beolvasása.

4. Az optikai bizonylatolvasás lényege, hogy az adatbevitelhez olyan bizonylatokat használnak, amelyet a számítógép és az ember egyaránt el tud olvasni.

5. A sornyomtatók lehetnek hengeres, íróláncos, írórudas elven működőek.

6. A mikrofilmes output előnyei:
kis helyen elférnek az eredmények, könnyebb a keresés, ha egy adott eredményadat vagy -tábla szükséges, könnyebben sokszorozható, egyszerűbb az eredmények védelme, szükség esetén a megsemmisítésük.

7. A mikrofilmes output hátrányai: az egyszeri ráfordítás költségei (olvasóberendezések beszerzése) nagyon nagyok.

8. A terminálok lehetőséget teremtenek a programozó számára, hogy a számítógép erőforrásaihoz kényelmesen, gyorsan tudjon hozzáférni, és feladatait rövidebb idő alatt meg tudja oldani.

9. A konzolon keresztül tart kapcsolatot egymással a számítógép és az operátor (gépkezelő).

10. A rajzolóberendezések segítségével lehet a számítógéppel ábrákat (térhatású ábrákat is) előállíttatni. Ez elsősorban a különböző tervezési feladatoknál nyújt segítséget.

11. A hazánkban található háttértárak: mágnesszalag, mágneslemez.

12. Mágnesszalagon használható írássűrűségek: 800, 1600 bit/inch.

13. A mágnesszalagra felvihető bájtok mennyisége alapvetően a blokkmérettől és az írássűrűségtől függ.

14. A blokk a számítógép által egy átviteli egységként kezelt adatmennyiség, amelyet a háttértárak és a központi tár között mozgat.

15. A blokkolt adattárolás előnye, hogy felgyorsítja a feldolgozás menetét (lecsökken az adatátvitelek száma), és jobb kihasználást biztosít a háttértáron.

16. Az adathordozói címke egyebek mellett az adathordozó szalag vagy lemez fizikai azonosítására szolgáló nevet tartalmazza.

17. A fájl bevezető címke azonosítja a fájlt. Ezen kívül a fájlra vonatkozó információkat is tartalmaz: belső szerkezet, megőrzési idő stb.

18. A mágneslemez mágnesezhető felületekből épül fel, minden felületen sávok találhatók. Az azonos sorszámú sávokat cilindernek nevezzük.

19. A sáv a lemez egy felületén levő koncentrikus kör, amelynek mentén az adatelhelyezés lehetséges. A cilinder az azonos sorszámú, de különböző felületeken levő sávok összessége.

20. Mágneslemez címkék: adathordozói címke, az operációs rendszer által használt címkék, felhasználói fájl címkék.

21. A mikrogépek legfontosabb perifériái: kazetta, hajlékony lemez, nyomtató, maga a billentyűzet és a képernyő.

VIII. FEJEZET

1. A szekvenciális fájlt az jellemzi, hogy rekordjai a rekordazonosítók növekvő (esetleg csökkenő) sorrendjében követik egymást.

2. Szekvenciális fájl bármilyen adathordozón (mágnesszalag, mágneslemez, lyukkártya stb.) létrehozható.

3. Az egyszerű (alapvető) és a komplex fájlszerkezetek közötti lényegi különbség az, hogy míg az előzőeknél a rekordok keresése magából a fájl adataiból megoldható, addig az utóbbiaknál a kereséshez további információk (mutatók, indexek) szükségesek.

4. A mágnesszalagon levő szekvenciális fájl karbantartása minden esetben a teljes állomány átmásolásával jár együtt.

5. Az indexelt szekvenciális fájl biztosítja a szekvenciális és a közvetlen elérést is.

6. Indexelt szekvenciális fájl csak címezhető adathordozón (pl. mágneslemez) hozható létre.

7. Az indexelt szekvenciális szerkezet lényege, hogy a rekordok azonosítóra növekvő sorrendben helyezkednek el. Vannak kitüntetett rekordok, amelyeknek címe és azonosítója különböző indextáblázatokban fel van jegyezve. A keresés alapvetően ezeken az indextáblázatokon át történik. Bővítés esetén az ún. túlcsordult rekordok elhelyezésére külön terület áll rendelkezésre. A túlcsordult rekordok láncolásával a fájl rekordjai logikailag mindig rendezettek maradnak, bár a fizikai sorrend tekintetében ez már nem igaz.

8. Az indexelt szekvenciális fájl fontosabb területei: elsődleges adatterület, cilinderindex terület, független túlcsordulási terület.

9. A sávindex területe és a cilinderindexek területe.

10. A sávindexsávon minden sávra vonatkozóan kétféle információ található: a normálindex tétel, amelyből kiderül, hogy az adott sávon mekkora a legnagyobb kulcsú rekord azonosítója, és a túlcsordulási index tétel, amelyből kiderül az adott sávhoz tartozó legnagyobb kulcsú rekord azonosítója és az ugyanezen sávhoz tartozó legkisebb kulcsú rekord címe.

11. Túlcsordulás keletkezik, ha egy olyan rekordot kell felírni a fájlba, amely nem fér el az elsődleges adatterületen a kulcsa szerint esedékes sávon.

12. Kétféle túlcsordulási területet ismerünk, a cilinder túlcsordulási területet (minden cilinderhez tartozik ilyen) és a független túlcsordulási területet.

13. A független túlcsordulási területet az IS állomány középső cilinderei táján célszerű elhelyezni.

14. A könyvtári állományok programok tárolására szolgálnak.

15. Könyvtári állomány csak címezhető adathordozón (pl. mágneslemezen) hozható létre.

16. Tagnak nevezzük a könyvtári állomány egy olyan elemét, amely egy program tárolására alkalmas.

17. A könyvtári állomány tartalomjegyzéke — egyebek mellett — tartalmazza minden tag nevét, első rekordjának a könyvtár elejéhez viszonyított címét, és a tag méretét (félszavakban mérve).

IX. FEJEZET

1. Az operációs rendszer a rendszerprogramok azon összessége, amelyek működése a számítógép erőforrásainak adott cél szerinti optimális kihasználását, a programozói munka megkönnyítését és az operátori munka egyszerűsítését eredményezi.

2. Az operációs rendszerek az optimalizálandó cél szerint csoportosíthatók, így beszélünk:

— kötegelt feldolgozást biztosító operációs rendszerekről;

— többfelhasználós, időosztásos operációs rendszerekről;

— valós idejű feldolgozást biztosító operációs rendszerekről;

— (több célt egyidejűleg megvalósító operációs rendszerekről).

3. A kötegelt feldolgozást biztosító operációs rendszerek jellemzője, hogy a munkák átfutási idejét igyekszik lerövidíteni.

4. A többfelhasználós időosztásos rendszerek jellemzője, hogy a terminálóknál ülő felhasználók számára kényelmes hozzáférést biztosítanak.

5. A job (munka) egy vagy több, általában logikailag összetartozó programot magába foglaló, a számítógép számára összeállított, végrehajtásra előkészített műveletsor.

6. A task (feladat) futás céljából a központi tárba betöltött program és a hozzá tartozó adatok.

7. Multiprogramozásról beszélünk akkor, ha biztosítva van a futó program megszakítása és a megszakítások által keletkezett „szünetekbe" másik program újraindítása.

8. A partíció olyan tárfelosztási forma, amelynél az operációs rendszer egy-egy partícióhoz egy-egy munkát rendel hozzá. Az egyes partíciókban levő feladatok között valósul meg a multiprogramozás.

9. A job osztály a job és a partíció egy paramétere. A job osztály által befolyásolható, hogy az egyes jobok mely partíciókba kerülhetnek be.

10. Megszakítás keletkezik, amikor futó program környezetében olyan változás áll be, amely akadályozza a program további futását. Ilyenkor a supervisor veszi át a vezérlést.

11. Fontosabb megszakítási okok: SVC megszakítás, input-output megszakítás, külső megszakítás, program-, ill. géphiba miatti megszakítás.

12. A felhasználó az operációs rendszerrel a munkavezérlő nyelv segítségével tud kapcsolatot teremteni.

13. Fontosabb munkavezérlő utasítások a JOB, az EXEC, a DD utasítás.

14. A JOB utasítás funkciója az elvégzendő munka azonosítása.

15. Az EXEC utasítás funkciója a végrehajtandó program (vagy eljárás) azonosítása.

16. A DD utasítás funkciója a feldolgozás során szükséges adatállományok azonosítása. Minden állományhoz külön DD utasítás kell.

X. FEJEZET

1. Gépteremnek nevezzük azt a helyiséget, amely a nagyszámítógépek üzemeltetéséhez szükséges és előírt módon van kialakítva.

2. A számítóközpont a nagyobb számítógépek üzemeltetéséhez szükséges helyiségek, berendezések összessége, amelyet megfelelő felkészültségű személyzet használ, ill. üzemeltet.

3. A számítóközpontok többféle céllal üzemelhetnek. Ezek közül a legfontosabbak:

— vállalati termelésirányítás,
— statisztikai feldolgozás,
— oktatás,
— bérmunka végzése,
— tudományos kutatások támogatása.

4. A géptermi álpadló funkciói: alatta lehet elhelyezni azokat a huzalokat és kábeleket, amelyek a számítógép egységei számára a villamos energiát biztosítják, és amelyek az egységet egymással összekapcsolják (pl.: adattovábbítás céljából).

Ezenkívül az álpadló alá vezetik be a friss levegőt, amely erre a célra vágott nyílásokon keresztül kerül a gépterembe. Az álmennyezetbe épített elszívókon keresztül hagyja el a géptermet a levegő, ide építik be a füstjelző készülékeket. Mindkettőnek van zajelnyelő funkciója is.

5. Az operátori pihenőre azért van szükség, mert a gépterem klimatizált és zajos, ami megterheli a szervezetet.

Az egyes fejezetek végén levő feladatok megoldása

Megjegyzés:

A feladatok nagy részének a megoldása egy-egy blokkdiagram elkészítését jelenti. Az itt megadott blokkdiagramok az adott feladat egy lehetséges megoldását adják. Természetesen ezektől eltérő megoldások is lehetnek jók.

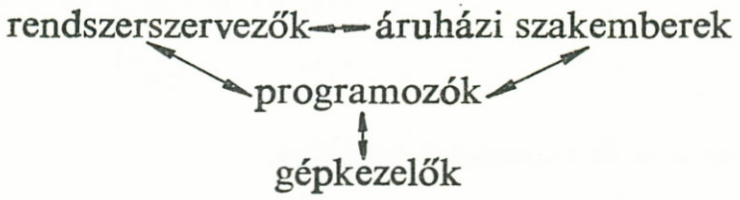
I. FEJEZET

1. Közös tulajdonságok: programozott, a benne levő program alapján emberi beavatkozást nem igényel. Eltérő tulajdonságok: a számítógép általános célú, az automata mosógép egycélú. A számítógépben levő program könnyen cserélhető, a mosógépben levő nem.

2. Az elektroncsöves („lámpás") rádiók nagyok voltak, nagy volt az energiafelvételük, gyakran elromlottak. Áruk is viszonylag magas. A tranzisztoros rádiók mérete csökkent, kisebb az energiafelvételük (elemről is működtethetők), csökkent a meghibásodások száma, olcsóbbak. Az integrált áramkörös rádiók egészen kicsik, minimális az energiafelvételük, kicsi a meghibásodás valószínűsége, áruk is egyre csökken. Hangminőségük sokkal jobb, mint a csöves vagy a tranzisztoros készülékeké. Mindez — értelemszerűen — igaz a számítógépekre is.

3. Az embert is érik külső behatások, s ezekre valami módon reagál. (Még „belső állapotában", pl. idegállapotában is történik változás.) Nyilvánvaló, hogy az emberi szervezet és értelem hallatlanul bonyolult rendszert alkot, amely azonban végeredményben egy nagyon bonyolult működésű automataként fogható fel.

4. A rendszerszervezők az áruházi szakemberekkel karöltve elkészítik a számítógépes rendszer tervét, amelyből a programozók készítik el a programokat. A programok kipróbálása során az áruházi szakemberek is közreműködhetnek (figyelni az outputok jóságát). A gép működése során a gépkezelők kezelik a gépet. Jelzik a programozónak, ha valami rendellenességet tapasztalnak.



II. FEJEZET

1. Napok: {hétfő, kedd, szerda, csütörtök, péntek, szombat, vasárnap}
Családtagok: {apa, anya, gyerekek, nagyszülők}
Négyszög: {paralelogramma, trapéz, deltoid, általános négyszög}

2. A kódrendszer pl. így nézhet ki.

A = 01; B = 02; C = 03; D = 04; E = 05; É = 06; F = 07; G = 08; H = 09; I = 10; J = 11; K = 12; L = 13; M = 14; N = 15; O = 16; Ö = 17; P = 18; Q = 19; R = 20; S = 21; T = 22; U = 23; Ü = 24; V = 25; X = 26; Y = 27; Z = 28

(A dupla betű pl. CS, SZ stb. nem kíván külön kódot, hiszen alkotóelemeiből összerakható. Ha igényesebbek vagyunk, képezhetünk külön kódot a hosszú magánhangzók számára.)

A szövegkészítést rátok bízzuk.

3. Ugye nem tökéletes a kódolt szöveg? Mi hiányzott a kódkészletből? Mindenekelőtt a szóköz (üres hely, amely a mondat szavait elválasztja. Azután az írásjelek, így a vessző, pont (esetleg a felkiáltójel, kérdőjel).

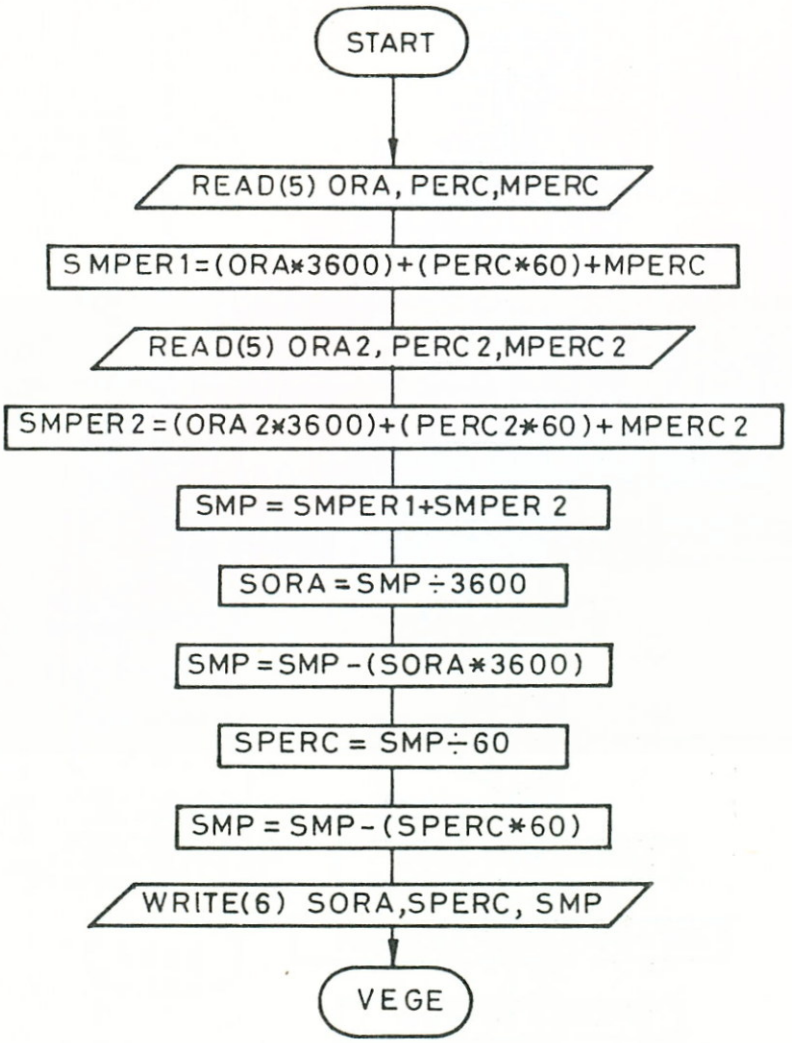
A kódrendszert így egészíthetjük ki

□ (üres hely) = 29; , (vessző) = 30; . (pont) = 31 stb.

4. A kártyaterv pl. a következő formájú lehet:

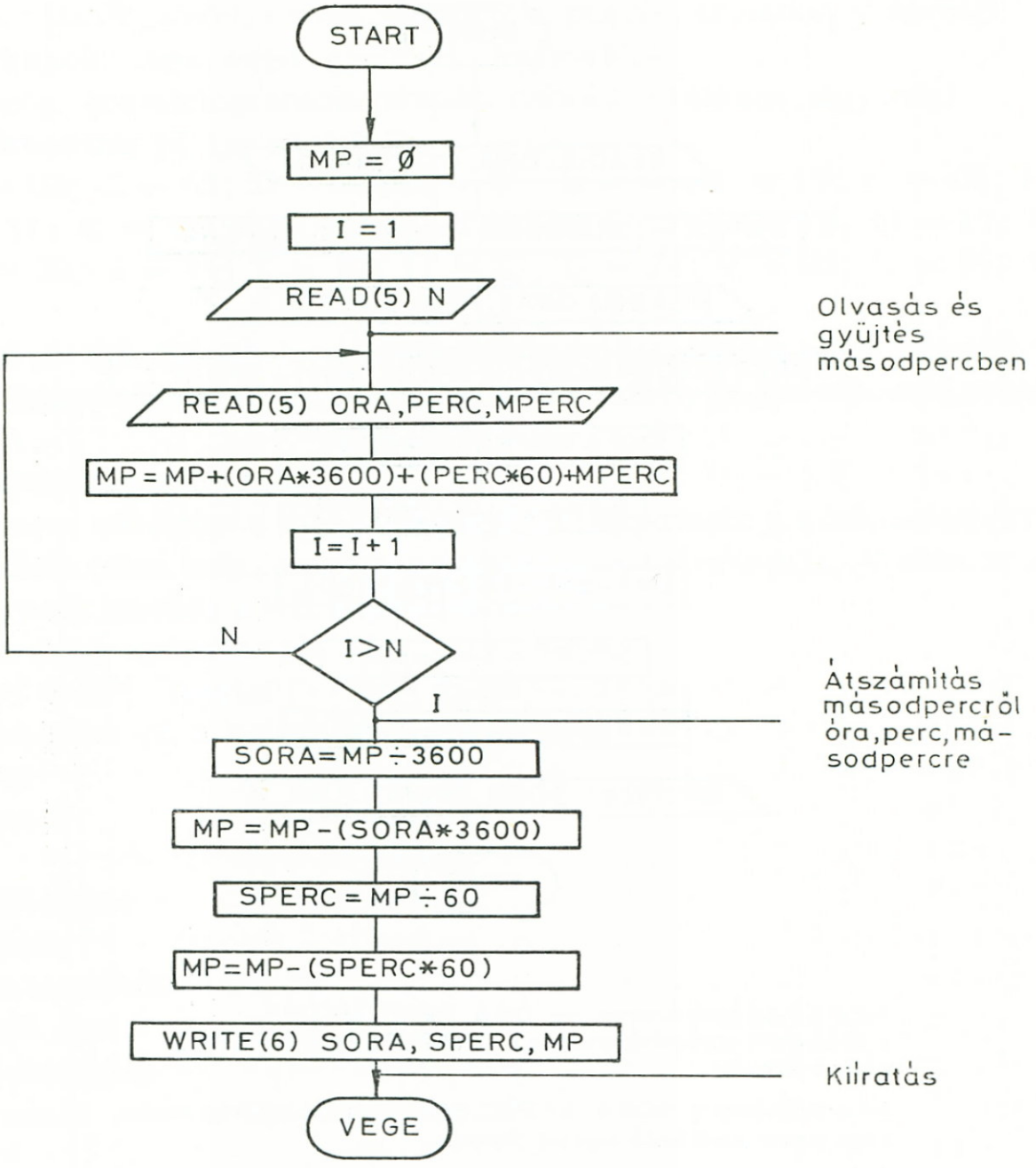
A tanuló neve	1—24 oszlop
Születési dátum*	25—32 oszlop
Lakhelye	33—56 oszlop
Eltartó foglalkozása	57—68 oszlop
Testvérei száma	69—70 oszlop
Középiskolai tanulmányainak kezdő éve	71—74 oszlop
Tanulmányi átlaga az előző tanév végén	75—78 oszlop

* a hónap sorszámát is arab számokkal kódoljuk.

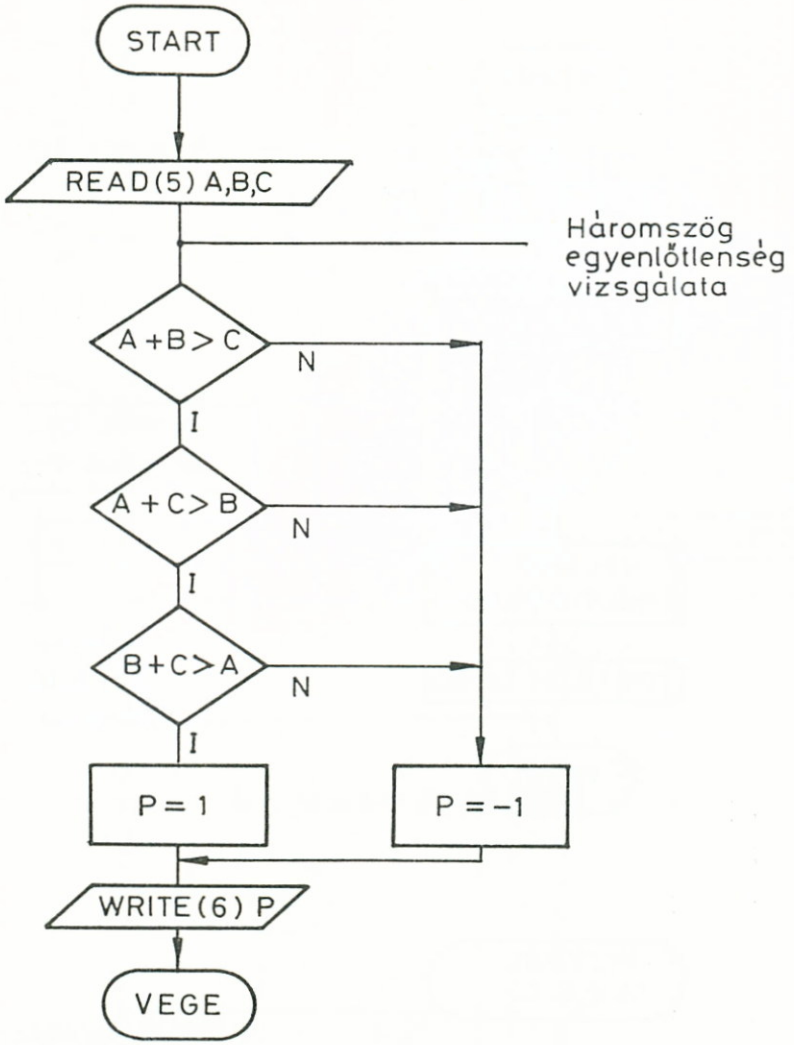


Az első számhármas az ORA, PERC, MPERC
a második számhármas az ORA2, PERC2, MPERC2
változókban.

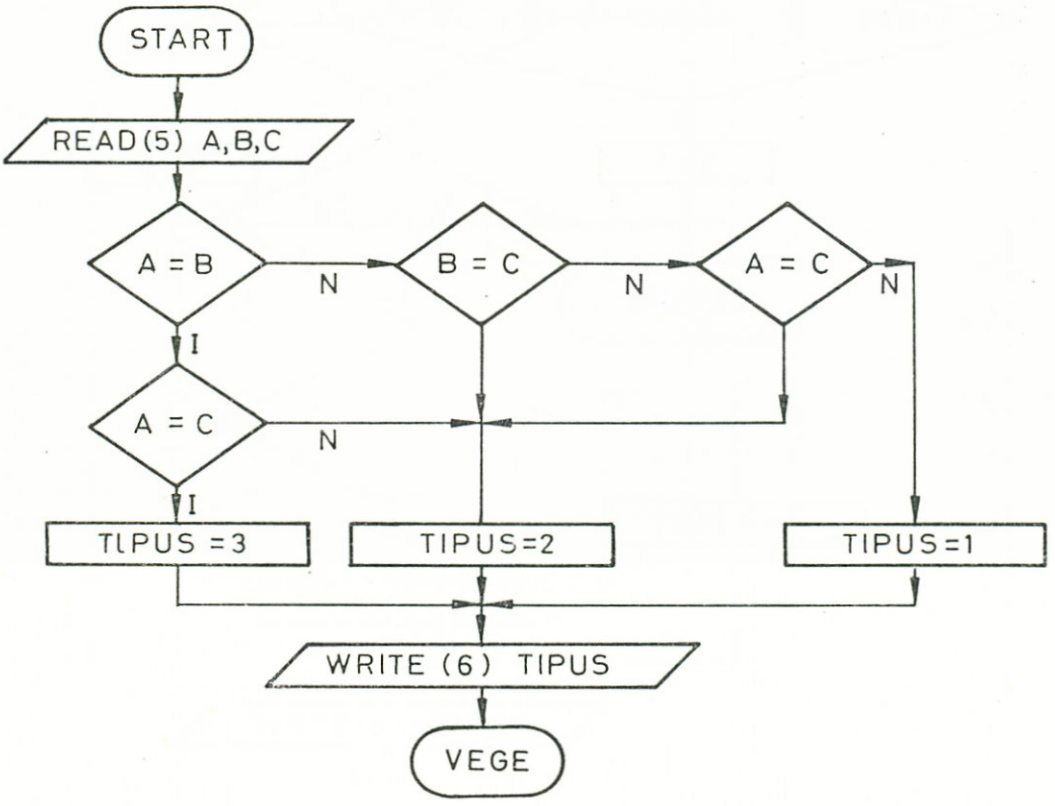
Az eredmény a SORA, SPERC, SMP változóba képződik.
Valamennyi változó egész típusu.

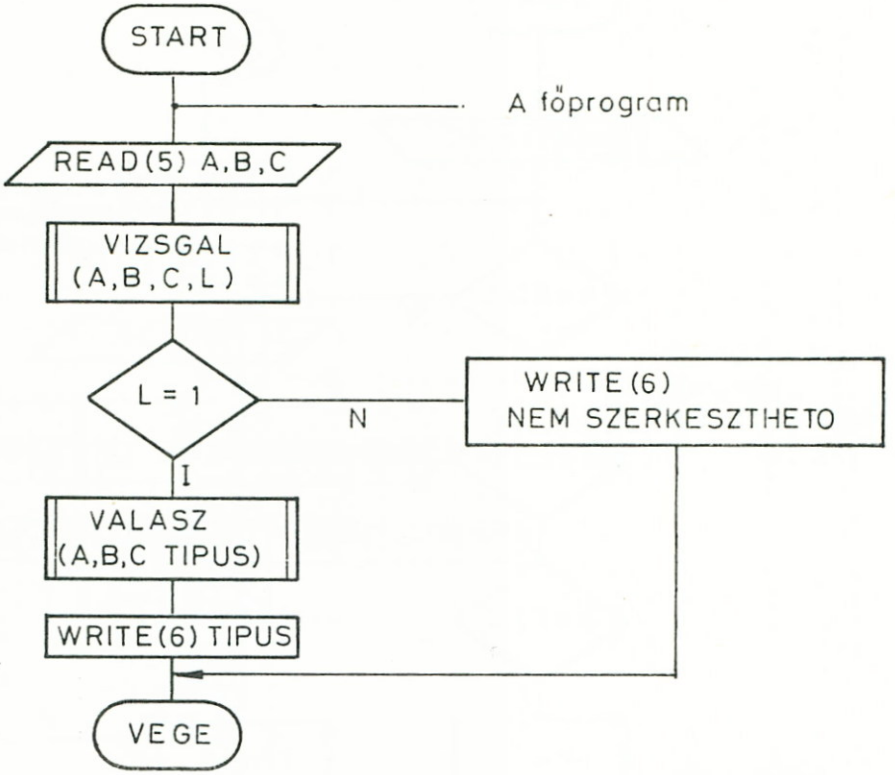

START
MP = Ø
I = 1
READ(5) N
Olvasás és gyüjtés másodpercben
READ(5) ORA,PERC,MPERC
MP = MP+(ORA*3600)+(PERC*60)+MPERC
I = I + 1
N
I>N
I
Átszámitás másodpercről óra,perc,másodpercre
SORA=MP÷3600
MP = MP-(SORA*3600)
SPERC = MP÷60
MP=MP-(SPERC*60)
WRITE(6) SORA, SPERC, MP
Kiiratás
VEGE

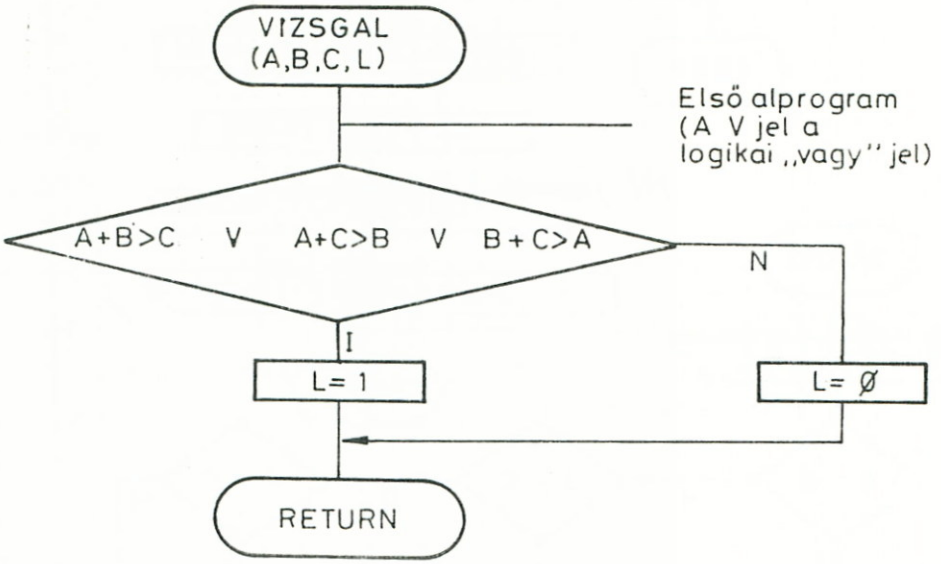
III. fejezet 2. feladat

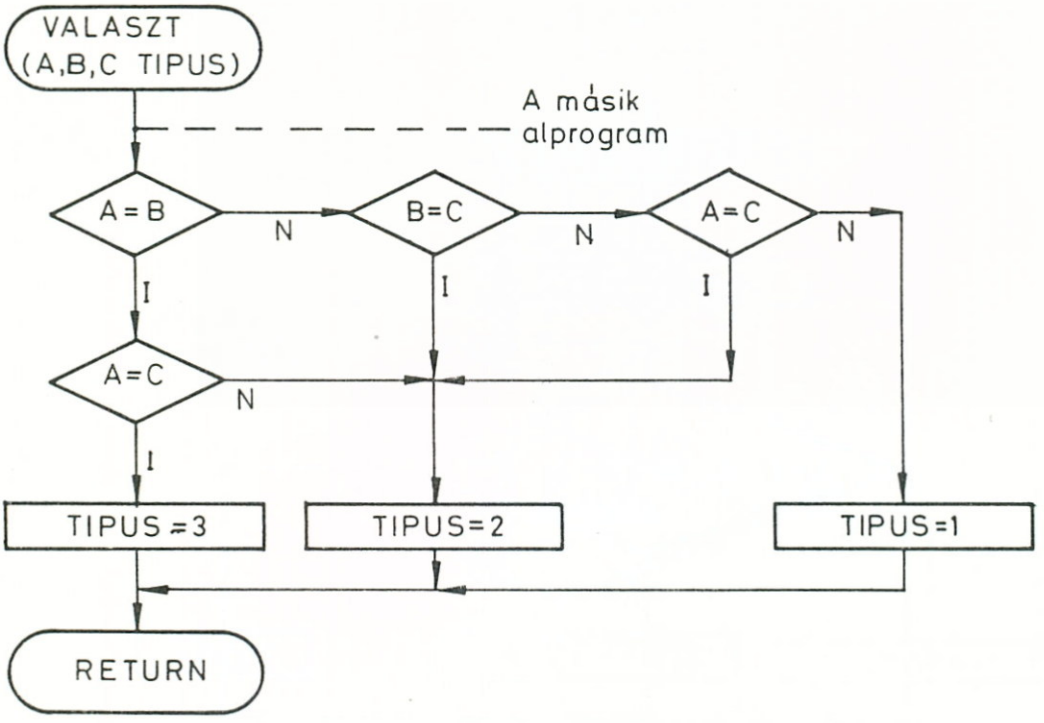


III. fejezet 3. feladat

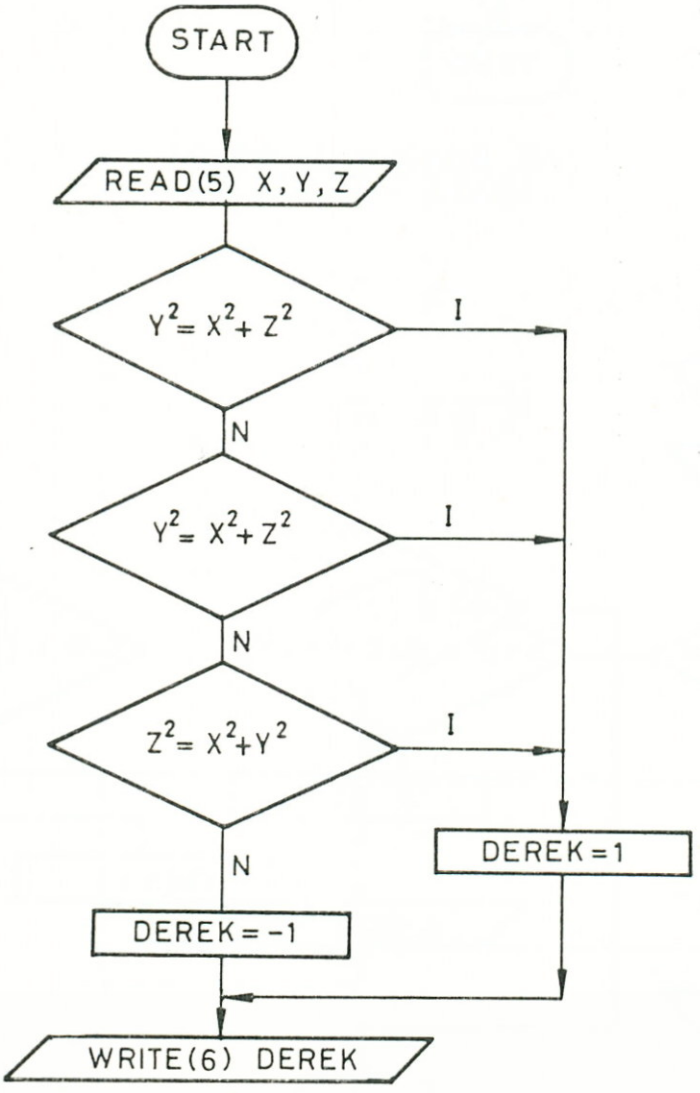



START
A főprogram
READ(5) A,B,C
VIZSGAL
(A,B,C,L)
L = 1
N
WRITE(6)
NEM SZERKESZTHETO
I
VALASZ
(A,B,C TIPUS)
WRITE(6) TIPUS
VEGE

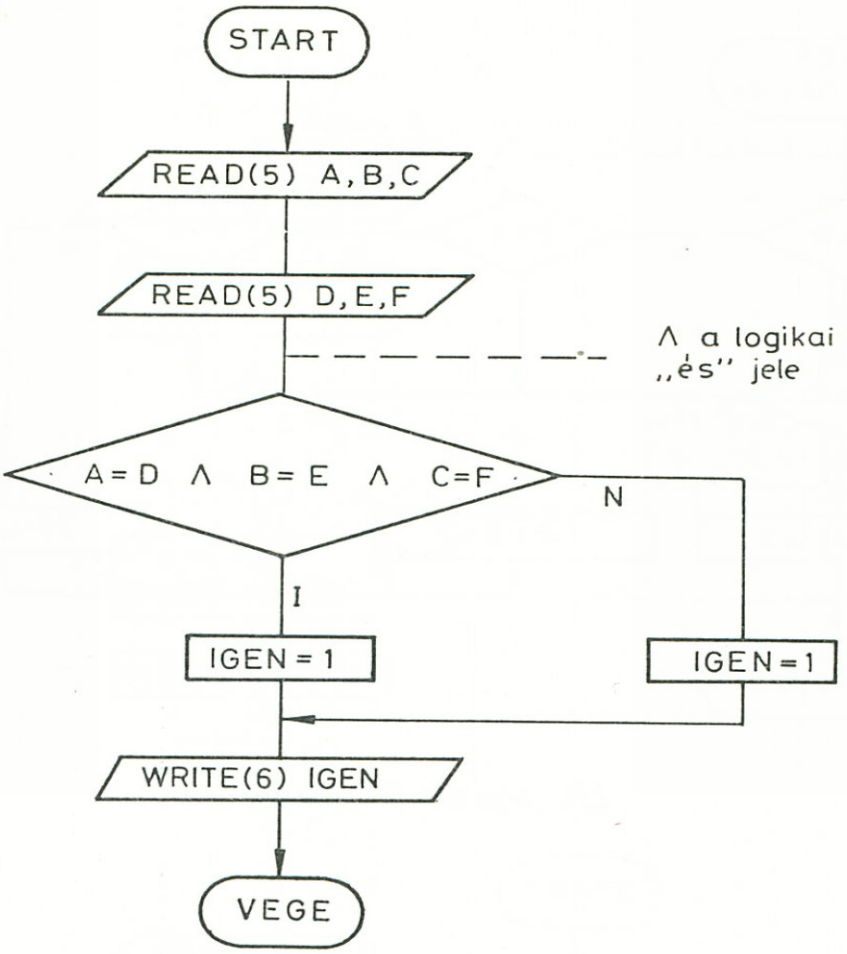

VIZSGAL
(A,B,C,L)
Első alprogram
(A V jel a
logikai „vagy" jel)
A+B>C V A+C>B V B+C>A
N
I
L= 1
L= Ø
RETURN



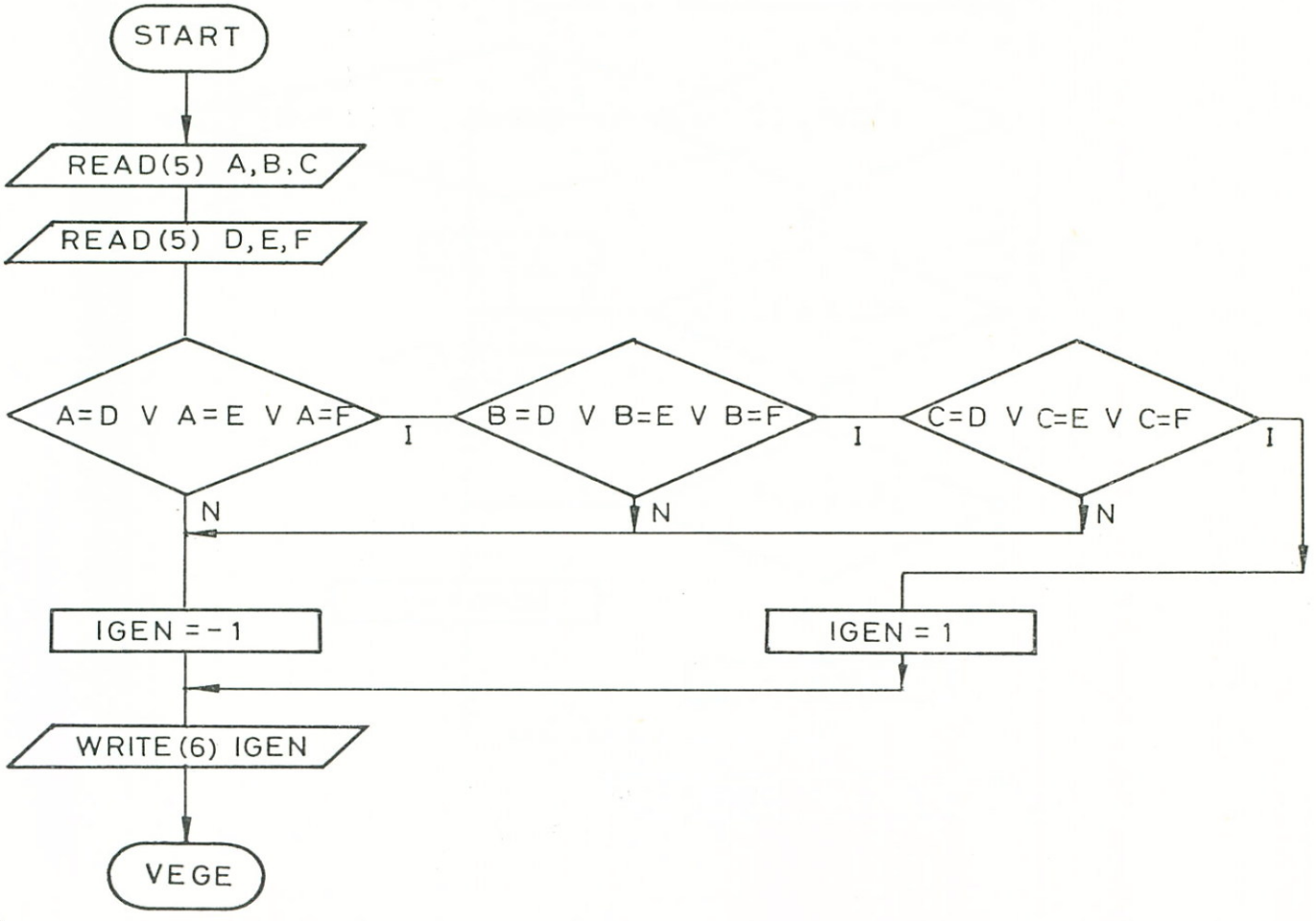
III. fejezet 5. feladat



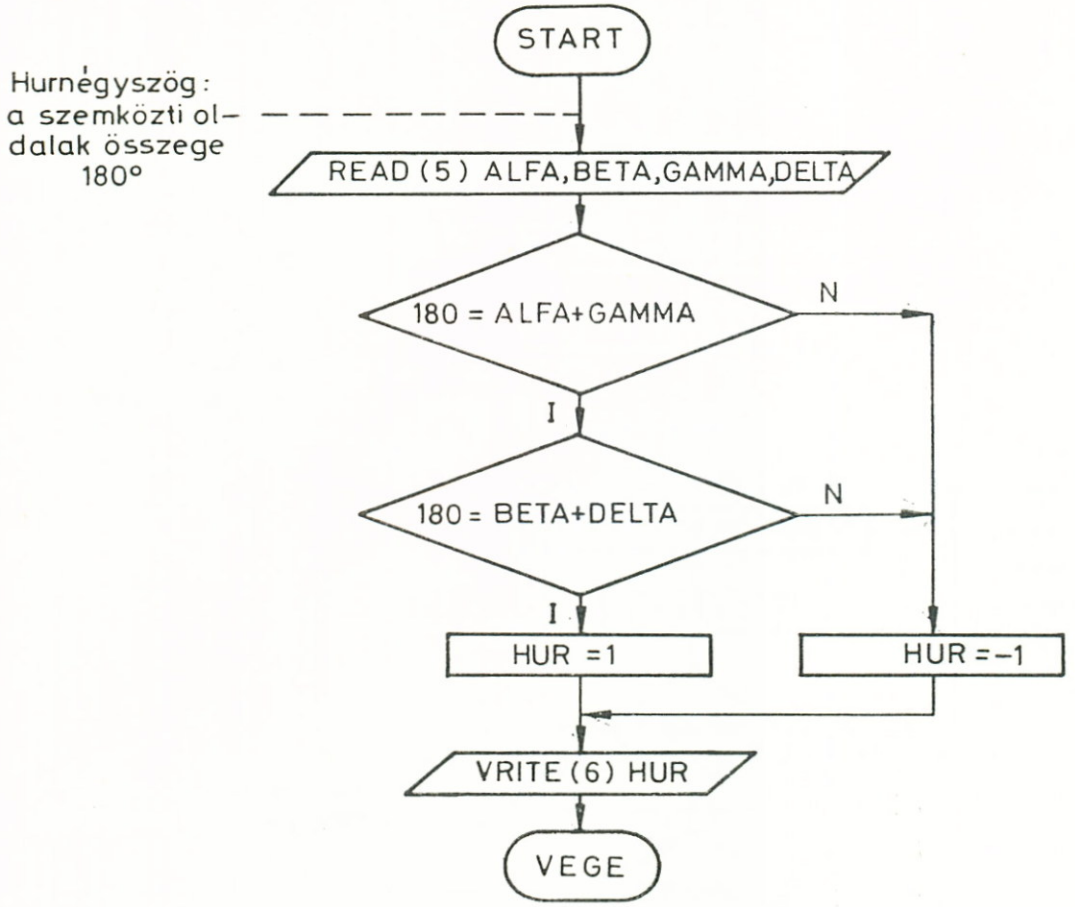
III. fejezet 6. a) feladat



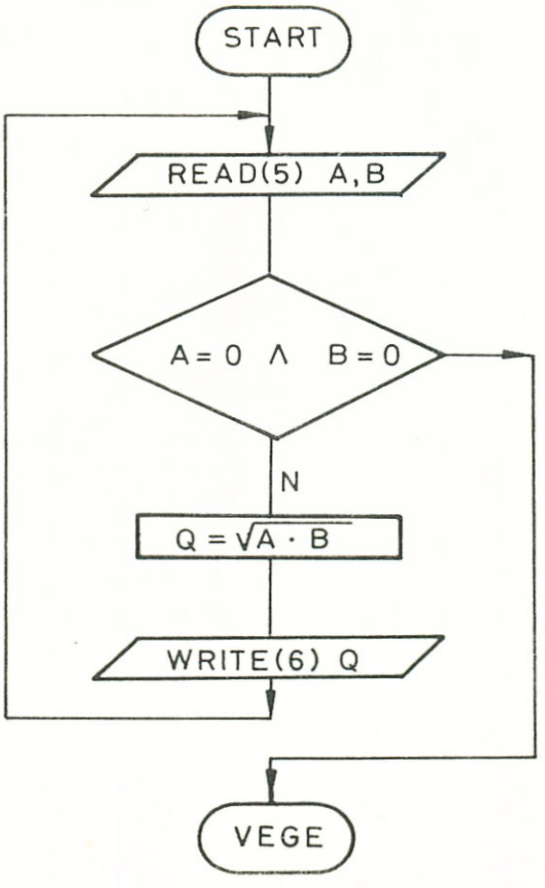
III. fejezet 6. b) feladat

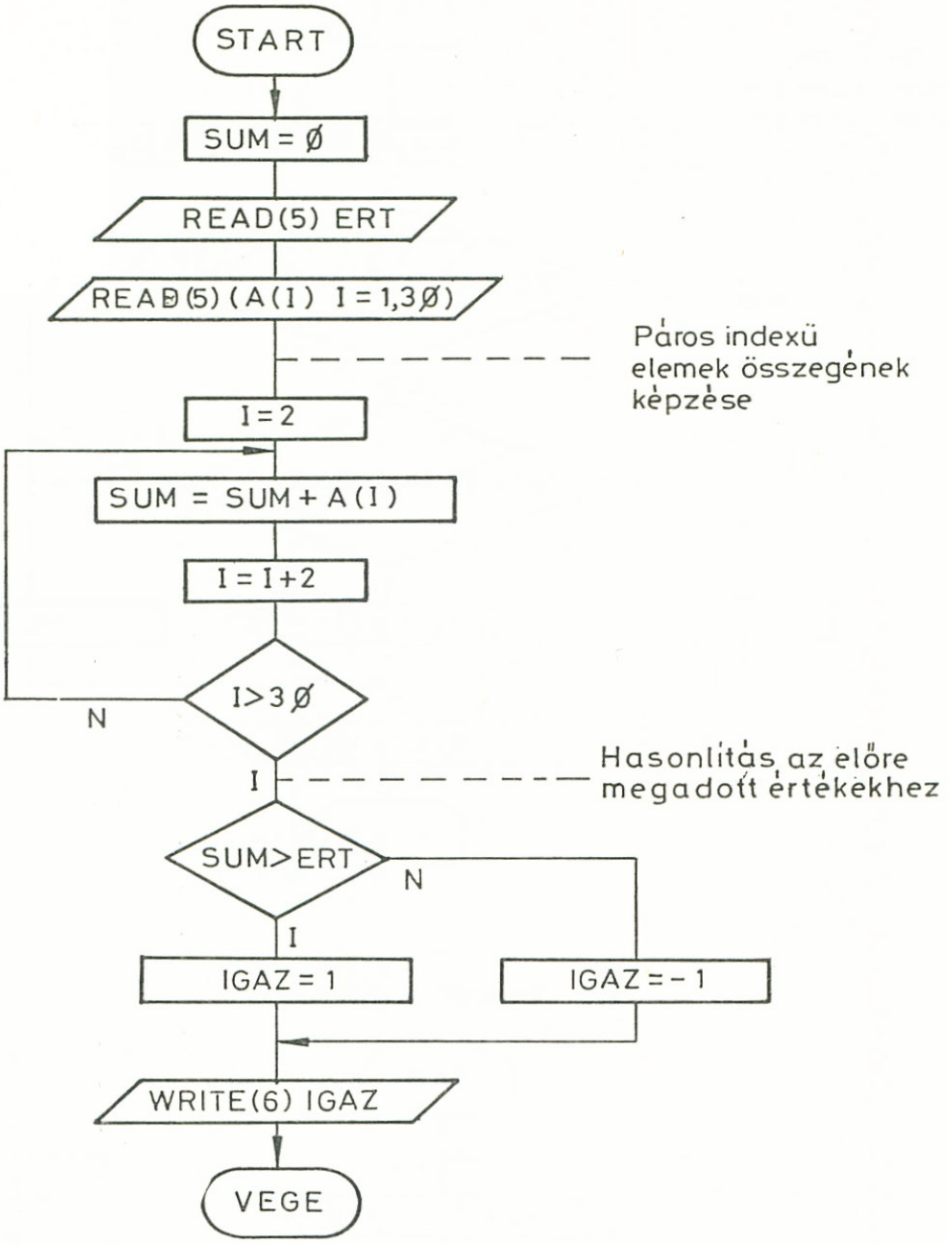


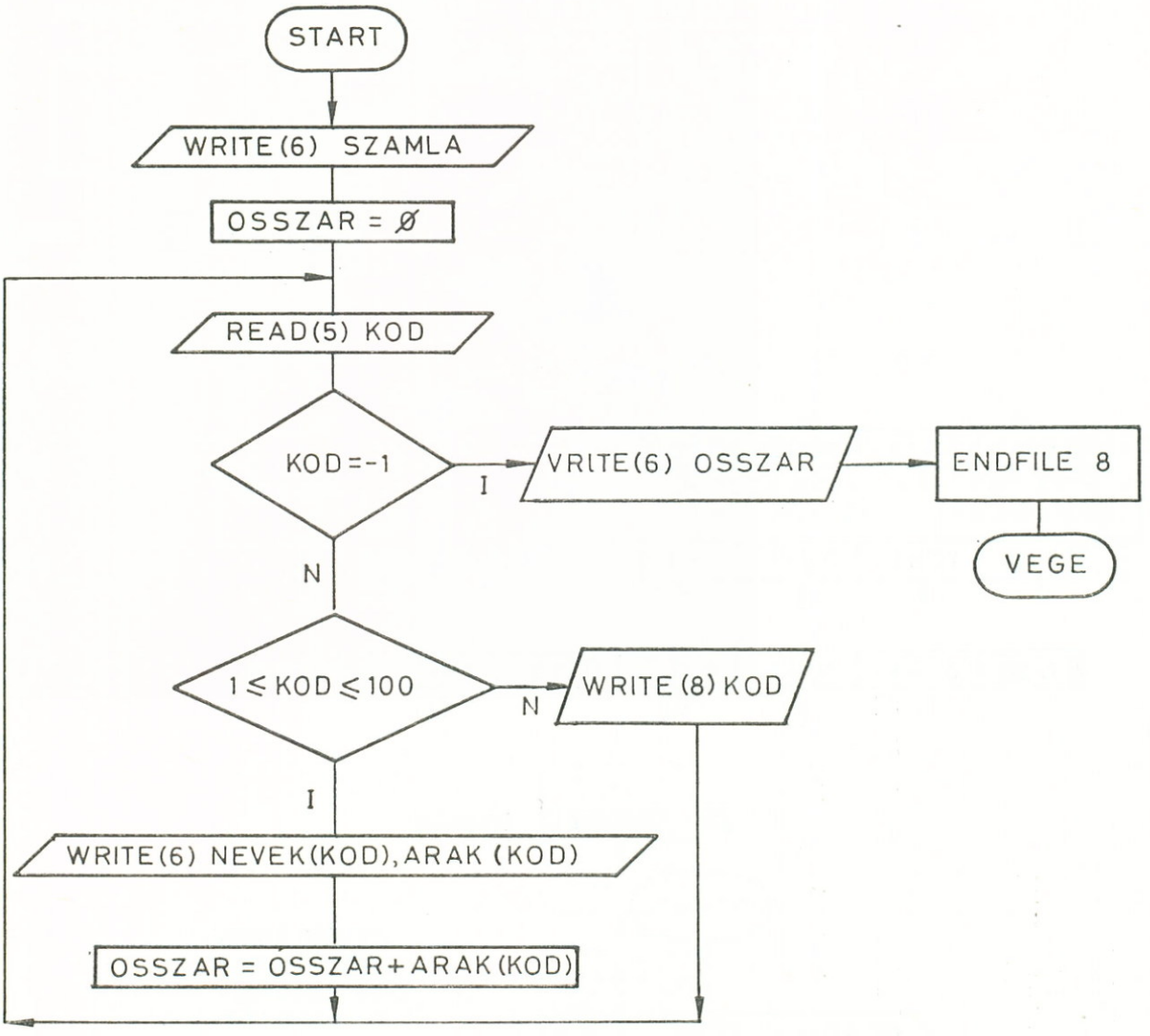
III. fejezet 7. feladat



III. fejezet 8. feladat




START
SUM = Ø
READ(5) ERT
READ(5) (A(I) I = 1,3Ø)
Páros indexü elemek összegének képzése
I = 2
SUM = SUM + A(I)
I = I+2
I>3Ø
N
I
Hasonlítás az előre megadott értékekhez
SUM>ERT
N
I
IGAZ = 1
IGAZ = - 1
WRITE(6) IGAZ
VEGE



Az algoritmus 1 vevő számára tud csak számlát készíteni.

IV. fejezet 1. feladat

1111	0011	1111	0111	1100	1000	= + 378
F	3	F	7	C	8	

1111	0110	1111	1001	1111	0011	= 693
F	6	F	9	F	3	

1111	0101	1111	1001	1111	0110	1101	0110	= − 5966
F	5	F	9	F	6	D	6	

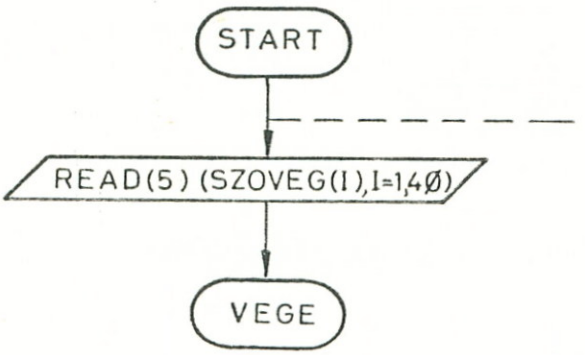
IV. fejezet 2. feladat

0011	0111	1000	1100	= + 378
3	7	8	C	

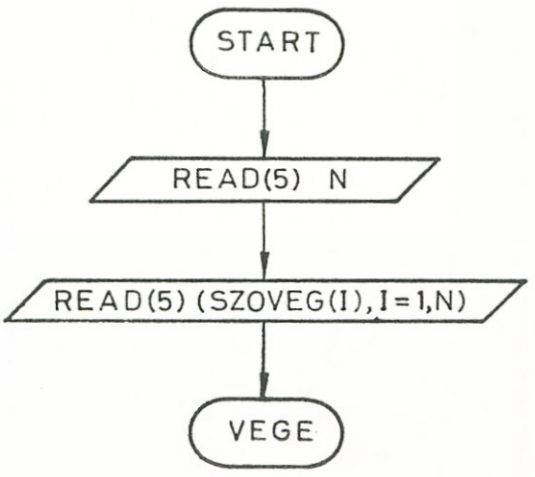
0110	1001	0011	1111	= 693
6	9	3	F	

0000	0101	1001	0110	0110	1101	= − 5966
	5	9	6	6	D	

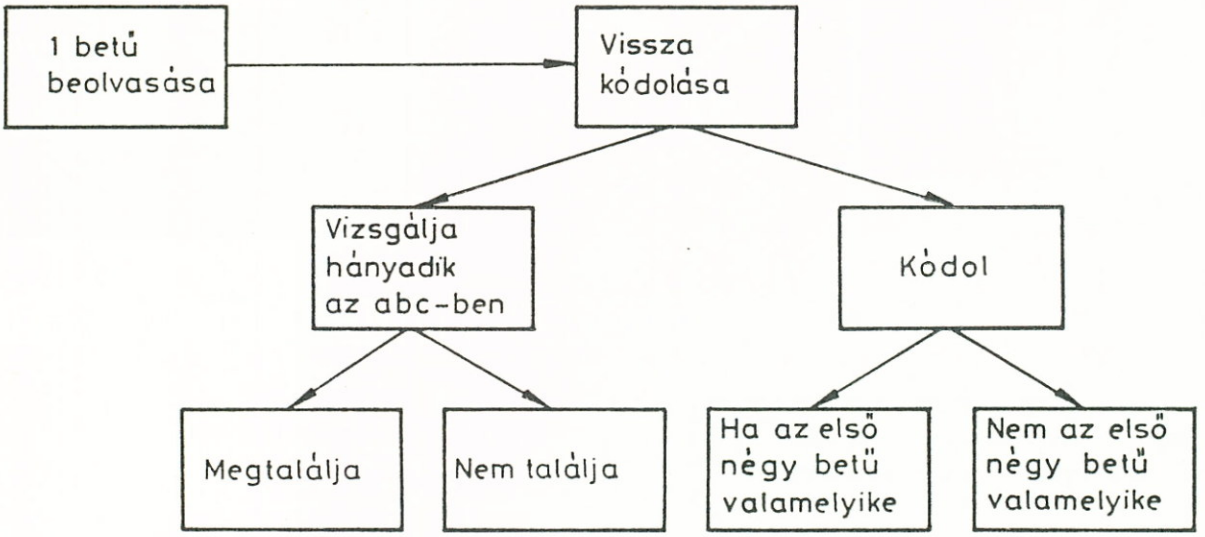
IV. fejezet 3. feladat

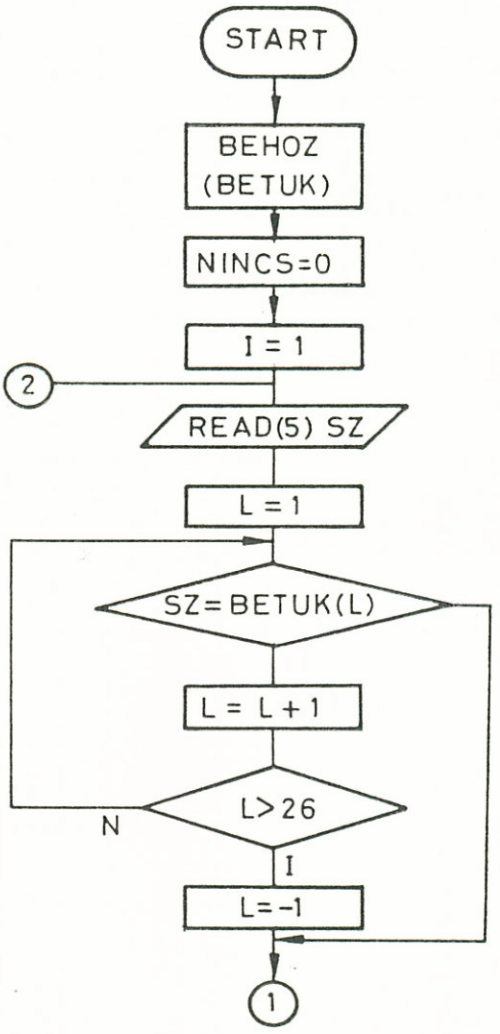


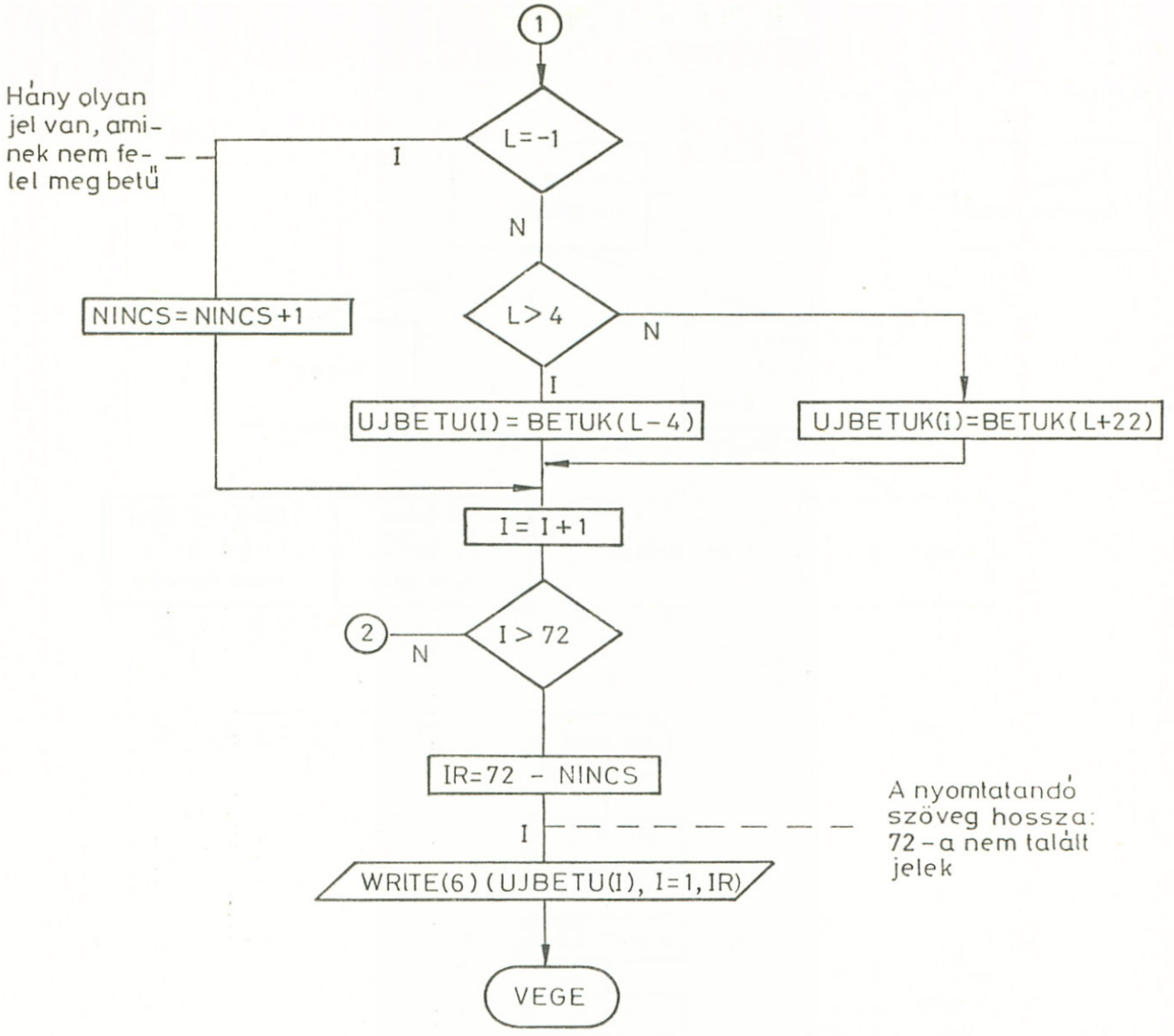
IV. fejezet 4. feladat

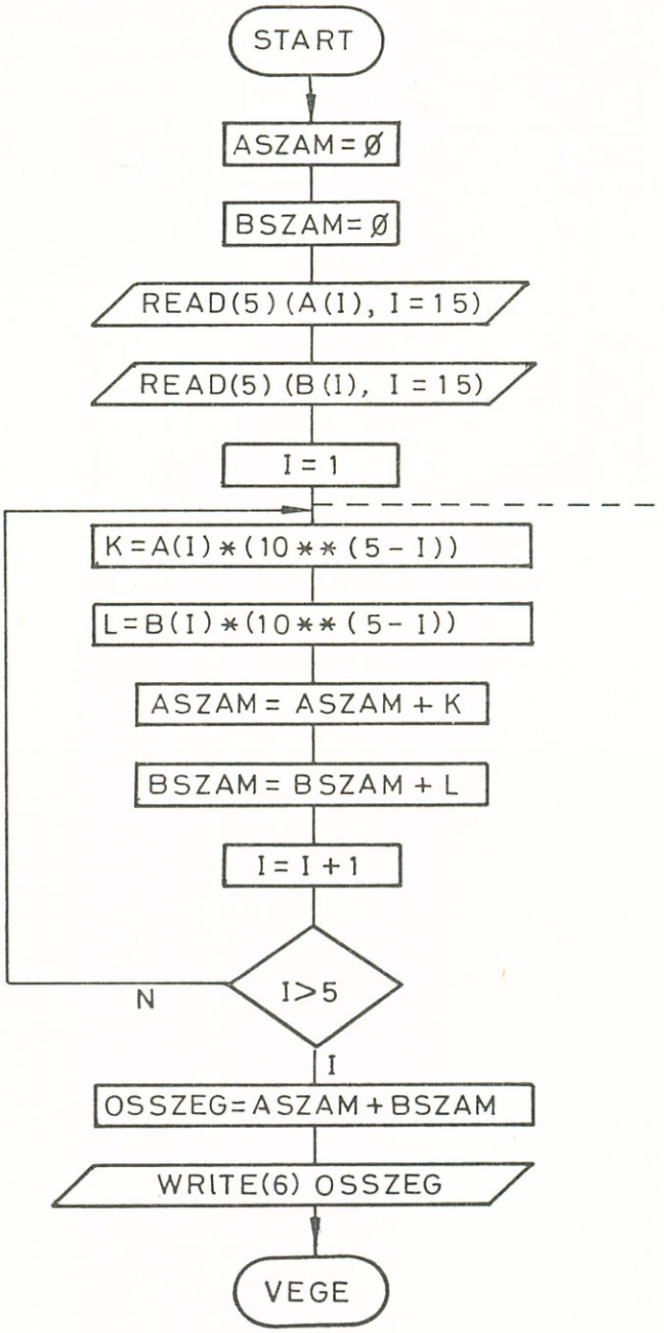


Minden betűre el kell végezni:

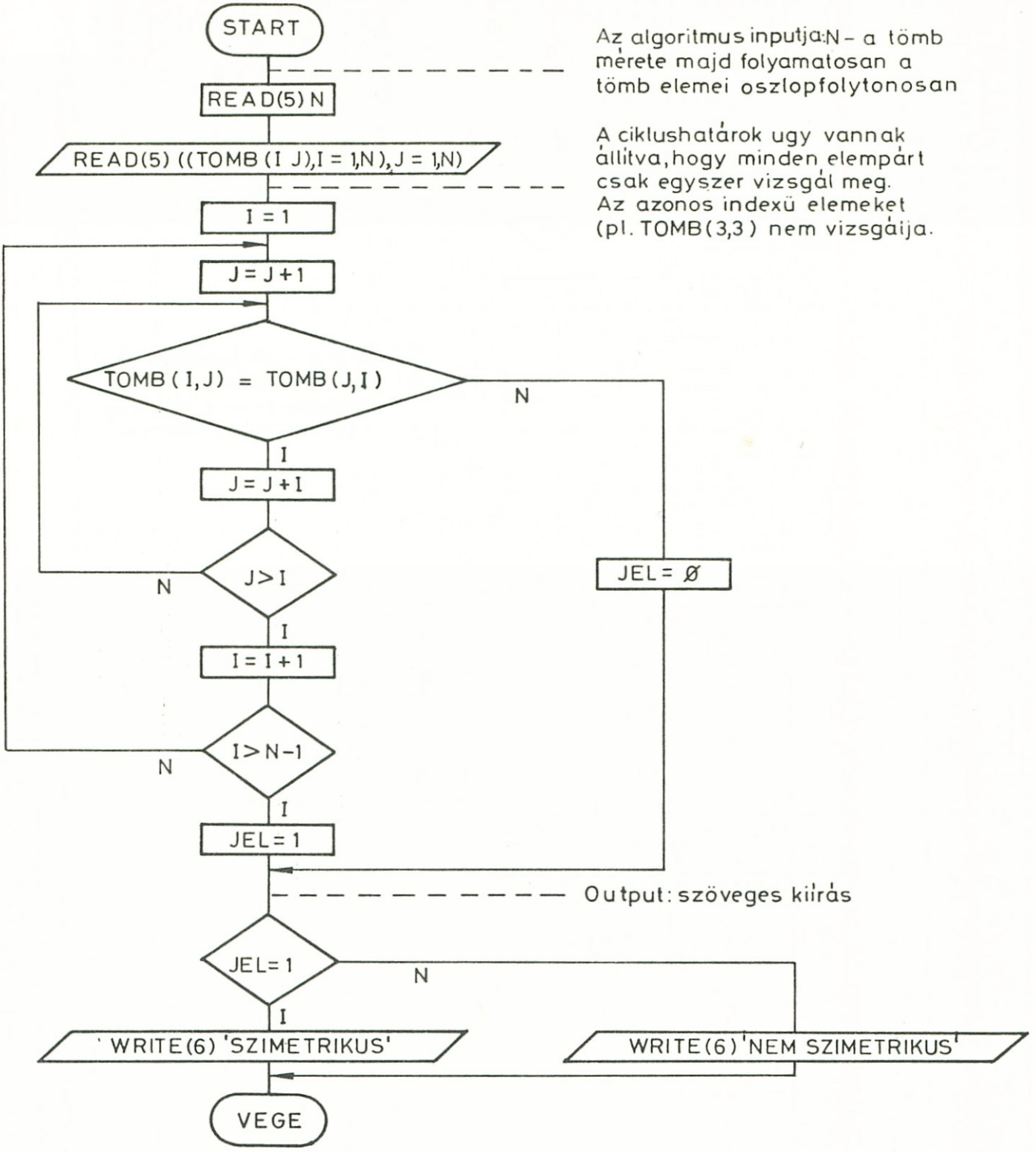


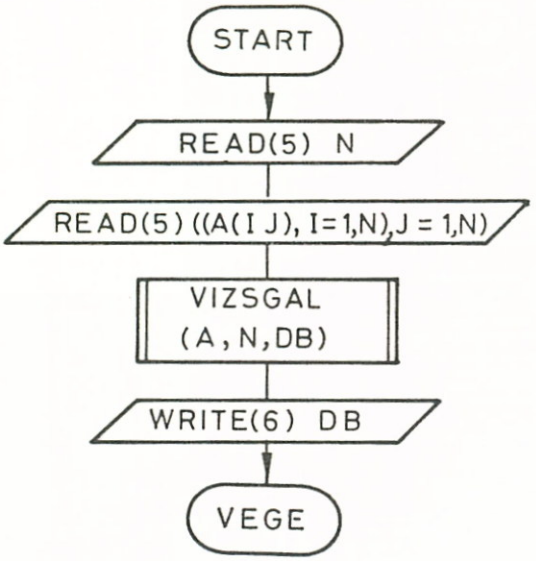


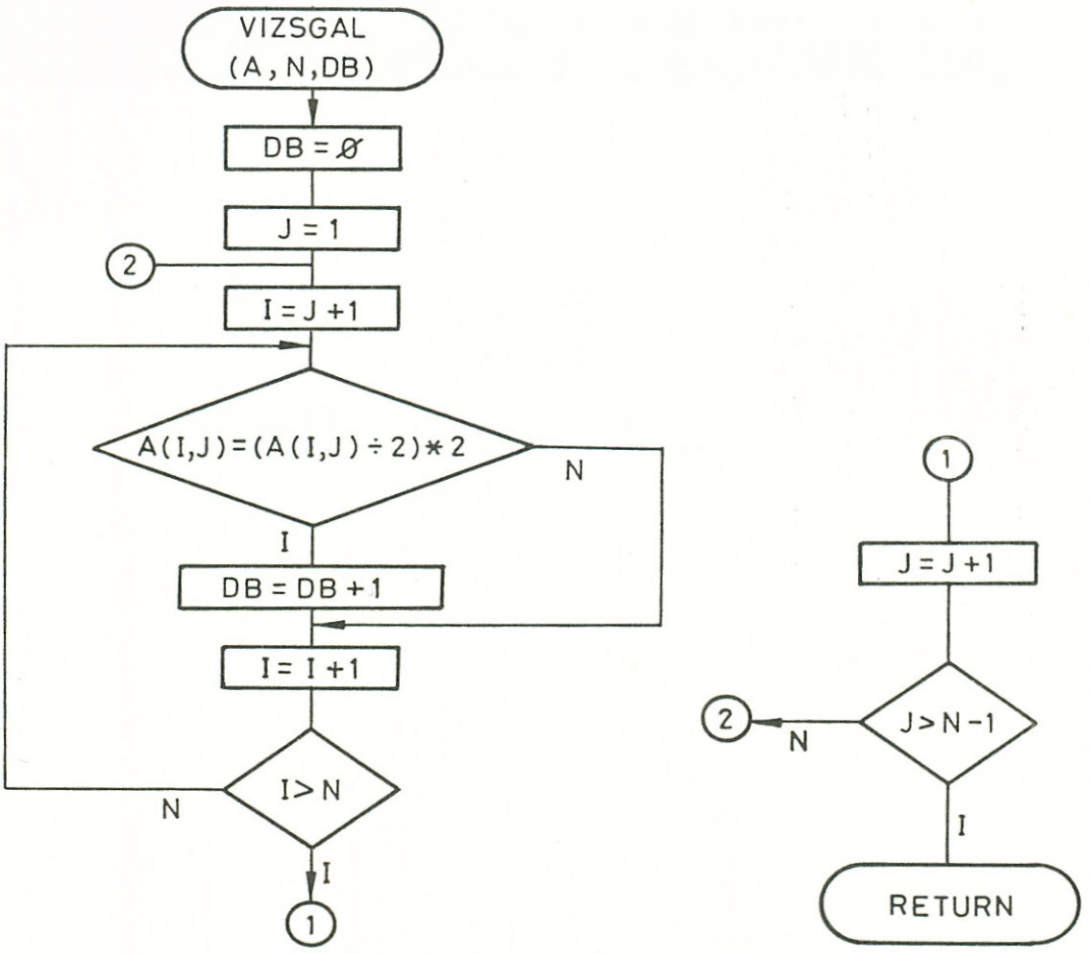

1
Hány olyan jel van, aminek nem felel meg betű
L=-1
I
N
NINCS=NINCS+1
L>4
N
I
UJBETU(I)=BETUK(L-4)
UJBETUK(I)=BETUK(L+22)
I=I+1
2
N
I>72
IR=72 - NINCS
I
A nyomtatandó szöveg hossza: 72 - a nem talált jelek
WRITE(6)(UJBETU(I), I=1,IR)
VEGE

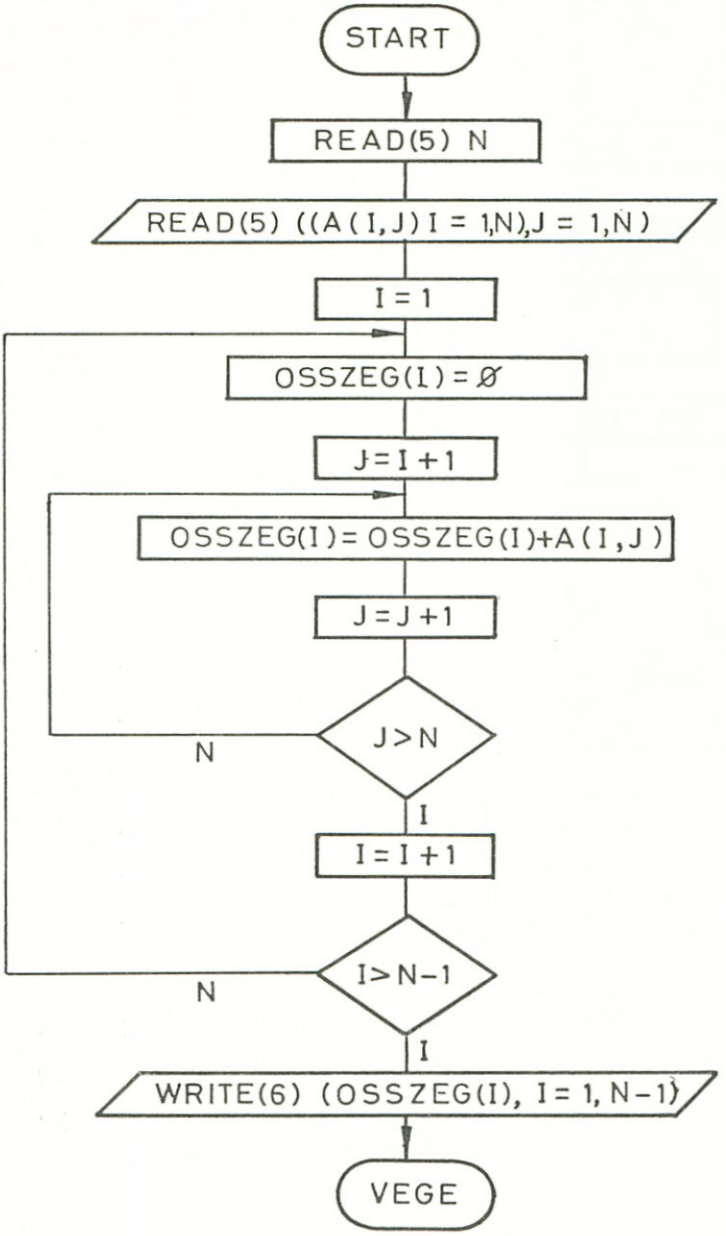


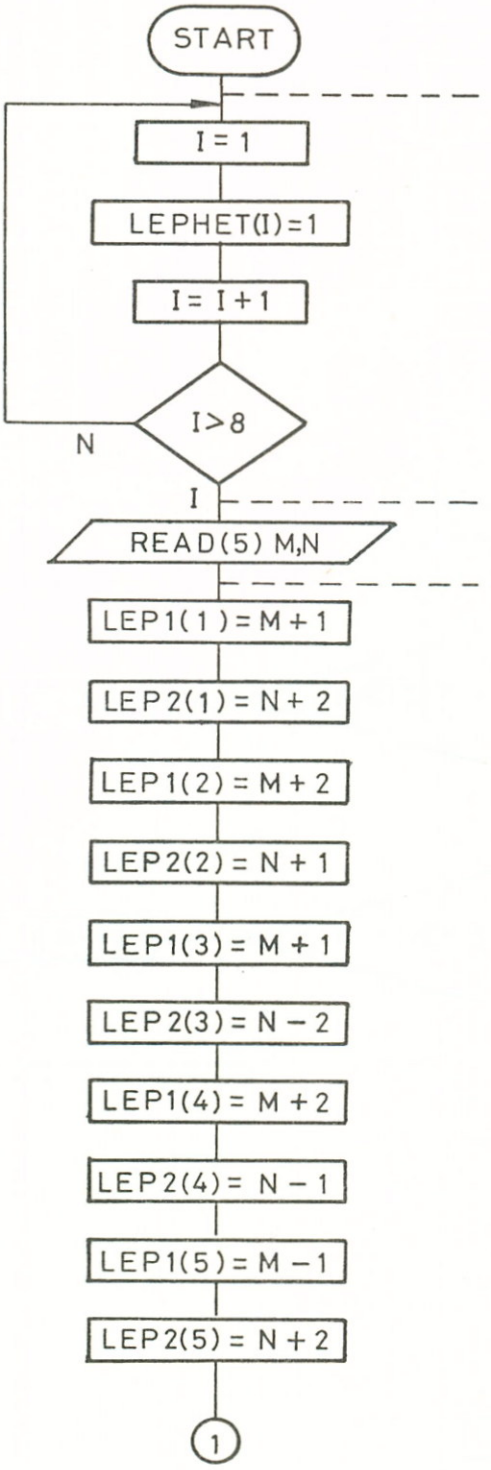
** a hatványozás jele
Minden számjegyet megszorzunk a helyértek szerinti 10 hatvánnyal.


START
READ(5) N
READ(5) ((TOMB (I J),I = 1,N),J = 1,N)
I = 1
J = J + 1
TOMB (I,J) = TOMB (J,I)
N
I
J = J + 1
J>I
N
I
I = I + 1
I>N-1
N
I
JEL = 1
JEL = Ø
Output: szöveges kiírás
JEL= 1
N
I
' WRITE(6) 'SZIMETRIKUS'
WRITE(6) 'NEM SZIMETRIKUS'
VEGE
Az algoritmus inputja: N - a tömb mérete majd folyamatosan a tömb elemei oszlopfolytonosan
A ciklushatárok ugy vannak állitva, hogy minden elempárt csak egyszer vizsgál meg. Az azonos indexü elemeket (pl. TOMB(3,3) nem vizsgálja.


START
READ(5) N
READ(5) ((A(I J), I=1,N),J = 1,N)
VIZSGAL
(A , N,DB)
WRITE(6) DB
VEGE


VIZSGAL
(A , N,DB)
DB = Ø
J = 1
2
I = J +1
A(I,J)=(A(I,J) ÷ 2)* 2
N
I
DB = DB +1
I= I +1
I> N
N
I
1
1
J = J +1
2
N
J>N -1
I
RETURN

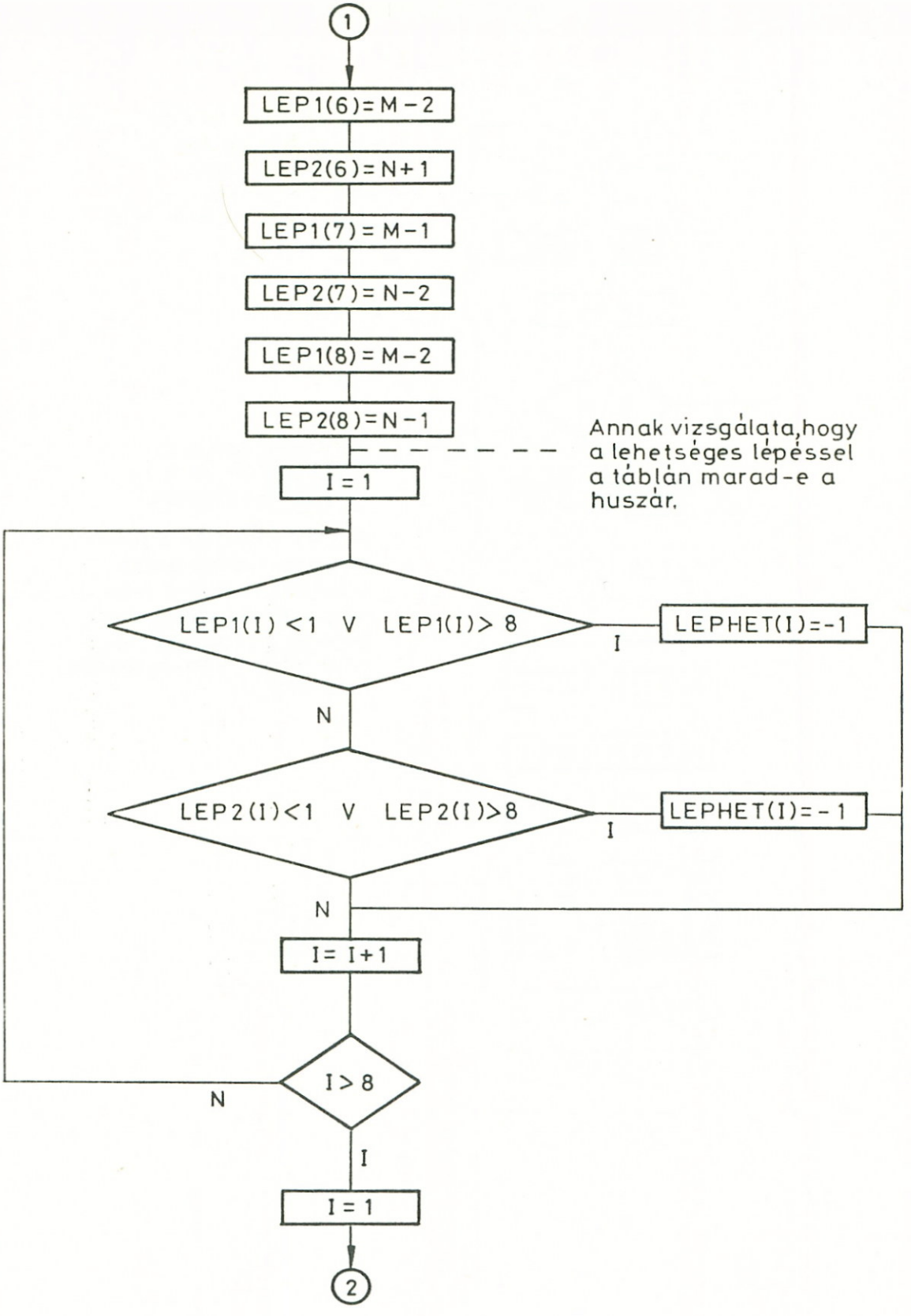

START
READ(5) N
READ(5) ((A(I,J) I = 1,N),J = 1,N)
I = 1
OSSZEG(I) = Ø
J = I + 1
OSSZEG(I) = OSSZEG(I)+A(I,J)
J = J + 1
J > N
N
I
I = I + 1
I > N-1
N
I
WRITE(6) (OSSZEG(I), I = 1, N-1)
VEGE

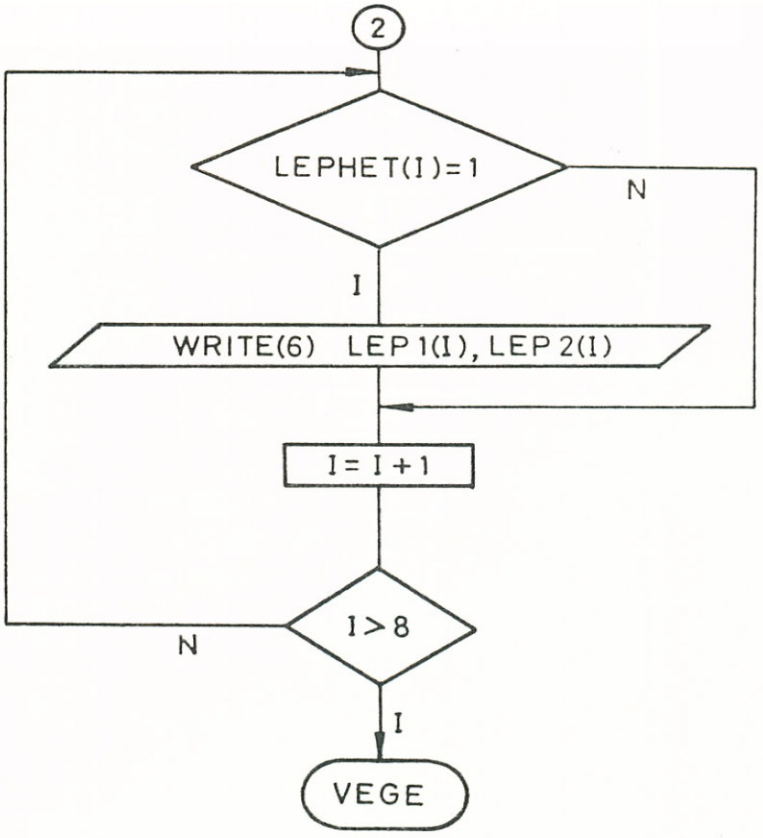


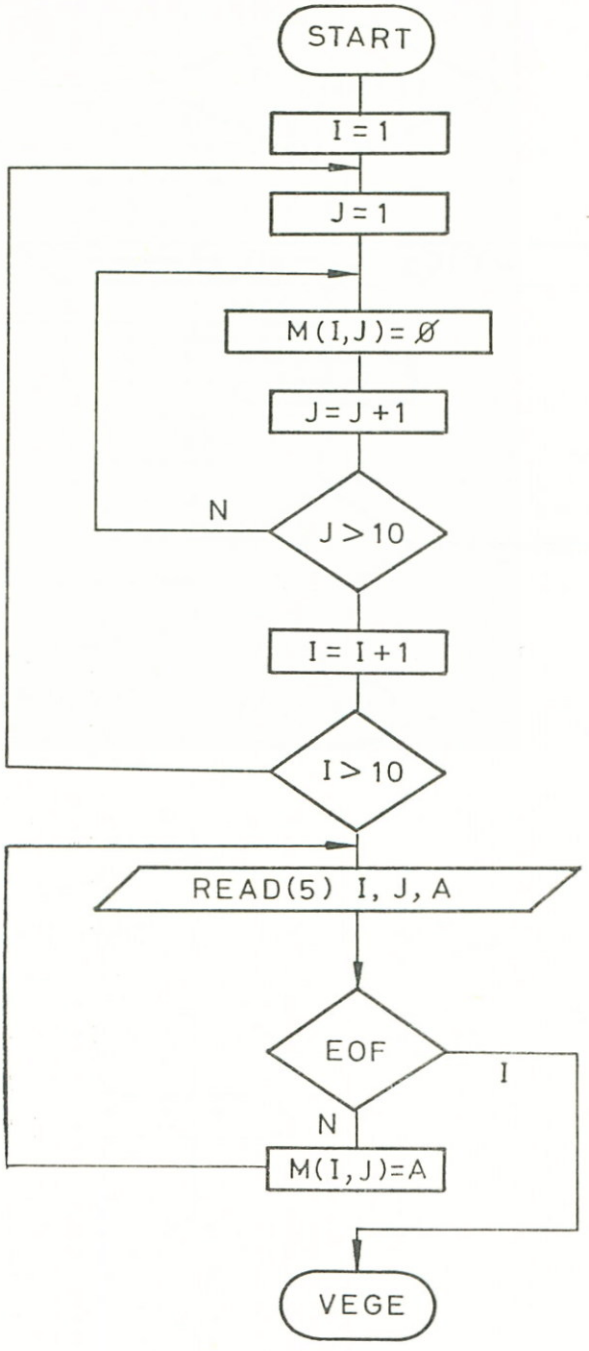
A LEPHET tömb mutatja, hogy hányadik helyre léphet a huszár. Ha 1 - léphet ha -1 - nem léphet

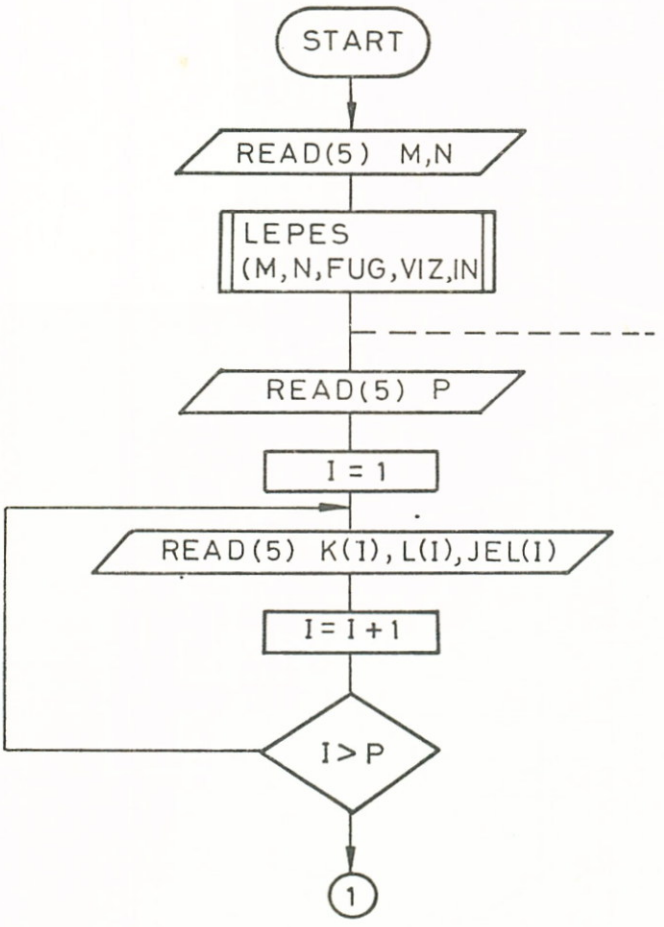
A huszár helye a sakktáblán az N és M adatokkal adott

A LEP1 és LEP2 tömb azonos indexű elemei a „lólépés" szerinti lehetséges lépéseket jelentik. (Annak vizsgálata, hogy ez a táblán van-e, később következik.)

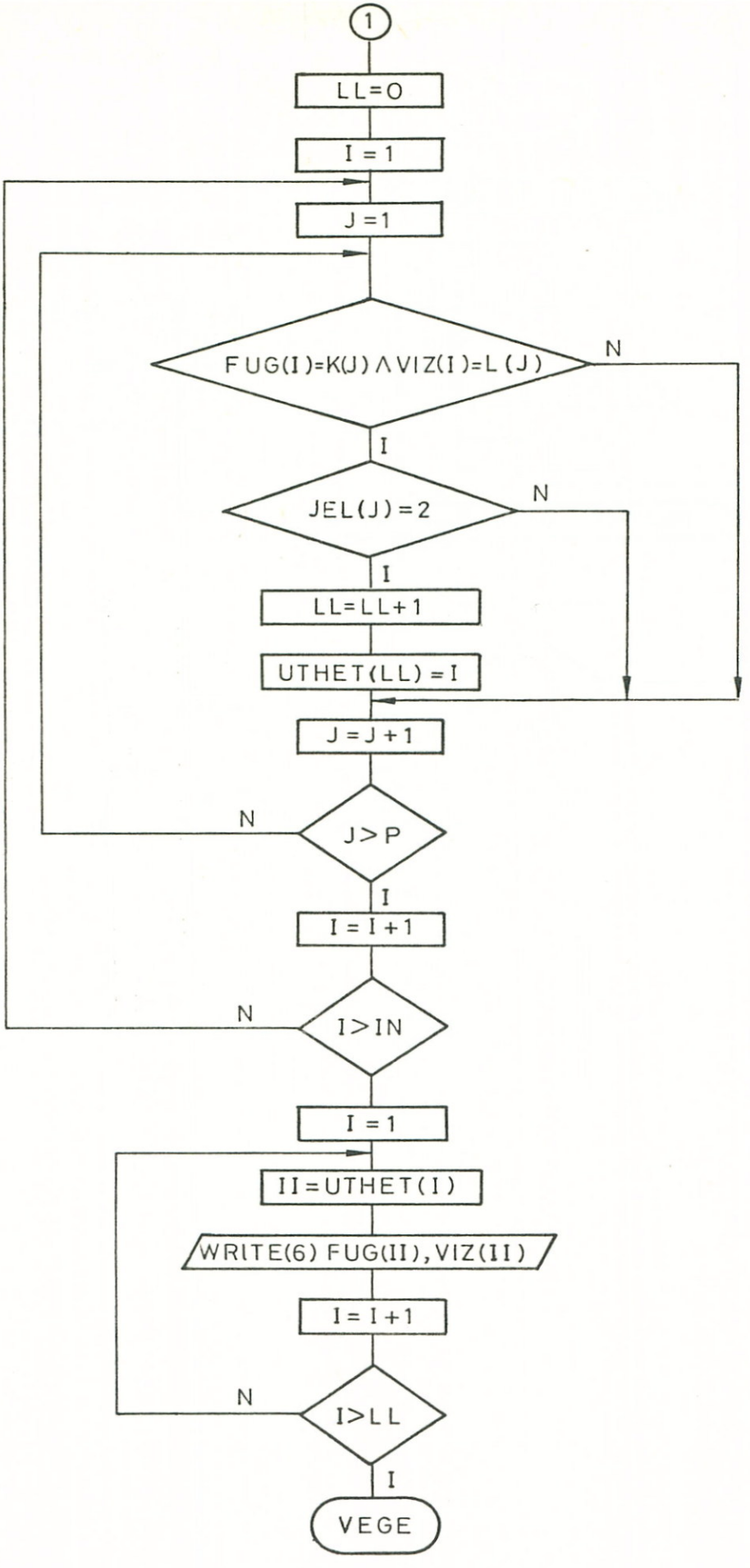

1
LEP1(6)=M-2
LEP2(6)=N+1
LEP1(7)=M-1
LEP2(7)=N-2
LEP1(8)=M-2
LEP2(8)=N-1
Annak vizsgálata,hogy a lehetséges lépéssel a táblán marad-e a huszár.
I=1
LEP1(I)<1 V LEP1(I)>8
I
LEPHET(I)=-1
N
LEP2(I)<1 V LEP2(I)>8
I
LEPHET(I)=-1
N
I=I+1
I>8
N
I
I=1
2

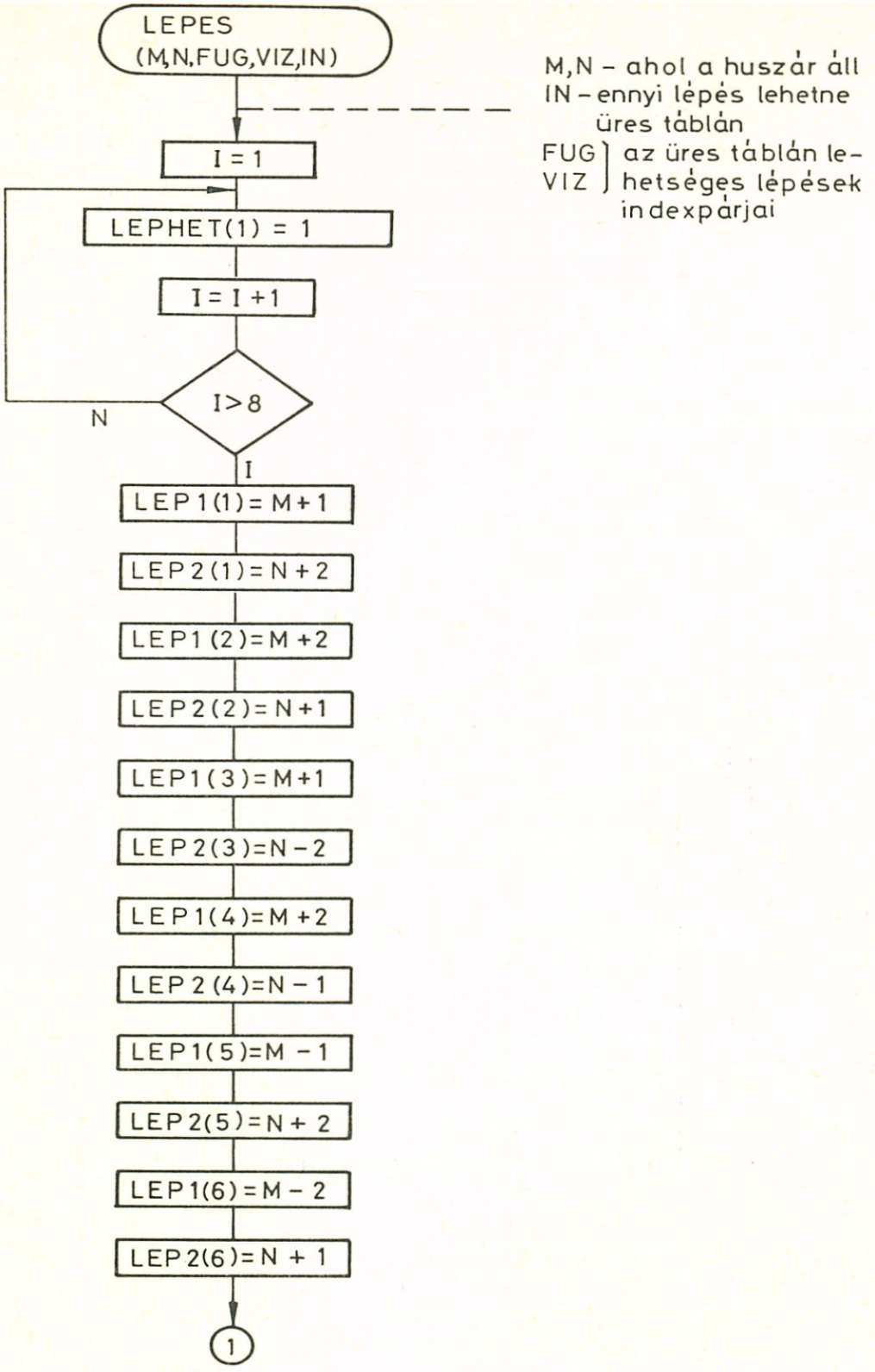

2
LEPHET(I)=1
N
I
WRITE(6) LEP1(I), LEP2(I)
I=I+1
I>8
N
I
VEGE

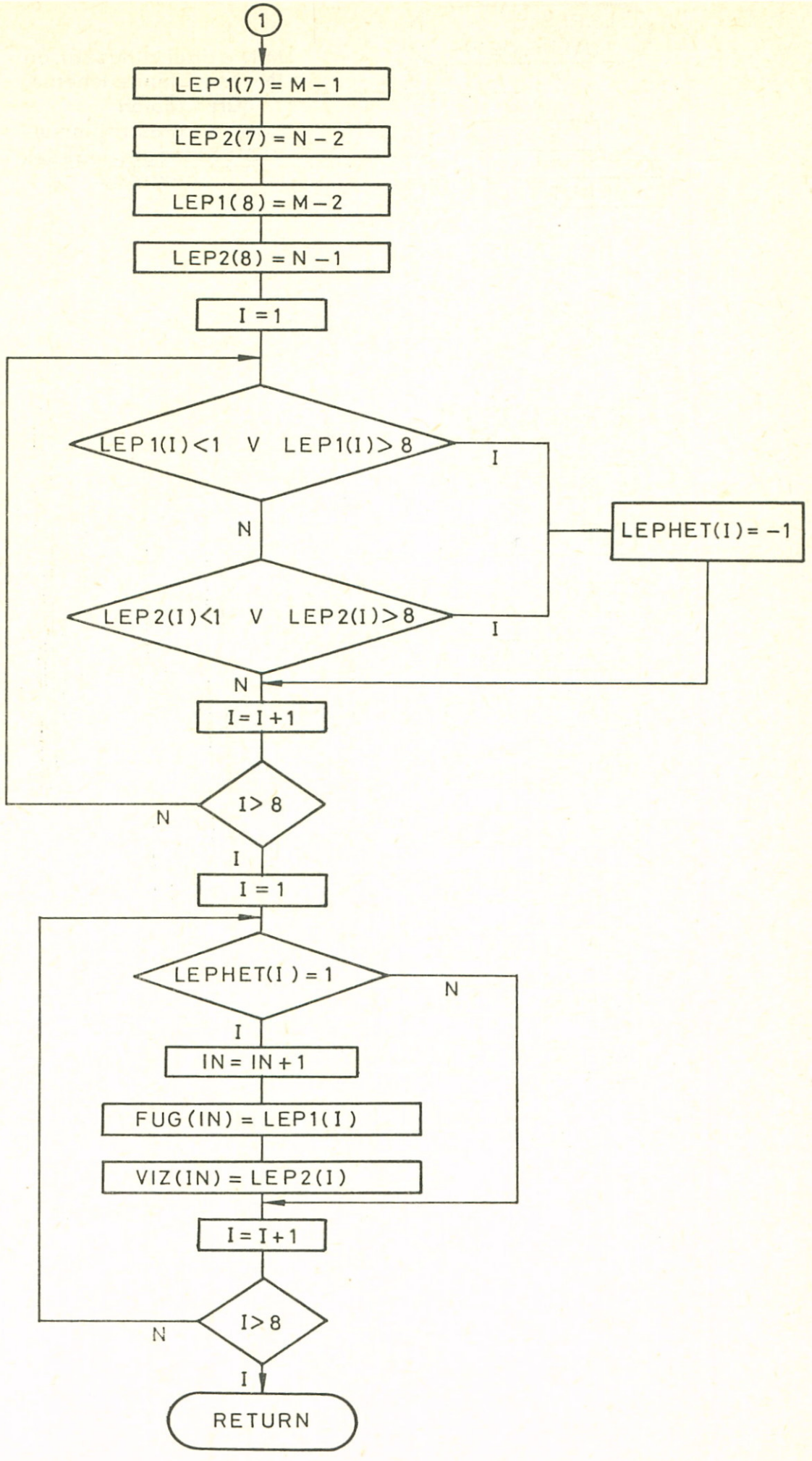

START
I = 1
J = 1
M(I,J) = Ø
J = J + 1
N
J > 10
I = I + 1
I > 10
READ(5) I, J, A
EOF
I
N
M(I,J)=A
VEGE



P a táblán lévő bábuk száma
K(I), L(I) az I. bábu pozíciója
JEL(I) a bábu „színe"


1
LL=0
I = 1
J =1
FUG(I)=K(J) ∧ VIZ(I)=L (J)
N
I
JEL(J) = 2
N
I
LL=LL+1
UTHET(LL) = I
J = J + 1
N
J > P
I
I = I + 1
N
I > IN
I = 1
II = UTHET(I)
WRITE(6) FUG(II), VIZ(II)
I = I + 1
N
I > LL
I
VEGE


LEPES
(M,N,FUG,VIZ,IN)
M,N – ahol a huszár áll
IN – ennyi lépés lehetne üres táblán
FUG } az üres táblán le-
VIZ } hetséges lépések indexpárjai
I = 1
LEPHET(1) = 1
I = I + 1
I>8
N
I
LEP1(1) = M+1
LEP2(1) = N+2
LEP1(2) = M+2
LEP2(2) = N+1
LEP1(3) = M+1
LEP2(3) = N–2
LEP1(4) = M+2
LEP2(4) = N–1
LEP1(5) = M–1
LEP2(5) = N+2
LEP1(6) = M–2
LEP2(6) = N+1
1


1
LEP1(7) = M – 1
LEP2(7) = N – 2
LEP1(8) = M – 2
LEP2(8) = N – 1
I = 1
LEP1(I)<1 V LEP1(I)> 8
I
N
LEPHET(I) = –1
LEP2(I)<1 V LEP2(I)> 8
I
N
I = I + 1
I> 8
N
I
I = 1
LEPHET(I) = 1
N
I
IN = IN + 1
FUG(IN) = LEP1(I)
VIZ(IN) = LEP2(I)
I = I + 1
I> 8
N
I
RETURN

VI. fejezet 1. feladat

```
10  REM  3. FEJEZET 1. FELADAT PROGRAMJA
20  DEFINT  T1,P1,M1,T2,P2,M2,T,P,M
30  INPUT   T1,P1,M1
40  S%=T1*3600+P1*60+M1
50  INPUT T2,P2,M2
60  Z%=T2*3600+P2*60+M2
70  S%=S%+Z%
75  REM  AZ INT FÜGGVÉNY KÉPZI AZ EGÉSZ RÉSZT
80  T=INT(S%/3600)
90  S%=S%-T*3600
100 P=INT(S%/60)
110 M=S%-(P*60)
120 PRINT T,P,M
130 END
```

VI. fejezet 2. feladat

```
10  REM  3. FEJEZET 3. FELADAT PROGRAMJA
15  DEFINT A,B,C,T
20  INPUT A,B,C
30  IF A=B AND A=C THEN T=3 ELSE IF A<>B AND B<>C AND
A<>C THEN T=1 ELSE IF A<>B AND B=C OR A=B AND A<>C OR
A<>B AND B<>C AND A=C THEN T=2
40  PRINT T
50  END
```

VI. fejezet 3. feladat

```
 10  REM  3. FEJEZET 9. FELADAT PROGRAMJA
 20  DIM A(30)
 30  S=0
 40  INPUT E
 50  FOR I=1,30
 60  INPUT A(I)
 70  NEXT I
 80  FOR I=1,30,2
 90  S=S+A(I)
100  NEXT I
110  IF S>E THEN J=1 ELSE J=-1
120  PRINT J
130  END
```

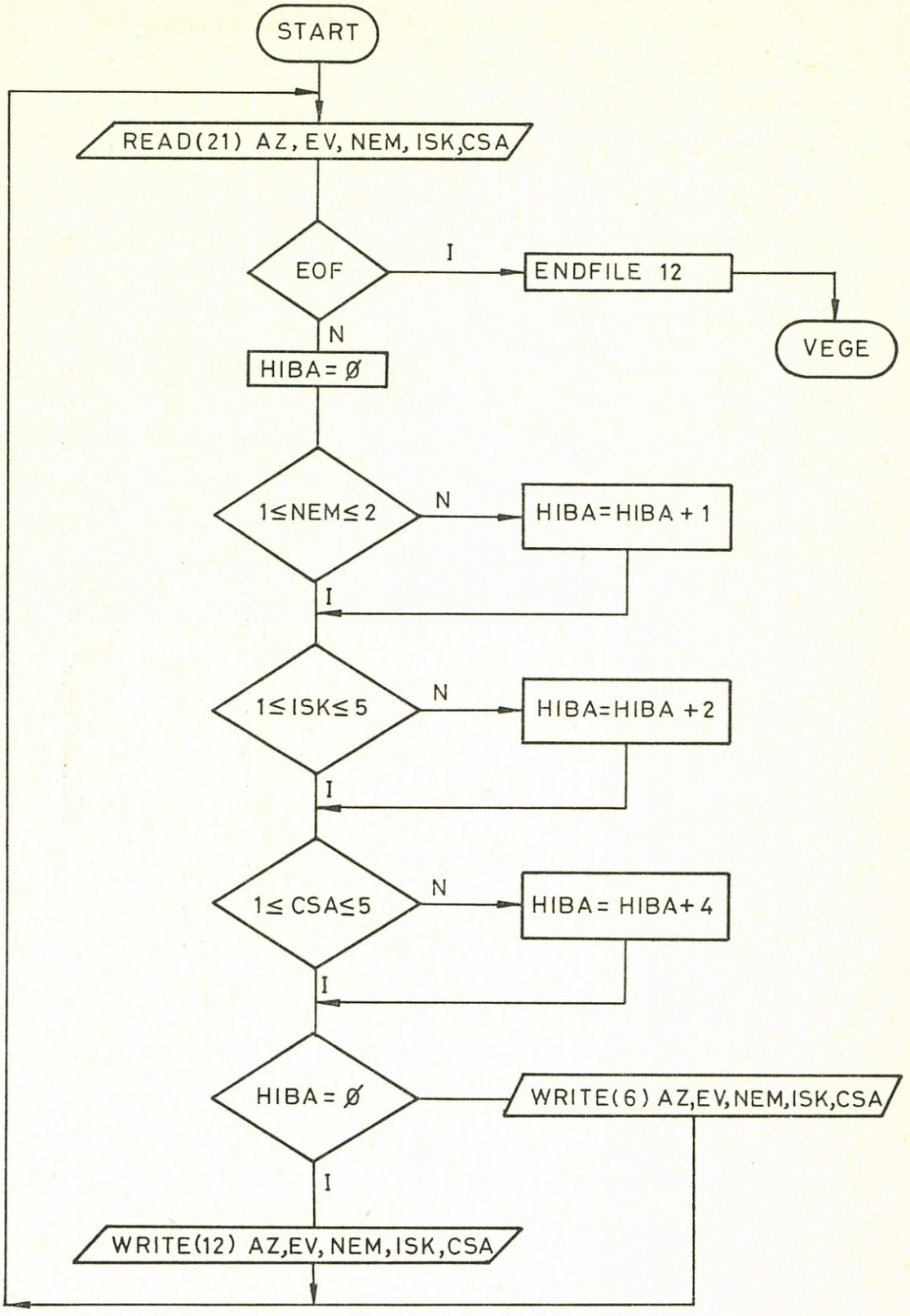
VI. fejezet 4. feladat

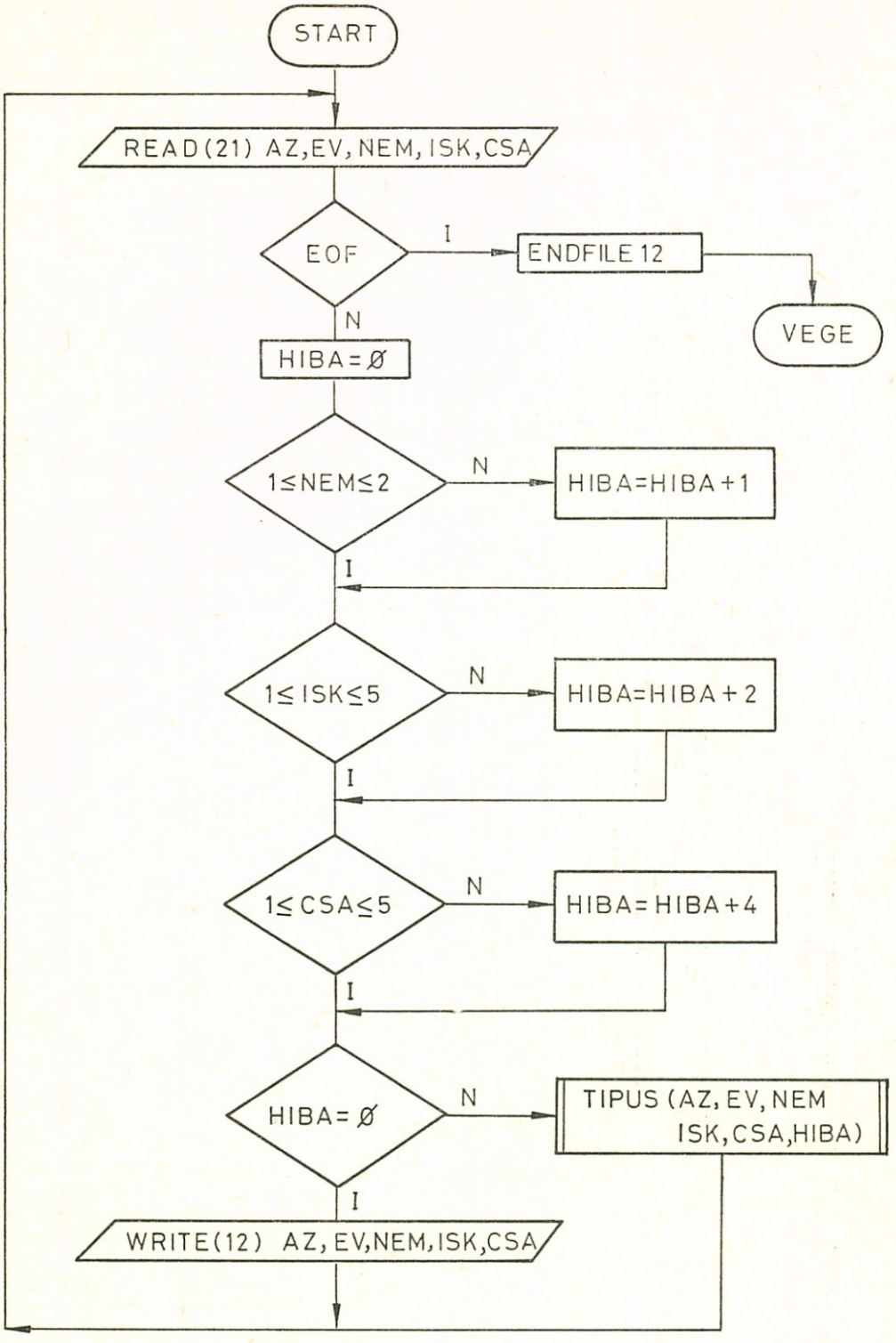
```
 10  REM  5. FEJEZET 3. FELADAT PROGRAMJA
 20  DEFINT A,D
 30  DIM A(30,30)
 40  INPUT N
 50  FOR J=1,N
 60  FOR I=1,N
 70  INPUT A(I,J)
 80  NEXT I
 90  NEXT J
100  GOSUB 130
110  PRINT D
120  END
130  D=0
140  FOR J=1,N-1
150  FOR I=J+1,N
160  IF A(I,J)=INT(A(I,J)/2*2)THEN D=D+1!
170  NEXT I
180  NEXT J
190  RETURN
200  END
```

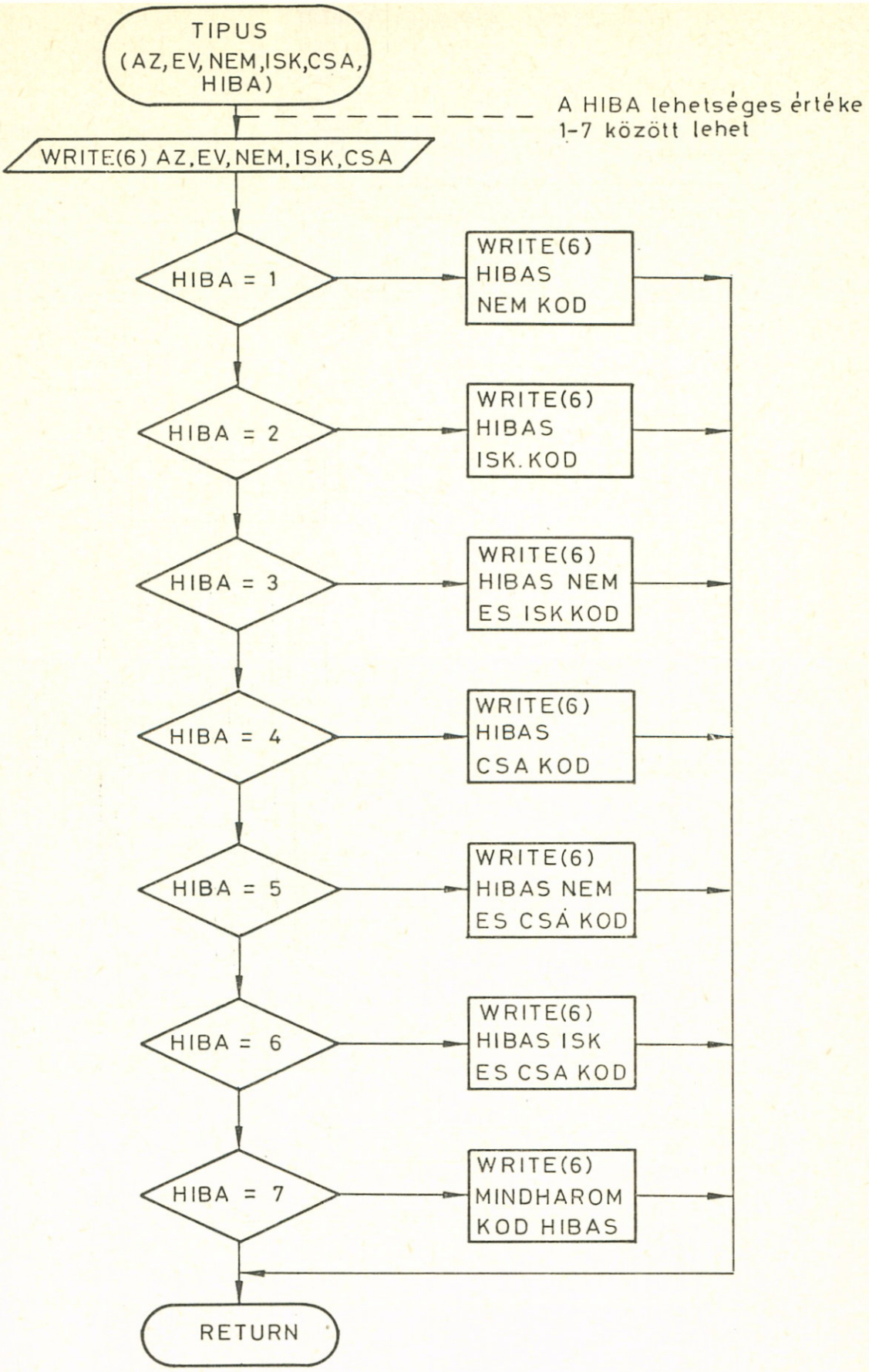
```
 10  REM  5. FEJEZET 4. FELADAT PROGRAMJA
 20  DIM A(30,30),S(30)
 30  INPUT N
 40  FOR J=1,N
 50  FOR I=1,M
 60  INPUT A(I,J)
 70  NEXT I
 80  NEXT J
 90  FOR I=1,N-1
100  S(I)=0
110  FOR J=I+1,N
120  S(I)=S(I)+A(I,J)
130  NEXT J
140  NEXT I
150  FOR I=1,N-1
160  PRINT S(I)
170  NEXT I
180  END
```

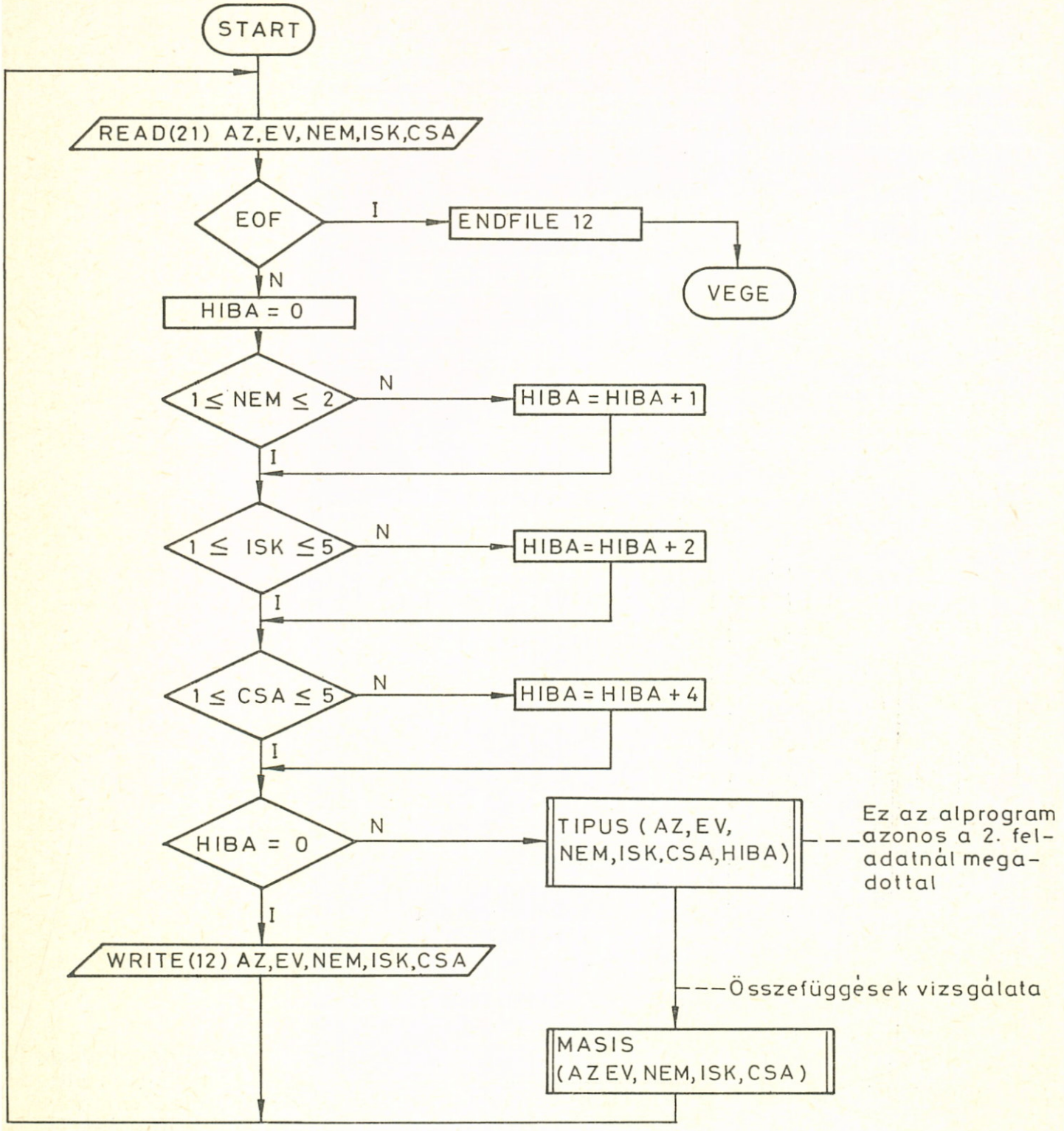
```
 10  REM  5. FEJEZET 5. FELADAT PROGRAMJA
 20  DIM L(8),V(8),F(8)
 30  FOR I=1,8
 40  L(I)=1
 50  NEXT I
 60  INPUT M,N
 70  F(1)=M+1
 80  V(1)=N+2
 90  F(2)=M+2
100  V(2)=N+1
110  F(3)=M+1
120  V(3)=N-2
130  F(4)=M+2
140  V(4)=N-1
150  F(5)=M-1
160  V(5)=N+2
170  F(6)=M-2
180  V(6)=N+1
190  F(7)=M-1
200  V(7)=N-2
210  F(8)=M-2
220  V(8)=N-1
230  FOR I=1,8
240  IF F(I)<1 OR F(I)>8 THEN L(I)=-1
250  IF(V(I)<1 OR V(I)>8 THEN L(I)=-1
260  NEXT I
270  FOR I=1,8
280  IF L(I)=1 THEN PRINT V(I),F(I)
290  NEXT I
300  END
```

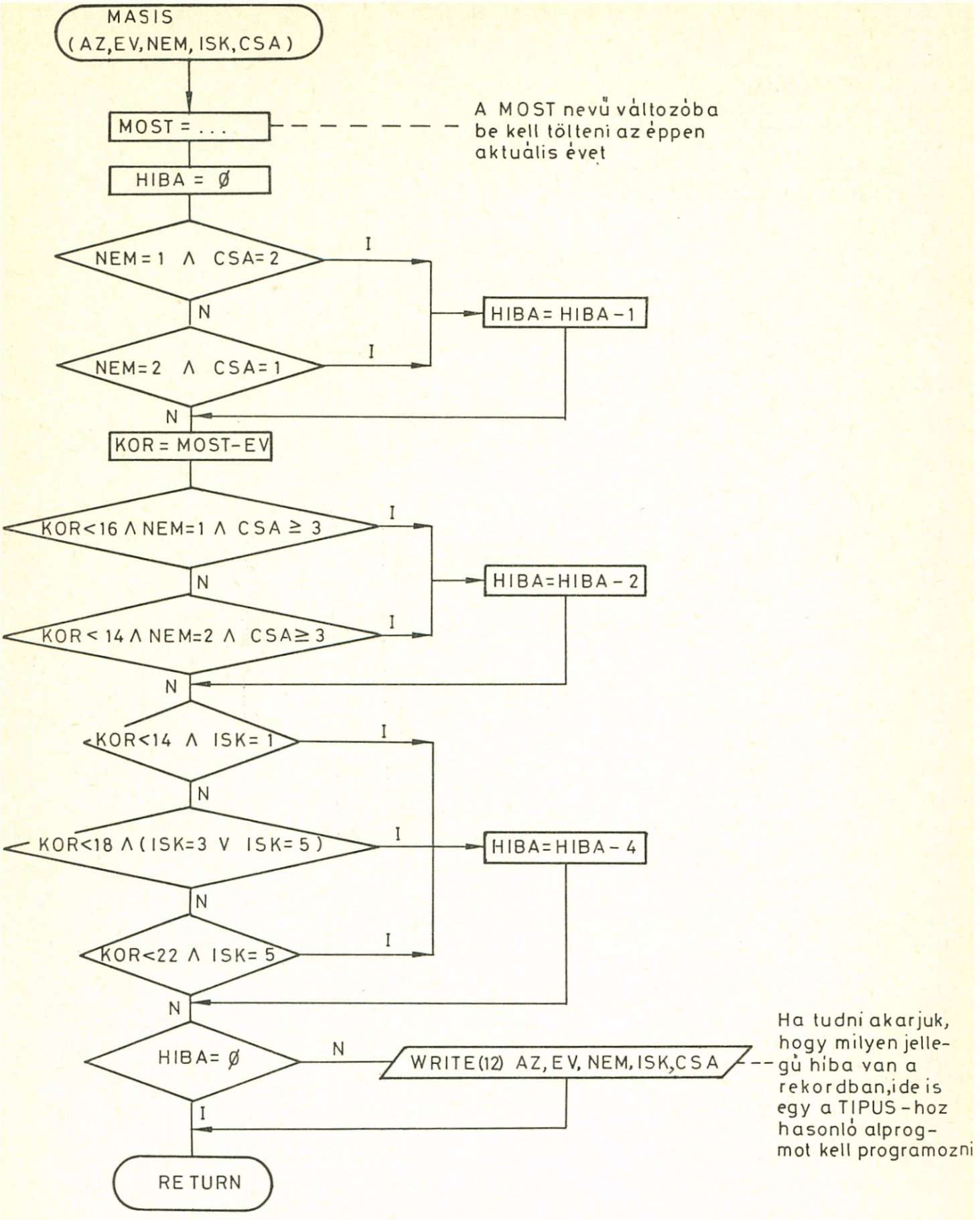
```
 1Ø  REM  5. FEJEZET 6. FELADAT PROGRAMJA
 2Ø  DIM M (1Ø,1Ø)
 3Ø  FOR I=1,1Ø
 4Ø  FOR J=1,1Ø
 5Ø  M(I,J)=Ø
 6Ø  NEXT J
 7Ø   NEXT I
 8Ø  INPUT I,J,A
 85  REM  FAJL VEGE HELYETT NEGATIV 1-IG MÜKÖDIK
 9Ø  IF I<Ø THEN 12Ø
1ØØ  M(I,J)=A
11Ø  GO TO 8Ø
12Ø  END
```

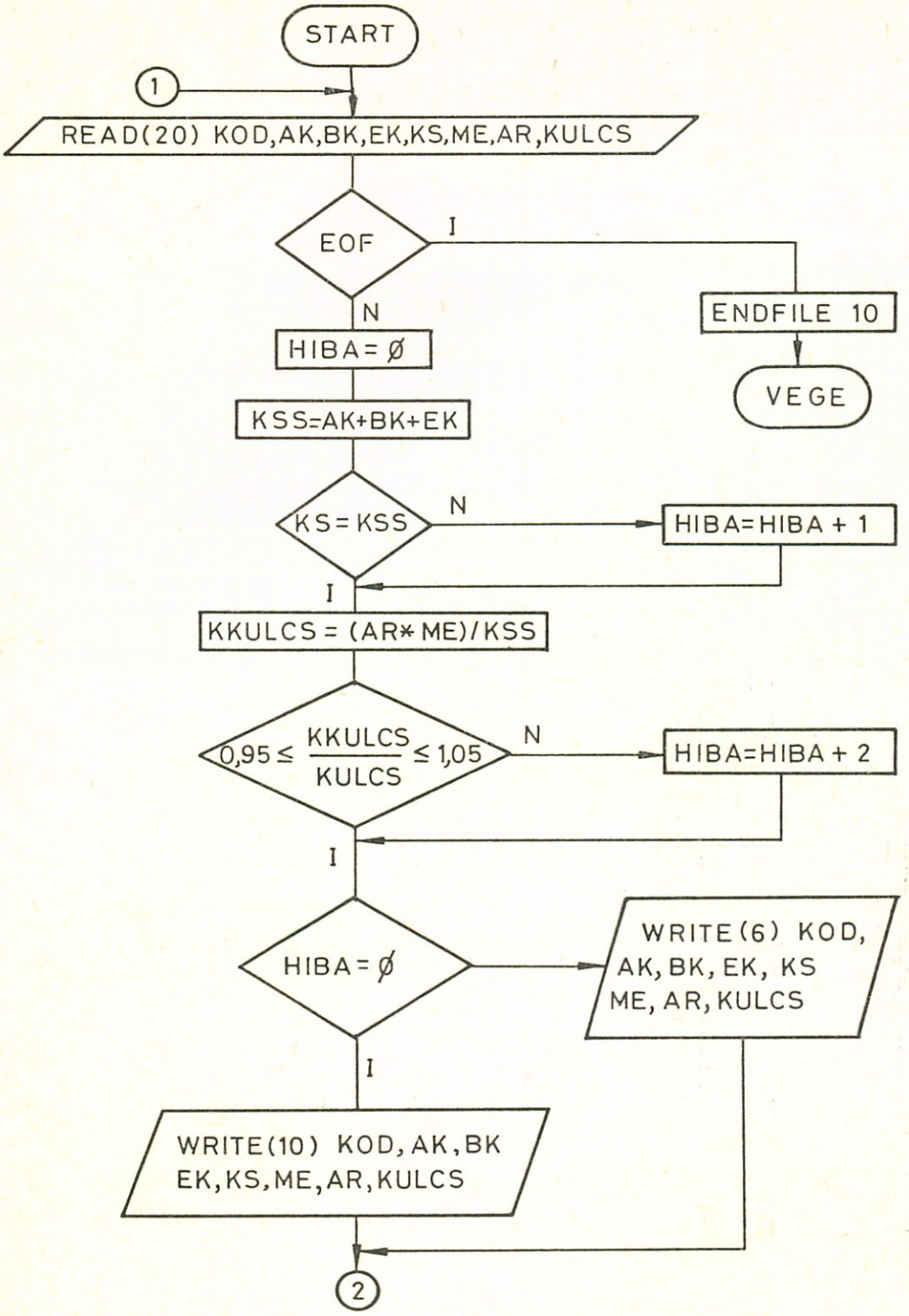

START
READ(21) AZ, EV, NEM, ISK,CSA
EOF
I
ENDFILE 12
VEGE
N
HIBA= Ø
1≤NEM≤2
N
HIBA=HIBA + 1
I
1≤ISK≤5
N
HIBA=HIBA +2
I
1≤ CSA≤5
N
HIBA= HIBA+4
I
HIBA = Ø
WRITE(6) AZ,EV,NEM,ISK,CSA
I
WRITE(12) AZ,EV, NEM, ISK, CSA

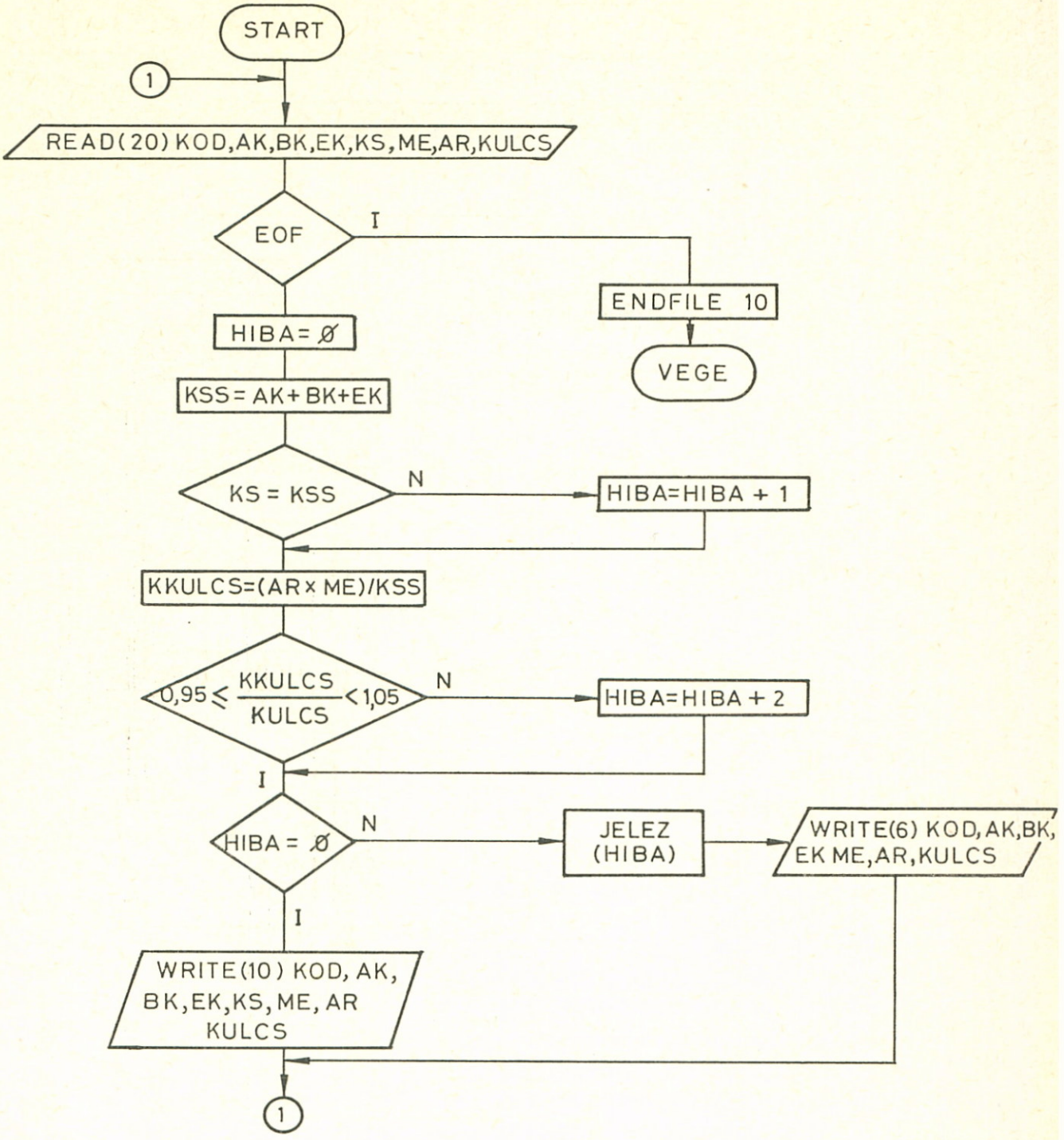

START
READ(21) AZ,EV,NEM,ISK,CSA
EOF
I
ENDFILE 12
VEGE
N
HIBA=Ø
1≤NEM≤2
N
HIBA=HIBA+1
I
1≤ISK≤5
N
HIBA=HIBA+2
I
1≤CSA≤5
N
HIBA=HIBA+4
I
HIBA=Ø
N
TIPUS (AZ, EV, NEM
ISK,CSA,HIBA)
I
WRITE(12) AZ, EV,NEM,ISK,CSA

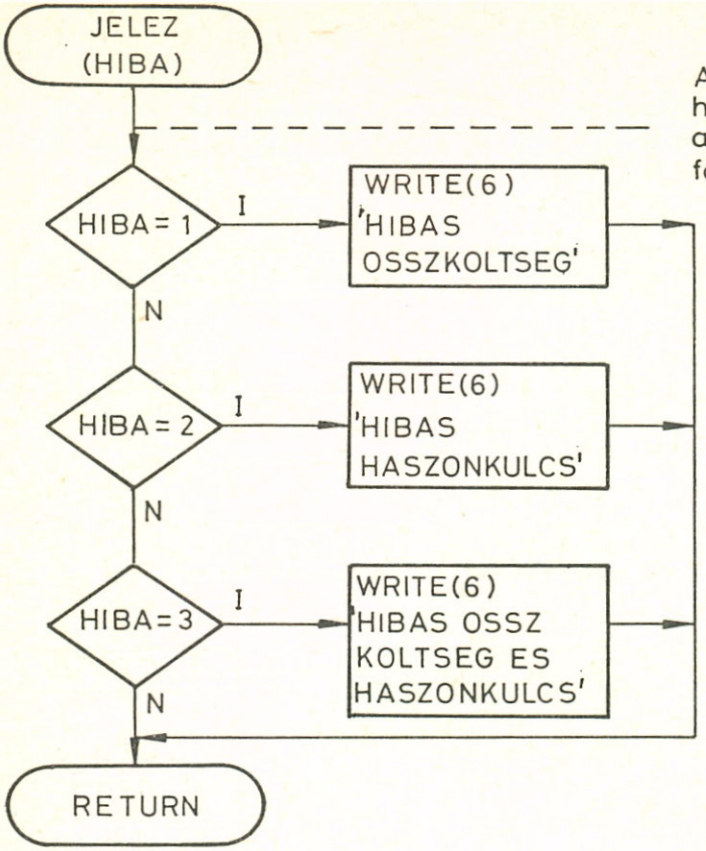

TIPUS
(AZ,EV,NEM,ISK,CSA,
HIBA)
A HIBA lehetséges értéke
1-7 között lehet
WRITE(6) AZ,EV,NEM,ISK,CSA
HIBA = 1
WRITE(6)
HIBAS
NEM KOD
HIBA = 2
WRITE(6)
HIBAS
ISK.KOD
HIBA = 3
WRITE(6)
HIBAS NEM
ES ISK KOD
HIBA = 4
WRITE(6)
HIBAS
CSA KOD
HIBA = 5
WRITE(6)
HIBAS NEM
ES CSÁ KOD
HIBA = 6
WRITE(6)
HIBAS ISK
ES CSA KOD
HIBA = 7
WRITE(6)
MINDHAROM
KOD HIBAS
RETURN


START
READ(21) AZ,EV,NEM,ISK,CSA
EOF
I
ENDFILE 12
VEGE
N
HIBA = 0
1 ≤ NEM ≤ 2
N
HIBA = HIBA + 1
I
1 ≤ ISK ≤ 5
N
HIBA = HIBA + 2
I
1 ≤ CSA ≤ 5
N
HIBA = HIBA + 4
I
HIBA = 0
N
TIPUS (AZ, EV, NEM, ISK, CSA, HIBA)
Ez az alprogram azonos a 2. feladatnál megadottal
I
WRITE(12) AZ,EV,NEM,ISK,CSA
Összefüggések vizsgálata
MASIS
(AZ EV, NEM, ISK, CSA)

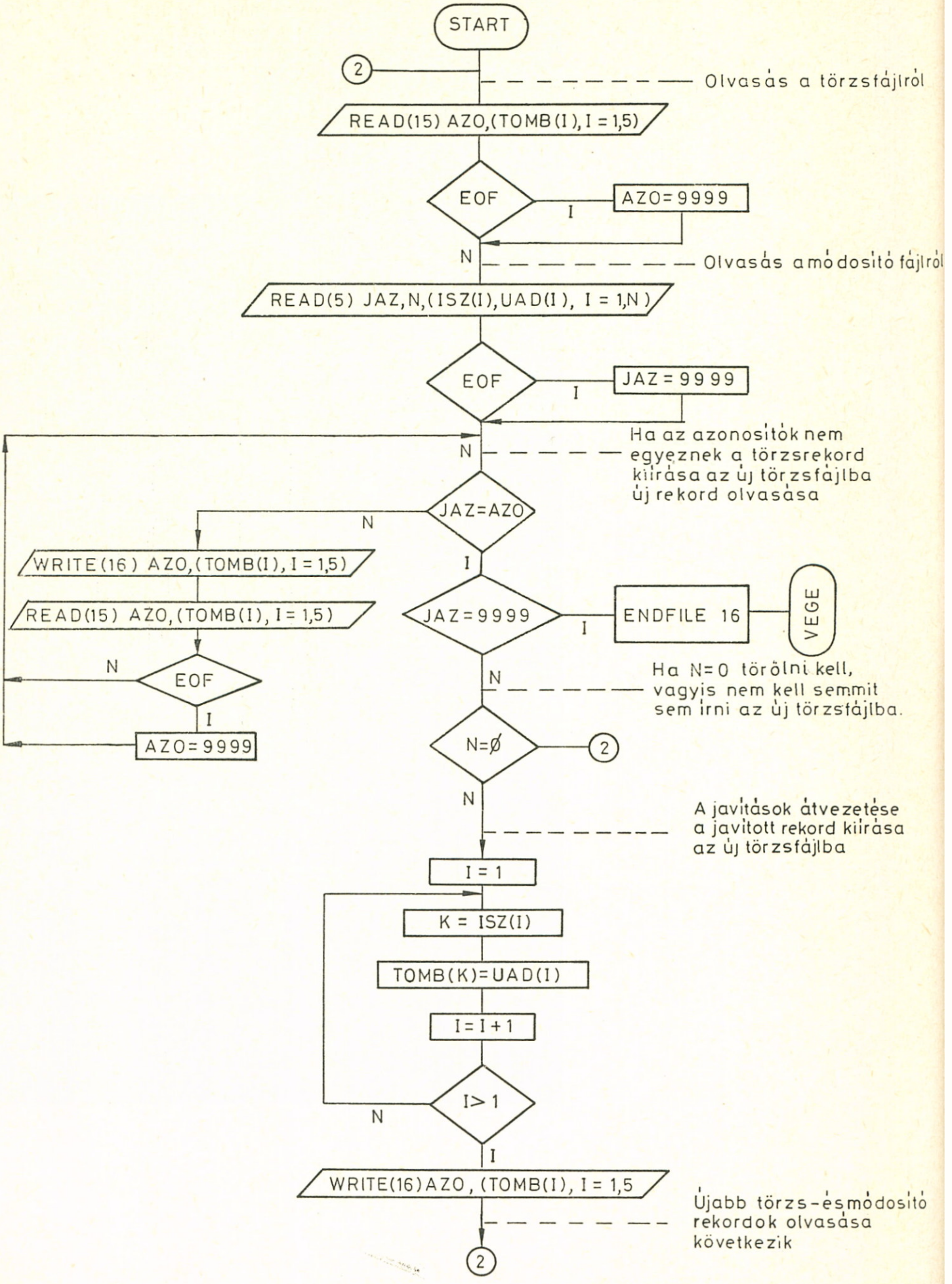

MASIS
(AZ,EV,NEM, ISK,CSA)
MOST = ...
A MOST nevű változóba be kell tölteni az éppen aktuális évet
HIBA = Ø
NEM=1 ∧ CSA=2
I
N
HIBA=HIBA-1
NEM=2 ∧ CSA=1
I
N
KOR=MOST-EV
KOR<16 ∧ NEM=1 ∧ CSA ≥ 3
I
N
HIBA=HIBA-2
KOR<14 ∧ NEM=2 ∧ CSA≥3
I
N
KOR<14 ∧ ISK=1
I
N
KOR<18 ∧ (ISK=3 ∨ ISK=5)
I
HIBA=HIBA-4
N
KOR<22 ∧ ISK=5
I
N
HIBA=Ø
N
WRITE(12) AZ, EV, NEM, ISK,CSA
Ha tudni akarjuk, hogy milyen jellegű hiba van a rekordban,ide is egy a TIPUS-hoz hasonló alprogmot kell programozni
I
RETURN

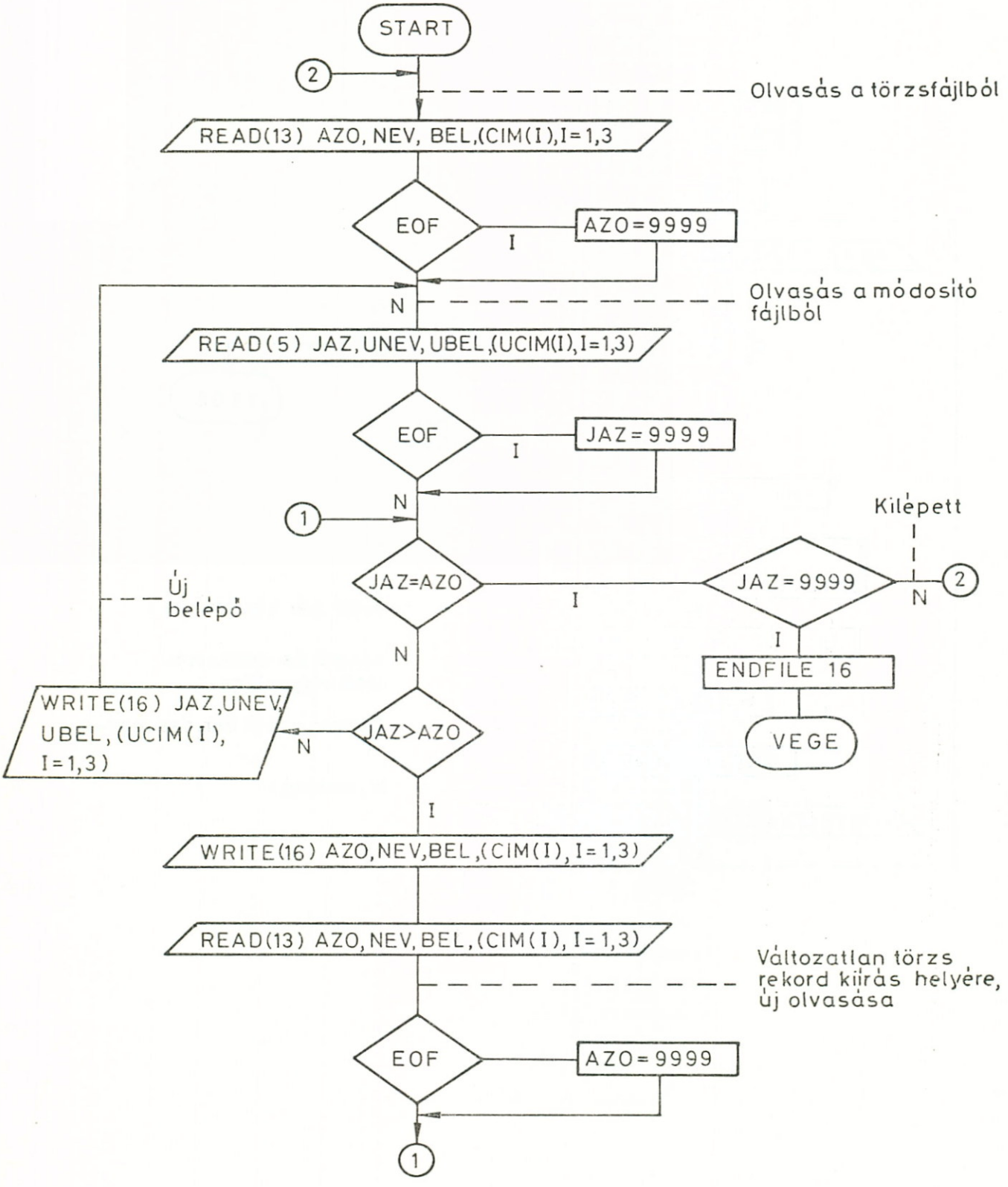

START
1
READ(20) KOD,AK,BK,EK,KS,ME,AR,KULCS
EOF
I
N
ENDFILE 10
VEGE
HIBA = Ø
KSS=AK+BK+EK
KS = KSS
N
HIBA=HIBA + 1
I
KKULCS = (AR* ME)/KSS
0,95 ≤ KKULCS / KULCS ≤ 1,05
N
HIBA=HIBA + 2
I
HIBA = Ø
WRITE(6) KOD,
AK, BK, EK, KS
ME, AR, KULCS
I
WRITE(10) KOD, AK, BK
EK, KS, ME, AR, KULCS
2

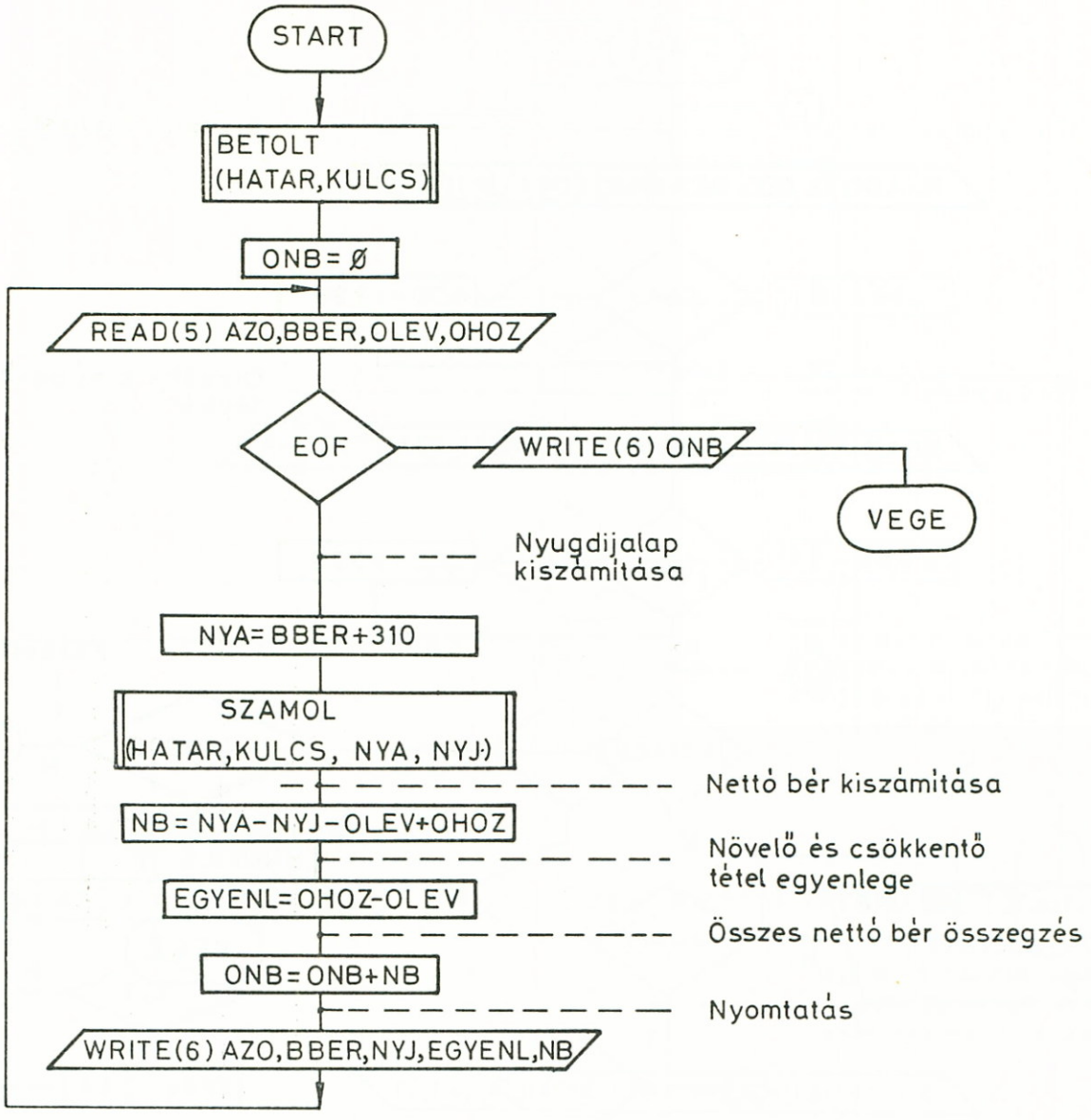

START
1
READ(20) KOD,AK,BK,EK,KS,ME,AR,KULCS
EOF
I
ENDFILE 10
VEGE
HIBA=Ø
KSS=AK+BK+EK
KS = KSS
N
HIBA=HIBA + 1
KKULCS=(AR x ME)/KSS
0,95 ≤ KKULCS/KULCS < 1,05
N
HIBA=HIBA + 2
I
HIBA = Ø
N
JELEZ (HIBA)
WRITE(6) KOD,AK,BK, EK ME,AR,KULCS
I
WRITE(10) KOD, AK, BK,EK,KS, ME, AR KULCS
1

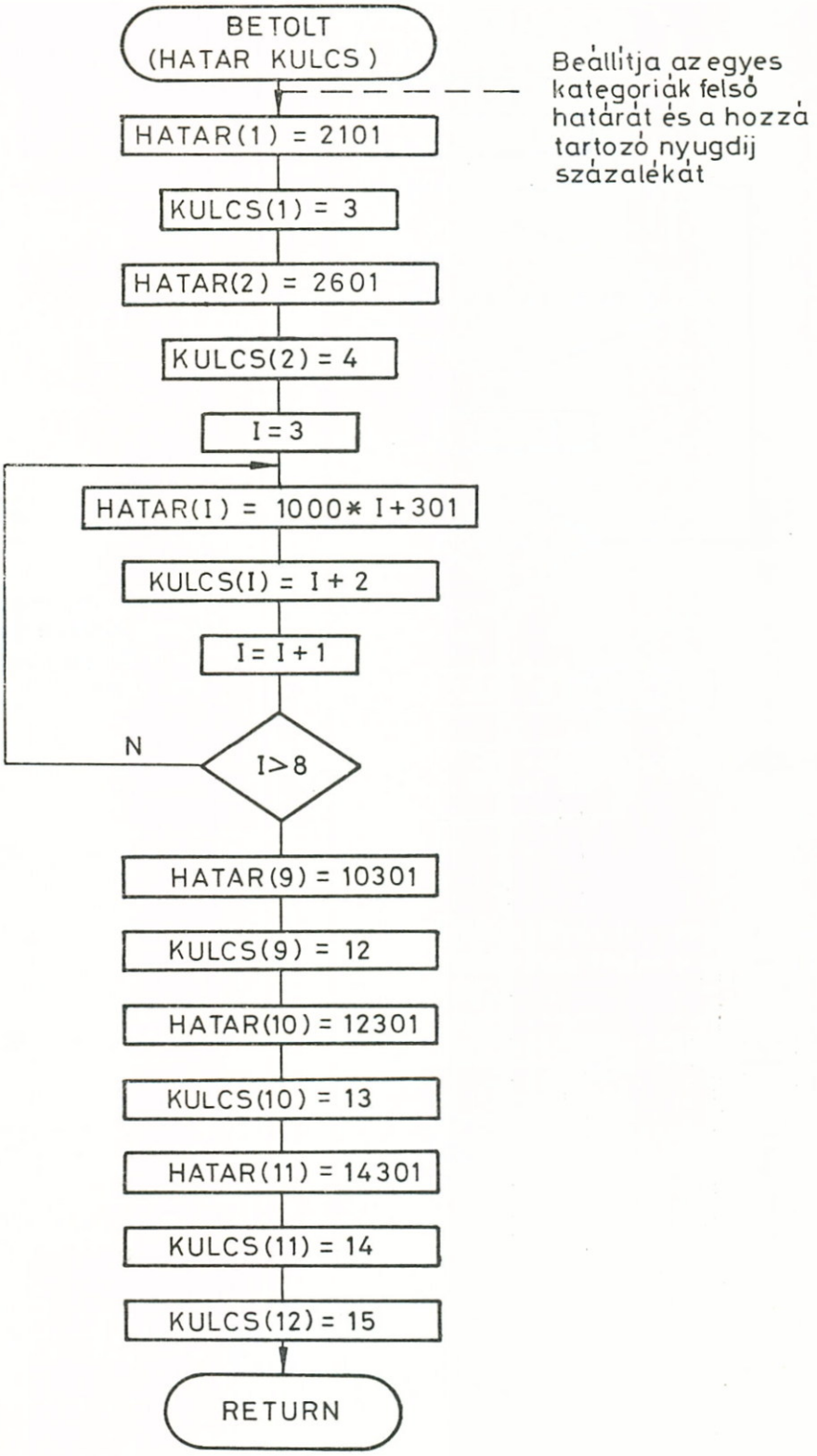


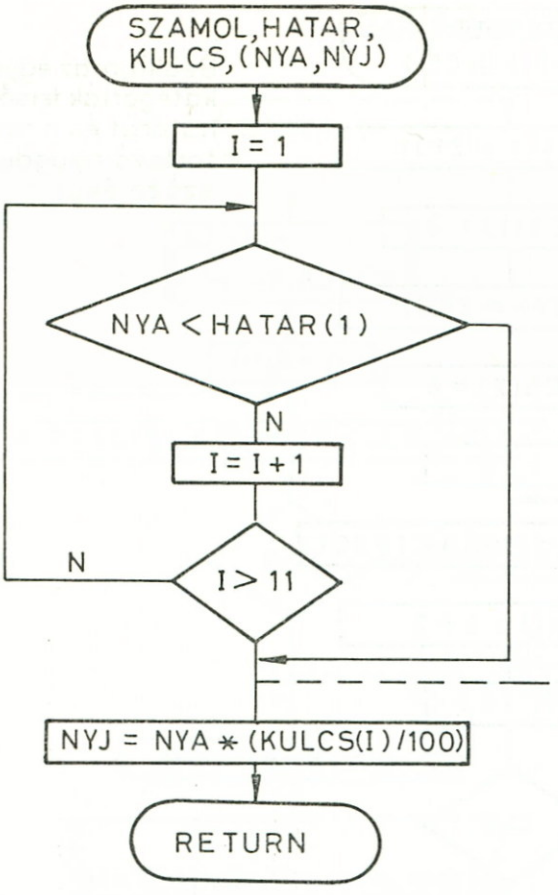
Az alprogram csak a hiba okait jelzi, magát a hibás rekordot a főprogram írja ki.


START
2
Olvasás a törzsfájlról
READ(15) AZO,(TOMB(I),I=1,5)
EOF
I
AZO=9999
N
Olvasás a módosító fájlról
READ(5) JAZ,N,(ISZ(I),UAD(I), I=1,N)
EOF
I
JAZ=9999
Ha az azonosítók nem egyeznek a törzsrekord kiírása az új törzsfájlba új rekord olvasása
N
JAZ=AZO
N
I
WRITE(16) AZO,(TOMB(I),I=1,5)
READ(15) AZO,(TOMB(I),I=1,5)
JAZ=9999
I
ENDFILE 16
VEGE
N
EOF
I
AZO=9999
N
Ha N=0 törölni kell, vagyis nem kell semmit sem írni az új törzsfájlba.
N=Ø
2
N
A javítások átvezetése a javított rekord kiírása az új törzsfájlba
I=1
K=ISZ(I)
TOMB(K)=UAD(I)
I=I+1
I>1
N
I
WRITE(16)AZO,(TOMB(I),I=1,5
Újabb törzs-ésmódosító rekordok olvasása következik
2

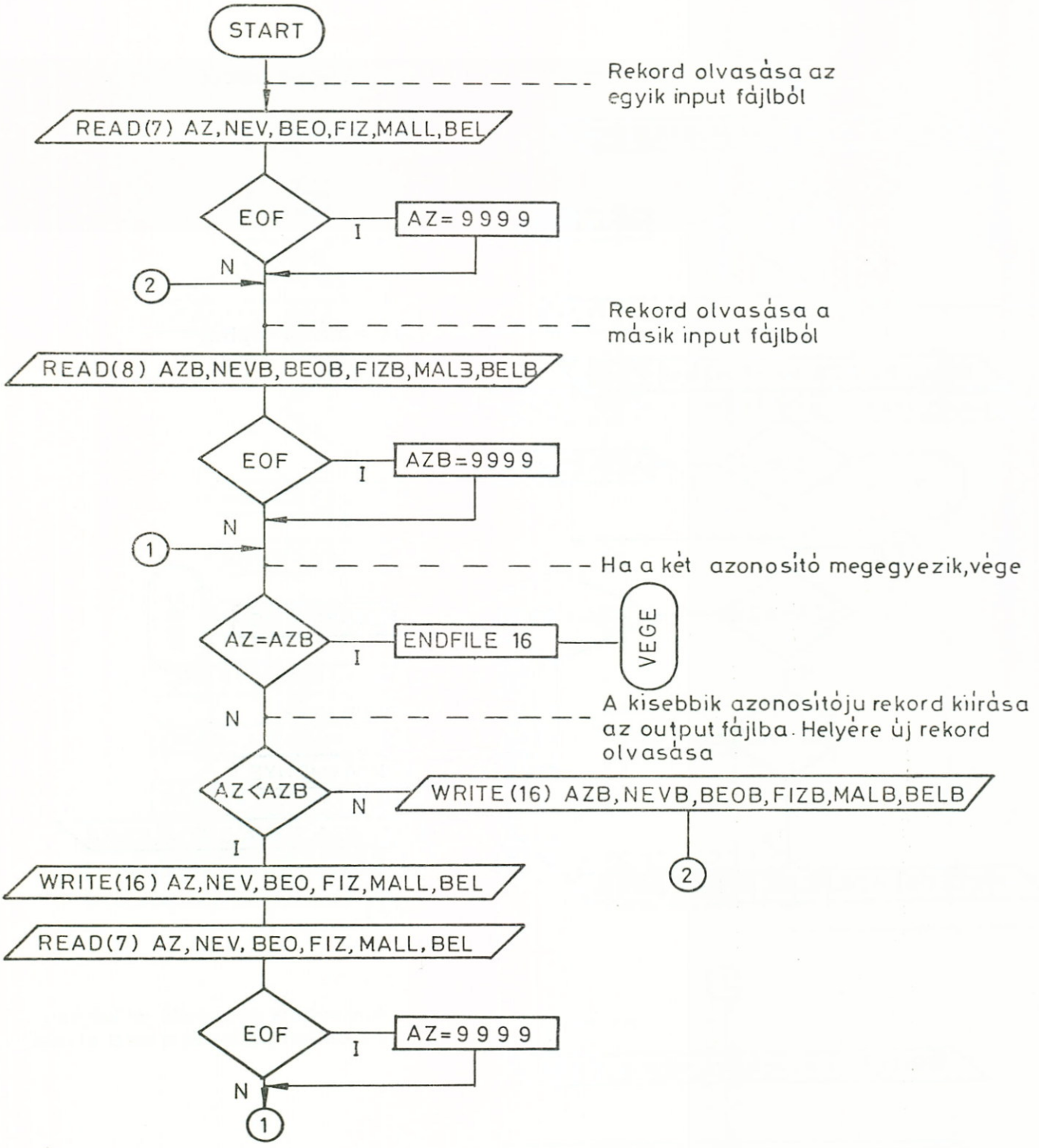

START
2
Olvasás a törzsfájlból
READ(13) AZO, NEV, BEL,(CIM(I),I=1,3
EOF
I
AZO=9999
N
Olvasás a módosító fájlból
READ(5) JAZ, UNEV, UBEL,(UCIM(I),I=1,3)
EOF
I
JAZ=9999
N
1
Kilépett
JAZ=AZO
I
JAZ=9999
N
2
Új belépő
N
I
ENDFILE 16
WRITE(16) JAZ,UNEV, UBEL,(UCIM(I), I=1,3)
N
JAZ>AZO
VEGE
I
WRITE(16) AZO,NEV,BEL,(CIM(I),I=1,3)
READ(13) AZO,NEV,BEL,(CIM(I),I=1,3)
Változatlan törzs rekord kiírás helyére, új olvasása
EOF
AZO=9999
1

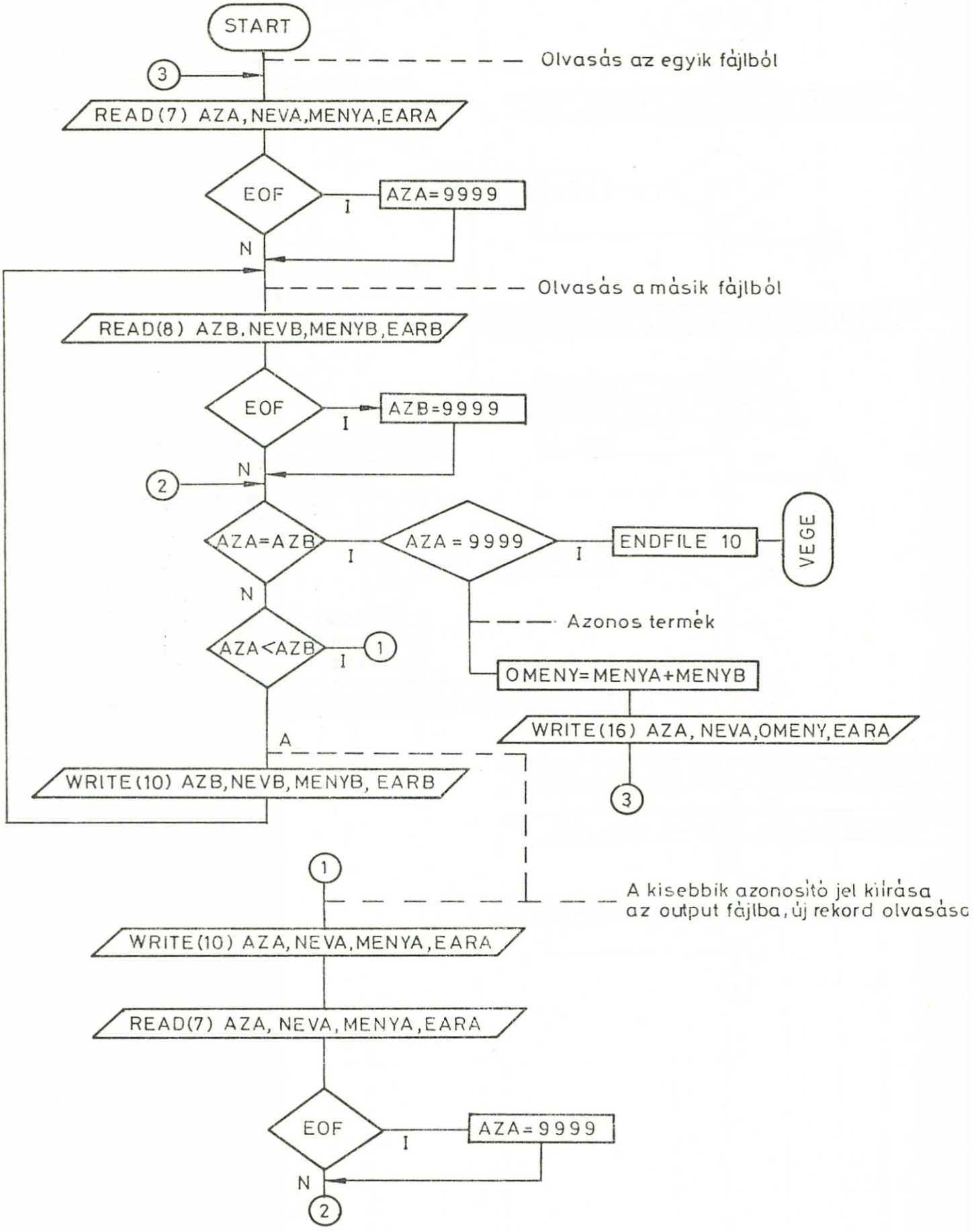

START
BETOLT
(HATAR,KULCS)
ONB=Ø
READ(5) AZO,BBER,OLEV,OHOZ
EOF
WRITE(6) ONB
VEGE
Nyugdijalap
kiszámítása
NYA=BBER+310
SZAMOL
(HATAR,KULCS, NYA, NYJ)
Nettó bér kiszámítása
NB=NYA-NYJ-OLEV+OHOZ
Növelő és csökkentő
tétel egyenlege
EGYENL=OHOZ-OLEV
Összes nettó bér összegzés
ONB=ONB+NB
Nyomtatás
WRITE(6) AZO,BBER,NYJ,EGYENL,NB

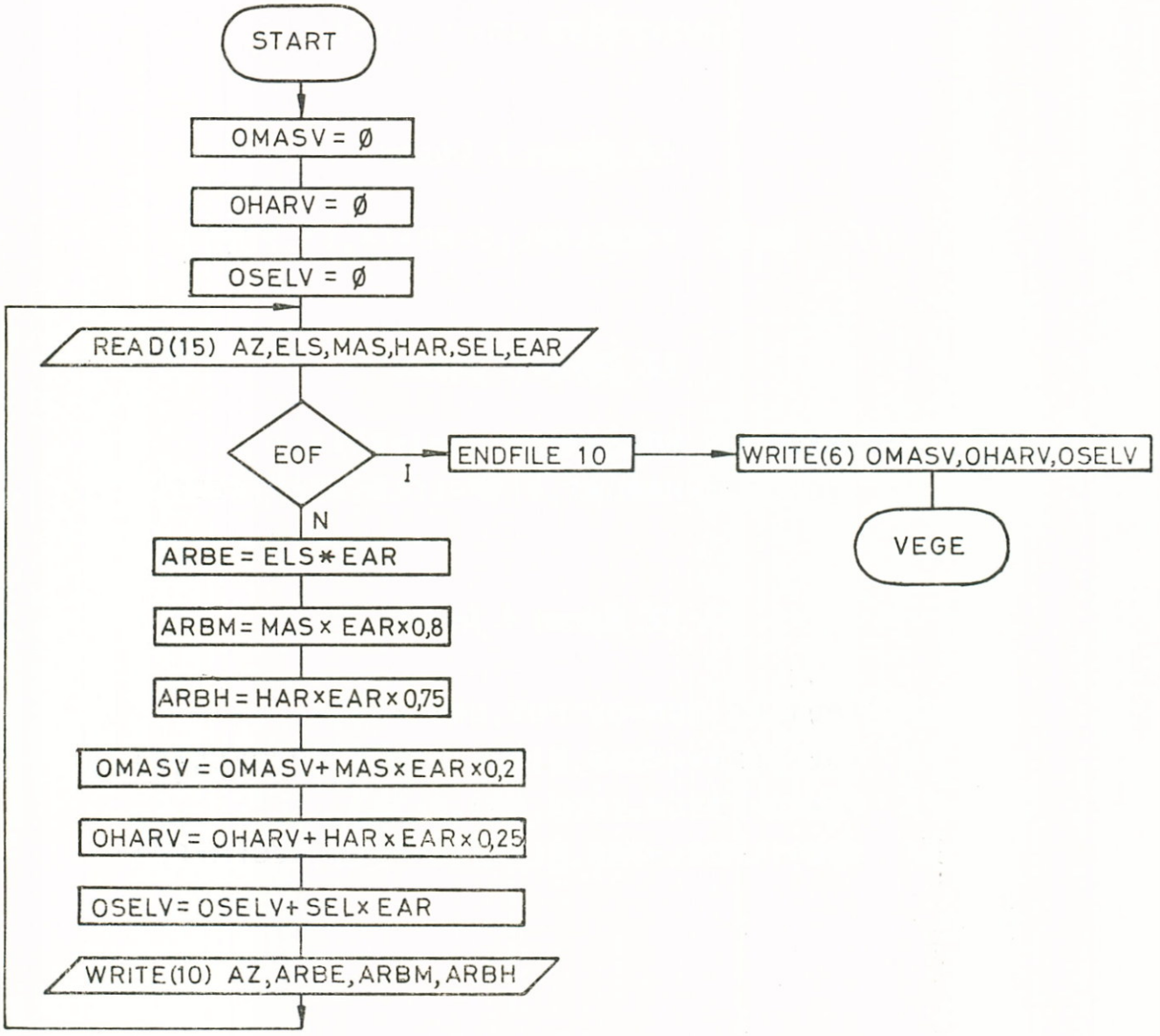

BETOLT
(HATAR KULCS)
Beállítja az egyes kategoriák felső határát és a hozzá tartozó nyugdij százalékát
HATAR(1) = 2101
KULCS(1) = 3
HATAR(2) = 2601
KULCS(2) = 4
I = 3
HATAR(I) = 1000* I+301
KULCS(I) = I + 2
I = I + 1
N
I>8
HATAR(9) = 10301
KULCS(9) = 12
HATAR(10) = 12301
KULCS(10) = 13
HATAR(11) = 14301
KULCS(11) = 14
KULCS(12) = 15
RETURN



I mutatja, hogy a KULCS tömb hányadik elemében van a nyugdíjjárulék %-os mértéke.


START
Rekord olvasása az egyik input fájlból
READ(7) AZ,NEV,BEO,FIZ,MALL,BEL
EOF
I
AZ=9999
N
2
Rekord olvasása a másik input fájlból
READ(8) AZB,NEVB,BEOB,FIZB,MALB,BELB
EOF
I
AZB=9999
N
1
Ha a két azonosító megegyezik,vége
AZ=AZB
I
ENDFILE 16
VEGE
N
A kisebbik azonosítóju rekord kiírása az output fájlba. Helyére új rekord olvasása
AZ<AZB
N
WRITE(16) AZB,NEVB,BEOB,FIZB,MALB,BELB
2
I
WRITE(16) AZ,NEV,BEO,FIZ,MALL,BEL
READ(7) AZ,NEV,BEO,FIZ,MALL,BEL
EOF
I
AZ=9999
N
1


START
3
Olvasás az egyik fájlból
READ(7) AZA,NEVA,MENYA,EARA
EOF
I
AZA=9999
N
Olvasás a másik fájlból
READ(8) AZB,NEVB,MENYB,EARB
EOF
I
AZB=9999
N
2
AZA=AZB
I
AZA = 9999
I
ENDFILE 10
VEGE
N
AZA<AZB
I
1
Azonos termék
OMENY=MENYA+MENYB
WRITE(16) AZA, NEVA,OMENY,EARA
3
A
WRITE(10) AZB,NEVB,MENYB, EARB
1
A kisebbik azonosító jel kiírása
az output fájlba, új rekord olvasása
WRITE(10) AZA,NEVA,MENYA,EARA
READ(7) AZA, NEVA, MENYA,EARA
EOF
I
AZA=9999
N
2


START
OMASV = Ø
OHARV = Ø
OSELV = Ø
READ(15) AZ,ELS,MAS,HAR,SEL,EAR
EOF
I
ENDFILE 10
WRITE(6) OMASV,OHARV,OSELV
VEGE
N
ARBE = ELS * EAR
ARBM = MAS x EAR x 0,8
ARBH = HAR x EAR x 0,75
OMASV = OMASV + MAS x EAR x 0,2
OHARV = OHARV + HAR x EAR x 0,25
OSELV = OSELV + SEL x EAR
WRITE(10) AZ,ARBE,ARBM,ARBH

IX. fejezet 1. feladat

```
//MTKFTC22 JOB 123456
```

IX. fejezet 2. feladat

```
//    EXEC  PGM=IEWL,COND=(4,LT,FORT)
```

IX. fejezet 3. feladat

```
//UNKAØØ DD DSN=VAHUR,UNIT=2314,
//   VOL=SER=KARAK2,DISP=(OLD,KEEP,KEEP)
```

IX. fejezet 4. feladat

```
//OUTPUT DD DSN=OUTPUT,UNIT=2314,
//   VOL=SER=VUKKAG,DISP=(NEW,KEEP,DELETE),
//   SPACE=(3200,(200,10),RLSE),
//   DCB=(LRECL=320,BLKSIZE=3200,RECFM=FB)
```