ERÉNYI ISTVÁN–DR. VAJDA FERENC

MIKROPROCESSZOROS RENDSZEREK FEJLESZTÉSE

2., javított kiadás

MŰSZAKI KÖNYVKIADÓ, BUDAPEST, 1983

Lektorálta:

Keresztély Sándor
okl. villamosmérnök

© Erényi István–Dr. Vajda Ferenc, 1981

ETO: 681.325.5–181.48

ISBN 963 10 3885 8 (az első kiadásé)
963 10 4855 1 (a második kiadásé)

Felelős szerkesztő: Kemény Tamásné okl. villamosmérnök

TARTALOM

Előszó 7

1. A mikroprocesszoros elemkészlet fejlődése 9
1.1. A számítástechnika és a mikroprocesszor-gyártás és -alkalmazás kölcsönhatása 12
1.2. Szabványosítás 18
1.3. Az elemek architektúrájának fejlődése 21
Irodalom 39

2. Programozható és programozott logika 40
2.1. Programozható logikai áramkörök 40
2.2. Programozott logika 52
Irodalom 60

3. A mikroprocesszoros rendszerek fejlesztése 61
3.1. Az alkalmazások csoportosítása 61
3.2. A fejlesztés menete 64
Irodalom 79

4. A mikroprocesszoros rendszerek fejlesztésének eszközei 81
4.1. Egyszerű eszközök 83
4.2. Mikroszámítógép-modulok 89
4.3. Egyszerű mikroprocesszoros fejlesztői segédeszközök 92
4.4. Miniszámítógép alapú rendszerek 95
4.5. Állapotanalizátorok 100
4.6. Nagyszámítógépes rendszerek 120
4.7. Integrált fejlesztői rendszerek 121
4.8. Programozóberendezések 139
4.9. Egyéb segédeszközök 144
Irodalom 145

5. Elemek, egységek, rendszerek ellenőrzése 149
5.1. Ellenőrzési módszerek 151
5.2. Elemek ellenőrzése 154
5.3. Önellenőrzés 164
Irodalom 171

6. Be- és kiviteli módszerek és eszközök 173
6.1. Programozott adatátvitel 174
6.2. Programozható LSI interfészelemek 186
6.3. Közvetlen tárhozzáférés 194

6.4. Perifériavezérlő LSI áramkörök 204
6.5. Perifériaprocesszorok 218
6.6. Nagytávolságú adatátvitel 227
Irodalom 231

7. Sínrendszerek 233
7.1. Síntípusok 233
7.2. A sínrendszer vezérlése 235
7.3. Kommunikációs eljárások 242
7.4. Az adatátvitel szervezése 249
7.5. Vonaladó és -vevő áramkörök 251
Irodalom 254

8. Operációs rendszerek 255
8.1. Az OS használatának általános elvei 257
8.2. OS funkciók és alkalmazások 262
Irodalom 274

9. Programfejlesztés assembler nyelven 275
9.1. A programírás előkészítése 275
9.2. Az assemblerek alapfunkciói 278
9.3. Programfelépítési módszerek 286
9.4. Makroutasítások és a feltételes fordítás 291
9.5. Áthelyezhető programok írása 298
Irodalom 300

10. A magas szintű programnyelvek alkalmazása 301
10.1. A BASIC programozási nyelv 304
10.2. A PLZ programozási nyelv 308
10.3. A PASCAL programozási nyelv 315
10.4. A szimuláció alkalmazásai 323
Irodalom 327

11. Mikroprogramozott rendszerek 329
11.1. Mikroprogramozott architektúrák 330
11.2. Mikroprogramozott rendszerek fejlesztése 338
11.3. Példa egy mikroprogramozott rendszerre 348
Irodalom 356

12. Multi-mikroprocesszoros rendszerek 359
12.1. Multiprocesszorok architektúrája 359
12.2. Funkcionális modulok csatlakoztatása 363
12.3. Operációs rendszerek 366
12.4. Párhuzamos folyamatok programozása 367
Irodalom 369
Tárgymutató 373

ELŐSZÓ

Könyvünkben célkitűzésünk, hogy a mikroprocesszoros rendszerek fejlesztésének legfontosabb eszközeit és módszereit összefoglaljuk e rendszerek hazai tervezői számára. Legfontosabbnak a tervezés módszereit és az ehhez kapcsolódó – a megvalósítás különböző fázisaiban használható – segédeszközöket tekintjük. Nem foglalkozunk a napról napra rendkívül gyorsan változó alapelemek leírásával. Ezek adatait a tervező ui. megtalálja az egyre színvonalasabb gyártói katalógusokban és alkalmazási útmutatókban. Ugyancsak nem tekintjük feladatunknak az elemek gyártásának ismertetését sem, bár a jövőben nálunk is biztosan aktív és kétirányú lesz az integrált áramkörök gyártói és alkalmazói közötti kapcsolat. Az elemek és a gyártástechnológia főbb jellemzői alapos ismerete tehát „hallgatólagosan" a tervező és így az Olvasó számára is szükséges „háttérinformáció".

A nagy bonyolultságú integrált áramkörök előállításának és alkalmazásának minden fázisában rendkívül nagy jelentőségűek a számítástechnikai módszerek és eszközök. Célunk így elsősorban az LSI és VLSI áramkörök alkalmazásához „adaptált" számítástechnikai eszközök és módszerek bemutatása. A könyv olvasásához az Olvasó általános tájékozottsága a számítástechnika valamelyik területén hasznos, de nem feltétlenül szükséges.

A konkrét alkalmazások köre és száma napról napra nő. Természetesen nem tekinthetjük feladatunknak az egyes alkalmazási területek specifikus problémáinak bemutatását, hanem helyesebbnek véltük rendszer-kategóriákban gondolkozni, és az alkalmazásoktól független általános, valamilyen formában minden rendszerben megtalálható tervezési, megvalósítási feladatoknak a lényegére, „magjára" koncentrálni.

A mikroprocesszor a „hagyományos" számítástechnika területén is jelentős változásokat hozott, létrehozva két új számítógép-ketegóriát, a mikroszámítógépet és a személyi használatú számítógépet. E kategóriák fő jellemzője, hogy általános célúak. A mikroszámítógépek esetén ugyanakkor megköveteljük, hogy a kategóriának megfelelő rendszerprogramok (operációs rendszer, assembler, magas szintű nyelvek fordítóprogramjai stb.) is rendelkezésre álljanak. Ezek a számítástechnikában kidolgozott módszereket, algoritmusokat és programokat közvetlenül felhasználják. Ezért a mikroszámítógépek alkalmazásával nem foglalkozunk részletesen.

Jelentős problémát jelentett a könyv írásakor a magyar (és sokszor a nemzetközi) terminológia kialakulatlansága is. Ennek ellenére szinte kivétel nélkül magyar (vagy „magyarosított") szakkifejezéseket használunk, de mivel ezek egy része a hazai szakirodalomban új fogalom, bevezetésükkor zárójelben megadjuk a legelterjedtebb angol nyelvű megfelelőjüket is.

Könyvünk végleges formájának kialakításához igen értékes segítséget kaptunk lektorunktól, Keresztély Sándortól, akinek ezúton is szeretnénk köszönetünket kifejezni. Nem feledkezhetünk meg közvetlen munkatársainkról és kollégáinkról sem, akikkel évek óta együtt dolgozunk a mikroprocesszorok különféle alkalmazásainak megvalósításán. Ugyancsak köszönet illeti a Magyar Tudományos Akadémia Központi Fizikai Kutató Intézete Mérés- és Számítástechnikai Kutató Intézetét, amely

lehetőséget adott a mikroprocesszoros fejlesztői rendszerek és alkalmazások terén végzett kutatómunkánkhoz. Hozzájárult a könyv tematikájának kialakításához annak a sokéves oktató munkának a tapasztalata is, amelyet a nappali és szakmérnök oktatásban és a mérnöktovábbképzésben szereztünk a Budapesti Műszaki Egyetem Villamosmérnöki Karán.

Külön köszönet illeti Kerek Margitot a kézirat gondos gépeléséért.

A szerzők

1. A MIKROPROCESSZOROS ELEMKÉSZLET FEJLŐDÉSE

A nagy bonyolultságú integrált áramkörök megjelenése előtt az adatgyűjtéssel, szabályozással, vezérléssel stb. foglalkozó szakemberek két polarizációs folyamatot is jól megfigyelhettek. Az egyik az analóg és a digitális technika egyértelmű szétválása mind az elemek, mind a tervezési módszerek területén. A másik – elsősorban a digitális építőelemek gyors fejlődésének következtében – a digitális eszközök és módszerek növekvő térhódítása. Az utóbbi folyamat különösen meggyorsult a digitális integrált áramkörök első generációjának az SSI (Small Scale Integration: kismértékben integrált áramkörök) megjelenésével és nagytömegű, egyre olcsóbb gyártásával. A napjainkban is fontos szerepet játszó digitális áramkörcsalád a TTL (Transistor-Transistor Logic: tranzisztor-tranzisztor logika) hosszú távra megfelelő alapnak látszott gazdaságos digitális berendezések tervezéséhez és gyártásához. A TTL logikai áramkörök inherens tulajdonságain túl ehhez hozzájárult az is, hogy a digitális rendszerek tervezéséhez (a kombinációs és sorrendi áramkörök tervezéséhez és alkalmazásához) jól használható módszerek álltak rendelkezésre. Ez lehetővé tette, hogy a logikai tervező néhány egyszerű szabály (késleltetések, terhelések stb.) ismeretében jól működő berendezések logikai terveit állíthassa elő, amelyek alapján könnyen bemérhető és szervizelhető rendszerek valósultak meg. A integrált áramkörök fejlődése azonban nem állt meg, és így egyre több alapáramkör volt elhelyezhető egyetlen lapkán, vagy tokban. Így kialakult az integrált áramkörök következő generációja, amelyeket MSI (Medium Scale Integration: közepesen integrált) áramköröknek neveztek. Míg az SSI áramkörök elemi kapuáramköröket (NAND, NOR, AND, OR, XOR stb. áramköröket), különböző típusú bistabil (D, J–K, R–S, T stb.), monostabil áramköröket stb. tartalmaztak, addig az MSI áramkörök ennél összetettebb funkciókat (kódoló–dekódoló, számláló, regiszter, multiplexer stb.) valósítottak meg egyetlen áramköri tokban. Ezek az áramkörök egyébként a szokásos digitális rendszerekben is megtalálhatók, így – hasonlóan az SSI áramkörökhöz – szinten szabványosított formában nagy tömegben gyárthatók voltak. Az MSI és az SSI áramkörök együttes alkalmazása tovább növelte a TTL áramkörök gazdaságosságát, csökkentette a rendszerek méretét és javította azok megbízhatóságát.

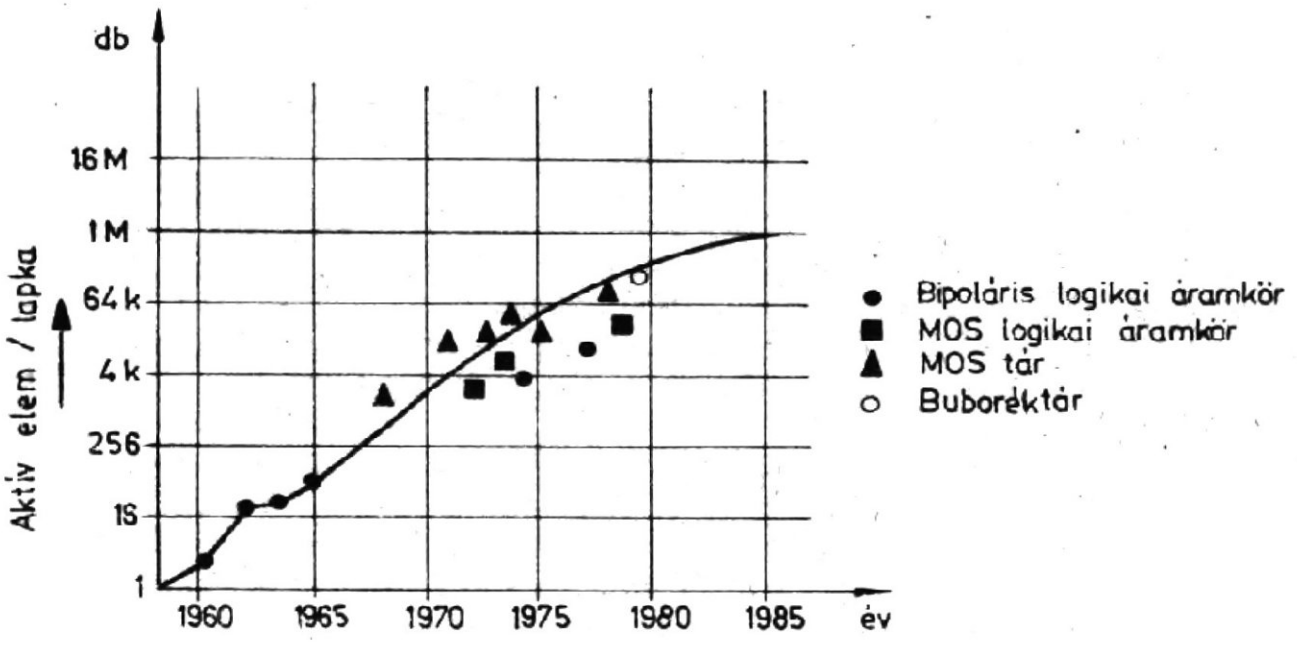
A mennyiségi növekedés a továbbiakban akadályokba ütközött és így az integrált áramköri elemek további fejlődéséhez minőségi változásokra volt szükség. A „hagyományos" eszközök és tervezési elvek alkalmazásával ui. a kombinációs és sorrendi áramkörökből álló digitális berendezéseknek olyan része volt elhelyezhető egyetlen tokban, amely már egyedi volt, ezért viszont nagytömegű gyártása csak akkor volt gazdaságos, ha sok adott berendezésre volt szükség. Ezen az alapon meg is születtek az első LSI (Large Scale Integration: nagymértékben integrált) áramkörök. Ezek azonban a felhasználó tervei alapján készültek (Custom design: rendelésre készülő tervezés), vagyis nem általánosan alkalmazhatók, hanem berendezésorientáltak voltak. Ez egyrészt olyan nagy fejlesztési költséggel járt együtt, amely csak adott darabszámú (általában minimum 10 000) azonos gyártmány esetén volt gazdaságos (eltekintve a kezdeti korszak „mindent elviselő" katonai és űrkutatási megrendelőitől). Másrészt a

technológiai nehézségek, valamint a megrendelő (rendszergyártó) és a megvalósító (alkatrészgyártó) közötti kapcsolat gyengeségei (melyeket a feladatmegfogalmazás és -megvalósítás „nyelvének" alapvető különbözősége okozott) miatt az átfutási idő nagyon hosszú – minimum 1 év – volt. Ebben az időszakban (kb. a 60-as évek második felében) az LSI tömeggyártás szempontjából nagyon jó tulajdonságú új félvezető elem: a FET-et (Field Effect Transistor: térvezérlésű tranzisztor) alkalmazó MOS (Metal Oxide Semiconductor: fémoxid félvezető) is elérte a nagy bonyolultságú áramkörök szintjét. (Ennek először a P-csatornás változata jelent meg, amit néhány év múlva követett az NMOS, majd a komplementer, azaz a CMOS.)

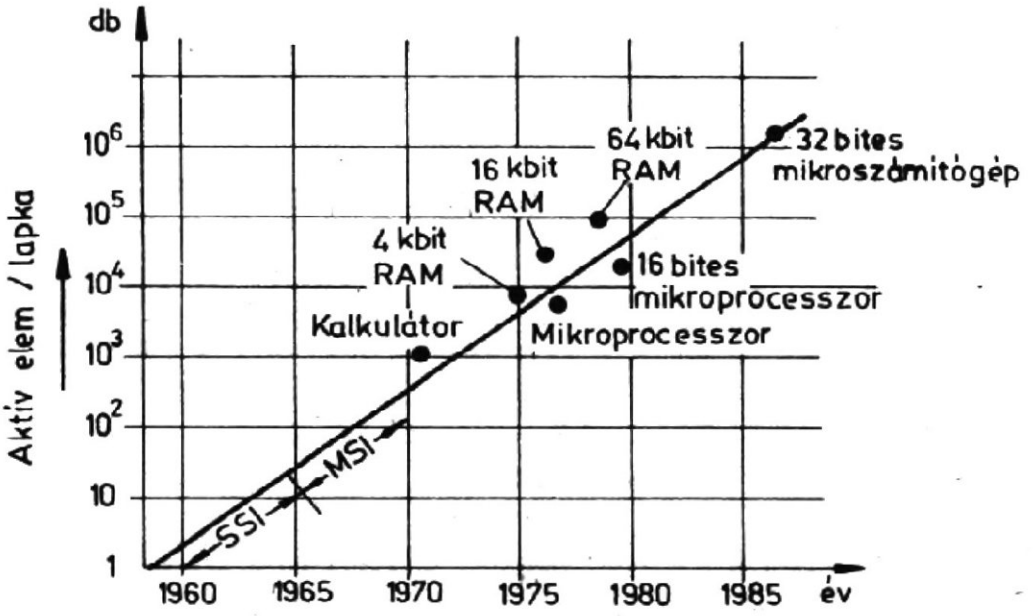
Ha az integrált áramkörök komplexitásának (elemi áramkör/lapka) a változását az elmúlt évek adatai alapján megvizsgáljuk, exponenciális növekedést találunk. A néhány éve Gordon Moore (Intel Corporation) által megfogalmazott és azóta „Moore-törvény"-nek nevezett alapelv szerint a tokonkénti elemszám évenként átlagosan megkettőződik.

Az 1.1. ábra görbéje [1] 1965 és 1969 között „szakadást" mutat, ekkor ui. a „klasszikus" tervezési eljárásokkal gazdaságosan megvalósítható nagy bonyolultságú LSI elemek még nem léteztek. A 60-as évek végén megjelentek először a különféle félvezető tárolóelemek, majd a kalkulátor-áramkörök. Ezt követően a 70-es évek elején megjelent egy – a digitális számítógép központi egységének egyszerűsített változatához hasonló felépítésű – „felhasználói" LSI áramkörből származó elem, az első mikroprocesszor (Intel 4004), amelyet azóta rendkívül nagyszámú és egyre összetettebb felépítésű társuk követett a piacon. A mennyiségi fejlődés lehetővé tette, hogy egy új kategória: a VLSI (Very Large Scale Integration: nagyon nagy mértékű integrálás) megjelenésével teljes számítógép „magot" (központi egység, tár, ki-, ill. bemeneti kapcsolat stb.) is egyetlen lapkán valósítsanak meg. Másrészt pedig az egyre fejlettebb tulajdonságú mikroprocesszorok megjelenésével lényegében eltűnt a határ a miniszámítógépek és a mikroprocesszorokat alkalmazó mikroszámítógépek között.

Tudatosan nem foglalunk állást az MSI–LSI, ill. az LSI–VLSI határok „számértékére" vonatkozóan, azt mégis érdemes az 1.2. ábra [2] alapján megemlítenünk, hogy 1985-re az 1 millió elemi áramkör/tok komplexitást a sorozatgyártás szintjén el lehet majd érni.

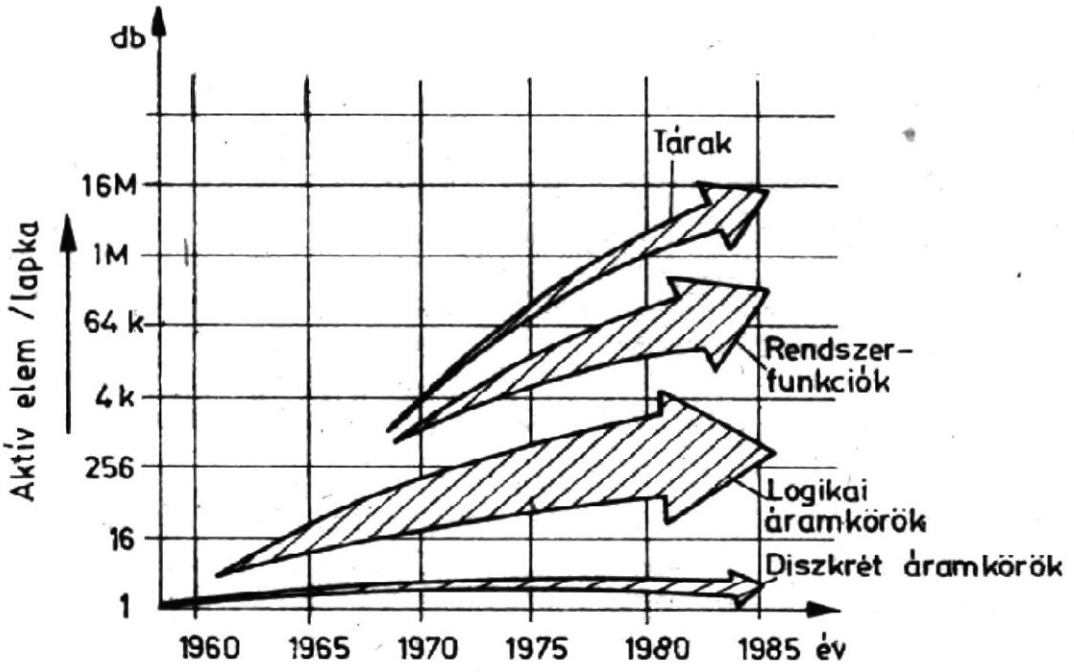


1.1. ÁBRA. Integrált áramkörök komplexitásának növekedése [1]



1.2. ÁBRA. Integrált áramkörök mennyiségi növekedése és a segítségükkel egyetlen tok formájában megvalósítható funkcionális egységek [2]

A mikroprocesszor és a kapcsolódó kiegészítő áramkörök megjelenésével két új kategória is megszületett: a mikroszámítógép és a programozott logika. Előbbi a kis- és miniszámítógépek alkalmazásával már eddig szerzett széles körű tapasztalatok még sokkal nagyobb területeken való hasznosíthatóságát tette lehetővé (alapvetően alacsonyabb árával és a „beépíthető" méretével). A programozott logika pedig a digitális rendszerek nagy sorozatban gyártható elemek alapján való, LSI alapú megvalósítását tette lehetővé, feloldva ezzel a felhasználói LSI problémáiból származó nehézségeket. Ezek a problémák azonban most már a VLSI talaján újratermelődtek. Itt a legfőbb korlátot a rendkívül magas tervezési költségek mellett a megfelelő összetett, nagy sorozatban gyártható elemek „kitalálása", azaz definiálása jelenti [1], [3]. Ebben a korszakban az elem- és rendszergyártók sajátos formában közelednek egymáshoz. Egyrészt a nagy félvezetőgyárak (Texas, National, Fairchild, Intel stb.) saját nagy bonyolultságú



1.3. ÁBRA. A legfontosabb báziskategóriák várható fejlődése [1]

integrált áramköreik felhasználásával (azok „támogatására") rendszereket is kezdtek gyártani, másrészt a legnagyobb „rendszercégek" (IBM, HP, DEC stb.) először kutatás-fejlesztési célokra, majd sorozatgyártásra képes „házon belüli" félvezetőgyártó üzemeket hoztak létre. Ezek száma az Egyesült Államokban 1975 és 1979 között 19-ről 43-ra emelkedett [1]. A jövőben a CAD (Computer Aided Design: számítógéppel segített tervezés) kiterjedt alkalmazásával és egy új technológiai eljárással, a maszk helyett elektronsugárral, a szilíciumszeletre való közvetlen „írás" segítségével, kis darabszámú, (egyedi) nagy bonyolultságú áramkörök is valószínű, hogy megvalósíthatók lesznek.

Természetesen a VLSI bonyolultságú áramkörök mellett kisebb számú aktív komponenst tartalmazó elemek is megmaradnak és fontos szerepet fognak játszani. Ennek megfelelően a legfontosabb elemkészlet-kategóriák várható fejlődését az 1.3. ábrán [1] mutatjuk be.

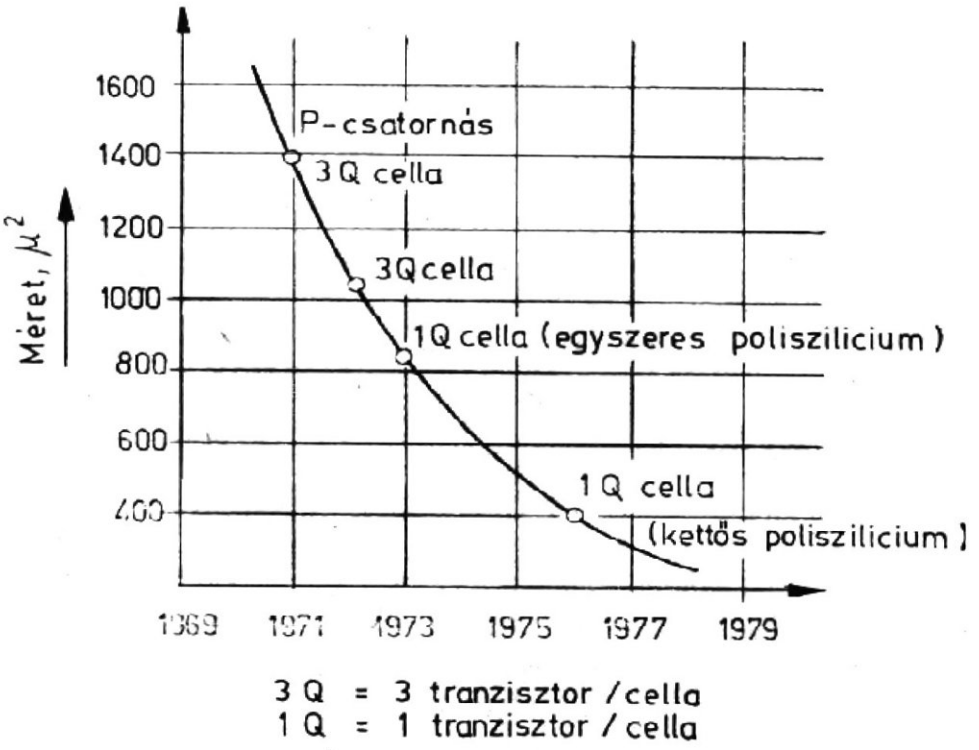
A MOS integrált áramkörök előbbiekben vázolt mennyiségi fejlődése az előállítás egész folyamatának alapvető minőségi fejlődése talaján jött létre. Ez az integráltság fokának lehetséges növelésén túlmenően számos fontos jellemző tulajdonság nagymértékű javulásával is együtt járt. Az alapvető MOS eljárások további tökéletesítése (pl. HMOS) mellett egyre újabbak jelennek meg, mint pl. az SBMOS (Shottky Barrier MOS: Shottky-diódás MOS) vagy a DIMOS (Dielectric Isolation MOS: dielektromos szigetelésű MOS). Az integrált áramkörök elektromos paraméterei az új eljárások segítségével megjavultak. Az 1.1. táblázatban [3] az elemi kapuáramkörök késleltetését és az ehhez tartozó teljesítménydisszipációt, ill. a sebesség–teljesítmény szorzat jellemző értékeit mutatjuk be.

1.1. TÁBLÁZAT. Integrált áramkörök jellemzőinek összehasonlítása

Paraméter	Év	1976	1980	1985
	Gyártási eljárás	NMOS	NMOS II	SBMOS DIMOS
Sebesség (belsőkapu), ns		6	2	1
Teljesítménydisszipáció, mW/kapu		3	0,5	0,25
Sebesség–teljesítmény szorzat, pJ		18	1,0	0,25
Tokonkénti teljesítmény-disszipáció, W		1	1,5	5

A gyártás fő jellemzőit az 1.2. táblázatban foglaltuk össze [3]. A fokozott komplexitást a geometriai felbontás javulása és a lapkaméret növelése tette lehetővé. Az elemi áramkörök méretcsökkenését jól szemlélteti az 1.4. ábra is, ahol a RAM-cella méretcsökkenését tüntettük fel az elmúlt néhány évben.

A gyártás gazdaságosságát, vagyis végső soron a gyárthatóságot és az árat meghatározó tényező a kihozatal, vagyis az összes legyártott elemekből a jók aránya.



1.4. ÁBRA. A RAM-cella méretcsökkenése a technológia fejlődésével [2]

1.2. TÁBLÁZAT. Integrált áramkörök alapvető gyártási paramétereinek összehasonlítása

Paraméter	Év		
	1976	1979	1985
Lapkaméret (Sorozatgyártási szint), mm²	4,0×4,0	6,35×6,35	11,4×11,4
Felbontás (sorozatgyártási szint) µm	6	3	1
Lapkánkénti áramkörszám (nem tárolóelem esetén)	10 000	80 000	>1 000 000
Szeletméret, mm	75	100	125

A szeletenkénti jó végtermékek számát közelítőleg a következő módon számíthatjuk [2]:

$$GFP = \frac{GD}{WF} \cdot Wy \cdot Ay \cdot FTy$$

ahol $\frac{GD}{WF}$ a jó lapkák száma/szelet; Wy a szelet kihozatal; Ay az összeszerelés kihozatala; FTy a végellenőrzés kihozatala.

1976-ban pl. átlagos $4{,}0 \times 4{,}0$ mm² lapkaméret és 75 mm szeletátmérő mellett a szeletenkénti jó lapkák száma 200 volt. Figyelembe véve a mikroprocesszorokra jellemző kihozatal értékeket (Wy = 0,4, Ay = 0,8 és FTy = 0,85) ez szeletenként 54 egységet jelentett. Vagyis pl. 10 000 szelettel indulva 540 000 mikroprocesszort lehetett gyártani.

1979-ben azonos méretű termékek esetén a szeletméret-növekedés (100 mm) és a kihozatalok változása (0,5, 0,9 és 0,9) következtében a lapkaszám 145-re javult, vagyis lényegesebb gyártókapacitás-növelés nélkül az 1976-ban gyártott mennyiségnek majdnem a háromszorosa (vagyis 1 450 000 mikroprocesszor) gyártható. Még a komplexebb (10 000 helyett nyolcszor annyi, vagyis 80 000 elemi áramkört tartalmazó, $6{,}35 \times 6{,}35$ mm² méretű) mikroprocesszor esetén is a jó lapkák száma 58.

A nagy bonyolultságú áramkörök megbízhatóságának a javulása is fontos feltétele volt a komplexitás növelésének. Ha valaki hagyományos TTL tokokból megépítene egy korszerű mikroprocesszor bonyolultságú rendszert, csak néhány hónap nagyságrendű MTBF (Mean Time Between Failure: meghibásodások közötti átlagos idő) értéket kapna. Ezzel szemben a 3000 kapuáramkörös bonyolultságú MC 6800 mikroprocesszor meghibásodási aránya (meghibásodási valószínűség 1000 órára számolva), ill. meghibásodások közötti átlagos ideje lényegesen jobb (1.3. táblázat) [2]. A meghibásodási arány aszimptotikusan tart a 10^{-9}/1000 óra értékhez [2], vagyis az új, nagyobb bonyolultságú VLSI áramkörök esetében sem várhatunk ennél jobb értéket. Egy kapuáramkörre számított meghibásodási arány millió elemi áramkör, vagyis kb. 300 000 kapuáramkör egyenérték esetén is $3 \cdot 10^{-15}$/1000 óra!

1.3. TÁBLÁZAT. Az MC 6800 mikroprocesszor hibaaránya és a meghibásodások közötti átlagos ideje (kerámiatokozás, 70 °C környezeti hőmérséklet esetén)

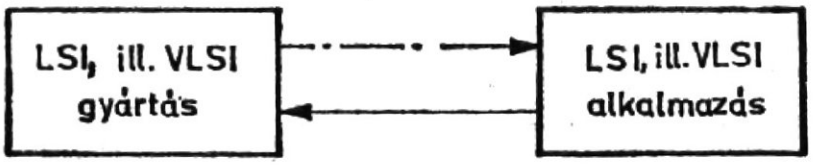
Év	Meghibásodási arány %/1000 óra	MTBF Meghibásodások közötti átlagos idő	
		óra	év
1974	1,27	78 000	9
1975	0,5	200 000	23
1976	0,12	833 000	95
1977	0,08	1 700 000	194
1978	0,013	7 700 000	877
1979	0,006	16 666 666	1901

1.1. A SZÁMÍTÁSTECHNIKA ÉS A MIKROPROCESSZORGYÁRTÁS ÉS -ALKALMAZÁS KÖLCSÖNHATÁSA

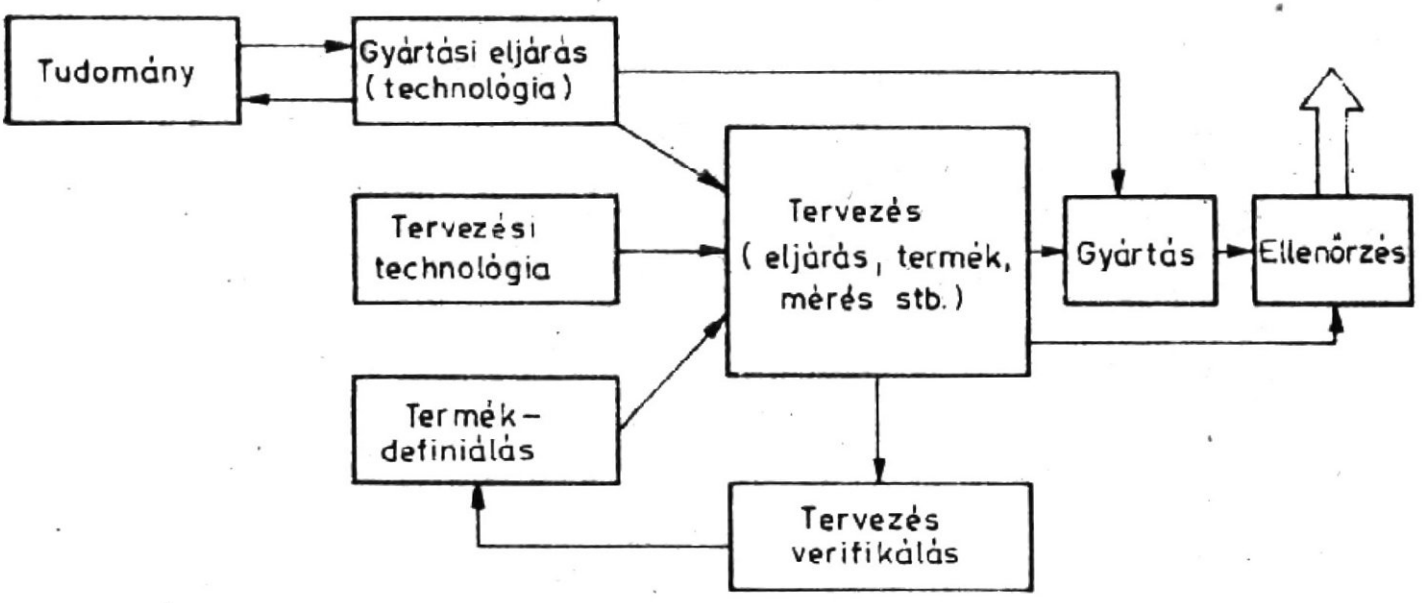
A nagy bonyolultságú integrált áramköröket az 1.5. ábra szerinti két elkülönült területen gyártják és alkalmazzák. A két terület között két nyíl jelzi a kapcsolatot. Az egyik kapcsolat esetén a gyártótól az alkalmazóhoz jutnak el a gyártott elemek. A másik, ellenkező irányú kapcsolat is rendkívül fontos, amely lehet közvetlen vagy közvetett is és elsősorban a gyártó (alkatrészgyártó) termékeinek kialakításában van jelentősége. Mint már láttuk az előzőkben, éppen ez a „visszahatás" az, amire egyre jellemzőbb az ennek „házon belüli" megvalósítására való törekvés. A számítástechnika mindkét területen, a gyártásban és alkalmazásban egyaránt fontos szerepet játszik. Túlzás nélkül állíthatjuk, hogy a modern számítástechnika eredményeinek közvetlen és közvetett alkalmazása nélkül korszerű nagy bonyolultságú integrált áramkörgyártásról nem beszélhetnénk. Ugyanez igaz a mikroprocesszor alkalmazására vonatkozóan is, ha a felhasználás alapmódszereire, ill. a módszerekhez kapcsolódó eszközökre gondolunk. A „visszahatás" itt többrétű. Egyrészt a korszerű számítógépek is elképzelhetetlenek a korszerű elemkészlet nélkül, másrészt a mikroprocesszor alapú rendszereknek is fontos a szerepe mind a gyártó rendszerekben, mind az alkalmazások létrehozásához szükséges „segédeszközökben".

Az integrált áramkörök és különösen ezek LSI és a VLSI bonyolultságú generációinak gyártása nagyon összetett. A gyártási technológiákhoz is nagyszámú, de önmagukban is komplex részfeladat megvalósítására van szükség. A gyártási technológia azonban csupán egyik összetevője az LSI és a VLSI gyártás folyamatának.

Az 1.6. ábrán tüntettük fel vázlatosan ezt a folyamatot egy-egy blokkal jelölve a fő tevékenységi területeket. Ezek mindegyikéhez ismét számos, önmagában is összetett lépés tartozik. A tudományos kutatásnak elsősorban a gyártási eljárásban (technológia) mutatkoznak meg közvetlenül az eredményei, a kölcsönhatás itt a legkézenfekvőbb.



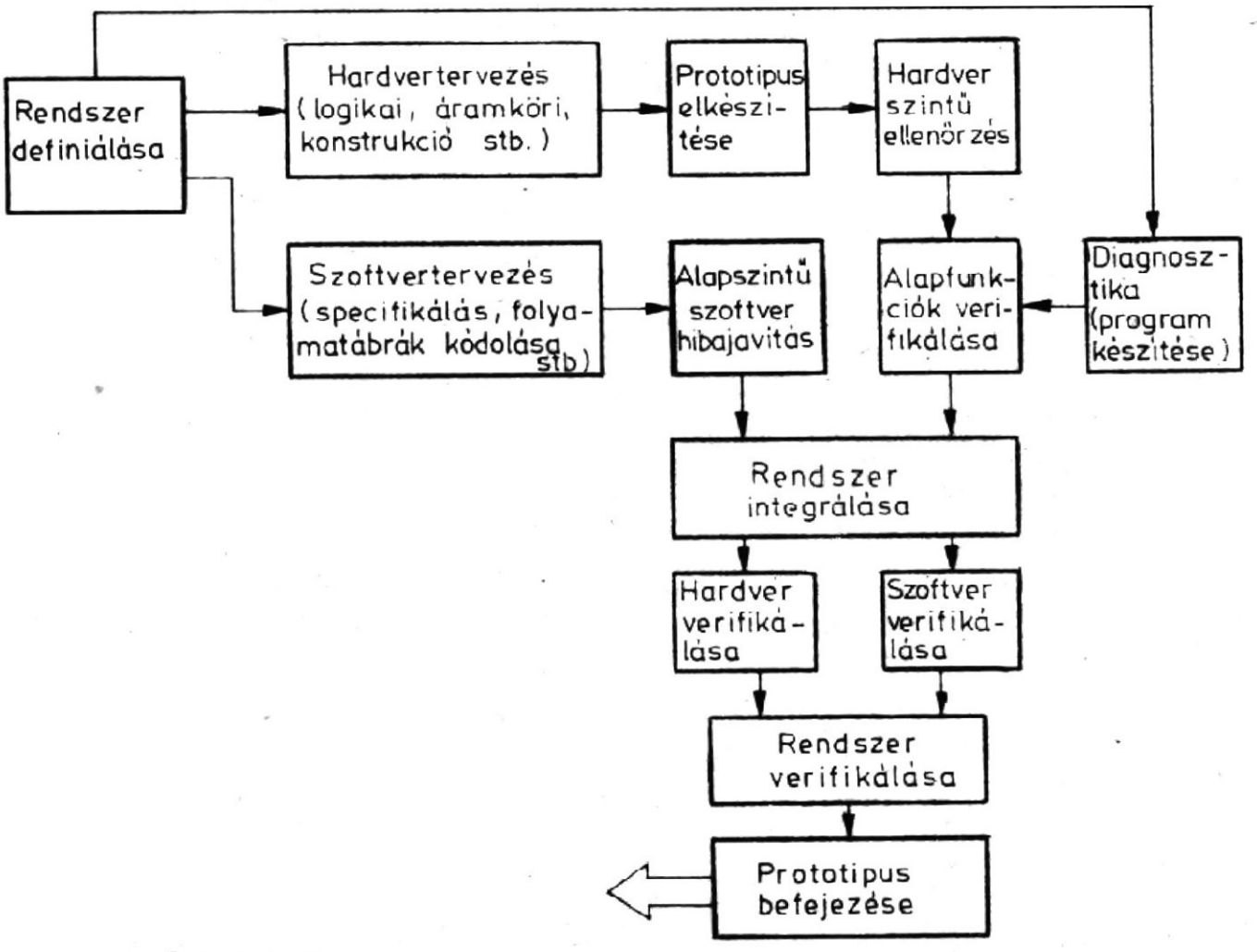
1.5. ÁBRA. LSI és VLSI gyártás és alkalmazás kölcsönhatása



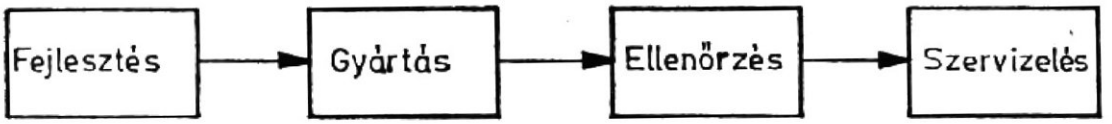
1.6. ÁBRA. Nagy bonyolultságú integrált áramkörök gyártásának folyamata

A számítástechnikai módszerek és eszközök kiterjedt alkalmazására azonban szükség van a folyamat szinte valamennyi fázisában. A teljesség igénye nélkül ezt inkább csak néhány jellemző példával illusztráljuk. A „legnagyobb" blokk a tervezést jelképezi. Tervezésbe ui. nem csupán a gyártási eljárás egyes lépéseinek a megvalósítása és sorrendje, az elhelyezés szabályai és az elemi áramkörök elektromos jellemzőinek leírása tartozik, hanem az adott LSI és VLSI termék megtervezése is, amely ismét sok részfeladatot jelent (logikai, áramköri tervezés, maszkelrendezések megtervezése stb.). Ide tartozik még a tervezés verifikálása is, amely közvetlenül visszahat a termék definiálására. Az LSI és a VLSI esetén a másik rendkívül igényes (és nagy költségkihatású) terület a mérés, ellenőrzés (teszt), amit meg kell tervezni. A tervezési technológia blokk jelképezi ábránkon azt, hogy minden egyes tervezési területhez külön tervezési eljárás kidolgozására is szükség van.

A számítástechnikának nagyon fontos feladata van a tervezés majdnem minden egyes részfeladatának megoldása során. A logikai és áramköri tervezés, a maszkok megtervezése és elkészítése elképzelhetetlen a számítógéppel segített tervezés módszerei és eszközei nélkül. A számítógépes modellezés és szimuláció is számos feladat megoldásának eszköze (verifikálás, funkcionális ellenőrzés stb.). A termék-definiálás során a struktúra alapját nagyon sok esetben számítógépegységek és részegységek adják. Mind a gyártás közbeni (szelet és lapka szintű) mérés, mind a tokozott termékek végellenőrzése és válogatása gazdaságosan csak számítógép alapú eszközökkel végezhető. A gyártási folyamat is nagymértékben automatizált, számítógéppel vezérelt, irányított, ellenőrzött, hiszen csak ezen a módon valósítható meg a szilíciumrudak húzása, előírt hőmérsékletek nagy pontosságú biztosítása, az ionimplantáció, a szerelés, a tokozás stb.



1.7. ÁBRA. Mikroprocesszoros rendszer prototípusa elkészítésének folyamata



1.8. ÁBRA. A prototípus útjának legfontosabb állomásai

Nézzük meg ezek után az alkalmazás ugyancsak összetett folyamatát. Elsősorban a mikroprocesszor alapú rendszerek megvalósítására vonatkozik az 1.7. ábra, amelyen egy ilyen rendszer prototípusa elkészítésének egyszerűsített folyamatábrája látható. Az ábra blokkjainak szerepéről és értelmezéséről a későbbi fejezetekben még részletesen lesz szó, most csak elsősorban a számítástechnikával való kölcsönhatás területeit kívánjuk vele illusztrálni.

A rendszer-definiálás, ill. a hardvertervezés kiindulási alapja minden esetben a digitális számítógépek architektúrája. Az elemkészlet fejlődése következtében lehetőség van a számítástechnika területén elméletileg megalapozott, kidolgozott és megvalósított összetett rendszerek eredményeinek adaptálására, amit a többprocesszoros, mikroprogramozott és párhuzamos feldolgozórendszerek stb. képviselnek. A szoftver „készítés" minden fázisában az alapot a számítástechnika által kimunkált eredmények adják a számítógép bázisú eszközök és segédeszközök (szerkesztő-, fordító-, szimuláló-, hibakereső- stb. programok) formájában. Ugyancsak számítástechnikai módszereket és eszközöket alkalmazunk a diagnosztika, a szoftver- és a rendszerverifikálás során, a prototípus elemeinek előállításakor (pl. tár és logikai elemek programozása stb.).

Az 1.8. ábrán a prototípus útjának legfontosabb fázisait (sorozatgyártás, ellenőrzés, szervizelés) tüntettük fel. Nem kívánunk itt külön kitérni ezek általános számítástechnikai vonatkozásaira, mint a gyártásautomatizálás és folyamatszabályozás, számítógépes mérő- és ellenőrző-rendszerek, számítógépes tervezés stb., vagy az adatbázis rendszerek alkalmazása a vállalati irányításban, gyártáselőkészítésben, dokumentációban stb.

Csupán olyan új – a termékek programozhatóságából és ehhez kapcsolódó erőforrásokból származó – lehetőségekre utalunk, mint a mikrodiagnosztika a hibák elhárításában vagy az „átprogramozásból" adódó flexibilitás. Ezekről a problémákról a későbbiekben még szó lesz.

A mikroprocesszor alapú rendszerek és számítástechnika kapcsolatát – ha nem is azonos súllyal –, de mindenképpen kölcsönhatásnak tekinthetjük. Egyrészt a korszerű LSI és VLSI elemek a számítástechnikai berendezések természetes elembázisát jelentik, és ezek lehetőségei, tulajdonságai visszahatnak a rendszerek felépítésére, tulajdonságaira is. Másrészt egy kis túlzással úgy is fogalmazhatunk, hogy a mikroszámítógépek tulajdonképpen a számítástechnikai berendezések hagyományos gyártóihoz képest „kívülről" kerültek be a számítástechnikai iparba, kiterjesztve az alkalmazások körét és megváltoztatva számos kialakult konvenciót. (Mini- és kisszámítógépkategóriák és mikroszámítógép közötti határ eltűnése, hagyományos architektúrák megváltozása stb.)

Ugyancsak így értelmezhető az intelligens vezérlő – intelligens periféria (pl. nagy mágneses perifériák, nyomtatók, rajzolók, terminálok, kommunikációs eszközök stb.) kapcsolat is, vagy a (dedikált, ill. azonos funkciójú elemeket tartalmazó) multimikroprocesszorok univerzális számítástechnikai rendszerként való alkalmazása.

1.2. SZABVÁNYOSÍTÁS

A szabványosítás fogalmát valószínűleg senkinek sem kell definiálni, előnyeit, szükségességét nem kell külön bizonyítani. A társadalmak egész életét egyre több területen szabványok nélkül ma már el se lehetne képzelni. Ugyanakkor annyira természetesekké is váltak, hogy szinte észre se vesszük őket. Gondoljunk csak kiragadott példaként az egyszerű ízzólámpától (feszültség, foglalatméret) kezdve olyan területekre, mint a hírközlés (telefon, távíró, rádió, televízió stb.) vagy közlekedés (autó, vasút, repülés, hajózás), de űrhajózás is, gondolva a közös szovjet–amerikai kísérletekre. A szabványosítás tehát a társadalmi élet ma már szükséges velejárója. Kis túlzással a társadalom technikai fejlettségének „mérőszámaként" is tekinthetnénk a szabványosított dolgok számát.

Vizsgáljuk meg a mikroprocesszorok (itt persze ezek egész „környezetét" értve) gyártásának és alkalmazásának helyzetét a szabványosítás foka szempontjából. Gondoljunk itt a két alapvető „forrásra", amelyből a mikroprocesszorok „széles folyama" táplálkozik: a félvezető integrált áramkörökre és a számítástechnikára. Mindkettő a szabványosítás elég fejlett fokán volt, amikor a mikroprocesszor létrejött, így ezek közül számosat természetes örökségként, szinte automatikusan megkapott.

Az integrált áramkörök oldaláról a teljesség igénye nélkül illusztrációként felsorolunk néhány szabványosított, kvázi-szabványosított, vagy majdnem szabványosított területet. A logikai szintek a TTL logikai család szerintiek, tápfeszültség: +5 V (±10%), a kimenet áramköri felépítése is egységesített (mindkét irányban kis impedanciájú meghajtást biztosító ún. totem-pole: totemoszlop, az egymás „tetején" levő tranzisztorból álló elrendezés hasonlósága alapján; nyitott kollektoros, amely egyszerű huzalozott VAGY kapcsolatot valósíthat meg, valamint kikapcsolt nagy impedanciájú harmadik logikai állapotba is vezérelhető rendszer, amely nélkül a bővítés, sínszerű szervezés ma már elképzelhetetlen). A tokozás típusa (szinte egyeduralkodó a két sor kivezetéses ún. dual-in-line tok), a tokok mérete, ill. a kivezető csapok közötti távolság szintén jól szabványosított jellemzők.

A mikroprocesszorokkal együtt a legnagyobb tömegben felhasznált kiegészítő elemek a különféle félvezetős tárak. Az elmúlt években a szabványosítás fejlődését jól megfigyelhettük ezen a területen is. A kezdeti időszak növekvő mértékű típusválasztékát („szétszóródás" korszaka) hamarosan vezértípusok rendezték, létrehozva az alapvető kategóriákat. (Egy-egy osztályon: RAM, ROM, PROM, REPROM stb. belül a kategóriát definiáló rendező a felépítés, a technológia, a sebesség, a kapacitás stb.) A kategóriák vezértípusai ezután egyenértékű típusokká váltak, biztosítva a különböző gyárak — sokszor nagymértékben különböző alaptechnológiával gyártott – termékeinek teljes csereszabatosságát. Azonos paraméterek és bekötés stb. valósítja meg az ún. helyére dugást biztosító kompatibilitást (plug-in-compatibility). A típusszám csökkentése a gyártás területén a gazdaságosság növelésének és így az árcsökkentésnek az alapfeltétele volt. Ez persze ebben a formában is hatott az alkalmazásokra. Ezen túlmenően azonban nehézséget okozott kezdetben a PROM (egyszer programozható „csak olvasható") és a REPROM (csak olvasható, de újraprogramozható) elemek nagyon eltérő és nehézkes programozása, amelyhez sokféle és elég bonyolult berendezésre (pl. ROM programozókra) is szükség volt. Az egységesedés és egyszerűsödés előnyei itt már közvetlenül az alkalmazásoknál voltak kézzelfoghatóan érzékelhetők.

Már az adott mikroprocesszoros rendszer gyártását (kis- és nagysorozat közelítő gazdaságossággal), ill. fejlesztését (közvetlen átmenet a labormintából a prototípusba) könnyítették meg a csereszabatosság tágabb értelmezését biztosító tárolóelem-

családokkal. (Egymás helyére dugható, kompatibilis RAM-, PROM-, REPROM- és ROM-típusok.)

A számítástechnikai eszközök használatát és elterjedését már viszonylag hamar szabványok segítették elő. Ezek közül ismét csak példaként mutatunk be néhányat, amelyeket a mikroprocesszoros rendszerek természetes örökségként vettek át. A különböző kódszabványok alkalmazása, amelyek közül a legelterjedtebb az ASCII (American Standard Code for Information Interchange: információcsere amerikai szabványos kódja), az információk kompatibilis továbbításának előfeltétele. A különféle adathordozók (lyukszalag, mágnesszalag, cserélhető tárcsás mágneslemez, hajlékony mágneslemez stb.) szabványai is változtatás nélkül alkalmazásra kerülnek. A mikroprocesszoros rendszerekhez – ár és méret szempontjából legjobban illeszkedő – hajlékony mágneslemez (floppy) szinte szabványosnak tekinthető, mivel IBM kompatibilis, vagyis a legnagyobb számítógépgyártó előírásainak megfelel.

A mikroprocesszoros rendszer a „külvilághoz" (amely alatt szokásos perifériákat pl.: képernyős megjelenítőt, távolban levő terminált vagy számítógépet, folyamatszabályozásban használt érzékelőket vagy végrehajtó elemeket egyaránt érthetünk) egyre inkább standard interfészeken keresztül szabványosított eljárás (kommunikációs protokol) szerint kapcsolódik.

A CCITT (Consultative Committee of International Telegraph and Telephone: Nemzetközi Távíró és Távbeszélő tanácsadó bizottság) által elfogadott V. 24. olyan szabvány, amely az interfész pontos definícióját (jelek és feladataik, csatlakozótípus stb.) is megadja. A Hewlett–Packard cég – elsősorban műszerek számára kidolgozott – sínrendszerét a mikroprocesszoros berendezésekhez is kiterjedten alkalmazzák. A sín „neve" ugyanakkor jól tükrözi a szabványosítás szokásos folyamatát is. Először HP sínnek hívták, ez gyári előírás volt. Majd IEEE 488-nak, mely amerikai szabvány. Rövidesen az IEC (International Electrotechnical Commission: Nemzetközi Elektrotechnikai Bizottság) szabványosította, és ezzel nemzetközi szabványjavaslattá vált. Ezután hazánkban is elfogadták (MI 1245–75). A különféle kommunikációs protokol közül az ISO (International Standards Organization: Nemzetközi Szabványosítási Szervezet) és az ANSI (American National Standard Institute: amerikai nemzeti szabványosítási intézet) közös kommunikációs protokol hierarchiáját említjük (SC–12 referenciamodell) [4]. A szabványok léte lehetővé tette mindezek LSI alapon való megvalósítását is. Így számos speciális interfészelemet dolgoztak ki. (Pl. soros aszinkron kommunikációra UART: Universal Asynchron Receiver Transmitter, azaz univerzális aszinkron vevő–adó, IEC sínillesztő stb.)

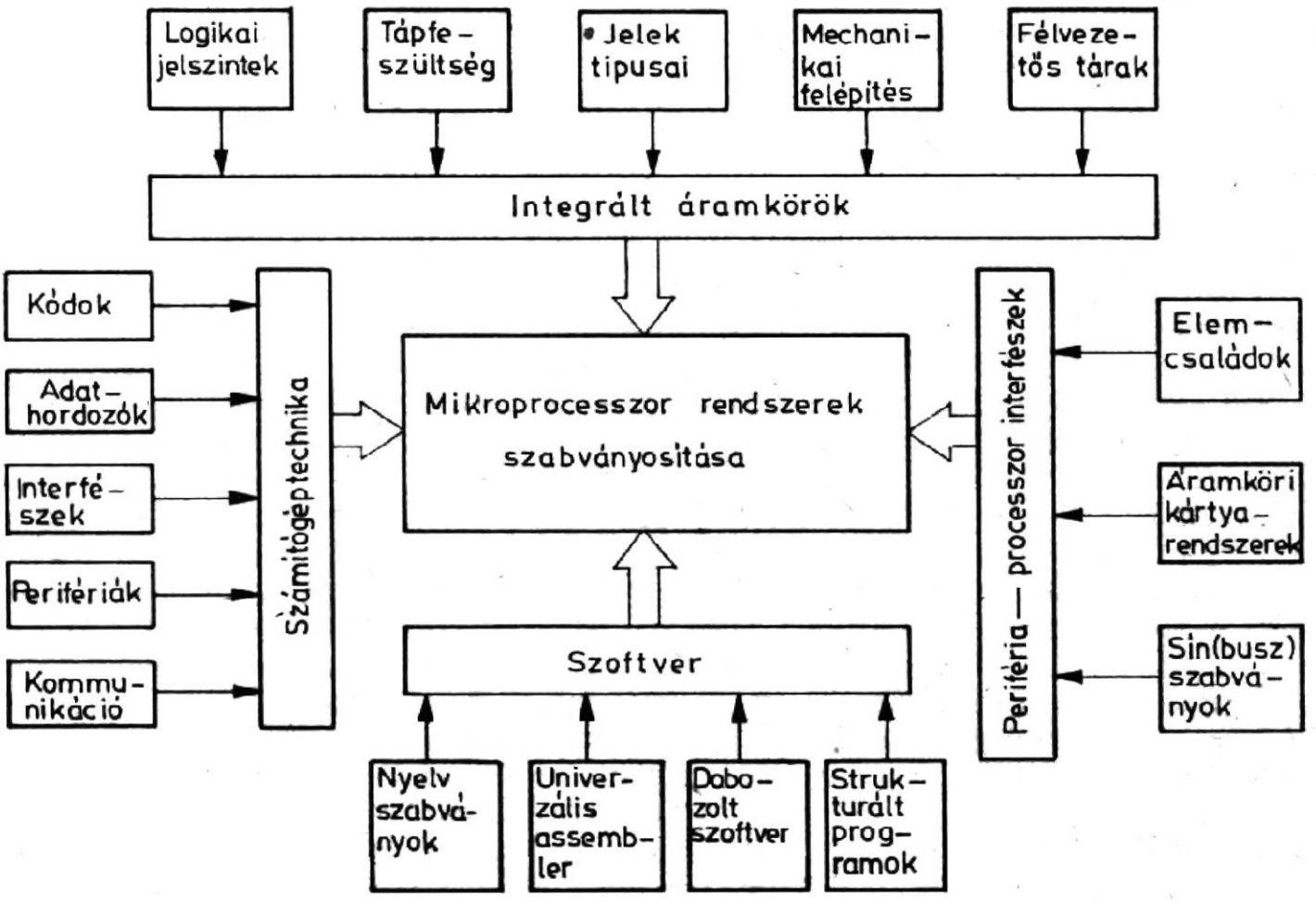
A mikroprocesszorok és kiegészítő elemeik hardverszabványosításának legfontosabb nyitott kérdése az elemek közötti kapcsolat – egykártyás rendszerek „belső" sínrendszere, a tokok jeleinek és kivezetéseinek – szabványosítása, ill. a funkcionális modulok közötti kapcsolat – többkártyás rendszer jeleinek és csatlakozó bekötéseinek – szabványosítása [5]. E feladatok megoldása rendkívül bonyolult, mivel nyilvánvaló, hogy egyetlen szabványos sínrendszer nem lehet alkalmas a különböző méretű, technológiájú, feladatú rendszerekhez. A rendszerek és elemek jelenleg alapvetően eltérőek (pl. processzorszóhossz: 1...32 bit; órajel-frekvencia 1...20 MHz; tárméret: 1 k...16 Mbyte; LSI technológia: NMOS, CMOS, I^2L stb. rendszer-architektúra: pl. multiplexelt és nem multiplexelt cím- és adatsín). A PPI (Processor-Peripheral Interface) szabványnak definiálnia kell digitális címek, adatok és vezérlési információk továbbítását processzorok és tárak, processzorok és kiegészítő elemek (Support devices): pl. interfészelemek, kombinált tár- és be-, ill. kimeneti elemek, perifériavezérlők, feldolgozók, valamint több processzor között. A szabványnak ki kell terjednie a funkcionális (milyen funkciók), működési (hogyan vannak implementálva), elektromos (logikai, jelszint és időzítési jellemzők) és mechanikai (csatlakozó, csapkiosztás) jellem-

zőkre is. A szabványosítás megvalósítása – bár a mikroprocesszoros rendszerek gazdaságos alkalmazásának a kulcskérdése – rendkívül sok, nagyon nehéz technikai, pénzügyi és „emberi" probléma megoldását teszi szükségessé.

Jelenleg a mikroprocesszorok hardverszabványosításának gyermekkorát éljük. Ezt a korszakot egyrészt a nagy cégek saját előírásai, másrészt a különböző nemzeti és nemzetközi bizottságok növekvő erőfeszítései jellemzik. Az elemek közötti kapcsolatot az elemcsaládokon belül jól meghatározott formában definiálták (pl. 8080 elemcsalád). Hasonló szinten egységesítettek az egyes gyártók által előállított kártyarendszerek csatlakozóira „telepített" sínrendszerek (pl. Z–8000 sin [6], MICROBUS [8], 9900 Busz [5] stb.). A PERCOM (**PER**sonal **COM**puter: személyi használatú számítógép) gyártók kiterjedten használják az S–100 sínrendszert [10]. A több processzorhoz (pl. MULTIBUS [9], ill. több processzortípushoz (pl. MUBUS [7]), alkalmazkodó sínrendszereket gyártóktól függetlenül dolgozták ki, de nem sikerült őket széles körben elterjeszteni [11].

Az 1.9. ábrán az előbbiekben röviden felsorolt alapvető szabványosítási problémákat még egyszer összefoglaltuk.

A hardverszabványosítás mellett legalább ennyire kulcskérdés a szoftver szabványosítása [12]. Ennek első „szint"-je – a nyelvek szabványosítása – szintén a számítástechnika eredménye. A BASIC nyelv mellett legnagyobb jelentőségre az APL (ill. ennek különböző gyártók által implementált részhalmazai (subset) pl. a PL/M, a PLZ) tett szert. Az utóbbi időben a két legelterjedtebb magas szintű nyelv – a FORTRAN és COBOL – helyét a PASCAL veszi át, amelynek a szabványát [13]a BSI (British Standards Institution: Angol Szabványügyi Intézet) az IEEE Computer Society (Amerikai mérnökegyesület számítógép társasága), az ISO és az ANSI együt-



1.9. ÁBRA. Mikroprocesszoros rendszerek szabványosításának „forrásai"

tesen dolgozták ki. A szabványosított nyelvek biztosítják a segítségükkel kidolgozott programok átvitelét (portability: „szállíthatóság") egyik rendszerről a másikra, de csak forrásnyelvi szinten. Az adott rendszerre megvalósított fordítóprogram (compiler) segítségével ezeket újra le kell fordítani, hogy a programok futtatható változatait megkapjuk. A PASCAL esetén egy közbülső fázis (ún. P-szint) léte és szabványosított formája az átvihetőséget elősegíti.

A különböző mikroprocesszorokra definiált gépi utasításkészletek alapvetően eltérnek. Ma a programok nagyobbik részét (elsősorban a gyorsabb működés és kisebb tárhelyfoglalás miatt) a gépi kóddal egyenértékű hatásosságú assembler nyelveken írják (és assembler segítségével kerül lefordításra). Az eltérő utasításkészletekhez eltérő tulajdonságú assemblereket dolgoztak ki. Az eltérés az utasításokat helyettesítő emlékeztető szimbólumokban (mnemonic), szimbolikus címzési lehetőségekben és operandusdefiníciókban, valamint az assemblert irányító parancsok, a direktívák típusaiban van. Így az assembler nyelven írt program – se forrás, se tárgy vagyis futtatható, lefordított formában – egyik mikroprocesszor-típusról a másikra általában nem vihető át. Kivételek az olyan mikroprocesszor-típusok, ahol az „alulról való" utasításrendszer szintű kompatibilitást tudatosan akarták biztosítani (pl. 8080 és Z–80).

Az eltérő assemblerek egységesítését kívánták megoldani a processzortípustól független emlékeztető szimbólum készletű univerzális assemblerekkel [14]. Ezek tényleges elterjedését megakadályozta, hogy a jelentős gyártók nem fogadták el egyiket sem.

A szoftverszabványosítás magasabb szintjét teszi szükségessé az „előregyártott szoftver" – ROM modulok formájában kapható – más szóval a „dobozolt szoftver" alkalmazása. Itt az egyes részek összeépítését és a felhasználói rendszer speciális feladataihoz és erőforrásaihoz való illesztését csak a modulok jól definiált és szabványosított „szoftverinterfésze" biztosíthatja. Ma csupán egyes gyártók adott rendszereinek keretei között van lehetőség ennek az elvnek a kezdetleges alkalmazására. Az egyes mikroprocesszor-típusok programkönyvtáraiból jó hatásfokkal csak számolásigényes algoritmusok és összetett műveletek részprogramjai és szubrutinjai vehetők át.

Az alkalmazó a „szabványosított" szoftver kidolgozásával, az ún. „szoftver engineering" [15] (mérnöki módon készített szoftver) módszereivel, elsősorban a strukturált programozáson [16] keresztül a „dobozolt" szoftver korszakának bekövetkezéséig is jelentős előnyöket biztosíthat.

1.3. AZ ELEMEK ARCHITEKTÚRÁJÁNAK FEJLŐDÉSE

A mikroprocesszoros rendszerek tulajdonságait meghatározó tényezők,

- az integrált áramköri technológia,
- a technológia alapján megvalósított elemkészlet, ill. annak a felépítése,
- a konkrét alkalmazás specifikus körülményei és problémái,
- a megvalósítás módszerei és eszközei,

mindegyike jelentősen átalakult az alkalmazások kezdeti szakaszában.

Az integrált áramköri technológia – bár minőségi és mennyiségi fejlődése nem lassult – egyre kevésbé meghatározó, mivel egyre kevésbé jelent korlátozó tényezőt az elemek architektúrája számára (akár a realizált funkciók számát, akár a végrehajtás sebességét tekintjük).

A konkrét elemtípusok komplexitása és működési sebessége egyaránt folyamatosan nő. A korszerű mikroprocesszorok és komplex kiegészítő elemeik lapkáin már 25 000...70 000 tranzisztort találunk (szemben a korábbi mikroprocesszorok 5000... 10 000 tranzisztor/lapka értékével), ami már alig egy nagyságrenddel kevesebb, mint a legnagyobb (maxi) számítógépekhez felhasznált elemszám. A sebesség növekedését jól

jellemzi az, hogy pl. a nagy kapacitású dinamikus tárolóelemek elérési ideje 100 ns alatti, amely néhány éve még a MOS technológiával elképzelhetetlen volt.

Az új elemek kidolgozása során ugyanakkor alapvető gazdasági követelmény – elsősorban a rendkívül magas tervezési költségek és a gazdaságos gyártáshoz szükséges darabszám miatt – a lehető legszélesebb alkalmazási terület és a lehető leghosszabb „élettartam", amit a változó igényekhez való alkalmazkodással lehet megnövelvelni.

A mikroprocesszoros rendszerekben felhasználásra kerülő elemek választéka rendkívül nagy és gyorsan változó. A továbbiakban csak azokat – az elemek felépítésére és architektúrájuk kialakítására vonatkozó – fő irányzatokat mutatjuk be, amelyeket a rendszertervezőnek figyelembe kell venni.

Mikroprocesszorok

A mikroprocesszorok architektúrája a gyorsan fejlődő technológia és az alkalmazási tapasztalatok hatására az elmúlt tíz év alatt nagyon sokat változott. Ez a változás mind a mennyiségi, mind a minőségi jellemzőkben megtalálható. Az első mikroprocesszorok (pl. 8008) csak egy számítógép központi egység (CPU: Central Processing Unit) magját tartalmazták, amelyet nagyszámú kiegészítő elem (SSI és MSI áramkör) segítségével lehetett csak használni. Ezt követően már lényegében egy teljes digitális számítógép központi egységet sikerült egy tokban integrálni (pl. 8080), amelynek a hatékony alkalmazását az interfészjelekkel kompatibilis felépítésű, növekvő számú és LSI bonyolultságú kiegészítő elem segítette elő (családelv). A technológia fejlődése ezután lehetővé tette nem csupán egy központi egység, hanem egy teljes számítógép „mag" (központi egység, tár, be-, ill. kimeneti interfész stb.) egyetlen tok formájában való realizálását, és így megszületett az egylapkás, egyetlen tok formájában megvalósított mikroszámítógép [17].

A fejlődés „mérőszámának" tekinthetjük a mikroprocesszorok segítségével közvetlenül feldolgozható bitek számát is. Az első mikroprocesszor 4-bites volt (4004), majd nagyszámú 8-bites típust dolgoztak ki (8080, 6800, Z80 stb). Ezt követően jelentek meg a 16/32 bites processzorok (I 8086, Z 8000, N 16 000, M 68 000 [18], [19], [20]).

Nagyon valószínűnek látszik, hogy a mikroprocesszor-típusok a jövőben a következő három – az alkalmazások bonyolultsága által definiált – kategóriába fognak tartozni [23]:

1. Processzorok elsősorban egyszerű vezérlési feladatok ellátására (kontrollerek). Itt a kompakt felépítés (egyetlen lapkás számítógépelv alkalmazása) mellett, az egyszerű utasításrendszer és rövid szóhossz (4...8 bit) a fő jellemzők.
2. Összetettebb alkalmazásokhoz használható processzorok. Mind a sebességet, mind az általános felépítést és utasításrendszert figyelembe véve, a jelenlegi átlagos 8-bites processzorok specifikációja kielégítő.
3. A számítógép-orientált processzorok, amelyek specifikációja eléri és meghaladja a jelenlegi kis, és miniszámítógépek jellemzőit. Így ezek átveszik azok helyét, ezzel megszüntetve a külön mikro-, mini- és kisszámítógép-kategóriákat. Ezek első változatainak kidolgozásakor fő szempont az utasítás szintű szoftverkompatibilitás volt a kidolgozott nagyszámú program „átmentése" céljából (pl. IM 6100 megfelel a PDP8-nak, az LSI11 a PDP11-nek és a 9445 a NOVA-nak stb.) Az ilyen „igazi" számítógépek számára készült processzorok kidolgozásakor fontos szempontnak tekintették [21]

- egy architektúracsalád definiálását, amely a technológia és az igények változásával különböző – eltérő változatok – kompatibilis megvalósítását teszi lehetővé,

- miniszámítógépszerű utasításrendszer és architektúra kialakítását, amely egyszerű programozást tesz lehetővé,
- bővíthetőséget, amely új jellemzők és szolgáltatások (pl. lebegőpontos és string-műveletek) későbbi beépítését teszi lehetővé,
- magas szintű programnyelvek implementálását és rendszerprogramok (pl. operációs rendszer) megfelelő támogatását.

Példaként az MC 68 000 processzor először megvalósított változata fő jellemzőit az 1.4. táblázatban foglaltuk össze [20].

1.4. TÁBLÁZAT. MC 68 000 fő jellemzői

Jellemzők	Értékek
Egész számok ábrázolása	8, 16 és 32 bit
Címzési tartomány	16 777 216 byte
Be-, ill. kimenet (I/O)	tárba ágyazott (memory-mapped)
Technológia	HMOS
Belső ciklusidő	250 ns
Tárhozzáférés	500 ns
Relatív teljesítőképesség	A 6800 processzor 10...25-szöröse
Tokkivezetésszám	64
Tápfeszültség	+5 V

A processzor a tárhoz való hozzáférést hosszú szó (32 bit), szó (16 bit), byte (8 bit) és bit (1 bit) egységekben teszi lehetővé. A központi egység erőforrásai között külön programszámlálót, állapotregisztert és tizenhat 32 bites általános célú regisztert találunk, amelyek közül nyolc elsősorban operandus forrás- és célregiszterként, valamint indexregiszterként, nyolc pedig elsősorban címzésre (pl. veremtármutató) használható.

A címszámítást az utasítás 6-bites mezője specifikálja. A következő címzési módok léteznek, amelyek bármelyike használható minden olyan utasítás esetén, amelynél címzés van.

- Közvetlen regisztercímzés. Az operandust az adat- vagy a címregiszter tartalmazza.
- Közvetett regisztercímzés. Az operandus címe a megadott címregiszterben van.
- Közvetett regisztercímzés utólagos címnöveléssel. Mint az előbbi, csak címzés után a kijelölt regiszter tartalmához, az operandus byte-száma (1,2 vagy 4) hozzáadódik.
- Közvetett regisztercímzés előzetes címcsökkentéssel. Mint az előbbi, csak a tartalomcsökkentés megelőzi az operanduscímzést.

1.5. TÁBLÁZAT. Az MC 68000 címzési módjai

KÖZVETLEN REGISZTERCÍMZÉS	
Adatregiszter közvetlen	EA = D_n
Címregiszter közvetlen	EA = A_n
Állapotregiszter közvetlen	EA = SR
KÖZVETETT REGISZTERCÍMZÉS	
Regiszter közvetett	EA = (A_n)
Regiszter közvetett utólagos címnöveléssel	EA = (A_n); $A_n \leftarrow A_n + N$
Regiszter közvetett előzetes címcsökkentéssel	$A_n \leftarrow A_n - N$; EA = (A_n)
Bázis relatív	EA = (A_n) + d16
Indexelt	EA = (A_n) + (X_n) + d8
PROGRAMSZÁMLÁLÓN KERESZTÜLI RELATÍV CÍMZÉS	
Relatív módosítások	EA = (PC) + d16
Relatív indexelt	EA = (PC) + (X_n) + d8
Rövid programszámláló relatív elágazás	EA = (PC) + d8
Hosszú programszámláló relatív elágazás	EA = (PC) + d16
ABSZOLÚT CÍMZÉS	
Rövid abszolút	EA = (következő utasításszó)
Hosszú abszolút	EA = (következő két utasításszó)
KÖZVETLEN ADATCÍMZÉS	
Közvetlen	DATA = következő utasításszó vagy szavak
Gyors közvetlen	DATA = utasítás-részmező (4 bit)

JELÖLÉSEK

EA: effektív cím
A_n: címregiszter
D_n: adatregiszter
X_n: indexregiszterként használt cím- vagy adatregiszter
SR: állapotregiszter
PC: programszámláló
d8: 8-bites módosítás
d16: 16 bites módosítás
N: operandushossz (1 byte, 2 szó és 4 hosszú szó esetén)
DATA: adat
( ) valaminek a tartalma

- Indexelt címzés. Az operandus címe a megadott regiszter tartalma plusz egy további (adat- vagy cím-) regiszter tartalma és egy az utasításban megadott 8-bites előjeles eltérés összege.
- Relatív bázisregiszteres címzés. Az operandus címe a megadott regiszter tartalma és az utasításban szereplő 16 bites előjeles eltérés összege.
- Relatív programszámlálón keresztüli címzés. Mint az előbbi, csak a bázisregiszter a programszámláló.
- Indexelt címzés programszámláló segítségével. Mint az előbbi, csak az alap a programszámláló aktuális értéke.
- Abszolút címzés. Az operandus címe van az utasításban.
- Közvetlen címzés. Maga az operandus van az utasításban.

Ezeken kívül van még a bitcímzés is, ahol a megcímzett szó bitértékét az utasítás vagy egy adatregiszter tartalmazza.

Az MC 68 000 címzési módjait a 1.5. táblázatban foglaltuk össze.

A címkezelést számos utasítás támogatja, amely az adott címregiszterre vonatkozik. Ilyen: **MOVE, COMPARE, INCREMENT, DECREMENT, ADD TO, SUBSTRACT FROM** (mozgatás, összehasonlítás, növelés, csökkentés, összeadás, kivonás). Ezenkívül van a **LOAD EFFECTIVE ADDRESS** utasítás (effektívcím-betöltés), amely a kiszámított eredményt a megadott címregiszterbe írja. Így lehetőség van a cím megelőző kiszámítására a processzor bármelyik címzési módjának felhasználásával.

Mivel nyolc címregiszter van, ritkábban van szükség címek időleges tárolására és visszaírására, valamint a program különböző helyein való újrakiszámolására. Ez nagymértékben csökkenti a címkezelésre fordított időt, amely jelentős volt a korábbi mikroprocesszorok esetében. Ugyanakkor az összetett címzési módokat felhasználva, leggyakrabban az utasítás címdefiniáló része adja meg a hosszú, 32 bites címet csupán 6-bites mező segítségével.

A 68 000 processzor különböző adattípusok használatát teszi lehetővé. Általában az adattípus független a címzési módtól, és az operandus nagysága is függetlenül adható meg. Az operandusok forrásai regiszterek vagy tárhelyek lehetnek, és az eredmény is kerülhet regiszterbe vagy tárrekeszbe (regiszter–regiszter, tár–regiszter, regiszter–tár osztályok). Az adatmozgató utasításokhoz két független címzési mód és címregiszter is rendelhető, így ezek tár–tár típusú műveletet valósítanak meg. Az adattípusok a következők:

- Egész. 1, 2 vagy 4 byte (szorzás és osztás esetén csak 2 byte).
- Logikai. 1, 2 vagy 4 byte (léptetés és forgatás művelete több bitpozíciónak megfelelő is lehet).
- Boole. 1 byte.
- Bit. Egyedileg címzett bitérték.
- Decimális. Két BCD (Binary Coded Decimal: binárisan kódolt decimális) számjegy egy byte-ban.
- Karakter. Karakterutasítások esetén.
- Cím. 1, 2 és 4 byte, címkiszámító műveletek esetén.
- Valódi (Real). Lebegőpontos műveletek esetén (az első változatnál még nem használták).
- Karaktersorozat. Ezekkel végezhető műveletek a MOVE, SEARCH és TRANSLATE vagyis mozgatás, keresés és fordítás (ezeket az első változatban még nem implementálták).

A **MOVE** és **COMPARE** utasítások, vagyis a mozgatás és összehasonlítás a tárban levő operandusokra is vonatkozhat (tár–tár típusú műveletek).

Az MC 68000 utasításkészletét az 1.6. tábázatban foglaltuk össze.

1.6. TÁBLÁZAT. Az MC 68000 utasításkészlete

Emlékeztető szimbólum	Jelentése
ABCD	Decimális összeadás kiegészítéssel
ADD	Összeadás
ADDX	Összeadás kiegészítéssel
AND	Logikai ÉS
ASL	Aritmetikai léptetés balra
ASR	Aritmetikai léptetés jobbra
BCC	Feltételes elágazás
BCHG	Bitvizsgálat és módosítás
BCLR	Bitvizsgálat és törlés
BRA	Feltétel nélküli elágazás
BSET	Bitvizsgálat és beállítás
BSR	Szubrutinra ugrás
BTST	Bitvizsgálat
CMK	Regiszterhatárok ellenőrzése
CLR	Operandustörlés
CMP	Aritmetikai összehasonlítás
DCNT	Csökkentés és ugrás, ha nem zérus
DIVS	Előjeles osztás
DIVU	Előjel nélküli osztás
EOR	Kizáró VAGY
EXG	Regisztercsere
EXT	Előjel-kiterjesztés
JMP	Ugrás
JSR	Szubrutinra ugrás
LDM	Többszörös regiszterbetöltés
LDQ	Gyors regiszterbetöltés
LEA	Effektív regiszterbetöltés
LINK	Veremtár-hozzárendelés
LSL	Logikai léptetés balra
LSR	Logikai léptetés jobbra
MOVE	Mozgatás
MULS	Előjeles szorzás
MULU	Előjel nélküli szorzás
NBCD	Decimális negálás kiterjesztéssel
NEG	Kettes komplemens
NEGX	Kettes komplemens kiterjesztéssel
NOP	Művelet nélküli utasítás
NOT	Egyes komplemens
OR	Logikai VAGY
PEA	Effektívcím-tárolás (push)
RESET	Külsőelem-törlés
ROTL	Balra forgatás kiterjesztés nélkül
ROTR	Jobbra forgatás kiterjesztés nélkül
ROTXL	Balra forgatás kiterjesztéssel
ROTXR	Jobbra forgatás kiterjesztéssel

1.6. TÁBLÁZAT (folytatás)

Emlékeztető szimbólum	Jelentése
RTR	Visszatérés és visszatárolás
RTR	Szubrutinból való visszatérés
SBCD	Decimális kivonás kiterjesztéssel
SCC	Feltételes beállítás
STM	Többszörös regisztertárolás
STOP	Állj
SUB	Kivonás
SUBX	Kivonás kiterjesztéssel
SWAP	Fél regiszter adatcsere
TAS	Vizsgálat és operandusbeállítás
TRAP	Csapda
TRAPV	Csapda túlcsordulás esetén
TST	Vizsgálat
UNLK	Veremtár-hozzárendelés megszüntetése

Az operációs rendszer hivásához tizenhat csapda- (trap-) utasítás van. A processzor két állapotban – felhasználói vagy felügyelő program állapotban (User vagy Supervisor state) – működhet, amelyet a külvilág felé is jelez. Felhasználói módban bizonyos utasítások (pl. STOP, RESET és azok, amelyek az állapotszót módosítják) nem megengedettek.

A felügyelői állapotba való – a felhasználói programtól független – visszatérést a megvalósított két független veremtármutató: egy a felhasználói, egy pedig a felügyelőprogram számára teszi lehetővé. A felügyelőprogram számára a felhasználói veremtármutató tartalma hozzáférhető. Ez a felhasználói és a rendszerállapot megkülönböztetés lehetővé teszi kis operációs rendszerek esetén is több feladatos vagy több felhasználós rendszerek kialakításánál védett virtuális címmező alkalmazását.

A 68 000 processzor architektúrájának tervezésekor fő szempont volt, hogy az hatásos támogatást nyújtson magas szintű programnyelvek számára mind a fordítás, mind pedig a végrehajtás, futtatás idején. Az utóbbi követelmény úgy is fogalmazható [20], hogy a végrehajtáshoz a programnyelveknek a futtatás során kell – speciális utasítások segítségével – megfelelő támogatást nyújtani. Példaként ezek közül is bemutatunk néhányat.

- Tömbkezelés. A **CMK** utasítás összehasonlítja az egy adatregiszterben levő, előzőleg kiszámított tömbindexet. Ha az index kívül van a tömb számára megadott határokon, „csapdába" kerül.
- Korlátozott pontosságú aritmetika. A **TRAPV** utasítás abban az esetben okoz csapdát, ha a megelőző utasítás túlcsordulást eredményezett. Ez hatásos túlcsordulás-ellenőrzés megvalósítását teszi lehetővé.
- Hurok megvalósítása. Hurok (for) valósítható meg egyetlen utasítással, amely a számlálót csökkenti, és ha az eredmény zérus, visszaugrik.
- Boole-kifejezés feldolgozása. Az **SCC** utasítások IGAZ vagy HAMIS értéket rendelnek egy logikai változóhoz ugyanolyan feltételek között, mint a **BCC** utasítások. Ez segíti a logikai kifejezések feldolgozását külön feltételes ugrási utasítások alkalmazása nélkül.

■ Eljáráshívás. Veremtármutató segítségével egymásba skatulyázott eljáráshívás valósítható meg. Egy magas szintű programnyelv eljáráshívása három utasítással (plusz paraméterként további utasítás) megoldható. Az egész hívási mechanizmus csak a veremtárat használja és teljesen „újra behozható" (re-entrant).

A fenti, kiragadott példák is bizonyítják, hogy a mikroprocesszorok új generációjának olyan tulajdonságai vannak, amelyek azelőtt csak a fejlett miniszámítógépeknek voltak.

Tárak

A félvezetős tár elemek tulajdonságai kiemelkedően fontos hatást gyakorolnak a mikroprocesszoros rendszerekre, és mint ezek alapvető, mindig jelen levő elemei, sok szempontból meghatározó tényezők a rendszer tervezése során. Áruk szinte megszabja a teljes hardver rendszer árát, teljesítményigényük annak teljesítményfelvételét. Elérési idejük a rendszerek műveleti sebességét nagymértékben befolyásolja. Mind az egyre összetettebb, nagyteljesítményű mikroprocesszorok, mind az ezekkel megoldható egyre bonyolultabb feladatok nagyobb és nagyobb kapacitású tárak használatát teszik szükségessé. A magas szintű programnyelvek egyre gyakoribb alkalmazásához is a nagy kapacitású gyorstárakra van szükség.

Félvezetős tárak mind bipoláris, mind MOS (Metal-Oxide Semiconductor: fém-oxid félvezető) technológiával készülnek. A MOS technológiával készülő mikroprocesszorokhoz ár, felépítés, műveleti sebesség stb. szempontjából jól illeszkednek a MOS-tárak, így ezek programjainak és adatainak tárolására szinte kivétel nélkül ezeket használják. A bipoláris tárak fő alkalmazási területe a gyors működésű vezérlők, ill. a gyors működésű – bipoláris technológiával készülő – mikroprocesszor-elemek (mikroprogramozott mikroprocesszor-szeletek) programjai (mikroprogramjai).

Az alkalmazott tárak vagy ROM-ok (Read Only Memory: csak olvasható tár) vagy RAM-ok (Random Access Memory: tetszőleges hozzáférésű tárak), amelyek a tárba írást és a tárból kiolvasást egyaránt lehetővé teszik.

A ROM-ok fő típusai:

■ Maszkprogramozható ROM. Ez esetben a tár tartalmát az elemgyártó írja be a gyártási folyamat részeként. Eleinte ezt az első ún. vezérlő- (gate-) maszk segítségével végezték a gyártás kezdeti fázisában. A gyorsabb átfutás érdekében ma már a negyedik kontaktmaszk vagy az ötödik fémezőmaszk segítségével programoznak. Ez lehetővé teszi a majdnem kész „szeletek" tárolását és a programozás befejező fázisának későbbi gyors elvégzését. Újabban lézer alkalmazásával is égetnek be programokat ROM-elemekbe. A beírás teljesen automatizált folyamatához a beírandó adatokat a felhasználó előre megadott formában kell, hogy szolgáltassa. Újabban listák helyett közvetlenül a beíráshoz felhasználható adathordozókon (lyukkártya, papírszalag vagy PROM) kell megadni az adatokat, vagy azok a távadatátviteli hálózaton keresztül is beküldhetők.

■ Elektromosan programozható és ultraibolya fénnyel törölhető EPROM (Electrically PROgraMmable: elektromosan programozható ROM) elemek törlés után újra programozhatóak. Ennek alapja az ún. FAMOS-elem (Floating Avalanche MOS: lebegő lavina MOS, mivel a lebegő vezérlőelektróda szigetelt „szigetére" való töltésbevitel lavinaletörés segítségével történik). A szilícium-dioxid réteggel szigetelt vezérlő befolyásolja az áramforrás (source) és nyelő (drain) elektróda közötti áramot. Programozáskor a forrás és vezérlő között alkalmazott nagy pozitív feszültség elektronokat visz a lebegő szilíciumvezérlőre (gate). A feszültség eltávolítása után az megtartja negatív töltését, mivel elektromosan szigetelt (szilícium-dioxid réteggel van körülvéve), és így lényegében nincs módja kisülni. Ez azután aszerint, hogy van-e töl-

tése vagy nincs, a vezérlő „sziget" alatt létrehoz vagy nem hoz létre egy vezetőcsatornát a forrás és a nyelő között. Ez jelenti azután azt, hogy bináris *0* vagy *1* értéket tároltunk a tár cellájában. A tárolt bitek ultraibolya fénnyel az elem kvarc ablakán keresztül törölhetők. Az ultraibolya fény fotoáramot hoz létre a szilícium-dioxid rétegben, amely a lebegővezérlő töltését az alaplemezbe viszi.

- Elektromosan törölhető EEROM- (Electrically Erasable: elektromosan törölhető ROM) elemeknél az összes bit egyidejű törlését elektromos impulzus segítségével végzik. Belső felépítésük hasonló a lebegővezérlő-elektródás EPROM-okhoz.
- Elektromosan módosítható EAROM- (Electrically Alterable: elektromosan módosítható ROM) elemekhez olyan speciális eljárást alkalmaznak, amely a tárhelyek újraprogramozását teszi lehetővé. Az újraprogramozás szóhasználat azt jelenti itt, hogy bár ezek „nem felejtő" (a tápfeszültség kikapcsolása után az információt megőrző) RAM-ok, de az írási idejük nagyon hosszú így „igazi" írható-olvasható tárként nem használhatók.
- A felhasználó által programozható PROM- (Programmable: programozható ROM) elemek olyan tármátrixot tartalmaznak, amelyben egy tranzisztor- vagy diódaelektródával sorban egy kiégethető „biztosíték" van. Azokat az elemeket, ahova *1* logikai értéket kell írni, egy programozókészülék segítségével megcímezik, amely azután ezen helyeken – a dióda vagy tranzisztor segítségével – akkora áramot bocsát át, amely a biztosítékokat kiégeti. Így ebben az ábrázolásban a nyitott (kiégetett) állapot jelenti a logikai *1*-et. Az információ ezen a módon állandóan megmarad a tárban, az egyszer kiégetett biztosíték többet vissza nem állítható, tehát az annak megfelelő tartalom nem módosítható.
  Az újra programozható ROM elemeket szokták REPORM-nak is nevezni

A RAM-ok fő típusai:

- Statikus RAM-ok, ahol az egyes cellákban az információ tárolását statikus működésű bistabil áramkörök végzik.
- Dinamikus RAM-ok, ahol az információ tárolását a MOS-tranzisztor nagy bemeneti impedanciájú vezérlőelektróda-kapacitása végzi. A nagyfokú integrálást a korszerű elemek (egy tranzisztor cellánként) felépítése teszi lehetővé. Az idővel csökkenő töltést azonban meghatározott időnként pótolni kell, így az ilyen típusú tárakhoz „felfrissítésről" kell gondoskodni.
- Kvázistatikus (Pseudostatic) RAM-ok, amelyeknél a dinamikus RAM-ok jó tulajdonságait (nagy kapacitás, olcsó ár, kis teljesítményigény stb.) igyekeztek párosítani a statikus RAM-ok előnyeivel (egyszerű használat).

Az elmúlt időszakban a félvezetős tárolóelemek látványos mennyiségi és minőségi fejlődésének voltunk a tanúi. Ez a folyamat változatlanul tart. A továbbiakban ennek olyan eredményeit kívánjuk röviden összefoglalni, amelyek az ezeket alkalmazó rendszertervező számára is lényegesek, és így azokat tervezőmunkájában közvetlenül vagy közvetve fel kell, hogy használja.

A RAM- és ROM-elemek fejlődésének általános irányzatai

- A belső felépítés egyszerűsödése, amely az egyetlen tokban levő lapkán, nagy sorozatban, jó kihozatallal gyártható elemek kapacitását nagymértékben megnövelte.
- A bitkapacitás növelésével és a nagy kapacitású elemek kisebb rendszerekben való gazdaságos alkalmazása érdekében minden csoportban – beleértve a nagy kapacitású dinamikus RAM-okat is – megjelentek a bitszervezés mellett a byteszervezésű tárelemek is.
- A tápellátás nagymértékben egyszerűsödött azzal, hogy a tárolóelemekhez csupán egyetlen tápfeszültségre: $+5\ V \pm 10\%$-ra van szükség.

- A teljesítményigény általános csökkentése mellett további jelentős disszipációs-teljesítmény-megtakarítás érhető el a tárolóelem „tartalék üzeme" (stand-by) megvalósításával. A rendszerben keletkezett meleg mennyiségének csökkenése ugyanakkor egyszerűsíti az ennek elvitelét végző hűtőrendszer tervezését és javítja a megbízhatóságot is. A csökkentett teljesítmény a feszültségkimaradás esetén sokszor szükséges tartalommegőrzést biztosító rendszer tervezését is egyszerűsíti. Az élvezérelt rendszerű megvalósítás esetén a tárolóelemet kis teljesítményű, tartalék üzemmódból az engedélyező- (Chip Enable) jel éle kapcsolja át aktív állapotba. A statikus tárak esetében a „tartaléküzem" teljesítményfelvétele tovább csökkenthető a tápfeszültség – általában +3 V-ig való – csökkentésével („takaréküzem").
- A kimenetek nagy impedanciájú („harmadik") állapotai és a kiválasztó- (CS: Chip Select) jelek a kapacitásbővítést problémamentessé tették.

A ROM-elemek minden típusánál megfigyelhető irányzat az információbevitel (programozás) nagymértékű egyszerűsödése. A maszkprogramozott ROM-ok esetén ez az egyszerűsödés a rendelő–gyártó kapcsolatban jelentkező flexibilitásban mérhető le. A gyártók az adatok megadására nagyszámú különböző formátumot tesznek lehetővé. Az elkészült elemek verifikációjának egyszerűsítését is hasonló módon támogatják.

Az egyszer programozható, „beégethető" PROM-elemek terén a fő irányzat a programozás egyszerűsödése, megbízhatóságának növekedése és a programozási eljárások egységesítése egyetlen gyártó különböző típusai, ill. a különböző gyártók különböző típusai között. Ennek következtében sokféle típushoz azonos programozóberendezések alkalmazhatók, ill. a még meglevő különbségek miatt szükséges ún. „personality board"-ok („egyéni jellegnek" megfelelő áramköri kártya) típusválasztéka csökkenthető.

A törölhető és újraprogramozható EPROM-elemek programozási eljárása ugyancsak sokkal egyszerűbb és kényelmesebb lett. A programozáshoz használt összes szükséges jelek lényegében megegyeznek a normális olvasási (READ) üzemmód jeleivel. A programozás bármely individuális címen elvégezhető a megfelelő címnek a címvezetékek segítségével való kiválasztásával, az adatvezetékekre a beírandó információt adva és az elemkiválasztást végző – és a programozás céljára is felhasznált – egyetlen bemeneti pontra (CE/PGM), a beírást végző egyetlen (tipikusan 25 V amplitúdójú és 50 ms széles) impulzust adva. Ez lehetővé teszi többek között párhuzamosan megcímzett elemek effektív programozását is. Az OE (Output Enable: kimenetengedélyezés) jel segítségével a beírt információ rögtön ki is olvasható, és így a programozás eredménye verifikálható.

A nagy kapacitású dinamikus RAM-ok szabványosított bekötésű, kisméretű 16 kivezetéses tokban való elhelyezését a címzés két lépésben való multiplexelt bevitelével teszik lehetővé. A két címrész a tármátrix felépítéséhez is alkalmazkodó sor- és oszlopcím. Ez a szokásos üzemmódokon: olvasás, írás, olvasás–írás, olvasás–módosítás–írás, túlmenően egy további üzemmód, az ún. lapműködési mód (page mode) definiálását is lehetővé teszi. Lap üzemmódban a tárelem – azonos sorcím mellett – egymást követő oszlopcímeken, növelt sebességgel való működést tesz lehetővé a teljesítményfelvétel növekedése nélkül. Ez a lehetőség számos alkalmazásnál (pl. képernyős megjelenítők „felfrissítő" tára, képfeldolgozás stb.) előnyösen felhasználható.

A tárak felfrissítését ugyancsak a belső mátrix felépítésének megfelelően – a 128 sor címhez tartozó felfrissítőciklus formájában – általában 2 ms-onként kell elvégezni. A normális tárciklus is elvégzi a felfrissítést. A felfrissítést jelentős teljesítménycsökkenés mellett is el lehet végezni (RAS-only mode: csak a sorcím beíró jelet alkalmazó üzemmód). A tárba beírás az oszlopcímbeíró jellel (CAS: Column

Address Strobe, azaz oszlopcímbeíró jel), melyet ún. korai beírásnak neveznek vagy, ha a beírandó információ ekkor még nem áll rendelkezésre, a WRITE (írás) jel segítségével (delayed write cycle: késleltetett írás ciklus) végezhető el.

A kvázistatikus RAM-elemek „belül" dinamikus tárakat tartalmaznak, amelyeket – mivel a felfrissítésüket megkönnyítették – „kívülről" a statikus tárakat megközelítő kényelemmel használhatunk. A felfrissítés megkönnyítése a gyakorlatban többféle – a rendszer vagy az alkalmazás kívánalmaihoz alkalmazkodó – felfrissítési-ciklus-típus közül választhatunk a felfrissítési üzemmódot megszabó külső vezérlőjel segítségével. Mint minden „igazi" dinamikus RAM-elem esetén, bármely aktív ciklus (írás vagy olvasás) felfrissítést is végez. A választott felfrissítési módtól függetlenül adott számú (pl. 128) felfrissítési ciklust kell minden megadott felfrissítési intervallumban (pl. 2 ms) végrehajtani. A felfrissítés típusa lehet külső, amikor a felfrissítési címeket (pl. az összes sorcím) kívülről kell a tároló elemre megadni, vagy belső, amikor egy beépített felfrissítőszámláló segítségével generáljuk ezeket. Ha a megadott számú felfrissítési ciklus (amely akár írás, akár olvasás lehet) egymást közvetlenül követi, a felfrissítés csoportos (burst). Ha a felfrissítési ciklusok között „valódi" tárhozzáféréseket is végzünk, a felfrissítés elosztott (distributed).

A felfrissítő logika nagymértékben egyszerűsíthető, sőt sok esetben el is hagyható, ha a belső felfrissítés üzemmódot alkalmazzuk. Ebben az esetben a felfrissítést a megfelelő, egyetlen bemeneti pontra adott él segítségével elindíthatjuk. Ez a mód, akár normál, aktív, akár tartalék üzemben használható. A legtöbb mikroprocesszor-típus alkalmazásával kialakított rendszerek esetén célszerű lehet minden utasítás-elővételi ciklusnál felfrissítést végezni.

Számos mikroprocesszor-típus (pl. az MK 3880, Z–80 stb.) külön felfrissítés-indító impulzust is ad ki. Ilyen esetben a kvázistatikus RAM a rendszer szempontjából teljesen statikusnak tekinthető. Ha tartalék üzemben érkezik indítóimpulzus, a felfrissítés azonnal elindul, míg aktív üzemben a tár megőrzi a felfrissítési parancsot, és csak az éppen folyó ciklus befejezése után indítja a felfrissítést.

Belső vezérlésű felfrissítés esetén a felfrissítési címet belső felfrissítőszámláló szabja meg. A számláló tartalma minden felfrissítési ciklus után automatikusan megnő.

Jó szolgálatot tesz egy további, gyakran rendelkezésre álló felfrissítési mód – az automatikus felfrissítés – alkalmazása. Ezt használhatjuk a külső tápfeszültség lekapcsolt állapota esetén, amikor szárazelem vagy akkumulátor segítségével való tápellátással őrizhetjük meg a tárba tárolt információt (ez az ún. battery back-up), vagy a rendszerek bemérésénél gyakran használt „egy lépés" (single step) üzemmódban. Ebben az üzemmódban a tár minden megadott időintervallumban (pl. 15 μs-onként) belső vezérléssel, automatikusan végrehajt egy felfrissítési ciklust.

A félvezető technológiákat tekintve a tárak számára legnagyobb jelentősége ma az NMOS technológiának van, mind a teljesítmény–ár tényezőt, mind az egyéb gyártási-alkalmazási jellemzőket figyelembe véve. Az átlagos mikroprocesszoros rendszer körülményei között így elsősorban különféle NMOS tárolóelemeket alkalmazunk.

Külön említést érdemelnek a CMOS technológia segítségével kidolgozott tárolóelemek. Ezeknek ui. egyéb jó tulajdonságok mellett (pl. széles működési tápfeszültség-tartomány, nagyfokú tápfeszültségváltozás-érzéketlenség, széles marginális határok a logikai szinteknél stb.), rendkívül kicsi a sebesség–teljesítmény szorzatértékük. Ez azt jelenti, hogy teljesítményigényük már normál működési körülmények között is nagyon kicsi. Különösen kicsi azonban a teljesítményfelvétel a tápfeszültségcsökkentéssel (tipikusan 2 V) megvalósítható takaréküzem esetén. (Tipikus érték pl. a HM–6504 típusszámú, 4k×1 bit RAM-nál a 0,01...5 μA áramfelvétel.) Ez a tápfeszültség-kimaradás esetén való RAM-tartalommegőrzés nagyon egyszerű módját

teszi lehetővé egy kisméretű (pl. Ni-Cad) akkumulátor segítségével. A rendkívül kis áramfelvétel nagyon hosszú megőrzést tesz lehetővé, így a CMOS statikus RAM-ok alkalmazása – a dinamikus NMOS RAM-ok mellett – a leggyakoribb a feszültségkimaradás esetén való információmegőrzést megkívánó mikroprocesszoros rendszerekben.

A gyorsabb (bipoláris) RAM-okkal és PROM-okkal a mikrovezérelt (mikroprogramozott) mikroprocesszor-elemekkel kapcsolatosan még találkozunk majd. Különösen az ECL (Emitter Coupled Logic: emittercsatolt logika) tette lehetővé a rendkívül gyors (tipikus érték 10 ns) tárolóelemek kialakítását.

A tárak felépítéséről, szervezéséről, ill. a speciális felépítésű tárakról kell még említést tennünk. Alapvető jellemző: a tárolóelem szervezése. A tárolóelem lehet bit-, karakter- (4 bit), byte- (8 bit) vagy szó- (több mint 8, pl. 16 bit) szervezésű. Ezek közül a szükséges tár nagysága alapján gazdaságossági, konstrukciós szempontok alapján választanak. Fontos tulajdonság, különösen, ha az elemekből való egyszerű tárépítés szempontjára gondolunk, az elem engedélyezőjeleinek száma és a tokon belüli dekódolása (felismerése), amely az adott elem kapacitását meghaladó címtartományú tárak esetén fontos. A RAM-elemek esetén a közös vagy független be- és kimenet, vagy általában a kimenet felépítése (nyitott kollektoros, ún. totemoszlop vagy nagy impedanciájú, kikapcsolt harmadik állapotú) is fontos szempont. A párhuzamosan kapcsolt tárolóelemek egyszerű alkalmazása során az utóbbi megoldás a legkézenfekvőbb és így egyeduralkodó.

A speciális felépítésű tárak közül említésre méltóak a többszörös hozzáférést – vagyis több címmel definiált, több be- és kimeneti interfészen keresztül való egyidejű írást, ill. olvasást – lehetővé tevő tárak (Multiaccess memory).

A LIFO (Last In First Out: utoljára be, először ki, vagyis utoljára beírt tartalom olvasható ki először) más szóval veremtárelemek, a FIFO (First In First Out: először be, először ki) más szóval puffertárelemek, és a CAM (Content Addressable Memory: tartalom szerint címezhető tár) vagy más szóval asszociatív tárolóelemek már ritkábban szerepelnek a rendszerekben.

A soros hozzáférésű elemek közül kettő nagy jelentőségű a korszerű nagy kapacitású (háttér-) tárak megvalósításában. Az egyik a CCD (Charge-Coupled Device: töltéscsatolt elem), a másik a buborék- (bubble) tár.

Az 1.7. táblázatban a legfontosabb tártípusok néhány jellemzőjét foglaltuk össze az 1980. év elejének megfelelő sorozatgyártási szintet figyelembe véve [22].

Kiegészítő elemek

Ahhoz, hogy a mikroprocesszor valóban rendszerek megvalósításának korszerű eleme lehessen, szükség volt a kiegészítő elemek széles körének azonos technológiai bázison való megvalósítására is. Ez volt ui. a rendszerek „egyenszilárdságú" tervezésének és kialakításának az alapvető feltétele. A kiegészítő elemek köre az a terület, ahol a fejlődés (mind a definiált és megvalósított elemeket, mind a hozzájuk kapcsolódó módszereket tekintve) a legnagyobb mértékű volt. Ez a folyamat ma is tart, biztosítva a mikroprocesszorok alkalmazási területeinek további bővülését, ill. a már „elfoglalt" területeken a korszerűbb (egyszerűbb és így olcsóbb) megvalósítási körülményeket.

Az eddig megtett utat és a fejlődés irányát figyelembe véve, két – egymással szoros kapcsolatban lévő – folyamatot figyelhetünk meg.

Az egyik fejlődési irány a rendszerek hardver „építőkövei", az alkalmazott elemek fejlődésének a sajátossága. Az egyre összetettebb funkciók LSI formában való megvalósítása egyre kevesebb – a feladathoz való illesztést lehetővé tevő – kis és közepes integráltságú elem használatát teszi szükségessé. Más szóval a „külvilág" és a mikroprocesszoros rendszerek kapcsolatát biztosító LSI bonyolultságú elemek

1.7. TÁBLÁZAT. Tipikus félvezetős tárak és jellemzőik (1980. januári sorozatgyártási szintnek megfelelően)

Típus	Technológia	Kapacitás	Áram-, teljesítményfelvétel (Tartaléküzem)	Hozzáférési idő	Tipikus példa típusszáma	Gyártó cég
ROM	NMOS	8 k×8 bit	220 mW (35 mW)	250 ns	MK 36000	Mostek
ROM	CMOS	8 k×8 bit	100 μA	33 ns	HM 6389	Harris
PROM	bipoláris	2 k×8 bit	175 mA	55 ns	93510	Fairchild
EPROM	FAMOS	8 k×8 bit	840 mW (131 mW)	450 ns	TMS 2564	Texas
EPROM	CMOS	2 k×8 bit	100 μA	350 ns	IM 6316	Intersil
EAROM	MNOS	2 k×4 bit		1,4 μs	7810	Nitron
EEROM	FAMOS	2 k×8 bit		250 ns	48016	Hitachi
RAM (dinamikus)	NMOS	64 k×1 bit	200 mW (10 mW)	120 ns	MK 4164	Mostek
RAM (statikus)	HMOS—II.	4 k×1 bit	180 mA (30 mA)	45 ns	2148	Intel
RAM	CMOS	2 k×8 bit	5 mW	200 ns	6516	Harris
RAM	bipoláris	1 k×1 bit		10 ns	MB 7071	Fujitsu
FIFO	bipoláris	64×4 bit		15 MHz	67401A	Monolithic Memories
Buborék		1 Mbit		41 ms	7110 (kit)	Intel
CCD		16×4k bit		2M bit/s	TMS 3064	Texas

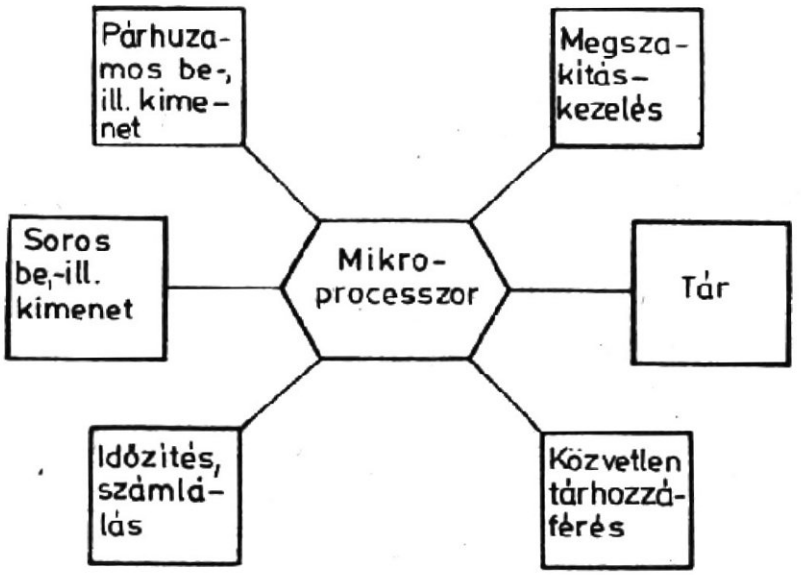
(be-, ill. kimeneti elemek, perifériavezérlők stb.) egyre több SSI és MSI elemet tudnak helyettesíteni, és ugyanakkor a különböző feladatokhoz is képesek jól alkalmazkodni. Az utóbbi követelmény két úton is megvalósul, egyrészt a programozhatóságon keresztül, a jellemzők változtatásával, másrészt – az egységesedés, szabványosítás terjedésével – egyre összetettebb funkciók és feladatok standard módon való megoldhatóságával.

A másik fejlődési irány az alkalmazások bővülésének és az alkalmazott rendszerek felépítésének a következménye. Az egyre összetettebb feladatok esetén és az ezek megoldásához szükséges egyre komplexebb rendszerekben a központi egység mikroprocesszora nem tud már a „periferikus" problémákkal foglalkozni. A „magas szintű" feladatoknál és ezekhez egyre gyakrabban felhasznált magas szintű programozási körülmények (rendszerprogramok, magas szintű programnyelvek stb.) között mind a perifériák és általában a be-, ill. kimeneti elemek vezérlését, mind az ezekkel való információcsere egész szervezését az önálló funkciók komplex megoldására képes kiegészítő elemeknek kellett átadni. Ugyanakkor a feladatok effektív és gyors megoldásához – a nagyteljesítményű rendszerekben – a központi processzorok saját hagyományos feldolgozási munkájuk részfeladatait is egyre inkább külső, kiegészítő elemekre bízzák. Ezeket a processzor típusú elemeket ismét kétféleképpen valósítják meg. Vagy valamilyen adott célra, megoldási módra orientált, valamilyen feladatra kifejlesztett speciális (dedicated) processzorok felhasználásával, vagy ismét a programozhatóság kihasználásával általános célú elemekkel. Az egyetlen tok formájában megvalósított mikroszámítógépek akár a felhasználó által programozhatók az adott feladatra, akár a gyártó „standard", előregyártott programokkal készítheti azokat. Az utóbbi eljárást megelőzte a programozható tárak gyári „standard" programozása és így új funkciók megvalósítása. Példaként gondoljunk a képernyős megjelenítőknél a karakterek alakját megszabó karaktergenerátorokra, amelyek – mint tudjuk – a gyártás szintjén egyszerű, maszkprogramozható ROM-elemek.

Az összetett rendszerek előbbiekben vázolt fejlődése (periferiális és feldolgozás típusú feladatok önálló processzorral való megoldása) a fő (master: mester, gazda) és alárendelt (slave: szolga) processzorok alkalmazása ezután elvezet a feladatorientált (dedicated) sokprocesszoros (multiprocesszoros) megosztott rendszerekhez.

A mikroprocesszorok alkalmazásának kezdeti korszakában nem volt semmiféle kiegészítő elem. Az első generációs mikroprocesszorok alkalmazásának természetes velejárója volt az a nagyszámú kis és közepes integráltságú elem, amely „körülvette" a mikroprocesszort, és lehetővé tette a kapcsolatát a megoldandó feladattal, hozzáillesztve az érzékelők és információforrások, ill. a végrehajtó, információkijelző, kiíróelemek jeleit a mikroprocesszorhoz. A jelek be- és kivitelét már a mikroprocesszor végezte, amely a kezdeti egyszerű alkalmazások körülményei között olyan feladatokat is ellátott, mint pl. egy numerikus vagy alfanumerikus billentyűzet figyelése, az egyes billentyűk állapotának egymás után való, folyamatos végigpásztázásával (scanning), majd egy billentyű jelentkezése után – annak a billentyűmátrixban elfoglalt pozíciója alapján való – kód hozzárendelése. A mikroprocesszorok ilyen egyszerű feladatokkal való „terhelése" a kezdeti, egyszerű alkalmazások körülményei között különösebb problémát nem okozott. Még külön előnynek is érezték a programozhatóságból adódó flexibilitást a be-, ill. kimeneti kapcsolatban is.

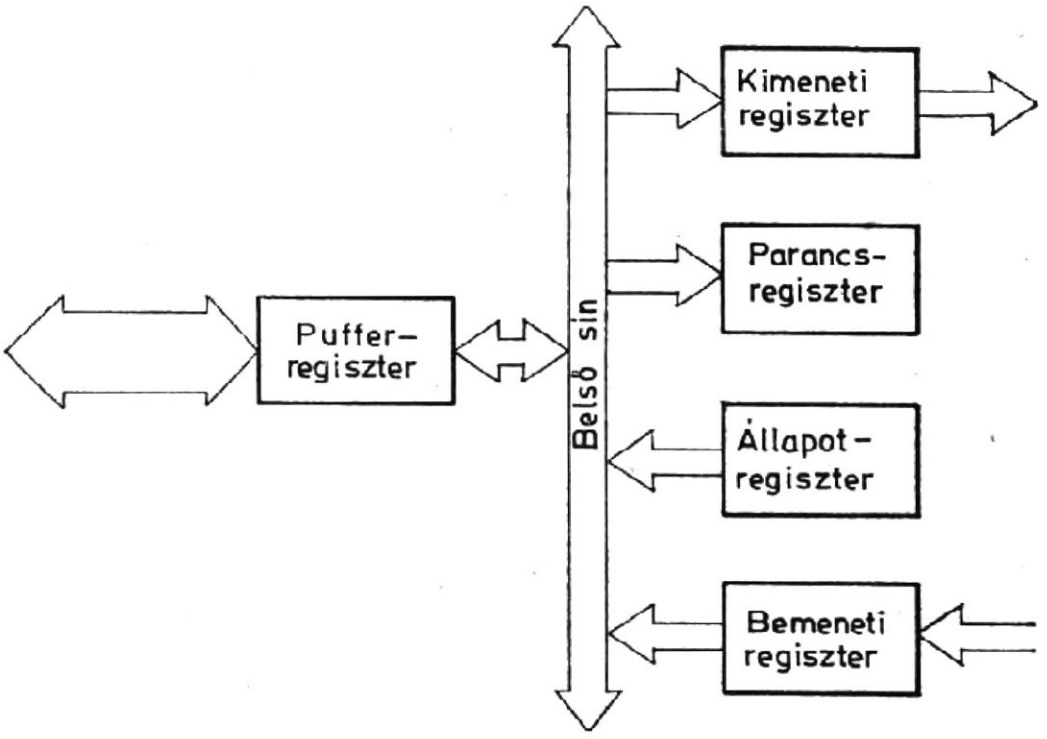
Így az első időszak rendszereinek hardverfelépítése csak nagyon kismértékben használta ki az LSI technológiából származó előnyöket. Ezen kívántak változtatni, amikor egyszerű kiegészítő funkciókra is egységeket hoztak létre, amelyekhez egy adott mikroprocesszossal való közvetlen (minden további elem nélkül való) kapcsolat létrehozására a mikroprocesszor jelei (cím-, adat- és vezérlőjelek) közvetlenül hozzácsatlakoztathatók voltak (családelv). Ilyen módon szinte valamennyi mikroprocesszor-



1.10. ÁBRA. Mikroprocesszorok alapvető kiegészítő elemcsoportjai

típushoz létrehoztak kiegészítő elemeket, amelyekkel a rendszerelemek közötti és a feladattal való kapcsolat könnyebben volt létrehozható. A 1.10. ábrán hat ilyen alap feladatkört mutatunk be, amelyekhez kapcsolódó elemeket ebben a „második generációs" korszakban dolgoztak ki. Ezek a megszakítás- (interrupt) kezelés, közvetlen hozzáférés (DMA: Direct Memory Access), tár- (és tárkiválasztó) elemek, időzítők (timer) és számlálók (counter), valamint soros és párhuzamos be- és kimeneti (I/O) elemek [24] [25]. Ezzel párhuzamosan – az integrált áramkörgyártás analóg ágának eredményeit is figyelembe véve – a közvetlen a folyamatokhoz való kapcsolat elemei (pl. analóg–digitális és digitális–analóg konverterek, multiplexerek stb.) és a nagytávolságú adatátvitel elemei: adó-vevők, kódoló-dekódolók (CODEC), modulátor-demodulátorok (MODEM) is többé-kevésbé processzor típustól független, de standardizált formában megjelentek.

A következő, „harmadik generáció" egyre összetettebb perifériavezérlői már programozhatók. A processzor saját cím-, adat- és vezérlőjelei segítségével információcserét folytat a kiválasztott vezérlő valamelyik regiszterével (sokszor egy puffer-regiszter közbeiktatásával). Ezek a regiszterek feladatuk szerint a következő négy alapcsoportba tartoznak (1.11. ábra): a be- és kimeneti regiszterek a külső adatforrás vagy -fogadó és a mikroprocesszor valamelyik belső regisztere közötti adatutat valósítják meg olyan módon, hogy a két oldalon, két független aszinkron működésű adatcserét lehessen létrehozni. A vezérlő parancsregisztere(i) a működés módját és paramétereit szabja meg a processzorból kapott utasításoknak megfelelően. Az átvitt információ tehát azonos módon juttatható a vezérlőbe akár adat, akár parancs típusú az. Ilyen módon programozható pl. egy soros átvitelt vezérlő elem esetén az átvitel típusa (szinkron vagy aszinkron), egy átvitt karakter hossza (bitszám), az átvitel sebessége (Baud-rate), a hibajelzést szolgáló paritásképzés módja (páros, páratlan, nincs) stb. A vezérlő mindenkori állapotát az állapot- vagy más szóval státuszregiszter tartalma mutatja. A processzor ennek kiolvasásával szerezhet tudomást pl. az átvitt információ meghibásodásáról vagy ezen belül is a meghibásodás módjáról (pl. keretezési hiba, amikor aszinkron átvitel esetén a záró stopbit helyén nem a megfelelő bit érkezik, tehát az információ „elcsúszott"). Hasonló elven készíthető mindenféle programozható elem. Pl. a megszakításvezérlő esetén az egyes megszakításkérésekhez rendelt prioritásszint programozhatóan változtatható, vagy a közvetlen tárhozzáférés esetén a szükséges átviteli paraméterek – kezdőcím és blokkhossz – mellett a mód is megadható. Ez lehet ciklusloposás (amely az egyes, a processzor



1.11. ÁBRA. Programozható illesztő- (vezérlő-) elem regiszterei

számára rendelkezésre álló tárciklusok közé ékelt ciklusokban végzi az átvitelt és ezért „átlátszónak": transparentnek is szokták nevezni), vagy egy vagy több blokk folyamatos átvitelét biztosító blokk- (burst: sorozat) átvitel. Egy vezérlő több azonos funkció kiszolgálását is végezheti és vannak többféle feladatot ellátó komplex vezérlők is. (Pl. az Intersil 6100, a DEC PDP8/E miniszámítógép központi egységet emuláló CMOS mikroprocesszor 6102 kiegészítő eleme DMA vezérlő mellett tárbővítőt, intervallum-időzítőt és megszakításvezérlőt is tartalmaz.)

A következő „negyedik generáció" kiegészítő elemei már nem csupán programozhatók, hanem összetett perifériák vezérlői, műveletcsoport vagy feladatorientált önálló processzorai. Ezeket ezért összefoglalóan szokták TPU- (Task Processing Unit: feladatfeldolgozó egység) egységnek is nevezni. Napról napra egyre újabb és újabb ilyen elemek jelennek meg, így még részkategorizálásuk is korai. Inkább illusztrációként néhányat bemutatunk közülük egy-egy tipikus feladat megoldására [26] [27].

Az új típusú, nagy (millió byte) címtartományú mikroprocesszorok és a korszerű operációs rendszerek alkalmazása a tárkezelés – számítógépek esetében már megszokott – formáit igényelte. Az MMU (Memory Management Unit: tárkezelő egység) a tár változtatható felosztását (segmentation), dinamikusan változtatható áthelyezését (relocation) és kijelölt szegmensek védelmét (protection) végzi.

A BAU (Bus Arbiter Unit: sínvezérlőegység) a sínrendszer osztott idejű felhasználásából adódó konfliktushelyzeteket oldja meg (Arbiter: döntőbíró), megfelelő elsőbbségi (prioritás) áramkörök segítségével előállítva a címdekódolás és a legáltalánosabb visszajelzéses aszinkron átvitel (hand-shaking: „kéz rázás") vezérlőjeleit is.

Az I/O (be-, ill. kimeneti) processzorok lényegében a számítógépek esetében már ugyancsak régen használt csatornaprocesszorok LSI technológiával megvalósított megfelelői. Saját – be-, ill. kiviteli műveletek szempontjából optimalizált – utasításrendszerükkel, saját tárukban lévő program segítségével valósítják meg a csatorna előírásainak megfelelő algoritmust. Nagy (1 M byte/s-nál nagyobb) átviteli sebességet is megengednek a csatorna számára. Pl. az Intel 8089-nél 5 MHz órajel-frekvencia esetén a maximális átviteli sebesség (transfer rate) 1,25 M byte/s, míg blokkos átvitel esetén a DMA kérésre a maximális válaszidő (latency: „lappangás") 2,4 μs.

Az APU (Arithmetic Processing Unit: aritmetikai processzor egység) az aritmetikai processzorok új osztályát jelképezi. Általában veremtár-orientált szervezésűek, amely az operandusok könnyebb kezelését teszi lehetővé, különösen többszörös pontosságú (több szavas) fixpontos adatformátum alkalmazása esetén. Ezeket a műveletkészlet, a műveleti sebesség (ahol a műveleti sebesség és a pontosság természetesen ellentétes tulajdonságok) és a rendszerbe illesztés módja szerint csoportosíthatjuk. Utóbbi csoportba tartozik az aritmetikai processzor, az alárendelt (slave) processzor vagy a társprocesszor (comprocessor). Az előbbi a központi egységként szereplő mester processzortól lényegében parancsok formájában kapott feladatot oldja meg, míg a társprocesszor maga figyeli a műveleti kódokat a rendszersínen, és felismeri és végrehajtja a neki szóló utasításokat. Alárendelt működésű aritmetikai processzor pl. az AM 9511. Egyszeres és kétszeres pontosságú fixpontos adatformátum mellett lebegőpontos formátumot is közvetlenül tud használni. Műveletkészlete az adatformátumoknak megfelelő alapműveleteken (összeadás, kivonás, szorzás, osztás) kívül trigonometrikus műveleteket (sin, cos, tg, arcsin, arccos, arctg) és további összetett műveleteket (négyze gyökvonás, hatványozás, ln, lg, e^x stb.) is tartalmaz. Manipulációs utasításai adatformátum-konverziót (fixpontosról lebegőpontosra és lebegőpontosról fixpontosra való áttérés), konstansok betöltését (π, e stb.) is lehetővé tesznek. Nagy pontosságú „ikertestvére" az AM 9512 már 64 bites számokat kezel és – a szabványosítás lassú, de biztos térhódítását bizonyítva – az IEEE lebegőpontos szabványának megfelelő felépítésű. A társprocesszor működésű Intel 8087 nemcsak nagy pontosságú (64 bit), és lebegőpontos aritmetikai műveleteket tud végezni, hanem olyan összetett trigonometriai függvényeket is képes kezelni, mint a hiperbolikus függvények.

Az SSP (Signal Processing Peripheral: jelfeldolgozó periféria) nem csupán összetett műveletek gyors és effektív megoldására képes, hanem komplex digitális algoritmusok gyors, sokszor a feldolgozás időtartományának (real time) megfelelő megoldására is. Ez a kategória tulajdonképpen olyan hardver úton megvalósított szubrutinmodulnak is tekinthető, amely a központi egység által megadott feladatot a „hívás" és az azzal együtt küldött paraméterek alapján oldja meg és csak az eredménnyel jelentkezik ismét (pl. megszakításkérés segítségével). Az alkalmazás olyan területeire, mint jelfeldolgozás, beszédfelismerés, képfeldolgozás stb. orientált processzorok egy-egy algoritmus pl. FFT (Fast Fourier Transform: gyors Fourier-transzformáció) szerint képesek optimális működésre. Felépítésük és utasításkészletük miatt is alkalmasak erre.

A komplex perifériavezérlők új generációja egyre összetettebb funkciókat valósít meg és egyre kevesebb külső kiegészítést tesz szükségessé. A mikroprocesszoros rendszerekhez legjobban alkalmazkodó tömegtár, a hajlékony lemezes tár programozható vezérlői az IBM formátum (az IBM 3740 és System 34-nek megfelelő) egyszeres és kettős sűrűségű rögzítést végeznek IBM-kompatibilis formában. A vezérlők a párhuzamos–soros, ill. soros–párhuzamos átalakításon túl frekvenciamodulálást és -demodulálást is végeznek. Az egyszeres sűrűség esetén a rögzítés módja FM (Frequency Modulation: frekvenciamoduláció), ahol az egy bitnek megfelelő cellánként két, míg a kétszeres sűrűség esetén alkalmazott MFM (Modified Frequency Modulation: módosított frekvenciamoduláció) esetén csak egy mágneses fluxusváltás van. A vezérlő képes a hajlékony lemezes tár formálására is, amelynél a változtatható szektorszervezésnek (soft sector) megfelelően rögzíti a szektoronkénti byte-számot, és gondoskodik az adatok hibavédelméről is CRC (Cyclic Redundancy Check: ciklikus redundanciavizsgálat) végrehajtásával. A hajlékony mágneslemezes tár vezérlője a külső kiegészítő elemeket: a DMA vezérlőt, amely a nagy adatátviteli sebesség miatt a központi egység felé való információtovábbításhoz kell, valamint az íráshoz szükséges

kompenzáció, amely a mágneses hatások miatti íróimpulzus-eltolódást helyesbíti és az olvasást megvalósító, általában PLL (Phase Locked Loop: fáziscsatolt hurok) áramkört közvetlenül vezérli.

Hasonló funkciójú vezérlőknél, amelyek a mágneslemezes tárak legfontosabb kategóriája (Winchester típusú diszkek) számára készülnek, a hibavédelmet – a kategória kívánalmainak megfelelően – külön processzor BEP (Burst Error Processor: csomósodott hibák processzora) végzi.

Megfelelő program irányítása alatt egy be-, ill kimeneti processzor alkalmazásával a mágneslemezestár-vezérlő olyan összetett feladatokat is önállóan végezhet, mint pl. keresési művelet (egy adott nyom–track–, nyomon belül egy szektor megkeresése, egy rekordban a megadott szavaknak a megtalálása).

További jellemző, a mikroprocesszoros rendszerekben fontos kategória a CRT (Cathode Ray Tube: katódsugárcső) megjelenítő vezérlők. A korszerű, egyetlen tok formájában megvalósított ilyen elem a következő jellemzők beállítását teszik lehetővé. Üzemmód: karakter és pont-raszter típusú grafikus megjelenítés. Specifikációs jellemzők: karaktertípus, beleértve a felhasználó által definiálható szimbólumokat (attribute) is, képernyőformátum: karakter/sor és karaktersor/lap, automatikus továbbmozgatás (scrolling: feltekerés) karakterenként, soronként vagy laponként, kurzortípus stb. Kiegészítő funkciók: billentyűzet, fényceruza vezérlése. A vezérlők a felfrissítést végző tárcímzését is automatikusan elvégzik. A felfrissítés olyan módon is szervezhető, hogy azt az azonos tárat használó rendszer ne érzékelje. Egyes CRT vezérlők átlapolt (interlaced) vagy átlátszó (transparent) más szóval versenymentes (contention-free) felfrissítést is képesek megszervezni.

Külön csoportot képeznek az adatátvitelhez és táv-adatfeldolgozáshoz felhasznált elemek. Ezek a kommunikációs protokolloknak egyre magasabb hierarchikus szintjén oldják meg a szabványosított szolgáltatásokat. A legalsó szint (ún. vonali protokollok) után jelenleg a második szint karakter (pl. IBM BiSync) [28], és a bitorientált (pl. IBM SDLC vagy ISO HDLC) protokolljainak megvalósítását végzik. Ezek az üzenet címzéséhez szükséges kiegészítő információ hozzáadása mellett összetett hibajelző és javító eljárásokat is megvalósítanak. Az alkalmazások egyre nagyobb körében szükséges adatvédelem megvalósítására olyan kiegészítő elemeket is kidolgoztak, mint az NBS (National Bureau of Standard: amerikai szabványügyi hivatal) előírásainak megfelelő rejtjelző (encryption) egység, amely 8 byte-ból álló adatblokkok – a rejtjelezés módszereivel való – védelmét látja el úgy, hogy az információt valamilyen adott módon „összezavarja" (scrambling).

Az előzőekben érintett, adott célra orientált elemek mellett jelentős szerepet kapnak a jövőben az UPC-k (Universal Peripheral Controller: univerzális perifériavezérlő). Ezek az adott célra programozható egytokos mikroszámítógépek (pl. Mostek 3870, Motorola 6809, Zilog Z8, Intel 8041 stb.) elsősorban a kis darabszámban gyártott, egyedi alkalmazások speciális perifériavezérlőinek korszerű megoldási módját adják.

A kiegészítő elemek, amellett, hogy az elembázis leggyorsabban fejlődő területét képezik, a mindenkori technológia alapján megvalósított legkomplexebb (legnagyobb lapkaméret, legtöbb elemű áramkör egy lapkán, a nem ismétlődő struktúrájú elemeket figyelembe véve) LSI és VLSI áramköröket is jelentik.

IRODALOM

[1] *Moore, G.:* VLSI: some fundamental challanges. IEEE Spectrum, April 1979. 30–37.

[2] *Queyssac, D.:* Projecting VLSI's impact on microprocessors. IEEE Spectrum, April 1979. 38–41.

[3] *Allan, R.:* VLSI: scoping its future. IEEE Spectrum, April 1979. 30–37.

[4] *Sugarman, R.:* Computers: our „microuniverse" expands. IEEE Spectrum, January 1979. 32–37.

[5] *Schmid, H.:* A framework for top-down, technology-independent monolithic processor standardization. Proceeding of μP'79, OMKDK Technoinform, Budapest 1979. 9–18.

[6] An introduction to the Z bus. Zilog Corp., 1978.

[7] *Nicond, J. D.:* MUBUS Standard. Microscope, April 1977. 1–10.

[8] Microbus (National Semiconductor Standard for microprocessor interfaces). National Semiconductor Corp., Oct. 1977.

[9] *Rolander, T.:* Intel Multibus interfacing. Application Note AP–28, Intel Corp., 1977.

[10] *Kells, A. E. etal.:* Standard specification for S–100 bus interface devices. Computer, July 1979. 28–52.

[11] *Yencharis, L.:* Multibus and S–100-ready for standardization. Electronic Design, Sept. 27, 1979. 30.

[12] *Brooks, F. P.:* An overview of microcomputer architecture and software. Proc. EUROMICRO 1976., North Holland Publ. Co. Amsterdam, 1976. 1–6.

[13] *Ravenel, B. W.:* Toward a Pascal standard. Computer, April 1979. 68–82.

[14] *Marshall, M.:* Assembly language plan raises dust. Electronics, January 17, 1980. 98–100.

[15] *Myers, W.:* The need for software engineering. Computer, February 1978. 12–26.

[16] *Dahl, O. J.–Dijkstra, E. W.–Hoare, C. A. R.:* Structured programming. Academic Press, London and New York, 1972.

[17] *Peuto, B. L.–Shustek, L. J.:* Current issues in the architecture of microproccessors. Computer, February 1977. 20–25.

[18] *Shima, M.:* Two versions of 16-bit chip span microprocessor, minicomputer needs. Electronics, December 21, 1978. 81–88.

[19] Z8000 CPU (Instruction Set). Zilog, February 1979.

[20] *Stritter, E. and Gunter, T.:* A microprocessor architecture for a changing world: the Motorola 68000. Computer, February 1979. 43–52.

[21] *Stritter, S.–Tredennick, N.:* Microprogrammed implementation of a single chip microprocessor. Proc. MICRO–11. 1978. 8–16.

[22] Memory data book and designers guide. Mostek, 1979.

[23] μP/μC chip directory. EDN, October 20, 1979. 133–240.

[24] *McKenzie, K.:* The evolution of peripheral devices in microprocessor systems. Computer, June 1978. 12–16.

[25] *Cushman, R. M.:* Support chips gain in depth but no surprises appear. EDN, October 20, 1979. 243–254.

[26] *Posa, J. G.:* Peripheral chips shift microprocessor systems into high gear. Electronics, August 16, 1979. 93–106.

[27] *Arnold, W. F.:* Now for the math-processing chips. Electronics, July 5, 1979. 99–100.

[28] *Cole, E. W.:* A data-comm interface chip, that really unloads your CPU-even in BISYNC. Electronic Design, April 12, 1979. 100–104.

2. PROGRAMOZHATÓ ÉS PROGRAMOZOTT LOGIKA

Kis és közepes „bonyolultságú" feladatok megoldásához sokszor a mikroprocesszoros rendszer túlméretezettnek bizonyul. Ha nem akarunk „ágyúval verébre lőni", és a felhasználói (berendezésorientált) áramkörrel való megoldás feltételei sincsenek meg, a kombinációs és sorrendi hálózatok alkalmazása jelenti a gazdaságos megoldást. Ezeket LSI, VLSI technológia alkalmazásával programozható vagy programozott logikai rendszerek segítségével valósíthatjuk meg.

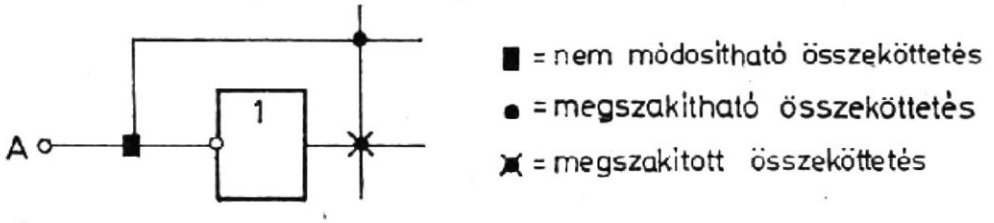
A programozható és programozott logikai kifejezéseket két – alapvetően különböző – megoldási mód jellemzésére használják. A különböző felhasználó által programozható logikai áramkörcsaládok alkalmazása esetén a hagyományos „huzalozott logika" módszereit használhatjuk. A logikai elemcsaládok gazdaságosabb (kisebb tokszámmal való) megvalósítást tesznek lehetővé, mint a hagyományos SSI és MSI logikai elemcsaládok, amelyek a feladathoz nem képesek alkalmazkodni. A programozás ebben az esetben azt jelenti, hogy a logikai elem felépítését – összeköttetések „kiégetésével" – a felhasználói rendszer kívánalmaihoz alakítjuk.

A *programozott logika* kifejezés már a mikroprocesszoros rendszerekhez kapcsolódik. A hagyományos logikai rendszereket ui. sokszor mikroprocesszoros rendszerekkel helyettesítjük. A helyettesítést vagy a logikai elemekre, vagy az egész feladatra, ill. annak egy nagyobb részére végezhetjük el. Mindkét esetben a megoldási módot szokás programozott logikának nevezni.

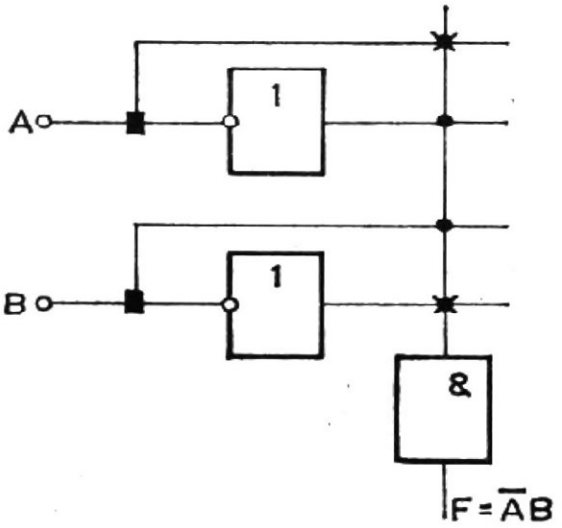
2.1. PROGRAMOZHATÓ LOGIKAI ÁRAMKÖRÖK

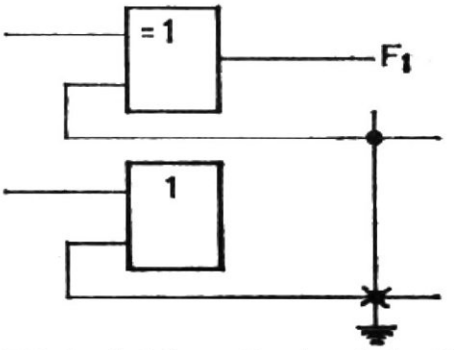
A kombinációs és sorrendi hálózatok megvalósításához egyre több logikai áramkörcsalád áll rendelkezésre. Ezek közül mutatunk be néhány fontosabbat, alkalmazási példán keresztül illusztrálva fő jellemzőiket.

Az FPLF (Field Programmable Logic Family: felhasználó által programozható logikai elemcsalád) áramköri rajzain az összeköttetésekhez a 2.1. ábra szerint a következő három jelölést fogjuk használni: a nem módosítható, a megszakítható (érintetlen, de „kiégethető") és a megszakított (már „kiégetett") összeköttetés ábrázolására. A sokbemenetű – a gyakorlatban megvalósított áramköröknél max. 32 – kapuáramkörök egyszerűsített jelölésére a 2.2. ábra szerinti elrendezést fogjuk alkalmazni. A kapuáramkör lehetséges bemeneteinek számát tehát az egyetlen vonallal jelölt



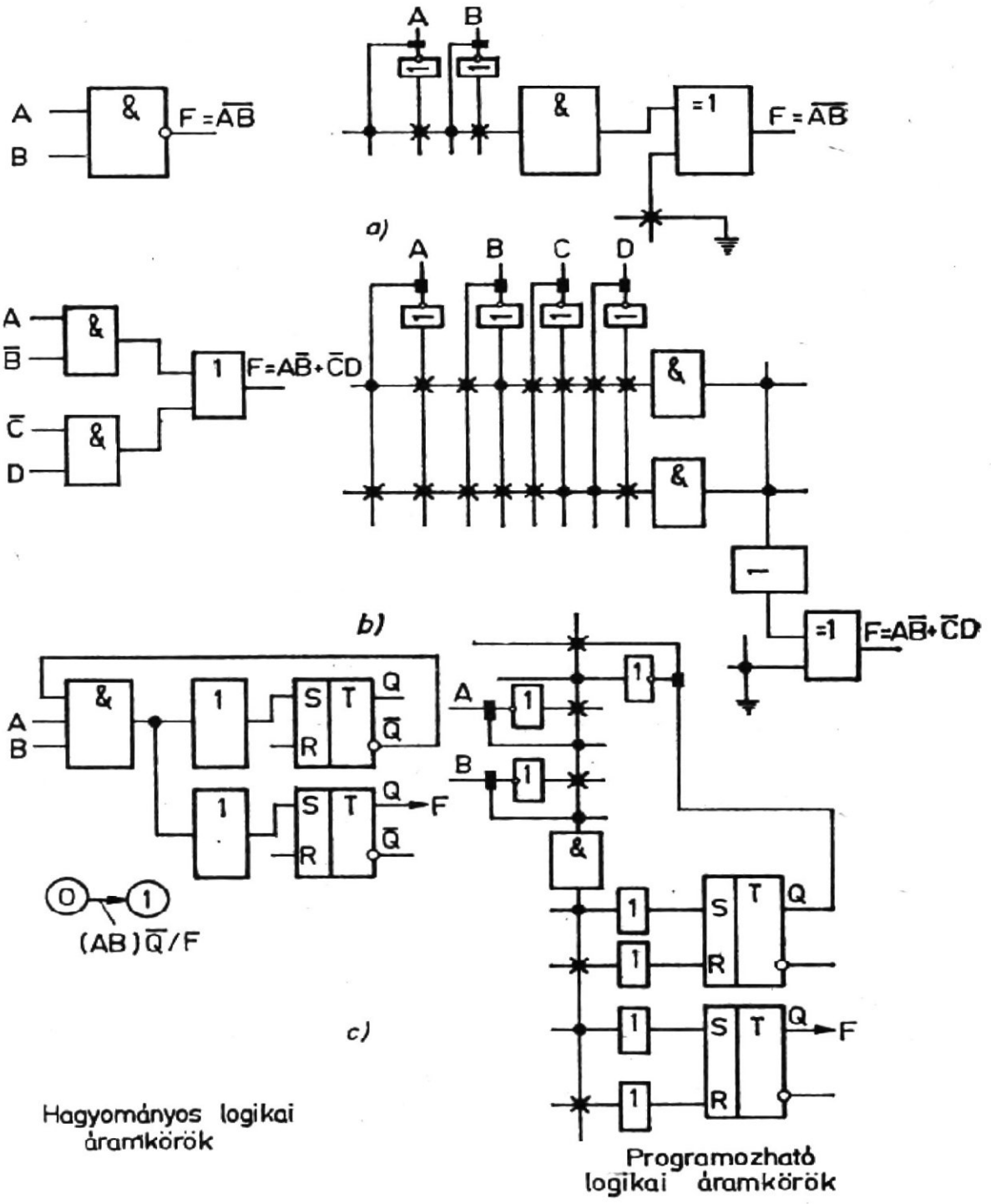
2.1. ÁBRA. Programozható logikai áramkörök összeköttetéseinek jelölése





2.3. ÁBRA. A kimenő adatok beállítása VAGY kapuk segítségével

◀ 2.2. ÁBRA. Kapuáramkörök egyszerűsített jelölése

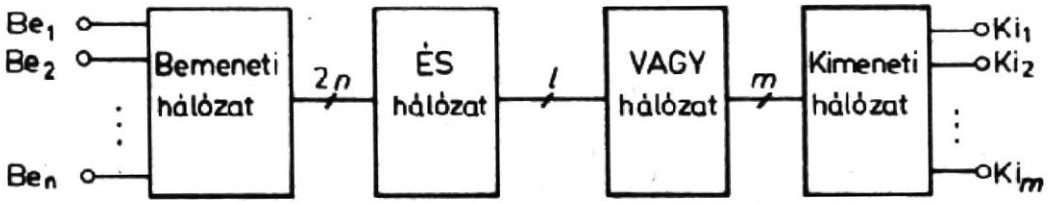


2.4. ÁBRA. A hagyományos és programozható logikai áramkörök közötti kapcsolat *a)* egyszintű, *b)* kétszintű kombinációs hálózat, *c)* sorrendi hálózat részlete

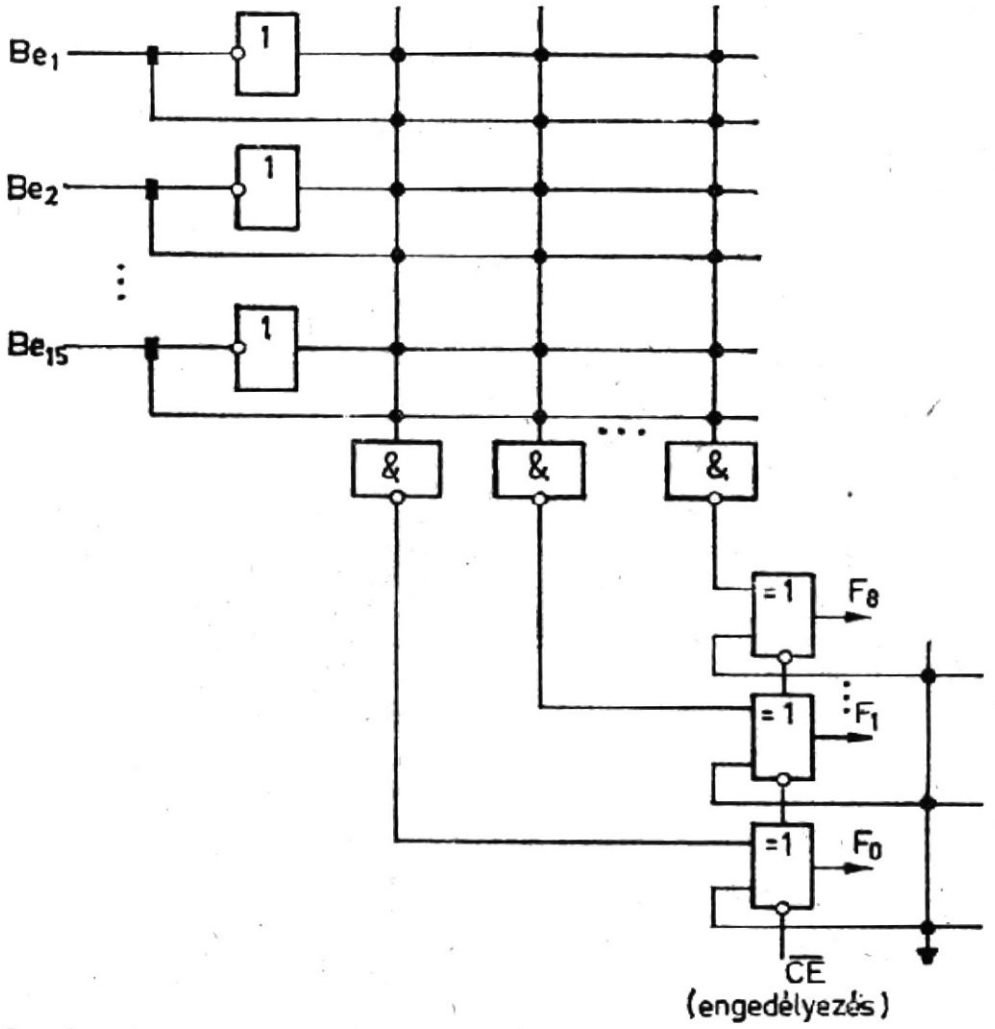
bemenetet keresztező vonalak száma mutatja. (Az ábra szerinti ÉS kapu tehát négy bemenetű, és a megadott módon programozott változata az $F = \bar{A}B$ logikai egyenletet valósítja meg.) Az egyes változók és azok komplemensei is lehetnek bemenő változók. A kapu tényleges bemeneteinek számát végül is az érintetlen összeköttetések száma fogja megadni.

A kizáró VAGY kapuk programozásával az elemek kimenetén az aktív állapotnak akár a magas, akár az alacsony logikai jelszint megfeleltethető. A 2.3. ábra szerint a felső kizáró VAGY kapu egyik bemenetének földelése aktív magas logikai kimeneti jelszintet, míg az alsó VAGY kapuelrendezés esetén a „biztosíték" kiégetés aktív alacsony logikai jelszintet eredményez. A De Morgan szabály alkalmazásával így ÉS, VAGY, NEM ÉS és NEM VAGY logikai függvények valósíthatók meg [5].

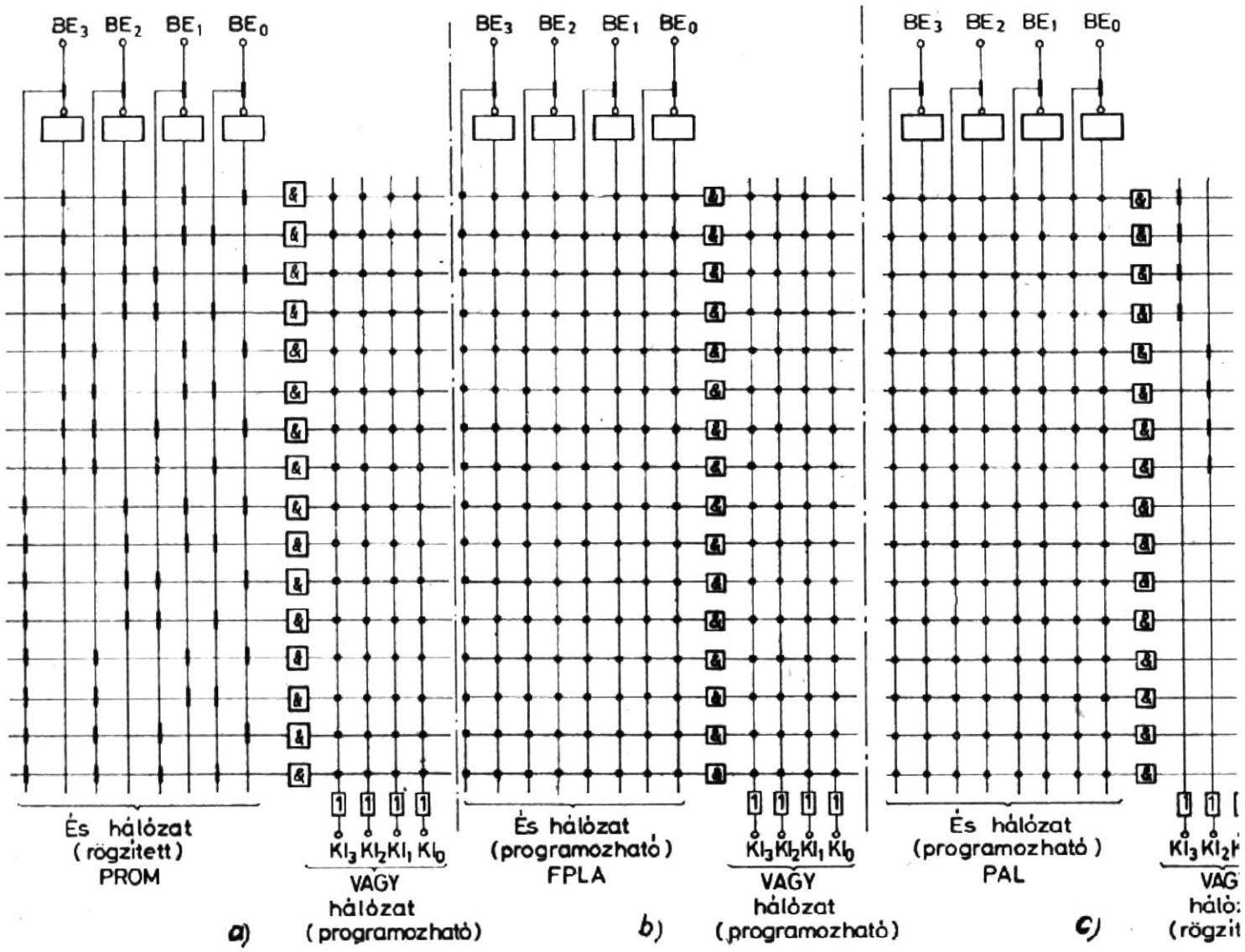
A hagyományos és a programozható logika közötti kapcsolatot legjobban talán példákon keresztül mutathatjuk be. A 2.4. *a* ábrán lévő példa egyszintű, a *b* egy kétszintű kombinációs hálózat, a *c* pedig egy sorrendi hálózat részlete hagyományos és programozható logikai áramkörök segítségével megvalósítva.



2.5. ÁBRA. Programozható kombinációs logikai elem általános felépítése



2.6. ÁBRA. FPGA felhasználó által programozható kapuáramkörelrendezés



2.7. ÁBRA. Kétszintű ÉS–VAGY kombinációs logikai függvények megvalósításának programozható eszközei *a)* PROM, *b)* FPLA, *c)* PAL

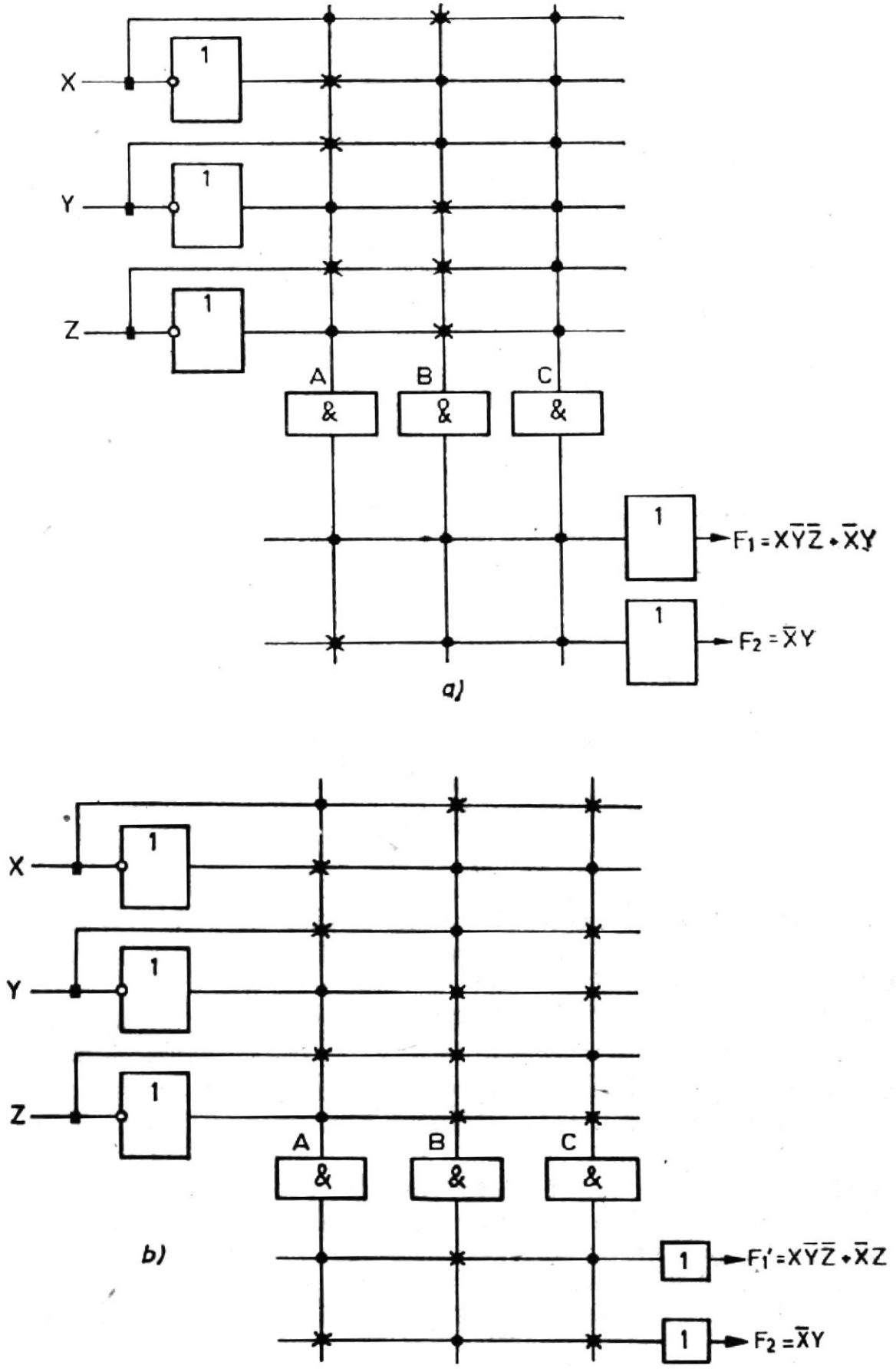
A programozható logikai áramkörök csoportosíthatók. Az első csoporttal kombinációs logikai hálózatok valósíthatók meg. Az elemek általános felépítését a 2.5. ábrán mutatjuk be.

A bemeneti logikai hálózat a bemenő változók komplementálását, míg a kimeneti a 2.3. ábra szerinti – programozható – polaritásváltoztatást valósítja meg.

A legegyszerűbb FPGA (Field Programmable Gate Array: felhasználó által programozható kapuelrendezés) elemek egyszintű kombinációs hálózatot valósítanak meg (2.6. ábra).

A kétszintű (ÉS-VAGY) kombinációs logikai függvények megvalósításának programozható eszközeit a 2.7. ábrán mutatjuk be. A PROM esetén csak a VAGY logikai hálózat, a PAL (Programmable Array Logic: programozható logikai elrendezés) esetén csak az ÉS logikai hálózat, míg az FPLA (Field Programmable Logic Array: felhasználó által programozható logikai elrendezés) esetén az ÉS és a VAGY hálózat egyaránt programozható [1]...[4].

Az FPLA áramkörök, bár az alkalmazott technológia miatt csak egyszer programozhatók, megfelelő mennyiségű tartalék áramkör esetén, bizonyos részek kiiktatásával és helyettesítésével módosíthatók. Erre láthatunk példát a 2.8. ábrán, ahol



2.8. ÁBRA. FPLA áramkör módosítása *a)* eredeti logikai függvények, *b)* módosított logikai függvények

$F_1 = X\overline{Y}\overline{Z} + \overline{X}Y$ és

$F_2 = \overline{X}Y$

az eredeti függvényeket a tervezési feltételek megváltozása miatt az

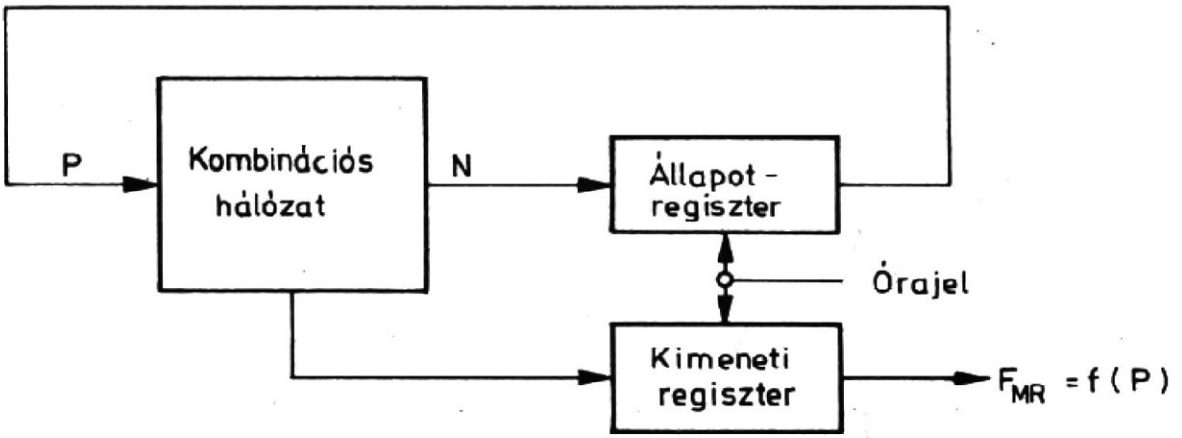
$F_1' = \overline{X}\overline{Y}\overline{Z} + \overline{X}Z,$

$F_2 = \overline{X}Y$

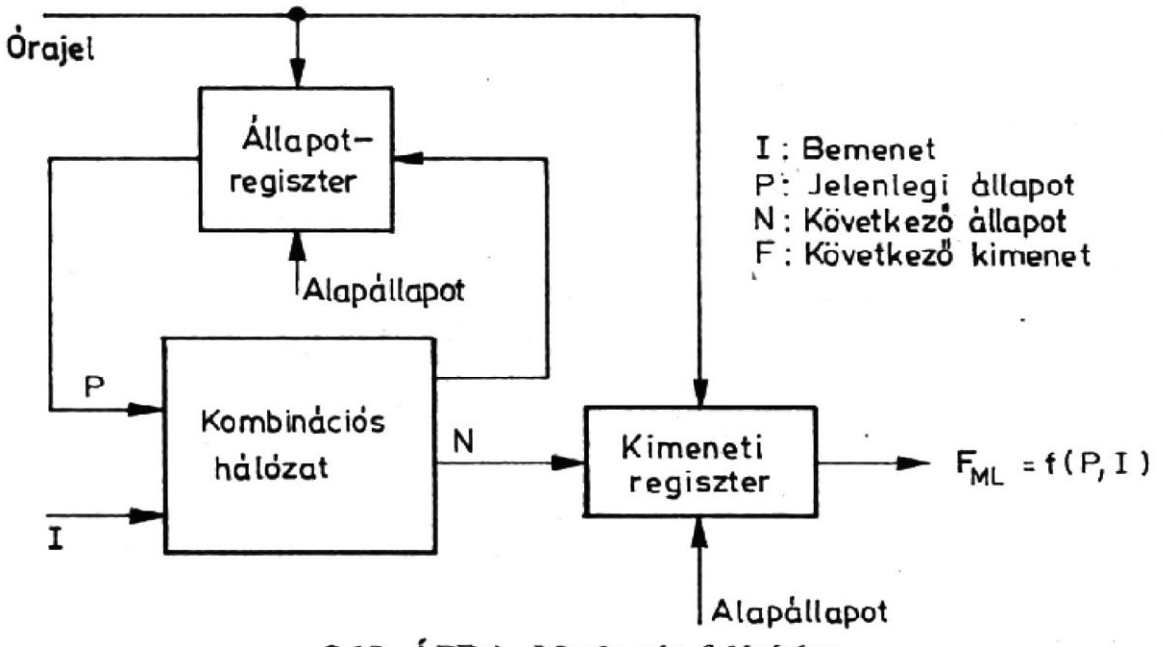
függvényekké alakítottuk át.

Az egyetlen tokban megvalósított ÉS kapuk száma nem jelent korlátozást az alkalmazásnál, mivel az áramkörök nyitott kollektoros változataival – a kimenetek „huzalozott" ÉS kapcsolatát képezve – azok könnyen bővíthetők. (Az egyetlen megkötés, hogy ebben az esetben a bővítésben szereplő kimeneteket aktív állapotban alacsony lotikai szintnek megfelelően kell programoznunk.)

A programozható logikai elemcsaládok legfejlettebb elemei az FPLS- (Field-Programmable Logic Sequencer: felhasználó által programozható logikai sorrendképző) elemek. Ezek regiszterekkel kiegészített FPLA- vagy PAL-áramkörökkel felépítettek, amelyek olyan módon programozhatók, hogy segítségükkel szinkron sorrendi



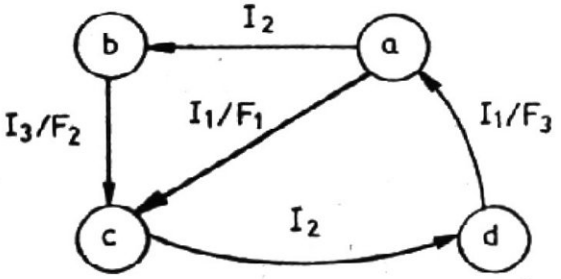
2.9. ÁBRA. Moore-gép felépítése



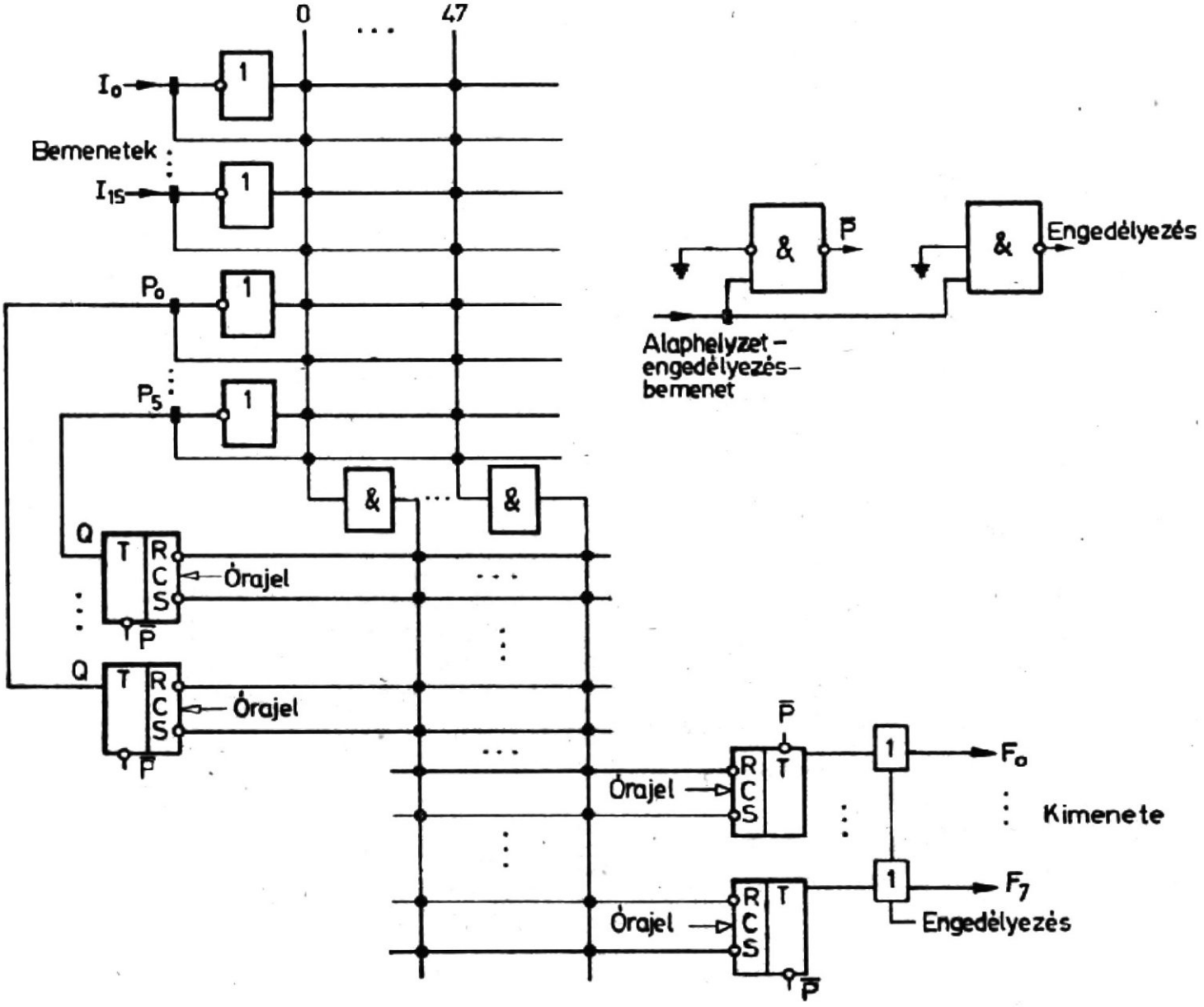
2.10. ÁBRA. Mealy-gép felépítése

hálózatokat valósíthatunk meg. Két alapformája ismeretes: a 2.9. ábrán látható Moore-gép, amelynek a $F_{MR} = f(P)$ kimenete csak a P jelenlegi állapotfüggvénye. A 2.10. ábra szerinti Mealy-gép esetén az $F_{ML} = f(P,I)$ kimenet mind a P jelenlegi állapot, mind az I aktuális bemenő változó értékeinek függvénye [1].

Az FPLS-áramkör felépítése egy Mealy-gép felépítésének felel meg. Az FPLS alkalmazásával minden olyan feladat megoldható, amely állapotátmenetek sorozatával fejezhető ki. Az állapotátmenetek a rendszert egyik stabil állapotából a másikba viszik, és ezeket az órajel által t időpontban vett bemenőváltozó-értékek ha-



2.11. ÁBRA. Egy állapotdiagram



2.12. ÁBRA. FPLS áramkör logikai felépítése

tározzák meg. Ebben a sorozatban az állapotok száma a megvalósított algoritmus összetettségétől és hosszától függ.

Egy tipikus állapotdiagramot mutatunk be a 2.11. ábrán. Azt az állapotot, amelyből indulunk, P jelenlegi állapotnak és ahova jutunk, az állapotátmenet után Q következő állapotnak nevezzük. Egy állapotátmenet mindig állapotváltozást okoz, de nem feltétlenül változtatja meg az F kimeneti függvényt.

Minden állapotot valamilyen módon megjelölünk és tárolunk az állapotregiszterben. Ennek bemenetére a kombinációs logikai hálózatból származó következő állapotinformáció és az órajel kerül. Állapotátmenet csak akkor következik be, amikor a változást meghatározó állapot IGAZ. Ez definíció szerint az ÉS kapcsolata az órajelnek, a P jelenlegi és az I bemenetek állapotainak, vagyis

$$T_n = t \cdot I \cdot P.$$

Mivel azonban valójában az órajelet az állapotregiszterre adjuk, ez elhagyható az egyenletből. Amikor T_n IGAZ, egy olyan vezérlőjel jön létre, amely a t órajel pillanatában az állapotregiszter tartalmát P-ről Q-ra változtatja, és ha szükséges, megváltoztatja a kimeneti regiszter tartalmát is. A 2.11. ábra állapotdiagramján lévő példában négy állapot (a, b, c és d) van. I_1–I_3 az állapotváltozás feltételei, amelyek F_1–F_3 kimeneten állapotváltozásokat is létrehoznak.

Tekintsük a 2.12. ábrán az FPLS ekvivalens logikai diagramját. Felépítése az FPLA elrendezésen alapul, amelyet regiszterekkel egészítettek ki. A regiszterek egyik csoportja kimeneti regiszterként működik, míg a másik csoport a kimenetek állapotától függetlenül programozható állapotregiszter. Az ÉS-VAGY hálózat az adott regiszterek R vagy S bemeneteit vezérli, míg az $I_0 \ldots I_{15}$ bemenő állapotváltozók (ill. azok komplemensei) mellett a megelőző órajelnél beállított állapot bistabilok ($P_0 \ldots P_5$) is – programozható módon – a 48 ÉS kapu bemenetére csatlakoztathatók. Minden belső regiszter a tápegység bekapcsolásakor logikai *1* állapotba kerül. Ezenkívül van egy külön bemenet is aszinkron módon való *1*-es állapotba állításra. Ugyanez a bemenet használható az $F_1 \ldots F_7$ kimenő jelek engedélyezésére is. (A nem engedélyezett kimenet nagy impedanciájú állapotba kerül, ezzel biztosítva a párhuzamos kötéssel való bővítést is.)

A $P_0 \ldots P_5$ visszacsatolt állapot bistabilok tartalma és az $I_0 \ldots I_{15}$ bemenő állapotváltozók ÉS kapcsolatba hozhatók egymással, ami megfelel az állapotdiagramban az egyik állapotból a következő állapotba való átvitel feltételének.

Ha pl. ez a feltétel:

$$T = Q \cdot (\overline{X} + \overline{Y} + \overline{Z})$$

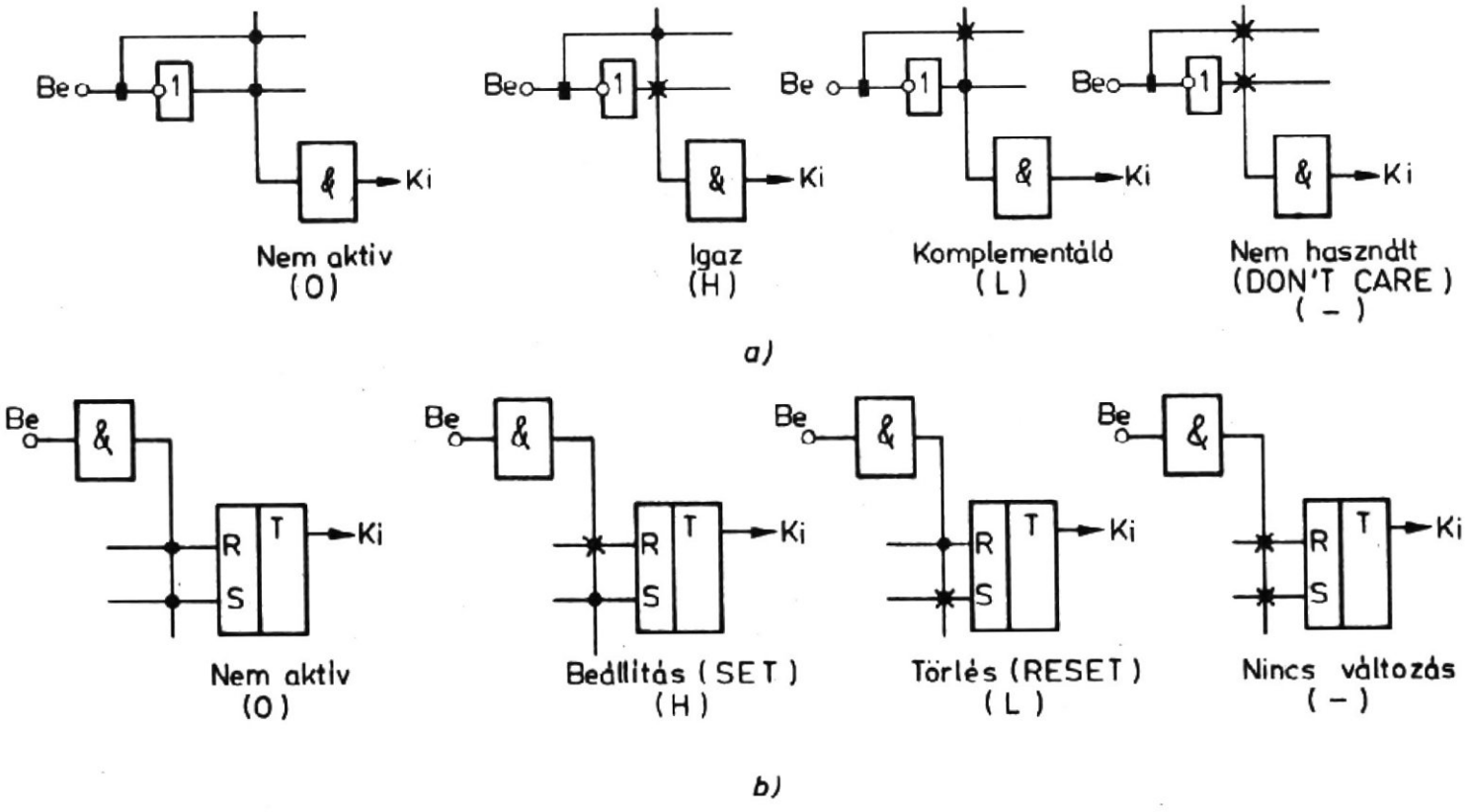
ahol Q az állapotregiszter kimenete és az X, az Y és a Z bemenetek, a FPLS három ÉS kapujára van szükség a megvalósításhoz. Ha azonban a De Morgan szabály [5] segítségével átalakítjuk a kifejezést a

$$T = Q \cdot \overline{XYZ}$$

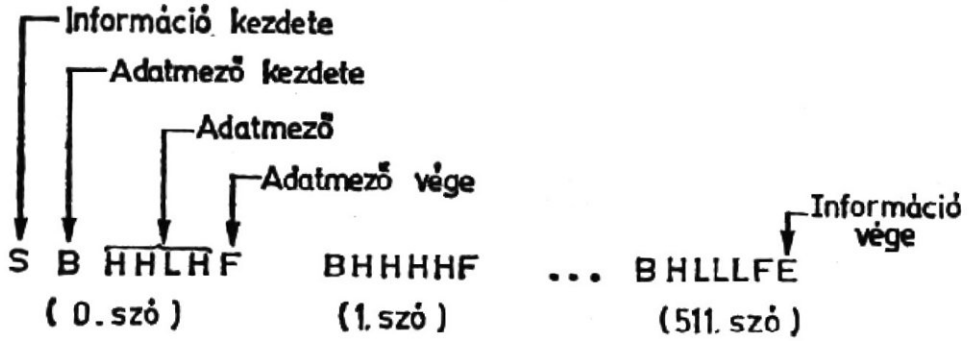
formára, akkor csupán két ÉS kapura lesz szükségünk.

A programozható logikai elemek alkalmazásához – a megszakított és a nem megszakított összeköttetések kombinációinak figyelembevételével – a 2.13. ábra szerinti jelöléseket szokták használni. (A 2.13. *a* ábrán példaként egy ÉS kaput, míg a *b* ábrán egy VAGY kapuk által vezérelt bistabilt tüntettünk fel.)

Az elemekbe való „beírás" gyakorlati megvalósításához eszközökre is szükség



2.13. ÁBRA. Programozható logikai elemek jelölései *a)* ÉS kapu, *b)* flip-flop

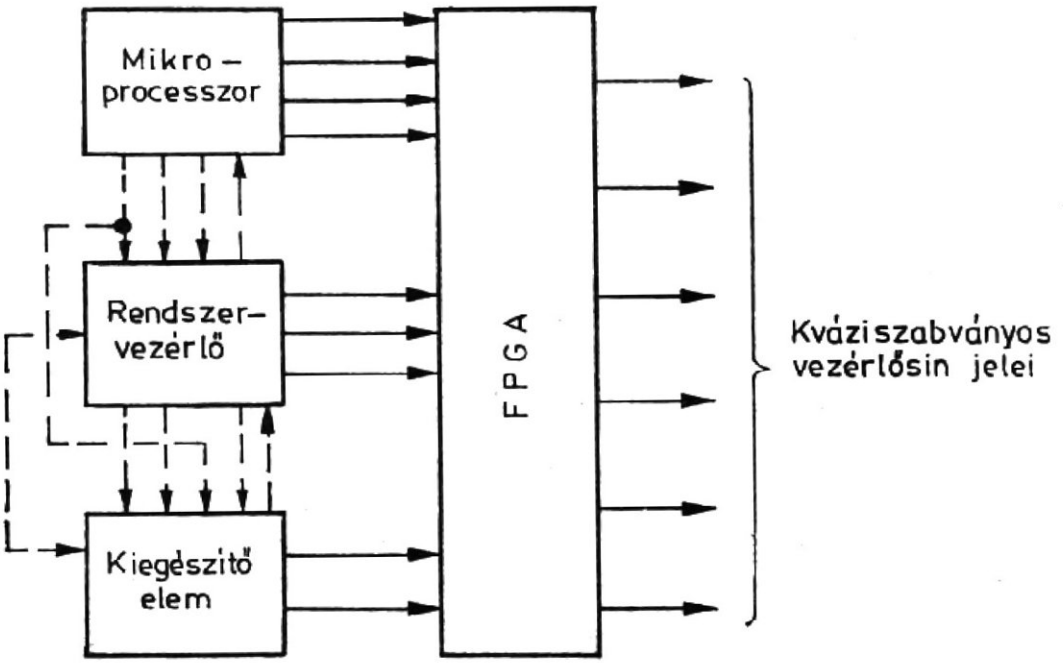


2.14. ÁBRA. BHLF formátum

van. Ezek pl. igazságtáblázat formájában megadott specifikációt és a megvalósításhoz felhasznált elemek típusszámait a konkrét programozó eszköz bemenő formátumára – és az egyes be-, ill. kimenetekhez: tokkivezetésekhez rendelt változónevek felhasználásával – fordítják le. Pl. a PALASM egy olyan PAL-elemekhez kifejlesztett assembler, amely akár logikai egyenleteket, akár táblázatokat elfogad bemenő információként, és ezeket a „beégetést" vezérlő formátumra fordítja le [2].

A leggyakrabban használt formátum a BHLF vagy BPNF (2.14. ábra). A két formátum lényegében megegyező, mindkettő ASCII karaktereket használ. Az egyes karakterek jelentése a következő: S információ (szalag) kezdete; B adatmező kezdete; F adatmező vége; E információ (szalag) vége; H vagy P magas logikai jelszintnek megfelelő adatbit; L vagy N alacsony logikai jelszintnek megfelelő adatbit.

A hexadecimális formátum alkalmazása esetén az adatok kezdetét az STX karakter jelzi, amelyet az egyes – sorrendbe rendezett és szóközzel (space) elválasztott – hexadecimális karakterrel megadott adatok követnek. Az információ végét az ETX karakter jelzi. A kinyomtatás megvalósításának elősegítésére általában bárhol közbeiktatható új sor és kocsi vissza karakter.



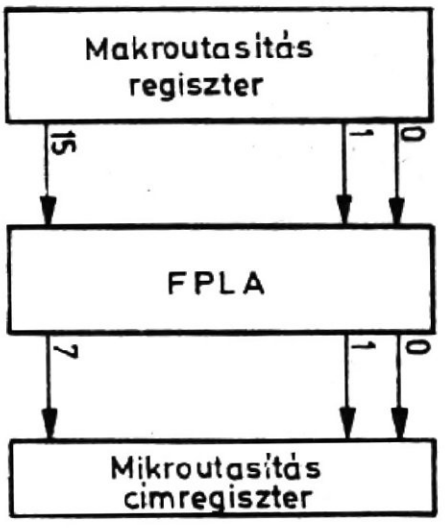
2.15. ÁBRA. Sínvezérlő jelek átalakítása FPGA alkalmazásával

A programozó eszközök egy további csoportja – az ún. logikai programozók – a logikai függvények táblázatai alapján közvetlenül képesek a beírást elvégezni. (Egyes gyártóktól ún. hibrid programozók is beszerezhetők, amelyek tár és logikai típusú programozást egyaránt képesek megvalósítani.)

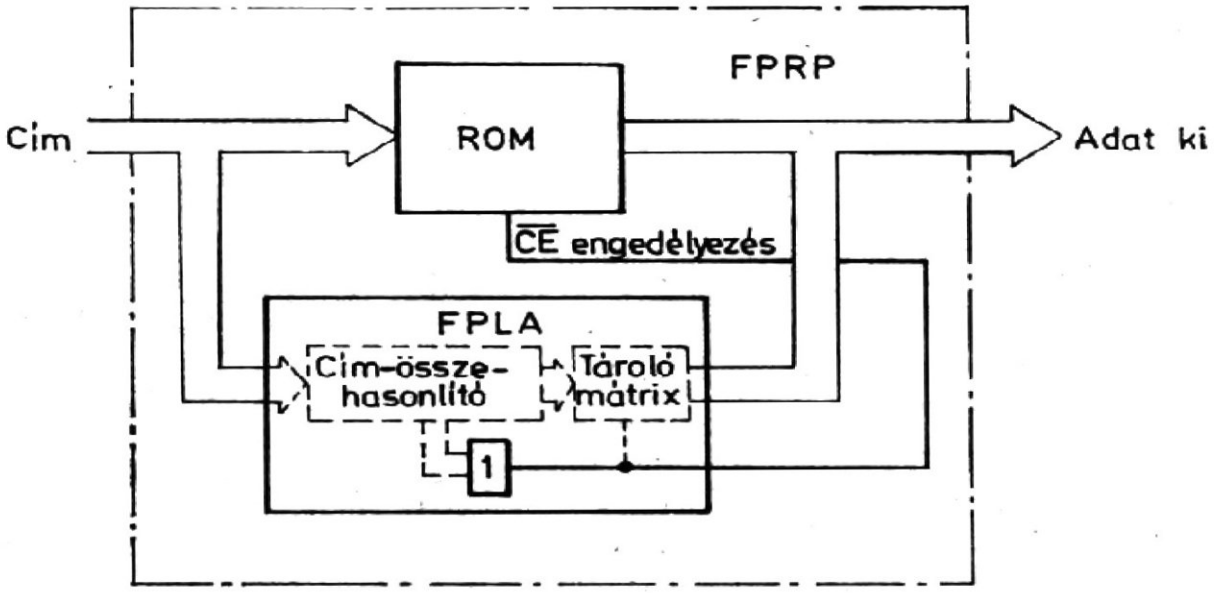
A programozható logika elemei – mikroprocesszoros rendszerekben – tipikus kiegészítő feladatok megoldására is alkalmazhatók. Ezek közül mutatunk be most néhányat.

Tipikus feladat egy adott mikroprocesszor-elemcsalád elemeivel megvalósított rendszer „illesztése" valamilyen adott sinrendszerhez. Az elemek, ill. sinrendszerek szabványosításának jelenlegi fázisában ez egy nagyon gyakran előforduló feladat. Általában egyetlen FPGA segítségével a 2.15. ábra szerint a mikroprocesszor és a rendszerelemek vezérlőjeleinek a kombinációjából kialakíthatjuk az adott sinrendszer pl. az S–100 sin [4] [6], vezérlőjeleit.

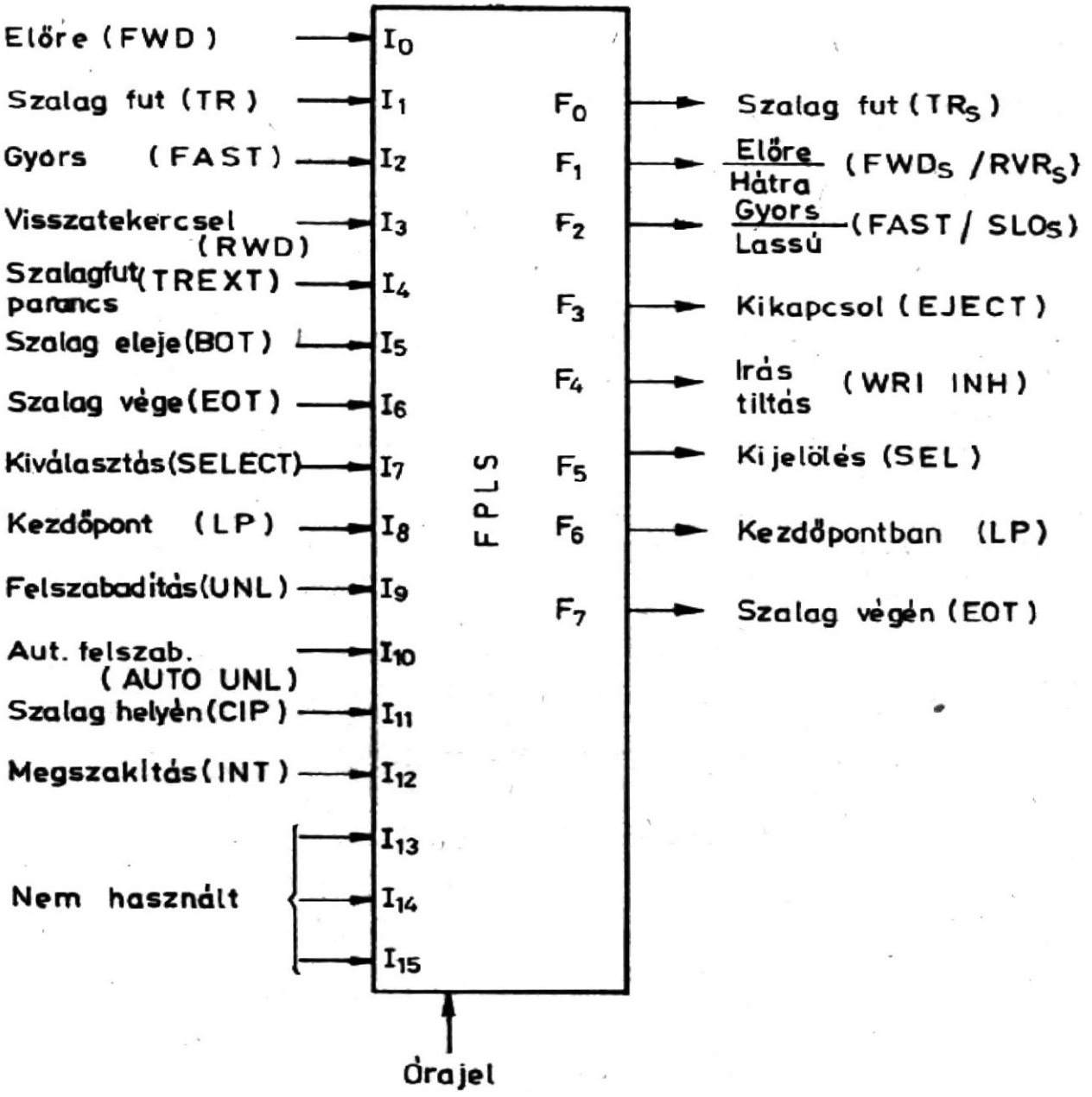
Az FPLA elemek tekinthetők cím – és természetesen tartalom szerint is – programozható ROM-oknak. Ezzel az értelmezéssel – ha nem teljes (n bitszám esetén 2^n) – kódhalmazt kell átalakítanunk, gazdaságos dekódoló vagy átkódoló valósítható



2.16. ÁBRA. Utasításdekódolás FPLA segítségével



2.17. ÁBRA. ROM „megfoltozása" FPLA segítségével



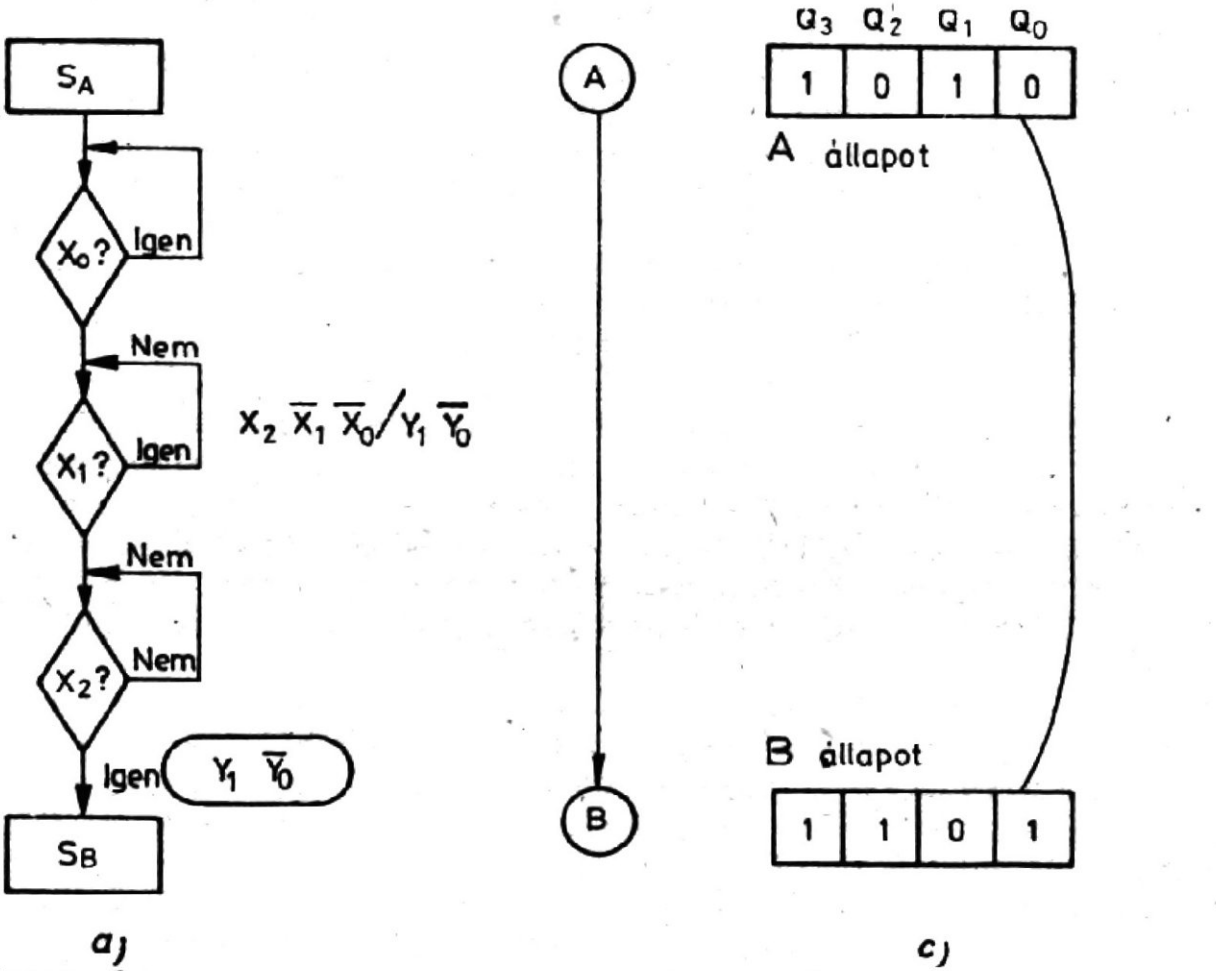
2.18. ÁBRA. Perifériavezérlő megvalósítása FPLS alkalmazásával

meg segítségükkel. Ilyen feladattal találkozunk pl. minden olyan mikrovezérelt rendszerben (l. a 11. fejezetet), ahol az általában sokbites gépi utasítás kódszavak lehetséges halmazának csak egy kis részhalmaza van értelmezve és ezekhez kell – az utasításértelmezés folyamatának részeként – a végrehajtást végző mikroprogramrész kezdőcímét hozzárendelni (2.16. ábra).

Hasonló feladatot jelent adott tartalmú, nem módosítható ROM-elemek „megfoltozása". Ez egy adott címtartományhoz tartozó tartalom helyettesítéssel való megváltoztatását jelenti, nem elsősorban a hibás, nagykapacitású ROM-elemek megmentését célozza. A fő cél „előregyártott" rendszer vagy alkalmazói programok adott feladat kívánalmainak megfelelő módosítása, ill. az általános célú megoldásoknak a specifikus alkalmazás paraméterei szerinti optimalizálása.

Egy FPRP (Field Programmable Rom Patch: felhasználó által programozható ROM „folt") elrendezést mutatunk be a 2.17. ábrán. Az ábrán láthatjuk, hogy az FPLA folyamatosan figyeli a címvezetékeket, és amikor azok a „beprogramozott" címértékekkel egyeznek meg, a FLAG kimenet jelzőbitjének segítségével tiltja a ROM kimenő jeleit, és helyette az adatvezetékekre a megfelelő rekeszeinek a tartalmát adja. (Természetesen „kikapcsolt" állapotban mind a ROM-, mind az FPLA-áramkör nagy impedanciájú, harmadik logikai állapotban van.) Az FPLA kimenetek párhuzamos kapcsolásával (huzalozott VAGY kialakításával), ha egyetlen FPRP nem elegendő, a kapacitást a szükségletnek megfelelően bővíthetjük.

Az FPLS elemek segítségével pl. viszonylag bonyolult perifériavezérlőket is megvalósíthatunk egy vagy néhány tokkal. A 2.18. ábra szerinti FPLS-nek pl. a bemeneteire parancsot adunk, és akkor az ezeknek megfelelő összes olyan vezérlőjelet előállítja, amely egy mágnesszalagos egység szalagtovábbító rendszerének működéséhez szükséges.



2.19. ÁBRA. Állapotregiszter *a)* folyamatábrája, *b)* állapotdiagramja és *c)* használata

Tervezéskor az állapotdiagram mellett folyamatábrát is használhatunk, amelyen az adott folyamatot – a bemenő változókat, állapotokat és kimenő függvényeket megfelelő változónevekkel jelölve – jobban nyomon követhetjük. A 2.19. ábrán egy állapotdiagram részleteként az A állapotból B állapotba való feltételektől függő átmenetet tüntettünk fel. A megfelelő folyamatábrán az összes feltételnek egyidejűleg kell ahhoz teljesülnie, hogy az állapotátmenet bekövetkezhessen. Legyen négybites állapotregiszterünk (Q_3, Q_2, Q_1, Q_0), három bemenő (X_2, X_1, X_0) és két kimenő (Y_1, Y_0) változónk. Ennek megfelelően az A-ból B-be való átmenetet a $10(1010_2)$ állapotból az $13(1101_2)$ állapotba való ugrás jelöli. A kimenet $2(10_2)$-ba megy át a következő órajelnél, ha a bemeneti kombináció $X_n = 4(100_2)$ értéke IGAZ.

Az állapot- és a kimeneti regiszterek átírása így a következő órajelnél a T átmeneti feltétel hatására következik be, ahol

$$T = P_3\overline{P}_2P_1\overline{P}_0I_2\overline{I}_1\overline{I}_0.$$

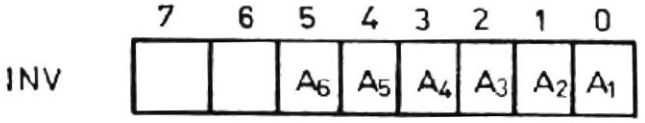
A T átmeneti feltételek jeleit a megfelelő regiszterek S beállító és R törlő vezetékeire kell kötnünk. (Az állapotregiszternél S_2, R_1 és S_0, a kimeneti regiszternél S_1 és R_0.)

2.2. PROGRAMOZOTT LOGIKA

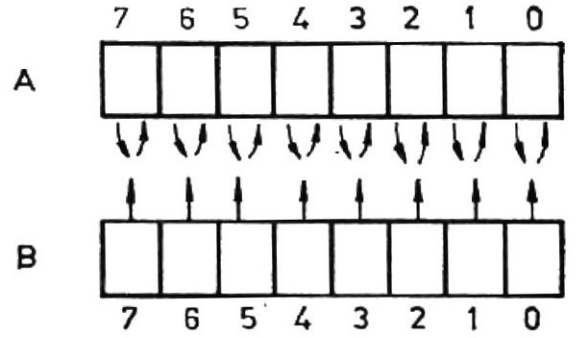
A digitális tervező, aki megszokta, hogy a feladatokat digitális logikai áramkörök segítségével oldja meg, analógiát találhat a különböző digitális logikai áramkörök és azonos feladatot ellátó mikroprocesszor programrészek között. Így idődiagramokat helyettesíthetünk program-folyamatábrákkal, logikai szimbólumokat assembler nyelven írt programokkal. Bár a programozással ismerkedő mérnök így talán könnyebben közelebb kerülhet a mikroprocesszor-program segítségével való feladatmegoldás gondolatmenetéhez, ez az analógia azonban sok szempontból sántít. A legfontosabb különbség az, hogy a mikroprocesszor a program egyes utasításait egymás után, sorban hajtja végre. Ezzel szemben a digitális logikai áramkörök alkalmazásánál jól ismert módon a különböző logikai változók és állapotok párhuzamos ágakon haladnak és a „csomópontokban" kerülnek kiértékelésre. A kétféle megoldás időzítése is alapvetően eltérő. Itt nem csupán a működési sebesség különbségére gondolunk (a mikroprocesszor segítségével megvalósított programozott logika utasításaival végrehajtási sebességéből következően több nagyságrenddel kisebb sebesség érhető el, mint a korszerű digitális áramkörök alkalmazásával), hanem az időzítések (késleltetések, szinkronizálás) megvalósítására is. A gyakorlatban ezért szinte sohasem „teszünk át" egy digitális logikai áramkörökkel megvalósított rendszert mikroprocesszoros programozott logikává, hanem az alapfeladatot oldjuk meg programozott logika, vagyis a mikroprocesszoros rendszer és a hozzá tartozó program segítségével.

A digitális tervezők számára – a programozással való könnyebb megismerkedés céljából – bemutatunk néhány egyszerű logikai áramkört helyettesítő programkészletet [8]. Az utasításoknál az i8080 assembler jelöléseit fogjuk használni [9]. A konkrét mikroprocesszoros rendszer felépítésével itt most nem foglalkozunk, mivel csupán a fontosabb alaputasításokat fogjuk használni.

Első példánk egy hexinverter (hat inverter egy tokban) a 7404 típusszámú TTL áramkör. Feleljen meg egy-egy bit a INV szimbolikus címmel jelölt tárrekeszben egy-egy inverternek (2.20. ábra).



2.20. ÁBRA. Hex inverter helyettesítése az INV rekesz tartalmával



2.21. ÁBRA. Kétszer négy kétbemenetű ÉS kapu helyettesítése

A hexinvertert ezek után a következő utasítássorozattal jellemezhetjük:

```
LDA INV  ; AZ INV REKESZ TARTALMÁT AZ
         ; AKKUMULÁTORBA VISZI
CMA      ; KOMPLEMENTÁLJA AZ AKKU TARTALMÁT
STA INV  ; AZ AKKU TARTALMÁT AZ INV REKESZBE
         ; TÁROLJA
```

Készítsük el ezután négy kétbemenetű ÉS kaput tartalmazó tok (7408) funkcióját helyettesítő utasítássort. (Mivel az i8080 regiszterei nyolcbitesek, mindjárt két ilyen tokot „szimuláljunk".)

A kapuáramkörök egyik bemenetét ábrázolják az akkumulátor bitjei, a másik bemenetet egy másik regiszter (pl. B regiszter) bitjei, az eredményt pedig írjuk vissza az akkumulátor megfelelő bitjeibe (2.21. ábra). Az egyes bemenő adatok természetesen jöhetnek az erre a célra kijelölt tárrekeszekből (SRA és SRB) és eredményt is elhelyezhetjük egy külön tárrekeszbe (DST). Így az utasítássorozat a következő lesz:

```
LDA SRB   ; BEÍRJA AZ EGYIK BEMENETEKET A-BA
MOV B,A   ; ÁTÍRJA A TARTALMÁT B-BE
LDA SRA   ; BEÍRJA A MÁSIK BEMENETEKET A-BA
ANA B     ; AZ „A ÉS B" MŰVELETET ELVÉGZI
STA DST   ; AZ EREDMÉNYT DST REKESZBE ÍRJA
```

Tekintsünk most hárombemenetű ÉS kapukat (7411), és az egyes bemenő adatokat, ill. az eredményt tartalmazó rekeszeket jelöljük ki egymás után az adattárban. Ekkor a hárombemenetű ÉS kapukat a következő módon helyettesíthetjük programmal:

```
LXI H, SRA  ; AZ ELSŐ CÍMET HL REGISZTERPÁRBA ÍRJA
MOV A, M    ; AZ ELSŐ BEMENETEKET TARTALMAZÓ
            ; MEMÓRIA TARTALMÁT A-BA ÍRJA
INX H       ; MEGNÖVELI A CÍMET
ANA M       ; A ÉS M MŰVELETET ELVÉGZI
INX H       ; CÍMET NÖVEL
ANA M       ; AZ ELŐZŐ MŰVELET EREDMÉNYÉT
            ; ÉS HARMADIK BEMENETEK
            ; MŰVELETET VÉGEZ
INX H       ; CÍMET NÖVEL
MOV M, A    ; EREDMÉNYT TÁROLJA
```

Végezzük el ezután egy kettős D flip-flop (7474) programmal való helyettesítését. Ebben az esetben a megfelelő logikai érték beírását a flip-flop órajele helyett a meg-

felelő utasítások elvégzése fogja helyettesíteni. Az első flip-flopnak feleljen meg az FLP című rekesz 0-ás bitje, a másodiknak pedig az egyes számú bitje. Ekkor a flip-flopokba *0* logikai értéket beíró program a következő:

```
MVI 0       ; 0 ÉRTÉKET ÍR AZ AKKU-BA
STA FLP     ; TÁROLJA A NULLA ÉRTÉKET AZ FLP
            ; REKESZBEN
```

Az egyes flip-flopnak megfelelő bitbe logikai 1 értéket író program a következő:

```
LDA FLP     ; FLP TARTALMÁT AKKU-BA ÍRJA
ORI 1       ; HOZZÁKAPUZZA AZ 1 ÉRTÉKET
ANI 0FDH    ; MASZKOL
STA FLP     ; VISSZATÁROLJA AZ AKKU ÉRTÉKÉT
```

Monostabil multivibrátort (74121), ill. annak megfelelő késleltetést adott időtartamú utasítások szükséges számban való végrehajtásával valósíthatunk meg. Az időzítéseket általában „programhurok" (adott számban végrehajtott utasítássor) segítségével valósítjuk meg. A hurokban végrehajtott műveleteket vagy egyetlen 8-bites tárrekesz (rövid időtartamok esetén), vagy két tárrekesz, azaz 16 bit (hosszabb időtartamok esetén) segítségével számoljuk. Mindkét esetben a késleltetést végző programot írjuk meg egy-egy szubrutin formájában. Ekkor az SDELY rövid idejű késleltetés szubrutinja a következő (2.22. ábra):

```
SDELY MVI A,TIME  ; IDŐÁLLANDÓT BEÍRJA AZ AKKUBA
LOOP  DCR A       ; AKKU TARTALMÁT CSÖKKENTI
      JNZ LOOP    ; HA NEM ZÉRUS AZ AKKU
                  ; TARTALMA VISSZAUGRIK
      RET         ; VISSZATÉR
```

Figyelembe véve, hogy a **DCR A** utasítás öt órajel-intervallum alatt, a **JNZ LOOP** utasítás pedig tíz órajel intervallum alatt hajtódik végre (l. az i8080 utasítástáblázatot [8]) – 2 MHz órafrekvenciát (azaz 500 ns órajel-intervallumot) feltételezve – a ciklus „magját" képező ezen két utasítás végrehajtásának ideje 15 ciklusidő = = 7,5 μs. Az időtartam tehát 7,5 μs lépésekben szabható meg ezen a módon és a TIME érték függvényében 7,5 μs (ha TIME = *1*) és 256×7,5 = 1920 μs (ha TIME = *0*) között állítható be.

Hosszabb idejű késleltetést a LDELY szubrutin segítségével valósíthatunk meg (2.23. ábra).

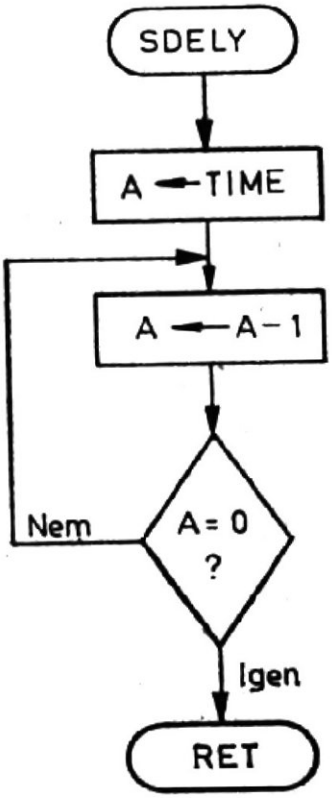
```
Ciklusszám LDELY LXI D,T16  ; BEÍRJA A T16 IDŐÁLLANDÓT
                            ; A DE REGISZTERPÁRBA
         5 LOOP  DCX D      ; CSÖKKENTI DE REGISZTERPÁR
                            ; TARTALMÁT
         5       MOV A,D    ; D–T ÁTÍRJA A-BA
         4       OR E       ; D = A = E = 0 ?
        10       JNZ LOOP   ; HA ≠ 0 VISSZAUGRIK
        24       RET        ; VISSZATÉR
```

A ciklus magja ebben az esetben 24·500 ns = 12 μs alatt hajtódik végre. Így az időzítés minimum 12 μs (T16 = *1*) maximum 2^{16}·12 μs = 0,786432 s (T16 = *0*).

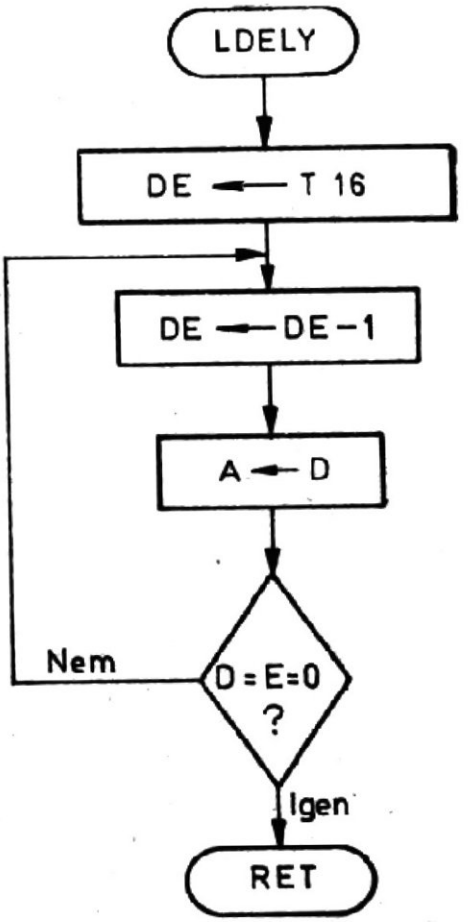
Az időzítést a szubrutin hívásával indíthatjuk:

```
CALL DELY
```

Gyakorlatban sokszor szükség van arra, hogy a késleltetés indítása után a prog-



2.22. ÁBRA. Rövid idejű késleltetés szubrutinjának folyamatábrája



2.23. ÁBRA. Hosszabb idejű késleltetés szubrutinjának folyamatábrája

ram végrehajtását folytassuk. Ilyenkor egy külső kiegészítő elemben lévő regisztert töltünk fel adott kezdőértékkel. Ezen regiszter tartalmának a növelése a program végrehajtásával párhuzamosan történik, és ha az előre beállított léptetésszám végrehajtásra került, a külső elem megszakításkéréssel jelentkezik.

Kombinációs logikai függvények minden esetben felírhatók logikai szorzatok összege formájában. Mint láttuk, a mikroprocesszorok a szükséges műveleteket – ÉS, VAGY, NEM műveleteket – egyszerű módon képesek elvégezni, így egy ilyen logikai függvényt megvalósító program is elkészíthető.

Az adott függvénynek megfelelő programot közvetlenül is megírhatjuk (Ad hoc módszer) [10]. Legyen pl. a következő négyváltozós függvényünk:

$$F(W, X, Y, Z) = \overline{W} \cdot X \cdot \overline{Y} + Z.$$

Tételezzük fel, hogy az egyes változók egymást követő *W, X, Y, Z* és *F* tárrekeszekben vannak (az IGAZ értéknek az *1*, a HAMIS értéknek a *0* feleljen meg), és a HL regiszterpár tartalma már az első változóra (*W*) „mutat". A programban a negálást az *1* értékkel való kizáró VAGY művelettel végezzük.

```
START MOV A,M     ; W-T BEHOZZA A-BA
      XRI 01H     ; NEGÁLJA W-T
      INX H       ; MEGNÖVELI HL-T
      ANA M       ; W̄ ÉS X
      MOV B,A     ; MEGŐRZI W̄X-T A B REGISZTERBEN
      INX H       ; MEGNÖVELI HL-T
      MOV A,M     ; Y-T A-BA ÍRJA
```

```
        XRI 01H      ; NEGÁLJA Y-T
        ANA B        ; W̄X ÉS Ȳ
        INX H        ; MEGNÖVELI HL-T
        ORA M        ; WXȲ VAGY Z
        INX H        ; MEGNÖVELI HL-T
END     MOV M,A      ; ELTESZI F-T A MEMÓRIÁBA
```

Ennek az egyedi megoldásnak, bár vitathatatlan előnye, hogy a program közvetlenül követi az adott logikai függvényt, számos hátránya van. Egyrészt nagyon gazdaságtalan minden egyes függvényhez programot írni, másrészt a tárhelyekkel sem gazdálkodtunk valami takarékosan, amikor egy bit helyett egy teljes byte-ot használtunk fel egy változó tárolására.

Dolgozzunk ki most egy olyan általános programot, amelynek segítségével egy adott változószámig bármilyen logikai függvényt kiértékelhetünk. A módszer alapja az egyes tényezők összehasonlítása előre tárolt olyan byte-okkal, amelyek az egyes lehetséges szorzattényezőknek (mintermeknek) felelnek meg.

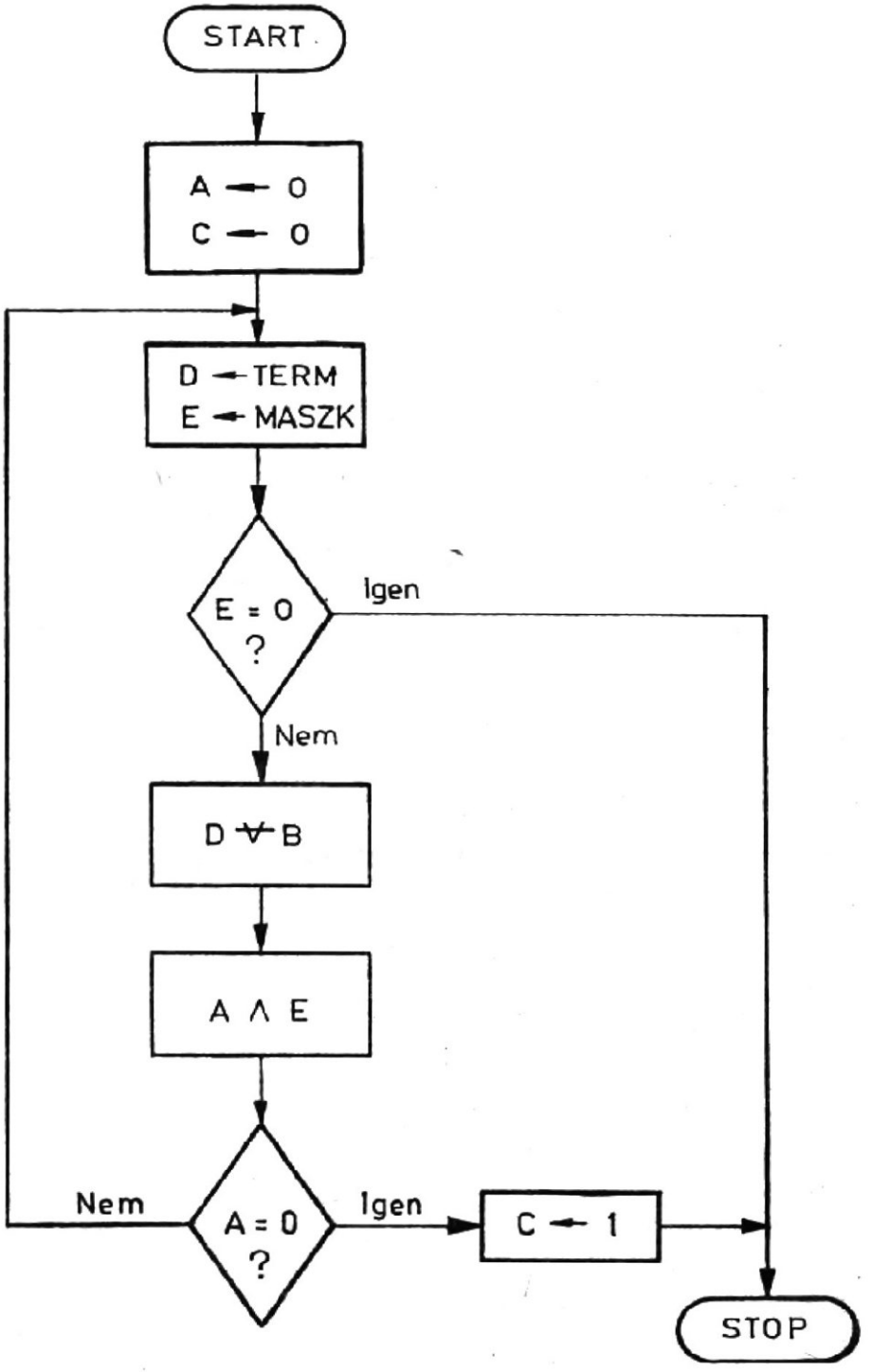
A változók előre meghatározott sorrendben egy-egy bitnek felelnek meg (az IGAZ értéket az *1*, a HAMIS értéket a *0* jelenti ismét). A program sorban összehasonlítja az egyes tényezőket, és az így kapott tényezők VAGY kapcsolata fogja a függvényt megadni. A keresett függvény azonban nem mintermek összege, vagyis nem szerepel minden tagban minden változó (vagy annak negáltja). Az ilyen minimalizált alak összehasonlítása előtt ezért maszkolással ki kell iktatni a nem szereplő (don't care) változókat.

Legyenek a változók a B regiszterben, az egyes összehasonlítandó tényezők pedig felváltva a megfelelő maszkokkal egy veremtárban. Ebből a tárból az egyes tényezőket és a hozzájuk tartozó maszkokat az i8080 veremtármutató regisztere segítségével vehetjük elő egymást követő **POP** utasítások segítségével. Jelentse a *0* tartalom ennek a veremtárnak a végét (2.1. átblázat). A maszk tehát megelőzi a megfelelő tényezőt (amelyben az X azt jelenti, hogy nem számít milyen érték szerepel, mivel az összehasonlítás során azon bit értékeket a maszkolás úgyis kitiltja).

2.1, TÁBLÁZAT. Maszkok és mintermek elrendezése a veremtárban

Elrendezés	Változók							
	S	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
1. Maszk	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*0*
1. Tényező	X	X	X	X	*0*	*1*	*0*	X
2. Maszk	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
2. Tényező	X	X	X	X	X	X	X	*1*
Veremtár vége	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
	X	X	X	X	X	X	X	X

A program folyamatábráját a 2.24. ábrán mutatjuk be. A kizáró VAGY művelet minden olyan bitpozícióban *0*-t ad, ahol egyező (mindkettő *0* vagy *1*) logikai értékek vannak, míg az ÉS művelet állítja be a nem szereplő változóknak megfelelő pozíciókban a bit értékét *0*-ra. Így a megfelelő változó pozíciókban való megegyezés



2.24. ÁBRA. Mintermekkel való összehasonlítás módszerrel készített program folyamatábrája

esetén az F-et (a C regiszter tartalma) *1* értékre állítja. A program a 8080 assembler jelölésével a következő:

```
START XRA A       ; A TÖRLÉSE
      MOV C,A     ; C TÖRLÉSE
      POP DE      ; TÉNYEZŐ ÉS MASZK D,E
                  ; REGISZTEREKBE TÖLTÉSE
      ADD E       ; MASZK ELLENŐRZÉSE ZÉRUS
                  ; ÉRTÉKRE
      JZ STOP     ; HA ZÉRUS, STOP
      MOV A,B     ; VÁLTOZÓ SZÓ A-BA TÖLTÉSE
      XRA D       ; NULLÁK, AHOL MEGEGYEZŐ
```

```
        ANA E          ; NULLÁK, AHOL NEM SZEREPLŐ
                       ; VÁLTOZÓK VANNAK
        JZ ONE         ; HA ZÉRUS, F = 1
        JMP START+2    ; HA NEM, KÖVETKEZŐ TÉNYEZŐ
ONE     INR C          ; C = F = 1
STOP
```

Az előbbi programot (maximum nyolc változóig) lehet bármilyen logikai függvényhez használni egyszerűen beállítva a veremtármutatót a megfelelő táblázat kezdetére (és beírva a megfelelő értéket a B regiszterbe).

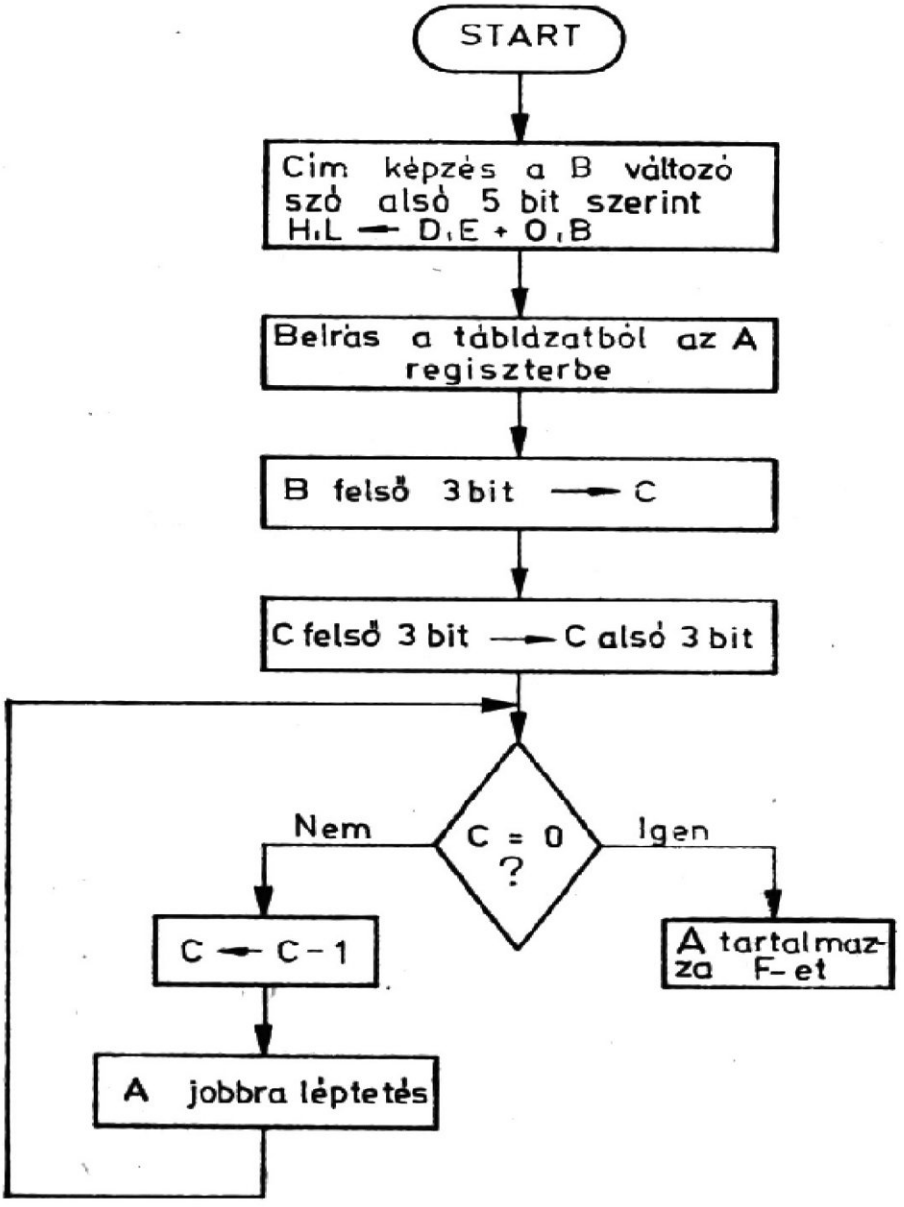
A kombinációs hálózatok hardvermegvalósításához hasonló eljárást alkalmazunk, amikor egy előre tárolt táblázatból keressük ki a megfelelő függvényértéket. A bemenő változókat, amelyeket egy szóban tárolunk, használjuk mutatóként a táblázatból való kereséshez. Nyolc bemenő változó 256 lehetséges címet jelölhet ki. Ha egyetlen kimenetünk van, harminckét 8-bites szóra van szükség. Ekkor öt bemenő változó választja ki a byte-ot és, a három maradék változó a megfelelő bitet a byte-on belül.

2.2. TÁBLÁZAT. Az $F = \overline{W} \cdot X \cdot \overline{Y} + Z$ függvény táblázata

Relatív cím	Tartalom	Relatív cím	Tartalom
00000	00000000	10000	00000000
00001	00000001	10001	00000001
00010	00000000	10010	00000000
00011	00000001	10011	00000001
00100	00000001	10100	00000001
00101	00000001	10101	00000001
00110	00000000	10110	00000000
00111	00000001	10111	00000001
01000	00000000	11000	00000000
01001	00000001	11001	00000001
01010	00000000	11010	00000000
01011	00000001	11011	00000001
01100	00000000	11100	00000000
01101	00000001	11101	00000001
01110	00000000	11110	00000000
01111	00000001	11111	00000001

Ha pl. a függvényünk $F = \overline{W} \cdot X \cdot \overline{Y} + Z$, a változókat beolvassuk és tároljuk egy regiszterben (használjuk erre a célra a B regisztert). Ennek a függvénynek a táblázatát a 2.2. táblázat mutatja. A változók sorrendje *STUVWXYZ*. A négy nem használt változó (*STUV*) mindig *0*, és így a kimenő bit mindig a legkisebb helyiérték pozíciójában van. Tételezzük fel, hogy a program végrehajtásakor a táblázat kezdőcíme már a D és E regiszterekben van. A program, amelynek folyamatábráját a 2.25. ábrán mutatjuk be, a következő:

```
START MVI A,1FH    ; MASZK AZ ALSÓ 5 BIT
                   ; HELYÉRTÉKÉNEK MEGFELELŐEN
```



2.25. ÁBRA. Előre tárolt táblázatból való keresés módszerével megvalósított program folyamatábrája

```
ANA B       ; A VÁLTOZÓ ALSÓ 5 BIT
            ; ÉRTÉKET A-BA ÍRJA
MOV L,A     ; TÁBLÁZAT RELATÍV CÍMMÉ
            ; ALAKÍTJA A VÁLTOZÓ BYTE-OT
MVI H,00H   ; H = 0 BEÁLLÍTÁSA
DAD D       ; ABSZOLUT CÍMMÉ ALAKÍTJA
            ; DE KEZDŐCÍMET HOZZÁADVA
MVI A,0E0H  ; BIT MASZKOT KÉPEZ A FELSŐ
            ; HÁROM HELYÉRTÉKNEK MEGFELELŐEN
ANA B       ; HÁROM FELSŐ BITET A-BA ÍRJA
RLC         ; A7A6A5 POZÍCIÓBÓL
RLC         ; A2A1A0 POZÍCIÓBA
RLC         ; LÉPTET
ADI 00H     ; BEÁLLÍTJA A FELTÉTEL JELZŐBITET
MOV C,A     ; BIT POZÍCIÓT C-BE ÍRJA
MOV A,M     ; KIMENŐ BYTE-T A-BA ÍRJA
```

```
TEST   JZ STOP      ; ELLENŐRZÉS ZÉRUSRA
       DCR C        ; BIT SZÁM CSÖKKENTÉS
       RRC          ; KIMENŐ BYTE JOBBRA LÉPTETÉS
       JMP TEST     ; VISSZA TEST-RE
STOP
```

A művelet végén a függvény értéke az A regiszterbe kerül.

IRODALOM

[1] *Cavlan, N.–Durham, S. J.:* Field-programmable arrays: powerful alternatives to random logic. Electronics, July 5, 1979. 110–114.

[2] PAL programmable array logic handbook. Monolithic Memories, 1978.

[3] The integrated circuits catalog for design engineer (Programmable logic arrays). Texas Instruments, 1978. 14.200–14.212.

[4] *Cavlan, N.–Durham, S. J.:* Sequencers and arrays transform truth tables into working systems. Electronics, July 19, 1979. 132–139.

[5] *Janovics S.–Tóth M.:* A logikai tervezés módszerei. Műszaki Könyvkiadó, Budapest, 1973.

[6] *Kells, A. E. et al.:* Standard specification for S–100 bus interface devices. Computer, July 1979. 28–52.

[7] *Madarász, L.:* Általánosan használható programozható logikai áramkörök. Automatizálás, 1979. szeptember 26–33.

[8] *Osborne, A.:* 8080 programming for logic design. Adam Osborne and Associates, Inc., Berkeley, California, 1976.

[9] *Csákány A.–Vajda F.:* Mikroszámítógépek. Műszaki Könyvkiadó, Budapest, 1977.

[10] *Pokoski, J. L.:* Software analyses for combinational logic. Computer Design, June 1978. 113–118.

3. A MIKROPROCESSZOROS RENDSZEREK FEJLESZTÉSE

A mikroprocesszoros rendszerek fejlesztése sokoldalú – általában csoportmunkát – igénylő feladat. Akár a központi egység, akár a tár, vagy a be-, ill. kimeneti áramkörök elemeit vagy felépítését vizsgáljuk, nagyon sok választási lehetőség, döntési változat létezik. De hasonlóan sok megoldási változat képzelhető el egy-egy alkalmazási feladat programrendszerére vonatkozóan is. Nem könnyű tehát a leggazdaságosabb, legjobb megoldás kiválasztása.

Ebben a fejezetben az LSI és a VLSI alapú rendszerek kidolgozásának, bemérésének és ellenőrzésének a kérdéseivel foglalkozunk. Bemutatjuk a rendszerek létrehozása során elvégzendő tervezési, programozási, bemérési munkák általános menetét. Ennek a megismerése segítséget nyújthat a mikroprocesszoros berendezésekkel, programokkal újonnan ismerkedő szakembereknek a könnyebb eligazodáshoz és tájékozódáshoz.

3.1. AZ ALKALMAZÁSOK CSOPORTOSÍTÁSA

A programozható és a programozott LSI és VLSI elemekre épülő készülékeket, berendezéseket és rendszereket már eddig is nagyon sok területen alkalmazták. Az alkalmazások köre és száma azonban egyre tovább nő. Az alkalmazásokat osztályozva három lényegesebb felhasználási csoportot jelölhetünk meg [1] [2]. Ezek:

- azoknak a feladatoknak a köre, melyeket már korábban is számítástechnikai eszközök (miniszámítógépek) felhasználásával oldottak meg,
- a vezérlőlogikai rendszerek vagy egyéb elektronikus irányító- és vezérlőberendezések (analóg vagy digitális rendszerek) kiváltása,
- olyan feladatok csoportja, amelyekhez korábban elektronikus irányító-, szabályozó- vagy mérő- és adatfeldolgozó eszközöket sehol sem alkalmaztak.

Az első csoportba sorolható alkalmazások – melyek mikroszámítógépes megoldásúak (l. 62. oldalt) – viszonylag kevés problémát jelentenek. Az a gazdag tapasztalat, amely a számítástechnikai rendszerek tervezése, kidolgozása, üzemeltetése és karbantartása terén az elmúlt évek során felhalmozódott, igen jól kamatoztatható és hasznosítható a miniszámítógépek mikroprocesszoros rendszerekkel – mikroszámítógépekkel – való kiváltásához. Az ilyen rendszerek hardverfelépítése és programstruktúrája nagyon hasonlít a számítógépes rendszerekéhez.

A következő két csoportba tartozó alkalmazások esetén – melyek a programozott vezérlő logikai áramkörökkel oldhatók meg – eltérő a helyzet. A hagyományos vezérlő logikai áramkörök tervezési, bemérési tapasztalatait és módszereit a programozott logikák (l. 40. oldalt) kidolgozásának az eljárásai és módszerei váltják fel, de sok problémát vethet fel az LSI és a VLSI alapú elektronikus rendszerek egészen új területekre való behatolása is.

Kidolgozásukat és a feladatmegoldás során követendő módszereket illetően a mikroszámítógépes és a programozott logikai rendszerek jelentősen különböznek

3.1. TÁBLÁZAT. A mikroszámítógép és a programozott logika típusú alkalmazások összehasonlítása

Jellemzők	Mikroszámítógépek	Programozott logikák
Hardverépítés kialakítása	A rendszer kész, működőképes modulokból építhető fel	A felhasználó tervezi és építi meg LSI és VLSI áramkörökből
Programtárolás és futtatás	Általában RAM-ban	Általában ROM-ban vagy PROM-ban
Fejlesztési segédeszközök	A kidolgozandó rendszer tartozékai	Különálló, csak a fejlesztés során használatos segédeszközök
Ki-, ill. bemenetek, interfészek	Hagyományos felépítés	Szorosan rendszer-kapcsolatú, általában LSI és VLSI alapú interfészmodulok
Rendszer tesztelés, ill. ellenőrzés	Hasonló a számítógépes teszteléshez, ellenőrzéshez	A vezérlési funkciók és a célfeladat ellenőrzésére korlátozódik
Felhasználói programok kidolgozása	A számítógépes rendszerekhez hasonló	Gyakran a minimális programtárolási (PROM) szükséglet elérése a célfüggvény
Gyártás	A kész rendszer általában kisebb darabszámú gyártása	A kész rendszer általában nagyobb darabszámú gyártása

egymástól. A leglényegesebb eltéréseket, ill. az eltérést mutató jellemzőiket a 3.1. táblázat foglalja össze [3].

A mikroszámítógép típusú rendszerek felhasználói a rendszerhardver felépítéséhez többnyire kész, működőképes hardvermodulokat szereznek be az azokat gyártó vállalatoktól [4]. Ezzel igen radikálisan csökkenthető a hardverépítéssel kapcsolatos problémák száma, csökkenthető a rendszer létrehozásához szükséges idő. A rendszer működtetését biztosító algoritmus – a felhasználói program – azonban már „egyedi", létrehozása a felhasználó feladata. Ez a mikroszámítógép alapszoftverjének a segítségével a megépített hardveren végezhető el. Az ilyen jellegű programfejlesztés természetesen szükségessé teszi, hogy a mikroszámítógépes rendszer viszonylag nagy kapacitású RAM-ot és hagyományos be-, ill. kimeneti perifériákat is tartalmazzon. Mivel a mikroszámítógép tára a fejlesztési feladatokhoz méretezett, s az esetek nagy részében a felhasználói program helyigényéhez képest nagyobb kapacitású, általában nem cél a program méretének a mindenáron való csökkentése. Az ilyen alkalmazások akkor célszerűek és kifizetődőek, ha nem túl sok rendszerről van szó, ezért a végtermék árát nagymértékben terhelik a fejlesztési költségek, vagyis akkor, ha a fejlesztési kiadásokat és a kidolgozási időt kell elsősorban csökkenteni.

A programozott logikák esetében, különösen, ha nagysorozatú gyártmány kifejlesztése a cél, a fejlesztési kiadások kevésbé terhelik a végtermék egy-egy darabját; annál inkább fontos azonban a gyártási költségeknek, s így a rendszer elemeire for-

dított költségeknek a csökkentése. Ezért lényeges, hogy a felhasználó mind a hardver, mind pedig a szoftver kidolgozása során a megfelelő körültekintéssel és gondossággal járjon el, s ne hozzon létre a szükségesnél nagyobb és bonyolultabb rendszert. A legtöbb alkalmazás esetén követelmény, hogy bekapcsolás után a „programozott logika" azonnal működőképes legyen. Ez csak „fix" programokkal, vagyis PROM-ba „égetett" vezérlőprogramokkal érhető el. Ez esetben nincs szükség a programoknak működés előtti tárba töltésére hagyományos (pl. lyukszalagos) perifériák segítségével.

Röviden összefoglalva elmondható, hogy az LSI és a VLSI alapú programozott logikák megtervezése és létrehozása során a fejlesztési munkáknak a következő lényeges sajátságai vannak:

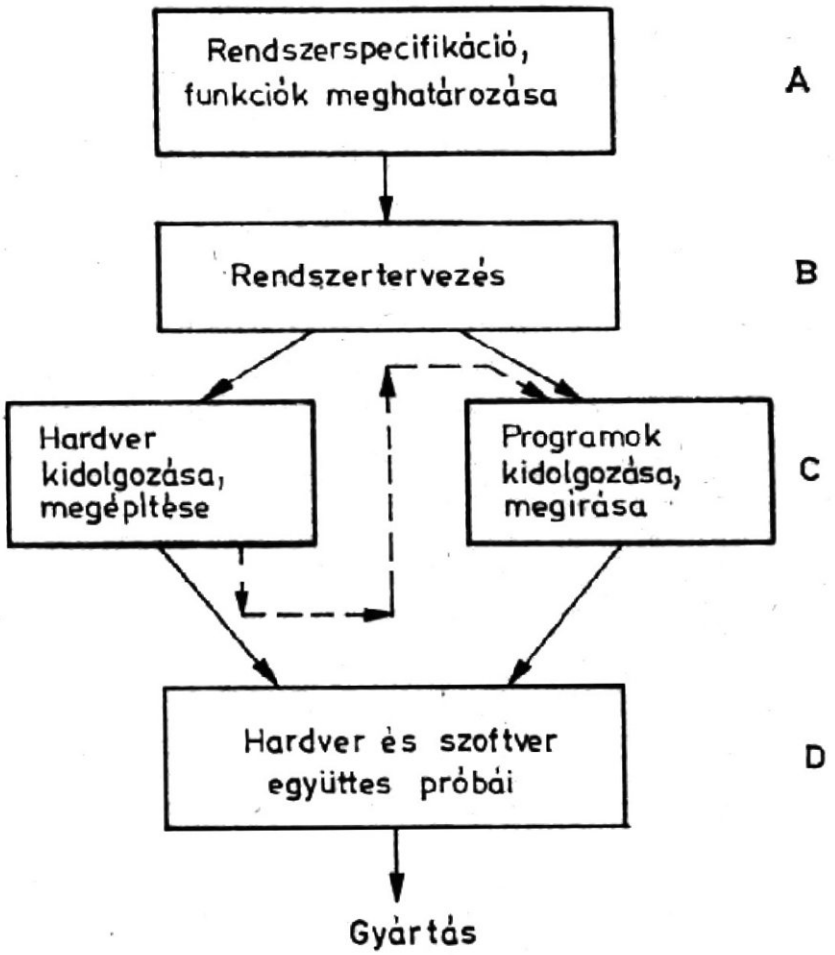
- A felhasználó részéről szükség van mind a hardver, mind pedig a szoftverfunkciók kidolgozására.
- A rendszer létrehozása az alábbi részfeladatokra bontható: hardver tervezése, megépítése; a felhasználói programok megírása; a hardver, ill. szoftver rendszer együttes próbája, ellenőrzése.

A programok próbája, valamint az együttes hardver-, ill. szoftverellenőrzés során figyelembe kell venni, hogy a működtető programok véglegesen ROM-ba vagy PROM-ba írt formában kerülnek felhasználásra.

- A kidolgozáshoz olyan segédeszközökre van szükség, amelyek a végterméknek nem tartozékai, tehát könnyen leválaszthatók az ellenőrzött és kipróbált gyártmányokról.

Az LSI és a VLSI alapú rendszerek kidolgozása közben felmerülő nehézségek a következők:

- Az alkatelemek belső felépítésének bonyolultsága, a lehetséges belső állapotok igen nagy száma, valamint a viszonylag kevés kivezetés igen megnehezíti, hogy



3.1. ÁBRA. A mikroprocesszoros rendszerek fejlesztésének szakaszai

az elemek be- és kimenetein keresztül az azokon mérhető jelszinteknek az alapján pontosan meghatározható legyen a még nem teljesen ellenőrzött kapcsolási elrendezés és a vezérlőprogram állapota, helyes működése.

- Az együttes ellenőrzés során észlelhető helytelen működés okát nehéz megkeresni, problémát okoz annak a kiderítése, hogy a hiba eredete a hardverben, a szoftverben vagy azok „illesztésében" rejlik-e.
- A programvezérlés miatt akár a hardver, akár a szoftver hibái között előfordulhatnak olyanok, amelyek nagyon ritkán jelentkeznek (esetleg egyszer, s ezután a rendszer „befagy"). Az ilyen nem ciklikus hibáknak a behatárolása és kinyomozása nagy problémát jelenthet.

3.2. A FEJLESZTÉS MENETE

A programozott irányítási és vezérlőrendszerek fejlesztési munkái a következő szakaszokból állnak (3.1. ábra):

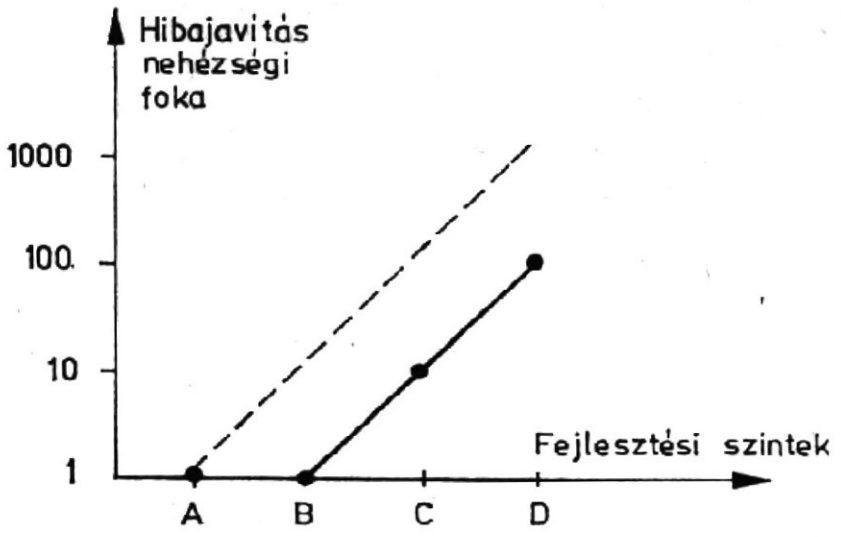
- rendszerspecifikáció, funkciók meghatározása,
- rendszertervezés, funkciók feladatokra bontása,
- a hardver kidolgozása, megépítése,
- a programok kidolgozása, megírása,
- a hardver és a működtető programok együttes próbája,
- gyártáselőkészítés, gyártás.

A rendszer specifikációja és a funkcióinak a meghatározása a konkrét alkalmazás célkitűzéseiből, feladatából elvégezhető. Ennek eredménye egy követelmény-, ill. funkciófelsorolás a lényeges paraméterek megjelölésével.

A *rendszertervezés* során már modulokra, blokkokra bontható a tervezendő rendszer. Ennek a fázisnak a leglényegesebb eredménye a hardver, ill. a szoftver feladatok körének a meghatározása.

Ha a *hardver* valamint a *programok kidolgozását* és részleges ellenőrzésüket egymással párhuzamosan sikerül elvégezni, lényegesen meggyorsítható a kidolgozás. Amennyiben ilyen párhuzamos munkavégzésre nincs lehetőség, általában a programok kidolgozása, megírása a hardverépítés után következik. A párhuzamos hardver- és szoftverkidolgozást csak egyes fejlesztői segédeszközök és a használatukhoz kapcsolódó módszerek teszik lehetővé (l. a 4. fejezetet).

Csak a *hardver- és a szoftverfunkciók együttes ellenőrzése* során lehet a prototípust véglegesen kipróbálni. Természetesen az észlelt hibák kijavítása sokszor csak az adott tervezési szakaszból egy korábbi szakaszra való visszatéréssel végezhető el.



3.2. ÁBRA. A fejlesztési szintek és a hibajavítás nehézségi fokának a kapcsolata

A prototípus teljes kipróbálását követi az LSI és a VLSI alapú *készülék vagy rendszer gyártása.*

Igen fontos az, hogy a tervezés, kidolgozás során elkövetett tervezési, megépítési hibák minél korábban felfedezhetők legyenek. A 3.1. ábrán az egyes szakaszoknak szinteket feleltetünk meg (A, B, C, D szintek az ábrán). Elmondható, hogy valamely szinten elkövetett hiba kijavítására annál több időt és energiát kell fordítanunk, minél későbbi szinten kerül sor a hiba felfedezésére. A szintek számával megközelítőleg exponenciálisan nő a kijavítás nehézségi foka (3.2. ábra) [5].

Az ábrán a B szinten, a hardver- és a szoftverfeladatokra bontás szintjén elkövetett hiba kijavításához egységnyi időre van szükség, ha még a következő szakaszra való áttérés előtt észrevehető és kijavítható a hiba. Ha csak a következő szakaszban (a hardver és szoftver, ill. mindkettő kidolgozása alatt) észleljük a hibát, kijavítása megközelítőleg 10-szeres, míg ennél későbbi észlelés esetén (az együttes próbák szakaszában) már kb. 100-szoros időt és energiát igényel.

Rendszerspecifikáció, funkciók meghatározása

Minden alkalmazási munka az adott feladat meghatározásával kezdődik. A feladat ismeretében előírhatók a kidolgozandó rendszerrel kapcsolatos követelmények, meghatározhatók azok az alapvető funkciók, amelyek ezeknek a követelményeknek a teljesítéséhez szükségesek. Az alapspecifikáció kidolgozása, s a legfontosabb funkciók meghatározása a további tervezési munkákhoz kiindulási alap. Sok alkalmazást rövidebb, ill. hosszabb távú specifikációs célkitűzéseket tartalmaz. Ez azt jelenti, hogy a kidolgozandó mikroprocesszoros rendszer kezdetben viszonylag kevés feladatot lát el, később azonban a további bővítése, a meglévőkhöz újabb feladatoknak a hozzá csatolása is szükségessé válik. A bővíthetőség és a továbbfejlesztés lehetősége igen lényeges követelmény, melyet a későbbiekben mind a hardver kidolgozása, mind pedig a programok fejlesztése során figyelembe kell venni.

Rendszertervezés

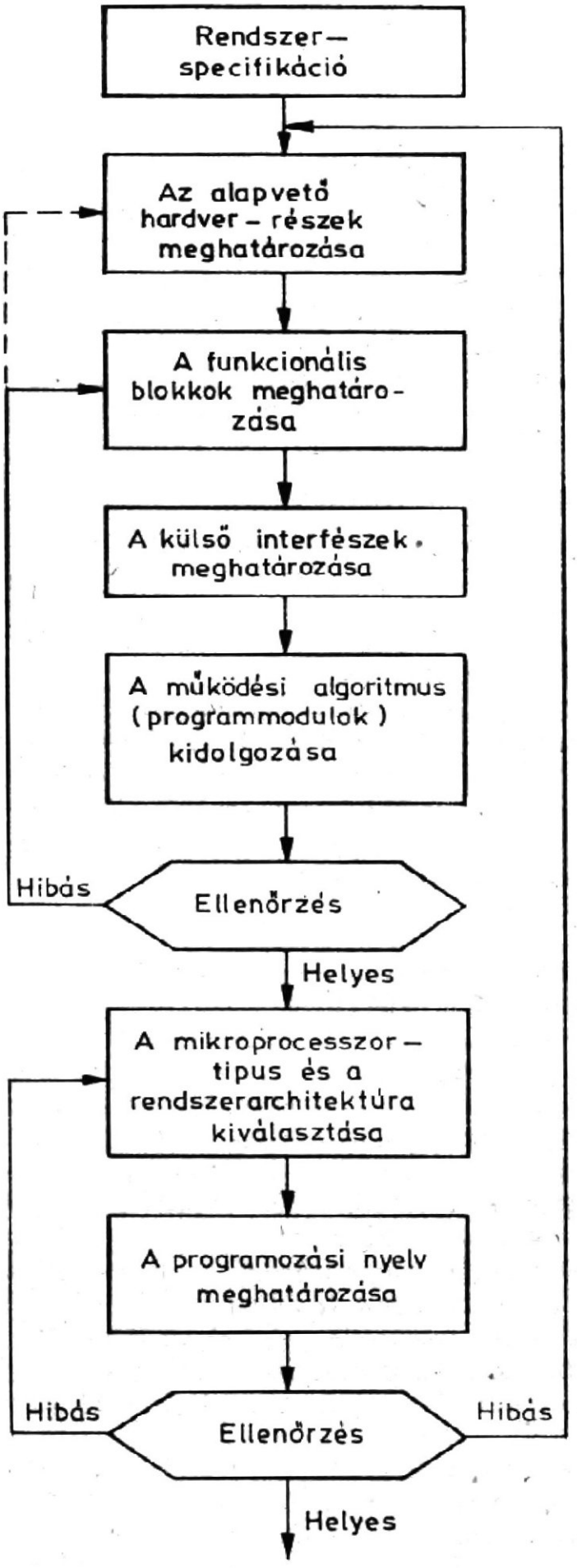
A rendszertervezési szakasz a további fejlesztési munkákat megalapozó rész. Ebben a szakaszban történik a rendszer funkcióinak a hardver-, ill. szoftverrészekre bontása, a hardver blokkjainak és moduljainak az előzetes meghatározása, valamint a szoftver főbb programrészeinek a kidolgozása. A rendszertervezési munkák gondos és körültekintő elvégzésétől, a hardver-, ill. a szoftverrészek közötti arányok helyes megválasztásától nagymértékben függ az egész fejlesztés sikere.

A 3.3. ábrán feltüntettük a rendszertervezés menetét. Az alkalmazási feladatok függvényében a rendszertervezési szakasz egyes fázisai kisebb vagy nagyobb jelentőségűek, mint mások, a sorrend is változhat.

Az alapvető hardverrészek meghatározása során kell eldönteni, hogy melyek azok a részfeladatok, amelyek csak hardverrel valósíthatók meg (pl. A/D, D/A konverzió, információátviteli csatornák, kijelzések stb.). Ezt követően elkészíthető a hardver funkcionális blokkokra bontásának az első változata, majd ezeknek a blokkoknak az egymás közötti s a külvilág felé való kapcsolatának a meghatározása.

Igen fontos lépés a programrészek működési algoritmusának a felvázolása, az algoritmus legfőbb funkcióinak a meghatározása.

Miután megközelítően már elvégeztük a rendszer hardver-, ill. szoftverfunkcióinak a meghatározását, célszerű ellenőrizni az eddig kidolgozott részeket, végiggondolni az eddigi döntések helyességét, s szükség esetén megismételni az eddigi lépéseket.



3.3. ÁBRA. A rendszertervezés folyamatábrája

A rendszertervezés ezt követő lépéseihez már kissé részletesebb kidolgozásra van szükség. Igen fontos a további kérdések eldöntése:

- Milyen típusú mikroprocesszort vagy mikroprocesszorokat és egyéb LSI és VLSI elemeket alkalmazzunk?
- Milyen legyen a rendszerarchitektúra, egy- vagy többprocesszoros rendszer elégíti-e ki a specifikációs követelményeket?
- A programok kifejlesztéséhez milyen eszközöket, programnyelveket célszerű használni?

A feladat ismeretében gondos mérlegelés után válaszolhatunk ezekre a kérdésekre. Természetesen a kérdésekkel kapcsolatos döntéseket is össze kell vetni egymással, s ellenőrizni azok helyességét. Az egyik legfontosabb rendszertervezési feladat annak az ellenőrzése (becslése), hogy a rendszertervezés eredményeként kidolgozott változat sebessége elegendő-e a specifikációs követelmények ellátásához.

A mikroprocesszorok két kategóriája ismeretes:

- a rögzített utasításkészletű mikroprocesszorok,
- a mikroprogramozott mikroprocesszorok.

A típus kiválasztását megelőzően fontos eldönteni, hogy az alkalmazási feladatnak a fenti két kategória közül melyik felel meg jobban. A döntéshez támpontot nyújthat előnyeik, ill. hátrányaik összevetése.

A *rögzített utasításkészletű mikroprocesszorok* alkalmazása egyszerűbb. Kevesebb elem beépítésére van szükség és hatékonyan alkalmazhatók a közepes, ill. nagysebességű vezérlési és adatkezelési feladatokhoz. Ezzel szemben a *mikroprogramozott mikroprocesszorok* bonyolultabb rendszereket eredményeznek, működtetésükhöz a makroutasításokat megvalósító mikroprogramok összeállítására is szükség van. Előnyük viszont a gyorsabb működés és a mikroprogramok megváltoztatásával elérhető flexibilis (makro) utasításkészlet (l. a 11. fejezetet). Ez utóbbi lényegesen megkönnyítheti az alkalmazáshoz „szabható" alkalmazói programok kidolgozását.

A kategórián belüli típusok közül a legmegfelelőbb megtalálásához gondosan át kell tanulmányozni azok funkcionális paramétereit, szóhosszukat, működési sebességüket, áraikat és azt, hogy mennyire bonyolult hardverrendszert eredményez az adott típus. A nyolcbites szóhossz pl. igen jól illeszkedik azokhoz a feladatokhoz, ahol az adatfeldolgozás során byte-os műveleteket, byte-ok mozgatását kell végezni. A nagyobb szóhossz (16bit) viszont potenciálisan nagyobb műveleti sebességet eredményez, ha sok a byte-nál hosszabb paraméterekre vonatkozó művelet. Fontos választási szempont lehet a processzor-architektúra is. A regiszterorientált (sok belső regisztert tartalmazó) mikroprocesszorok az irányítási–vezérlési feladatokhoz jól illeszkednek, míg a tárorientált processzorok (indexregisztereikkel, közvetett címzésre szolgáló regisztereikkel, bázisregisztereikkel) kellemesebben alkalmazhatók az adatfeldolgozáshoz [6].

A központi egység funkcióit ellátó mikroprocesszor mellett egyéb LSI és VLSI áramkörök is szükségesek a rendszerépítéshez. Ezért igen lényeges megvizsgálni, hogy milyen könnyen építhetők össze egy berendezésbe az általunk kiválasztott elemek. Abban az esetben, ha a kiválasztott mikroprocesszor és a kiegészítő LSI és VLSI elemek egy „családba" tartoznak, az összeépítés nem jelent problémát, más esetekben azonban körültekintőbben kell eljárni a kiválasztás, ill. a hardver megtervezése során.

A rendszerarchitetkrúrára vonatkozó kérdésre a következő lehetséges válaszok adhatók:

- egyprocesszoros vagy
- több mikroprocesszort tartalmazó

rendszerrel valósítható meg a feladat. Amennyiben a feladat teljesítéséhez egyetlen processzor elegendő, célszerűbb az előbbi mellett döntenünk. Egyetlen processzor működtetéséhez a program kidolgozása egyszerűbb, könnyebb.

Bonyolultabb feladatok ellátásához több processzort is építhetünk a felhasználói berendezésbe (l. a 12. fejezetet).

A programnyelv kiválasztására vonatkozó kérdés kapcsán az assembler előnyeit és hátrányait mutatjuk be a magasabb szintű programnyelvekkel összevetve.

Az assembler nyelvű programok gépi utasításonként egy-egy programsort programutasítást tartalmaznak. A magas szintű programnyelvek egy-egy programsora több utasításnak felel meg. Programok írása magas szintű programnyelven rövidebb ideig tart, hatékonyabban végezhető el (l. a 10. fejezetet). Meggyorsul a hibakeresés is. Ugyanakkor azonban a végrehajtási időt és a tárgyprogram hosszát illetően a magas szintű programnyelvek használata hátrányokkal jár: nagyobb tárkapacitásra van szükség és nagyobb műveleti időket eredményeznek.

Az alkalmazások egy részénél nem kritikus a végrehajtási idő. Ez esetben a nagyobb tárkapacitásigény jelenti a hátrányt. A magas szintű nyelven készült tárgyprogram helyigénye – alkalmazástól, compilertől, a programozó „ügyességétől" függően – 10% ... 100%-kal haladhatja meg az assembler nyelven írt program hosszát [7]. Assemblert használva a programírás tehát költségesebb, viszont a tár kevesebb anyagi ráfordítással építhető meg.

A programírás költsége a programsorok számával arányos. Egy-egy sornyi program költsége közel azonos, függetlenül a használt programnyelvtől. Így a kidolgozási költség:

$$s \cdot p,$$

ahol s a sorok száma, p pedig a programozási költség soronként.

A tár ára:

$$s \cdot b \cdot m$$

ahol m egy bit ára, b pedig az egy (átlagos) utasítószó bitjeinek a száma.

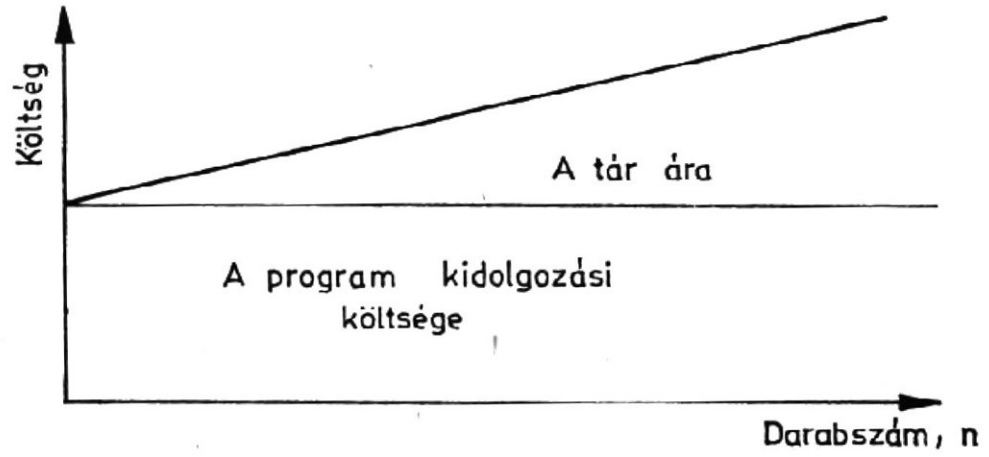
Sorozatgyártás esetén, ha n darab készülék ugyanazzal a programmal működik, az n készülék programmal való ellátásának a költsége a következő:

$$n \cdot s \cdot b \cdot m + s \cdot p.$$

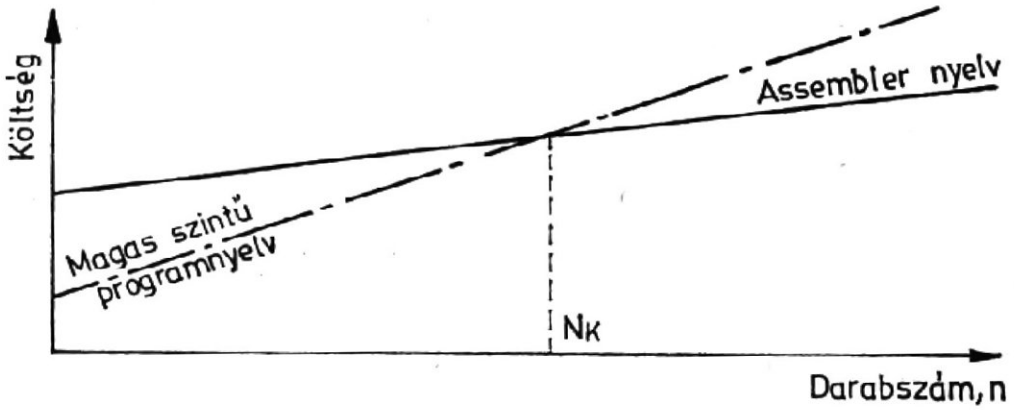
Az assembler nyelv alkalmazásával s_a sorból álló, soronként átlagosan b_a bitnyi tárgyprogramot kapunk. A magas szintű programnyelv programsorainak a számát s_c-vel, az egy átlagos programsorhoz tartozó bitek számát b_c-vel jelöljük. A tár ára (m) független a programnyelvtől, az egy sornyi program kidolgozására fordított költséget (p) is közel programnyelvfüggetlennek tekinthetjük. A 3.4. ábrán ábrázoltuk n készülék programellátási kiadásait. Az $s_c < s_a$ viszonynak köszönhetően kis darabszámok (n) mellett a magas szintű nyelv alkalmazása előnyös, mivel azonban a $b_c > b_a$, a költségek a darabszám növekedésével meredekebben emelkednek, mint az assembler nyelvű programok esetén. A két egyenes így egy pontban az n_k darabszámhoz tartozó pontban metszi egymást (3.5. ábra).

Az $n = n_k$ értékre felírható a következő egyenlőség:

$$n_k \cdot s_c \cdot b_c \cdot m + s_c \cdot p = n_k \cdot s_a \cdot b_a \cdot m + s_a \cdot p.$$



3.4. ÁBRAA készülék programmal való ellátásának költsége a darabszám függvényében



3.5. ÁBRA. Az assembler és a magas szintű programnyelv használatának a gazdaságossága a darabszám függvényében

A tárgyprogramokra vonatkoztatva bevezethető a programnyelv tárkapacitásigény-növekedési tényezője: az e_c (Expansion factor). Ez azt mutatja, hogy hányszor nagyobb kapacitású programtárra van szüksége a nem assembler nyelvben írt tárgyprogramnak az assembler nyelvű programhoz képest.

$$e_c = \frac{s_c \cdot b_c}{s_a \cdot b_a}.$$

A gyakorlatban e_c értéke 1,1 és 2,0 közötti, esetleg ennél nagyobb is lehet. Ez az érték természetesen a programozó „ügyességétől" is függ.

Az n_k kritikus darabszámra tehát felírható a következő kifejezés:

$$n_k = \frac{p}{m} \cdot \frac{s_a - s_c}{s_c \cdot b_c - s_a \cdot b_a},$$

vagy behelyettesítve az e_c-t [7]:

$$n_k = \frac{p}{m} \cdot \frac{\frac{1}{b_a} - \frac{e_c}{b_c}}{e_c - 1}.$$

Ez utóbbi kifejezés már könnyen kezelhető, hiszen p és m értéke ismertnek tekinthető, az egy gépi utasításhoz tartozó átlagos b_a bitszám, továbbá a programnyelv egy-egy állításához tartozó b_0 bitszám a mikroprocesszor és az alkalmazott nyelv ismeretében jól becsülhető. Kis gyakorlattal (esetleg a korábbi alkalmazások tapasztalatára támaszkodva) e_0 értéke is megbecsülhető.

Összegezve az eddigi megállapításokat, azokból néhány fontos következtetés vonható le. Az alábbi esetekben gazdaságos a magas szintű programnyelv alkalmazása:

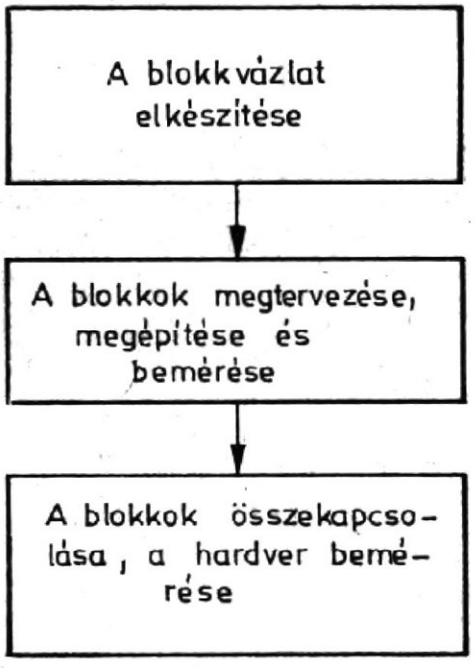
- Ha a kifejlesztendő berendezés kis darabszámban készül (n kisebb, mint néhányszor tíz).
- Ha a nagy darabszámú gyártmány programjait a megrendelők kívánsága szerint a leszállítandó készülékben változtatni kell.
- Ha a végfelhasználó (end user) a saját programját maga készíti el (a darabszám nagyon csekély, leggyakrabban $n = 1$).

A programnyelv kiválasztásához figyelembe kell venni a rendelkezésre álló segédeszközöket. A magas szintű nyelvek használatát csak megfelelő architektúrájú eszközök teszik lehetővé.

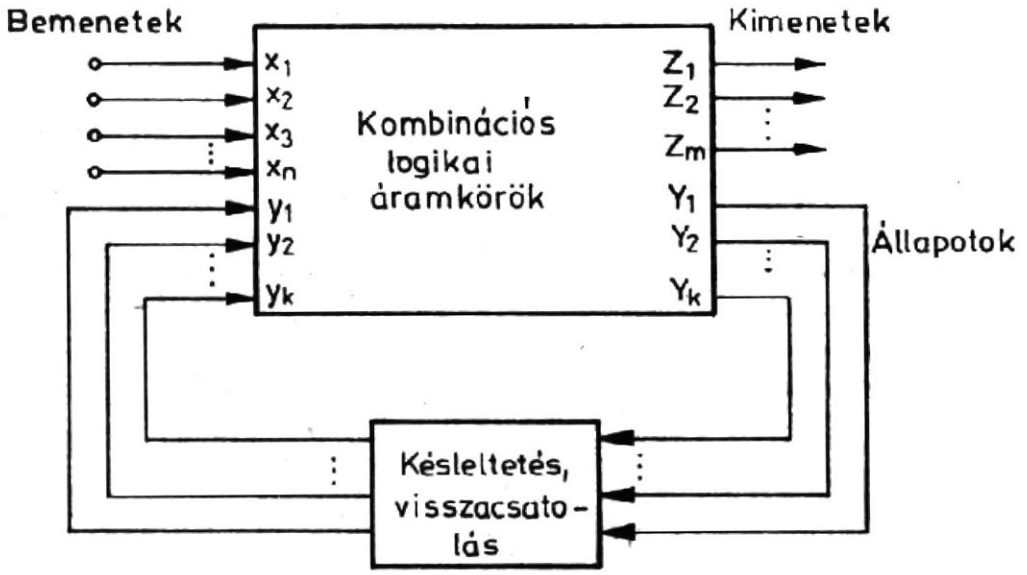
Általában a felhasználói szoftver kidolgozása során egyes programrészeket még akkor is assembler nyelven kell elkészíteni, ha a programrendszer megírásához magas szintű programnyelvet használunk.

A felhasználói hardver kidolgozása

A hardvertervezés tipikus folyamatát a 3.6. ábrán mutatjuk be. A tervezést a feladat részekre bontásával kezdjük. A részfeladatoknak blokkokat vagy modulokat feleltethetünk meg. Az LSI és a VLSI áramkörök alkalmazásakor gyakori, hogy a tulajdonképpeni mikroszámítógép-hardvernek a tervezése ezzel a fázissal be is fejeződik. Ha készen vásárolunk a gyártók által kínált áramköri kártyakészletekből kiválasztott mikroszámítógép-modulokat (bare-bones) s ezeket a modulokat az ugyancsak készen vásárolt tápegységgel együtt a hozzájuk tartozó mechanikai fiókba vagy házba beszereljük, készen áll a mikroszámítógépünk: Sok ún. „minimális" vagy az ehhez közel álló kis rendszer esetében hasonlóképpen járhatunk el akkor is, ha nem kész modulok felhasználásával, hanem néhány LSI és VLSI áramkörrel építjük meg a



3.6. ÁBRA. A hardvertervezés és bemérés folyamatábrája



3.7. ÁBRA. A sorrendi logika modellje

rendszerünket. Ilyen kis konfigurációkra az LSI és a VLSI gyártók kész „recepteket" közölnek az alkatelemekhez tartozó leírásokban, ezek alapján gyorsan megépíthető az egyszerű feladatok ellátását végző hardver. Természetesen mind a modulkészlet, mind pedig a „receptek" felhasználásakor komoly figyelmet kell szentelnünk a központi rész (mikroprocesszor, tárak, be-, ill. kimeneti csatornák) és a külvilág kapcsolatát megvalósító interfész megtervezésére.

A mikroprocesszorokhoz kapcsolódó be-, ill. kimeneti LSI és VLSI alkatelemek azonban az interfész megvalósítását is jelentősen megkönnyíthetik (l. a 6. fejezetet).

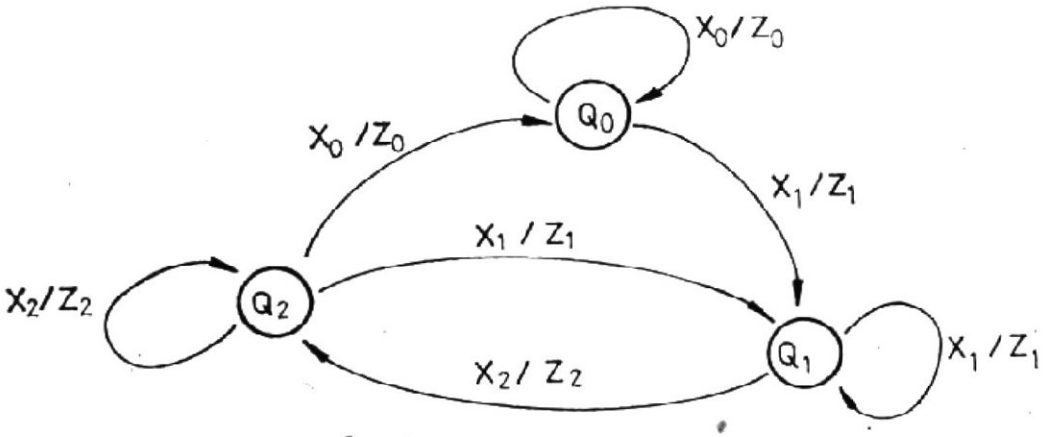
Bonyolultabb hardver tervezésekor a blokkok vagy modulok belső logikai kapcsolásának a kidolgozása is szükségessé válik. Ez esetben idődiagramok, igazságtáblázatok, állapotdiagramok és a minimalizálási eljárások segítségével oldható meg a feladat. A kombinációs logikai áramkörök igazságtáblázatának elkészítése és minimalizálása viszonylag egyszerű feladat. A sorrendi logikai áramkörök és a kombinációs áramkörök kapcsolata a 3.7. ábrán látható modell szerint képzelhető el. A modellen négy logikaiváltozó-csoportot tüntettünk fel:

$x_1, \ldots, x_n$ bemenő változók,
$y_1, \ldots, y_k$ előző állapotváltozók,
$Y_1, \ldots, Y_k$ jelen állapotváltozók,
$Z_1, \ldots, Z_m$ kimenő változók.

Az állapotváltozók kombinációihoz állapotok tartoznak. Az állapotokat, valamint az egyik állapotból a másik állapotba való átmeneteket állapotdiagramokkal ábrázolhatjuk. Két változatuk terjedt el: a Mealy-állapotdiagramok, és a Moore-állapotdiagramok. Az előbbire mutat példát a 3.8. ábra [8]. Itt a Q_0, Q_1 és Q_2 egy-egy állapot, amelyet az állapotváltozók (Y_i) kombinációi határoznak meg.

Az állapotdiagramok azt mutatják meg, hogy az állapotok és a kimenő jelek milyen függvénykapcsolatban vannak az előző állapotokkal és a bemenő változókkal. Az ábrán pl.: Z_1 és Q_1 a Q_0-nak és X_1-nek a függvénye (Z_1 a kimenő változók – $z_1 \ldots z_m$ – egyik kombinációját, X_1 pedig a bemenő változók egyik kombinációját jelenti). Az állapotdiagram felrajzolása után a kombinációs logikai áramkör működésére igazságtáblázat készíthető, melynek alapján az már könnyen megtervezhető.

A kombinációs és a sorrendi logikai áramkörök tervezését bővebben nem ismertetjük. A tervezési eljárásokat a [9], [10], [11]-ben részletesen tárgyalják. Megjegyzésként csak annyit fűzünk a tervezéshez, hogy LSI és VLSI áramkörök közül a



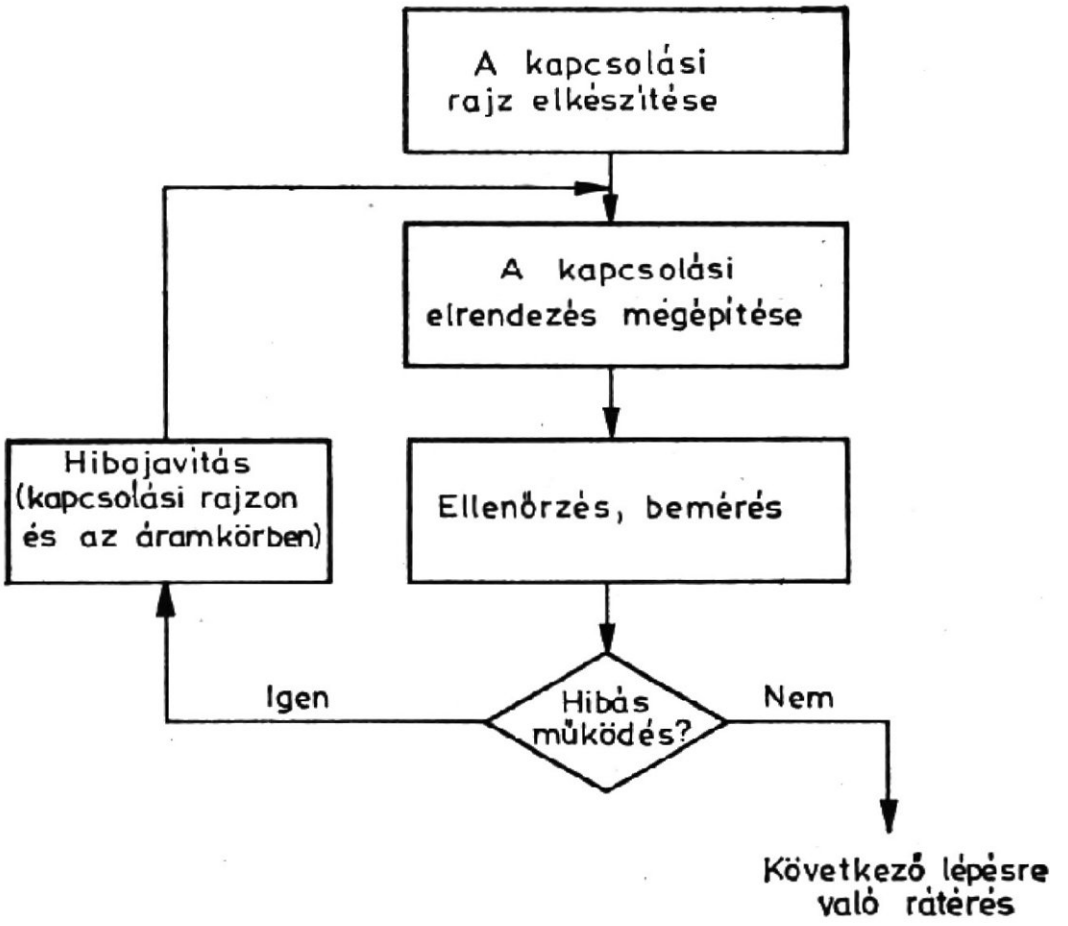
3.8. ÁBRA. Mealy állapotdiagram

PROM-mal vagy a REPROM-mal, ill. FPLA integrált áramkörökkel igen könnyen valósíthatók meg kombinációs logikai függvények [12]. A tárakban a címkiválasztó-áramkörök a címbitek valamennyi kombinációjához (mintermhez) egy rekeszt rendelnek. A megfelelő rekeszek (mintermek) adott bitjébe logikai *1* értéket égetve előállítható a REPROM-ban vagy a PROM-ban a címbemenetekre adott bemenő változók függvénye. Tehát a szóhossznak (bitszámnak) megfelelő számú függvényt képes így előállítani egy tárelem. Az FPLA eszközök nem teljes tárként is kezelhetők, azaz olyan tárként, melyben nincs meg minden címkombináció rekesze. Csak a beégetett címkombinációkhoz (AND-term) tartozik egy-egy rekesz. A használható kombinációk száma így ugyan viszonylag kevés, míg a bemenő változók száma viszonylag nagy (l. a 2. fejezetet).

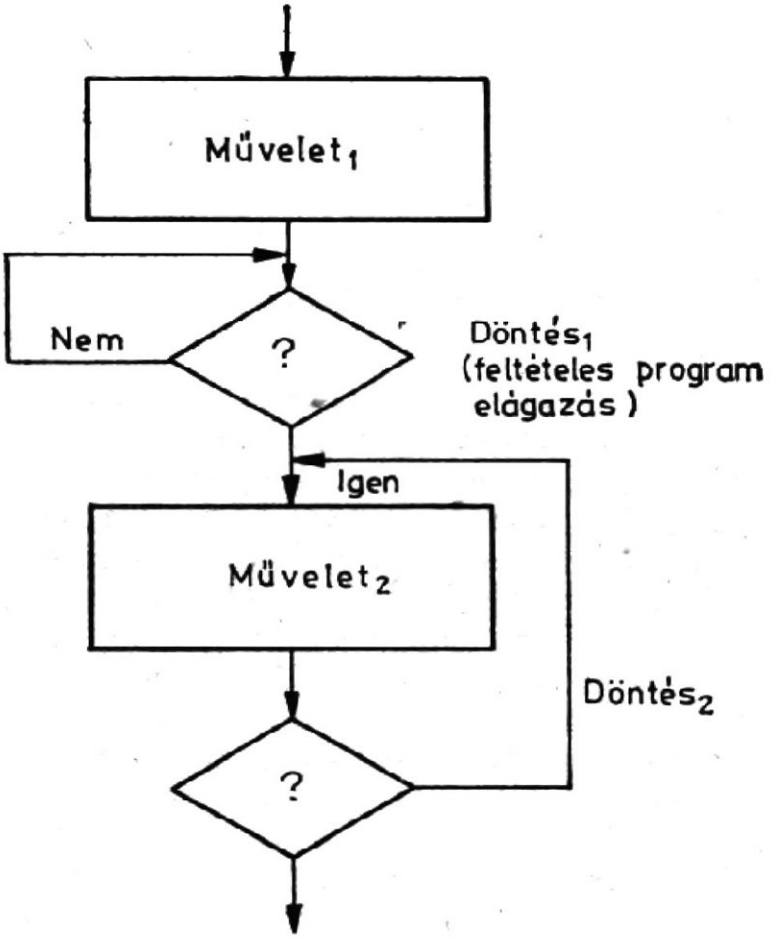
A kidolgozás során a hibakeresés és hibajavítás az egyik legtöbb energiát és időt igénylő lépés. A hardvermodulok, valamint a modulokból összeálló rendszer tervei kapcsolási rajzok formájában készülnek. A kapcsolási rajz szimbólumaiból (blokkok, logikai kapuáramkörök, alkatelemek s a közöttük lévő kapcsolat) összeálló ábra megkönnyíti a működés ellenőrzésekor az egyes elemek működésének a nyomonkövetését, vizsgálatát. A kapcsolási rajz alapján készíthető el a hardver (vagy annak egy modulja). Az elkészült hardver (prototípus) bemeneti pontjaira adott jelekkel (ezek a végleges működés közbeni bemenő jelváltásokat imitálják) működésbe hozható a hardver, ellenőrizhetők a kimenetén megjelenő válaszjelek, megfigyelhetők a belső állapotok, ill. állapotátmenetek. Az állapotok, ill. azok váltakozási sorrendje, valamint a kimenő jelek alapján a hardver működésének a helyessége is megállapítható. A hibás működés megszüntetésére mind a megépített hardverben, mind pedig (a későbbi gyártás alapját képező) kapcsolási rajzokon el kell végezni a szükséges módosításokat, majd a bemérés folytatható. A hibakeresés lépésének a kidolgozási folyamatban elfoglalt helyét a 3.9. ábrán láthatjuk.

A felhasználói programok kidolgozása

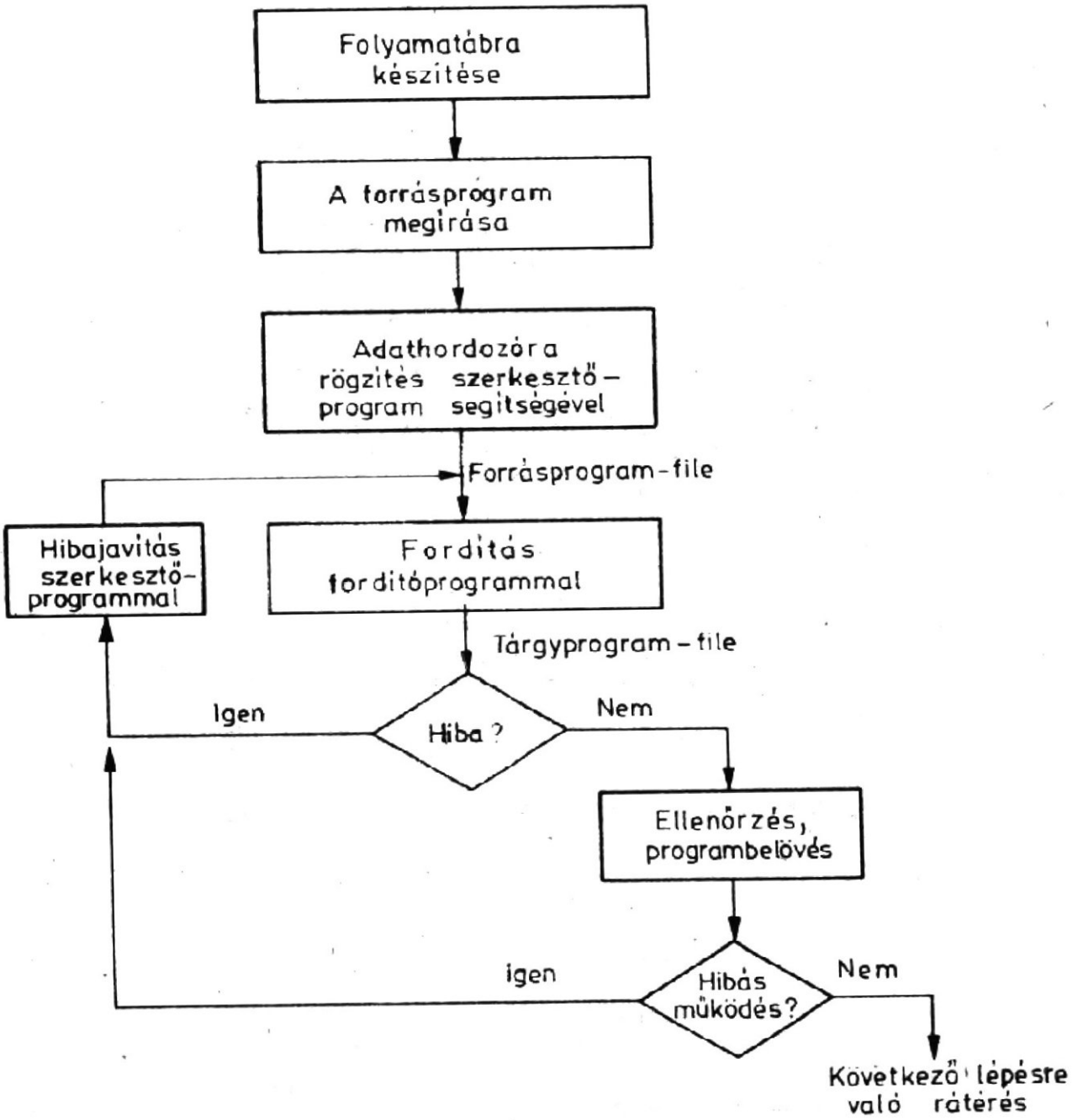
A mikroszámítógépes vagy a mikroprocesszoros rendszerek funkcionálásához működtetőprogramok szükségesek. Ezek a programok valósítják meg a működési algoritmust. A program utasításait kellő sorrendben végrehajtja a mikrogépes hardver, és az utasítások sorozatai által előírt műveletekből „áll össze", valósul meg a működési algoritmus. A működési algoritmust a tervezés során először – hasonlóan a hardver blokkokra bontásához – részekre bontjuk. A részekre bontás eredményeként felrajzolható a program folyamatábrája. A folyamatábra feltünteti az algoritmus részeinek egymás közötti kapcsolatát is. A programot: az utasítások sorozatát, azután már a folyamatábra alapján írhatjuk meg. A 3.10. ábra egy tipikus folyamatábra-részletet mutat be. A folyamatábra szimbólumai: a művelet jelölésére szolgáló téglalap, és a dön-



3.9. ÁBRA. A hardver megépítésének és bemérésének a folyamatábrája



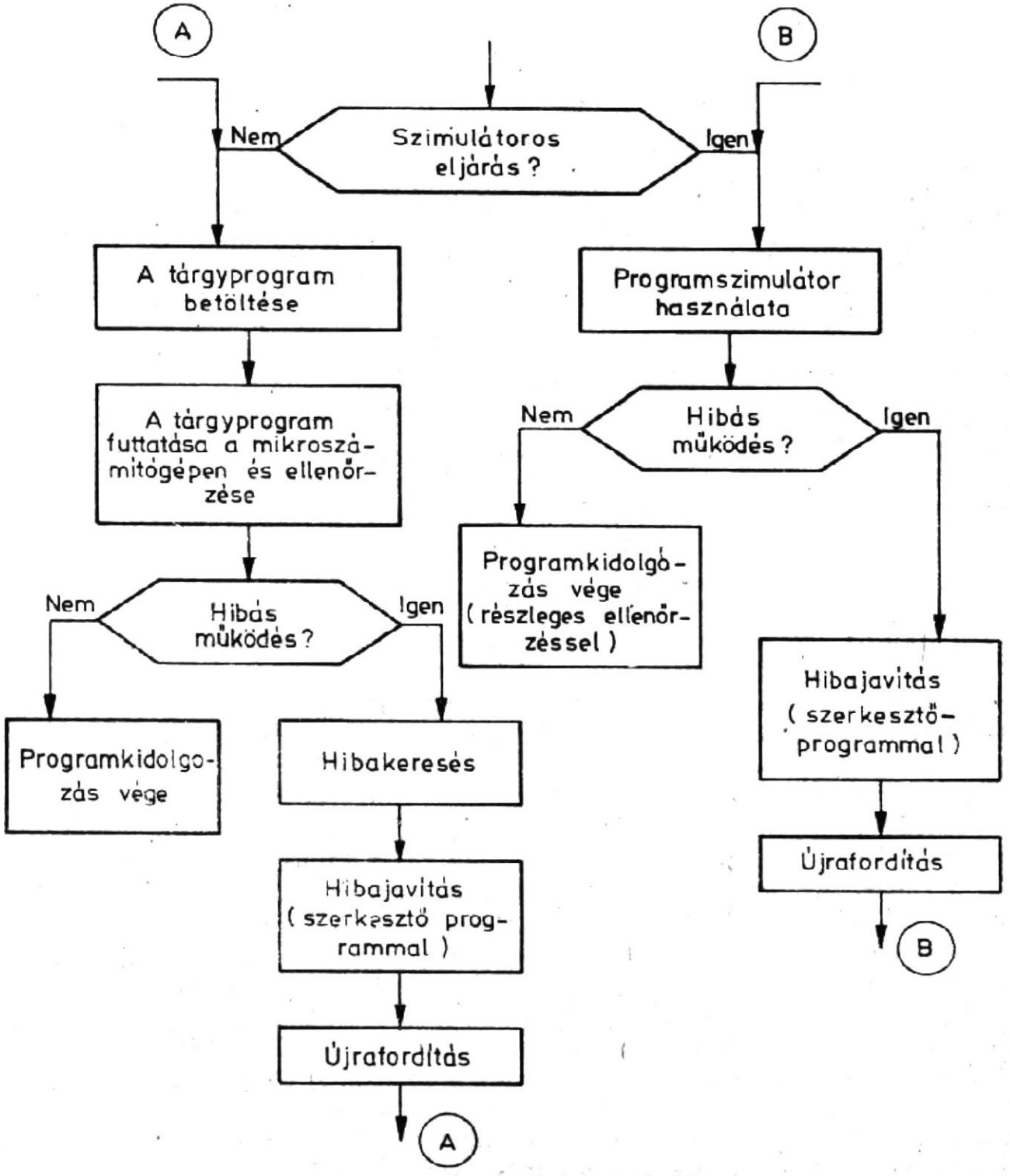
3.10. ÁBRA. Program-folyamatábra részlete



3.11. ÁBRA. A programok készítésének folyamata

téshozatalt jelző rombusz vagy hatszög. A művelet megnevezését, valamint a döntés feltételét a szimbólumokba írhatjuk be. A műveletnek egy kilépési pontja van, míg a döntéshozatal rombuszát több kimenettel látják el, megjelölve a kimenetekben a feltételnek azt az értékét (logikai IGAZ-igen, HAMIS-nem, vagy aritmetikai pozitív, negatív, esetleg 0 érték), amelyikhez a kérdéses kilépési pont tartozik.

A folyamatábra alapján a program utasítások sorozatára bontható. A megírt utasítások (akár assemblert, akár magas szintű programnyelvet használunk) alapján készül el a forrásprogram. A forrásprogram írásához és adathordozóra rögzítéséhez a számítógép, ill. annak programja nyújt segítséget. Ezt a programot szerkesztőprogramnak nevezzük. A forrásprogramfile-t (adathordozón rögzített forrásprogramot) fordítóprogrammal lefordítva megkapjuk a tárgyprogramot (3.11. ábra). A tárgyprogram mellett a fordítóprogram hibajegyzéket is készít a forrásprogramban található szintaktikai hibákról, és kérésre programdokumentációt (listát) is ad. Assembler nyelvű forrásprogram fordítására az ún. assembler programot használjuk, míg a magas szintű programnyelvek fordítóprogramjának a neve: compiler (l. a 9. és a 10.



3.12. ÁBRA. A programellenőrzés menete

fejezeteket). A szintaktikai hibák kijavítása és a forrásprogram újrafordítása után a tárgyprogram (gépi kódú program) futtatható ellenőrzés és a tervezési hibák megkeresése céljából.

A program ellenőrzésére és a hibák megtalálására két alapvető eljárást használnak. Az egyik a prototípus vagy a prototípust emuláló fejlesztési segédeszköz használatán alapul, a másik a szimulátoros eljárás. Az ellenőrzés és hibakeresés folyamatát mindkét eljárásra vonatkoztatva a 3.12. ábra mutatja. A szimulációs eljárás hátránya, hogy a nem azonos idejű próbák miatt a programok ellenőrzése nem teljes, a felhasználói rendszeren futtatva, a szimulátorral bemért tárgyprogram működésében gyakran újabb hibák léphetnek fel. A hibakeresés módszereivel és eszközeivel bővebben a 8. fejezetben foglalkozunk.

A hardver- és a programkidolgozás módszerei között több rokon vonás figyelhető meg. A tervezés és hibakeresés menetének hasonlóságán túl a hardver állapotdiagramja és a programok folyamatábrái között is szoros kapcsolatot fedezhetünk fel. A 3.8. ábra állapotdiagramján egy-egy állapotnak a 3.10. ábra folyamatábráján levő egy-egy műveleti téglalap (az „utasításról utasításra" való normál programvégrehajtás) felel meg. A hardverállapotok közötti átmenet a bemenő változók és az előző állapotok függvénye. A program a döntéshozatal során (folyamatábrán rombusz vagy hatszög) vizsgálja meg a program belső állapotjelzőit, ill. a program vagy programrész bemenő változóit, s a döntés, esetleg több döntés eredményétől függően egy új állapotba (folyamatábrán téglalap) lép a program. A döntéshozatal tehát az állapotátmenetek megfelelője [8].

A hardver és a szoftver együttes próbái

A fejlesztési folyamat legkritikusabb pontjához akkor érkezik el a felhasználó, amikor a programját vagy programmoduljait betölti a hardverrendszerbe és megkezdi az azonos idejű működés ellenőrzését. Ebben a fázisban különösen fontos, hogy az azonos idejű működést megfelelően „mérő" és nyomon követő eszközök és módszerek álljanak rendelkezésre. Ezek, anélkül, hogy a felhasználói rendszer működését megzavarnák, lehetővé teszik, hogy az ellenőrzést végző mérnök vagy programozó pontosan megfigyelhesse és megvizsgálhassa, hogy az „összeintegrált" hardver és szoftver hogyan működik. Az ilyen eszközöket és a használatukkal kapcsolatos hibakeresési módszereket a 4. fejezet részletesebben ismerteti.

A hibakeresés során csakúgy, mint a tervezés során a legcélravezetőbb út a rendszeres ellenőrzés. A szisztematikus hibakeresés első lépése a meghibásodás eredetének a behatárolása, a gyanús hardver vagy programrészek körének a szűkítése. Mind a felhasználói hardver, mind pedig a programok funkcionális blokkokra bonthatók. A behatárolás a hibásan működő blokk megkeresésével kezdődik.

Tipikus digitális rendszerekben a funkcionális blokkokat átviteli sínek kötik össze (l. a 7. fejezetet). A rendszer működéséről és állapotáról képet kaphatunk a síneken lezajló átviteleknek és a sínek vezérlőjeleinek, azok időbeni változásának a megfigyelésével. Ily módon sok probléma behatárolásához nyújthat segítséget a sínek vizsgálata. Néhány hardver- és szoftverműködési problémát és zavart sorolunk fel a 3.2. táblázatban. A táblázatból láthatjuk, hogy milyen típusú sínek megfigyelésével lokalizálhatók ezeknek a hibáknak vagy zavaroknak a forrásai [13].

A hardver- vagy szoftver-blokkok szintjén behatárolt hibaforrás további kereséséhez már a blokkon belüli működés vizsgálata szükséges.

Hardverblokkon belül az alkatelemek bemeneti és kimeneti pontjain megfigyelhető jelek analizálásával juthatunk el a hiba eredetéhez. Mikroprocesszoros rendszerek fejlesztése esetén gyakrabban tervezési hibából ered a helytelen működés. Az áramkör újratervezésével és átépítésével a tervezési hiba megszüntethető. Gyártmányok esetén, s ritkábban a fejlesztés során is, hibás alkatrész vagy az alkatrészek rossz összeköttetése (nyomtatott áramköri huzalozás, áramköri kártyák közötti huzalozás) is megzavarhatja a helyes működést. A rossz alkatrészek cseréjével vagy az összeköttetés megjavításával a hiba elhárítható.

Programhibák behatárolásához a programmodulon, programblokkon belüli utasítások vizsgálatára van szükség. A programhibák is lehetnek tervezési hibák, ritkábban hardvermeghibásodási zavarok (pl.: meghibásodott tárolóeszközök) is okozhatnak helytelen programműködést. A tervezési hibák megszüntetése a forrásprogram újraszerkesztésével és újrafordításával, ritkábban közvetlenül a tárgyprogram javításával érhető el.

A mikroprocesszoros rendszerekben bizonyos hardverfunkciók gyakran nem

3.2. TÁBLÁZAT. A sínek vizsgálatával behatárolható hibák

A hibás működés, zavar megnevezése	A hibával vagy működési zavarral kapcsolatos sín
Rövid idejű zavarok, zajok, vezetékek közötti áthallások, időzítési problémák	Vezérlő jelek sínje
Programeredetű zajok, program-„párbeszéd"- és időzítési problémák	Vezérlőjelek sínje, Címvezetékek
Algoritmushibák, programciklusba „ragadás", kihagyott programrészek, programfutási idővel kapcsolatos problémák	Címvezetékek
Adatfüggő hibás működés, algoritmushibák	Adatsín, Címvezetékek
Adatátviteli problémák	Adatsín, Perifériák csatlakozási sínjei
Be-, ill. kiviteli zavarok „párbeszéd"- és multiplexelési problémák, rövid idejű zajok	Be-, ill. kimeneti interfészek sínjei
Soros-párhuzamos átalakítási problémák	Be-, ill. kimeneti interfészek sínje, Soros átviteli vonal
A—D és D—A konverzióval kapcsolatos zavarok	A konverterek és a „külvilág" közötti sín, Be-, ill. kimeneti interfészek sínje

különíthetők el a működtetőprogram-algoritmustól, vagyis sem a hardver, sem a programok nem vizsgálhatók meg önállóan a funkciók összességére kiterjedően. Ezért nem különítjük hardver és szoftver hibakeresési eljárásokra a bemérési, rendszerellenőrzési módszereket, melyeket a következőkben röviden ismeretetünk.

A mikroprocesszoros rendszerek fejlesztéséhez, a helytelen működés okainak a megtalálásához az alábbi legfontosabb módszerek használatosak:

- a lépésenkénti programvégrehajtás módszere,
- az ellenőrzött programfuttatás módszere,
- a nyomkövetés módszere,
- az áramköri emuláció módszere.

Az első három módszer lényege, hogy a nagysebességű mikroprocesszoros készülék működésének valamelyik szakaszát emberi értékelés céljából megfigyelhetővé tegyék. A legkisebb szakaszok, az utasítások, sőt bizonyos esetekben az utasításokon belüli gépi ciklusok megfigyelésére a *lépésenkénti végrehajtás módszere* nyújt lehető-

séget. Lényege, hogy minden utasítás vagy gépi ciklus után leállítjuk a felhasználói rendszert, vagyis annak a működését, így azt az ember észlelési és vizsgálati sebességéhez lassítjuk le. Ezzel alkalom nyílik mind a hardverállapotoknak, mind pedig a programfunkcióknak a megfigyelésére. Az *ellenőrzött programfuttatás* módszerével nagyobb szakaszokhoz tartozó funkciók vizsgálatára (gyorsabb vizsgálatra) nyílik lehetőség. A vizsgálandó szakaszok határainak a megadására különböző hardver vagy szoftverparaméterek használatosak. Leggyakoribb paraméter a programszakasz kezdetét jelentő utasítás, valamint a végét jelentő utasítás címe. Míg a lépésenkénti végrehajtás során a működés nem azonos idejű, az ellenőrzött programfuttatás módszerével már a programszakaszon belül azonos idejű működés érhető el. A *nyomkövetéssel* (trace) azonos idejű működés közben is (nemcsak a programszakasz befejezése után) adatokat gyűjthetünk a rendszer viselkedéséről. Ezek az adatok az ún. nyomkövetési tárba kerülnek futás közben, leállás után pedig onnan – már az emberi észleléshez és megfigyeléshez alkalmas, tetszőleges ütemben – kiolvashatók. Ezzel a gyors működés közbeni funkciók is értékelhetők.

Az *áramköri emuláció módszere* az LSI és a VLSI eszközök alkalmazásával jelent meg. Segítségével a felhasználói áramkörön belül emulálható a mikroprocesszor, s esetenként a felhasználói rendszer egyéb részei (tár, interfészek) is. A módszert megvalósító fejlesztési segédeszközök helyettesítik (emulálják) a felhasználói rendszer megfelelő LSI és VLSI alkatelemeit, lehetővé téve egyrészt a rendszer azonos idejű működését, másrészt pedig azt, hogy az emulált alkatelemek belső adataihoz, állapotához a működés felfüggesztése után a felhasználó értékelés és módosítás céljából hozzáférjen. További előnye ennek a módszernek az, hogy segítségével kipróbálhatók a felhasználói rendszer megépített hardverrészei és megírt programrészei is a teljes rendszer elkészülte előtt. Így a fokozatos rendszerépítés és ellenőrzés is lehetővé válik, megkönnyítve ezzel a fejlesztési tevékenységet.

A felsorolt négy módszer hierarchikus kapcsolatban van egymással. Általában a felsorolás fenti sorrendjében mindegyikük feltételezi és magába foglalja az előzőeket. Az áramköri emuláció nyújtotta belső adatellenőrzés és -módosítás akkor szükséges, ha az ellenőrzött programfuttatási eljárást alkalmazva a rendszer működését éppen felfüggesztette a bemérést végző mérnök vagy programozó. A lépésenkénti programvégrehajtás pedig részesete az ellenőrzött programfuttatási eljárásnak. Talán az egyedüli elvi kivételt a nyomkövetés jelenti, mert ezt nem feltétlenül foglalja magába az áramköri emuláció. A gyakorlatban azonban az áramköri emulátoros fejlesztői eszközök legtöbbje alkalmas a nyomkövetés módszerének a megvalósítására.

Az említett módszerek mindegyikét részletesen ismerteti a 4. fejezet, amelyben a különböző fejlesztői segédeszközöket tárgyaljuk.

A hardver-szoftver együttes próbáinak és a gyártás, valamint a karbantartás közbeni ellenőrzéseknek hasznos eszközei a tesztprogramok. Tesztprogramokkal a felhasználói rendszerek egyes részeinek a funkciója és a teljes felhasználói rendszer is ellenőrizhető. A tesztprogramok használatának két előnye van (a funkciók végső „működik/nem működik" jellegű próbájával szemben). Ezek:

1. A fejlesztésben részt vevő mérnökök igényei alapján olyan tesztek készíthetők, amelyek a hardver (esetleg szoftver) „kényes" pontjait és részeit kedvezőtlenebb körülmények között is ellenőrizhetik, mint amik az alkalmazás során előállnak. Az ilyen tesztek a működés helyességéről többet mondanak, mint a „működik/ /nem működik" próbák.
2. Megfelelően felépített tesztek megkönnyíthetik a hibabehatárolást. A működik/ /nem működik próbákban az egész felhasználói rendszer részt vesz, helytelen működés esetén sokszor hosszas hibabehatárolási munkákkal deríthető csak ki a meghibásodás oka. A jól felépített tesztek először csak egy-egy rész (blokk, modul

működését ellenőrzik, lehetőleg nem érintve a többi részt. Majd a már ellenőrzött és jónak bizonyuló modulokat is bevonva letesztelik a teljes rendszert. A tesztprogram hibát észlelve jelzi, vagy utal arra, hogy hol kell keresni a hibás működés eredetét.

IRODALOM

[1] Design and development. Infotech state of art report, Microprocessors, 1. Analysis and bibliography, 1977. 193–214.

[2] *Soucek, B.:* Overview: New trends in microprocessor system development and application. Infotech State of art report, Microprocessors, 2. 1977. 243–262.

[3] *Kummel, L.–Schultz, G.:* Advances in microcomputer development systems. Computer, Febr. 1977. 13–18.

[4] *Teja, E.:* The single-board-μC alternative expands OEM design horizons. EDN, Oct. 20, 1979. 257–290.

[5] *Hartenstein, W.:* LSI chip design: from evolution to revolution? Proceedings of μP'79 OMKDK Technoinform, Budapest, 1979. 1–7.

[6] *Schindler, M.:* Computer-based systems design–many decisions, few guidelines. Electronic design, May 24, 1979. 57–83.

[7] *Gibbons, J.:* When to use higher-level languages in microcomputer-based systems. Electronics, Aug. 7, 1975. 107–111.

[8] *Lin, W. C.:* Microprocessor-based digital system design fundamentals and the development laboratory for hardware designers and engineering executives. Proceedings of the IEEE, Aug. 1977. 1138–1161.

[9] *Carey, J.:* Formal Methods expedite microprocessor based system design. Computer Design, Feb. 1977. 97–105.

[10] *Janovics S., Tóth M.:* A logikai tervezés módszerei. Műszaki Könyvkiadó, Budapest, 1973.

[11] *Chu, Y.:* Digitális számítógépek tervezésének alapjai. Műszaki Könyvkiadó, Budapest, 1966.

[12] *Petrale, J. J.:* PROM controller makes fast work of serial jobs. Electronics, Apr. 12, 1979. 134–138.

[13] *Эрени И.:* Обзор проблем и состояния разработок, направленных на применение микропроцессоров. Совещание Академий Наук социалистических стран, Дрезден, ГДР, сентябрь, 1978.

[14] *Marshall, J.:* Now choosing the right logic analyzer reguires a logical approach. Electronic Design, Nov. 22, 1978. 108–115.

[15] M 6800 microprocessor application manual. Motorola Semiconductor Products Inc., 1975.

[16] *Soucek, B.:* Microprocessors and microcomputers. Wiley-Interscience Publication, New York, 1976.

[17] Sotfly, softly... Systems International, Jan. 1980. 19–21.

[18] *Beeson, C. E.:* Software minimizes multiple output Boolean functions. Computer Design, Jan. 1979. 118–123.

[19] *Lee, E.:* When programing microprocessors, use your hardware background. Electronics, July 8, 1976. 93–100.

[20] *Chung, D. H.:* Four design principles get the most out of microprocessor systems. Electronics, Jan. 20, 1977. 102–110.

4. A MIKROPROCESSZOROS RENDSZEREK FEJLESZTÉSÉNEK ESZKÖZEI

Az előzőekben láttuk, hogy a programozott LSI és VLSI alapú készülékek, berendezések és rendszerek fejlesztése, kidolgozásuk sajátságos problémákat vet fel. E problémák megoldásához részben a régi, részben pedig egészen új segédeszközök és módszerek állnak a felhasználó rendelkezésére. Ebben a fejezetben áttekintjük azokat a készülékeket, műszereket és berendezéseket, valamint szoftver segédeszközöket, amelyek igen hatékonyan alkalmazhatók a fejlesztési munkák során. A segédeszközök egy részénél a hardver működése olyan szorosan kapcsolódik a programfunkciókhoz, hogy egymástól való elkülönítésük ugyanolyan nehézségekbe ütközik, mint sok felhasználói rendszer esetében. Ez különösen igaz azokra a segédeszközökre, amelyek mikroprocesszor vezérlésűek.

A fejlesztési munkák szakaszainak megfelelően a fejlesztési segédeszközöket osztályozhatjuk:

- a hardver kidolgozását elősegítő,
- a programok kidolgozását segítő,
- az együttes próbákat segítő eszközökre és
- az integrált rendszerekre.

Ez utóbbi segédeszközök, az integrált rendszerek a fejlesztés kezdeti szakaszaitól a végső próbákig hatékonyan előmozdítják mind a felhasználói hardver, mind a programok kidolgozását, ellenőrzését.

A hardverkidolgozást elősegítő fejlesztési segédeszközök feladata egyrészt az, hogy a hibabehatárolást, hibakeresést hatékonyabbá – ill. az esetek egy részében egyáltalában megoldhatóvá – tegyék, másrészt pedig az, hogy a felhasználói rendszer építése során beépíthető részekként szolgáljanak. A rendszerbe való beépítés célja kettős lehet: a segédeszközöket vagy építőelemekként a végleges berendezésbe építik be, vagy pedig csak ideiglenes jelleggel kapcsolódnak a rendszerhez a fejlesztés egyes időszakaiban. Az ideiglenes összekapcsolás célja az, hogy a felhasználó részenként is ellenőrizhesse a már megépített berendezését úgy, hogy az elkészült részt kiegészíti működőképes és ellenőrizhető, megvizsgálható berendezéssé. A hibakeresés eszközei közé sorolhatók [1]: egyrészt a hagyományos műszerek (pl.: oszcilloszkópok), amelyek vezérlő-, adat- vagy egyéb jelek vizsgálatához, rövid idejű jelváltozások, zajok megfigyeléséhez kitűnően alkalmazhatók, vagy a logikai jelszint-kijelzők és számlálók. Ezeknek az eszközöknek a közös tulajdonsága az, hogy egy vagy két jel megfigyelésére, egymáshoz való idő- és szintviszonyaiknak az ellenőrzésére használhatók. A következő csoportot a párhuzamos sínek, adat- és címvezetékek állapotainak a megfigyelésére, kijelzésére használatos eszközök alkotják. Közös jellemzőjük, hogy széles, sokvezetékből álló sínek jeleinek, párhuzamos adatoknak és kódolt logikai értékeknek a megfigyelésére is alkalmasak. Közülük az egyszerűbbek csak annyiban teszik megfigyelhetővé a síneken, a blokkok és áramköri részek között lezajló jelváltások, jelek sorozatát, hogy a működést lelassítják az ember számára érzékelhető megfelelő sebességűre (lépésenkénti műveletvégrehajtás). A bonyolultabbaknak már tárolójuk (jelanalizálási anyagot memorizáló eszközük) is van, elvégzik a gyors mű-

4.1. TÁBLÁZAT. A fejlesztési segédeszközök és felhasználási területeik

Segédeszköz	Felhasználás		
	Rendszerbe építéshez	Hiba-kereséshez	Programok írásához
Egyszerű műszerek, oszcilloszkópok	nem	igen	nem
Logikai jelek kijelzői	nem	igen	nem
Impulzusszámlálók	nem	igen	nem
Egyszerű sínvezérlők és logikai sinek kijelzői	igen	igen	nem
Logikai komparátorok	nem	igen	nem
Mikroszámítógép-építő modulok	igen	részben	részben
Egyszerű mikroprocesszoros fejlesztői eszközök	részben, ideiglenesen	főként programhibák kereséséhez	igen
Mikroprocesszoros integrált fejlesztői rendszerek	ideiglenesen a fejlesztés során	igen	igen
Egyszerű miniszámítógépes keresztrendszerek	nem	programhibák kereséséhez	igen
Assemulátoros minigépes rendszerek	nem	igen	igen
Logikai állapotanalizátorok	nem	igen	nem
Mikroprocesszor- és jelanalizátorok	nem	igen	nem
Kis-, közép- és nagyszámítógépes, szimulátoros programfejlesztői rendszerek	nem	programhibák kereséséhez	igen

ködés közbeni adatgyűjtést, majd a begyűjtött adatokat elemzésre kijelzik a hibakeresést végző mérnök vagy programozó számára.

Természetesen számítógép és megfelelő szoftver szükséges ahhoz, hogy a felhasználói programok kidolgozását is el lehessen végezni. A fejlesztői szoftverrendszerekről a későbbiekben még bővebben írunk. Itt arra kívánjuk felhívni a figyelmet, hogy a szoftverhibák megkereséséhez általában ugyanazokat a segédeszközöket veszik igénybe, amelyek a (programvezérelt) hardver beméréséhez is használatosak. Ez érthető is, hiszen a programok, vagyis a programokat képező utasítások sorozatának a vizsgálatát az adat-, a cím- és a vezérlősíneken megjelenő jelváltások figyelésével végezhetjük el. Talán az egyedüli kivétel a szimulációs módszer, a szimulátorprogramok alkalmazása. A szimulációval később önálló részben foglalkozunk még.

A 4.1. táblázatban foglaltuk össze a fejlesztési segédeszközöket, röviden ismertetve azt is, hogy a tervezés, rendszerépítés, hibakeresés során milyen célra használhatók fel.

A továbbiakban részletesebben tárgyaljuk a táblázatban felsorolt segédeszközöket.

4.1. EGYSZERŰ ESZKÖZÖK

Mérőműszerek, oszcilloszkópok

A mérnöki gyakorlatban igen nagy múltra tekinthet vissza az egyszerű műszerek, valamint az oszcilloszkópok használata. Természetesen a mikroprocesszoros berendezések létrehozása során is jól használhatók. Az egyszerűbb műszerek bizonyos hibák (jelszintek értékei, rövidzárlat jellegű meghibásodások stb.) megkereséséhez, a tápfeszültségek szabályozásához nyújthatnak komoly segítséget.

Az oszcilloszkópok használata nem kerülhető el a bonyolultabb programozott logikai berendezések, számítástechnikai rendszerek beméréseknél sem. Az igen gyors jelek váltásának, a nagyfrekvenciás zajösszetevőknek, a tranziens állapotok időviszonyainak a megfigyelését szinte kizárólagosan csak az oszcilloszkópok teszik lehetővé. E műszerek korszerű típusaihoz alkalmazott megoldások közül kettőt szeretnénk kiemelni igen hasznos hibakeresési lehetőségként. Az egyiket a kettős időalap, a másikat pedig a tárolócsövek adják.

Ismeretes, hogy a programvezérelt eszközök, valamint a kódolt jelátvitel nagyban megnehezítik a jelek megfigyelését. Programvezérelt működés során gyakran nehéz ciklikus, kellő gyakoriságú jeleket előállítani, valamint a katódsugár eltérítéséhez szükséges szinkronjelet találni. A megfelelő szinkronjel kiválasztását tovább nehezítik a kódolt utasítások, parancsok. Gyakori feladat a viszonylag rövid idejű, a szinkronjelhez képest nagy késéssel fellépő jeleknek a megfigyelése. Ezeket a nehézségeket jelentős mértékben csökkenthetik a korszerű oszcilloszkópok.

Kettős időalapú készülékkel megfigyelhetők a szinkronimpulzustól távol eső rövid jelek is. Hasonlóan jól megfigyelhetők a szinkronizálás után létrejövő jelsorozatok jelei és azok jelalakjai is, akkor is, ha e jelek időtartama sokszorosan kisebb, mint az eltérítési ciklusidő. A kettős időalap ui. lehetővé teszi, hogy a képernyőn megjelenő ábra egy részét – a vizsgált részt – az alapidőtől eltérően más időléptékben figyeljük meg.

A tárolócsöves oszcilloszkópok a nagyon ritkán (pl.: csak egyszer) előálló jelváltások megfigyelésének kitűnő eszközei. Megfelelően indítva felrajzolják a képernyőre a megvizsgálni kívánt diagramot, és azt a felhasználó által meghatározott ideig kijelzik.

Logikai jelszintkijelzők

A jelszintkijelzők kapcsán három eszköz használatát ismertetjük. Ezek a:

- logikai jelszintkijelző,
- logikai impulzusadó,
- áramkijelző-kereső [2]...[5].

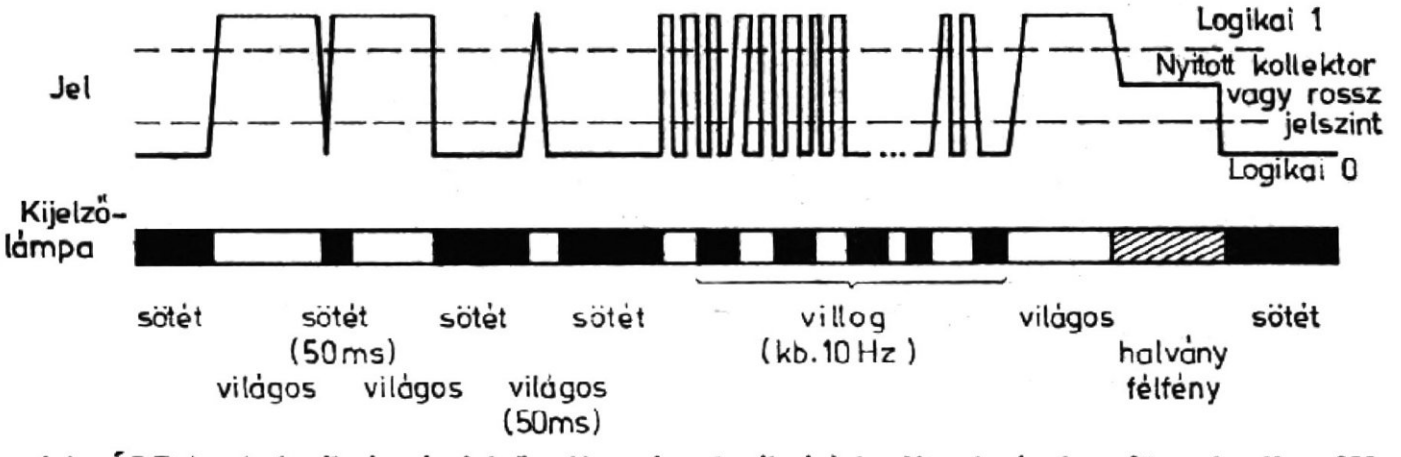
A három eszköz közül leggyakrabban és önállóan is használatos a jelszintkijelző, ritkábban az áramkijelző-kereső. Az impulzusadót a másik két kijelzőeszközzel együtt célszerű használni. Ezek az eszközök hibakeresésre kis méretük miatt különösen alkalmasak. Általában töltőtoll formájú alakban készülnek, hegyükkel kell megérinteni a vizsgálandó pontot. Kijelzésre apró beépített lámpát vagy fénykibocsátó diódát használnak. Kis méretük ellenére viszonylag bonyolult funkciókat is képesek ellátni.

A logikai jelszintkijelző alkalmas logikai *0* vagy *1* jelszintek és közbülső állapotok (illegális szint vagy nagy impedanciájú pont) kijelzésére. Egy vagy két fényforrás jelzi a szinteket, egy fényforrás esetében a fényerőváltozás is logikai állapotot tükröz. Rövid időtartamú jelek megfigyelésére, ill. kijelzésére ember számára is észlelhető fényvillanást generálnak. A 4.1. ábrán mutatjuk be egy szintjelző eszköz működését. A logikai jelszintkijelzők közül a korszerűbbeket impulzusmemóriákkal, ill. – számlálókkal is ki lehet egészíteni. Ezek a gyors egymásutánban érkező impulzusokat megjegyzik, s kérésre impulzusszámot közölnek, vagy az impulzussorozatra jellemző felvillanásokkal jelzik a felhasználónak a memorizált sorozatot.

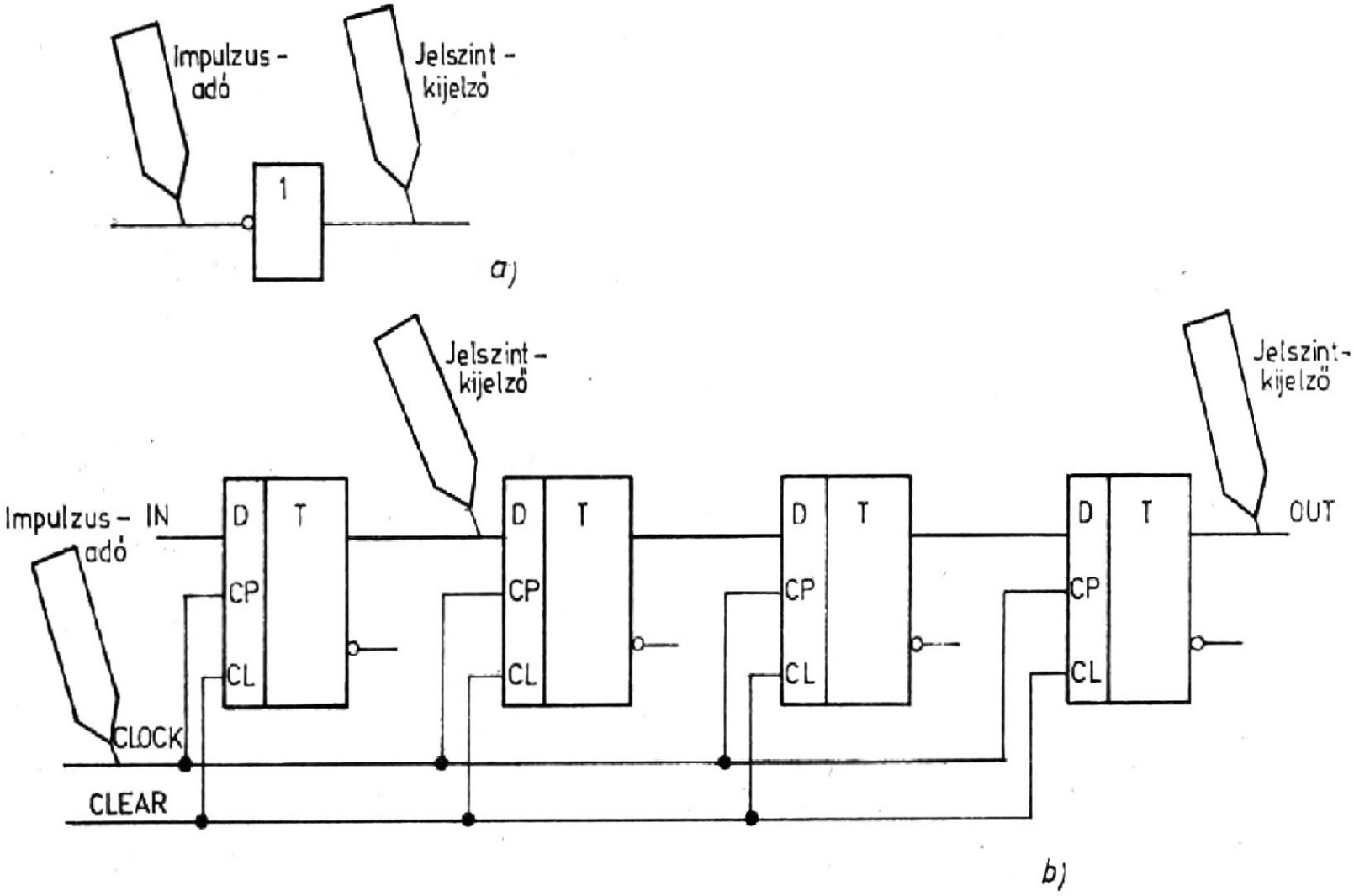
A logikai jelszintkijelzők típustól függően TTL, CMOS és DTL áramkörökhöz alkalmazhatók, de van olyan típus is, amely mindezen áramkörökhöz használható.

A logikai impulzusadóval digitális árakörök ki- vagy bemenetei gerjeszthetők. Az áramköri kapcsolás megbontása nélkül, az impulzusadót a kivezetések egyikére helyezve, akár logikai *1*, akár logikai *0* jelszintet kényszeríthetünk az adott pontra. A válaszjeleket megfigyelve (pl.: szintjelzővel) megvizsgálható, hogy a vizsgálat tárgyát képező kapuáramkör jól működik-e. Az impulzusadó kimenetén kellő teljesítményű áramjel generálódik, ez azonban annyira rövid idejű, hogy az áramköri elemeket nem teszi tönkre. Korszerűbb változataik több üzemmódban is képesek működni, így pl.: egyetlen impulzus, impulzussorozat különböző frekvenciával stb. kibocsátására is képesek. Hasonlóan az előző segédeszközhöz TTL, CMOS és DTL áramkörökhöz alkalmazhatók, de van olyan típus is, amely az említett logikai áramkörök bármelyikéhez is használható. Az impulzusadó használatát a 4.2. ábrán mutatjuk be.

Az áramkijelző-kereső feladata, hogy a nyomtatott áramköri kártyákon az alkatelemek kiszerelése és kapcsolásuk megbontása nélkül is lehetővé tegye az áramutak, vezetékek végigkövetését. Nagyon sok hibakeresési funkció célja a technológiai rövid-



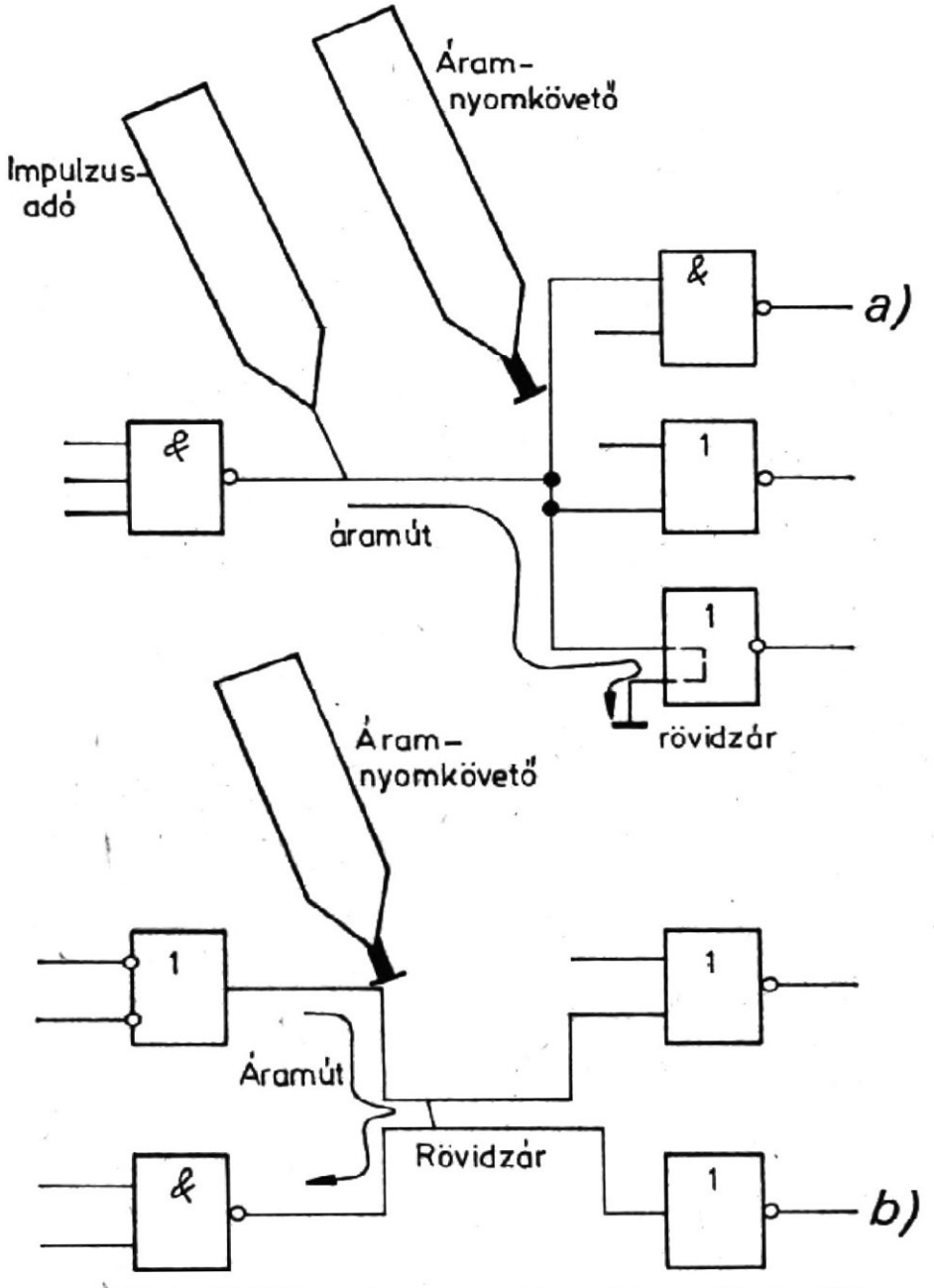
4.1. ÁBRA. A logikai szintjelző válaszai a logikai jel változásainak a függvényében [2]



4.2. ÁBRA. Az impulzusadó használata *a)* kapuáramkör és *b)* léptetőregiszter működésének a vizsgálatára [2]

rezárások megkeresése. Hagyományosan az ilyen munkák hosszadalmasak, nem egyszer sok elem eltávolítása után található csak meg a hiba helye. Az áramnyomkövetővel sokkal könnyebben, gyorsabban juthatunk el célunkhoz. Működési elve a következő: az áramúttal galvanikus kapcsolatot nem létesítve érzékeli az áram mágneses terét, s azzal arányos intenzitású fényjelet bocsát ki. A meglevő eszközök igen széles tartományban (pl.: 1 mA-től 1 A-ig és 10 MHz sebességű áramimpulzusokig) képesek működni. Az áramutat a gyanús vezetékek végigkövetésével lehet megkeresni. Az áram azonos intenzitása jelenti az áramutat, az intenzitás megváltozása azt jelenti, hogy letértünk az áramútról. A 4.3. ábrán két példát mutatunk be az áramnyomkövető használatára. Az *a* esetben az áramjelet egy impulzusadó generálja, a rövidzárat pedig egy logikai kapuáramkör meghibásodása okozta. A *b* esetben a nyomtatott áramkör egyik vezetéke összeér egy másik vezetékkel, ez okozza a hibát. Ilyen esetben a kapcsolás egyik aktív kapukimenetén áramjel van.

Az impulzusadók használatához szeretnénk egy gyakorlati tanácsot adni. Tekintve, hogy rövid idejű, de nagy teljesítményű jelet adnak, tápfeszültség-ellátásukat célszerű körültekintően megválasztani, lehetőség szerint a felhasználói berendezés tápegységétől különálló tápegységről kell megoldani.



4.3. ÁBRA. Példák az áramnyomkövető használatára [2]

Impulzusszámlálók

Ezek az eszközök a logikai jelszintkijelzők bonyolultabb változatai. Néhány logikai szintkijelzőnek számlálója is van, amely képes valamely impulzussorozat jeleit összeszámlálni és kijelezni. A drágább változatok az impulzusmemóriával kiegészített kijelzők. Az impulzusmemóriák használatával nemcak a jelek számát, hanem a szomszédos jelek közötti időviszonyokat is lehet érzékeltetni [6].

Sínvezérlők és -kijelzők

A sínkijelzők közül a legegyszerűbb, de széleskörűen alkalmazott típus az integrált áramkörökre csíptethető kijelző. Ez a kijelző tápfeszültséget a rácsíptetés révén kap. Egyidejűleg fénykibocsátó diódával jelzi az integrált áramkör minden kivezetésén a logikai jelek értékét. Bonyolultabb változataik a feszültségszintek alapján automatikusan letesztelik, hogy melyik a tápfeszültség kivezetése, így a nem szabványkiosztású IC-khez is használhatók. Ugyancsak létezik olyan változat is, amelyik használható a TTL, CMOS és DTL áramkörökhöz is [2], [7].

A speciális sínkijelzőket és -vezérlőket valamilyen sínrendszerrel kapcsolatosan alkalmazzák. Ez a sínrendszer lehet mikroprocesszor-függő, azaz valamelyik mikroprocesszor-család elemeihez szabott, de lehet független sínrendszer is. A sínvezérlők és -kijelzők feladata, hogy a síneken az adatforgalmat vezéreljék, és a kezelőnek jelezzék a mindenkori „sínállapotot".

A sínrendszerekkel részletesebben a 7. fejezetben foglalkozunk.

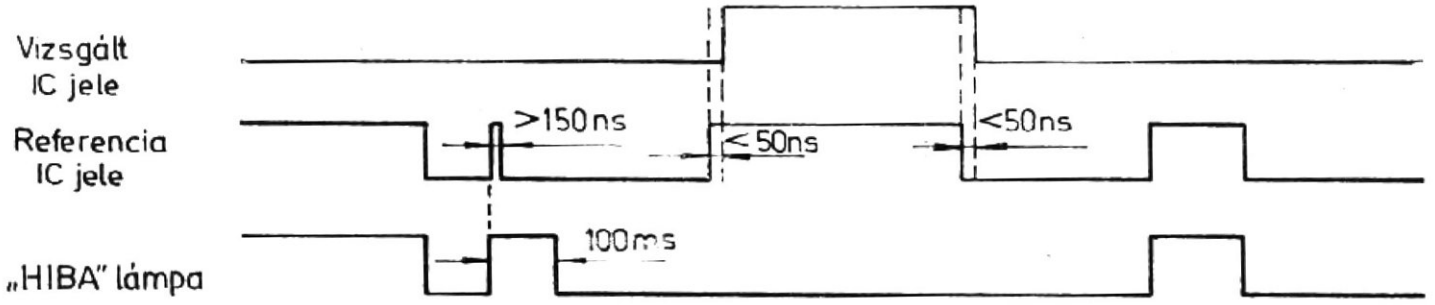
Programozott logikai eszközökhöz, mikroprocesszoros rendszerekhez használatos sínkijelzők és sínvezérlők legfontosabb szolgáltatásai a következők:

- a sín mindenkori állapotának kijelzése,
- kézi adatátvitel kezdeményezésének lehetősége a felhasználó részére (beavatkozási lehetőség),
- a programfutás lelassítása, lépésenkénti üzemmód lehetősége,
- több funkciójú, több processzoros rendszer esetén a szelektív kijelzés,
- több részleges sínből álló sínrendszer esetén a részsínek állapotainak szelektív kijelzése, és a szelektív beavatkozás lehetősége.

E szolgáltatások biztosítják azt, hogy a programvezérelt mikroprocesszoros berendezés működését a kezelő ellenőrizni tudja az egyes részfunkciók, részműveletek megfigyelésével. Ugyancsak szükségesek ezek a szolgáltatások ahhoz, hogy a részműveletek ellenőrzéséhez a kezelő a kiindulási adatokat megadhassa, ill. a működés befejezése után a feldolgozott adatok alapján a működés helyességét ellenőrizhesse.

Logikai komparátorok

Integrált áramkörök működésének a vizsgálatához használatos eszközök a logikai komparátorok. Segítségükkel gyorsan, kényelmesen és az integrált áramköri tokoknak az eltávolítása nélkül dönthető el, hogy egy-egy elem helyesen működik-e. Működési elvük igen egyszerű. A bemérendő berendezésben levő tok kivezetésein észlelt jeleket hasonlítják össze egy ugyanolyan típusú, jól működő áramkör jeleivel. Az összehasonlítandó jeleket az integrált áramkör kivezetéseire helyezhető csipesz segítségével érzékeli. A referenciaként használt IC-tokot a felhasználó cserélheti a berendezésben. Működési elvéből következik, hogy a mérendő IC bemeneti pontjain levő hibát nem érezheti, csak a kimeneten megjelenő hibákat. Így a hibát mindig a jelforrásként működő kimeneten érzékeli, a további lokalizálás más eszközökkel végezhető el. TTL áramkörök esetén általában a 150...200 ns-os jeleitéréseket már jelzi (4.4. ábra). Némelyik típusban nemcsak IC-t használnak referenciaként, hanem egy ún. referenciacsipeszt is alkalmazhatnak. Ezt akkor használják, ha egy jó, működőképes



4.4. ÁBRA. A logikai komparátor működése [2], [7]

4.2. TÁBLÁZAT. A logikai komparátor alkalmazási lehetőségei [2], [7]

Áramkörtípus	Használhatóság jellemzői
Kombinációs logikai áramkör (AND, NAND, OR, NOR, XOR stb.)	Kiváló Ez a kategória képviseli a megvalósított berendezések logikai áramköreinek túlnyomó részét
Sorrendi logikai áramkör (bistabilok)	Kiváló A referencia- és a tesztelendő áramköröket a kiindulási állapotban szinkronizálni kell (pl. reset jel)
Tárak Léptetőregiszterek	Kiváló Hosszabb ideig rákapcsolva elérhető, hogy mind a referencia, mind a tesztelendő IC belső tartalma azonos legyen
Monostabilok	Általában jó Időzítés elállítódhat, mert az RC tagokat a referencia IC is terheli
Nyitott kollektoros és háromállapotú logikai körök	Rossz Tekintve, hogy a kimenet vonalakat más IC-k is használják, a tesztelendő vonalak jele helyes működés közben is eltérhet a referenciajelektől
Analóg áramkörök	Nem használható Analóg jelszint összehasonlítására a komparátor nem alkalmas

berendezés (áramköri kártya stb.) integrált áramköreivel mérik össze a tesztelendő berendezés (áramköri kártya stb.) azonos típusú és funkciójú áramköreit.

Az IC-k logikai komparátorait különösen előnyösen alkalmazhatjuk digitális visszacsatolások vizsgálatára, hiszen az áramkör megbontása nélkül annak a tényleges működése vizsgálható.

A 4.2. táblázatban röviden összefoglaltuk, hogy a logikai komparátor alkalmazásával milyen a különböző áramkörök vizsgálatának a hatékonysága.

A logikai jelszintkijelző, impulzusadó, áramút-nyomkövető, IC-re csíptethető kijelző és logikai komparátor áramkörökből különböző készleteket is kínálnak a gyártók [2], [7]. Ezekkel a készletekkel a digitális berendezésekben a hibabehatárolási munkák túlnyomó többsége elvégezhető.

4.2. MIKROSZÁMÍTÓGÉP-MODULOK

A mikroprocesszorok és az LSI és VLSI eszközök megjelenése a digitális rendszerek tervezői számára azt jelentette, hogy nemcsak a rendszer hardverjét kell megépítenie és az abba „beleintegrált" működési algoritmust megvalósítania, hanem a programíráshoz is kell értenie. A működési algoritmus nagy része a programban realizálódik. A programírás követelményének a megjelenésével azonban együtt jár a hardvertervezési munkák csökkenése. A hardvertervezési munkák nagy részét ui. az integrált áramkörök gyártói vállalják magukra.

A hardver létrehozására irányuló tevékenységet elősegíti az is, hogy az alkatelemek gyártói olyan felhasználói kézikönyveket is adnak a megrendelőknek, amelyek kész „recepteket" tartalmaznak a különböző feladatokhoz, a különböző bonyolultságú berendezések összeállításához. Ezek a „receptek" szinte teljesen kész kapcsolási rajzok, amelyeken általában egy mikroprocesszort és a köré építendő tárat (PROM, RAM), be-, kimeneti áramköröket és egyéb kiegészítő elemeket és a közöttük létrehozandó kapcsolatot tüntetik fel.

A bonyolultabb, nagyobb kiépítettségű felhasználói rendszerek tervezését egyéb módon is segítik a gyártó cégek. A nyomtatott áramköri kártyák széles választékát kínálják, amelyek között szerepelnek egykártyás mikroszámítógépek, de találhatók mikroszámítógép-alegységek (tárak, processzorok, be-, kimeneti egységek, kijelzőegységek stb.). Ezek azonos méretű és felépítésű kártyák, kiegészítő elemekkel, amelyek lehetővé teszik az egy közös sínre való csatlakoztatást is. Az áramköri kártyakészlethez természetesen a sínrendszer, tápegység és a kártyákat magába foglaló mechanikai váz (doboz) is kapható. Felhasználva ezeket az építőblokkokat (prototyping kit), a megrendelő a számítógépeknél megszokott módon „konfigurálhatja" a hardvert.

A felhasználók kiegészítő áramköreiket a készletbe tartozó ún. általános célú áramköri kártyákon helyezhetik el. Ezek a kártyák foglalatokat tartalmaznak, amelyekbe az integrált áramkörök behelyezhetők. Az alkatelemek közötti kapcsolat „csavart kötéssel" (wire wrap) valósítható meg.

Példaként a 4.3. táblázatban megadjuk, hogy a Zilog „Z–80 Microcomputer Board Series" készlete milyen modulokat tartalmaz (a felsorolás nem teljes, egyes áramköri kártyatípusokból több altípus is ismeretes). Hasonló áramköri modulkészletet kínál sok más LSI és VLSI alkatelemet gyártó cég is a saját mikroprocesszoraihoz [11].

4.3. TÁBLÁZAT. Példa egy mikroszámítógépes modulkészletre

Nyomtatott áramköri kártya vagy blokk	
Típusa	Megnevezése
Z–80 MCB	Mikroszámítógép; Z–80 CPU, CTC, PIO, 4 vagy 16 kbyte RAM, 4 kbyte EPROM,
Z–80 MDC	Tár- és mágneslemezes egységek vezérlője 16 vagy 48 kbyte RAM, 3 kbyte PROM, max 8 mágneslemez vezérlése
Z–80 RMB	RAM max 64 kbyte, RAM + max 16 kbyte PROM
Z–80A IOB	Be-, ill. kimeneti kártya 64 be-, ill. kimeneti vonal (8×8 bites)
Z–80 SIB	Soros interfészkártya 4 független duplex I/O vonal
Z–80 AIB	Analóg bemeneti kártya 32 egyvezetékes (16 kétvezetékes) bemenet
Z–80 AIO	Analóg be-, ill. kimeneti kártya 32 egyvezetékes bemenet + 2 analóg kimenet
Z–80 PMB	PROM max 32 kbyte
Z–80 VDB	Videoképernyős megjelenítő vezérlőkártyája
Z–80 PPB	PROM programozókártya
Z–80 SCE–4	4 kártyahelyes asztalra helyezhető doboz
Z–80–SCC	9 kártyahelyes fiók
Z–80 WWB	Felhasználói wire-wrap-es kártya
PS 05 PS 20	Tápegységek

Nemcsak az integrált áramköri elemeket gyártók, hanem más tervező, gyártó vállalatok is kínálnak hasonló készleteket. Ezek a készletek gyakran gazdagabb választékot, különböző mikroprocesszoros elemcsaládok köré épített egységeket tartalmaznak. Érdekes, hogy néhány cég speciális modulokat kínál egy-egy meglevő készletet kiegészítve. (Ilyenek pl. a Burr Brown által kínált A/D, D/A stb. modulok, melyekkel pl.: az Intel kártyakészlete bővíthető [12].)

Természetesen nemcsak a hardverépítést, hanem a programok fejlesztését is segítik a modulkészletet gyártó vállalatok. Ilyen szoftverfejlesztési támogatással a felhasználó viszonylag könnyen elkészítheti és felélesztheti a saját programjait. A szoftvertámogatásról, valamint a programok fejlesztéséről az egyszerű és az integrált fejlesztői rendszerekkel, valamint a program-segédeszközökkel foglalkozó fejezetekben bővebben szólunk.

Az áramköri kártyakészletek között is fontos helyet foglalnak el azok a segédeszközök, amelyek felhasználásával sokprocesszoros rendszereket hozhatunk létre. Ezek az egyszerű mikroprocesszoros rendszerek alkalmazási nehézségein kívül egy sor olyan hardverprobléma és kommunikációs kérdés megoldásához nyújtanak segítséget, amelyek a sokprocesszoros rendszerekkel kapcsolatosak.

Példaként megemlítjük a [13]-ban ismertetett moduláris multiprocesszoros rendszert. Ez a rendszer a processzorok számára elérhető IBUS közös cím-, adat- és vezérlősínre csatlakozik. A mikroprocesszoros állomások az ún. megosztott helyi

tár elven kommunikálnak egymással. Ez azt jelenti, hogy egy-egy mikroprocesszoros állomás általában a saját tárát használja, de bármelyik másik processzorral létesítendő kapcsolat céljából hozzáférhet annak a saját tárához is. Az IBUS-on a mikroprocesszoros állomások prioritás szempontjából egyenrangúak, az ún. „aszinkron forgásos prioritási technikát" használják. Ez azt jelenti, hogy a huzalozott prioritás engedélyezőjele „forog", azaz mindig más processzor bemenetén generálódik. A készlet:

- TMS 9900 bázisú mikroprocesszoros állomást,
- tárállomásokat,
- szinkron és aszinkron átviteli állomásokat,
- DMA vezérlőt és
- tártérképmodult

tartalmaz. A tártérképmodul a rendszerben alkalmazott max tárkapacitást 128 M-szóban (16 bites szó) határozza meg.

Természetesen a rendszernek megfelelő szoftvertámogatása is van.

Az ilyen típusú sokprocesszoros fejlesztési segédeszközcsalád előnyeit fejlesztői a következőkben összegezik:

- nagyfokú modularitás, bővíthetőség,
- viszonylag egyszerű szoftverkezelési lehetőség,
- olcsóság,
- nagyobb megbízhatóság (elsősorban a moduláris és olcsó hardvernek köszönhetően, mely redundáns rendszerfelépítést tesz lehetővé).

Az igen gyors fejlődés következtében nincs arra lehetőség, hogy egy ilyen könyv keretében többé-kevésbé teljes képet adjunk a kínálatról. Nagyon sok alkatrészgyártó, de talán még több számítástechnikával, elektronikus eszközök gyártásával foglalkozó cég kínál különböző készleteket. Ezeket a készleteket a gyártók katalógusai, gépkönyvei ismertetik. Mi csak néhány általánosnak tekinthető gyakorlati tanácsot szeretnénk adni a saját feladatát programozható mikroszámítógéppel vagy programozott logikával megoldani kívánó felhasználónak.

Az első kérdés, amelyre a felhasználónak választ kell adni az, hogy rendszerét vagy annak a prototípusát a készen vásárolt áramköri kártyák felhasználásával akarja-e megépíteni. A döntés meghozatalához legfontosabb mérlegelnie az alábbiakat:

- mekkora darabszámban kívánja gyártani a fejlesztendő berendezést,
- mennyi idő áll rendelkezésre a hardver megépítéséhez,
- hogyan tervezi a felhasználói programok megírását, milyen támogatásra számíthat a programok kidolgozása során.

Amennyiben a darabszám kicsi (esetleg egy), és nagyon rövid idő alatt kell létrehozni a berendezést, célravezető a kész modulokból, áramköri kártyákból való rendszerépítés. (Ha ezek rendelkezésre állnak!). Ha a programíráshoz egyéb számítógépet, azon futtatható keresztassemblert, szimulátort is igénybe tud venni (sok gyártó cég kínál magas szintű programnyelven megírt, tehát számítógéptípustól gyakorlatilag függetlenül használható ilyen kereszt-szoftvert), a programok összeállítása és próbája sem jelenthet különösebb problémákat.

A következő kérdés már a vásárolandó áramköri kártyakészlet kiválasztására vonatkozik. A választáshoz figyelembe veendő legfontosabb szempontok [14]:

- A mikroprocesszor tulajdonságai, lehetőségei maximálisan kihasználhatók-e?
- Az áramköri kártyakonfiguráció alkalmazkodik-e a processzorral kialakítható konfigurációhoz?
- Könnyű-e a rendszerépítés?
- Elég flexibilis-e a hardver?
- Az áramköri kártyakészletet továbbfejleszti-e a gyártó?
- Elérhető-e a megfelelő fejlesztői szoftvertámogatás?

Igen fontos, hogy a mikroprocesszor műveleti sebességét, megszakítási lehetőségeit, címzési rendszerét, a közvetlen tárhozzáférés kiépíthetőségét az áramköri kártyarendszer ne korlátozza. A felhasználó már dönthet úgy, hogy pl.: lassúbb tár miatt gyakran várakozó állapotba kényszeríti a mikroprocesszort, és így korlátozza a műveleti sebességét. Ezt azonban ne maga a prototípusépítéshez használt készlet korlátozza. Nem szerencsés, ha a konfigurációs paramétereket lényegesen megváltoztatja az áramköri kártyakészlet. A többletfunkciók (pl.: „handshake" adatátvitel kialakítása túl bonyolult, többlethardverrel kialakított megszakítási rendszer, új címzési módok stb.) zavarólag hathatnak a felhasználóra. Ilyen esetben már nem az általa tanulmányozott és programozandó mikroprocesszorral van dolga, hanem valamiféle vegyes mikroprocesszoros-miniszámítógépes rendszerrel.

A rendszerépítést befolyásoló tényezők között megemlíthető: a modulok csatlakozási sínjeinek a megfelelő meghajtása (vonali pufferek), be-, ill. kimeneti tárak, háromállapotú vonalak alkalmazása. Ipari alkalmazások esetén különösen fontos a zajérzékenység problémája. Különösen zajérzékenyek az *RC* tagos időzítések, monostabilok.

A hardverflexibilitás megfelelő modulmérettel (kártyanagyság) érhető el. Nem mindig célravezető a felhasználó kezét megkötő modulok használata viszonylag bonyolult és kiterjedt áramköri kártyán belüli fix konfigurációval. Egy-egy konkrét feladat kapcsán azonban az „egykártyás" rendszerek is megfelelőek lehetnek.

A prototípuskészlet továbbfejlesztésén azt értjük, hogy a gyártó szándékában áll-e a későbbiekben bővíteni a választékot. Új LSI és VLSI eszközökkel megjelenő új modulokkal a felhasználó a későbbiekben lényegesen javíthatja berendezésének specifikációját, paramétereit anélkül, hogy a teljes konfiguráció átépítése vagy a szoftver nagymértékű átdolgozása szükségessé válna.

A felhasználói programok kidolgozásához jelentékeny segítséget nyújtanak a készlethez adaptálható szoftvereszközök vagy a különböző számítógépeken futtatható keresztprogramok.

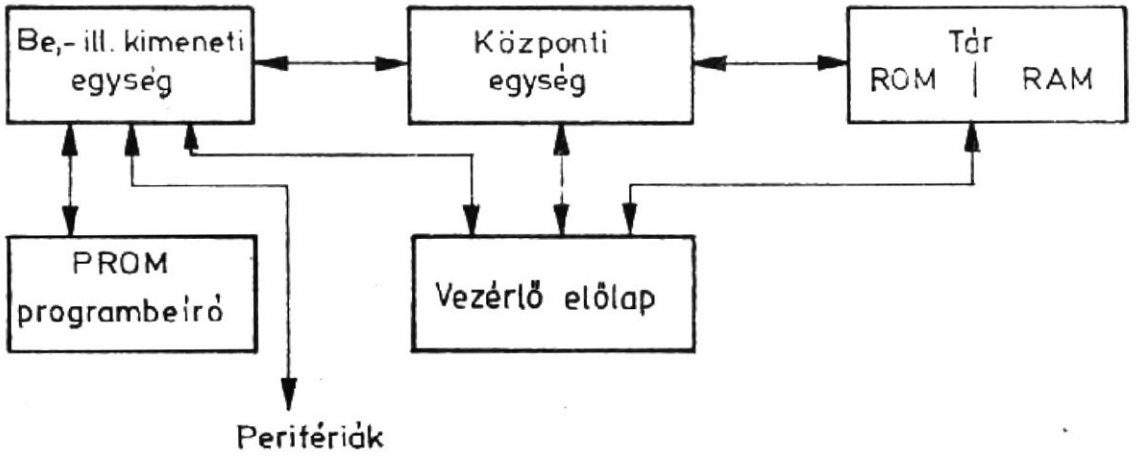
4.3. EGYSZERŰ MIKROPROCESSZOROS FEJLESZTŐI SEGÉDESZKÖZÖK

Az egyszerű mikroprocesszoros fejlesztői eszközöket két csoportba sorolhatjuk. Az egyik csoportba a szoftverfejlesztési célokra alkalmas, s többé-kevésbé általános célú mikroszámítógépek sorolhatók. A második csoportba az egyszerű, áramköri kártyánként megvehető és bővíthető fejlesztői segédeszközök tartoznak (gyakran az alapkonfiguráció csak egyetlen kártyából áll). Tekintettel arra, hogy fejlesztési célokra készültek ezek az egy vagy néhány kártyás fejlesztői eszközök, ezért ezeket a szoftver kidolgozásához és a hibakereséshez programokkal – általában ROM-ba égetett formában – látják el. Használatukat tekintve átmenetet képeznek az építőelemként szolgáló kártyakészlet és a fejlettebb integrált fejlesztői segédeszközök között, bár bonyolultságuk szerint az előbbiekhez lényegesen közelebb állnak.

A mikroszámítógépek

Ezek a miniszámítógépekhez hasonló felépítésűek. A különbség talán annyi, hogy a programfejlesztési céloknak megfelelően speciális előlappal és általában PROM, ill. REPROM programbeíró modullal is ellátják őket (4.5. ábra). Az előlapról kiváltható funkciók közül általánosként megemlíthetők [15]:

- a központi egység összes lényeges vezérlő-, cím- és adatjelének a kijelzése,



4.5. ÁBRA. A programfejlesztési célokat szolgáló mikroszámítógép egyszerűsített vázlata

- beavatkozás a központi egység működésébe (várakozás állapotba helyezés, megszakításkérő jel generálása, lépésenkénti programvégrehajtás),
- tár olvasása, beírás,
- be-, ill. kimeneti műveletek kezdeményezése,
- programban nem levő utasítások végrehajtatása,
- csapdák elhelyezése stb.

Az egyszerűbb, olcsóbb mikroszámítógépes eszközöknek általában csak lyukszalagos és írógép (képernyős megjelenítő) perifériájuk van, míg a bonyolultabb konfigurációkat már hajlékony mágneslemezes egységekkel is ellátják. A hajlékony lemez alkalmazása jelentősen megnöveli a programfejlesztés hatékonyságát.

Fejlesztési célokra a következő alapszoftvermodulok vannak:

- egyszerű operációs rendszer, monitorprogram,
- lyukszalagos vagy hajlékony mágneslemezes szerkesztőprogram,
- fordítóprogramok (assemblerek, compilerek),
- egyszerű hibakereső programok,
- file-kezelő programok (hajlékony mágneslemezes rendszer esetén) stb.

A programok közül az operációs rendszer, ill. a monitorprogram a ROM-ban helyezkedik el, így bekapcsolás után a rendszer azonnal működni kezd. A ROM-ban való tárolás gyakorlati haszna: elkerülhető a hosszadalmas és nehézkes kezdeti programbeolvasás.

A hajlékony mágneslemezes rendszerek sokkal kényelmesebb adat- és programkezelést tesznek lehetővé. A könnyebb kezelhetőség, a mágneslemezen tárolt programok és adatok programmal való könnyű olvasása és írása sokkal fejlettebb és bonyolultabb rendszerprogramok használatát eredményezi.

Az egyszerű egy- vagy néhány kártyás fejlesztői segédeszközök

Az integrált áramkörök gyártástechnológiájának tökéletesedése mind több funkció és információ kezelését, ill. tárolását teszi lehetővé egy-egy alkatelemben. Az alkatelemek komplexitásának a növekedése következtében egy-egy nyomtatott áramköri lapon is mind több funkció valósítható meg. Így jelentek meg azok a fejlesztői segédeszközök, amelyek egy-egy kártyán egész mikroszámítógépet – viszonylag jól használható szoftvertámogatással – valósítanak meg. Természetesen, a fejlesztés támogatásán kívül akár építőmodulokként is használhatók. Más szavakkal, ezek az eszközök olyan prototípuskészlet részeit képviselik, amelyet ROM-ban rögzített fejlesztői szoftverrel is ellátnak. A fejlesztési célok miatt azonban megfelelő ki- és bemeneti interfészeken keresztül adatok bevitelére és kihozatalára is szükség van. Ezt a funkciót kihasználva, valamint a rendszer RAM-ját igénybe véve, további, nem ROM-rezidens programok is bevihetők az ilyen rendszerekbe. Példaként a 4.4. és a 4.5. táblázatban bemutatjuk

4.4. TÁBLÁZAT. A Z–80 SDB–80E jellemzői [16], [17]

Hardver	Központi egység	Z–80 mikroprocesszor
	Tár	4k...16 kbyte RAM 5k...20 kbyte ROM
	Be-, ill. kimeneti egység	4 párhuzamos csatorna (8 bit adat + handshake) Soros interfész (110...9600 Baud) 4 időzítő–számláló csatorna
	Méretek	1 kettős euro-card 250 mm×233,4 mm×18 mm
PROM-ban levő programok	Operációs rendszer (DDT–80)	2 kbyte
	Assembler/editor és debug funkciók (ASMB–80)	8 kbyte
Egyéb programok	XFOR–80	FORTRAN-IV-ben írt keresztassembler Z–80-as forrásprogramok fordításához
	XMDS	MDS-en (Intel 8080A mikroprocesszoron) futtatható keresztassembler
	XMDS–80D	XMDS–80 hajlékony lemezes változata

4.5. TÁBLÁZAT. SDB–80-nal kompatibilis kiegészítő hardvereszközök [16], [17]

RAM–80AE	16 kbyte RAM
RAM–80BE	16 kbyte RAM és 4 párhuzamos be-, ill. kimeneti csatorna
AIM–80E	In-circuit-emulációs debug áramköri kártya
FLP–80E	hajlékony lemezes interfészáramköri kártya
RIO–80E	16 kbyte ROM, 4 párhuzamos be-, ill. kimeneti csatorna, soros interfész és számláló–időzítő csatornák
VDI–P	Párhuzamos interfész képernyős megjelenítőhöz
VDI–S	Soros képernyős megjelenítő interfész
PPG–08	PROM programbeíró (PROM: 2708)
BACK–80E	hátlap nyolc áramköri kártyahellyel
Készülékdoboz és tápegység	Development Station Z–80

a Zilog, ill. a Mostek SDB–80E típusú Z80-as programok fejlesztési céljaira használatos egykártyás mikroszámítógépét, annak szoftverrendszerét és kiegészítő moduljait.

Bár a korlátozott tárméret miatt a ROM-rezidens programok kevesebb és egyszerűbb szolgáltatásokat nyújtanak, mint pl.: az integrált fejlesztői rendszerek programjai (l. a 3.7. szakaszt), mégis igen jól használhatók. Az alábbi műveleteket tartalmazza az SDB–80E DDT–80 operációs rendszere:

- tárgyprogram be- ill. kivitele,
- tárba, be-, ill. kimeneti csatornába írás, olvasás,
- töréspont megadása,
- felhasználói program futtatása,
- szabványos perifériák kezelőprogramjai.

Az assembler feladatai:

- „hárommenetes" fordítás végrehajtása,
- feltételes fordítás,
- relokálható tárgyprogram generálása,
- relokálható program betöltése.

A szerkesztőprogram pedig a sor- és a karakterműveleteken kívül makroparancsok definiálását és használatát is engedélyezi.

4.4. MINISZÁMÍTÓGÉP ALAPÚ RENDSZEREK

Programfejlesztés

Azokat a programfejlesztési eszközöket, amelyek számítógépen futtatható programok, de feladatuk az, hogy egy másik számítógépre írt szoftvert dolgozzanak ki, kereszt (cross) jelzővel illeti a szakirodalom [18]. Így pl.: keresztprogramokról, keresztrendszerekről vagy keresztassemblerekről beszélhetünk.

Azok a mikroprocesszoros felhasználók, akik már korábban is a számítástechnika területén dolgoztak szívesen fordulnak számítógépekhez az LSI és a VLSI elemekből felépülő rendszerek programjainak létrehozása során. A kisszámítógépek (csakúgy mint a nagyobbak vagy a számítógép-hálózatok) ilyen irányú igénybevétele lehet az egyik célravezető út. A korszerű kisszámítógépek legtöbbjének mágneslemezes háttértára, ennek megfelelően lemezes operációs rendszere is van. Az operációs rendszer irányítása alatt futó alapprogramok igen jól támogathatják a mikroprocesszoros berendezések programjainak az írását

A forrásprogramok írásának kényelmes eszköze a szerkesztő- (editor) program, amellyel bármilyen betűkből, számokból és írásjelekből álló, sorokra tördelt szöveginformáció írható és javítható, így tartalmától, szimbólumaitól függetlenül a mikroprocesszor forrásnyelvű programja is. Mágneslemezes rendszer esetében a forrásnyelvű program a mágneslemezen file formájában helyezkedik el, ahonnan a felhasználó kérésére a szerkesztőprogrammal javítás céljából, ill. a fordítóprogramokkal fordítás során kényelmesen elérhető.

A legnagyobb jelentőségű szoftversegédeszköz a fordítóprogram (assembler). Egy új keresztassembler elkészítéséhez nagyon sok munkára és nagy rendszerprogramkészítői gyakorlatra van szükség. A felhasználók, szerencsére, hozzájuthatnak egyéb módon is különböző típusú keresztassembler-programokhoz. Általában két lehetőség van, ezek közül azonban csak szerencsés körülmények között választhatunk.

1. A mikroprocesszort gyártó cégek általában valamilyen magas szintű programnyelven megírt, s így számítógéptípustól függetlenül futtatható keresztassemblereket is kínálnak.
2. Egyes miniszámítógépes assemblerek viszonylag kevés munkával átalakíthatók

jól használható keresztassemblerekké. Az ilyen assemblerek gyakorlatilag bármilyen mikroszámítógép utasításainak a fogadására alkalmassá tehetők az „utasítás-szimbólumtáblázat" kicserélésével, ill. kiegészítésével. Esetleg további módosításra is szükség lehet azért, hogy a tárgyprogramkimeneten a lefordított program a mikroszámítógép (vagy PROM programbeíró) által megkívánt formában álljon elő.

A különböző file-manipulációkhoz, programok listázásához, perifériára való kiviteléhez, beolvasásához jó segítséget nyújtanak a miniszámítógépes operációs rendszer egyéb szolgáltatásai.

Az elkészült tárgyprogram kipróbálására is felhasználhatók a miniszámítógépek. Az egyik eljárás az assemulátoros minigépes rendszert használja, más eljárások lényege a szimulátorprogramok használata [19].

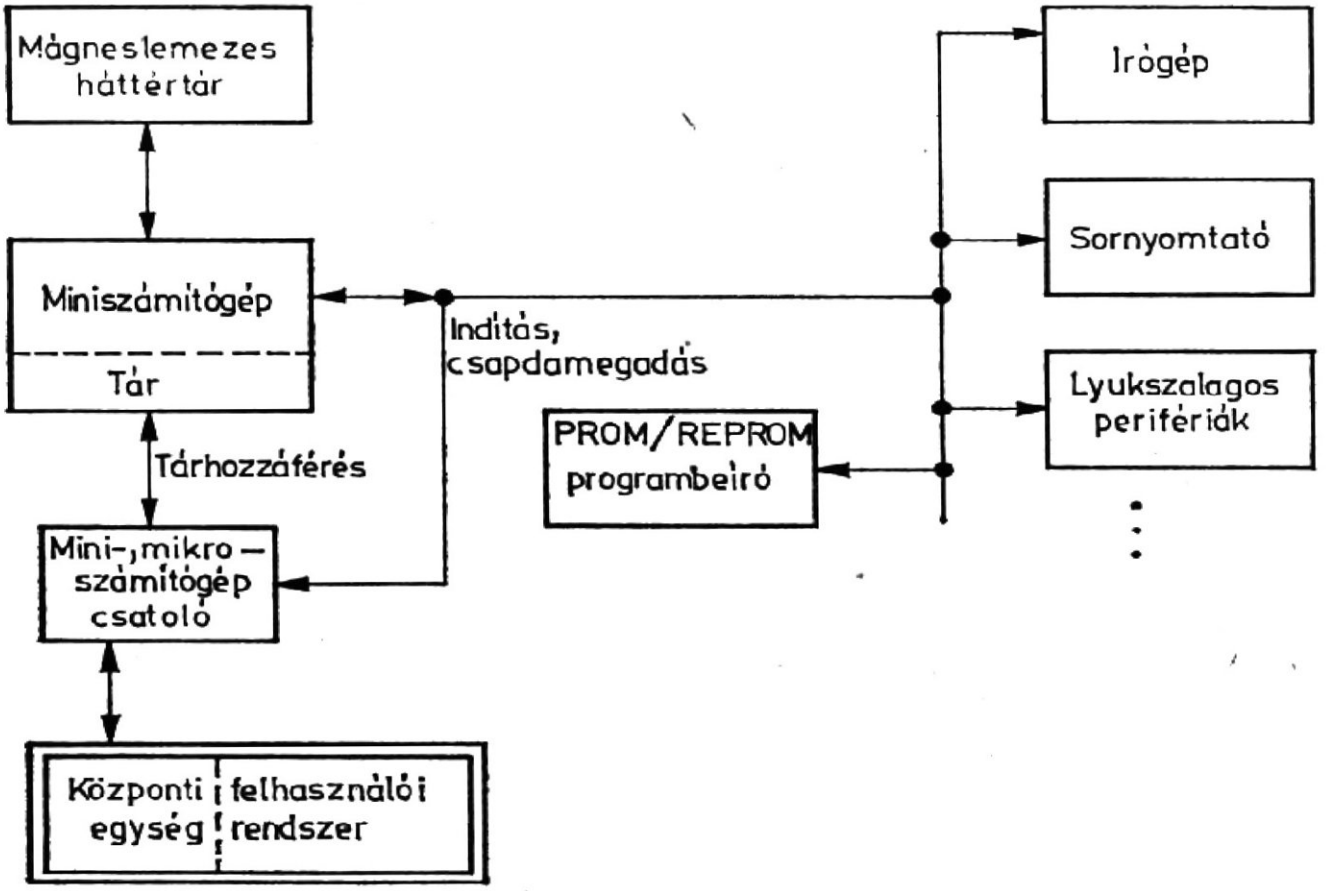
Assemulátor

Az assemulátoros miniszámítógépeket a programfejlesztési segédeszközökön túlmenően olyan hardver- és szoftverkiegészítő modulokkal is ellátják, amelyeket program-, hardver vagy mindkét hiba behatárolására, hibakeresésére hoztak létre. A működési alapelv a következő: a programpróbák idején a miniszámítógép operatív tárának egy részét a mikroprocesszoros felhasználói berendezés (mikroszámítógép) tára helyett használjuk a kipróbálandó programok és adatok tárolására. Ezt a módszert ROM-szimulációnak nevezhetjük, hiszen a miniszámítógép tára imitálja, helyettesíti, azaz szimulálja a mikroszámítógép majdani programtároló tárát, a ROM-ot (PROM-ot vagy REPROM-ot). Ehhez természetesen egy viszonylag egyszerű csatolóegységet kell kialakítani a kisszámítógép és a felhasználói rendszer között, amelynek a közvetítésével a mikroszámítógép hozzáférhet a miniszámítógép tárához (a miniszámítógépben célszerű a gyorsabb közvetlen tárhozzáférést használni). Lényeges, hogy assemulátorunkkal az ellenőrzött programfuttatás műveletét is meg tudjuk valósítani. Ehhez a csatolóegységben olyan lehetőségre van szükség, amellyel a kisszámítógépen futó monitor- vagy hibakereső programmal együttműködve mind a tetszőleges címről való elindítás, mind pedig a tetszőleges helyen való megállás megvalósítható [15].

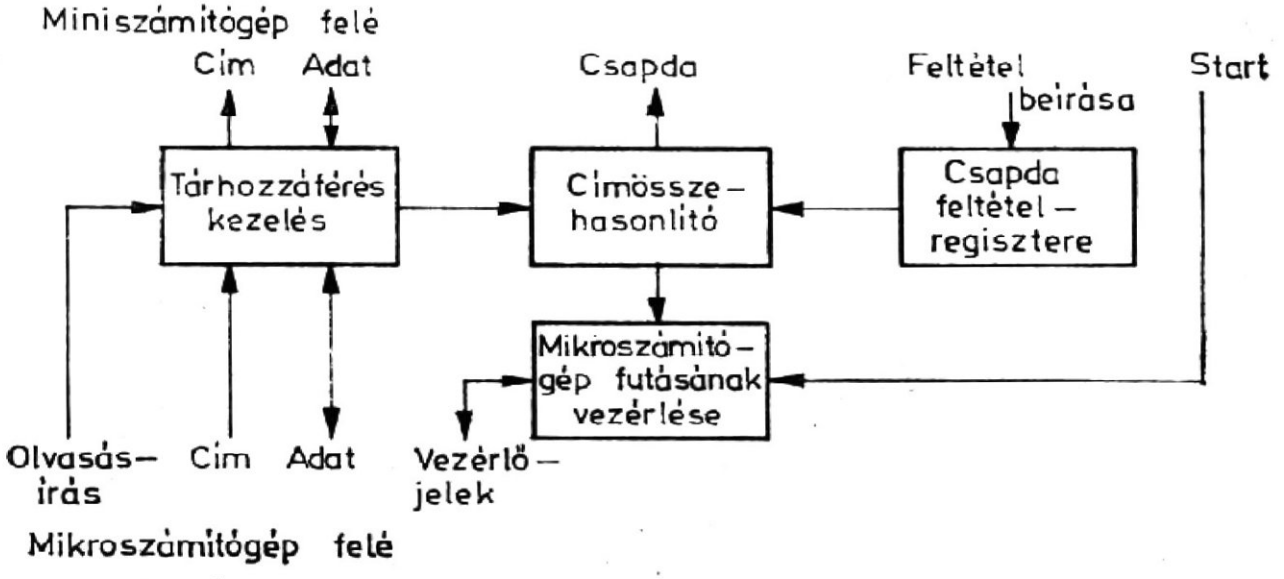
Az ilyen programfuttatás viszonyai a mikroszámítógép adatfeldolgozó központi egysége és be-, ill. kimeneti egységei számára a valódi körülményekhez igen közel álló feltételeket adnak, ellentétben a szimulációs módszerrel.

Az assemulátor hardverelrendezésére mutat be egy lehetséges változatot a 4.6.ábra. A mini–mikroszámítógép csatolóegysége itt a közvetlen tárhozzáférési csatornán keresztül írja, ill. olvassa a tárban lévő programutasításokat és adatokat, a monitorprogram pedig a programvezérelt csatornán keresztül a csatolóegység vezérlését végzi el. A 4.7. ábrán a csatolóegység blokkdiagramja látható. A mikroszámítógép futását vezérlő blokk start parancsot kap a monitorprogramtól, míg a leállást kiváltó jelet a csapda elérésekor generálja a címösszehasonlító egység. Indítás előtt a kezelő a monitorprogram közvetítésével megadja a leállási (csapda) feltételt. A mikroszámítógép a tárhozzáférést kezelő blokk útján ír a miniszámítógép tárába, ill. olvas belőle. A cím-összehasonlító a tárcímek állandó figyelésével detektálja a „csapda" elérését, ekkor megállítja a mikroszámítógép futását, de jelzést ad a monitorprogramnak is [20].

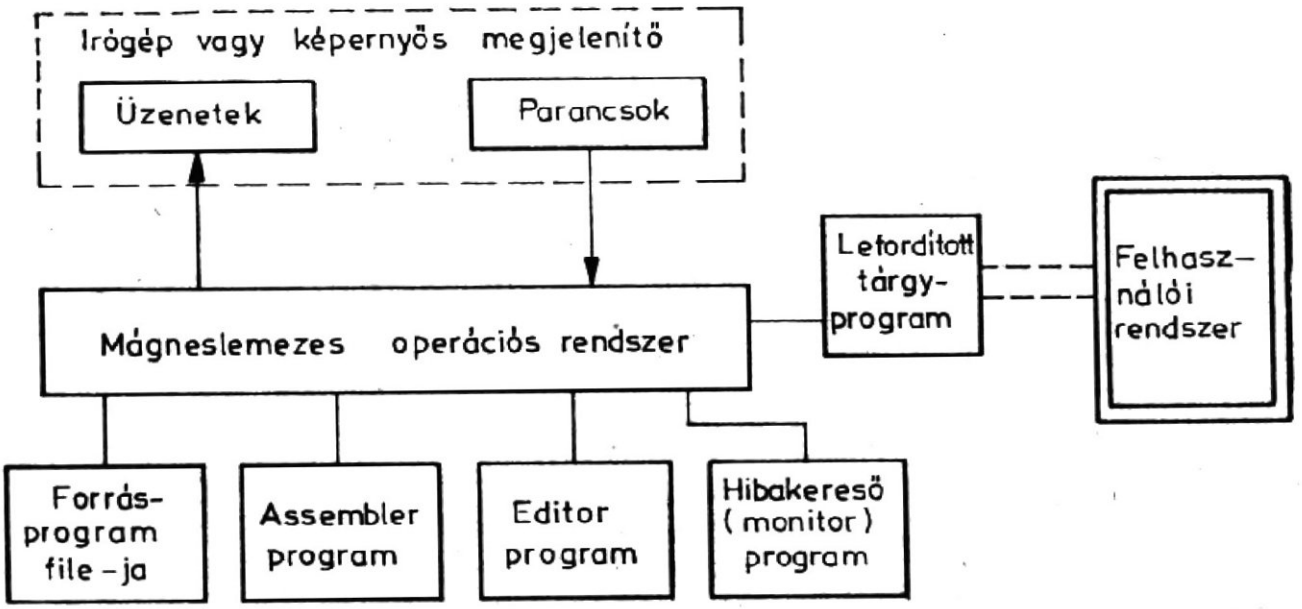
Nagymértékben egyszerűsíthető a csatolóegység megoldása, ha a miniszámítógép tárának nagyobb a szóhossza, mint a mikroprocesszornak. Ekkor a csapda jelzése elhelyezhető a mikroprocesszor által nem használt valamelyik bitbe is. Így elhagyható a csatolóegység cím-összehasonlító blokkja és feltételtároló regisztere, helyette egy csapdabit-detektáló blokkra van szükség.



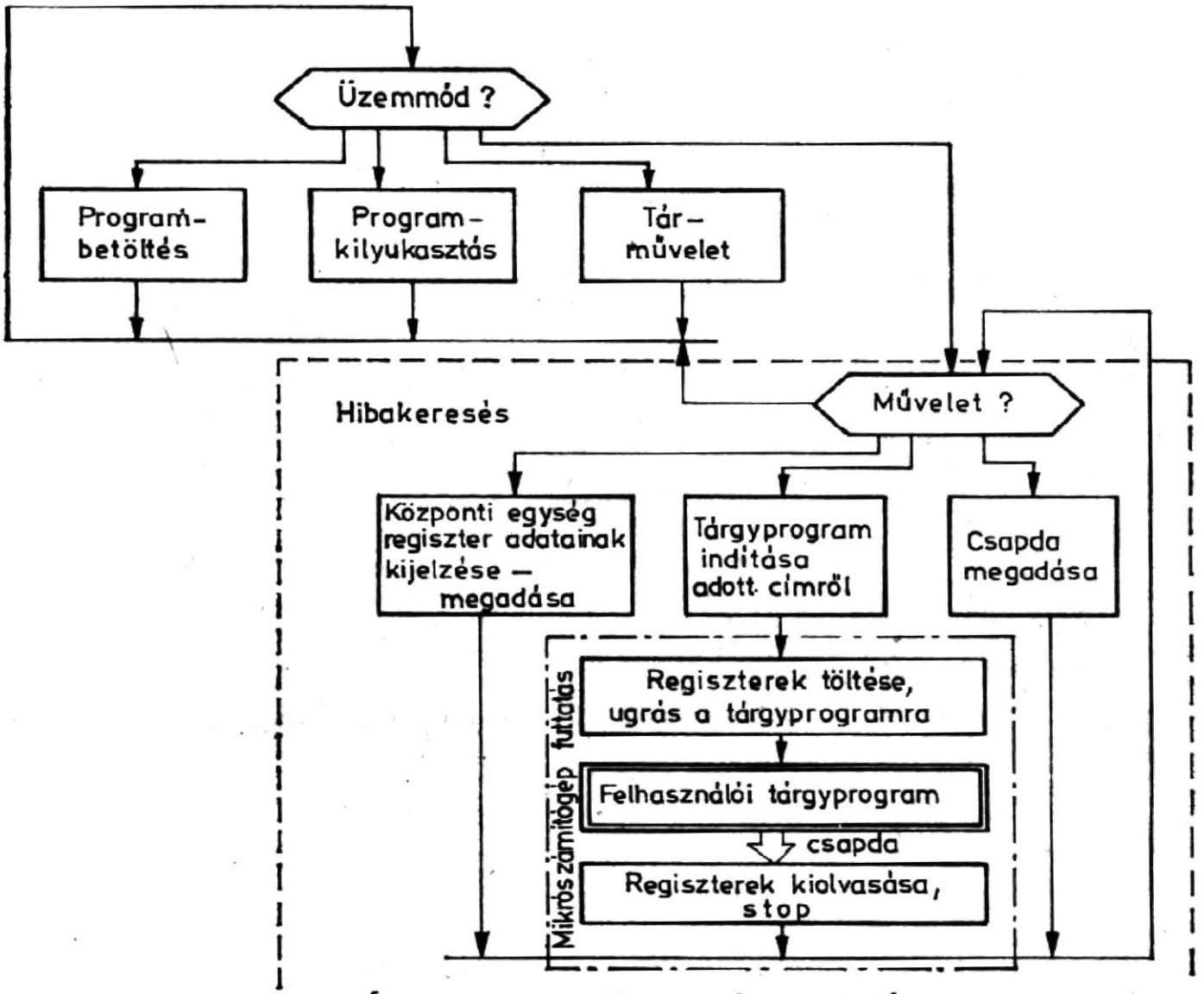
4.6. ÁBRA. Az assemulátor vázlata



4.7. ÁBRA. A miniszámítógép–mikroszámítógép csatolóegysége



4.8. ÁBRA. Az assemulátor programrendszere [15]

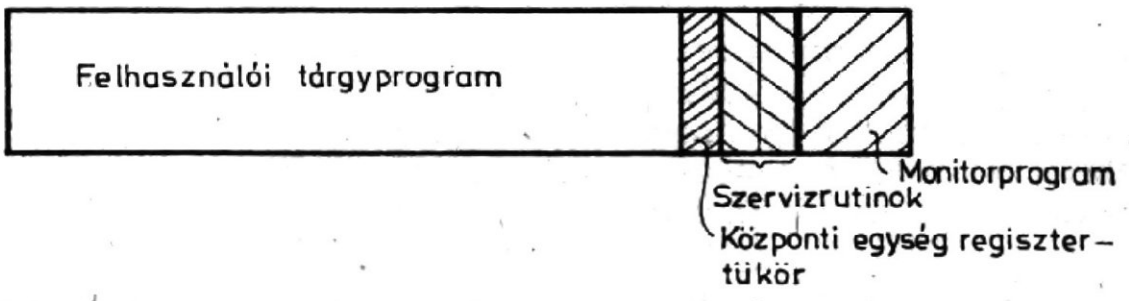


4.9. ÁBRA. Az assemulátor monitorprogramja

Az assemulátor programrendszerének a blokkvázlatát a 4.8. ábra mutatja be. A megfelelő programokat a miniszámítógép lemezes operációs rendszerének a felhasználásával hívhatjuk be. Írógép vagy képernyős megjelenítő segítségével parancsokkal vezérelhetjük a programelőkészítés és -kipróbálás folyamatát, amelyekre a rendszertől ugyancsak az írógépen vagy a képernyős megjelenítőn keresztül kapunk üzeneteket. A munkát a forrásprogram összeállításával, lefordításával, majd az észlelt hibáknak a kijavításával kezdjük. Ehhez a szerkesztő, ill. a fordítóprogram segítségére van szükségünk. Ezt követően a lefordított programot betöltjük a miniszámítógép operatív tárának a mikroszámítógép által használt részébe, majd elkezdjük az ellenőrzött futtatás műveletét. Ez a hibakereső vagy monitorprogram felügyelete alatt végezhető el. A hibakereső programot néha nyomkövető névvel is említi a szakirodalom.

A 4.9. ábra bemutatja a monitorprogram egyik változatát. A monitorprogramot írógépről vezérelheti a gépkezelő. Segítségével lyukszalagról betölthetők a kipróbálandó tárgyprogramok a miniszámítógép tárába, vagy az ellenőrzött és kipróbált programok lyukszalagra kivihetők. A programozó a tárgyprogram kisebb hibáit a tárban lévő gépi kódok közvetlen kijavításával is elvégezheti. A tényleges hibakereséshez hozzátartozik a mikroszámítógép központi egysége (CPU) regisztereinek az ellenőrzése. Ez oly módon történhet, hogy a monitorprogram (ami a miniszámítógép gépi kódú programja) tartalmaz ún. szervizrutinokat. Ezeket nem a miniszámítógép, hanem a mikroprocesszor hajtja végre, így a szervizrutinok annak a gépi kódjait tartalmazzák. A felhasználói program futása előtt az egyik szervizturin feltölti a tárból a központi egység regisztereit. Futás után pedig a másik szervizturint is végrehajtatja a monitorprogram a mikroszámítógéppel, s ennek eredményeképpen a központiegység-regiszterek tartalma a miniszámítógép tárába kerül. A tárnak azt a részét, ahol a regiszterek tartalmát tárolják, regisztertükörnek is szokták nevezni. A 4.10. ábrán egy példán mutatjuk be a miniszámítógép tártartalmát hibakeresés alatt. A regiszterek tartalmának ellenőrzését és módosítását tehát a regisztertükör és a szervizrutinok közbeiktatásával érhetjük el.

Természetesen az assemulátoros fejlesztői segédeszközök sokféle változatát és lehetséges megoldásukat nem ismertethetjük. Az előbbiekben csak egy jellemző példával mutattuk meg az ilyen rendszerek felépítését. Általában miniszámítógép alapúak, de ismeretes mikroszámítógépes változatuk is. Ez azonban sem a felépítésük lényegét, sem pedig a használatukat nem érinti.



4.10. ÁBRA. A tárgyprogram és a monitorprogram elhelyezkedése a miniszámítógép tárában

4.5. ÁLLAPOTANALIZÁTOROK

Logikai állapotanalizátorok

A kódolt jelek átvitele, valamint a párhuzamos adatátviteli csatornák működésük vizsgálatához elengedhetetlenül szükségessé teszik olyan vizsgáló „műszerek" alkalmazását, amelyek oszcilloszkópszerűen végigkövetik a jelek váltásait, felrajzolják a jelalakokat, alkalmasak arra, hogy párhuzamos jeleken adott időviszonyok mellett, adott időpillanatokban végezzék el ezt a vizsgálatot. Ahogy az oszcilloszkóp alkalmas a gyorsan változó jeleknek, ember által nem észlelhetően rövid idejű jelenségeknek az emberi érzékszervek számára felfoghatóvá, érzékelhetővé tételére, kellő idejű megjelenítésére, így kell ezt a kódolt és párhuzamos jelek vizsgálatára használatos műszernek is megvalósítania.

Ismétlődő jelek esetében ezt az oszcilloszkóp elektronsugara szinkronjelre indulással és a képernyőre való sokszori felrajzolással éri el. Egyszer vagy nagyon ritkán megjelenő jelek esetében az elektronsugár az oszcilloszkóp képernyőjére egyszer rajzolja fel a jelalakot, amelynek az emberi észlelését a hagyományos oszcilloszkópok esetén a nagy intenzitásra (fényerőre) állított elektronsugár és a fényemittáló foszforréteg utóvilágítása teszi lehetővé. Mindenki tudja azonban, hogy mennyire nehéz az ilyen jelet észlelni, ez a módszer a korszerű hibakereséshez már alig alkalmazható. Nehézkes és hosszadalmas a fényképezéssel kombinált vizsgálat is. A korszerűbb készülékek tárolócsöves (memorizáló képernyős) kivitelben készülnek. Az egyszeri megjelenítés és a memorizálás megoldott tehát az oszcilloszkópokban. Nem megoldott azonban sok jelnek (párhuzamos sínek jeleinek) a vizsgálata, hiszen az oszcilloszkópok általában csak két jelalak egyidejű megfigyelésére, esetleg egy harmadik – indító – jellel való időviszonyaiknak a megfigyelésére használhatók. Igaz ugyan, hogy további előny az oszcilloszkópok esetében, hogy analóg jelek vizsgálatára is kitűnően alkalmasak. Digitális kapcsolások esetén azonban erre nincs mindig szükség.

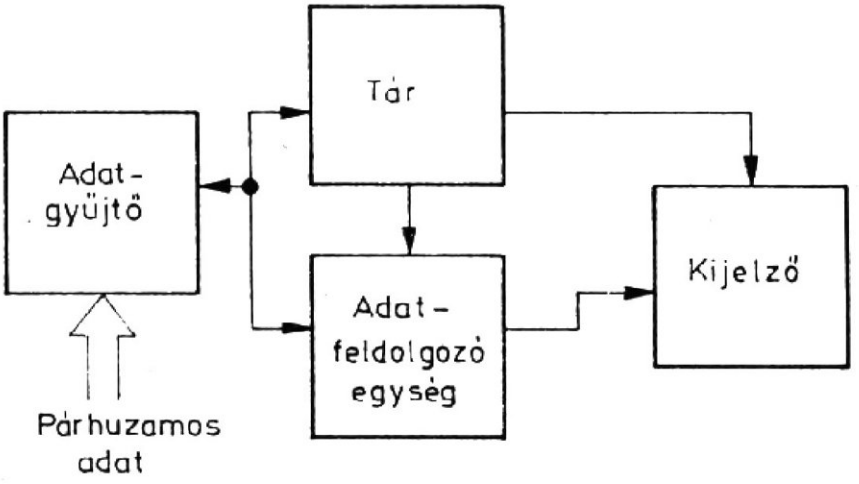
Több párhuzamos rövid idejű vagy gyorsan váltakozó jel megfigyelését végzik a logikai állapotanalizátorok. Nagyon nehéz közülük tipikusnak tekinteni valamelyik konkrét típust. „Egyetlen új területen sem jelent meg még ennyi eltérő gyártmány" – olvashatjuk a logikai analizátorokról [22]. Erre utal a szakirodalomban található sokféle elnevezés is: állapotanalizátorok, időanalizátorok, logikai diagramkijelzők, logikai analizátorok és univerzális, ill. dedikált mikroprocesszor-analizátorok. A következőkben mégis megpróbáljuk röviden összegezni közös vonásaikat, majd áttekinteni a legfontosabb funkcióikat és a velük kapcsolatos fogalmakat.

Bármilyen is az analizátorok formája, felépítése, majdnem mindegyikük tartalmaz adatgyűjtő részt, tárat, adatfeldolgozó és értelmező részt, valamint kijelzőt. A további egységek, mint pl: kimeneti csatlakozóegység, állapotindikátor-egységek, kezelést megkönnyítő funkciókat megvalósító egységek a típustól függően találhatók. A 4.11. ábrán feltüntettük a négy alapblokkot és egymás közötti kapcsolatukat. Az alábbiakban az egyes részeket és velük kapcsolatosan néhány kérdést ismertetünk [22].

Az adatgyűjtő feladata a vizsgálandó párhuzamos adatoknak az analizátor tárolójába való bevitele. Az adatgyűjtés végezhető: beállítható frekvenciával futó belső órajelre (mintavételi jelre), vagy külső indító jelre. Igen fontos részét képezik a szondák. A szondák kiviteli formája lehet:

- integrált áramkörre csiptethető,
- integrált áramköri foglalat,
- egyszerű beforraszthatő huzalok,
- különálló egyvezetékes szondák.

Közöttük vannak mind passzív, mind pedig aktív szondák. Lényeges paramétereik



4.11. ÁBRA. A logikai állapotanalizátorok alapvető egységei

közé tartoznak: bemeneti impedancia (ohmikus és kapacitív), lezáró- és meghajtófokozat, holtidő stb. Ezek a paraméterek alapvetően befolyásolják az adatgyűjtési sebességet. Az adatgyűjtési sebességet – belső órával – a vizsgált logikai áramkör működési sebességének kb. az 5...10-szeresére kell megválasztani.

Zajérzékenység szempontjából fontos, hogy a küszöbszintek állíthatók legyenek a bemeneteken. Az állítható küszöbszintek lehetővé teszik, hogy az analizátor különböző típusú logikai áramkörökhöz (DTL, TTL, MOS, CMOS stb.) is alkalmazható legyen. Egyes analizátoroknak a bemenetén tüskejel-érzékelő is van. Ez „megnyújtja" a szonda által érzékelt, de a mintavételi időnél rövidebb tüskejeleket, és így lehetővé teszi azok tárba írását. Természetesen a tüskejel-érzékelő a felhasználó kívánságára ki is iktatható. Az adatgyűjtő csatornaszáma általában megegyezik az analizátortár egyes szavainak bitszámával. Gyakoriak a 8, 16 és 32 bites analizátorok, de ismeretesek bővíthető csatornaszámú típusok is [23].

A tár feladata, hogy a gyors adatgyűjtés és az emberi érzékelésre alkalmas lassú kijelzés közötti időbeli ellentmondást a segítségével áthidalhassuk. Mérete és működési sebessége meghatározó az analizátor szempontjából. Mikroprocesszoros berendezések fejlesztéséhez kívánatos, hogy az mind az adat-, mind a címinformáció, valamint a vezérlőjelek befogadására alkalmas legyen. Másrészt a vizsgált jelenség időtartama a tár kapacitásával van kapcsolatban. Hardverhibakeresés során általában aszinkron, gyors mintavételezés a kívánatos, s ehhez a csatornánkénti 256 mintavételnél kevesebb jelkövetés nem ad kielégítő eredményt. Szoftverhibakeresés során az esetek nagy részében már 50 utasítás nyomon követése is elegendő. Igaz ugyan, hogy a hardvermunkákhoz általában sokkal kisebb szóhossz szükséges, mint a programok ellenőrzéséhez. A programfuttatáshoz általában az 1...4 MHz-es mintavételi gyakoriság is megfelelő, ezért egyes analizátorokban oly módon alakítják ki a tárat, hogy a hardvervizsgálathoz kisebb szóhosszú, de nagyobb szószámú – pl.: 1 k×8 bit – gyors tárként használható. Programellenőrzés során négy 8 bites rekeszt összekapcsolva – az előbbi példánál maradva – 256×32 bites szó beírására nyílik lehetőség. Kihasználva az alacsonyabb mintavételi gyakoriságot az egy szóhoz tartozó négy tárrekesz időmultiplex módon beírható egy mintavételi periódus alatt.

A különböző típusú analizátorokban az adatfeldolgozó és értelmező rész megvalósítására nagyon sokféle változatot alkalmaznak. Funkciói részben a mintavételezés indításával és leállításával (a triggereléssel) kapcsolatosak, másrészt pedig a kijelzés vezérlését végzik. A mintavételi gyakoriság (az oszcilloszkóp analóg funkciója az időtengely mentén való eltérítés) mellett szükség van a triggerelésre is. Különösen fontos ez a mikroprocesszoros hibakeresésnél, amikor gyakori követelmény 30...50

trigger előtti és utáni utasításnak a nyomonkövetése. Ugyancsak hasznos lehet, ha az analizátor bizonyos fokú interaktív kapcsolatot is megvalósít a gépkezelő és a vizsgált mikroprocesszoros berendezés között. Az ilyen kapcsolat magában foglalhatja a töréspontok (csapdák) megadását, a program megszakítását stb. Természetesen ezeknek a lehetőségeknek a kialakítására bonyolultabb adatfeldolgozó egységre van szükség. A kijelzéssel kapcsolatos feladatokat a kijelzőegység tárgyalása során mutatjuk be.

A kijelzőegységek és a kijelzés formátuma is igen változatos. A legegyszerűbb analizátorok pusztán fénykibocsátó diódás, szegmenses számjegykijelzőket tartalmaznak. Más analizátorok kijelző egysége külső oszcilloszkóp meghajtását és vezérlését látja el. Sokkal szélesebb körben terjedtek el azonban azok az analizátorok, amelyek beépített képernyős megjelenítőket használnak.

Az analizátorokkal kijelzett információval kapcsolatosan három új megjelenítési móddal kell megismerkednünk: idődimenziós (time domain: időtartomány), az adatdimenziós (data domain: adattartomány) móddal, valamint az „adatutak kontúrjainak" kirajzolásával (mapping out) (4.12., 4.13. és 4.14. ábrák). A párhuzamosan begyűjtött adatokat a tárba írja az analizátor, majd az információt a képernyős kijelzőn az ábrákon látható módok valamelyikén megjeleníti. A megjelenítési formák analizátortípusonként változhatnak, egyes analizátoroknak azonban többféle kijelzési módjai is vannak, és a felhasználó kívánsága szerint a neki legmegfelelőbb alakot kérheti [24], [25].

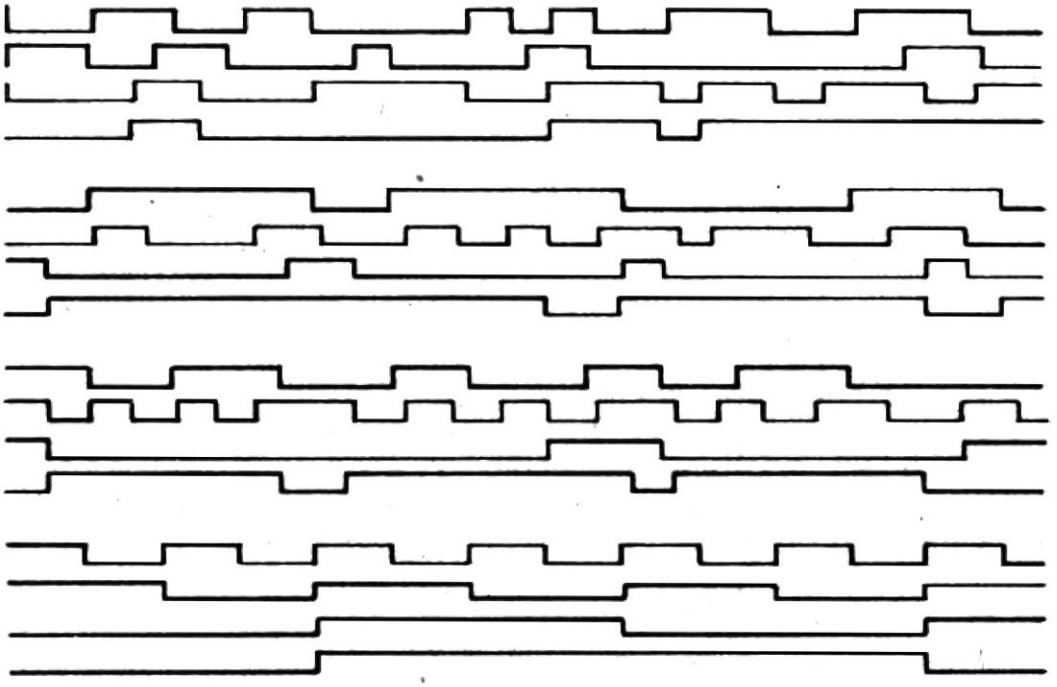
Az *idődimenziót* (idődiagramot) főleg a hardver bemérése és vizsgálata során használják (a csatornajelek diagramjai egymás alatt foglalnak helyet).

Az *adatdimenziót* főleg programanalízishez használják. A csatornák egyidejű adatai, ill. logikai értékei egymás mellett találhatók. (A 4.12. ábrán 16 csatorna adatát tüntettük fel.) A sorok elején a mintavételi sorszám (tárhely címe) látható, a sorok végén pedig kívánságra pl.: oktális, vagy hexadecimális formában is megjeleníthető a párhuzamos adat (pl.: az adatsín vagy a címvezetékek tartalma).

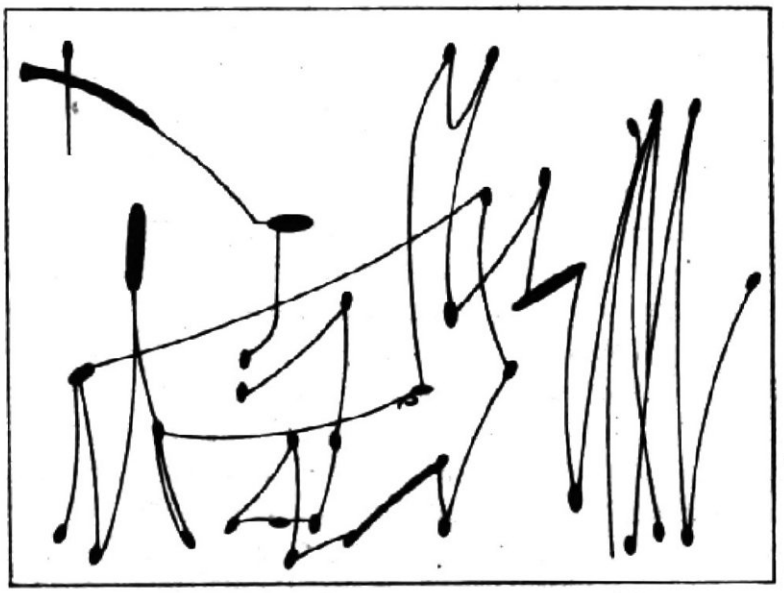
Az *„adatutak kontúrjai" üzemmód* a párhuzamos csatornák adatairól egy-egy fénypontot rajzol ki a képernyőn. Pl.: 16 bites adatnak 2^{16} pontot lehet megfeleltetni (256×256-os raszter). Ez különösen hasznos futó programok vizsgálatára oly módon,

001	1110	1110	1000	0000	EE80
002	1010	1111	0100	0100	AF44
003	0010	1000	1001	0011	2893
004	0010	1001	1001	0011	2993
005	0000	1001	1000	0100	0984
006	0000	1101	0111	0100	0D74
007	0000	1111	1000	0010	0F82
008	0000	0011	1010	0011	03A3
009	1000	1100	1001	0001	8C91
010	1100	1100	1010	0001	CCA1
011	1110	0000	1111	0001	E0F1
012	0111	0000	1001	0110	7096
013	1111	1111	1011	1111	FFBF
014	0010	0000	1011	0111	20B7
015	0100	0000	1101	1000	40D8
016	1111	1111	1100	0110	FFC6

4.12. ÁBRA. Adatdimenziós megjelenítés képernyőn



4.13. ÁBRA. Idődimenziós megjelenítés képernyőn



4.14. ÁBRA. Az „adatutak kontúrjai" képernyőn

hogy a tárcímeket jelentetjük meg egy-egy fénypont formájában. A fénypontok mutatják, hogy a program mely tárhelyekre ír vagy olvas (adatokat és utasításokat). A fényerő arányos az adott tárhelyhez fordulás gyakoriságával. A 4.14. ábra alapján észlelhető pl.: az, hogy tiltott helyre „kalandozik-e el" a program. Ez a kijelzési mód különösen hasznos tesztprogramok futása esetén a helyes vagy helytelen működés kérdésének a megválaszolására. Ha az ábra a tesztprogramra jellemző alakú (amit az ellenőrzést végző szakember már jól ismer), akkor a teszt helyesen fut le, ellenkező esetben nem.

A fenti három megjelenítési módon kívül még egyéb megjelenítési lehetőségek is ismertek. Ezekről bővebben a mikroprocesszor-analizátorok kapcsán lesz még szó.

Idő- és adatdimenziós megjelenítés során az összegyűjtött adatok teljes egészében nem jeleníthetők meg. A képernyő mérete korlátozza ezt. Időmenzió-megjelenítés esetén a jobb vizsgálatot teszi lehetővé az, hogy a megjelenített ábrát időben (kevesebb mintavételt rajzolva az ernyőre) „nyújtani" lehet. A „nyújtáshoz" ún. cursor-jel tartozik, amely azt mutatja, hogy melyik (hányadik) mintavételtől kezdődik a megjelenítés. Az időnyújtás mértékétől pedig az függ, hogy hány táradat (hány mintavételi idő) fér el a képernyőn. Mind a cursor-jel értéke, mind a nyújtás mértéke írott szöveg formájában megjelenik a képernyőn. Adatdimenzió esetén ugyancsak a cursor-jel segítségével lehet végignézni a tárba begyűjtött teljes tartalmat, a 4.12. ábra szerint.

A logikai állapotanalizátorok és a mikroprocesszor-analizátorok igen közeli rokonságban vannak egymással. Néhány igen érdekes, s a logikai analizátorokban is alkalmazott megoldást a mikroprocesszor-analizátorok tárgyalása során ismertetünk. Ennek oka az, hogy ezeknek a megoldásoknak s a velük kapcsolatos új szolgáltatásoknak a haszna legnyilvánvalóbb formában éppen az LSI és a VLSI alapú rendszerek bemérése, a mikroprocesszor-hardver és -szoftver ellenőrzése közben jelentkezik. A logikai analizátorok a tervezés során igen jelentősen megnövelhetik a hibakeresés hatékonyságát. Egyes vélemények szerint az időmegtakarítási tényező értéke 20...400 is lehet [26]. Ez a tényező az oszcilloszkóppal való bemérési munkák és a logikai analizátor segítségével végzett bemérési munkák időráfordításának az aránya.

Röviden ismertetjük a logikai állapotanalizátorok által nyújtott lehetőségeket az egyik típus [27] (a továbbiakban analizátor) jellemzőinek segítségével.

Az analizátor négy 8 bites bemeneti csatlakozási felületű, ezek küszöbszintje egymástól függetlenül állítható —10 V és +10 V között. A megjelenítő adatformátuma is igen rugalmasan változtatható. Ennek köszönhetően jól használható a különböző alkalmazásokhoz, a feladatok széles skálájához és az LSI és a VLSI áramkörök különböző típusaihoz. A bitszeletelt (bit- slice) mikroprocesszorok mikroprogramjainak az ellenőrzéséhez is elegendően magas a vizsgálandó adatokat mintavételező órajele. Ez esetben a „feltételdetektáló" egység számára viszonylag egyszerű logikai összefüggések alapján történik a nyomkövetés adatszelekciója. Makroutasítások vizsgálatához alacsonyabb mintavételezési sebesség is elegendő, itt viszont a nyomkövetési feltételek bonyolultabbak, az analizátor ezekhez a feladatokhoz is jól alkalmazható.

A 4.6. táblázatban röviden ismertetjük a legfontosabb specifikációs jellemzőit, a továbbiakban pedig néhány gyakorlati példa kapcsán bemutatjuk, hogy milyen segítséget kaphatunk a készüléktől a program-, a hardver- vagy mindkét hiba kereséséhez.

Az analizátor legfontosabb alkalmazási funkciója a programvezérelt digitális berendezések és rendszerek adat-, cím- és vezérlősínjein a jelváltások, adatátvitelek nyomonkövetése és kijelzése. A korszerű mikroszámítógépes rendszerek programjai, működési algoritmusa nagyon sok elágazást (ugrásokat, hurkokat, szubrutinhívá-

4.6. TÁBLÁZAT. A HP Model 1610A logikai állapotanalizátor lényegesebb jellemzői [26]

Jellemzők	Értékek
Mintavételi órajel- és adatbemenetek	
Mintavételi frekvencia	max 10 MHz
Bemeneti impedancia	50 kΩ, 14 pF
Bemeneti küszöbszintek	
TTL	kb. +1,5 V
állítható	±10 V között
Maximális bemeneti jelszint	–15 V-tól +15 V-ig
Adatjelek száma	4×8 bit
Trigger és jelkimenetek	TTL jelszint
Tár	
Szóhossz	32 bit
Kapacitás	64 szó
Eseményszámláló	32 bit
Megjelenítés	
képernyőn (egy időben)	max 20 szó (32 bit)
cursorral	a teljes tártartalom
Futási idő mérése	
felbontás	100 ns
mérési tartomány	max 429,4 s
Tartozékok	
8-bites adatszondablokk	4 db
mintavételi órajel szondája	1 db

sokat, egymásba ágyazott hurkokat és szubrutinokat, többszörös belépési pontú rutinokat stb.) tartalmaz.

Az analizátornak e feladat ellátásához a következő tulajdonságai vannak:

- szekvenciális indíthatóság (sequential triggering),
- szelektív nyomkövetés (selective trace),
- szekvenciális újraindíthatóság (sequence restart),
- időmérés és állapotszámlálás (time and state count).

Az *indíthatóság* lehetővé teszi, hogy a logikai állapotanalizátor az adatgyűjtést csak a felhasználói program egy bizonyos pontjától kezdve végezze el. A programok legtöbbje nagyon sok belső állapotba (elemi gépi ciklusba) vezérli a vizsgálandó rendszert, a vizsgálni kívánt állapotok azonban csak a program egy bizonyos pontjától kezdve jelentkeznek. Ezt az indítási (trigger) pontot a felhasználó előre definiálhatja. A *szekvenciális indíthatóság* bonyolultabb indítási feltétel előírását is lehetővé teszi. Csak akkor indul az adatgyűjtés az adott ponttól, ha azt megelőzően más kijelölt

pontokon (állapotokon) is áthaladt a program. Természetesen a megelőző pontok kijelölését is a mérést végző kezelő végzi. Így vizsgálandó pl.: egy szubrutin működése akkor, ha azt a program megadott helyéről, megadott adatokkal hívtuk. Ilyenkor a megelőző állapotokat a hívás helye és az adatok jelentik, az adatgyűjtés kezdetét pedig a szubrutinban lévő pont jelentheti. Ha ugyanezt a szubrutint a program más helyről, vagy nem az előírt adatokkal hívja, a triggerelés feltétele nem teljesül, az adatgyűjtés szünetel.

A szükségtelen adatok begyűjtését iktathatjuk ki a *szelektív nyomkövetési* lehetőséggel. Így elérhető, hogy triggerelés után sem gyűjt be az analizátor minden állapotot, hanem csak azokat, amelyek adott feltételeknek tesznek eleget (adott címhez tartoznak stb.).

Elképzelhető, hogy indítás után, már csak adatgyűjtés közben derül ki, hogy nem ez a vizsgálni kívánt programrész (pl. ha aritmetikai műveletek szubrutinjainak a vizsgálatakor csak az a futás érdekli a felhasználót, amelynek során a számítások eredménye bizonyos feltételnek tesz eleget – negatív szám, nulla stb.). Ebben az esetben igen hasznos a *szekvenciális újraindíthatóság*, vagyis az analizátornak az a tulajdonsága, hogy a program futása során az adatgyűjtés a triggerelési feltételek teljesülése esetén újrakezdődhet, mindaddig, amíg szükséges.

Az *időmérés* lehetővé teszi annak a meghatározását, hogy a program egyes lépései (állapotai) között mennyi idő telik el. Annak az ismerete, hogy a program egyes hurkok, szubrutinok elvégzésére mennyi időt fordít, a programozó számára hasznos információt jelent hibakeresés során, de programoptimalizáláshoz is szükség van erre az adatra.

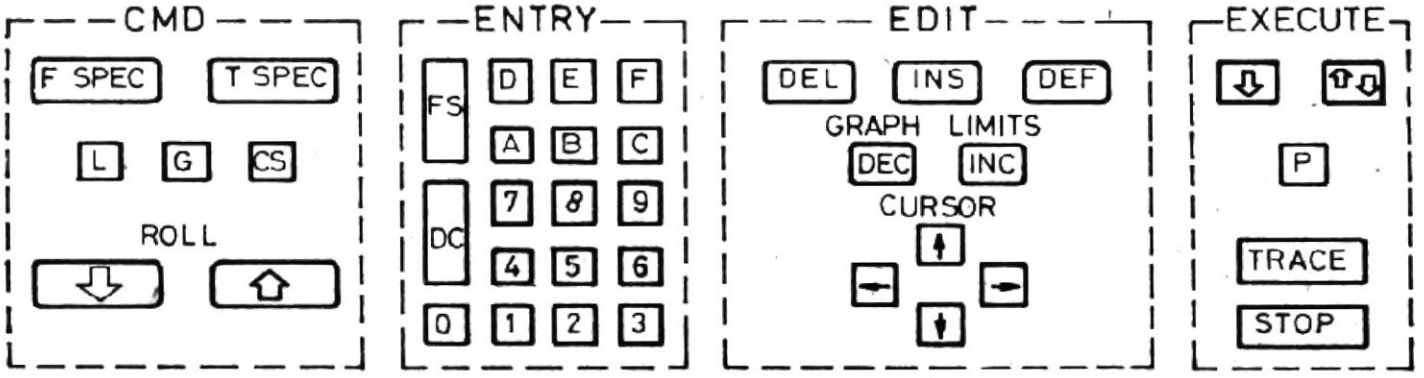
Az *állapotok számának a megállapítása* az időméréshez hasonló célokat szolgálhat. Ennek alapján megtudhatjuk pl. hogy szelektív nyomkövetés mellett az egyik begyűjtött és kijelzett állapotból hány nem „begyűjtött" állapoton (gépi cikluson) keresztül jutott el a program a következő nyomkövetési feltételig.

A sokféle indítást, adatgyűjtést, a működéssel kapcsolatos feltételeket meglehetősen bonyolult szabályok szerint kell általában megadni. Az analizátor viszonylag kevés kezelőszervet tartalmaz, a vezérlő billentyűzet, a feltételek megadása nagyon egyszerű. Kettős célt követünk, amikor a kezelőszerveket, használatukat bővebben tárgyaljuk. Egyik célunk az analizátor ismertetése, használatának a bemutatása. A második cél az alkalmazott eljárás leírása. Ez az eljárás számítástechnikai rendszerekben, így a mikroprocesszoros rendszerekben is, igen célszerűen használható az ember–gép kapcsolat javítására. Az eljárás lényege: a képernyős megjelenítőre maga a készülék, ill. a számítástechnikai rendszer rajzolja fel a „menüt", azaz mindazokat az adatokat, információkat, amelyekből a kezelő kiválaszthatja az általa megadni kívánt parancsokat, kódokat. A „menü" használata egyrészt azzal segíti a kezelőt, hogy nincs szükség az általa megadandó sokféle üzenet megjegyzésére, másrészt pedig a képernyőn meglévő felsorolásból a kiválasztottra való „rámutatással" megadható az üzenet, nincs szükség annyi külső kezelőszervre (pl.: billentyűre). A „rámutatás" széleskörűen alkalmazott eszköze a képernyős megjelenítők cursorje (fénypont, fénycsík), amely a képernyőn mozgatható. A mozgatás a program kezdeményezésére vagy a kezelő kezdeményezésére végezhető.

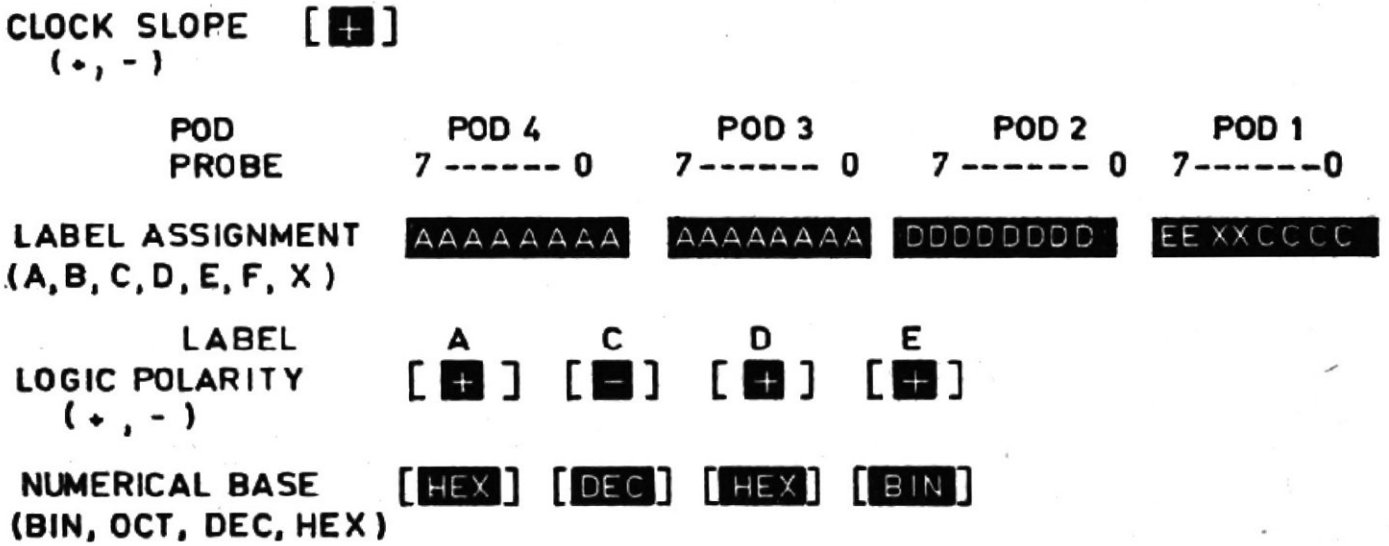
A 4.15. ábrán látható a vezérlő billentyűzetnek a vázlatos rajza. A billentyűket négy csoportba oszthatjuk:

- kijelzéssel kapcsolatos billentyűk (CMD jelű),
- paramétermegadó billentyűcsoport (ENTRY),
- üzenetszerkesztéssel kapcsolatos billentyűk (EDIT),
- működést vezérlő billentyűk (EXECUTE).

A kijelzéssel kapcsolatos billentyűk (CMD: Current Measurement Display)



4.15. ÁBRA. Az analizátor billentyűzete



4.16. ÁBRA. A formátum megadása

közül az F SPEC jelűvel (Format **SPEC**ification) az adatformátum meghatározásához szükséges „menüt" kérhetjük a képernyőre. A „menü" a 4.16. ábra szerinti alakban jelenik meg (kissé leegyszerűsítve). A gépkezelő megadhatja:

- a mintavételi órajel mintavevő élét,
- a 4×8 bitnyi bemenő jel egymáshoz tartozását, csoportokra osztását,
- az egymáshoz tartozó jelek logikai polaritását,
- az egymáshoz tartozó jelek (párhuzamos adatok) kijelzéséhez a számrendszeralapot.

Az ábra szerinti példán sötét alapon jelöltük a kezelő által megadott értékeket. A többi információ a „menünek" az automatikusan kijelzett része, amely tartalmazza a megadandó funkció megnevezését (és paramétereit: bemenő bitek sorszáma, egymáshoz tartozó bitek címkéje), valamint zárójelek között a gépkezelőtől várt üzenet lehetséges paramétereit. A megadás kapcsán még az alábbi momentumokra hívjuk fel a figyelmet: az összetartozó címbiteket azonos címkével jelöli a kezelő (A-tól F-ig). Az összetartozási határ (szóhatár) a nyolcas csoporton belül is bármelyik bitnél elhelyezhető. A figyelmen kívül hagyandó (nem használt) bemeneti pontokat X jelöli. A megadáshoz a cursort mozgató négy nyomógombot (EDIT billentyűk nyilakkal ellátva) használva rámutatunk a megadandó üzenet helyére, majd az ENTRY nyomógombokkal megadjuk az üzenetet. Az X-eket a DC (Don't Care) feliratú nyomógombbal adhatjuk meg. Az ábrán a szögletes zárójelek közötti üzenetet automatikusan váltja az analizátor a FS (Field Select) nyomógomb megnyomásakor. Így a kezelő addig váltja ezeket az üzeneteket, amíg a kiválasztottat megtalálja.

Az L (List), G (Graph), CS (Compare-Store) nyomógombokkal kezdeményezhető a (mérés utáni) kijelzés:

- lista (számok, logikai szintek),
- grafikon vagy
- egy előzőleg betárolt adattal való összehasonlítás eredménye formájában.

Ezeket a kijelzési formátumokat a 4.17., 4.18. és a 4.19. ábrák mutatják.

A *listák formátuma* megfelel a korábban említett adatdimenziós megjelenítésnek. Grafikon módban könnyen és gyorsan áttekinthető az analizátor tárába tárolt 64 állapot. A függőleges tengely a kért paraméter értékének felel meg, a vízszintes pedig, az adott érték megjelenési sorrendjének a tengelye. Igen kényelmesen vizsgálható így pl.: a gyorstár használata vagy egyes programhurkok. A grafikonon két adott értékhatár közötti paraméterértékek láthatók. A határokat az EDIT csoport billentyűivel (DEC, INC) változtathatjuk.

Az összehasonlítás és a vele kapcsolatos adatkijelzés célja, hogy egy „jó futás" során mért és memorizált adatok alapján gyorsan, könnyen ellenőrizhessük, hogy az ellenőrzendő futás során ugyanazokat az állapotokat veszi-e fel a felhasználói rendszer. Ilyenkor az ellenőrzendő futás adatait összehasonlítva a „jó adatokkal" csak az eltéréseket jelzi ki az analizátor *0*-tól eltérő szám formájában.

A kijelzés során a képernyőn maximum húsz sor fér el. A 64 állapot kijelzéséhez használják a ROLL nyomógombokat, amelyekkel „fentről" vagy „lentről" újabb sorok léptethetők a képernyőre.

Az adatgyűjtést vagy nyomkövetést megelőzően azonban meg kell adni a nyomkövetési feltételeket. Ezt az adatformátumhoz hasonlóan „menü" megjelenítésével kezdhetjük. A nyomkövetési feltételmenü lehívását a T SPEC nyomógomb végzi. Erre példát mutat a 4.20. ábra.

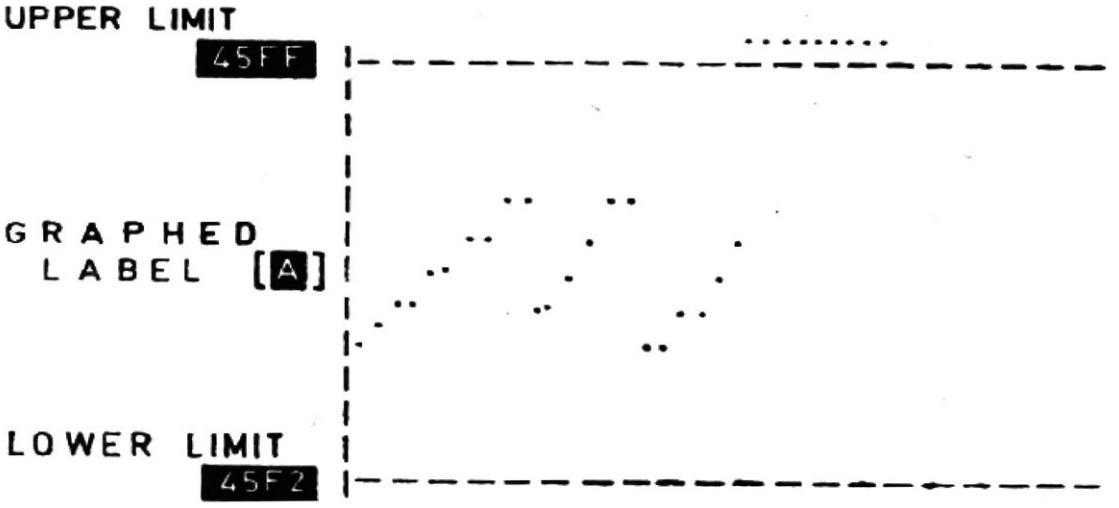
Az ábrán a zárójelek és a sötét alapú részek értelmezése azonos az adatformátum megadásával kapcsolatban elmondottakkal. A példa szerinti megadás jelentése: a nyomkövetést az A címkével jelölt bitek AB00 értékének és az E címkével jelölt bitek közül az alacsonyabb helyiértékű bit *1* értékének a megjelenítésekor kell kezdeni. Az OCCUR feliratú paraméterben számot adhatunk meg, amely azt jelzi, hogy az előírt feltétel hanyadik megjelenésekor (teljesülésekor) kezdődjék a nyomkövetés. A következő sorban a tárba beírandó (nyomkövetendő) adatokra vonatkozó feltételt írtunk elő. Az ábrán látható esetben: memorizálni kell minden olyan állapotot (első megjelenésekor), ahol az E címkéjű adat kis helyiértékű bitje *1*. A tárba beírandó állapotok között az analizátor megszámlálja azt, hogy hány figyelmen kívül hagyandó állapotot vesz fel a mérendő készülék. Ez a COUNT paranccsal kérhető. A példán ezt a szolgáltatást nem igényeltük.

Az alábbiakban a szekvenciális indításra, a szelektív nyomkövetésre, az újraindításra és az időmérésre, ill. az állapotszámlálásra mutatunk be példákat. A példák kapcsán felvetett problémákhoz, hibakereséshez hasonlók a gyakorlatban is sűrűn előfordulhatnak.

A 4.21. *a* ábrán látható a vizsgálandó program vázlata. Az FP főprogram több helyről: a 12AB cím előtti és utáni, ill. a 134F utáni pontokról hívja az SR1 szubrutint. Az SR1 szubrutin kétszer hívja az SR2 szubrutint. Egyszer a 25C2 című utasítás után, majd a 25E2 pont után. Az SR2 szubrutin kezdőcíme 2ABC. „Programbelövés" idején az SR2 rutint kívánjuk nyomkövetni, de csak akkor, amikor az SR1 rutint megelőzően a 134F című utasítás, majd az SR2 rutint megelőzően a 25E2 című utasítás is végrehajtásra került már. A kívánt nyomkövetést a következő feltételek megadásával érhetjük el (szekvenciális indítás): először előírjuk a 134F cím megkeresése feltételt, majd a 25E2 feltételt. Ezt követően adhatjuk meg a nyomkövetés kezdetét jelentő START üzenetet a 2ABC címtől. A megadásnál a következőképpen

LABEL BASE	A HEX	C DEC	D HEX	E BIN
START	AB00	15	C3	10
01	AB01	05	12	11
02	AB02	05	80	11
03	8012	15	FF	10
04	FFFC	07	80	00
05	FFFB	07	13	00

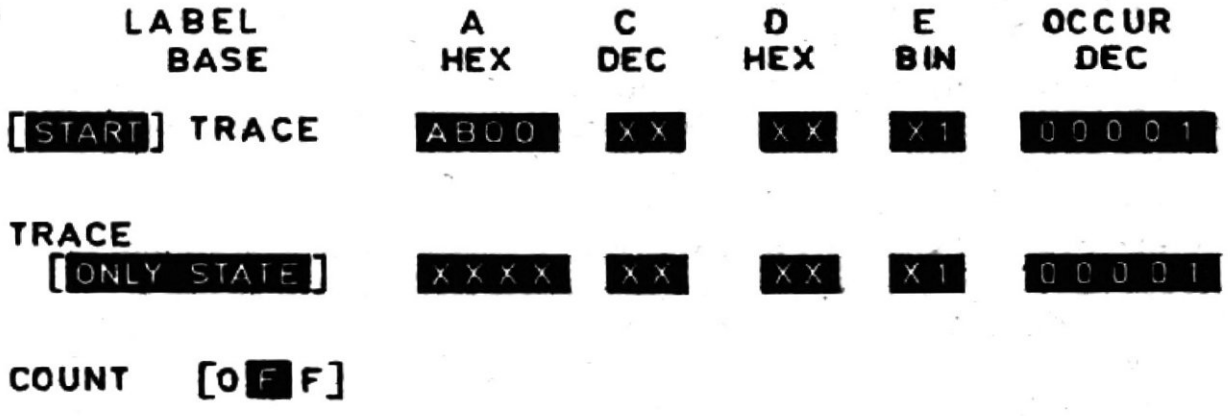
4.17. ÁBRA. A lista kijelzése (adatdimenzió)



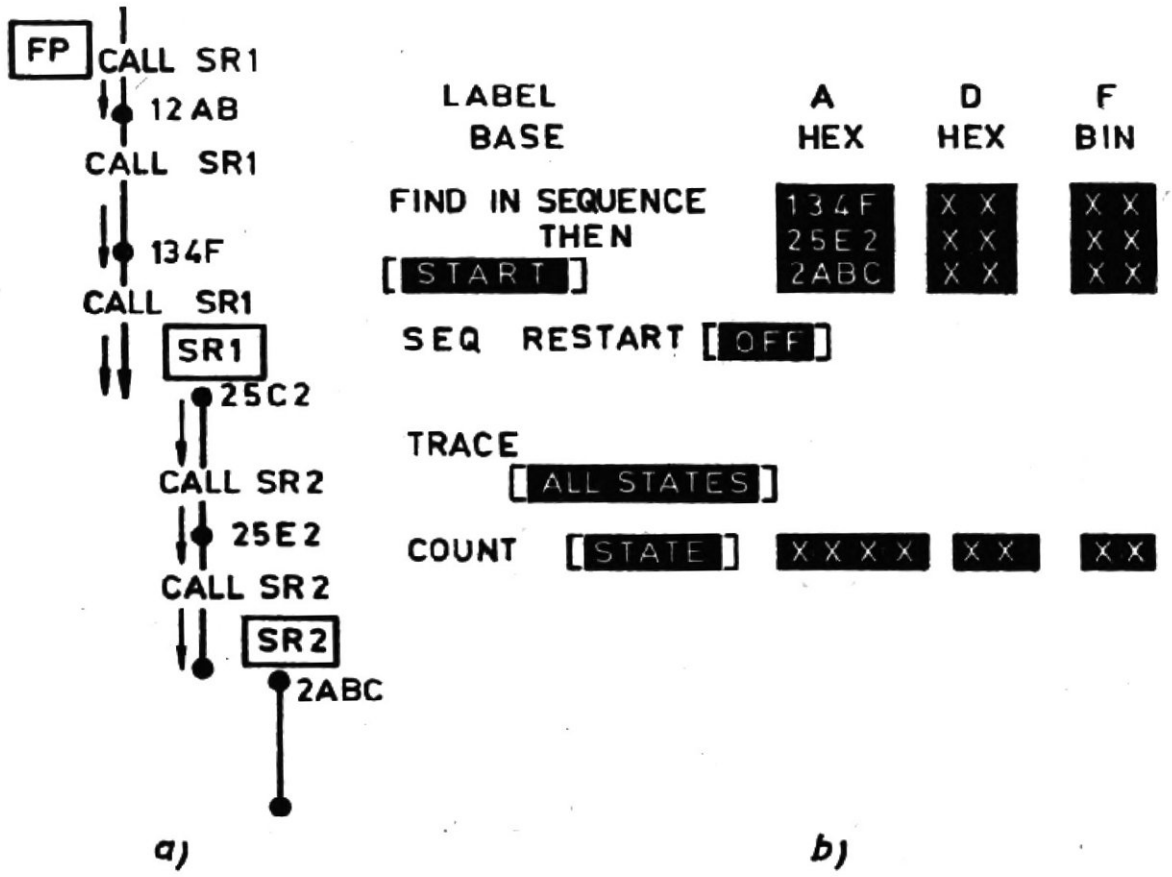
4.18. ÁBRA. Kijelzés grafikon formában

LABEL BASE	A HEX	C DEC	D HEX	E BIN	COMPARED TRACE MODE
START	0000	00	00	00	
01	0000	00	00	10	
02	0000	00	00	10	
03	0000	00	00	00	
04	0000	00	00	00	
05	0000	00	00	01	

4.19. ÁBRA. Az összehasonlítás eredményének a kijelzése



4.20. ÁBRA. A nyomkövetési feltételek megadása

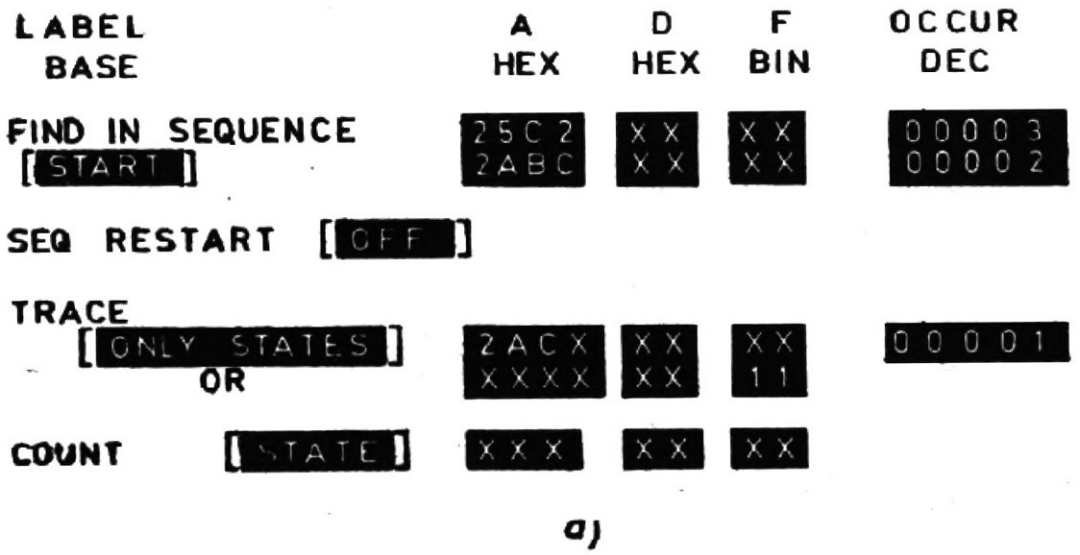


LABEL BASE	A HEX	D HEX	F BIN		STATE COUNT
SEQUENCE	134F	3A	01	—	82
SEQUENCE	25E2	11	01	—	23
START	2ABC	3E	01		0
	2ABD	13	11	+	1
	2ABE	02	01	+	2
	2ABF	32	01	+	3
	2AC0	BF	10	+	4

c)

4.21. ÁBRA. A szekvenciális indítás használata
a) a vizsgált program vázlata, *b)* az analizátor parancsainak megadása, *c)* a vizsgálat eredményeinek kijelzése

járhatunk el: a menü lehívásakor a START üzenethez helyezett cursor jel megjelenése után az EDIT billentyűcsoport INS (insert: beszúrás) nyomógombját használjuk. Ekkor jelenik meg a 4.21. *b* ábra szerinti első feltételsor (FIND IN SEQUENCE), ahol megadhatjuk a 134F feltételparamétert. Amennyiben újabb feltétel is szükséges, az INS nyomógombot újra használhatjuk, ez megjeleníti a második feltételsort (THEN), ahol az újabb paraméter 25E2 is megadható. További feltételek esetén (max hét megengedett) az INS nyomógomb további megnyomása válik szükségessé. Végül a START indítási feltétel, ill. a nyomkövetési feltétel is megadható. Az állapotszámlálás megadására szolgáló üzenet is látható a 4.21. *b* ábrán a COUNT jelű sorban. A példa szerint minden állapot megszámolását kérjük, mert feltételparamétereket nem írtunk elő. A nyomkövetés befejezése után a 4.21. *c* ábra szerinti listát kaphatjuk meg.



a)

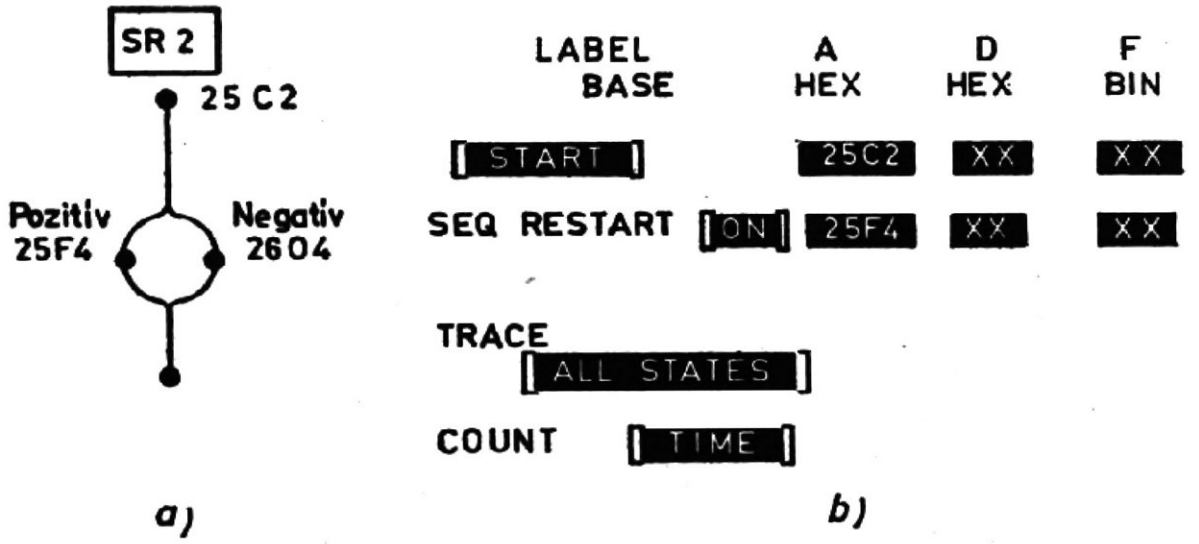
LABEL BASE	A HEX	D HEX	F BIN		STATE COUNT	
SEQUENCE	25C2	47	01	−	78	
START	2ABC	3E	01		0	
	2ABD	13	11	+	1	
	2AC0	BF	10	+	4	
	2AC1	18	01	+	5	Ugrási utasítás
	2AC2	0C	11	+	6	Ugrási utasítás
	2ACF	3E	01	+	7	
	2AD4	44	11	+	12	
	2ADA	7F	11	+	20	

b)

4.22. ÁBRA. A szelektív nyomkövetés *a)* parancsok megadása, *b)* eredmény kijelzése

Szelektív nyomkövetés megadására látunk példát a 4.20., ill. a 4.22. *a* ábrán. Az első esetben csak az E címkéjű adat alacsony helyiértékű bitjeiben *1* értéket tartalmazó állapotok nyomkövetését kérjük, míg a második esetben az SR2 szubrutin 2AC0...2ACF című állapotai, valamint az F adatbitekben *11* értéket tartalmazó állapotok nyomkövetését kérjük. Max hét nyomkövetési feltétel adható meg. Új feltételhez az INS nyomógombot használjuk. A 4.22. *a* ábrán példát mutatunk egy további lehetőségre is – a feltételek megadása egy megjelenési sorszámmal (OCCURance: megjelenés) is kombinálható. A szekvenciális indítás feltétele pl. a 4.21. *a* ábrán bemutatott FP, SR1 és SR2 jelzésű programrészekre ugyanazt jelenti, mint a 4.21. *b* ábra szerinti. Az SR1 harmadik hívása (2EC2 cím megjelenése), majd ezt követően az SR2 második hívása (2ABC cím) jelenti a nyomkövetés kezdetét. A szelektív nyomkövetés eredménye a 4.22. *b* ábrán látható lista. A nyomkövetett állapotok csak a 2AC0...2ACF közötti címhez, ill. az F adat *11* értékéhez tartozók. Az állapotsorszám „ugrálva" növekszik, hiszen az összes állapot összeszámlálását (így a nem nyomkövetettekét is) kértük, a sorszám megjelenítése azonban csak a kijelzetteknél történik meg.

A következő, 4.23. ábrán az újraindítás megadására láthatunk példát. Tételezzük fel, hogy az SR2 szubrutin aritmetikai műveletet hajt végre. A részeredmény függvényében elágazik, pozitív részeredmény esetén a 25F4 címet érinti, míg negatív



4.23. ÁBRA. Az újraindítási feltételek használata
a) vizsgált szubrutin vázlata, *b)* parancsok megadása

részeredmény esetén a 2604 címen keresztülhalad a program. Bennünket nem a pozitív részeredmény, hanem csak a negatív ágnak a nyomkövetése érdekel. Így a 25F4-et tartalmazó ágon való áthaladást követően a nyomkövetést fel kell függeszteni, az indítási feltételek figyelését újra kell kezdeni. Az újraindíthatóságnak köszönhetően szükségtelenné válik ilyen esetekben az esetleges többszörös mérés.

Az időmérés megadására is példát láthatunk ezen az ábrán. Időmérés után a listán a mért adatok oszlopa után megjelenik az időparaméter (az állapotszámhoz hasonlóan). Az időparaméter után az időegység (μs, ms vagy s) is látható. A mérés kijelzése az indítástól mért abszolút időpontokra vagy az állapotok között eltelt időintervallumokra vonatkozhat.

A nyomkövetéshez tartozó START üzenet azt jelenti, hogy az itt leírt feltétel az indító feltétel, ennek a teljesülése után indul az állapotok begyűjtése. START üzenet helyett CENTER vagy END üzenet is adható. CENTER esetében a triggerpontot megelőző 31, ill. az ezt követő 32 állapot, END esetében pedig a triggerpontot megelőző 63 állapot kerül be a tárba.

Az EDIT billentyűcsoport DEL nyomógombjának szerepe gépkezelőművelet esetén ellentétes az INS nyomógomb szerepével, üzenettörlésre, -javításra szolgál. A DEF nyomógombbal ún. default értékek adhatók meg. Az EXECUTE csoport nyilakkal ellátott billentyűi az „összehasonlítás-üzemmódú" nyomkövetés kezelésével kapcsolatosak. A TRACE és STOP nyomógomb a mérés indítását, ill. leállítását vezérli. Az analizátorhoz papírszalagos nyomtatószerkezet is csatlakoztatható. A nyomtatást a P (PRINT) billentyűvel vezérelhetjük.

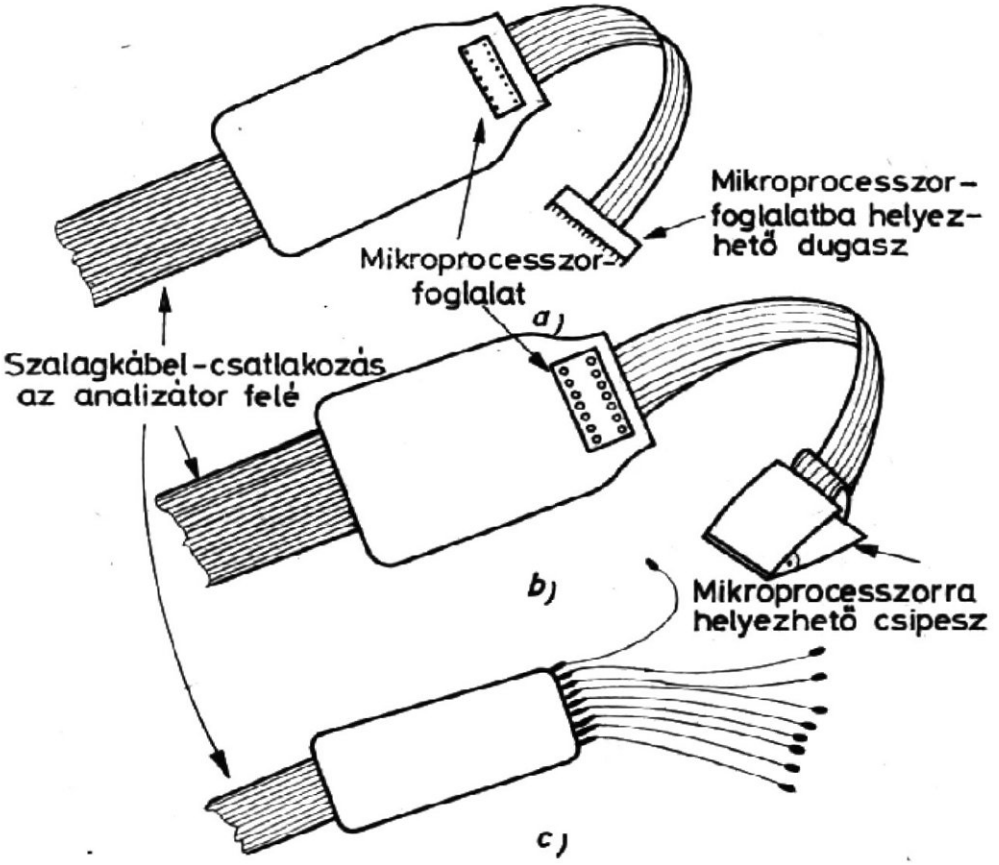
Az itt bemutatott logikai állapotanalizátor egyike a legkorszerűbb típusoknak. A sokkal egyszerűbb típusoknak nemcsak kevesebb mérőpontjuk, kisebb számú feltételkombinációjuk van, hanem a kijelzésük is kevésbé kényelmes használatot tesz lehetővé. Így ismeretesek szegmenses fényemittáló-diódás számmegjelenítőjű analizátorok is. A logikai állapotanalizátorok egyes típusai már átvezetnek a mikroprocesszor-analizátorok kategóriájába. A két kategória között azonban a határvonal nehezen húzható meg. Néhány analizátor esetén a logikai állapotanalizátor alapkonfigurációja kiegészítő modulokkal mikroprocesszor-analizáláshoz szükséges többletfunkciók és szolgáltatások elvégzésére is átalakítható.

Mikroprocesszor-analizátorok

A logikai állapotanalizátorokhoz hasonlóan a mikroprocesszor-analizátorok közül is egy típust választunk ki, melynek az ismertetésével kívánjuk bemutatni ezt a műszerkategóriát. Már az előzőekben tárgyalt analizátor kidolgozásánál is messzemenően figyelembe vették a mikroszámítógépek és programjaik kidolgozása során felmerülő problémákat. A következőkben olyan mikroprocesszor-analizátort ismertetünk, amelyet kifejezetten a mikroprocesszorok működésének az analizálására optimalizáltak. Segítségével gyorsan, könnyen megkereshetők az ellenőrizendő mikroprocesszoros berendezés hardverjének a tervezésekor, a programok megírásakor, ill. a hardver-szoftver interfész-módok kidolgozásakor elkövetett hibák, pontatlanságok [28], [29].

Az analizátor működését mikroprocesszor vezérli. A kezelővel a kapcsolattartást egy egyszerű billentyűzet és képernyős megjelenítő segíti. A képernyős megjelenítővel lehet a mérési feltételeket megadni, ill. a mérési eredményeket kijelezni.

A mikroprocesszoros feladatokra való specializálódás a mérőszondák (érzékelők) kiviteli formájára is rányomta a bélyegét. Kétféle „mikroprocesszor-szondát" használhatunk (4.24. *a* és *b* ábra). Ha több és nagyobb hely áll rendelkezésre, a mikroprocesszorra helyezhető csipesz (általában 40 érintkezős a mikroprocesszorok kivezetésszámának megfelelően) segítségével csatlakozhatunk a mérendő jelekhez. Ha erre nincs mód, vagy csak körülményesen oldható meg a csipeszes csatlakoztatás, használhatjuk a mikroprocesszor kivezetéseihez hasonló kivitelű dugaszt is. Ez esetben a foglalatából eltávolított mikroprocesszor helyére csatlakozik a dugasz, a mikroprocesszort pedig a szondán lévő foglalatba kell helyezni azért, hogy a felhasználói



4.24. ÁBRA. A mikroprocesszor-analizátor csatlakozószondái
a) mikroprocesszor-szonda dugasszal, *b)* mikroprocesszor-szonda csipesszel, *c)* külső hardverjelek szondája

rendszer továbbra is működőképes maradhasson. (Ha esetleg a csipeszes szonda alkalmazásakor is valamilyen okból kifolyólag kényelmesebb a mikroprocesszor eltávolítása, használhatjuk az ezen a szondán lévő foglalatot is.)

Az analizátor a mikroprocesszor jeleinek az analizálásán túlmenően a felhasználó által tesztelni kívánt hardverjelek (max nyolc) nyomkövetését is elvégzi. A 4.24. *c* ábrán látható harmadik szonda nyolc, egy jel fogadására alkalmas szondavezetékkel és egy földpontvezetékkel csatlakoztatható a vizsgálni kívánt pontokhoz.

Érdekes a kijelzés. A mérés eredménye – a begyűjtött adatok – több formátumban jelezhető ki. A képernyőn a mérés befejeztével a CIM-vezetékek és ADAT-vezetékek tartalma, a kiegészítő (vezérlőjelre, vagyis a gépi ciklusra vonatkozó) információ és a külső hardverjelek értéke jelentethető meg. A cím- és az adattartalom oktális vagy hexadecimális, míg a hardverjelek bináris formában rajzolhatók fel a képernyős megjelenítőre, a kiegészítő információ pedig szöveges felirat.

Az adatvezetékek tartalma a forrásnyelvű programokban használatos emlékeztető szimbólumok (mnemonikok) formájában is kiíratható. Ehhez az analizátor belső mikroprocesszorát a mérendő mikroprocesszor típusától függő visszafordító programmal is ellátták. A 4.25.*a* és 4.25.*b* ábrákon láthatók ugyanannak a programrésznek a nyomkövetési adatai hexadecimális, ill. emlékeztető szimbólumok formájában. Az ilyen formátumú kijelzés tömörebb, hiszen a többbyte-os utasítások kiíratásához csak egy sorra van szükség. A kiíratás hasonlít a programlistára, bár tartalmazza azokat a „többletállapotokat" is, amelyek nem az utasításbyte-ok olvasását jelentik, hanem egyéb gépi ciklusok (veremtár vagy adattár írása, ill. olvasása, be-, ill. kimeneti műveletek és megszakításazonosítás).

Ennek a kijelzésnek a nagy előnye, hogy a programozóknak nem kell fejben visszafordítaniuk a gépi kódot, elvégzi helyettük az analizátor. A programozók nagy része sohasem használja a gépi kódot, ezért számukra annak kezelése nehéz és lassú. Az emlékeztető szimbólumok azonban még akkor is könnyen és hamar megtanulhatók, ha a vizsgálat tárgyát képező programot nem assembler nyelven, hanem valamely magas szintű programnyelven írta meg eredetileg a programozó.

Az adatgyűjtés indításához (triggereléshez) hat feltételmegadó nyomógomb tartozik. Ezekkel a címvonalak tartalmára egyenlőségi feltétel, ill. kisebb vagy egyenlő és nagyobb vagy egyenlő feltétel adható meg. Az adatvonalak tartalmára és a külső hardverjelekre csak egyenlőségi feltétel adható meg. A triggerfeltételbe teljesülési sorszám is bekeverhető. A hat feltételmegadó nyomógomb mellett a paramétermegadásra a korábban tárgyalt analizátor ENTRY billentyűzetéhez hasonló billentyűcsoport tartozik. Az egyedüli különbség az, hogy az FS nyomógomb hiányzik. Az indítási feltételkeresés a mikroprocesszor-analizátoron belül tiltható, ill. engedélyezhető. (A tiltás, ill. engedélyezés funkciókkal a már leírt ún. szekvenciális indítás valósítható meg.) A tiltás, ill. engedélyezés funkciókhoz 3-3 nyomógomb tartozik; ezekkel címegyenlőségi, adategyenlőségi és külső hardverjelekre vonatkozó egyenlőségi feltételek adhatók meg. Az adatgyűjtés a triggerponthoz képest max. 63 állapottal korábban kezdhető, ill. max 65 472 állapottal késleltethető. Ezek megadásához is egy-egy funkcionális nyomógomb tartozik. A feltételek előírását a képernyős megjelenőn látható felirat is segíti. A 4.25. *a* és 4.25. *b* ábrák fejlécén címegyenlőségre, ill. címtartományra (kisebb vagy egyenlő és nagyobb vagy egyenlő feltételek együttes megadása is engedélyezett) vonatkozó előírást láthatunk.

Az analizátorral való mérés négy különböző „futtatási módban" végezhető. Nyomkövetési módban a triggerponttól kezdve (korábban kezdett vagy késleltetett indításnál értelemszerűen a pontot megelőzően vagy ahhoz képest késleltetve) a 64 egymást követő állapot az analizátor tárába kerül. Triggerelt nyomkövetés módban a triggerfeltétel nem a nyomkövetés indítását és az azt követő állapotok egymás utáni

TRIGGER	ADRESS 024A	DATA	EXTERNAL

ADRS	OPCODE/DATA		EXTERNAL
024A	C3	OPCODE	00000000
024B	44	READ	00000000
024C	03	READ	00000000
0344	32	OPCODE	00000000
0345	BC	READ	00000000
0346	05	READ	00000000
05BC	A5	WRITE	00000000
0347	3E	OPCODE	00000000
0348	1B	READ	00000000
0349	CD	OPCODE	00000000

a)

TRIGGER	ADDRESS < = 024A > = 0FFF	DATA	EXTERNAL

ADRS	OPCODE/DATA		EXTERNAL
024A	JMP	0344	0000 0000
0344	STA	05BC	0000 0000
05BC	A5	WRITE	0000 0000
0347	MVI	A, 1B	0000 0000
0349	CALL	07B2	0000 0000
0FF9	4C	WRITE	0000 0000
0FF8	03	WRITE	0000 0000
07B2	CPI	0D	0000 0000
07B4	JNZ	07C3	0000 0000
07C3	CPI	0A	0000 0000

b)

4.25. ÁBRA. A mikroprocesszor-analizátorral nyomkövetett program kijelzése
a) **hexadecimálisan,**
b) **emlékeztető szimbólummal**

memorizálását jelenti az analizátornak, hanem csak azoknak az állapotoknak a „begyűjtését" kívánja, amelyek az előírt feltételnek eleget tesznek. Ez az üzemmód a logikai állapotanalizátor szelektív nyomkövetési módjának felel meg. A nyomkövetés tiltás, ill. engedélyezés feltételekkel kombinálva igen jól írható elő. Ebben a futtatási módban – a nem összefüggő állapotok memorizálása miatt – nem használható (nincs is értelme) az emlékeztető szimbólumok megjelenítése.

A következő két futási mód időmérésre, ill. triggerpont összeszámlálásra vonatkozik (4.26.*a* és 4.26.*b* ábrák). Az időmérés a triggerengedélyezési és -tiltási pontok között eltelt időtartamra vonatkozik. Ezzel a két móddal a programok optimalizálásához és a programfutási hurkok vizsgálatához komoly támogatást nyújthat az analizátor.

A mikroprocesszor-analizátor képes valamennyire beavatkozni a mikroprocesszor futásába is. A mikroprocesszor-típusnak megfelelő ún. personality-modul ui.

	ADDRESS	DATA	EXTERNAL
ENABLE	2ABC	78	
TRIGGER	XXXX		
DISABLE	012A		

TIME = 879 MICROSECONDS

a)

	ADDRESS	DATA	EXTERNAL
ENABLE	2ABC	78	
TRIGGER	10F0		
DISABLE	809E		

COUNT = 47 840 EVENTS

b)

4.26. ÁBRA. Kijelzés *a)* időmérés eredménye, *b)* eseményszámlálás eredménye

4.7. TÁBLÁZAT. A HP 1611A típusú analizátor jellemzői [28], [29]

Jellemzők	Értékek
A mikroprocesszor és az analizátor közötti kapcsolat	40 kivezetéses DIL dugasz, 40 érintkezős IC csipesz
Tárméret	64 állapot 32 bit/állapot
Képernyőn a megjelenítés	16 sor max cursorral mind a 64 állapot kijelezhető
Időmérés	
pontosság	0,1 %±1 μs
max idő	kb. 16,7 s
Állapotszámlálás	max 16,7 millió állapotig
Külső hardverjelek szondája	
bemeneti impedancia	1 MΩ, 25 pF
küszöbjelszintek	logikai *1* 2,4 V...5 V logikai *0* — 0,8 V...0,8 V
beállási idő	min 250 ns az órajel lefutóéle előtt és 0 ns az órajel lefutóéle után

megállíthatja a mikroprocesszor futását. A megállítás a triggerponthoz rendelhető. Ezzel lehetővé válik akár a lépésenkénti programvégrehajtás s ezzel párhuzamosan a felhasználói hardver lépésenkénti vizsgálata is. A hardver beméréséhez további segítséget is kaphatunk: az analizátor triggerkimeneti pontjairól oszcilloszkópot indíthatunk, s a gyanús jeleket azzal vizsgálhatjuk. A kimeneti pontról a felhasználói rendszer egyik bemenetének vizsgálatához szükséges jel is generálható.

A 4.7. táblázatban röviden összefoglaljuk egy tipikus logikai és mikroprocesszor-analizátor főbb jellemzőit. A 4.8. táblázat pedig, a példaként kiválasztott Intel 8080A mikroprocesszor personality-moduljának a specifikációs paramétereit tartalmazza. Hasonló personality-modulokat más típusú mikroprocesszorokhoz is kidolgoztak.

4.8. TÁBLÁZAT. A HP 1611A logikai állapotanalizátor Intel 8080A personality modulja [28], [29]

Jellemzők	Értékek
Órajel Φ_2	
frekvencia	300 kHz...4 MHz
impulzusszélesség	min 75 ns high min 75 ns low
küszöbjelszint	logikai *1* 9 V...13 V logikai *0* –1 V...0,8 V
bemeneti impedancia	12 kΩ, 25 pF
Adat-, címbenetek, WAIT, READY, HLDA, INTE, SYNC jelek	
küszöbjelszint	logikai *1* 3 V...6 V logikai *0* –1 V...0,8 V
bemeneti impedancia	1 MΩ, 25 pF
Beállási idők (a megfelelő órajelélekhez viszonyítva)	
READY jel	100 ns órajel előtt, 0 ns órajel után
többi jel	100 ns órajel előtt, 25 ns órajel után
READY kimenet	TTL nyitott kollektor, 8 mA
Kimenetek	
jelszintek (50 Ω)	logikai *0* 0,4 V logikai *1* 2,0 V
TRIGGER impulzus	75 ns széles

A personality-modulok az alábbi feladatokat látják el:

- mikroprocesszor szondákkal csatlakoznak a felhasználói rendszerhez,
- megfelelő fogadóáramköröket tartalmaznak (jelszint fogadás),
- mintavételi jelet állítanak elő az órajel (és esetleg más vezérlőjelek) alapján,
- megfelelő ROM-ba égetett programot tartalmaznak a gépi kód – emlékeztető szimbólum konverzióhoz.

Ezeknek a feladatoknak a megvalósításához a mikroprocesszor-analizátor personality-moduljai az alábbi részekből állnak [30]:

- mikroprocesszor-szondák,
- mikroprocesszor-jelfogadó (personality) áramköri kártya,
- ROM rész,
- előlap-vezérlőegység.

Az előlap-vezérlőegység előlapra helyezett kezelőszerveket is tartalmaz. Ezek a billentyűzettől különálló kezelőszervek egyediek, a personality-modul típusától függően.

A mikroprocesszor-analizátorok után vizsgáljuk meg a logikai állapotanalizátorok egy másik csoportját, a sínrendszer-analizátorokat.

Sínrendszer-analizátorok

A különböző típusú szabványosított vezérlő és adatátviteli csatornák és sínrendszerek a mérés- és irányítástechnika, valamint a számítástechnika berendezéseiben egyre inkább terjednek. A mikroprocesszoros technika, az LSI és a VLSI eszközök megjelenése, elterjedésük is komoly kölcsönhatásban van a meglevő és az újonnan megjelenő szabvány sínrendszerekkel. Mindez indokolja, hogy a nagybonyolultságú áramköri alkatelemek kapcsán foglalkozzunk azokkal a speciális segédeszközökkel, amelyek specifikációját, szolgáltatásait a sínrendszereken alapuló készülékek, berendezések beméréséhez, ellenőrzéséhez alakították ki.

Az egyik legkomolyabb problémát a soros adatátvitelek ellenőrzése jelenti. Igaz ugyan, hogy ez esetben a soros vonal jelváltásának a megfigyelésére – az átviteli vezetékek száma alapján – az oszcilloszkópok alkalmasak lennének, de a képernyőre rövid időre kirajzolt kép alapján a gyorsan változó információk megfigyelése és ellenőrzése szinte lehetetlen. A megfigyelés csak akkor válik lehetővé, ha ciklikusan ugyanazt az adatot (kódot) tudjuk a vizsgálat időtartama alatt a vonalra kiadni. Ez viszont nem mindig lehetséges, ill. nem mindig nyújt kellő információt a vonal viselkedésére vonatkozóan. Pl., ha egy számítógép és valamelyik terminálja között a kapcsolat meghibásodott, bizonyos esetekben nehéz lehet annak a megállapítása, hogy a soros vonalon van-e a hiba, vagy a terminál, ill. a számítógép vonali illesztőegységében rejlik a helytelen működés oka.

Az előző két pontban ismertetett logikai, ill. mikroprocesszor-analizátorok családjába tartoznak a soros vonali jelek analizátorai is (Serial Data Analyzer) [31]. A készülék a CCITT V.24 szabványú soros átviteli vonalra csatlakoztatható. Rendszerbe kötve a soros vonal adatait összegyűjtheti, és tárba tárolhatja későbbi kijelzés céljából. Kétféle üzemmódban működtethető: passzív és aktív üzemmódban.

Passzív módban csak „hídként" köti össze a megszakított adatátviteli vonal rácsatlakoztatott két végét, és az átvitelbe való beavatkozás nélkül memorizálja az adatokat, max 2048 karaktert. Ebben az esetben a vonal mindkét irányú adatátvitelét figyeli, majd kijelzésnél külön megjelöli normál és inverz betűkkel (fekete betű fehér alapon) a különböző irányokhoz tartozó adatokat. A kijelzési sorrend megfelel a „párbeszéd" idősorrendjének.

Aktív üzemmódban helyettesítheti a vonalon levő egyik állomást. A tárába beírt választ (akár a max 2048 karakter sorozatából álló egy vagy több válaszblokkot is)

küldi vissza automatikusan az általa szimulált készülék helyett. A választ a tárban vagy egy előző mérés során begyűjtött adatsorozat jelenti, vagy pedig a kezelő billentyűzetről írhatja be azt a mérés megkezdése előtt. További lehetőség, hogy a kezelő PROM-ba égeti a szükséges válaszkódokat és ezt a PROM-ot helyezi be az analizátor megfelelő PROM foglalatába.

A karaktersorozat kijelzésével a kezelőnek lehetőséget nyújt az analizátor arra, hogy megvizsgálja az átvitt kódok helyességét. Az analizátortól azonban a kódokon túlmenően más átviteli paraméterek ellenőrzését is kérhetjük. Ezek:

- a párbeszéd karakterei közötti időtartam,
- blokkos átvitelnél a blokkvég és a válaszblokk kezdete közötti időtartam,
- az átviteli sebesség (és bitváltási időpontok) helyességének ellenőrzése, az időhibához tartozó soros adat kijelzésével együtt,
- a paritáshiba ellenőrzése.

A következő adat- és kódformátumok kezelésére alkalmas az analizátor:

- ASCII kódok,
- EBCDIC kódok,
- hexadecimális kódok,
- felhasználó által kért kódok,
- szinkron adatátvitel,
- aszinkron adatátvitel,
- paritás nélküli kódok,
- paritásbites (páros vagy páratlan) kódok,
- többféle ciklikus redundanciakód generáláson (CRC) alapuló hibajelzéses üzenetek közvetítéséhez és vizsgálatához,
- 19,2 kbaud sebességig a szabvány szerinti bármelyik adatátviteli sebesség lehet.

A mérésindítási feltételeket (triggerfeltételek) a korábban tárgyalt analizátorokhoz hasonlóan sok paraméter függvényében adhatjuk meg, így:

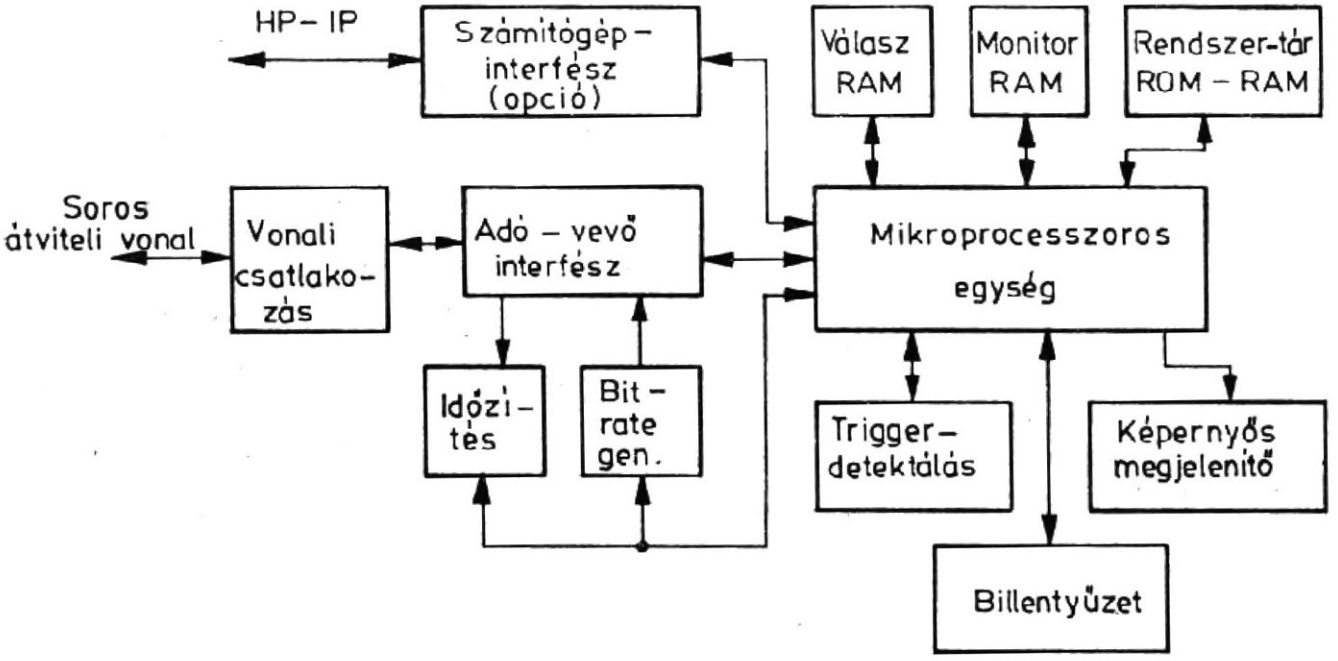
- karakterkód formájában,
- adatátviteli irány és bitkombináció formájában,
- paritáshiba,
- CRC hiba,
- időzítési hiba,
- karaktersorozat megjelenésének a formájában.

A karaktersorozatban „sorozathiba" is előírható triggerelési feltételként (pl.: 1 2 3 4 5 sorozat esetén triggerfeltételként az 1 2 3 $\overline{4}$ 5-öt adhatjuk meg, ahol $\overline{4}$ azt jelenti, hogy az adott helyen 4-től különböző kód jelenik meg).

A kódformátumok, triggerfeltételek, üzemmódok bonyolult megadását a már ismert „menü-technikával" segíti az analizátor. A mérési adatok feldolgozásának a meggyorsítására, valamint aktív üzemmódban a bonyolult válaszadatok definiálására számítógépes vezérlési lehetőség is nyílik.

A soros adatanalizátor bonyolult funkcióit, működését beépített mikroprocesszor, ill. annak programja vezérli. A felépítés blokkvázlatát a 4.27. ábra mutatja.

Az azonos idejű adatgyűjtés és a triggerfeltételek ellenőrzése különösebb műveleti sebességi követelményeket sem támaszt a mikroprocesszoros vezérlővel szemben. A legmagasabb működési sebesség mellett is, ha az adatátvitel 19200 baudos, egy-egy karakter átvitele kb. 0,5 ms (függ a kódformátumtól). A mikroprocesszornak és programjának ez az idő elegendő ahhoz, hogy az adatgyűjtés, válaszadás és triggerdetektálás vezérlésének funkcióit ellássa. Ennek köszönhetően a programozott vezérlés könnyen megvalósítható, és jelentősen csökkentheti az analizátor hardverjének a bonyolultságát, méreteit.



4.27. ÁBRA. A soros adatátviteli analizátor felépítése

4.6. NAGYSZÁMÍTÓGÉPES RENDSZEREK

A mikroprocesszorok programjait bármely számítógép segítségével meg lehet szerkeszteni (szerkesztő- vagyis editorprogramja minden számítógépnek van), keresztfordító programok igénybevételével pedig a forrásprogram a megfelelő gépi kódra lefordítható [19].

A félvezetőeszköz-gyártó cégek, de a nagy szoftverkidolgozó vállalatok is kínálnak keresztprogramokat különböző típusú mikroprocesszorok számára. Ezek a keresztprogram-fejlesztési segédeszközök általában gépfüggetlen magasabb szintű programnyelven készülnek, ezért gyakorlatilag bármely közép vagy nagyszámítógépen futtathatók. A nagyszámítógép-hálózatokat üzemeltető vállalatok is kibővítették a programkönyvtárukat a gyakrabban használt mikroprocesszorok keresztszoftverjével. A hálózat használói (a terminháltulajdonosok) így hozzáférhetnek és használhatják ezeket a programokat. Meg kell azonban jegyeznünk, hogy a számítógépes hálózatokon való programfejlesztés költségei – a hálózati gépidő magas díja miatt – meghaladják általában az egyéb módon végzett programfejlesztési kiadásokat.

Előfordul, hogy a mikroprocesszoros rendszerek egyéb fejlesztői segédeszközeit (pl.: az integrált fejlesztői rendszereket) számítógéppel összekötik. A nagyszámítógépek használatának előnyei: a meglévő igen komoly programerőforrásokhoz való hozzáférés, a számítógép általában nagykapacitású és gyors háttértárainak és perifériás rendszerének a használata, és nem utolsó sorban a számítógépes szakemberek komoly tapasztalainak felhasználása. Hátrány azonban a költségesebb fejlesztés, és az, hogy a számítógépes rendszer nem mindig hozzáférhető (általában sokan, más célokra is használják még).

A lefordított és szintaktikai hibáktól mentes tárgyprogramot ezután különböző módszerekkel „éleszthetjük" fel. Amennyiben van hardver, amelyen futtatható hibabehatárolás céljából, az ismert hardver, szoftver hibakeresési módszerek és eszközök igénybevehetők. A megtalált hibák ezután számítógépes támogatással kijavíthatók. A javítást újabb hibakeresési fázis követheti.

A programok másik, ettől alapvetően eltérő „belövési" módja a szimuláció. Ebben az esetben felhasználói hardver nélkül, a számítógépes fejlesztői rendszeren, annak a szimulátorprogramjával végzehető el a felhasználói program próbája, és a hibabehatárolás is. A szimuláció módszerével bővebben a 10.4. szakaszban foglalkozunk.

4.7. INTEGRÁLT FEJLESZTŐI RENDSZEREK

A mikroprocesszoros berendezések és készülékek kidolgozása, bemérése többirányú tervezési, hibakeresési tevékenységet követel meg. A mikroprocesszoros hardver megépítése és ellenőrzése, valamint a működtető algoritmusok kidolgozása, programformába öntése, a programok összeállítása és „belövése" összetett, sokoldalú feladat. Az előzőekben tárgyalt és ismertetett segédeszközök egyike sem alkalmas arra, hogy a fejlesztési munkák csaknem minden szakaszában a tervezési, hibakeresési és a működés-ellenőrzési munkákat segítse. Közülük más eszközök használhatók a programok megírására, összeállítására és lefordítására, és más eszközök segítik a működés vizsgálatát.

Az olcsó LSI és VLSI áramkörök megjelenése lehetővé tette a segédeszközök két fajtájának egyetlen berendezésben való gazdaságos megvalósítását. Ez az ún. integrált fejlesztői rendszer. Az integrált jelző arra utal, hogy a fejlesztés minden szakaszában használható rendszer magába foglalja (magába integrálja) a korábban több különböző eszközzel megvalósított szolgáltatásokat. Az ilyen segédeszköz képes a programok írásának és a javításának a segítésére, a programok lefordítására. Ezek a feladatok csak számítógéppel oldhatók meg, ezért az integrált fejlesztői rendszerek számítógép alapúak. Az alapul szolgáló számítógép – gazdasági okok miatt – általában mikroszámítógép. A hibakeresési funkciók ellátásához az integrált fejlesztői rendszereket a legkülönbözőbb egységekkel és szolgáltatásokkal (pl.: emulátorokkal, nyomkövetési funkciókkal stb.) látták el. Természetesen ezeknek az egységeknek a vezérlésében és működésük irányításában a rendszer alapját képező mikroszámítógép is részt vesz.

Az LSI és VLSI áramkörökre épülő rendszerek létrehozása a tervezésen, hibakeresésen túlmenően egyéb járulékos műveletek elvégzését is megköveteli. Ezek a műveletek részben a fejlesztési munkákhoz, részben azonban már a berendezések gyártásához sorolhatók. Ilyenek pl.: a programdokumentáció elkészítése, a programozható logikai áramkörök (FPLA), ill. a programozható és újraprogramozható csak olvasható tárak (PROM, REPROM) „beégetése". Az integrált segédeszközök legtöbbjéhez csatlakoztathatók vagy megrendelhetők az ezeket a műveleteket elvégző kiegészítő egységek is.

Fejlesztési eljárások

A mikroprocesszoros berendezések kidolgozása az alábbi fejlesztési szakaszokra osztható [32]:

- a működés hardver- és szoftverfunkciókra bontása,
- a hardver megtervezése, bemérése, működésének ellenőrzése,
- a programok megírása, lefordítása, „belövése",
- a hardver és a programok együttes próbái,
- a diagnosztikai eljárások, programok kidolgozása.

A megépített rendszer működésének az ellenőrzéséhez diagnosztikai eszközök és eljárások szükségesek. A működés ellenőrzése kiterjed mind a hardver-, mind pedig a felhasználói programok funkcióira. A diagnosztikai (vizsgálati) eszközök ugyancsak

készülékekre és diagnosztikai programokra oszthatók. Diagnosztikai készülékként különböző jelgenerátorok (bemenő adat- és vezérlőjelek előállítására) és jelfogadó-kijelzőeszközök működhetnek, de maga az integrált fejlesztési segédeszköz is használható ilyen célra. Az alkalmazástól függően a diagnosztikai műveletek kisebb hányadához vagy csaknem a teljes diagnosztikához elegendő a fejlesztői rendszer. A vizsgáló programok kidolgozása, megírása, sőt működésük ellenőrzése az LSI és a VLSI áramkörökre épülő berendezések tervezési és ellenőrzési munkáinak lényeges és nélkülözhetetlen része. Segítségükkel a legegyszerűbb funkciókat is kipróbálhatjuk, de olyan „üzemeltetési körülményeket" is szimulálhatunk, amelyek a kidolgozandó berendezés szempontjából kritikusak, kedvezőtlenek. Ez lehetővé teszi, hogy olyan vizsgálatokat is elvégezhessünk, amelyek adatokat szolgáltatnak a berendezés kedvezőtlen (átlagosnál kedvezőtlenebb vagy bizonyos paraméterekre vonatkoztatva legkedvezőtlenebb) viszonyok közötti működéséről is.

A diagnosztikai és a felhasználói programok megírását, lefordítását és kipróbálását azonos módon végezhetjük el a fejlesztői segédeszköz igénybevételével.

A tervezést a működés hardverműveletekre és programokkal vezérelt funkciókra való felosztásával kezdjük. Ehhez a szakaszhoz kevesebb segédeszközre van szükség. Bár a műveletek felosztása nem mindig egyszerű, a kielégítő eredmény gyakran többszöri próbálkozás után található meg. Ilyenkor a hardver és a szoftver helyes arányának a megkereséséhez is kaphatunk segítséget a fejlesztői rendszertől. Pl.: a hardver- és szoftverfelosztástól a kidolgozandó berendezés működési sebessége igen komoly mértékben függ. Annak a megválaszolásához, hogy a feladat programmal való megoldása kielégítő működési sebességhez vezet-e vagy sem, célszerű a fejlesztésnek ebben a szakaszában a kérdéses programrészek futási idejét megmérni vagy megbecsülni. Ehhez ezek a segédeszközök igen jól használhatók. Az eredménytől függően ezután eldönthető, hogy melyek azok a feladatok, amelyek megvalósítását a gyorsabb hardverre kell bíznunk.

A hardver- és a szoftverfelosztás helyességének ilyen próbaprogramokkal való ellenőrzése nagyon hasonló a programok diagnosztikai vizsgálatához. A mikroprocesszoros berendezések kidolgozási szakaszaiban, az integrált fejlesztői segédeszköz használata kapcsán a következő feladatok lehetnek:

- a hardver kidolgozása, próbái,
- a programfejlesztési munkák,
- a hardver és a programok együttes próbái és
- az egyéb (járulékos) munkák [33].

Az integrált fejlesztői rendszerek használata három alapvető eljárás vagy módszer megvalósítását teszi lehetővé, ezek:

- a fokozatos rendszerépítés módszere,
- az áramköri emuláció módszere,
- az ellenőrzött programfuttatás módszere [34].

A fokozatos rendszerépítés és az áramköri emuláció módszere egymással szoros kölcsönhatásban van. A fokozatos rendszerépítés (akár készülékhardverről, akár programról van szó) hatékonyabbá teheti a kidolgozási munkát. Egy-egy kisebb egység vagy blokk megépítése, a meglevő és kipróbált részhez való hozzáadása meggyorsíthatja a hibakeresést és ellenőrzést, hiszen az új rész működése és kapcsolata a már meglevővel könnyebben áttekinthető, mint egy bonyolult, összetettebb modulé.

Programok írásakor, lefordításakor és ellenőrzésekor a fokozatos programkiépítés, bővítés különösebb gondot nem okoz. A számítógéppel támogatott programösszeállítás és az „automatikus" programfordítás a hozzáadást, módosítást igen megkönnyíti. Hardver esetében a fokozatos rendszerépítés elengedhetetlen feltétele a részegységek működésének a kipróbálása. Tekintve, hogy programvezérelt mikro-

processzoros berendezések létrehozásáról van szó, a részegységek (pl.: a központi egység) próbája elképzelhetetlen a hiányzó egyéb modulok (tár-, be-, ill. kimeneti egységek) funkcióinak „imitálása" nélkül. Ezt az „imitálást" valósítja meg az emuláció módszere. A megépített központi egység próbájához a tár- és a be-, ill. kimeneti műveleteket, valamint a programot a fejlesztői segédeszköz emulálja mindaddig, amíg azok nem készülnek el [35], [36].

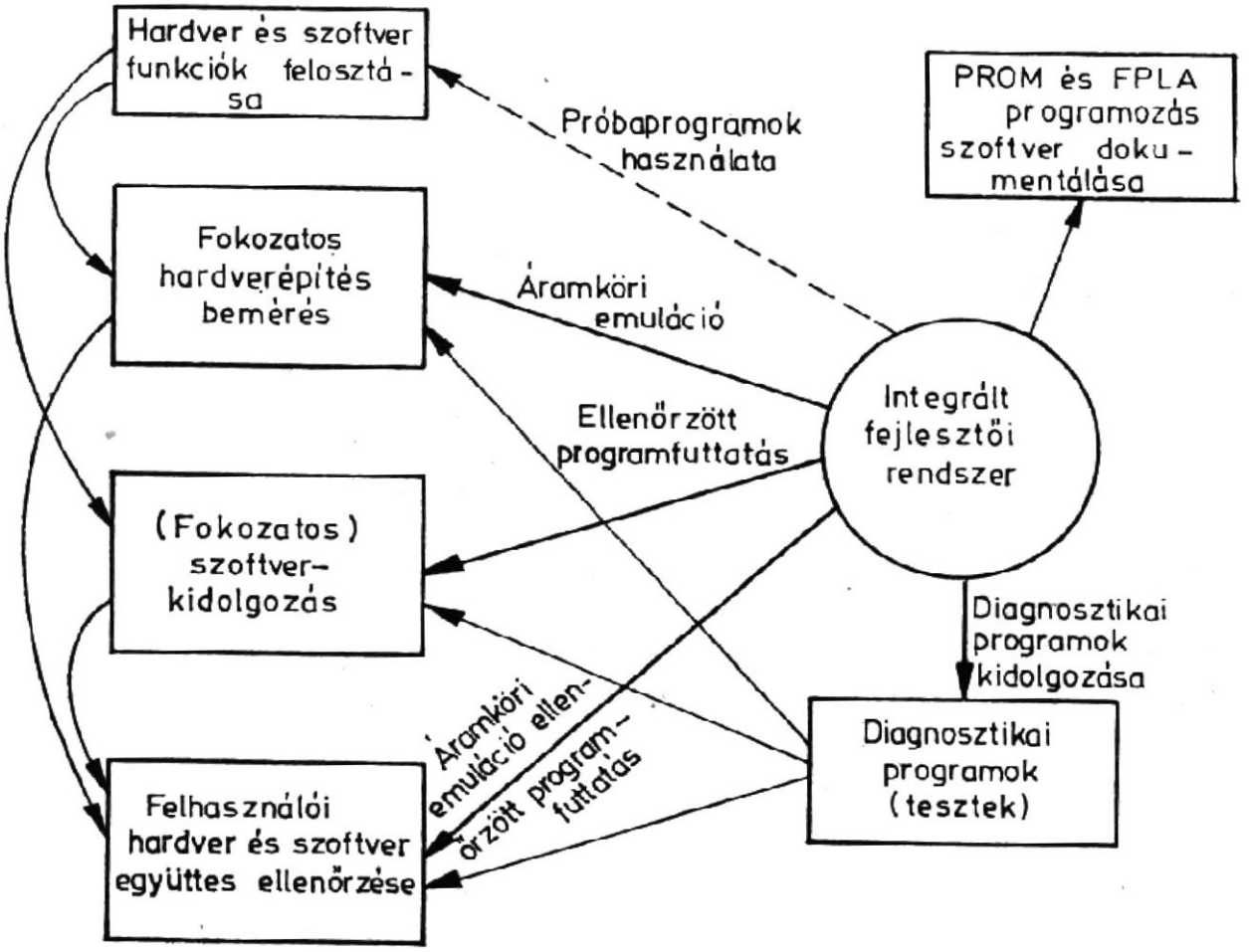
Az áramkörön belüli emuláció (in- circuit emulation, in- prototype emulation) módszere nemcsak a fokozatos rendszerépítést teszi lehetővé. E módszer – mikroprocesszorok esetében – feltétlenül szükséges az ellenőrzött programfuttatási eljárás megvalósításához is, de a felhasználói rendszer tényleges működési idejének megfelelő (azonos idejű) mód „aktív" kipróbálásához is nélkülözhetetlen. Az aktív jelző itt arra utal, hogy a vizsgálat tárgyát képező berendezés működésének passzív megfigyelésén és nyomkövetésén túl, az emulátoregység képes beavatkozni annak a működésébe is. Ez a beavatkozás a működés megszakítását és az adatok és vezérlőjelek megváltoztatását, valamint újraindítást jelenthet. Az áramkörön belüli emuláció a központi egység funkcióit ellátó mikroprocesszor helyettesítésére vonatkozik. Lényege, hogy a mikroprocesszor köré épített áramkör megbolygatása nélkül, az adott áramkörön belül helyettesítjük a mikroprocesszort a fejlesztői berendezés emulátoregységével. A tényleges működési sebességnek megfelelő futás úgy érhető el, hogy folyamatos működés közben (tehát nem a beavatkozás pillanatában) az emulátor az áramkörhöz csatlakozó felületén tökéletesen helyettesíti a mikroprocesszort; úgy viselkedik, mintha azt el sem távolították volna. Az emuláció célja kettős: a passzív megfigyelés lehetővé tétele és a beavatkozás (a későbbiekben bővebben szólunk még erről). Ezeken túlmenően további előnyei is vannak. Annak köszönhetően, hogy a központi vezérlő mikroprocesszor adat-, cím- és vezérlőjelei képviselik a csatlakozási felületet, az emulátoregységben könnyen megvalósítható a tár, ill. a be-, ill. kimeneti műveletek helyettesítésének a megszervezése is a segédeszköz megfelelő egységeinek (tár, interfészegységek) a felhasználásával. Így pl.: ha valamelyik tárrésznek a helyettesítésére van szükség, az emulátoregység az adott tárcím és -művelet pillanatában az adatátvitelt nem a felhasználói rendszer áramköreinek az aktivizálásával végzi el, hanem a segédeszköz tárához indít egy átviteli műveletet.

Az ellenőrzött programfuttatási eljárás lényege két követelményként fogalmazható meg. Ezek:

- a program indítható legyen tetszőleges kiindulási adatokkal és tetszőleges helyről,
- a program leállítható legyen előre definiált tetszőleges helyen (előre definiált feltételek teljesülésének pillanatában), ill. a megállás pillanatára jellemző adatok hozzáférhetők legyenek további vizsgálat céljából [37].

A kiindulási adatok megadása és a leálláskor megvizsgálandó adatok vonatkozhatnak a tárban levő értékekre, be-, ill. kimenő információra, de a mikroprocesszor belső regisztereinek a tartalmára és bizonyos vezérlőjeleinek az állapotára is. A mikroprocesszor belső adataihoz a hozzáférés az áramköri emulátorral és ezeket az adatokat kiolvasó rövid programrutinokkal valósítható meg.

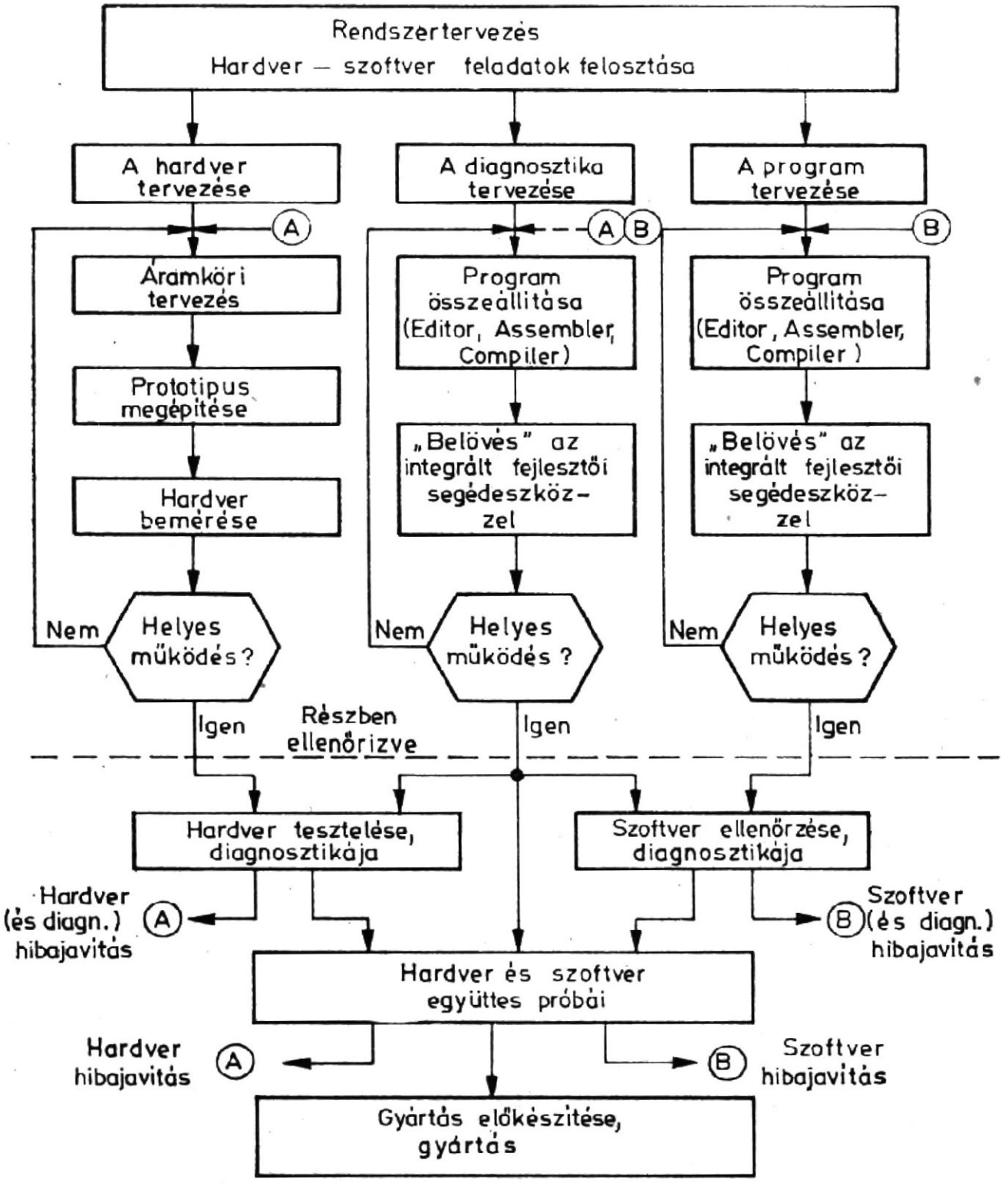
A 4.28. ábrán tüntettük fel a mikroprocesszoros készülékek kidolgozásának szakaszait, valamint az integrált rendszer fejlesztés közbeni használatának a vázlatát. A vázlat nyilai mentén látható feliratok az adott szakaszban alkalmazott módszerre vonatkoznak. A fejlesztési szakaszokból kiinduló nyilak a tervezés, bemérés munkáinak az időbeni sorrendjét ábrázolják. A leegyszerűsített ábrán nem tüntettük fel azokat a kölcsönhatásokat, melyek egy-egy tervezési hiba észlelésekor jelentkezhetnek. Ilyen pl.: ha az együttes ellenőrzés során észlelt hibák kijavítása miatt a hardverbemérés vagy a szoftverkidolgozás szakaszába vissza kell térni. A vázlaton jól meg-



4.28. ÁBRA. Az integrált fejlesztői rendszer használata és a kidolgozás szakaszai közötti kapcsolat

figyelhető, hogy a hardverépítés és a szoftverkidolgozás időben független egymástól. Ez annyit jelent, hogy az integrált fejlesztői rendszerek használata lehetővé teszi e kétféle munka egyidőben való (párhuzamos) elvégzését, s így a teljes kidolgozásra fordított idő csökkentését. Más módszerekkel és az integrált segédeszköztől eltérő fejlesztői berendezések használatával a végrehajtásukhoz szükséges hardver megépítése nélkül nem ellenőrizhetők a programok hatékonyan és megbízhatóan. Az áramköri emuláció alkalmazásával a központi programvégrehajtó egység, a tár és a be-, ill. kimeneti interfészegységek emulálásával azonban hardver hiányában is megvalósítható a hatékony programkidolgozás és ellenőrzés. Kissé részletesebben tünteti fel a 4.29. ábra a fejlesztés menetét. Itt jól követhető a párhuzamos egy időben történő hardver- és szoftverfejlesztés. A diagnosztikai programok kidolgozásának a vázlatában figyelmen kívül hagytuk azokat a rövid tesztprogramokat, amelyek a hardver-, ill. a felhasználói programok fokozatos kidolgozása közben használhatók a funkciók ellenőrzésére. A hardver–szoftver párhuzamos kidolgozásának a fázisában a helyes működést elegendő részlegesen ellenőrizni, ez további előnye a rendszer használatának. A párhuzamos hardver- és programkidolgozáson túlmenően a prototípus kidolgozására és bemérésére fordított idő és energia tovább csökkenthető úgy, hogy az együttes próbákat megelőzően csak részben végezzük el a felhasználói hardver és a programok ellenőrzését. Az integrált fejlesztési segédeszközök hatékony segítséget nyújtanak ahhoz, hogy az együttes próbák során akár a hardver, akár a szoftver kidolgozása, esetleg a kettő közötti interfésznek a megtervezése közben elkövetett hibákat, pontatlanságokat megtaláljuk. Az így megtalált hibákat azután kijavíthatjuk, visszatérve egy korábbi kidolgozási szakaszra (az ábrán A és B betűkkel jelöltük ezeket a visszatérési pontokat).

Előfordulhat, hogy éppen az ellenőrzési célokra szolgáló diagnosztikai programok kidolgozása közben fordultak elő pontatlanságok, figyelmetlenségek vagy előre

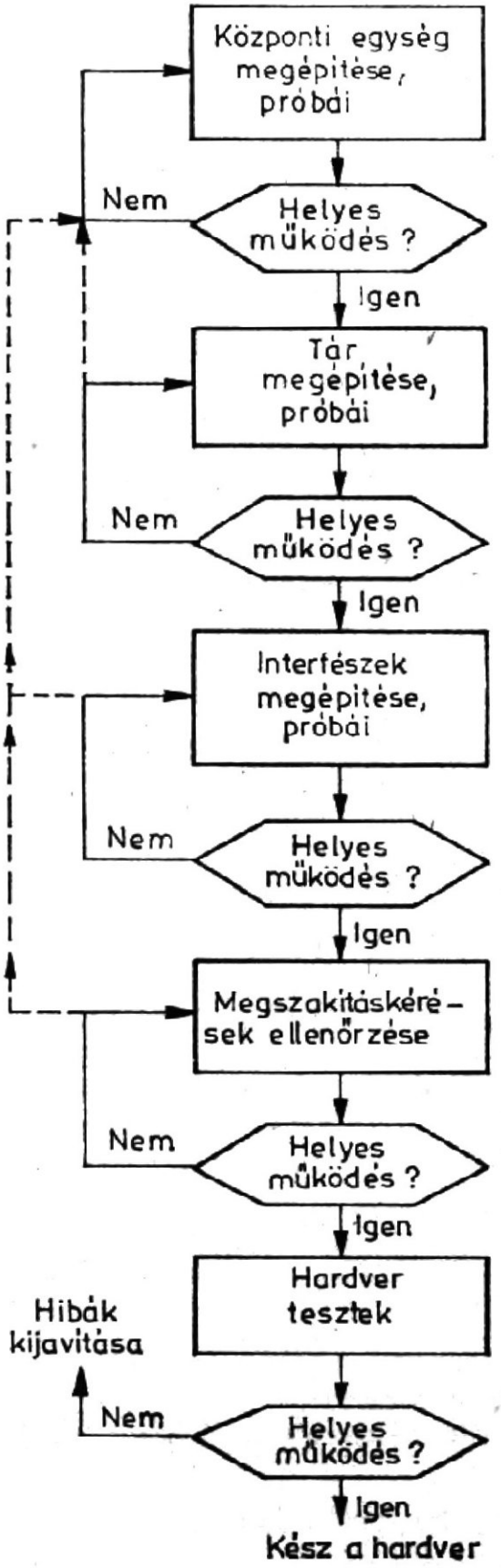


4.29. ÁBRA. Az integrált fejlesztői rendszerrel támogatott kidolgozási munkák menete [38]

nem látott ellenőrzési funkciók miatti hiányosságok. Ez esetben vissza kell térnünk a diagnosztikai programok kijavításához vagy átdolgozásához.

A hardver prototípusának a megépítése és bemérése is elvégezhető részenként. Az ilyen jellegű, kis részekre bontáson alapuló hardverépítés előnye a könnyebb és – alkalmazástól függően – esetleg gyorsabb bemérés, hibakeresés, ill. csoportmunka esetén a részek párhuzamos, egyidejű fejlesztése, ellenőrzése. A 4.30. ábrán példaként azt az esetet tüntettük fel, amikor a mikroprocesszoros készülék kidolgozását a következő részletekben oldjuk meg:

- a mikroprocesszort közvetlenül kiszolgáló, központi egységet alkotó blokk kidolgozása,



A mikroprocesszor áramköri emulálása és a tár-(be – ill. kimeneti) műveletek fejlesztői rendszerrel való emulálása

Működésellenőrzés diagnosztikai programmal

A mikroprocesszor áramköri emulálása

Működés-ellenőrzés tárteszt-programokkal

A mikroprocesszor áramköri emulálása

Működés-ellenőrzés be-,ill. kimeneti tesztekkel (megszakításkérés nélkül)

A mikroprocesszor áramköri emulálása

Működésellenőrzés be-,ill. kimeneti tesztekkel, megszakításkérés tesztjeivel

---

A mikroprocesszor áramköri emulálása

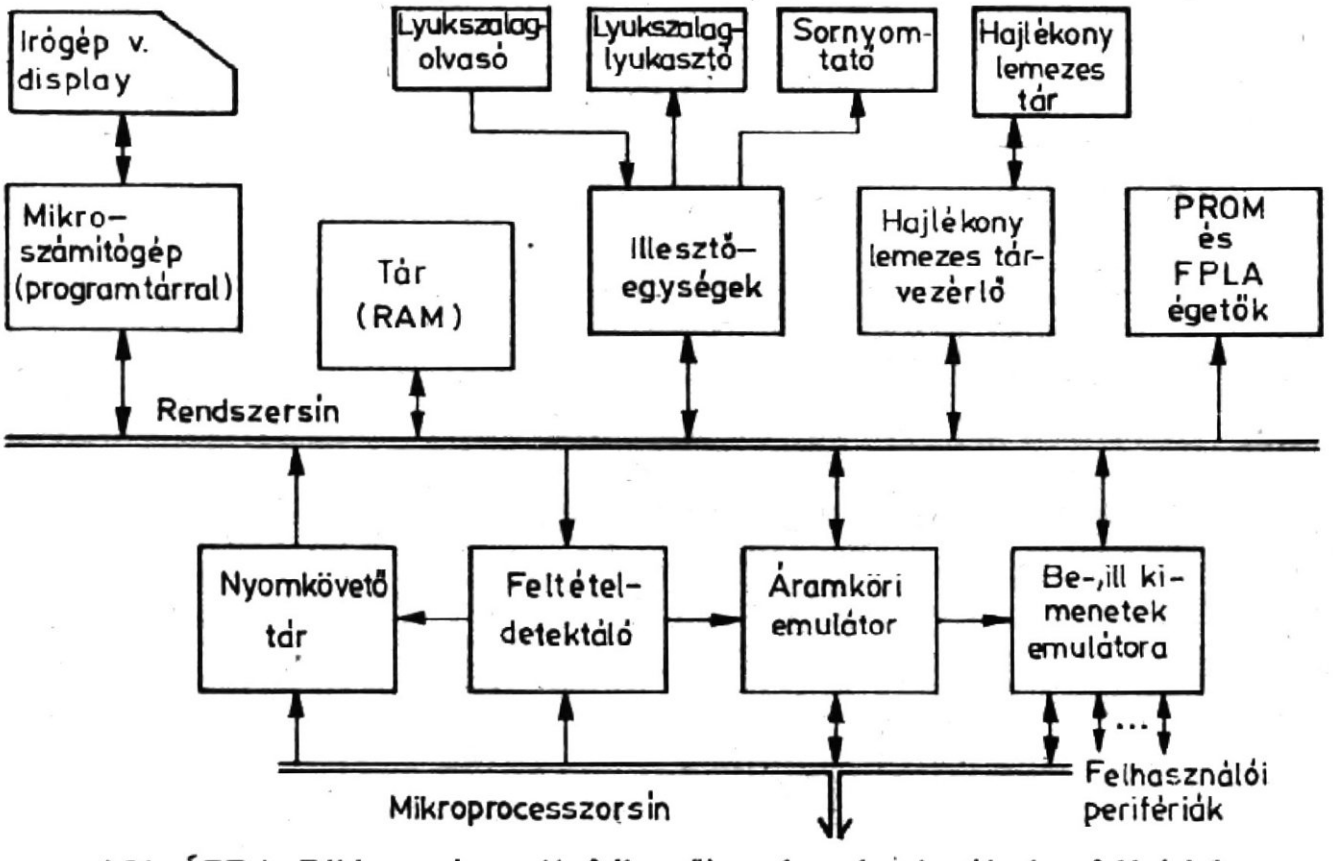
Működésellenőrzés hardver teszt-programjaival

4.30. ÁBRA. Példa a fokozatos hardverépítésre

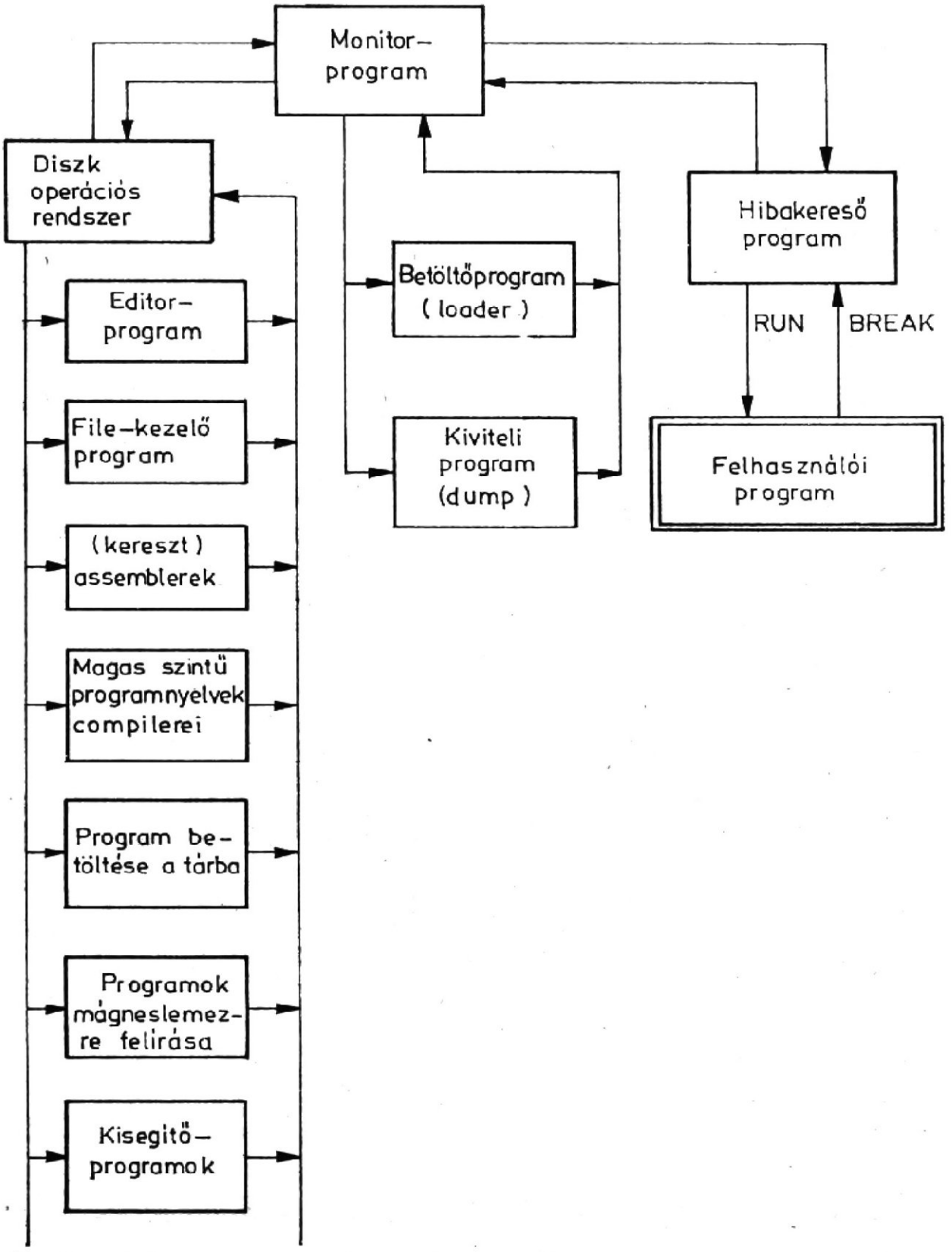
- a tár kidolgozása,
- a be-, ill. kimeneti interfészek kidolgozása,
- a megszakításkérés-rendszer próbái és
- a teljes hardver ellenőrzése,

Az ábrán a szaggatott vonallal feltüntetett nyilak azokat az eseteket jelzik, amikor egy-egy ellenőrzési lépésnél azt tapasztaljuk, hogy nem az újonnan megépített részben rejlik a hibás működés oka, hanem az előzőekben már önállóan bemért rész valamelyik olyan áramkörében, amit csak a jelen ellenőrzés során vizsgálunk. Az ábrán láthatóan az ellenőrzési munkákat mindvégig a fejlesztői rendszer segítségével, áramköri emuláció módszerével végezzük. Tekintve, hogy a fejlesztői rendszer emulátorcsatlakozójának (emulátorkábelje) ugyanolyan jelei vannak, mint az emulált mikroprocesszornak, ez megengedett. Jó emulátor és jó mikroprocesszor-áramkör használatakor biztosak lehetünk abban, hogy az emulátorkábel helyére visszahelyezve a mikroprocesszort, a hardver működés nem fog eltérni az emulátorral vizsgált működéstől.

A felhasználói és diagnosztikai programok kidolgozásakor a program-összeállítás menete megegyezik a mikro-, mini- vagy nagyszámítógépes fejlesztési segédeszközök használatakor leírtakkal. Tehát a munkák sorrendje: forrásprogram-készítés szerkesztőprogrammal (editorral), majd tárgyprogramkészítés assemblerrel, vagy magas szintű programnyelv fordítóprogramjával (compilerrel), majd a szintaktikai hibák kijavítása és újabb fordítás. Meg kell itt említenünk, hogy a kényelmes és gyors program-összeállítás elengedhetetlen feltétele a nagyobb kapacitású háttértárak használata. Mint ezt a 4.31. ábrán is feltüntettük, legáltalánosabban a hajlékony lemezes háttértárak terjedtek el, és használatosak az integrált fejlesztői rendszerekben. A kényelmes használat miatt (pl. lemezek másolásakor) szinte kizárólagos a kettős hajlékony lemezes háttértár. A program-összeállítással ellentétben a programok „belövése" eltérő a számítógépes fejlesztői rendszereknél alkalmazottól. A 4.31. ábrán az integrált fejlesztői rendszerek felépítésének egyik lehetséges megvalósítási módját mutatjuk be. A rendszersín fölött ábrázolt modulok egy hagyományos számítógép -



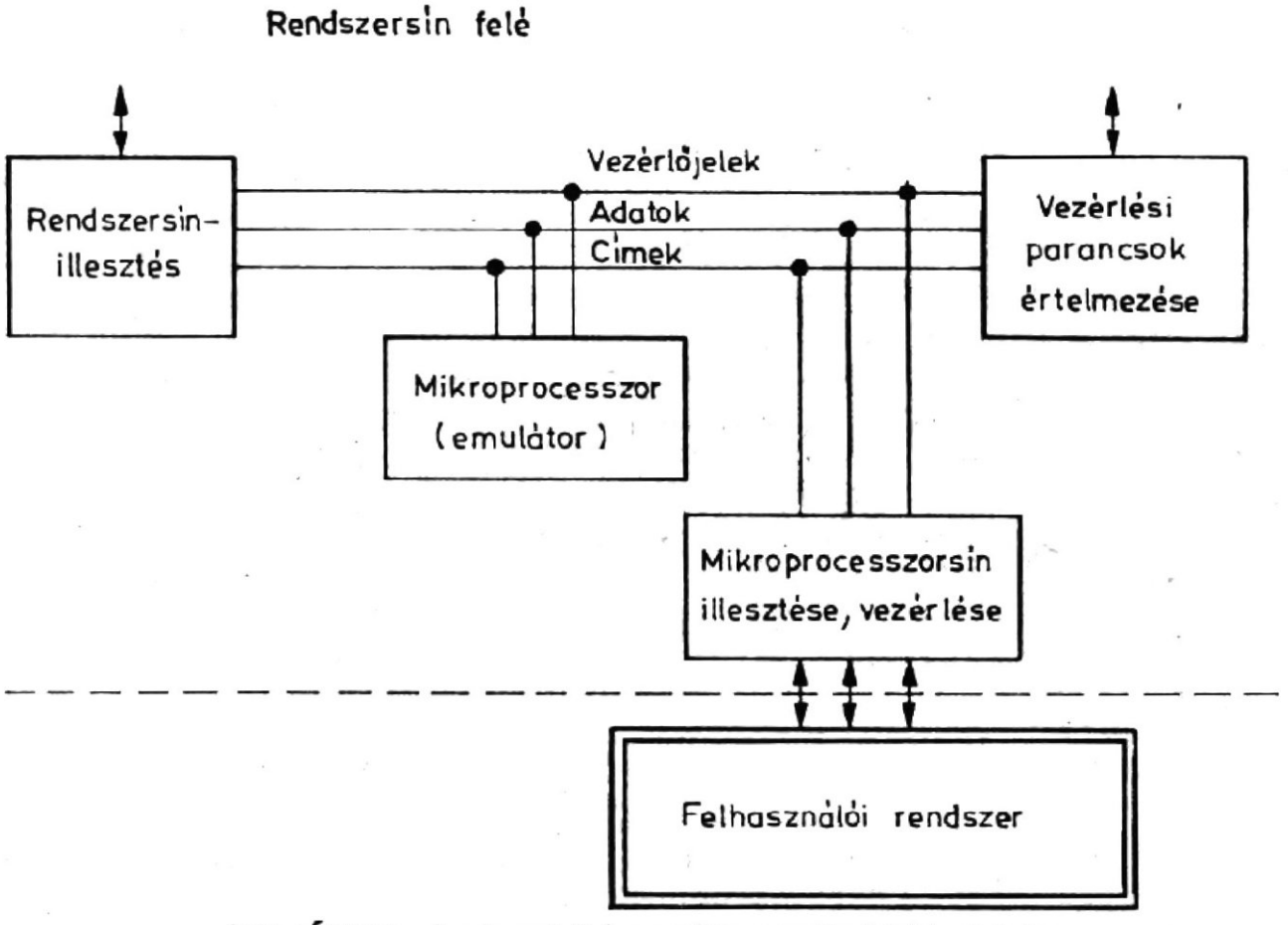
4.31. ÁBRA. Példa egy integrált fejlesztői rendszer hardverjének a felépítésére



4.32. ÁBRA. Példa egy integrált fejlesztői segédeszköz programrendszerének a felépítésére

struktúrát alkotnak. Talán az egyedüli kivételt a PROM és FPLA áramkörök beégetésére alkalmazott, perifériának is tekinthető, kiegészítő modul jelenti. A programösszeállításhoz csak ez a hagyományos számítógépes elrendezés szükséges. A belövéshez azonban igénybe vehetők a fejlesztési segédeszköznek a hibakereső modulokból alkotott, kifejezetten „élesztési", bemérési célokra használatos részei is. Az eltérést ezeknek a moduloknak és a használatuknak a sajátosságai magyarázzák. A hibakereséshez a hardver modulokon kívül nélkülözhetetlen a hibakereső program (debugprogram). Ez a program részben olyan funkciókat tartalmaz, amelyek a programozókezelő számára megkönnyítik a fejlesztői rendszer használatát, másrészt pedig vezérlik a hibakereső modulok működését. A 4.32. ábrán az integrált fejlesztői rendszerek programrendszerének egyik lehetséges változatát tüntettük fel, bemutatva a hibakereső program és az ellenőrzés tárgyát képező felhasználói program kapcsolatát is. A példán a programrendszer központi magjaként szolgáló monitorprogramból külső perifériáról (lyukszalagolvasóról) történő programbetöltés, ill. perifériára (lyukasztóra) való programkivitel kezdeményezhető. A monitorprogram lehetővé teszi a mágneslemezes operációs rendszer hívását. A mágneslemezes operációs rendszer indítja mindazokat a programokat, amelyek a hajlékony lemezeket használják. Így innen hívható:

- a szerkesztőprogram (editor),
- a file-kezelő programok (file maintenance),
- az assembler és a keresztassemblerek,
- a különböző magas szintű programnyelvek compilerjei,
- a mágneslemezről tárba töltő program (disk-loader),
- a tár tartalmát mágneslemezes tárra kiíró program (save-program),
- egyéb kisegítő programok (ezek közé tartoznak pl.: a PROM és FPLA égető modulok működését irányító égető programok vagy a programdokumentálást elősegítő programok).



4.33. ÁBRA. Az áramköri emulátoregység blokkvázlata

A mágneslemezes operációs rendszerből a monitorprogramba lehet visszalépni, ahonnan ezután a program „belövésének" a céljából át lehet térni a hibakereső programba. A hibakereső program lehetővé teszi a felhasználói program futtatását, módosítását, ill. a felhasználói program működésének az ellenőrzését, adatainak a megvizsgálását. A felhasználói programnak (vagy egyes részeinek) a végrehajtására a kezelő (programozó) ad parancsot – **RUN** parancs –, míg végrehajtása az ellenőrzött programfuttatás módszere esetén használatos töréspontig (csapda) vagy leállási pontig tart. A leállásról azután a hibakereső programnak jelzést küldenek a hibakereső modulok, amely a töréspont eléréséről üzenetet továbbít a kezelőnek, ill. a felhasználónak. A hibakeresés részletesebb tárgyalására a továbbiakban még visszatérünk.

A fejlesztői rendszerek felépítése

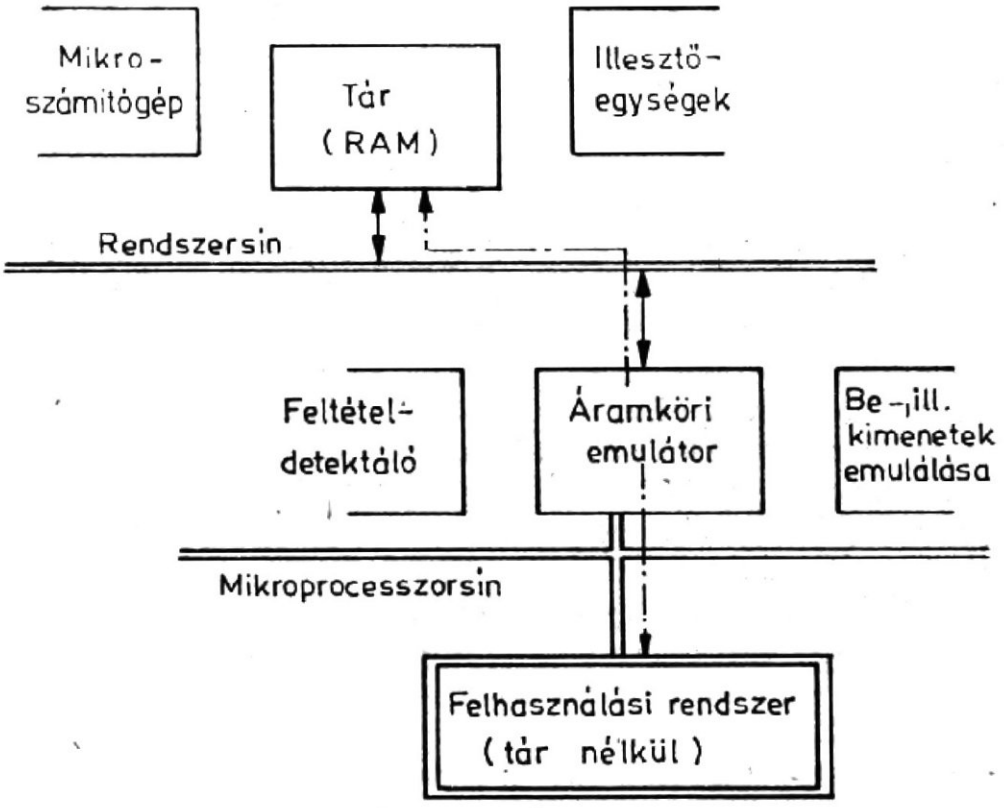
A 4.31. ábra blokkvázlatán a következő hibakereső modulokat láthatjuk:

- áramköri emulátormodul,
- feltételdetektáló modul,
- nyomkövető tár vagy eseménytároló modul,
- a be-, ill. kimenetek emulátora.

Bizonyos értelemben a hibakereső modulok közé sorolható még a mikroszámítógép RAM-ja is, hiszen a felhasználói programok tárolása és a felhasználói tár helyettesítése is a feladata.

A hibakereső modulok egy második sínre, az emulálandó mikroprocesszorsínre is csatlakoznak. Ez a sín, ill. ennek a sínnek a jelei szolgálnak a felhasználói rendszer csatlakozási felületeként. A hibakereső modulok feladata éppen ennek a sínnek a kiszolgálása, ill. figyelése. Az áramköri emulátormodul általában egy ugyanolyan típusú mikroprocesszort tartalmaz, mint amelyiknek az emulálására használatos (4.33. ábra). Ennek a mikroprocesszornak az adat-, cím- és vezérlőjelei megfelelő erősítés, illesztés (esetleg logikai kapuzás) után alkotják a mikroprocesszorsín jeleit. Valójában tehát ez a mikroprocesszor emulálja a felhasználói rendszer mikroprocesszorát annak működése közben. A mikroprocesszorsínt általában egy viszonylag hosszú kábel (1–2 m) segítségével juttatjuk el a felhasználói rendszer mikroprocesszorának a foglalataig. A hosszú kábel miatt szükséges a kábel csatlakozóval ellátott „külső" végének a közelében is a megfelelő jellezárás, ill. jelillesztés. Ezt a kábelre épített jelillesztő-erősítő dobozba szerelt elektronikus kapcsolókörökkel lehet megoldani. Az áramköri emulátor a kezelőtől a rendszersín közvetítésével és a mikroszámítógép segítségével vezérlési parancsokat (adatok megadása, indítás stb.) kap. Ily módon biztosítható az emulátor részéről az ellenőrzött programfuttatás megvalósítása. A mikroprocesszoros központi egység helyettesítésén kívül az emulátor egység feladata még a felhasználói tárnak, ill. a be-, ill. kimeneti egységeknek a fejlesztői rendszer RAM-jával, illetve a be-, ill. kimeneti emulátorral való helyettesítéséhez a megfelelő vezérlés ellátása. Ezért az emulátor mikroprocesszor-rendszersín illesztőegységet is tartalmaz. A 4.34. ábrán látható, hogy milyen úton megy végbe az adatátvitel a RAM és a felhasználói rendszer között tár emulálása közben. Ez az út: a tár, a rendszersín, az áramköri emulátor rendszersín-illesztőegysége, (esetleg az emulálást végző mikroprocesszor), a „mikroprocesszorsín illesztése–vezérlése" egység, mikroprocesszorsín és a felhasználói rendszer.

A programfejlesztéshez és a programok belövéséhez általában nem szükséges a felhasználói rendszer és a fejlesztői rendszer összekötése. Amennyiben a felhasználói hardver működtetésére (a csatlakozó perifériás egységek vezérlésére, adatok bevitelére és kiadására) nincs szükség, a felhasználói program végrehajtható az integrált fejlesztői rendszeren belül is. Ekkor a programtárolást az adatok megőrzését a RAM látja el, és be-, ill. kimeneti egységként a be-, ill. kimeneti emulátort használjuk, amely-



4.34. ÁBRA. A fejlesztői rendszer tárának és a felhasználói rendszernek a kapcsolata táremulálás közben

hez – korlátozott számban ugyan – perifériákat is csatlakoztathatunk, míg a program értelmezését, az adatok feldolgozását, a számítástechnikai műveleteket az áramköri emulátor egység mikroprocesszora végzi el.

Az ellenőrzött programfuttatás módszerét – mind a hardver tesztek futtatásához, mind pedig a felhasználói program önálló vagy hardverrel együtt végzett próbáinál – a feltételdetektáló modul és a nyomkövető tár (eseménytároló modul) használata teszi teljessé [39].

A feltételdetektáló modul feladatai:

- a felhasználói program futtatásának a végét jelentő (töréspont) jel generálása az áramköri emulátor felé,
- nyomkövetési jelek generálása az eseménytároló modulnak,
- szinkronimpulzus generálása oszcilloszkóp számára.

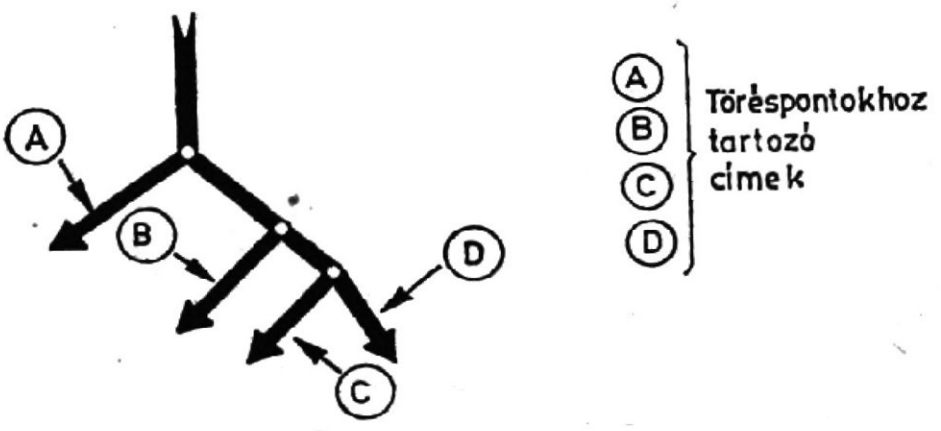
E funkciójelek megjelenését a programbelövést végző kezelő a program indítása előtt különböző feltételekhez kötheti. Ezek a feltételek az emulátor mikroprocesszor adat-, cím- és vezérlőjeleinek az értékére, ill. e jelek logikai kombinációira vonatkozhatnak. Példaként (töréspontra vonatkoztatva) elmondjuk milyen paraméterek szerepelhetnek a feltétel megadásában:

- az emulátor cím-, adat- és vezérlőjeleinek tartalma,
- adatmaszk (nem az összes adatbitre vonatkozó paraméter esetén),
- ciklusszám (programhurkok vizsgálatához),
- címtartomány (alsó és felső határértékekkel megadva),
- késleltetési paraméter (ha a töréspont feltételt követően még valamennyi ideig tovább kívánjuk futtatni a programot) stb.
- Igen gyakori, hogy nemcsak a mikroprocesszor jelei szolgálnak adatokként, hanem a felhasználó kívánsága szerint kiválasztott egyéb logikai jelek értékei (ún. külső hardver bitek) is csatlakoztathatók a fejlesztői segédeszköz bemeneteire, és ezekhez a jelekhez is köthető pl. a leállási feltétel értéke [40].

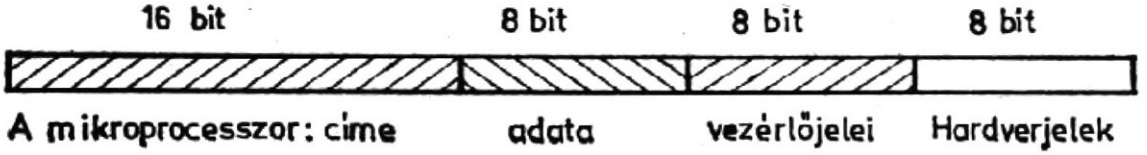
A felhasználói hardver tökéletesebb vizsgálatához ad segítséget az oszcilloszkóp-szinkronjel, amelynek a generálását ugyancsak feltételekhez előírtan végezheti el a feltételdetektáló modul. A nyomkövetési jelek szerepéről az eseménytároló modullal kapcsolatban beszélünk.

Az emulálás közbeni azonos idejű üzemben való programfuttatás megvalósítása az egyik legfontosabb követelmény, amelyet a fejlesztői segédeszközökkel szemben támasztanak. Az azonos idejű üzemben való programvégrehajtás közben gyakran szükséges a programelágazásoknak olyan vizsgálata, amely esetén az elágazás környezetét is azonos idejű működés közben kívánjuk ellenőrizni. Az ilyen vizsgálatokat megkönnyíti az, ha egyidejűleg több feltétel (töréspont) is megadható. A 4.35. ábrán látható ilyen programelágazás, ahol az elágazási ponton még teljes sebességgel futott a program, a több töréspont megadásának köszönhetően azonban az elágazás után bármelyik ágban megállítható, és ellenőrizhető a program végrehajtása. Az ábrán a töréspontfeltételeket az A, B, C és D címeken levő utasítások végrehajtása jelenti.

A programok futásának az ellenőrzése igen hatékonyan végezhető el, ha a futás közben a végrehajtásról adatokat gyűjtöttünk be. Ezeket az adatokat leállás után a kezelő kiértékelheti, s így képet kap a program helyes vagy helytelen voltáról. A begyűjtött adatok alapján az utasítások nyomon követhetők. Azonos idejű nyomkövetés céljára az integrált fejlesztői eszközök hardvermodult tartalmaznak, melynek neve: eseménytár (event storage) vagy nyomkövető tár (trace-memory). A 4.36. ábrán példaként egy lehetséges eseménytár egyik eseményszavát mutatjuk be. A különböző típusú fejlesztői rendszerek különböző hosszúságú eseményeket képesek rögzíteni. Az eseményszavak általában az emulátor mikroprocesszorának a teljes cím-, adat- és vezérlőjeleit tartalmazzák, de néhány típus esetén egy-egy esemény további adatokat is magába foglalhat (pl.: hardverjeleket). Ugyancsak eltérő az eseménytármodul kapacitása. Gyakori a 256 esemény tárolására alkalmas modul, de ennél kisebb vagy nagyobb kapacitású is ismeretes [41]. Nem mindegy, hogy az eseménytárba beírandó adatok „előválogatását" milyen feltételek megadásával írhatjuk elő. Fontos, hogy az eseménytárba írás (előválogatás) kellőképpen szelektív legyen, azaz bonyolult feltétellel lehessen előírni. A jól előszelektált feltételek beírásakor nem csordul túl gyorsan az eseménytár, így nem vesznek el fontos adatok. Másrészt könnyebb a beírt, majd leállás



4.35. ÁBRA. Programelágazások vizsgálata több törésponttal



4.36. ÁBRA. Példa az eseményszó összetételére

után a kezelő által kiolvasható feltételek értékelése is. Néhány fejlesztői rendszer nem engedi meg, hogy az esemény beírási feltétele ugyanolyan bonyolult kifejezés formájában legyen megadható, mint a töréspont (pl.: eseményírás csak a mikroprocesszor vezérlőjeleinek a függvényében adható meg). Más fejlesztői rendszerekben találkozhatunk olyan több feltétel megadását lehetővé tevő feltételdetektáló modullal, amelyik bármelyik feltételhez különböző funkciójeleket rendel (töréspont, nyomkövetési jel, szinkron impulzus) a kezelő kívánsága szerint [42].

A hibakereső program

A hibakeresési munkákat könnyíti meg a szükséges funkciókat tartalmazó, jól felépített hibakereső program. Az általánosan használatos kezelői parancsok és a hibakereső program szolgáltatásai a következők:

- „Konfigurálási" parancsok, amelyekkel az emulálandó tárrészek címei és a be-, ill. kimeneti interfészcímek adhatók meg.
- Tárírás, -olvasás műveletek.
- Tárban tárolt adatblokkok összehasonlítása, egyik helyről a másik helyre átírása.
- Az emulátor-mikroprocesszor belső regiszterek adatainak, vezérlőjeleinek a kijelzése, beírása.
- Feltételek és funkciójelek megadása.
- Eseménytárba beírt adatok kiolvasása.
- Futási idő mérése és kijelzése.
- Felhasználói program indítása.

Ezeknek a parancsoknak a jelentését nagyobb jelentősége miatt bővebben magyarázzuk. Az alábbiakban példaként bemutatunk néhány parancsmegadást [35] (a számok hexadecimális számok):

S M 1002 14 AB 28 7B

írj az 1002 tárcímtől kezdődően a tárba
14-et, AB-t, 28-at, 7B-t.

MV 2008 3008 100

másold át a 2008 címmel kezdődő 100 helyre a 3008 címmel kezdődő 100 adatot.

S A 0F

állítsd az A regiszter értékét 0F-re.

S PC 120B

állítsd a PC (programszámláló) értékét 120B-re.

B MR MW A 4701 D 0A DM 0F

töréspontfeltétel: ha tárolvasás (MR) vagy tárírás (MW) műveletet hajt végre az emulátor a 4701 tárhelyen (A 4701: a címfeltételt jelenti) és az adatok értéke a 4 kis helyi értékű bitben A (D 0A: az adat értéke 0A; DM 0F: az adatmaszk értéke 0F). A felhasználói program futását a feltétel teljesülése felfüggeszti, a program leáll, ha a 4701 tárcímen írás vagy olvasás műveletet hajt végre és az adat értéke ...1010.

```
T  PR
```

nyomkövetési feltétel beviteli műveletekre (PR: Port Read).

```
G  1234
```

a felhasználói program indítása (G: Go) az 1234-es címtől kezdődően.

Példaként bemutatjuk a fejlesztői rendszer kijelzéseit (az első sor a parancsot jelenti, a továbbiakat a rendszer automatikusan kijelzi).

```
D  M  10  16
0010  47  5B  12  4D  AC  17  08
```

tártartalom kijelzése a 0010 címtől a 0016 címig terjedően.

```
CM        2008        3008        100

2197 : 00     3197 : F0
219B : 08     319B : F8
2200 : 01     3200 : 71
2314 : 05     3314 : 15
```

tártartalom összehasonlítása a 2008-tól kezdődő és a 3008-tól kezdődő címekről, (100 byte), a nem egyező értékeket (a példában 4 helyen) kijelzi a rendszer.

```
CPU

PC  SP    A  F  B  C  D  E  H  L  A' F' B' C'... IX   IY
1234 2FFC 08 14 44 7B 8E 90 55 18 40 44 11 30...AB80 1238
```

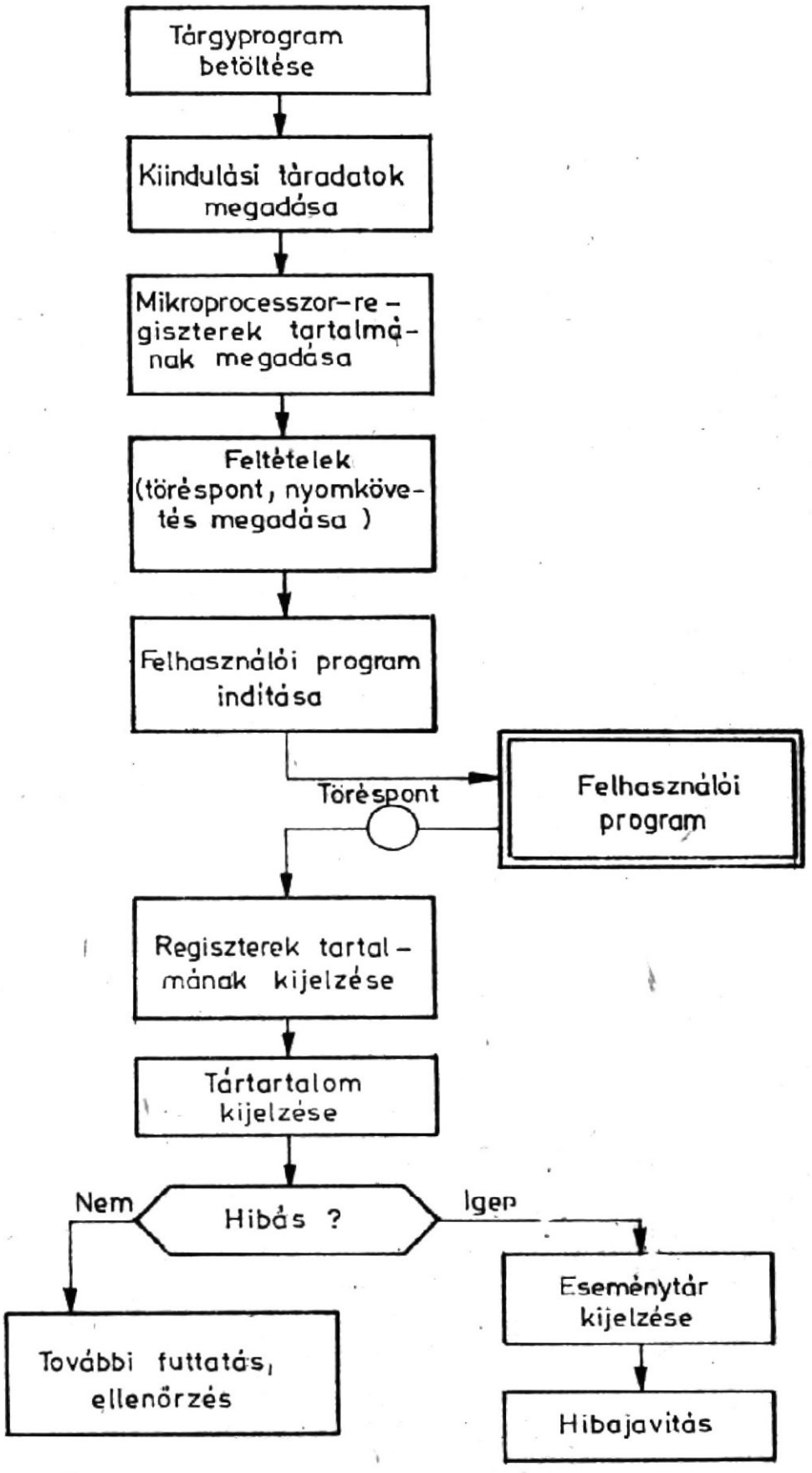
a Z80 mikroprocesszor belső regiszterei tartalmának a kijelzése (a regiszterek felsorolása nem teljes).

```
H  3
44  82  10  15
44  83  04  F0
44  84  03  73
```

eseménytár tartalmából három esemény kijelzése (címnek a magas helyiértékű byte-ja, címnek az alacsony helyiértékű byte-ja, adat, vezérlőjelek).

A 4.37. ábrán a felhasználói program próbájának a menetét láthatjuk. A tárgyprogram tárba töltését a kiindulási adatok megadása (tár, regiszterek tartalma stb.), majd a feltételek előírása követi. Ezután indítható a felhasználói program az adott címtől. A törésponton a felhasználói program leáll. Ezután a tár adatainak, ill. az emulátor-mikroprocesszor belső adatainak a kijelzése következik. Helyes programfutást követően a program ellenőrzése tovább folytatható. Hibás programműködés esetén az eseménytár adatai segíthetik a hiba helyének a felkutatását. A hiba megtalálása, majd a kijavítása után a program próbái folytathatók. A hardver diagnosztikai programokkal való ellenőrzése, vagy a hardver és a programok együttes próbái esetén az eljárás hasonló, csak annyi a különbség, hogy ez esetben a hardver jeleit, működését is figyelemmel kell kísérnünk.

Az emulátor mikroprocesszorának a belső adataihoz való hozzáférést az asse-



4.37. ÁBRA. A felhasználói program ellenőrzésének a folyamata

mulátoros fejlesztési segédeszköznél ismertetett módon szervízrutinokkal oldják meg az integrált fejlesztői rendszerek is. A 4.38. ábrán bemutatott módon közvetlenül a felhasználói program futása előtt az emulátor végrehajtja a mikroprocesszor vezérlőjeleit és regisztereit a kívánt értékre beállító szervizrutint, majd a felhasználói program megállása után (töréspont) rátér a regiszterek tartalmát kiolvasó és a vezérlőjelek állapotát meghatározó szervizrutinra. A kiolvasott értékek tárolásra a tár meghatározott helyére, az ún. regisztertükörbe kerülnek.

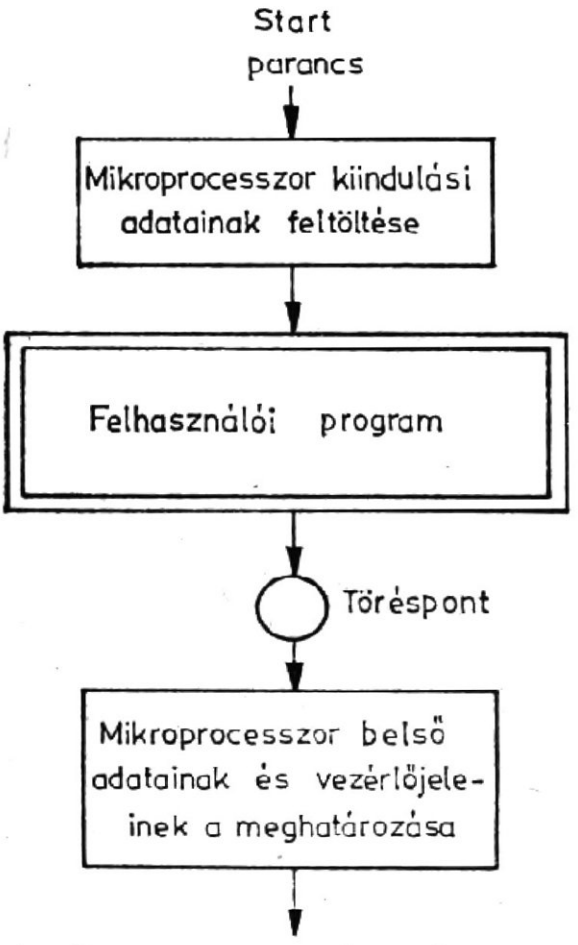
A programok és a hardver-szoftver együttes próbáinak a befejezése után a végleges felhasználói program PROM-ba égetését, ill. a programdokumentáció elkészítését a kisegítő programokkal végezhetjük el.

Az integrált fejlesztői rendszerek használata

A továbbiakban részletesebben tárgyaljuk a fejlesztői segédeszközök által nyújtott különböző szolgáltatásokat. Megmutatjuk, hogy milyen előnyökkel, ill. hátrányokkal jár ezeknek a funkcióknak a megléte vagy hiánya.

Az integrált fejlesztői rendszerek használata nemcsak a fokozatos fejlesztés lehetővé tételével, valamint a hardver-szoftver együttes próbáival segíti a felhasználói rendszer hatékonyabb létrehozását. Igen kényelmes a hibakeresés és a hibajavítás, ha az ellenőrzés közben észlelt programhibák azonnal kijavíthatók, ha kell, a forrásnyelvi program újraszerkesztésével és újrafordításával.

Az operációs rendszer (mágneslemezes operációs rendszer) és a hozzá kapcsolódó programok használatát a kényelmes kezelhetőség megkönnyítheti [43], [44]. Pl.: a programfejlesztési munkák nagy hányadát a szerkesztőprogram használata tölti ki. Ezért nem mindegy, hogy egy-egy újraszerkesztés megtételéhez mennyi időre van



4.38. ÁBRA. A mikroprocesszor regisztereinek a feltöltése, majd kiolvasása a felhasználói programok ellenőrzése során

szükség. A kényelmes használattal kapcsolatosak a vezérlő konzolírógép, a képernyős megjelenítő által nyújtott kezelési jellemzők is. Ezek a fejlesztői rendszer használata alatt – akár programfejlesztésről, akár hibakeresésről beszélünk – a kapcsolattartás eszközei a programozó vagy a mérnök és a fejlesztői rendszer között. Szorosan ide csatlakozik a háttértár használatának a kérdése is. A háttértár (hajlékony lemezes tár), nemcsak a fejlesztői rendszer programjainak a tárolására és gyors elérés használható, hanem a felhasználó által írt forrásprogramok, lefordított tárgyprogramok, s ezek különböző javított változatainak a megőrzését is ellátja (a kidolgozás időtartamára). Használatával a felhasználói programok tárolása, kezelése is egyszerűsödik.

A rövidebb felhasználói programok megírásához az assembler vagy keresztassembler programok használata is elegendő. Az assemblerekkel szemben is követelmény, azonban, hogy univerzálisak legyenek. Ez annyit jelent, hogy lehetővé tegyék:

- A relokálható programírást, ami nagyban megkönnyíti a részenkénti programfejlesztést.
- A makroutasítások használatát. A makroutasításokkal a felhasználó saját „utasításokat" vezethet be, amelyek számára könnyebben kezelhetők.
- A feltételes fordítást. Feltételes fordítással egy-egy programrész többirányú és több helyen való felhasználását könnyíthetjük meg, hiszen a feltételek függvényében az assembler a programrészt kívánság szerint „adaptálja" az adott helyen.

A magas szintű programnyelvek alkalmazása (l. a 10. fejezetet) nagyban meggyorsíthatja a bonyolultabb programok írását, csökkentheti az elkövetett hibák számát. A fejlesztői rendszer szoftverjének feltétlenül lehetővé kell tennie a felhasználói programok készítését különböző programnyelveken (BASIC, PL, PASCAL stb.).

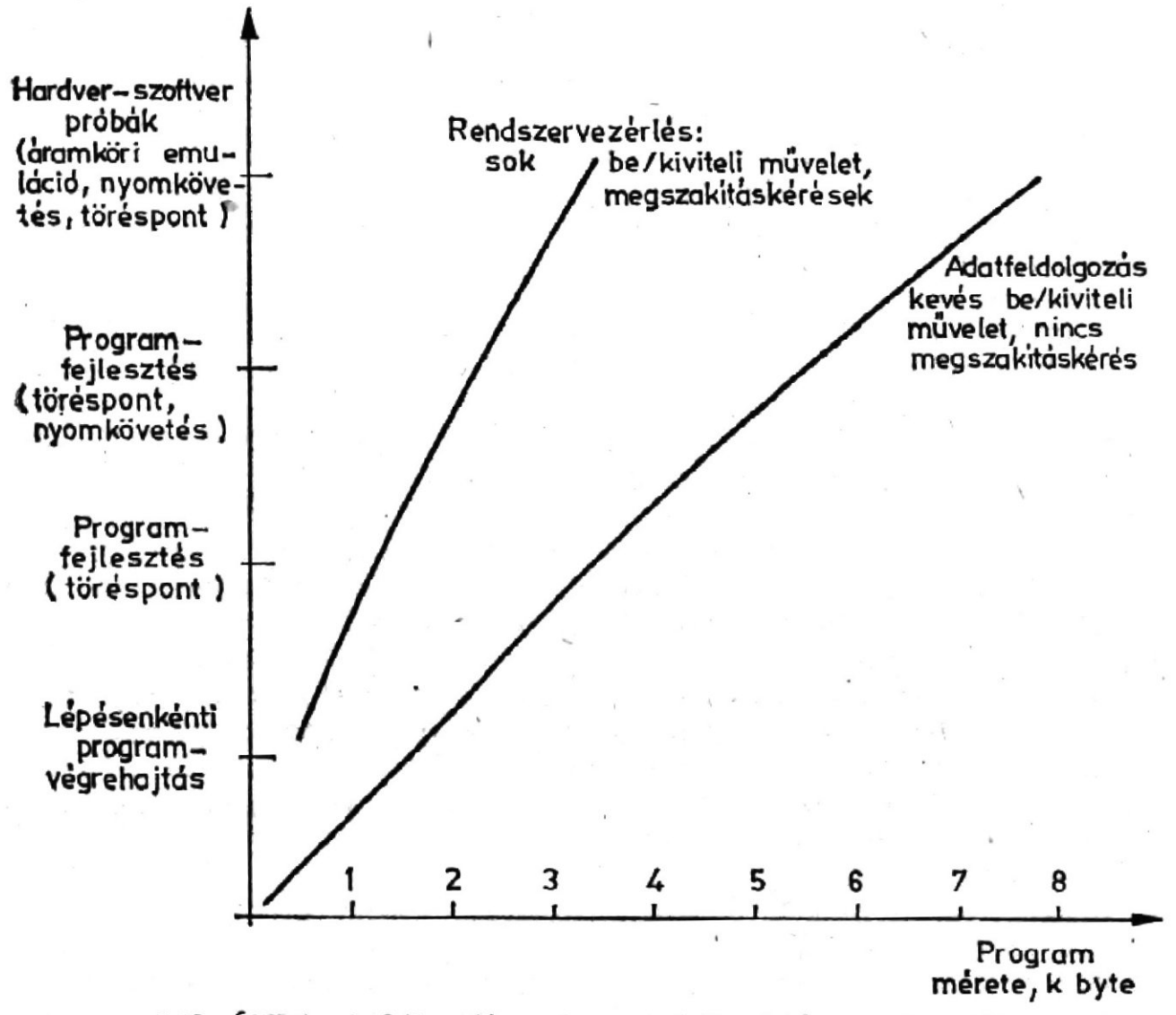
A hardver, szoftver és az együttes hibakeresés munkái alatt az integrált fejlesztői rendszer szolgáltatásainak előnyei:

- A mikroprocesszor lépésenkénti működtetése, programjának lépésenkénti végrehajtása az egyik legalapvetőbb és legszükségesebb hibakeresési eljárás. Ezzel a hardver és a program legalapvetőbb funkcióit, műveleteit figyelheti meg és ellenőrizheti az „élesztést" végző mérnök vagy programozó.
- A töréspont használata a nagyobb programrészek, a sokutasításos műveletek végrehajtásának hatékony eszköze. A sokrétű ellenőrzéshez flexibilis töréspontfeltételek (sokparaméteres logikai kombinációk) megadására és egyidejűleg több töréspontfeltétel használatára is szükség van.
- A programnyomkövetés fontos eszköze az események gyűjtését végző eseménytároló modul. Segítségével (különösen, ha külső jelek fogadására is alkalmas) a logikai állapot- és mikroprocesszor-analizátoroknál ismertetett hibabehatárolási feladatok nagyon jól megvalósíthatók. Igen hasznos tulajdonság lehet a szelektív nyomkövetés, amely az események „előválogatásával" nagyban megkönnyíti az eseménytár tartalmának ember általi kiértékelését.
- A programstatisztikai funkciók közé sorolható a felhasználói program futási idejének a mérése, gépi ciklusainak az összeszámlálása, meghatározott címhez tartozó tár- és be-, ill. kimeneti műveletek összeszámlálása, programhurkok kezdeti és végpontjainak (utasításcímeinek) a megkeresése stb. Ezek a funkciók a töréspont és eseménytárolás mellett jól kiegészíthetik a felhasználói rendszer működéséről szerzett információkat, további kiegészítő tájékoztatást adnak a program végrehajtásáról.

A tényleges működési sebességnek megfelelő működés, a felhasználói programnak a „teljes" sebességű végrehajtása az alkalmazások nagy része esetén igen lényeges követelmény. A mikroprocesszoros egység a felhasználói rendszerek egy részénél igen szoros funkcionális kapcsolatban áll az általa vezérelt külső egységekkel. Ezeknek a külső egységeknek a működése sok esetben szigorúan meghatározott időfüggvény

szerinti (pl.: ipari folyamatok irányítása esetén). Ezekben az esetekben nagyon fontos, hogy a fejlesztői rendszer segítségével még a fejlesztés ideje alatt pontos képet kaphassunk a működésről.

Az integrált fejlesztői rendszerek egyik csoportja az egy mikroprocesszor-típushoz kidolgozott segédeszközök. Ezek csak egyetlen mikroprocesszor-típus áramköri szintű emulációját teszik lehetővé, csak a vele kapcsolatos alkalmazási munkák elvégzésére használatosak. A fejlesztői rendszerek másik csoportját az ún. univerzális fejlesztői rendszerek alkotják. Ilyent mutattunk a 4.31. ábrán is. Ezekben a rendszerekben az emulátormodulok cserélhetők, a velük szoros kapcsolatban levő hibakereső program is kicserélhető. A különböző típusú programok lefordítására keresztassemblerek használhatók. A cserélhető hardver- és szoftvermoduloknak köszönhetően az univerzális rendszerek különböző típusú mikroprocesszorok alkalmazási munkáihoz is jól használhatók. Az univerzális fejlesztői rendszerek többprocesszorosak, egyik processzor (a host-processzor) a rendszer mikroszámítógépében a rendszervezérlést, valamint a rendszerprogramok végrehajtását látja el, míg a másik processzort emulációs célokra használják. Az egy mikroprocesszorhoz kidolgozott fejlesztői rendszerek is lehetnek többprocesszorosak (a feladatmegosztás hasonló a cserélhető emulátoros fejlesztői rendszereknél elmondottakhoz), de ismeretesek egyprocesszoros rendszerek is, ahol az emulátorfunkciókat is, de a rendszerprogramok végrehajtását is ugyanaz a mikroprocesszor végzi el.



4.39. ÁBRA. A fejlesztői rendszer szolgáltatásainak a használata és a felhasználói rendszer bonyolultsága közötti kapcsolat [32]

A 4.39. ábrán az alkalmazási feladatok bonyolultságának és a fejlesztési segédeszközök által nyújtott szolgáltatások igénybevételének a szükségességét vetjük össze. Az ábrán a két grafikon két különböző típusú feladathoz tartozik. A sok be-, ill. kimeneti műveletet végző, megszakításkéréseket tartalmazó programok ellenőrzése nehezebb feladat, mint a túlnyomóan csak adatfeldolgozási műveleteket használó programoké. A két grafikon erre a két feladattípusra vonatkozik. Az abszcissza mentén a felhasználói programok bonyolultságát érzékelő programméret van feltüntetve. A grafikonok azt a határvonalat jelentik, ahol a program bonyolultságának függvényében gazdaságossá válik az adott szolgáltatással rendelkező fejlesztői rendszer beszerzése. Az ábrán feltüntetett adatok az igénybevétel gazdaságosságára vonatkoztatva csak az irányzatot jelölik. Az LSI és a VLSI eszközök, valamint a fejlesztői rendszerek árának változása miatt a grafikonok „meredeksége" állandóan változik. A jövőben a fejlesztői rendszerek használata – árcsökkenési irányzatok következtében – várhatóan egyre kifizetődőbb lesz az egyszerűbb feladatok kidolgozásához is.

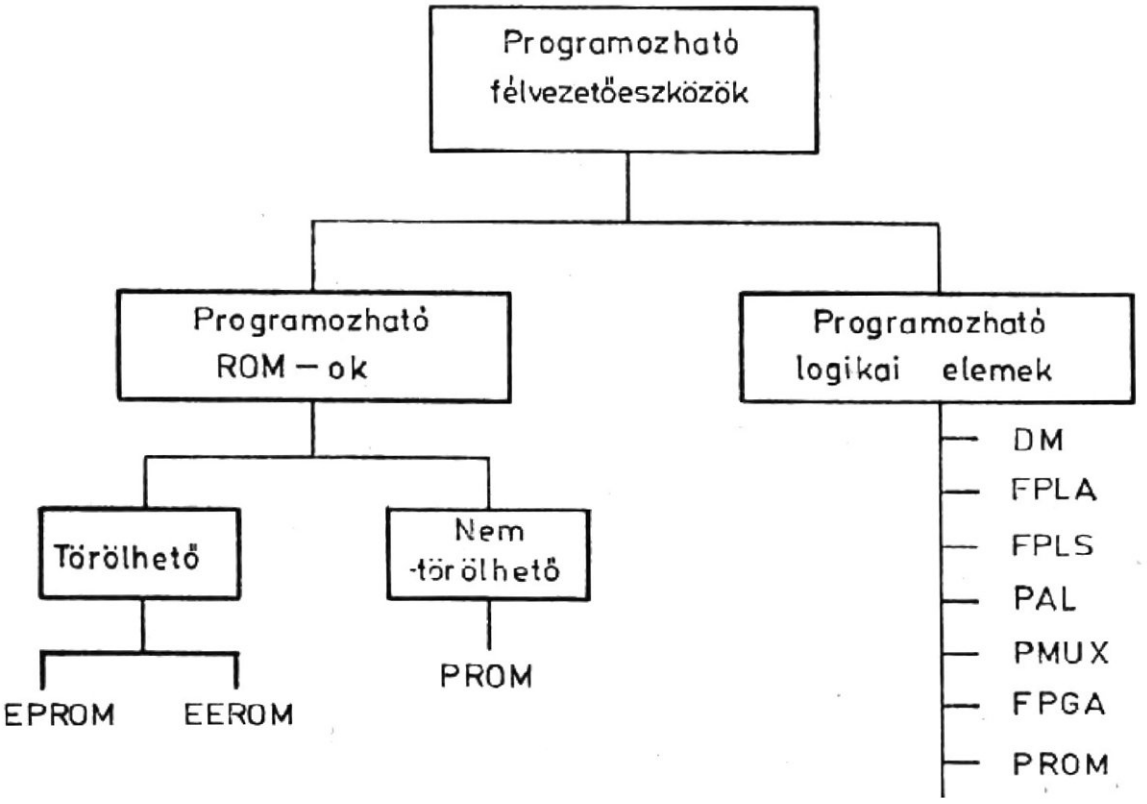
4.9. TÁBLÁZAT. Az integrált fejlesztői segédeszköz szolgáltatásainak a használata [32]

Fejlesztési szakasz	Szerkesztő program (Editor)	Fordító program (Assembler, Compiler)	File kezelés	Töréspont	Nyomkövetés	Áramköri emuláció	Lépésenkénti végrehajtás	PROM–FPLA égetés
Hardver bemérése						+	+	
Programok összeállítása, javítása	+	+	+					
Programok próbája				+	+	+	+	
Hardver-szoftver együttes próbái				+	+	+	+	
Gyártás, bemérés				+	+	+	+	+
Szervizelés				+		+	+	

A 4.9. táblázatban összefoglalva tüntettük fel, hogy az integrált fejlesztői eszközök hardver- és szoftverszolgáltatásai közül melyeket használhatjuk a különböző fejlesztési szakaszokban.

4.8. PROGRAMOZÓBERENDEZÉSEK

A félvezető-technológia fejlődésével egyre nagyobb választékban és egyre jobb jellemzőjű programozható eszközök közül választhatnak a mikroprocesszoros rendszerek tervezői. A 4.40. ábra szerinti két nagy csoport közül az első a ROM-ok kategóriája, amely ismét két részre bontható: törölhető és nem törölhető tárelemekre. A nem törölhető (PROM) elemekben a programozás – információbeírás – során a tárelemen belüli



4.40. ÁBRA. A programozható félvezetőeszközök csoportosítása

kapcsolatokat kell megváltoztatni, összekötéseket megszakítani vagy zárni. Az alapvető technológiák a következők:

- Fémbiztosítékok (általában króm-nikkel vagy titan-tungsten ötvözetek), amelyeket a programozás folyamán – a bit értéktől függően – ki kell „égetni". Ezek kezdeti problémáját, azt ti., hogy a nyitott biztosítékmaradványok később sokszor „összeforrtak", ma már a technológia továbbfejlesztésével megoldották.
- Szilícium „biztosítékok", amelyek polikristályos szilíciumból készülnek. Ezeknek viszonylag nagy (20...30 mA) áramra van szükségük a programozáshoz, nem csupán a „kiégetéshez", hanem a kellően magas hőmérséklet (kb. 1400 °C) létrehozásához, amely a szilíciumot oxidálja, és ezáltal egy jól szigetelő réteget is kialakít.
- Egymással szembekapcsolt diódák formájában megvalósított nagy impedanciájú összeköttetés. Programozás során az egyik dióda ellenőrzött körülmények közötti letörésével a feszültség és a létrehozott hő alumíniumionokat vándoroltat, amelyek rövidrezárást hoznak létre. Ezt a módszert szokták AIM (Avalanche Induced Migration: lavinaletöréssel létrehozott elvándoroltatás) technológiának hívni. Bár a programozási algoritmus elég komplex, és maga a művelet is nagyfokú precizitást igényel (könnyen bekövetkezhet „túlprogramozás", amikor is a második réteg is megsérül) külön előnye ennek a módszernek, hogy minden egyes összeköttetés impedanciája is ellenőrizhető programozás után.
- A törölhető REPROM elemekben a beírás folyamán töltéseket juttatunk a MOS térvezérelt tranzisztor (MOSFET: Metal Oxide Semiconductor Field Effect Transistor) oxid szigetelővel körülvett lebegő vezérlőelektródájára (floating gate), amelyek ott a nagy szigetelési ellenállás miatt rendkívül hosszú ideig gyakorlatilag változatlanul megmaradnak [45].

Az egész elem törlését egy kvarc ablakon keresztül ultraibolya fénnyel végezhetjük (EPROM). A szokásos érték 254 nm hullámhossz és 15 Ws/cm^2 összdózis, amely egy átlagos kvarclámpa 12 mW/cm^2 teljesítményét feltételezve kb. 3 cm távolságról

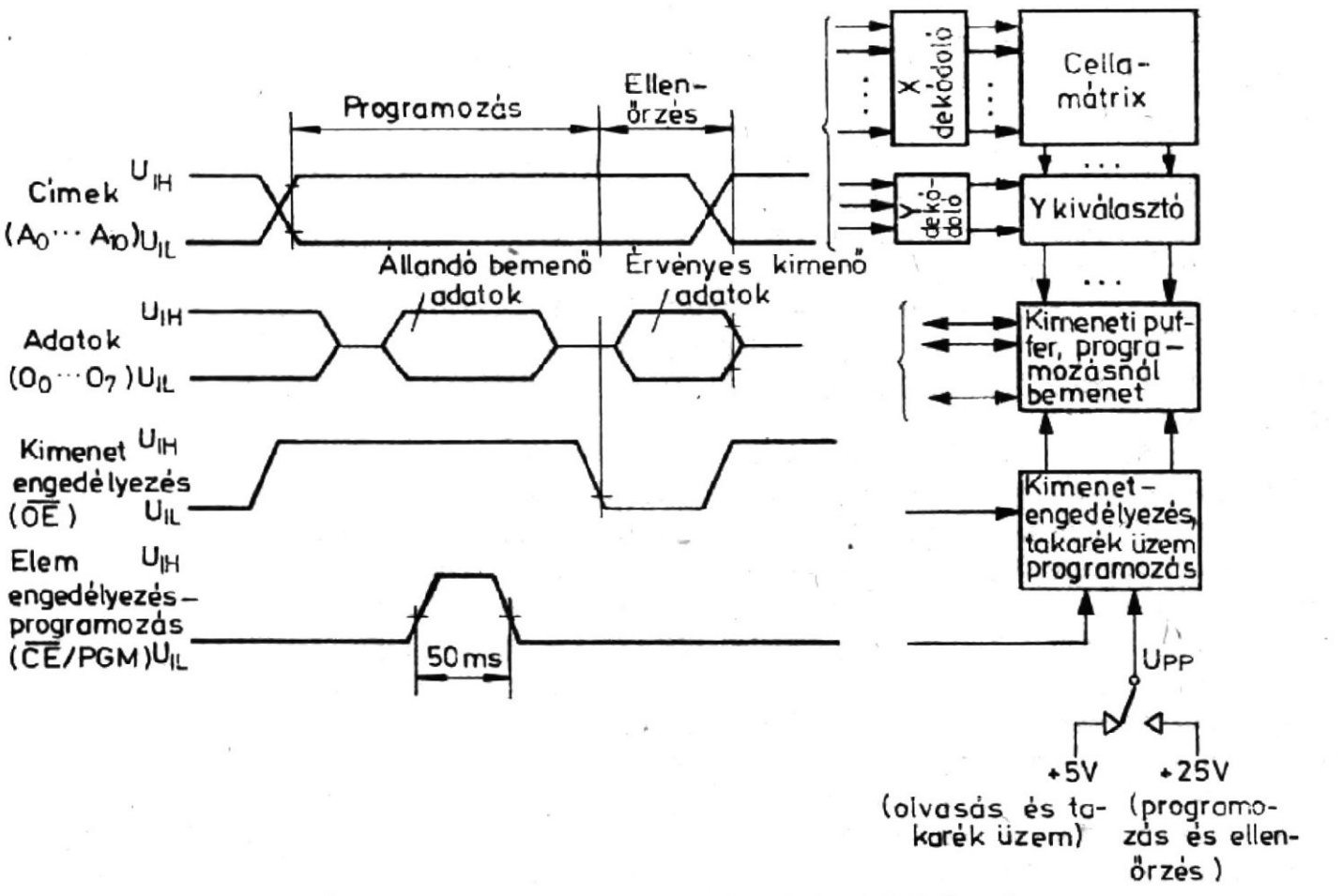
15...20 perc törlési időt eredményez. A másik lehetőség (EEROM) egy elektromos vezérlőelektróda segítségével való törlés- és beírásvezérlés. Ez elvileg lehetővé teszi az áramkörön belüli programozást. Azonban ezen elemekhez jelenleg speciális feszültségekre és elég összetett programozó jelsorozatra van szükség, amely a rendszeren belül gyakorlatilag gazdaságosan nem állítható elő.

A programozható logikai elemek lényegében a PROM-oknál ismertetett eljárásokat alkalmazzák.

A sokféle eljárás és különösen azok gyakorlati megvalósítása nagyon nagyszámú, különböző programozási módszert jelent. Sokszor a különböző gyártók által készített egyébként kompatibilis (plug-compatible: egymás helyére dugható elemek, vagy read compatible: olvasás szempontjából egyenértékű elemek) PROM-ok is eltérő módon programozhatók. A MOS EPROM-ok, amelyek típusszáma megegyező, általában programozás szempontjából is kompatibilisek. A MOS EPROM-ok szavanként programozhatók, a bipoláris PROM-ok ezzel szemben – függetlenül szervezésüktől – bitenként. Újabban a PROM elemek gyártói saját gyártmánycsaládjaikon belül, házilag szabványosított vagy más szóval általános programozási eljárásokat alkalmaznak.

A sokféle programozási technika gyakorlati megvalósítására a programozó készülékek gyártói olyan többé-kevésbé univerzális (tehát sokféle elem programozására alkalmas) berendezéseket dolgoztak ki, amelyek cserélhető modulokkal oldják meg a különböző típusú alkatelemek programozását.

A programozás szabványosodása mellett másik jellemző irányzat a programozási technikák egyszerűsödése. Erre példaként a 4.41. ábrán bemutatjuk egy (4 kbyte kapacitású) EPROM elem [46] alapvető felépítését és a programozáshoz szükséges jelek idődiagramját. Programozáskor a cím kiválasztása az olvasáshoz hasonlóan történik. A programozás üzemmód az U_{pp} pontra adott +25 V feszültséggel és a kimenetenge-



4.41. ÁBRA. EPROM elem programozásának idődiagramja

délyezés (OE) felső logikai jelszintre való kapcsolásával állítható elő. Ekkor a kiválasztott címnek megfelelő kívánt adatot a kimeneti pontokra kell adni, amelyek ekkor bemenetként működnek. Az adat- és a címvezetékek előírt idejű stabilizálódása után egy byte, az engedélyezés/programozás (CE/PGM) bemenetre adott, egyetlen adott hosszúságú (50 ms) – TTL aktív felső logikai szintnek megfelelő – impulzussal programozható. Így lehetőség van egymással párhuzamosan megcímzett elemeket különböző adatoknak megfelelően feltölteni csak arra adva programozóimpulzust, amelyet éppen programozni akarunk.

A beprogramozott tartalom azonnal ellenőrizhető is a cím módosítása előtt (U_{pp} megváltoztatása nélkül), az OE jel segítségével engedélyezve a beírt tartalom megjelenését a kimeneti pufferen.

A különböző elemek programozási módja a gyártók alkatrész-katalógusaiban ma már világosan és egyértelműen le van írva. Az FPLA és FPLS elemek programozása – a felépítés viszonylagos bonyolultsága miatt – elég összetett feladat [47], [48].

Ha programozóberendezést vásárolunk, vagy magunk akarunk készíteni ilyet, alapvetően három – jól elhatárolható – feladat és így progamozási környezet igényei szerint kell meghatároznunk a programozó specifikációs jellemzőit. Ezek: a tervezés, a sorozatgyártás és a szerviz.

A tervezéshez és a prototípuskidolgozáshoz sokszor elég egy egyszerű manuális (kézi) programozást biztosító berendezés. Ez akkor igaz, ha nem sokszor és nem hosszú programokat akarunk bevinni, amikor nem érdemes egy számítógéppel segített rendszerhez szükséges szoftver írást „befektetni". Természetesen az automatikus rendszerekben is szükség van kiegészítő funkcióként kézi programozásra. „Nyelvként" manuális programozás esetén hexadecimális kódokat érdemes használni és „segédeszközként" az egyes címekhez tartozó tartalmakat egy jól áttekinthető táblázatba írni. Egy hexadecimális billentyűzettel lehet programozni (címet és tartalmat megadni), és egy ugyancsak hexadecimális képernyős megjelenítő biztosítja a beírt tartalom ellenőrzését. Félautomatikus rendszerek azonos környezetben egy második foglalatba dugaszolható mester (master) PROM segítségével másolást és összehasonlítást (ill. ellenőrzést) is lehetővé tesznek.

Automatikus rendszerekkel a programozást és a kapcsolódó kiegészítő feladatokat (pl. dokumentálás) számítógéppel lehet megoldani. A programozó perifériaként kapcsolódik egy mikroprocesszoros fejlesztői rendszerhez, vagy egy (általában mikro-, mini- vagy kis-) számítógéphez, esetleg – modem segítségével telefonvonalon keresztül – egy nagyszámítógéphez vagy hálózathoz. Más esetekben a programozó a számítástechnikai rendszer által előállított programozási információt valamilyen közbülső adathordozó (pl. lyukszalag, hajlékony lemez) közbeiktatásával kapja meg (közvetett módszer).

Az automatikus programozásban használt kódokra mutatunk be példákat a 4.10. táblázatban. A korszerű berendezések [49] mikroprocesszor vezérléssel működnek, ami lehetővé teszi egyetlen alaprendszerrel széles körű igényeket kielégítő programozócsalád megvalósítását. Ezen belül a módosulatok a specifikációra (különböző speciális opciók megvalósítása), a környezetre (papírszalagos periféria-interfész, fejlesztői rendszer vagy számítógép-csatlakoztatás stb.) és természetesen újabb elemek programozására (a cserélhető modulkészlet bővítésével) vonatkoznak. Egy ilyen programozóberendezés [49] a kézi programozás alapszolgáltatásain túlmenően a következőkben ismertetett funkciókat valósíthatja meg.

A 4.10. táblázatban látható valamelyik kódban megadott cím és azt követő adat bevitele egy puffer-(RAM)-tárba. Ez az egyszerű adatkezeléseken (beszúrás törlés, eltolás, változtatás stb.) túlmenően egész címtartományoknak megfelelő, mezőrészek kicserélését is lehetővé teszi, vagy pl. olyan feladatok megoldását, mint

4.10. TÁBLÁZAT. Automatikus programozóberendezések kódformátuma

Formátum	Érvényes karakterek	8-bites példa	A kezelt karakterszám
Bináris	256 bináris kombináció	1001 1010	1 (párhuzamos átvitel)
ASCII bináris	BNPF, BHLF, B10F	BPNNPPNPNF	10
ASCII oktális	0, 1, 2, 3, 4, 5, 6, 7	232	3
ASCII hexadecimális	0–9, A, B, C, D, E, F	9A	2

kisebb kapacitású PROM-ok tartalmának adott sorrendbe való beírása egy nagyobb kapacitású PROM-ba. Így a másolás a normál (MASTER→COPY: a mester PROM-ból a másolt PROM-ba) üzemmód mellett megvalósítható két további formában is (MASTER→BUFFER: a mester PROM-ból a pufferbe és BUFFER→COPY: a pufferből a másolt PROM-ba).

Az ellenőrzés a másoláshoz hasonlóan vonatkozhat a MASTER-re vagy BUFFER-re, mint referenciára. Eltérés esetén a cím és a helyes és hibás (ellenőrzésnél talált) tartalom kiírása (ill. kijelzése) is kívánatos.

A programozás során sokszor szükség van csak adott résztartományok bevitelére, ill. ellenőrzésére. Ezt a címtartomány megfelelő beállításával (Address Field Manipulation: címmezőkezelés) lehet elvégezni. Szükséges a programozás megkezdése előtt az egész tárolóelem ellenőrzése, hogy az alaphelyzetben – típustól függően csupa nulla vagy csupa egyes – van-e. Ezt részenként való programozás esetén az adott tartományokra is el kell végezni.

A PROM-nak az alkalmazói rendszeren belüli későbbi ellenőrzését sokszor ellenőrző összeg alkalmazásával is elősegítik. Ez szokás szerint nem más, mint a teljes tárolóelemben levő egyesek száma, amelyet egy erre a célra fenntartott rekeszbe írnak be. Az ilyen lehetőséggel ellátott programozóberendezésnek alkalmasnak kell lennie az ellenőrző összeg létrehozására (kiszámítás és beírás) és ellenőrzésére (ismételt kiszámítás és összevetés az előzőleg beírt értékkel).

A kiírás (LIST) üzemmódban a megadott címtartomány és a hozzá tartozó (PROM vagy puffertár) tartalom a rendelkezésre álló, kijelölt listázó periférián (írógép, sornyomtató, képernyős megjelenítő, lyukszalag, hajlékony lemez stb.) lista formájában kiíratásra kerül.

A sorozatgyártás körülményei között csupán három egyszerű műveletet: másolást, ellenőrzést és üresség vizsgálatot kívánunk végezni, de ezeket nagy tömegben és lehetőleg gyorsabban. Ha nem indokolt a nagyon költséges automatikus elemadagolást végző berendezések megvásárlása, csoportos (gang) programozás megvalósításával nagymértékben meggyorsíthatjuk a programozás folyamatát. Ez több elem párhuzamos kezelésével érhető el. Ha erre a célra két foglalat csoportot valósítunk meg, míg az egyikben folyik a programozás, a másikban kicserélhetjük az elemeket. Az alkalmazott foglalatok minden programozó berendezésben a könnyű, sérülésmentes tárolóelem be- és kivitelét végző, erre a célra gyártott olyan tár foglalatok, amelyekben a rögzítő rugóerőket egy kar segítségével oldani lehet (zero-insertion: nulla beillesztési erő).

A szerviz során a programozóberendezések a felhasználó telephelyén való módosítást, új változatok beszerelését és ezen kívül ellenőrzést és hibaelhatárolást kell, hogy lehetővé tegyenek. A hordozható berendezések, amelyeket ezen kívánalmak szerint alakítanak ki, kiegészíthetők a hibadiagnózist elősegítő egyszerű PROM szimulátorokkal is.

4.9. EGYÉB SEGÉDESZKÖZÖK

E szakaszban nem a fejlesztés közben, a tervezés, programírás és rendszerépítés során elkövetett hibák behatárolásának az eszközeivel és módszereivel foglalkozunk, hanem azokkal a segédeszközökkel és eljárásokkal, amelyek a meglévő és már működő rendszerek karbantartási munkáit segítik, meghibásodásukkor a hiba gyors megtalálását és kijavítását segítik. Az ilyen eszközökkel szemben más követelményeket támasztunk. Az installált berendezések működése közben észlelt hibák természete más, mint a tervezési hibáké. Eredetüket az esetek nagy részében könnyebb megtalálni és könnyebb megszüntetni a hibaforrást is.

Az egyik legalapvetőbb követelmény, hogy könnyű, kisméretű – tehát könnyen szállítható – legyen az ellenőrző berendezés. Alkalmazásukat tekintve általában követelmény, hogy alkalmasak legyenek a számítógépes (mikroprocesszoros) rendszereken belüli legkülönbözőbb mérések, ellenőrzések és beállítások elvégzésére. Ezek közül néhány példát felsorolunk: elektromechanikus eszközök beállítása, perifériák ellenőrzése, adatátviteli vonalak (soros és párhuzamos is) tesztelése stb.

Általában a számítógépes rendszerek szervizelése költséges. Ennek oka a drága gépidőn kívül az is, hogy gyakran magasan képzett szakemberek végzik ezt a munkát. Ezért fontos követelmény az ilyen segédberendezések esetén a kényelmes és hatékony hibakeresési lehetőség mellett a könnyen és gyorsan elsajátítható kezelhetőség is.

Összegezve a digitális rendszerek szervizelését végző szakemberek tapasztalatait, megvizsgálva azokat a módszereket és eszközöket, amelyekre legnagyobb szükség van, a következő következtetésekre juthatunk. Szervizelés során leggyakrabban a következő ellenőrzésekre van szükség:

- az ellenőrizendő rendszer paramétereinek mérése,
- a funkcionális jelek meglétének a vizsgálata,
- szinkronizálások, szinkron együttfutások ellenőrzése,
- a hibás modul, egység, áramköri kártya vagy alkatrész behatárolása.

Példaként ismertetünk egy digitális ellenőrző műszert (tesztert) [50]. Ez a kisméretű készülék három nagy impedanciájú jelbemenetével, és a feszültség- és ellenállásmérők szokásos mérőbemeneteivel, valamint egy hőmérsékletszondájával 22-féle mérést képes elvégezni. Kezelése rendkívül egyszerű, tulajdonképpen a csatlakoztatásokon, jelszintbeállításokon és jelleolvasáson (valamint az időnkénti egyszerű „önkalibrálás" elvégzésén) kívül semmi egyebet nem kell tenni.

A 22 érték amit mérni lehet az alábbi:

- az egyenfeszültség,
- a váltakozófeszültség,
- az ellenállásérték,
- a negatív feszültségcsúcs (A),
- a pozitív feszültségcsúcs (A),
- az impulzuskitöltési tényező (A),
- a frekvencia (A),
- a periódusidő (A),
- az impulzusszélesség (A),

■ a tranziens időtartam (A),
■ az impulzusszámosság (A),
■ a két frekvencia arányának (A,C),
■ a két jel közötti harmadik jelimpulzusok számának (A a B és a C jelek között),
■ a két jel közötti időtartamnak (B,C),
■ a két jel közötti harmadik jeltranziensek számának (A a B és a C jelek között),
■ a két jel koincidenciája időtartamának (B,C) a mérése,
■ a jel alacsony (logikai *0*) küszöbszintjének,
■ a jel magas (logikai *1*) küszöbszintjének,
■ a B jel küszöbszintjének,
■ a C jel küszöbszintjének

a beállítása,

■ a hálózati feszültség és
■ a hőmérséklet

mérése. A zárójelekben elhelyezett betűk a három nagy impedanciájú bemenetet (A, B és C) jelölik. Közülük az A bemeneten két küszöbjelszint is beállítható, míg a B és a C bemenetnek csak egy küszöbjelszintje van.

A mérési és funkciómódok váltása egyetlen 22 állású körkapcsolóval végezhető el. A teszter a mérési eredmény megjelenítését öt decimális számjegy, valamint az előjel kijelzésére alkalmas számjegykijelzővel, továbbá az A bemenethez tartozó három fénykibocsátó diódával (logikai *0, 1* és közbülső jelszint) és a B, C bemenetekhez tartozó két-két fénykibocsátó diódával oldja meg. A mérési tartomány (μs, ms, kHz, V stb.) kijelzését ugyancsak fénykibocsátó diódák végzik.

A felsorolt funkciók alapján látható, hogy igen sok mérési és jelparaméter összevetésére nyílik lehetőség. Az olyan egyszerű méréseken kívül, mint az egyen- és váltakozófeszültség vagy hőmérsékletmérés, lehetőséget nyújt a készülék egy sor karakterisztikus jelparaméter mérésére is.

A közvetett jellemzők mérésével és összevetésével a mikroprocesszoros készülék összetett funkciói is ellenőrizhetők. Példaként megemlíthetjük, hogy soros adatátviteli vonalon egy-egy ismert kód átvitelének helyességét jellemzik: a logikai szintek értékei, a bitváltások száma, egy-egy logikai szint meglétének időtartama (impulzusszélesség stb.). Hasonló módon egy-egy ismert áramköri rész működése közben (pl.: tesztprogram futtatásakor) számos olyan jelparaméter határozható meg, amelynek a mérésével az adott áramköri rész helyes vagy helytelen működésére, esetleg a hiba okára is lehet következtetni.

IRODALOM

[1] *Kummel, L.–Schultz, G.:* Advances in microcomputer development systems. Computer, Feb. 1977. 13–18.
[2] The IC troubleshooters. Hewlett-Packard, Sep. 1976.
[3] *Beckwith, J. F.:* Current tracer: a new way to find low-impedance logic circuit faults. Hewlett–Packard Journal, Dec. 1976. 2–8.
[4] *Quenelle, R. C.:* New logic probe troubleshoots many logic families. Hewlett–Packard Journal, Dec. 1976. 9–11.
[5] *Bronson, B.–Chan, A. Y.:* A multifunction, multifamily logic pulser. Hewlett–Packard Journal, Dec. 1976. 12–15.
[6] Techniques of digital troubleshooting. Application Note 163–1. Hewlett–Packard, 1976.
[7] *Priebe, D.:* Multifamily logic clip shows all pin states simultaneously. Hewlett–Packard Journal, Dec. 1976. 18–20.

[8] Z–80 Microcomputer Board series. Zilog Inc., 1979.
[9] MCZ–1/05 Microcomputer system. Zilog Inc., 1978.
[10] MCZ–1/35 Product specification. Zilog Inc., Oct. 1978.
[11] *Teja, E.:* The single-board-μC alternative expands OEM design horizons. EDN, Oct. 20, 1979. 257–290.
[12] Low cost analog I/0 systems fully compatible with single-board microcomputers. Computer Design, Feb. 1977. 148–149.
[13] *Hirschman, A. D.–Ali, G.–Swan, R.:* Standard modules offer flexible multiprocessor system design. Computer Design, May 1979. 181–189.
[14] *Fisher, E. R.:* Guidelines for selecting a microprocessor prototyping kit. Wescon Professional Program, San Francisco, 1975.
[15] *Csákány A.–dr. Vajda F.:* Mikroszámítógépek. Műszaki Könyvkiadó, Budapest, 1976.
[16] Mostek Z80 microcomputer family. Mostek Corp., 1977.
[17] Mostek microprocessor support. Z80 Software development board (SDB–80E). Mostek Corp., 1977.
[18] *Sapiecha, K.–Ambroziak, K. Kott, K.:* An universal cross-assembler for microcomputer systems. Proceeding of μP'79, OMKDK Technoinform, Budapest, 1979. 353–363.
[19] M 6800 microprocessor application manual. Motorola Inc., 1975.
[20] *Эрени И.:* Методы и средства проектирования цифровых устройств, выполненных на базе микропроцессоров. Изд. МТА СТАКИ, Будапешт, 91/1979. 109—124.
[21] *Runyon, S.:* Focus on logic and μP analyzers. Electronic Design, Feb. 1977. 40–50.
[22] *Pin, K.:* Logic analyzers know what you need. Electronic Product Magazine, June 1977. 36–45.
[23] *Marshall, M.:* What to look for in logic timing analyzers. Electronics, March 29, 1979. 109–114.
[24] *Down, R.:* Understanding logic analizers. Computer Design, June 1977. 188–191.
[25] *Lorentzen, R.:* Troubleshooting microprocessors with a logic analyzer system. Computer Design, March 1979. 160–164.
[26] Lower your development costs with this revolutionary window to digital systems. Hewlett–Packard, 1976.
[27] *Haag, G. A.:* A logic state analyzer for evaluating complex state flow. Hewlett–Packard Journal, Feb. 1978. 2–10.
[28] *Smith, J. H.:* A logic state analyzer for microprocessor systems. Hewlett–Packard Journal, Jan. 1977. 2–11.
[29] *Saponas, T. A.:* Firmware for a microprocessor analyzer. Hewlett–Packard Journal, Jan. 1977. 12–15.
[30] *Ogden, D. J.:* Adapting the 1611A logic state analyzer to work with the F8 microprocessor family. Hewlett–Packard Journal, Feb. 1978. 28–32.
[31] *Erdmann, R. E.:* A serial data analyzer for locating faults in decentralized digital systems. Hewlett–Packard Journal, Oct. 1979. 23–28.
[32] *Clark, T.:* Systems design flexibility. Systems International, March 1979. 19–22.
[33] *Erényi I.:* Development tools for designing and debugging microprocessor-based system. MIMI'78 Symposium, Acta Press, Calgary-Anaheim-Zürich, 1978. 58–62.
[34] *Bóna G.–Erényi I.–Vajda F.:* Development and application of a universal system of microprocessor design aids. Proceedings of MIMI'79 Symposium, Acta Press, Calgary–Anaheim–Zürich, 1979. 56–61.
[35] Z80 microprocessor development system. User's manual. Zilog Inc., 1976.
[36] ICE–80 In-circuit emulator reference manual. Intel Corp., 1975.
[37] *Hughes, T. P.–Sawin III, D. H.:* Breakpoint design for debugging microprocessor software. Computer Design, Nov. 1978. 99–107.

[38] *Gladstone, B. E.:* Comparing microcomputer development system capabilities. Computer Design. Feb. 1979. 83–90.
[39] *Francis, R.–Teitzel, R.:* Realtime prototype analysis as a microprocessor design aid. Computer Design, Dec. 1978. 65–73.
[40] *Dum, S.–Philpott, D.:* In circuit emulation aids microprocessor system design. Electronic Engineering, Nov. 1978. 87–90.
[41] *Schindler, M.:* Can one μC development system be 200 times more powerful than another? Shop and see. Electronic Design, Dec. 6, 1978. 60–67.
[42] *Bóna G.–Erényi I.:* Modular integrated microprocessor development aid. Proceedings of μP'79, OMKDK Technoinform, Budapest, 1979. 285–289.
[43] *Rosenfeld, P. R.:* Next-generation development systems are costing less and doing more. Electronics, March 16, 1978. 113–117.
[44] Z80 RIO operating system. User's manual. Zilog Inc., 1977.
[45] *Frohman, K.–Bentchkowsky, D.:* FAMOS read-only memory. IEEE Trans. on Solid State Circuits, Oct. 1971. 36–41.
[46] Memory data book and designers guide. Mostek Corp., March 1977.
[47] *Sieves, B.:* Intersil's „AIM" programmable element. Intersil, 1978.
[48] Signetics integrated cicuits. Bipolar and MOS memories. Signetics, Philips, 1978.
[49] PROM user's guide. Prolog Corp., Monterey, California, Jan. 1979.
[50] 851 Digital Tester Application literature. Tektronix Inc., 1978.
[51] *Головкин, Б. А.:* Параллельная обработка информации. Программирование, вычислительные методы, вычислительные системы. Известия АН СССР, Техническая кибернетика, март—апрель, 1979. 116—151.
[52] *Александрова, З. И.—Александров, А. Г.:* Проектирование систем на базе микропроцессорный фамилии 600. Proceedings of μP'79, OMKDK Technoinform, Budapest, 1979. 105–114.
[53] *Бобко, В. Д.—Бредихин, С. В.—Головин, В. Ф.—Золотухин, Ю. Н.—Мелешихин, В. А.—Песлях, П. М.—Ян, А. П.:* Распределенная система для разработки и отладки аппаратных и программных средств микропроцессорных устройств. Proceedings of μP'79, OMKDK Technoinform, Budapest, 1979. 311–316.
[54] *Гусев, О. З.—Катасонова, Л. И.—Кузнецов, В. В.—Прохожев, В. В.—Смородинов А. Н.—Томашевская, Л. Ф.:* Анализатор логических состояний на базе микропроцессора. Proceedings of μP'79, OMKDK Technoinform, Budapest, 1979. 711–716.
[55] *Сикиевич, Т.:* Микропроцессорная система вспомогательных средств для проектирования и наладки микропроцессорных систем. Proceedings of μP'79, OMKDK Technoinform, Budapest, 1979. 317–326.

5. ELEMEK, EGYSÉGEK, RENDSZEREK ELLENŐRZÉSE

A mikroprocesszorok és a kapcsolódó nagy integráltságú elemek felhasználásával készülő berendezések és rendszerek előállításának talán legfontosabb és ugyanakkor legkevésbé megoldott problémája a gyártott termékek ellenőrzése. Az ellenőrzés maga ugyanakkor nagyon sok általános problémát is magába foglal, amelyek nem a mikroprocesszor és az LSI elemek felhasználásával kapcsolatosak. Ezeket itt természetesen csak nagyon felületesen érintjük, és a feladat összetettsége miatt egyébként is csak a leglényegesebb kérdésekre térünk ki.

Maga az ellenőrzés nem tekinthető függetlenül egy adott termék előállításának folyamatától. Bonyolult elektronikus berendezések esetén pedig egyértelműen megállapíthatjuk, hogy az ellenőrzés feltételeinek megteremtése, a hibaelhatárolások módjainak kidolgozása a tervezési fázis alapvető feladata. Más szóval a gyártmánytervezés során – annak részeként – meg kell tervezni a gyártás közbeni és végellenőrzés egész folyamatát is, figyelembe véve annak kívánalmait a gyártott eszköz jellemzőinek, felépítésének kialakításában.

Anélkül, hogy a gyártási ellenőrzési hierarchia részleteibe mennénk, mint alapelvet leszögezhetjük, hogy az alkatrészek ellenőrzésétől elindulva egészen a késztermék végellenőrzéséig ugyanannak a hibának a megtalálása több nagyságrenddel költségesebb lehet, minél későbbi fázisban fedezzük fel. Természetesen általában az sem gazdaságos, ha minden részművelet után az összes hiba feltárására törekszünk. Különösen igaz ez az LSI bonyolultságú áramköri elemek és ezeket tartalmazó rendszerek esetén, ahol – majd látni fogjuk – az összetettség miatti vizsgálati idők elméleti és gyakorlati határokat fognak szabni. Így az ellenőrzés „hatásfokát" (az esetleges hiba megtalálásának valószínűségét) mindig ésszerű gazdasági megfontolások fogják megszabni. Az ellenőrzés után az egyes fázisokban „megmaradó" hibákat azután a működés közbeni meghibásodás elhárításához hasonló módon – és annak eszközeit is felhasználva – végezhetjük el.

A vizsgálatok célját tekintve a következő három kategóriát különböztethetjük meg [1] [2] [3]:

- Funkcionális vizsgálat, amely az adott elem (modul, rendszer) helyes működésének ellenőrzését jelenti, pl. célszerűen az alkalmazás valóságos körülményei között. Az ilyen ellenőrzés alapvető célja a minősítés, amelynek leggyakoribb esete a működik/nem működik (go/no go: megy/nem megy) vizsgálat.
- Parametrikus vizsgálat, amely a jellemzők mérését jelenti. Megjegyezzük, hogy sok esetben nem a paraméterek pontos értékének, hanem valamilyen határok közötti voltának a meghatározása a cél.
- Hibafelderítő (diagnosztikai) vizsgálat, amelynek során nemcsak észleljük a hibás működést, hanem valamilyen szintű (pl. funkcionális egység, áramköri kártya, alkatrész) hibabehatárolást is végzünk [4].

A hibatípusok is rendkívül sokfélék lehetnek. Reprodukálhatóság szempontjából megkülönböztethetünk állandó hibákat (pl. egy nem szükséges rövidzár) és véletlenszerű, időszakos hibákat (pl. egy becsapódó elemi részecske hatására megváltozik a

félvezetős tár tartalma). Az utóbbi típusú hibákat csak nagyon körülményesen tudjuk reprodukálni, ami nagyon megnehezíti a diagnózis kialakítását is. Beszélhetünk statikus és dinamikus hibákról. Utóbbiak a működési sebesség függvényei. Hasonló az eset, amikor valamilyen másodlagos paraméter (pl. hőmérséklet) bizonyos értékei mellett jelentkezik a hiba (parametrikus hiba). Mintaérzékenységnek nevezzük azt a jelenséget, amikor egy elem bizonyos részét környezete valamilyen állapota bonyolult módon zavarja, bár nem áll vele szándékos, közvetlen logikai kapcsolatban. Ez a típusú hiba az LSI áramkörök (tárak, mikroprocesszorok) esetén sokszor előfordul, és csak bonyolult, hosszadalmas vizsgálattal határozható meg. Előfordulása szerint megkülönböztethetünk egyedi és szisztematikus hibákat. Utóbbiak a gyártási sorozat minden egyes darabjánál vagy legalábbis azok többségénél előfordulnak.

Ezenkívül beszélhetünk tervezési, konstrukciós, gyártási stb. hibákról is. Az elnevezések a hiba forrására utalnak. A logikai áramkörök hibáinak modellezésekor két hibafajtát szokás megkülönböztetni: fizikai hibák (defect), amelyek a helyes megvalósítástól való eltéréseket (rövidzár, szakadás, tápfeszültséghiány, helytelen összeköttetés stb.); logikai hibák (fault), amelyek pl. valamely elem meghibásodása következtében a helyes logikai értéktől való eltéréseket (pl. egy elem bemenetének leválása vagy egy vonal logikai értékének állandó értéken tartása) jelentik.

A konkrét ellenőrzés elvégzésére – a gyakorlatilag szinte sohasem alkalmazható kézi megoldásokat nem tekintve – vagy valamilyen külső automatikus – számítógép vagy mikroprocesszor vezérelt – mérő ATE (Automatic Test Equipment: automatikus ellenőrző berendezés) rendszert használnak, vagy az intelligens eszköz (pl. mikroszámítógép) képes részt venni saját tesztelésében. Ez utóbbi megoldást önellenőrzésnek nevezik.

Az ún. vezetett hibafelderítés esetén a hibafelderítésre különféle szimulációs rendszereket és hozzájuk tartozó vizsgálóberendezéseket használnak [5]. Ez esetben a hibakeresés útját a berendezés jelöli ki és a gépkezelő feladata csupán az, hogy a mérőfejet a kijelölt áramköri csomópontra helyezi. Az előzőhöz hasonló, de azért elvileg eltérő az ún. visszafelé nyomozás módszere. Az eljárás lényege, hogy minden belső csomóponton ismert az egyes vizsgáló jelsorozatokhoz tartozó helyes válasz. A hibás választ jelző kimeneti pontból visszafelé haladva, a kapott értékeket összehasonlítjuk a csomóponti állapotlistában levő helyes értékkel. Ha megtaláljuk az első jó értéket, akkor ezt követően keletkezik a hiba. Ez a módszer akkor tér el alapvetően az előzőtől, ha az útkeresés a felhasználó ember feladata [2].

Az elemekhez való csatlakozás szempontjából megkülönböztethetünk tűágyas ellenőrző berendezéseket, ahol a tűágyra fektetett áramköri kártya minden gerjesztési és ellenőrzési pontjára egy-egy érintkező tű nyomódik. Minden tűhöz tartozik egy érzékelőerősítő és egy kimeneti meghajtóerősítő. Ezek és a tűágy költségessé teszik a berendezést, ezzel szemben kiváló diagnosztikai lehetőséget nyújt a nagyszámú megfigyelési és gerjesztési pont következtében, mivel méréskor az egyes áramköri részek logikailag izolálhatók. Egyszerűbb változataiban a kártya csak a csatlakozóján keresztül gerjeszthető és a tűk csak érzékelésre képesek. A legegyszerűbb rendszerekben mind gerjesztésre, mind megfigyelésre csak a kártyacsatlakozót használják. A tervezés során persze lehetőség van a nem használt kártyacsatlakozó-érintkezők felhasználásával az áramkör néhány belső pontjának kivezetésére is.

A mikroszámítógépek önellenőrző programjai többféle felépítésűek lehetnek. A programok alapvető problémája, hogy egy potenciálisan rossz mikroszámítógépnek kell elvégeznie a saját diagnosztizálását. Ennek az elvi és gyakorlati nehézségnek a feloldására három eljárás terjedt el [6] [7].

Az első megoldást úgy képzelhetjük el, mintha egy külső ellenőrző készüléket beépítettünk volna a berendezésünkbe, amely részt vesz annak normál működésében

is. Az ellenőrzés úgy történik, hogy a mikroszámítógép tesztelő része gerjeszti a többi részegységet és ellenőrzi a válaszokat. Az eljárás lényegéhez tartozik, hogy ennek a résznek megbízhatóbbnak kell lennie a többi résznél. Erről a részről így feltételezzük, hogy hibamentes. Az eljárás is erről kapta a nevét, szilárd mag (hard core) eljárásnak nevezik. Ez a feltételezés egyben a megoldás gyenge pontja is, mivel általában nehéz az ellenőrzés elvégzéséhez szükséges „magot" (mikroprogramvezérlő, mikroprogramtár, dekódolók stb.) a többi résznél sokkal megbízhatóbbá tenni.

A második megoldás megvalósításához arra van szükség, hogy a mikroszámítógépet több független, de önálló „intelligens" részre (A, B, C stb.) lehessen osztani. Így az ellenőrzés során az A rész ellenőrzi a B, a C, a D stb. részt. Gyakorlatilag ezt a megoldást csak multiprocesszoros rendszerek esetén (l. a 12. fejezetet) alkalmazhatjuk előnyösen. Még ilyen esetekben is gondot okoz a közös erőforrások (tár, sínrendszerek stb.) olyan kialakítása, hogy az említett függetlenségre vonatkozó feltételezések jogosak legyenek. Az eljárást – az előző eljárás elnevezésének analógiájára – MSC (Multi Soft Core: több „puha" mag) ellenőrzési módszernek nevezik [7] [8].

A harmadik módszer szerint a számítógép egy kis részét a gépkezelőnek manuálisan kell ellenőriznie. Ez a rész a további vizsgálatok során a „szilárd mag" és ellenőrzi a mikroszámítógép egy újabb kis részét. Ha ez hibátlan, akkor az újonnan megvizsgált rész is beépül az ellenőrzést végző részbe, ami újabb részegység vizsgálatát kezdi meg. Az eljárás az egész berendezés ellenőrzéséig folytatódik. A módszert a használt eljárás lényegére utalóan „csizmahúzó" (boot strap) rendszerű ellenőrzésnek szokták nevezni.

Az önellenőrzést végző program szintje kétféle lehet. Ha a program a mikroszámítógép gépi utasításaiból áll, *makrodiagnosztikai programnak* nevezik. Ha a mikroszámítógép mikrovezérelt, vagy más szóval mikroprogramozható (l. a 11. fejezetet), az ellenőrzőprogram mikroutasításokból áll és mikroprogram szinten fut. Az eljárást ebben az esetben *mikrodiagnosztikának* nevezik. Az ilyen programok jó diagnosztikai felbontásúak, mert mikroszinten a mikroszámítógép egyes részeinek a hozzáférhetősége általában igen jó.

Az önellenőrző programok jelentős előnye, hogy azokat nemcsak a gyártó használhatja a végellenőrzés fázisában, hanem segítségükkel az üzembentartó is időszakos ellenőrzések során meggyőződhet berendezése hibátlanságáról. Így ezek a karbantartás hasznos eszközei. Egyes esetekben, ahol ezt a felhasználás megkívánja, az önellenőrző programok a berendezés működtetőprogramjainak szerves részei lehetnek és így az automatikus hibaellenőrzés (és karbantartás) alapját képezhetik [9].

5.1. ELLENŐRZÉSI MÓDSZEREK

A nagy bonyolultságú integrált áramkörök ellenőrzése rendkívül bonyolult és költséges feladat. A félvezetőgyártók minden egyes elemet legalább kétszer ellenőriznek. Először a gyártási folyamat közben még „szelet" stádiumban, hogy az egyértelműen rossz elemeket „kiszűrjék". (Jól ismert, hogy a mérés mellett nagyon költséges gyártási fázis a tokozás, így el akarják kerülni, hogy eleve rossz példányokat betokozzanak.) A végellenőrzés fázisa az, amikor a különböző gyárak előírásai szerinti [10] sokkal átfogóbb ellenőrzésekre sor kerül. A gyártók által használt számítógépvezérelt automatikus ellenőrző berendezések nagyon drágák (200–300 000 dollár az alaprendszerek szokásos ára), és vagy a független ellenőrzőberendezés-gyártók, vagy a nagy félvezetőgyártókhoz kapcsolódó gyárak (pl. Fairchild Instrument Inc.) állítják őket elő. Hasonlóan az igen nagy méretű számítógépekhez itt is megfigyelhető a típusválaszték

csökkenése és csak néhány jól ismert típus maradt (pl. Sentry és Megatest családok). A rendszerek nagy működési sebességűek, mivel az elemek bonyolultságának növekedésével egyre növekvő számú ellenőrzési fázis és minta szükséges. (Pl. a Texas' cég 9900 mikroprocesszora végellenőrzésénél több, mint 10 000 különböző ellenőrzési mintát alkalmaz minden egyes elemre.) Fontos jellemzőjük, hogy általános célúak (képesek sokféle mikroprocesszor, tár és kiegészítő elem ellenőrzésére), egyszerűen programozhatók, amely ebben az esetben a különböző ellenőrzésorientált „természetes" programozási nyelvek [10] [11] [12] alkalmazását jelenti. Ezek a nagy ellenőrző berendezések képesek valamennyi elektromos paraméter (időzítés, logikai feszültségszint, kimenetipont-terhelés stb.) változtatására, bármelyik működési mód szimulálására, az adott elemre jellemző legkedvezőtlenebb esetnek megfelelő ellenőrzési mód előállítására. Bár a fő szempont az ellenőrzés biztonsága (test validity), mégis a nagy bonyolultság statisztikus következménye, hogy viszonylag nagy számú nem tökéletesen működő elem (pl. mikroprocesszorok esetén kb. 5% [10]) jut el a felhasználókhoz.

Mivel a kész mikroprocesszoros rendszerek és rendszerelemek (áramköri kártyák) hibakeresése és javítása nagyon költséges, a felhasználóknak is általában érdemes valamilyen módon felkészülni a beérkező elemek vizsgálatára. Ez a felhasználás „jellegétől" függően jelentheti az elemgyártóknál használt bonyolultságú mérőautomaták alkalmazását, az erre a célra gyártott kisebb ellenőrző berendezések használatát, vagy a legkisebbek esetén legalább a kártyaellenőrzés körülményei közötti vizsgálatot. A kártyavizsgálók nem nyújtják a legmegfelelőbb körülményeket az elemvizsgálatra, de a kártyavizsgálat kapcsán, ha a nagy bonyolultságú elemek foglalatba kerülnek, lehetőség van a rossz példányok „kiszűrésére".

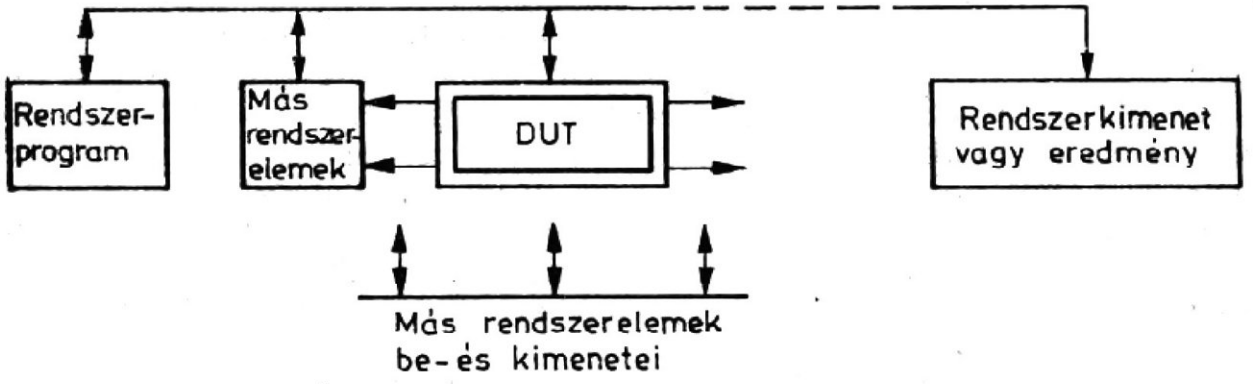
Az alapvető ellenőrzési módszereket, amelyeket az LSI bonyolultságú elemek ellenőrzése során használnak, a következőkben ismertetjük [12] [13] [14] [15].

Rendszeren belüli vagy „behelyettesítéses" módszer

Ebben az esetben a DUT-t (Device Under Test: ellenőrzött elem) ismert működő rendszerbe helyezik és viselkedését természetes körülmények között, általában a tényleges alkalmazás feltételeinek megfelelően vizsgálják meg (5.1. ábra). Pl., ha az adott program fut és megfelelő kimenő adatot vagy eredményt kapunk, a DUT-t jónak nyilvánítjuk.

Az eljárás nyilvánvaló előnye, hogy nem kell külön ellenőrző rendszert vásárolni és minimális a szoftverkészítés költsége is. Ui., ha szükség is van külön ellenőrzést végző programra, azt a rendszertervező számára ismerős környezetben és programnyelven lehet megírni.

A hátrányok ugyancsak kézenfekvőek: ezzel a módszerrel valódi minősítő vizsgálatot nem lehet végezni és hibadiagnosztikai eredményeket is csak nehezen kap-



5.1. ÁBRA. „Behelyettesítéses" ellenőrzési módszer

hatunk. A legfőbb probléma, hogy mivel a végtermék az ellenőrzés eszköze, általában nem lehet teljes elektromos vizsgálatot végezni (a „marginálásnak", forszírozott üzem elvégzésének stb. a körülményei nincsenek meg).

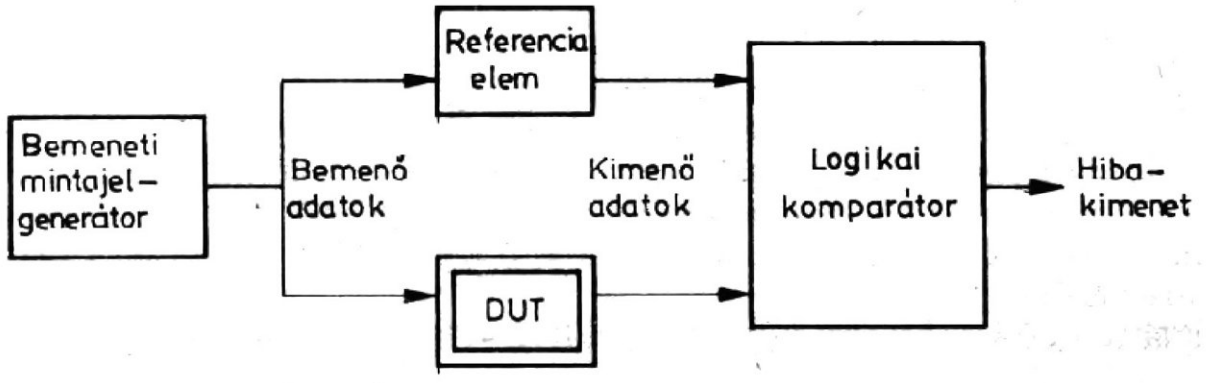
Ellenőrzés összehasonlítással

Ebben az esetben egy – a vizsgált elemmel azonos típusú – referenciaelemmel hasonlítjuk össze a DUT-t (5.2. ábra). A megoldás egyszerű és olcsó, mivel a referenciaelem segítségével képezzük a vizsgálóminta hatására a mérendő elemtől várható választ. Az egyszerűség árát ebben az esetben ismét a módszer korlátaival fizetjük meg. Paraméterváltozások hatását nehéz mérni. Ha mégis ilyen ellenőrzést is akarunk ezen a módon megvalósítani, a referencia- és a mérendő elemre azonos paraméterváltozásokat kell adni. A referenciaelemnek az adott típus rendelkezésre álló példányai közül a „legjobbnak" kell lennie. Szükség szerint, lehetőség van mind a vizsgáló mintagenerátor, mind az összehasonlító elem programozott változatának alkalmazására is.

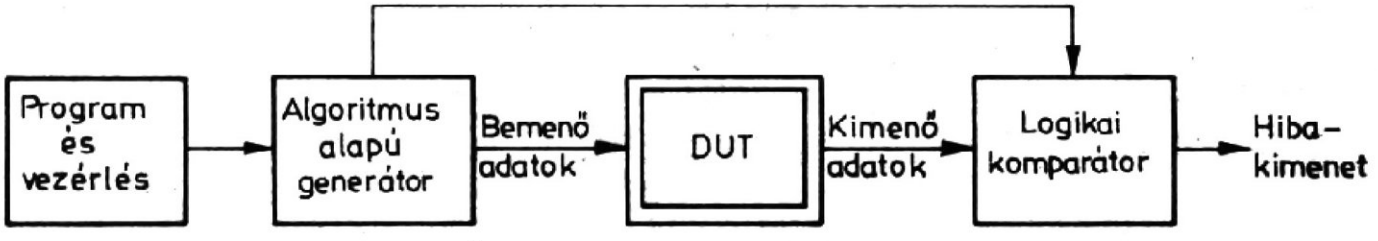
Algoritmus alapú ellenőrzés

Az algoritmus alapú ellenőrzés a DUT működését egy adott algoritmus szerinti viselkedésre egyszerűsíti. Az ellenőrző berendezés (amely gyakorlatban egy nagy sebességű speciális működésű számítógépnek tekinthető) azután ezt az algoritmust használja fel – a DUT által adott – válasz kiszámítására. Ez a módszer nagyon jól használható tárak és tárolóelemek ellenőrzésére. Lehetőség van a szükséges nagy sebességű ellenőrzőminták (l. később) generálására és ez könnyen kombinálható különböző paraméterek változtatásával is (5.3. ábra).

A mikroprocesszorok és komplex kiegészítő elemeik ilyen ellenőrzése azonban – mivel mind a DUT, mind az algoritmus részletes ismeretét szükségessé teszi – nagyon bonyolult és hatalmas mennyiségű programozói munkát is jelent. Így csak a legfejlettebb (és legdrágább) ellenőrző berendezéseknek (pl. Fairchild Sentry) vannak a legfontosabb mikroprocesszorok vizsgálatához szükséges algoritmusai.



5.2. ÁBRA. Ellenőrzés összehasonlítással



5.3. ÁBRA. Algoritmus alapú ellenőrzés

Ellenőrzés előre tárolt igazságtáblázat segítségével

Az 5.4. ábra szerint az előre tárolt vizsgálómintát a DUT-re adjuk, és a kimenetén lévő jelet összehasonlítjuk az ugyancsak előre tárolt, várt (helyes) eredménnyel. Az eljárás ugyan nagyon egyszerű, de összetett elemek esetén a tárkapacitás szükséglete nagyon nagy. Ezért gyakorlati megvalósításához az alkalmazott számítógép segítségével megfelelő hardver- és szoftver-eljárásokat (minták részekre bontása, algoritmusszerű szubrutinok, programhurkok stb.) használnak, hogy a tárkapacitás szükségletet gazdaságos szinten tartsák.

5.2. ELEMEK ELLENŐRZÉSE

Félvezetős tárak

A digitális integrált áramkörökhöz alkalmazott funkcionális ellenőrzés, amely a teljes igazságtáblázat ellenőrzéséből állt, már a viszonylag kis kapacitású tárak esetén sem volt megfelelő.

Egy n bemenetű kombinációs hálózatra ui. 2^n különböző mintát adhatunk (egy ugyanennyi bemenetű, de m szintű hálózatra pedig már $2^{n \cdot m}$-t) [16] [17] [18].

Félvezetős tárolóelemek mérése és ellenőrzése (a gyártók és felhasználók számára egyaránt) a következő csoportokba sorolható:

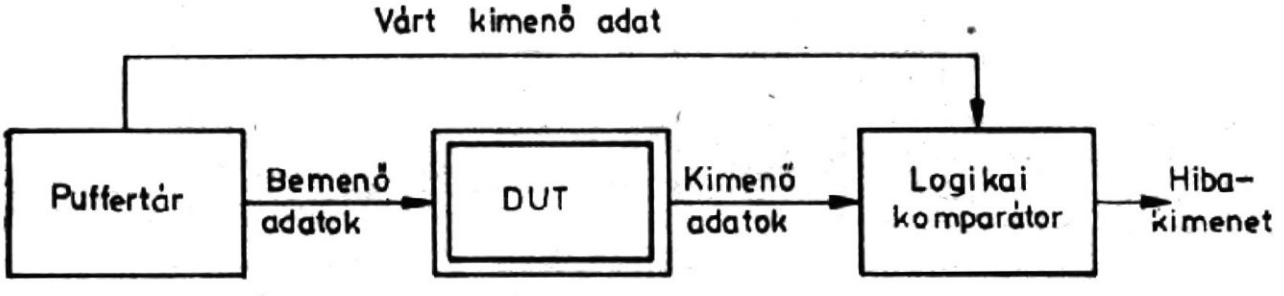
- Egyenáramú paraméterek mérése.
- Funkcionális vizsgálat.
- Dinamikus paraméterek vizsgálata.
- Felfrissítési idő ellenőrzése (csak dinamikus tárak esetén).

Az egyenáramú paraméterek mérése csak durva ellenőrzésre, ill. durva hibák (pl. nagy szivárgási vagy tápáram, nem megfelelő letörési feszültség stb.) gyors felderítésére alkalmas. A mérési módszer egyéb integrált áramköröknél szokásos: vagy feszültséget adva áramot mérünk, vagy áramot kényszerítve feszültséget.

Nagy kapacitású félvezetős tárak funkcionális vizsgálatában legnagyobb probléma a legjobb kompromisszumot megtalálni a mérési idő és az ellenőrzés alaposssága között. Erre a célra az elmúlt években nagy számú mintasorozatot [16]...[20] dolgoztak ki.

Nagy kapacitású RAM ellenőrzésének stratégiája a következő [17]:

1. Az előforduló hibák analizálása.
2. Az egyes tipikus hibákat kiváltó mintasorozatok kiválasztása.
3. Az egyes mintákhoz tartozó jelsorozatokat különböző körülmények (tápfeszültség és hőmérséklet) között futtatva a legkedvezőtlenebb eset kiválasztása (characterisation).
4. A legkedvezőtlenebb eset körülményei között rövid, de átfogó vizsgálat végzése.



5.4. ÁBRA. Ellenőrzés tárolt igazságtáblázat felhasználásával

Egy RAM-cella felépítése különböző, attól függően, hogy a tár statikus vagy dinamikus, MOS vagy bipoláris technológiával készült [17]. Egy tetszőleges cella kiválasztásakor a sordekódolónak a megfelelő sort, az oszlopdekódolónak a megfelelő oszlopot kell kiválasztania. A hibátlan működéshez, a megfelelő címzésen kívül jól működő tárcellákra, író–olvasó áramkörökre, be-, ill. kimeneti kapuáramkör-rendszerre is szükség van. A tárak jellegzetes hibái [16] [18] ezekhez a részekhez kapcsolódnak. Ezek lehetnek:

- a tárcellák szakadása, ill. rövidzára,
- a címző-, dekódoló-áramkör szakadása, ill. rövidzára,
- az egyes elemek egymásra hatásából adódó hibák (pl. egy cella helyett többe való írás),
- a hozzáférési idő megnövekedése.

A következőkben bemutatunk néhány olyan tárvizsgáló algoritmust (elrendezést), amely a fenti tipikus hibák kimutatására legjobban elterjedt.

A cellák közötti rövidzár vagy a kivezetés szakadása a legprimitívebb és így legegyszerűbben felderíthető működési hiba. Kimutatására elég a szomszédos cellákba egyszer azonos, másszor ellentétes adatot írni.

Egyik legátfogóbb, egyszerű minta (5.5. ábra) a menetelő (marching), amelyet a következő algoritmus ír le [21]:

$$\sum_{i=0}^{r} \sum_{j=0}^{c} [RC_{ij}W(1)C_{ij}],$$

ahol az egyes betűk jelentése a következő: $W(1)$ írj *1*-et; C tárcella; i a sorok sorszáma (címe); j az oszlopok sorszáma (címe); r a sorok száma -1; c az oszlopok száma -1; R olvass.

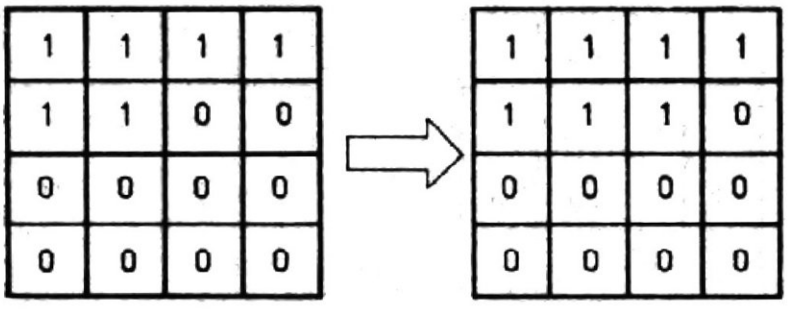
A minta előállítása (mint minden későbbi vizsgálatnál is, ha ezt külön nem említjük) tiszta *0* tartalommal indul.

A vizsgálatot el szokták végezni a zérus címtől növekvő és a legnagyobb címtől csökkenő címekkel (menetelő *0*), sőt csupa *1*-es tartalommal indulva, *0*-t beírva is (menetelő *1*).

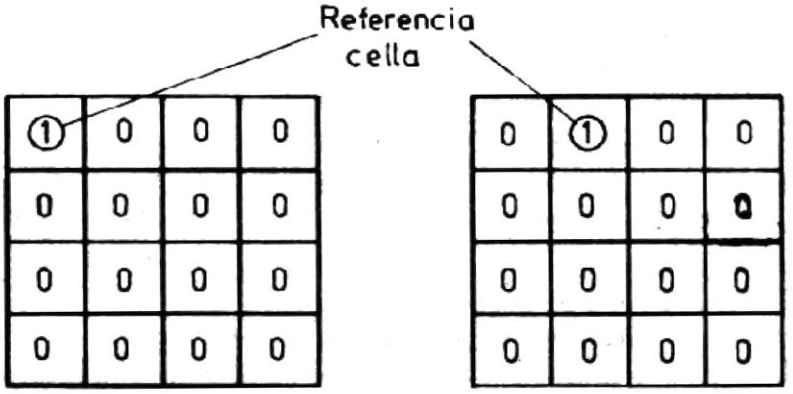
Jellegzetes címdekódoló hibára vezethető vissza a hozzáférhetetlen cella, a több címen hozzáférhető cella, az egy címen több hozzáférhető cella, ill. ezek kombinációi [22].

Az előző vizsgálat nem mutatja ki az, ha többszörös címzés esetén a téves helyre beírt adat nem véglegesen marad meg. Olyan algoritmusra van tehát szükség, amely minden címen ellenőrzi, hogy a cím–cella egymáshoz rendelés a teljes tármezőre nézve kölcsönös és egyértelmű-e.

Ilyen algoritmus a sétáló (walking) mintáé (5.6. ábra) [21].



5.5. ÁBRA. Menetelő minta



5.6. ÁBRA. Sétáló minta

$$\sum_{i=0}^{r} \sum_{j=0}^{c} [W(1)\, C_{ij},\ R(\text{összes}),\ W(0)C_{ij}]$$

ahol az új jelölések jelentése a következő:

R(összes) olvasd ki az összes cellát; *W*(0) írj 0-t.

Ezen ellenőrzés során tehát végigtolunk egy logikai *1*-et a *0*-val feltöltött táron és minden lépés után ellenőrizzük az összes cellát, hogy nem változott-e meg valahol egy cella tartalma. Nyilvánvaló, hogy az *R*(összes) műveletét úgy érdemes megvalósítani, hogy a referenciacella tartalmát mindig utolsóként olvassuk ki. (Így ui. azt is ellenőrizhetjük, hogy az olvasások során fellépő tranziens jelek nem módosították-e a többitől eltérő adatot tartalmazó cellák tartalmát.)

E megoldás nagy kapacitású (4kbitnél nagyobb) tárak vizsgálata esetén már túl hosszú időt igényel, ezért ezekhez már egyszerűsített mintákat alkalmaznak.

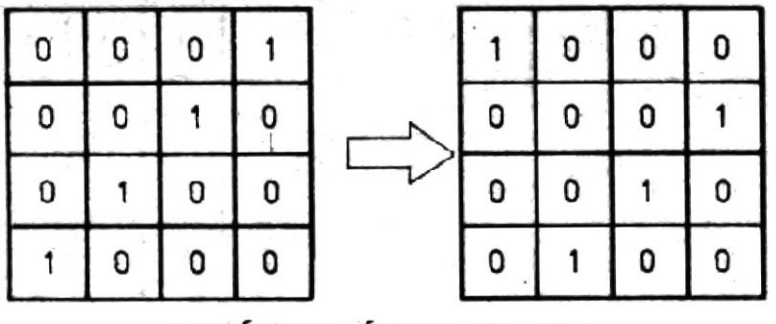
Ilyen vizsgálóminta az átlóeltolás (walking diagonal vagy diagonal shifting) (5.7. ábra) [17].

Négyzetes elrendezésű tármátrixot feltételezve:

$$\sum_{k=0}^{r} \left\{ \sum_{j=0}^{c} \sum_{i=j+k}^{c} [W(1)C_{ij}],\ R(\text{összes}) \sum_{j=0}^{r} \sum_{i=j+k}^{r} W(0)C_{ij} \right\},$$

ahol *k* egy másik változó az oszlopok címzésére, és az *R*(összes) műveletet soronként kell elvégezni.

Ez az eljárás először az átló elemeit tölti fel *1*-esekkel, majd kiolvassa sorok szerint a teljes tárat. Ezután „eggyel jobbra tolja" az átlót és újra soronként kiolvas. Az eljárást addig ismétli, amíg az egyesek az átló kezdeti helyére vissza nem térnek.



5.7. ÁBRA. Átlóeltolás minta

A címdekódoló vizsgálatán kívül ez az ellenőrzési mód alkalmas egy jellegzetes elérési idővel kapcsolatos hiba (az érzékelőerősítő feléledési hibája) felderítésére is. Az olvasással soronként haladva ui. hosszú ideig, azonos logikai jelszintű adat olvasása után, mindig következik egy ellenkező szintű adat.

Egy adat valamelyik cellába való beírása megzavarhatja a környező cellák tartalmát a közöttük lévő csatolástól függően. A cellának környező zavarokra való érzékenységét a következő tényezők határozzák meg:

- a zavaró forrás feszültségváltozásának sebessége (meredeksége),
- a csatolást biztosító (parazita-) kapacitás nagysága,
- a megzavart cella impedanciája.

Ennek alapján világos, hogy a jelenség általában nem kétirányú. (Bár a második tényező mindkét irányban azonos, de az első és harmadik már egészen különböző lehet.)

Már az egyszerű menetelő program is ad információt a környezeti zavarok kimutatására. Legátfogóbb módszer az írásra visszatérő galopp (write recovery galopp), amelyet az elérési idő vizsgálata kapcsán majd még bemutatunk. Ez azonban nagy lépésszáma miatt nagy kapacitású tárak esetén gazdaságosan nem alkalmazható. Ennek egy alcsoportja az oszlop galopp (column galopp) [17] egyben a felfrissítési időnek a legkedvezőtlenebb esetre vonatkozó ellenőrző szekvenciája dinamikus tárak esetén:

$$\sum_{j=0}^{c}\left\{\sum_{i=0}^{r}[RC_{ij}, W(1)C_{ij}], \sum_{i=0}^{r}\sum_{k=0}^{c}[RC_{ij}, RC_{ij}], \sum_{i=0}^{c}W(0)C_{ij}\right\}.$$

Az elérési idő a félvezetős tárak egyik legfontosabb és legnehezebben ellenőrizhető jellemzője. Ez a paraméter azért vizsgálható nehezen, mert értéke az adott áramkör – megvalósítással kapcsolatos – jellemzőin kívül (áramkör topológiája, kivezetések rendezése) az „előzményektől" (a vizsgálat előtti tartalom) is függ.

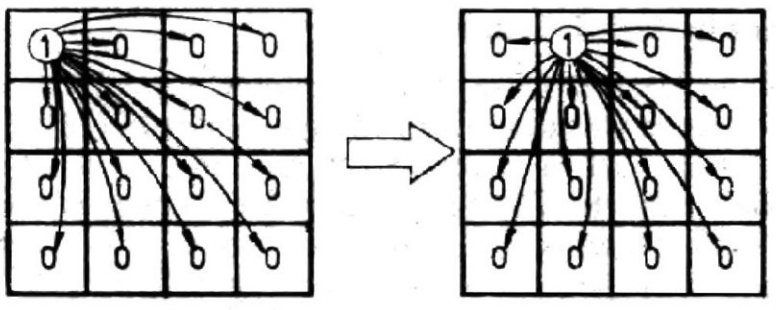
Egy cella megcímzésekor ui. két alapvető esemény zajlik le [22]:

- az előző cella kimenete lekapcsolódik,
- az új cella kimenete rákapcsolódik a közös be-, ill. kimenetre.

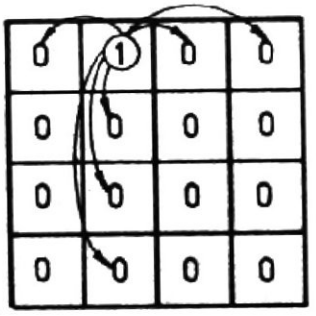
E két esemény versenyfutásától függ, hogy helyes vagy hibás a működés. Ebből következik, hogy ennek időtartama függ az előző címtől, mivel a címkódoló felépítése miatt [22] az eltérő címekre különbözőek a jelutak.

Az elérési idő vizsgálatakor leginkább ajánlott algoritmus: a galopp [21], amely minden kiolvasott cella után újra és újra kiolvassa a referenciacella tartalmát, így ellenőrizve az egymást követő címváltozások minden lehetséges kettős kombinációját (5.8. ábra).

$$\sum_{i=0}^{r}\sum_{j=0}^{c}\left[RC_{ij}, W(1)C_{ij}\sum_{k=0}^{r}\sum_{i=0}^{c}(RC_{k_1}, RC_{ij}), W(0)C_{ij}\right].$$



5.8. ÁBRA. Galopp minta



5.9. ÁBRA. Butterfly (pillangó) minta

Az összefüggésben $i \neq k$ és $j \neq 1$.

Mivel a galopp nagy kapacitású tárak esetén már nem futtatható gazdaságosan, rövidített mintákat fejlesztettek ki. Ezekkel az egyes hibáknak megfelelő legkedvezőtlenebb esetet vizsgálják.

Pl., ha a vizsgálat során a megelőző cella nyitva marad, akkor az új cellával csak akkor tudják rövidrezárni egymást, ha közös oszlopban vannak. Ilyen hibát ki lehet mutatni a pillangó (butterfly) minta segítségével (5.9. ábra) [17]:

$$\sum_{i=0}^{r} \sum_{j=0}^{c} \left\{ RC_{ij},\ W(1)C_{ij},\ \sum_{k=1}^{i+r} [RC_{kj}, RC_{ij}]\ \sum_{l=j}^{j+c} [R_{il},\ RC_{ij}],\ W(0)C_{ij} \right\}.$$

A fentieken kívül még gyakran előforduló és így figyelembe veendő hiba [17] [22] az érzékelőerősítő feléledési idejének a megnövekedése, amely a hosszú ideig azonos logikai jelszint után az ellenkező jelszint olvasásakor lép fel. Ennek kimutatására az ún. írásra visszatérő (write recovery) mintákat ajánlják [20].

A legátfogóbb algoritmus nagy kapacitású tárak elérési idejének mérésére a csillag (star), amely a pillangó és az átlós galopp kombinációjából álló minta:

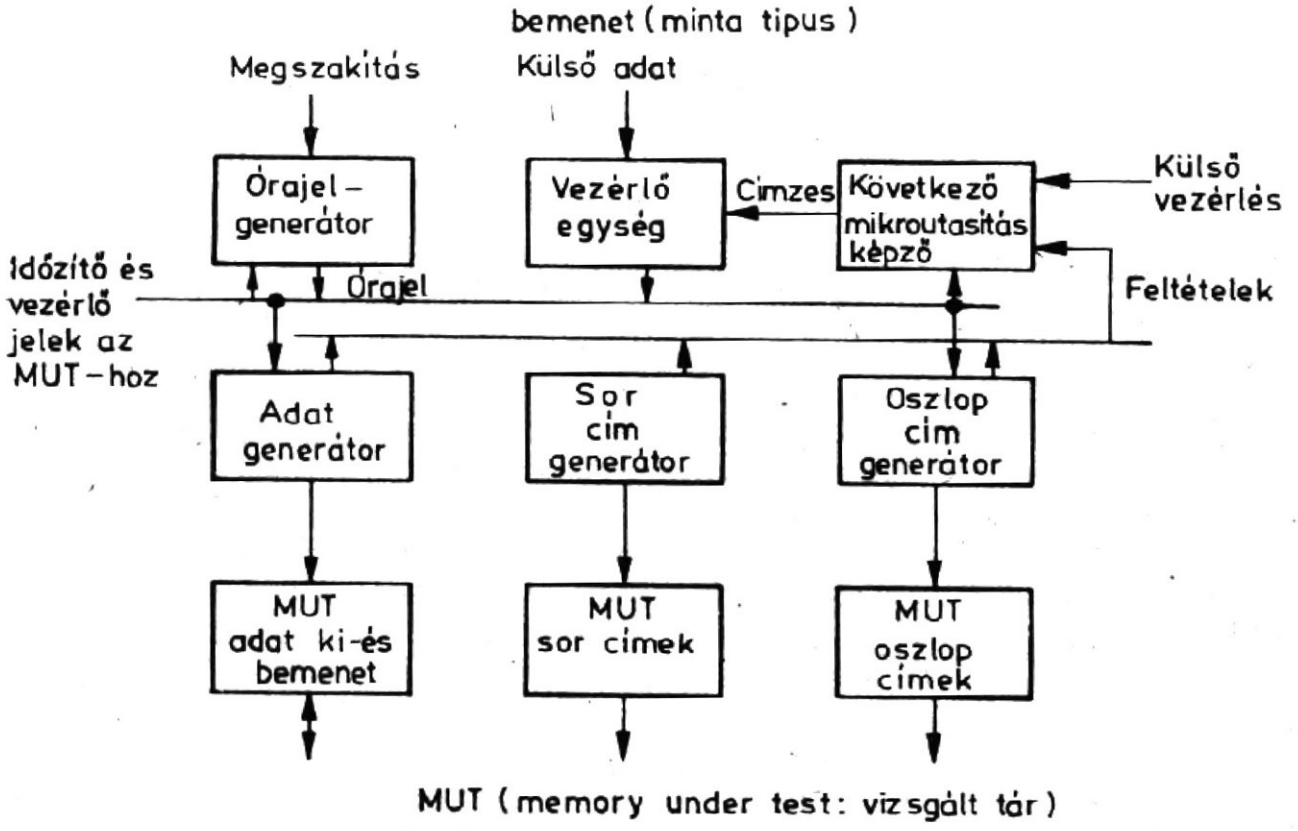
$$\sum_{i=0}^{r} \sum_{j=0}^{c} \left\{ RC_{ij}\ W(1)C_{ij},\ \sum_{l=j}^{j+c} [RC_{kl},\ RC_{ij}],\right.$$
$$\left. \sum_{k=1}^{j+c} \sum_{l=j}^{i+r} [RC_{kl},\ RC_{ij}],\ \sum_{k=1}^{j-c} \sum_{l=j}^{i-r} [RC_{kl},\ RC_{ij}],\ W(0)C_{ij} \right\}.$$

A félvezetős tárolóelemek funkcionális vizsgálatához ma már szinte kizárólag mikroprogramozható (l. a 11. fejezetet), nagy sebességű, algoritmikus jelmintagenerátorokat [18] [19] használnak (5.10. ábra).

Az 5.1. táblázatban a legfontosabb ellenőrző minták megvalósításához szükséges tárciklusok számát tüntettük fel, ahol az egyes formulákban N a tárrekeszek száma (kapacitása). Az utolsó oszlopban 500 ns ciklusidőt és 4 kbit kapacitást feltételezve megadtuk a minta egyetlen teljes végigfuttatásához szükséges időt is.

Mikroprocesszorok és kiegészítő elemek

A mikroprocesszorok és komplex kiegészítő elemeik vizsgálata – az elemek bonyolult felépítése miatt – hosszú vizsgálati időt igényel, akármelyik, az előzőekben ismertetett ellenőrzési módszer szerinti berendezést használjuk is. Pl. a Fairchild Sentry VII., az egyik legjobban ismert – félvezetőgyárakban tömeges ellenőrzésre használt – „profi" berendezés 8080 diagnosztikai programjában [10] az utasításkészlet egyes utasításai csak egyszer szerepelnek, mások azonban többször is, hogy hatásuk a különböző állapotjelzőbitekre is megvizsgálható legyen.



5.10. ÁBRA. Algoritmikus jelminta-generátor

5.1. TÁBLÁZAT. Különböző RAM vizsgálóminták „komplexitása"

Minta	Vizsgálati idő x ciklusidő	Vizsgálati idő (N = 4 kbit és 0,5 μs ciklusidő)
Menetelő (marching)	$2N$	4 ms
Sétáló (walking)	$2N^2$	16 s
Átlóeltolás (walking diagonal)	$4N+2N^{3/2}$	261 ms
Galopp	$6N+4N^2$	32 s

Gyakorlati mérési idő korlátok miatt a gyárak csupán kb. 1 másodpercet fordítanak minden gyártott komplex integrált áramkör vizsgálatára, amely csak kb. egy tizede a teljes vizsgálathoz szükséges időnek [10]. Ez egyben azt is jelenti, hogy a felhasználó nem lehet biztos abban, hogy a vásárolt elem hibamentes.

Az átlagos felhasználó számára semmiképpen sem gazdaságos a rendkívül drága, komplex LSI vizsgáló berendezések megvásárlása és üzemeltetése, így valamilyen másféle – a mérési módszer és eredmény szempontjából kompromisszumot jelentő – megoldást kell választani.

A gyakorlati szempontból számításba jövő lehetőségek a következők:

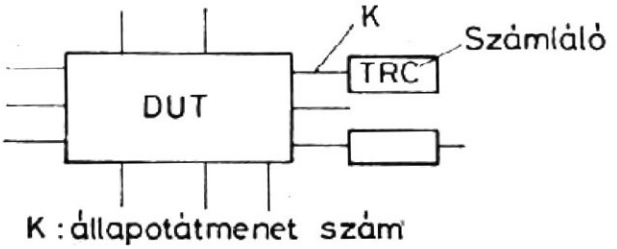
1. Kártyavizsgáló berendezés felhasználásával az elem – a konkrét alkalmazás körülményeinek megfelelő és így korlátozott – vizsgálata. A jól működő áramköri kártyába ekkor az elemet foglalat segítségével dugaszolható formában célszerű behelyezni. Logikai kártyavizsgálók azonban arra a célra készülnek, hogy sok elemet tartalmazó, komplex megszerelt kártyákat vizsgáljanak, így nem a legalkalmasabbak az egyes, a kártyán lévő, egyedi komponensek ellenőrzésére. Ilyen célra tesztprogramok írása elég nehéz feladat. Gyakorlatban ezt a vizsgálóprogramot vagy a kártyavizsgáló saját, vizsgálatorientált „nyelvén" [23] [24], vagy a vizsgált elem (pl. mikroprocesszor) saját nyelvén (utasításkészlete felhasználásával) lehet megírni [25].
2. Kidolgoztak és gyártanak egyszerű, a nem nagy tömegben felhasználó pénztárcájához illeszkedő áruvizsgáló berendezéseket is. Ezekben az összehasonlító funkcionális vizsgálat alapját képező referenciaelem a vizsgált típussal megegyező. Ez a második, az előzővel azonos típusú elem, amely a vizsgálati mintát generálja. Ezt a vizsgálati módot természetes körülmények közötti ellenőrzésnek szokták nevezni, mivel a vizsgált logikai mintasorozatot „természetes körülmények között" vagyis (a referencia- és a vizsgált elem utasításrendszere segítségével generáljuk [23]. Szokás azt a megoldást is választani [12], hogy a referenciaelem egy órajel ciklussal az ellenőrzött elem órajelciklusa előtt van. Ennek ki- és bemenő jeleit külső regiszterbe visszük, vagyis időzítését függetlenítjük.
3. Lehetőség van az egyes elemek kártyán belüli ellenőrzésére az áramkörön belüli vizsgálat (in-circuit test) eszközeivel is. Ennek módszereivel majd még a későbbiekben részletesebben foglalkozunk.
4. Mivel a mikroprocesszor maga is intelligens (vagyis programozható) eszköz, képes részt venni a rendszer elemeinek és a saját ellenőrzésében is. Ezt az önellenőrzést egy-két példa kapcsán a 5.3. szakaszban majd még bemutatjuk.

Ellenőrzést végezhetünk származtatott (redukált) jellemzők segítségével is. A mikroprocesszorok és általában az LSI áramkörök kis számú kivezetésén nagyon komplex adatfolyamot találunk, amelyet – különösen szervíz kapcsán – egyszerű eszközökkel nehéz ellenőrizni. Ez volt a fő oka, hogy olyan módszereket dolgoztak ki, amelyek segítségével ez a hosszú digitális minta komprimálható. Az ilyen módon előállított jellemzők – adott, általában a berendezés saját működéséből származtatott gerjesztés esetén – minden egyes jellemző áramköri ponthoz hozzárendelhetők, és egyszerű (hordozható) vizsgáló eszközökkel az értékük ellenőrizhető. Így lehetőség van a rosszul működő áramköri kártyán a hibás alkatrész megtalálására. Szükség szerint ez a módszer a kártyák ellenőrzésére is alkalmas egy jó kártya adott pontjáról összegyűjtött jelek és a vizsgált kártya ugyanazon pontjain kapott értékek összehasonlításával.

Az *egyik elterjedt módszer* egy adott bemenő jellemző esetén a vizsgált ponton megjelenő bináris sorozatot az ún. állapotátmenetek számával jellemzi [26] [27].

A vizsgálat során az adott pontra csatlakoztatott állapotátmenet-számláló (TRC: **TR**ansition Counter) tartalmát minden órajelnél megnöveljük eggyel, ha a megjelenő adatbit eltér az előzőtől (5.11. ábra).

A *másik ismert módszer,* minden vizsgált ponthoz az átviteltechnikában hibajavításra használt ciklikus redundancia ellenőrzés kód (CRC: **C**yclic **R**edundancy **C**heck) segítségével rendel egy jelzőszámot (signatura). Ez a jelzőszám, ami valamiféle ellenőrző összeghez hasonló, egy álvéletlen sorozatot (PRBS: **P**seudo **R**andom **B**inary **S**equence) előállító generátor segítségével képezhető. Az adott ponthoz rendelt jelzőszámot a kapott jelzőszámmal összehasonlítva analizálhatjuk a meghibásodott áramköri elemeket. A módszer neve: jelzőszám-analizis (SA: **S**ignature **A**nalysis) [26] [27] [30].



5.11. ÁBRA. Ellenőrzés az állapotátmenetek számlálásával (TRC)

A módszer az álvéletlen jelsorozatok tulajdonságain alapul [28] [29].

Egy álvéletlen bináris jelsorozat bináris *0*-ák és *1*-esek olyan elrendezése, amely véletlenszerűnek látszik. A minta azonban valamilyen jelsorozat-hosszúság után megismétlődik. A sorozat bitjei véletlenszerűek, tehát ideális statisztikus tulajdonságúak, azonban a generálás módjának ismeretében a sorozat egyes bitjei „megjósolhatók". Egy n-bites PRBS generátor által létrehozott sorozat – ismétlődés előtt – $2^n - 1$ bit hosszúságú lehet. (Ezt a sorozatot maximális hosszúságúnak nevezik.)

PRBS generátort bináris sorrendi áramkörök segítségével alakíthatunk ki. A szokásos megoldás csak kétféle alapelemből – egyszerű D-típusú (élvezérelt) bistabil multivibrátorból és kizáró VAGY kapuból (más néven mod 2 összeadóból) – épül fel [28]. A bistabilokból léptető-regisztert alakítanak ki, amelyeknek különböző fokozataiból a kimenetet – kizáró VAGY kapuk segítségével – visszacsatolják a bemenetre. A léptetőregiszter kimenetén kapjuk meg az álvéletlen sorozatot, amely megfelelően megválasztott visszacsatoló „megcsapolások" esetén maximális hosszúságú lesz.

5.2. TÁBLÁZAT. 4-bites PRBS generátor által létrehozott szekvencia

	Ciklus	Q_1	Q_2	Q_3	Q_4	$D_1 = Q_1 \oplus Q_4$
Kezdeti állapot:	0	1	0	0	0	1
	1	1	1	0	0	1
	2	1	1	1	0	1
	3	1	1	1	1	0
	4	0	1	1	1	1
	5	1	0	1	1	0
	6	0	1	0	1	1
	7	1	0	1	0	1
	8	1	1	0	1	0
	9	0	1	1	0	0
	10	0	0	1	1	1
	11	1	0	0	1	0
	12	0	1	0	0	0
	13	0	0	1	0	0
	14	0	0	0	1	1
Ismétlődés kezdete:	15	1	0	0	0	1

Az 5.12. ábrán egy $n = 4$ bites ilyen generátort tüntettünk fel. A kezdőérték 0001 állapotnak felel meg, amely után az 5.2. táblázat szerint a generátor – a maximális hossznak megfelelő – $2^n-1 = 15$ bitből álló sorozatot kiadva végiglépked négy bit összes lehetséges kombinációján (a csupa nulla elrendezést kivéve, amely állapot ki van zárva, mivel a generálási folyamat megszűnését okozná). Ha az 5.12. ábrán szaggatott vonallal jelölt bemenetre egy bitsorozatot adunk, ez megváltoztatja — megzavarja – a generátor által létrehozott belső sorozatot.

Egy léptetőregisztert, amely egy órajelnek megfelelően késleltet egy jelsorozatot, szokás egy D operátorral jellemezni, amelyet a következőképpen definiálunk:

$$x(t) = Dx(t-1).$$

Ennek alapján az 5.12. ábra szerinti regiszterbe belépő sorozat az egy és négy órajel-periódusidővel késleltetett minták és az eredeti bemenő adat összege, vagyis így írható:

$$D^4x(t) + Dx(t)+x(t) \text{ vagy egyszerűen } X^4+X+1.$$

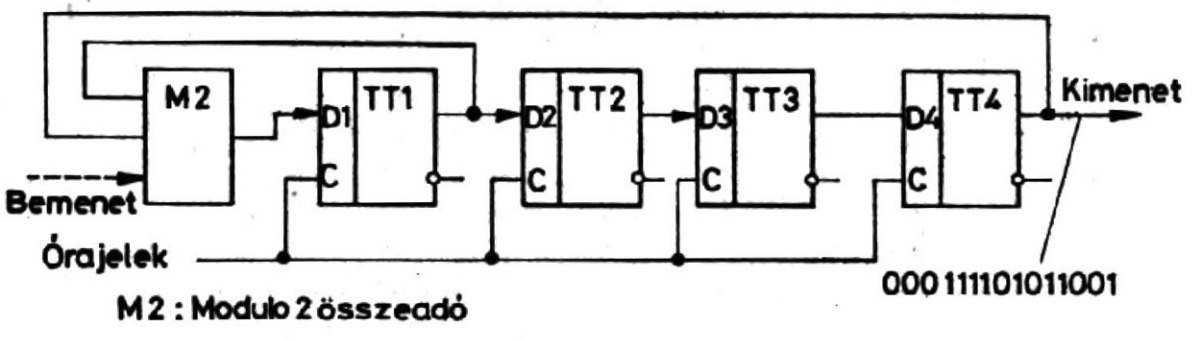
Ezt az adott léptetőregiszter-elrendezés, vagyis a generátor jellemző polinomjának (characteristic polynomial) szokták nevezni. Ennek felhasználásával a PRBS generátorba bevitt sorozat hatása úgy írható le, mint az adott sorozat elosztva a generátor jellemző polinomjával [26].

Ha most már két különböző $P(X)$ és $P(Y)$ bemenő sorozatot adunk és az előbbiekben definiált módon képzett, ezekhez tartozó kimenő sorozatok: $Q(X)$ és $Q(Y)$, akkor kimutatható, hogy

$$Q(X+Y) = Q(X)+Q(Y).$$

Ez a tulajdonság lehetőséget ad arra, hogy a PRBS generátorok segítségével hibát detektáljunk.

Ha ui. az előbbi összefüggést úgy értelmezzük, hogy X a helyes adatfolyam és $X+Y$ a hibás bemenet (Y ebben az esetben a hibasorozat), bebizonyítható, hogy bármilyen egyetlen bitnek megfelelő hiba (függetlenül attól, hogy mikor keletkezik) mindig detektálható olyan módon, hogy megállítjuk bármely időpillanatban a regisztert, és összehasonlítjuk a benne levő tartalmat (maradékot) azzal az ismert értékkel, amely a hibátlan esetnek felel meg [26]. Ez az algoritmus és a hozzá tartozó módszer tehát alkalmas adatsorozatok kompprimálására és azok meghibásodásának egyszerű eszközökkel való analizálására. A módszer lényege tehát úgy fogalmazható meg, hogy a kártya (berendezés) minden ellenőrzésre felhasznált pontjához a visszacsatolt léptetőregiszter mérési idő végén kapott tartalmát rendeljük.



5.12. ÁBRA. 4-bites álvéletlen (PRBS) generátor

A jelzőszám-analízis tulajdonságait és így hatásosságát is – az átalakított jelsorozatban keletkező – hibák jelzési valószínűségével jellemezhetjük. A módszer egyszeres hibát minden esetben jelez, vagyis $P^{(1)}_{\text{PRBS}} = 0$. Többszörös hiba esetén annak valószínűsége, hogy nincs jelzés [26]:

$$P_{\text{PRBS}} = \frac{\text{Nem jelzett hibák}}{\text{Összes hibák}} = \frac{2^{m-n}-1}{2^m-1},$$

ahol n a visszacsatolt léptetőregiszter hossza és m a jelsorozat hossza.

Így, ha m nagy,

$$P_{\text{PRBS}} \approx \frac{1}{2^n}.$$

Ugyanez a valószínűség az állapotátmenetek számlálásán alapuló módszer esetén, egyetlen meghibásodott bit figyelembevételével, mivel két azonos bit közül az egyik meghibásodása – pl. 001, 011, 100 vagy 110 bithármasok esetén a középső bit megváltozása – nem változatja meg az átmenetek számát [26]:

$$P^{(1)}_{\text{TRC}} = \frac{m-1}{2m} \approx \frac{1}{2}.$$

Általánosan, az összes hibát tekintetbe véve [26]:

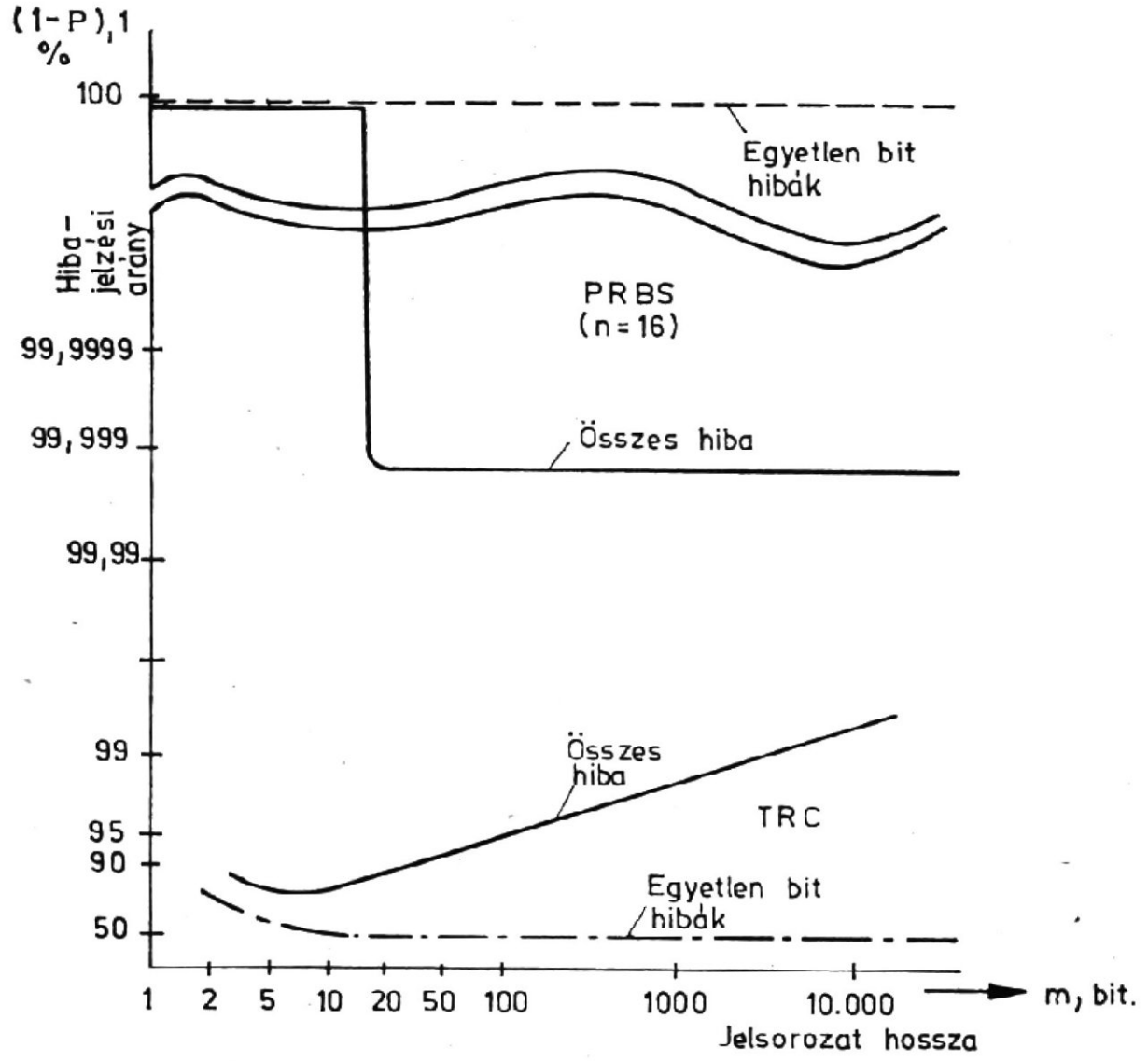
$$P_{\text{TRC}} \approx \frac{1}{\sqrt{m\pi}}.$$

Érdekes, hogy az állapotátmenetek számlálásának a módszere a legrosszabb egy hiba esetén, pontosan akkor, amikor a visszacsatolt léptetőregiszter sohasem hibázik.

A két módszer tulajdonságait jól áttekinthetjük az 5.13. ábra segítségével, ahol az m jelsorozat hosszának függvényében ábrázoltuk [26] a hibajelzés $(1—P)$ százalékos értékét mindkét módszer esetén. (A PRBS módszer esetén $n = 16$ bit, azaz négy hexadecimális karakteres jelzőszámot tételeztünk fel, amely még nagyon könnyen kezelhető.)

A jelzőszám-analízis jó hatásfokú alkalmazásának feltétele, hogy a tervező – már a tervezés fázisában – gondoskodjon az ellenőrzés módjának kidolgozásáról is. Természetesen az ellenőrzést – a szerviz és hibajavítás követelményeinek figyelembevételével – csak a kicserélhető alkatrészek „mélységéig" érdemes megvalósítani. (Nem érdemes tehát tovább analizálni egy mikroprocesszor vagy nagy kapacitású tárelem hibája esetén, hogy az elemen belül mi a hiba oka.)

A gyakorlati megvalósítás eszközeinek felhasználásával [20] a szervizhez el kell készíteni a hibaelhatárolás folyamatábráját [31], amely az egyes csomópontokhoz megadott helyes jelzőszámértékek – meghibásodás következtében való – megváltozása alapján útmutatást ad a további ellenőrzések sorrendjére és a meghibásodott elem megkeresésére. Az adatok beolvasásához természetesen mindig szükség van egy megfelelően megválasztott órajelre is. Ez azt jelenti, hogy elsősorban szinkron rendszerek



5.13. ÁBRA. A hibadetektálás valószínűsége jelzőszám-analízis és állapotátmenet-számlálás módszer alkalmazása esetén

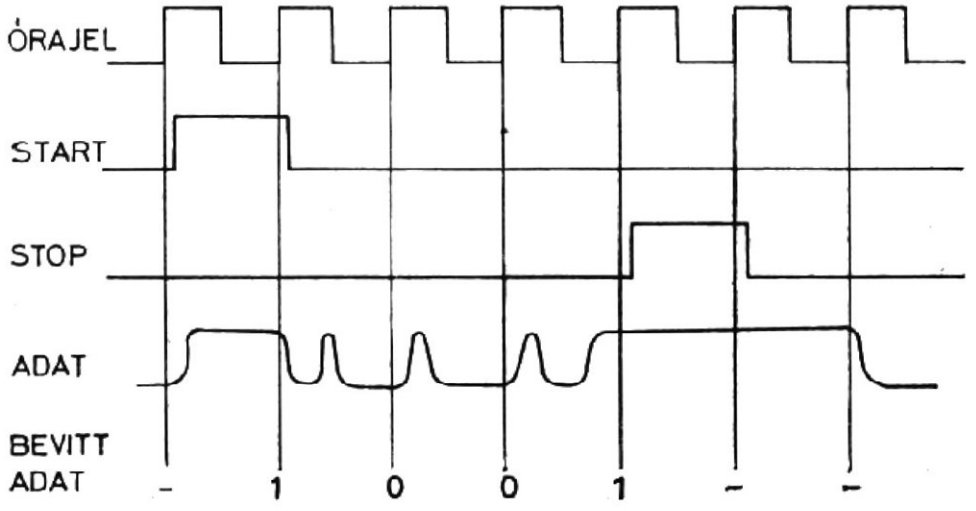
vizsgálatára alkalmas a módszer. Persze órajelnek bármilyen erre alkalmas jel választható, ami lehetővé teszi, hogy aszinkron áramköröket is ebből a szempontból szinkronnak tekintsünk.

A gyártott berendezések (pl. [30]) szondája bármelyik mérendő pontra rácsíptethető, hasonlóan végezhető el az órajel csatlakoztatása is, valamint, ha a működés szerint erre szükség van, egy további START indító és STOP megállító jelátmenet is kiválasztható (5.14. ábra).

A mérés legkényesebb pontja a megfelelő jelátmenet-pillanatok biztosítása. Az adat az órajel választott pozitív élének megfelelő időpont előtt (data setup time) és után (hold time) meghatározott ideig nem változhat. Az egyes jelek (adat, órajel, START, STOP) késleltetésének egyenlőnek kell lennie. Ezen kívánalmak teljesítésének megkönnyítésére az ellenőrző berendezések megfelelő tervezésével (azonos felépítésű áramkörök alkalmazásával az egyes csatornák esetén) gondoskodnak.

5.3. ÖNELLENŐRZÉS

A mikroprocesszoros rendszerek programozhatósága lehetővé teszi az ellenőrzés elvégzését a rendszer által nyújtott körülmények között, ill. az elemek felhasználásával. Ez a lehetőség a gyakorlatban széles körben nyújt nagyon gazdaságos módszereket.



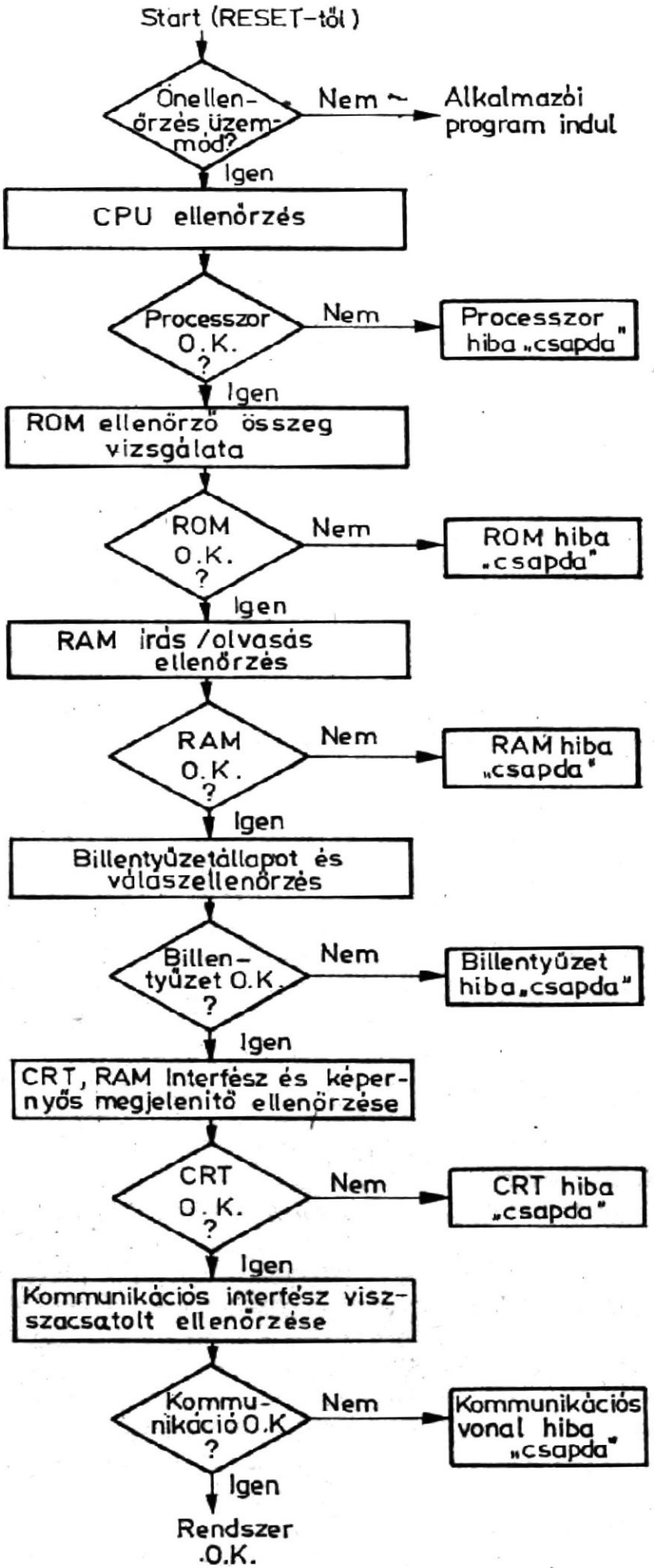
5.14. ÁBRA. A mérési idő kiválasztása az órajel-periódusok egész számú többszöröseként (START, STOP és ÓRAJEL számára a pozitívba menő éleket választottuk ki)

Így felhasználhatjuk a mikroprocesszoros alkalmazói rendszert a beérkező elemek vizsgálatára (jól működő rendszer áramköri kártyáiba dugaszolva őket), kártyák gyártás közbeni vagy végellenőrzésére (a vizsgált kártyákat egyszerűen egy jól működő rendszerbe dugaszolva vagy egy jól működő, referenciakártyával végzett pl. összehasonlító vizsgálat alapján), a szervizelésre (a vizsgálatot végző speciális pl. ROM-ban tárolt programok bedugaszolásával a felhasználói program helyett vagy az erre a célra fenntartott külön foglalatokba). A rendszerek megbízhatóságát, rendelkezésre állását (availability) nagy mértékben javíthatjuk beépített (resident) vagy behívható ellenőrzőprogramok alkalmazásával. Ezek automatikusan (pl. minden bekapcsolás után vagy adott időszakonként, ill. állapotok esetén), vagy külön kezelői parancsra megvizsgálják a rendszert, ill. annak előírt részeit. Az ellenőrzés irányulhat csak a helyes működés regisztrálására vagy hiba esetén az okok felderítésére, ill. a nem működő funkcionális elem lehető elhatárolására, esetleg az ún. hibatűrő rendszerek esetén a jól működő modulok újraelrendezését is elvégezheti (l. a 12. fejezetet).

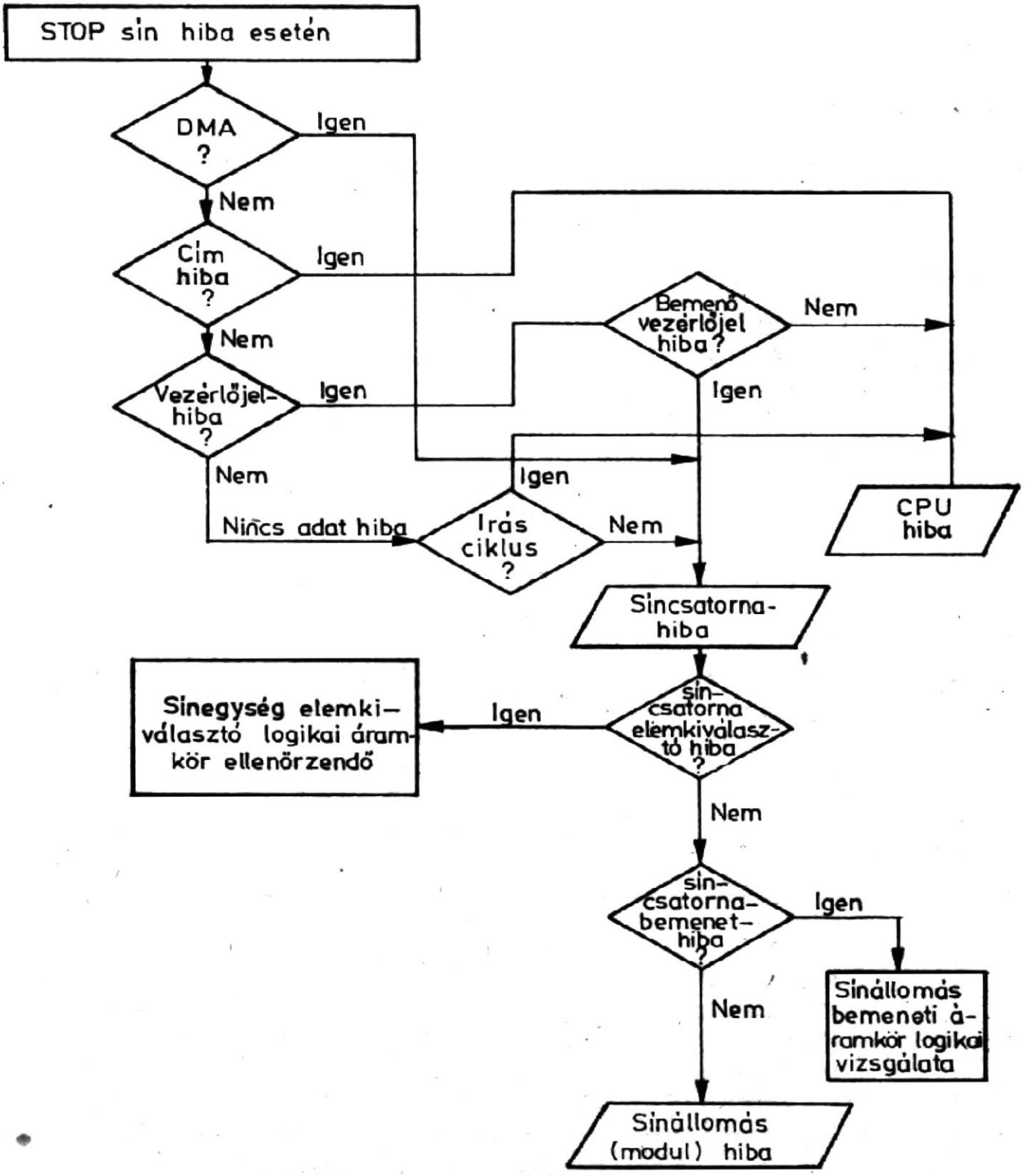
Áramköri kártya formában megvalósított funkcionális egységek és ezek közötti sínkapcsolat esetén a vizsgálatot az egyes funkcionális modulok szerint szervezhetjük (5.15. ábra). Természetesen a vizsgálat elvégzését egyes részek (elsősorban a közös sín, a processzormodul) meghibásodása meggátolhatja. Ekkor külső vizsgálórendszerek segítségével kell a hibát okozó modult elhatárolni. Egy logikai állapotanalizátor (l. a 4. fejezetet) segítségével az 5.16. ábra folyamatábrája szerint diagnosztizálhatjuk az analizátorral tárolt eredményt. Az azonosításnál az állapotinformáció is segít, valamint az a cím, és adatinformáció, amely a hiba bekövetkezésekor tárolásra került az analizátorban.

Az 5.16. ábra szerinti önellenőrző program pl. egy intelligens CRT (Cathode Ray Tube: katódsugárcsöves) megjelenítő programja [24], amelyben a vizsgálati sorrend is jellemző. A modulok ellenőrzési sorrendjének megállapításakor arra kell törekedni, hogy először a további részek ellenőrzésénél felhasznált modulok (központi egység, programtár) hibamentességéről győződjünk meg.

A programtár (általában ROM, PROM, REPROM) hibamentességéről valamilyen hibajelzési módszer alkalmazásával győződhetünk meg. Ez azt jelenti, hogy egy vagy több külön szó formájában helyezünk el a tartalommal összefüggésben lévő kiegészítő információt (pl. ellenőrző összeg, amely az összes logikai *1* bitek száma), amelyet azután a kiolvasás után ellenőrzünk (pl. az ellenőrző összeg képzésével és a tárolt értékkel való összevetéssel). Hasonló módon – pl. CRCC: Cyclic



5.15. ÁBRA. Képernyős megjelenítő funkcionális egységeinek ell enőrző programja



5.16. ÁBRA. Hibadiagnosztizálás folyamatábrája

Redundancy Check Character: ciklikus redundanciaellenőrző karakter vagy függőleges és vízszintes paritás alkalmazásával – ellenőrizhetjük a háttértárakból (pl. hajlékony lemezes tár) beolvasott információt is [29].

A RAM vizsgálatát valamilyen vizsgálóalgoritmus alkalmazásával végezhetjük. Az 5.17. ábrán a sétáló algoritmust megvalósító program folyamatábráját adjuk meg [24]. A program i8080 típusú mikroprocesszor utasításrendszerét felhasználó változatát az 5.3. táblázatban láthatjuk.

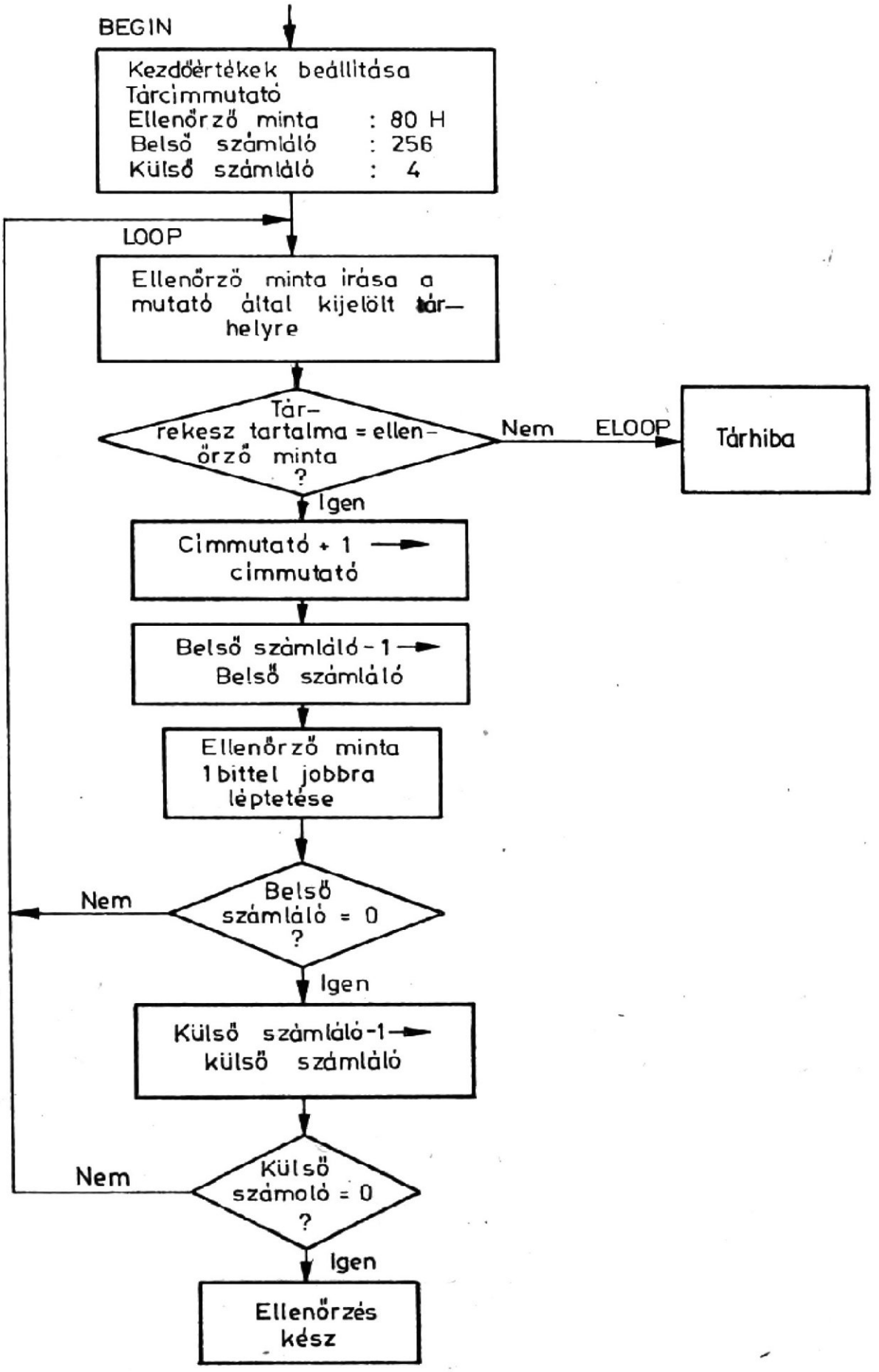
5.3. TÁBLÁZAT. 8080 RAM ellenőrző program

Sor-szám	Cimke	Utasítás	Megjegyzés	Cím/program kód
1	BEGIN:	LXI H, MEM	;MEM. CÍM POINTER H, L	0041 210010
2		MVI A, 80H	;ELLENŐRZŐ MINTA ;KEZD.	0044 3E80
3		MVI B,04H	;KÜLSŐ SZÁML.=4	0046 0604
4		MVI C,00H	;BELSŐ SZÁML.=256	0048 0E00
5		;MEGJ.: 1024 RAM HELY SZÁMLÁLÁS;		
6		;KÜLSŐ SZÁML. X BELSŐ SZÁML.= ;4×256 = 1024		
7	LOOP:	MOV M, A	;ELL. MINTA ÍRÁS	004A 77
8		CMP M	;TARTALOM ELLENŐRZÉS	004B BE
9		JNZ ELOOP	;UGRIK, HA HIBA	004C C25C00
10		RRC	;MINTA JOBBRA LÉPTETÉS	004F 0F
11		INX H	;CÍMPOINTER NÖVELÉS	0050 23
12		DCR C	;BELSŐ SZÁML. ;CSÖKKENTÉS	0051 0D
13		JNZ LOOP	;UGRIK, HA NEM NULLA	0052 C24A00
14		DCR B	;KÜLSŐ SZÁML. ;CSÖKKENTÉS	0055 05
15		JNZ LOOP	;UGRIK, HA NEM NULLA	0056 C24A00
16		JMP BEGIN	;ISMÉT KEZDI	0059 C34100
17	ELOOP:	JMP ELOOP	;HIBA „CSAPDA"	005C C35C00
18		ORG 1000H		1000
19	MEM:	DS 1024	;MEM. KEZDŐCÍM=1000H	1000

A rendkívül egyszerű programot szinte minden i8080 mikroprocesszoros rendszerben felhasználhatjuk.

A be-, ill. kimeneti modulok ellenőrzését nagymértékben egyszerűsíthetjük, ha azokat olyan módon képezzük ki, hogy az ellenőrzés idejére – speciális csatlakozók és kábel segítségével – a ki-, ill. bemeneteket egymással összeköthetjük. Számos kétirányú soros adatátvitelt megvalósító periféria-áramkör (pl. National 8250) programvezérléssel is biztosítja az ilyen módon való hurokképzéssel megvalósítható ellenőrzési módot.

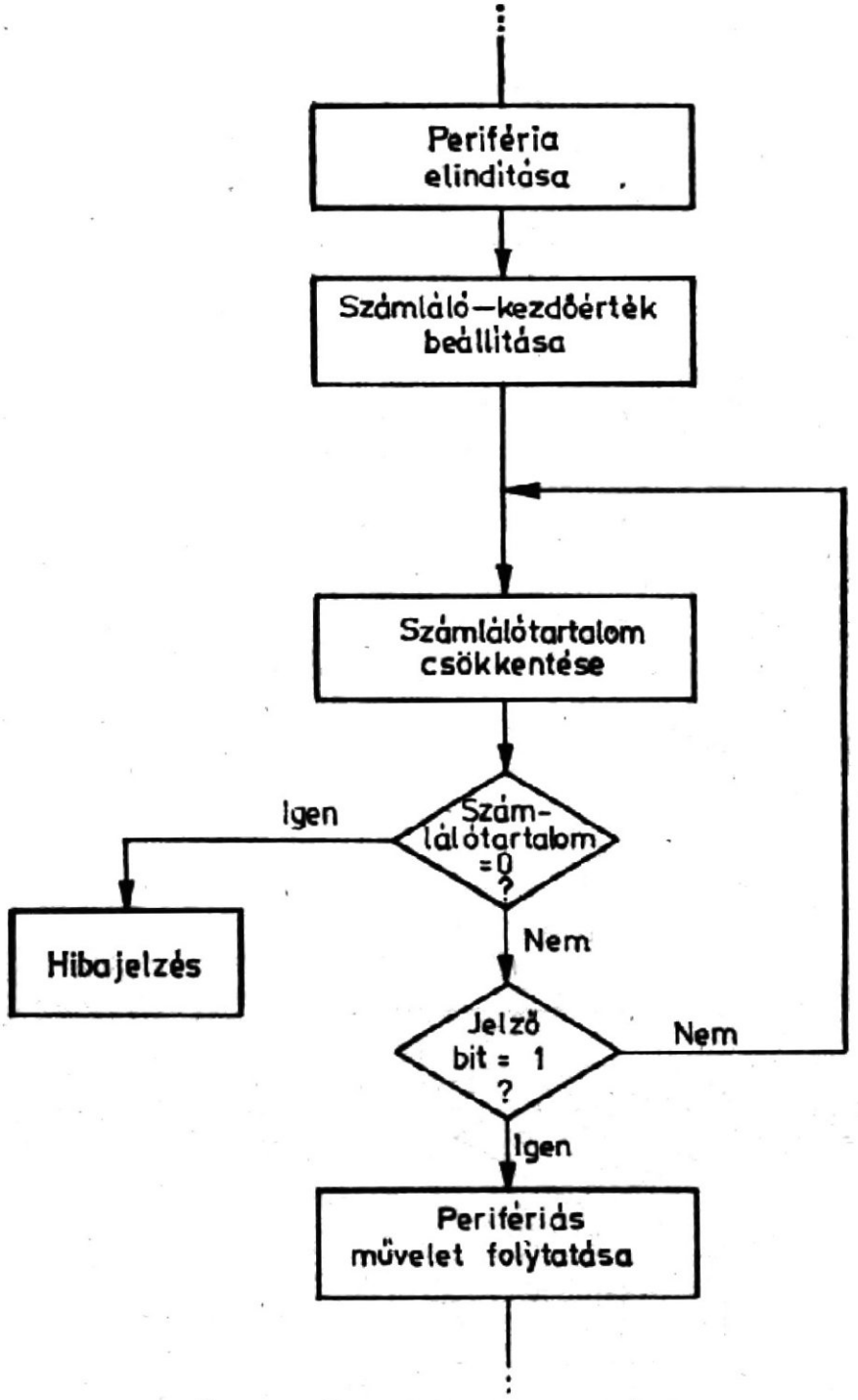
A processzor és a perifériák gyakran olyan módon működnek együtt, hogy a processzor elindít egy folyamatot és vár a perifériára, amely visszajelentkezik – általában egy jelzőbistabil bebillentésével – ha az adott feladatot elvégezte. Ha a válasz nem érkezik meg (mivel az illető periféria meghibásodott), a processzor csak vár és



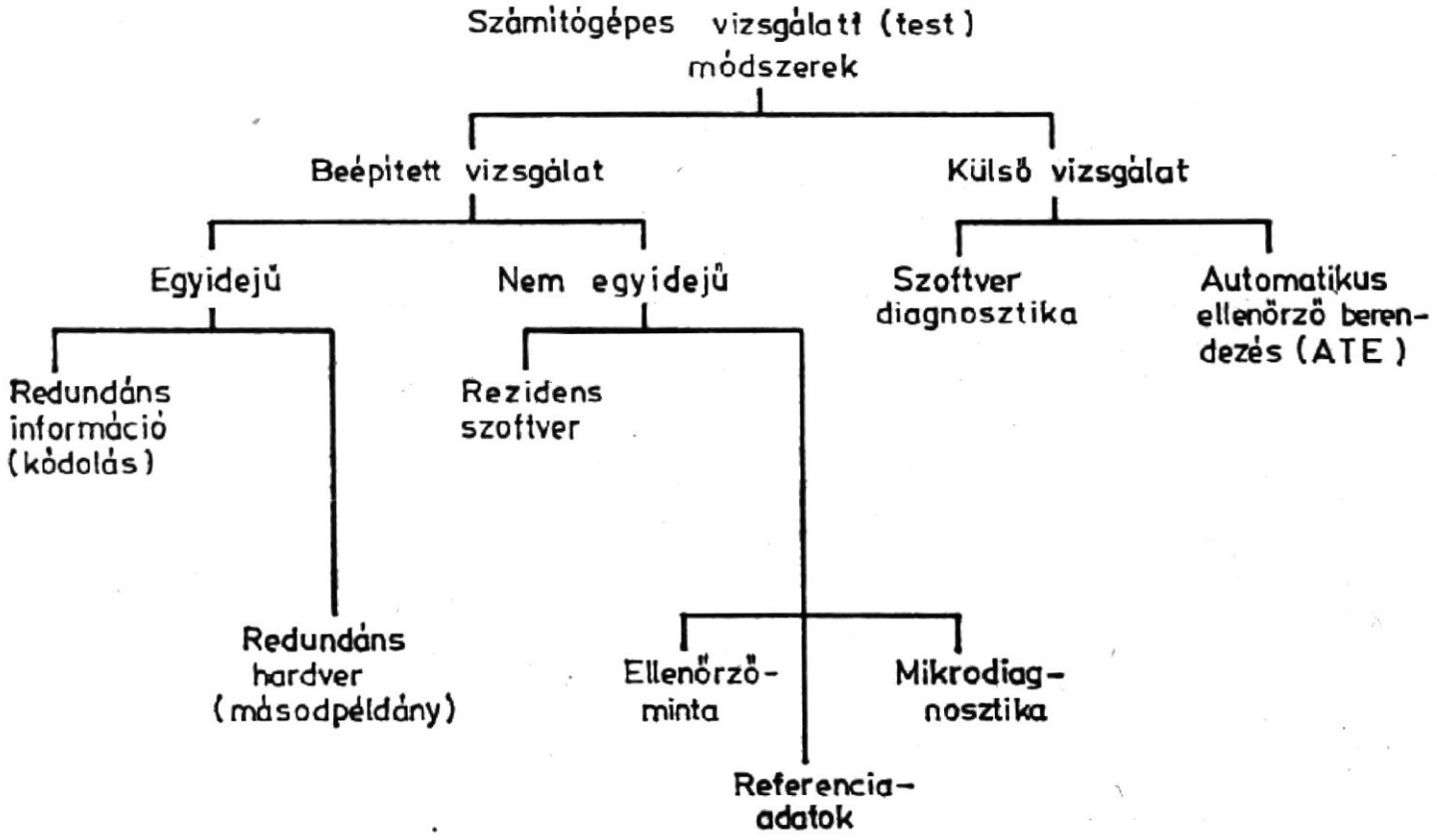
5.17. ÁBRA. Sétáló algoritmust megvalósító program folyamatábrája

a rendszer megbénul, ahogy mondani szokták, befagy. Ezt az állapotot elkerülhetjük egy általában igen kevés munkával megvalósítható ellenőrzési móddal. A módszer lényege az, hogy a processzor a válaszidőt általában egy regisztertartalomnak a változtatásával és egy előre megszabott tartalom elérésének figyelésével „méri" (5.18. ábra). Ezt, amelyet watch-dog (házőrző kutya) alkalmazásának neveznek sínrendszeren való, visszajelzett (hand-shaking) kapcsolat (l. a 7. fejezetet) meghibásodása esetén is felhasználhatjuk a rendszer működése szempontjából létfontosságú sín befagyásnak elkerülésére.

Az 5.19. ábrán még egyszer összefoglaltuk a különböző vizsgálati módszereket [33].



5.18. ÁBRA. Válaszidő mérése számlálással



5.19. ÁBRA. Számítógépes vizsgálati módszerek csoportosítása

IRODALOM

[1] *Alvarez, D.:* Consideration in semiconductor tester selection. Computer Design, 1976. No. 22. 67–74.

[2] *Kiss E.–Jobbágy A.–Tóth Z.:* Logikai áramkör tervezési elvek az automatikus tesztelés szempontjából. Mérés és Automatika, 1978. október. 370–374.

[3] *Ramamoorthy, C. V.:* Fault-tolerant computing: an introduction and an overview. IEEE Trans. on Computers, November 1971. 1241–1244.

[4] *Srini, V. P.:* Fault diagnosis of Microprocessor Systems. Computer, January 1977. 60–65.

[5] *Pynn, C. P.:* In-circuit tester using signature analysis adds digital LSI to its range. Electronics, May 24, 1979. 153–157.

[6] *Forbes, R. E. et al.:* A self-diagnosable computer. Proc. AFIPS Fall Joint Computer Conference, 1965. 1073–1086.

[7] *Toldi G.:* Mikroprogramozott célszámítógépek élesztése és tesztelése. (Szakmérnöki diplomaterv). BME, Budapest, 1979.

[8] *Ballard, D. R.:* Designing fail-safe microprocessor systems. Electronics, January 4, 1979. 139–143.

[9] *Neil, M.–Goodner, R.:* Designing a serviceman's needs into microprocessor-based systems. Electronics, March 1, 1979. 122–128.

[10] *Scrupski, S. E.:* Why and how users test microprocessors. Electronics, March 2, 1978. 97–104.

[11] *Sebestyén B.:* Számítógépirányítású mérőrendszerek. Műszaki Kiadó, Budapest, 1976.

[12] *Bluestone, A.:* Logical environment comparison testing handles complex LSI devices. Computer Design, April 1979. 95–102.

[13] *Bisset, S.:* LSI tester gets microprocessors to generate their own test patterns. Electronics, May 25, 1978. 141–145.

[14] *Cave, T.:* Compressing test patterns to fit into LSI testers. Electronics, Oct. 12, 1978. 136–140.
[15] *Campbell, J. F.:* A new real-time function test generation system for complex LSI testing. IEEE Trans. on Manufacturing Technology, 1975. No. 2. 52–55.
[16] *Crafts, J. M.:* Techniques for memory testing. Computer, October 1979. 23–31.
[17] *Dunajcsik Gy.–Szamosközi Z.:* LSI memóriák mérése. HIKI Közlemények, 1977. 2. szám. 21–36.
[18] *Marshall, M.:* Through the memory cells-futher exploration of IC-s in testing land. EDN, 1976. No. 2. 1–9.
[19] *Jackson, K.–Sear, B. E.:* Measurement techniques for testing high speed memories. Computer Design, May 1975. 127–132.
[20] *Morgan, M. K. M.:* New patterns for testing 4k RAMs. Electron, Feb. 1976. 12–14.
[21] *Barrachough, W.–Chiang, A. C. L.–Sohl, W.:* Techniques for testing the microcomputer family. Proc. of the IEEE, June 1976. 943–950.
[22] *Kucke, G.–Mize, J. P.–Carr, W. N.:* Semiconductor Memory Design and Application. McGraw-Hill Book Co. New York, 1973.
[23] *Dunajcsik Gy.–Szamosközi Z.–Szenczi Á.:* LSI áramkörök mérőrendszerei. HIKI Közlemények, 1977. 3. szám. 46–73.
[24] *Kippman, M. D.–Donn, E. S.:* Design forethougth promotes easier testing of microcomputer boards. Electronics, January 1979. 113–119.
[25] *Foley, E.:* The effects of the microelectronics revolution on systems and board test. Computer, October 1979. 32–38.
[26] *Frohwerk, R. A.:* Signature analysis: a new digital field service method. Hewlett–Packard Journal, May 1977. 2–8.
[27] *Nadig, M. J.:* Signature analysis – Concepts, examples, and guidelines. Hewlett–Packard Journal, May 1977. 15–21.
[28] *Golomb, S. W.:* Shift-register sequences. Holden-day Inc., San Francisco, 1967.
[29] *Peterson, W. W.–Weldon, E. J. Jr.:* Error-correcting codes. The MIT Press, Cambridge, Massachusetts, 1972.
[30] *Chan, A. Y.:* Easy-to-use signature analyzer accurately troubleshoots complex logic circuits. Hewlett–Packard Journal, May 1977. 9–14.
[31] *Stone, P. S.:* Circuit Board Testing: Cost-effective production test and troutleshooting. Hewlett–Packard Journal, March 1979. 2–8.
[32] *Runyon, S.:* Testing LSI-based boards: many issues, many answers. Electronic Design, March 15, 1979. 58–66.
[33] *Clary, J. B.–Sacane, R. A.:* Self-testing computers. Computer, October 1979. 49–59.
[34] *Крабский, В. В.—Пархоменко, П. П.—Сейтова, Г. Е.:* О построении проверяющих последовательностей для дискретных устройств с памятью. Автоматика и телемеханика, май, 1979. 167—177.
[35] *Правильщиков, П. А.—Щепин, В. С.:* Составление структурных программ в диалоговом режиме с одновременной генерацией тестов. Автоматика и телемеханика, август, 1979. 129—137.

6. BE- ÉS KIVITELI MÓDSZEREK ÉS ESZKÖZÖK

A mikroszámítógépek és a külvilág közötti kapcsolat és információközlés a be-, ill. kimeneti csatornák (interfészek) és a hozzájuk csatlakozó perifériák közvetítésével valósítható meg. Az általánosan használt perifériák közé sorolhatók: az írógépek, képernyős megjelenítők, sornyomtatók, lyukszalagolvasók, -lyukasztók, mágnesszalagos egységek (mikroszámítógépekhez elsősorban kazettás egységek), mágneslemezes egységek, analóg–digitális és digitális–analóg átalakítók stb. A perifériák és a mikroszámítógép közötti kapcsolatot interfészáramkörök valósítják meg. Ezek feladata a perifériáktól a mikroszámítógép felé, ill. a mikroszámítógéptől a periféria felé irányuló információátvitel adatainak a mikroszámítógép, valamint a periféria számára megfelelő formátum szerinti átalakítása. Általában az interfészáramkörök perifériákkal kapcsolatos ún. állapotadatokat is előállítanak (ilyen adat pl. a FOGLALT-KÉSZ jel). Ezenkívül a számítógép műveleti sebességét összehangolják a perifériák működésével.

A periféria-interfészek négy, általánosan jelentkező feladata:

1. az adatpufferelés,
2. a címdekódolás vagy eszközkiválasztás,
3. az utasításdekódolás és értelmezés,
4. az időzítés és a vezérlés.

A *pufferelésre* az adatátvitel szinkronizálása miatt van szükség. A *címdekódolás* akkor kell, ha egynél több perifériát használ a rendszer. A címdekódolással választható ki a szükséges periféria. *Utasításdekódolásra* általában azért van szükség, mert a periféria vagy az interfészáramkör nemcsak egyféle műveletet (adatátvitelt) láthat el. Az éppen szükséges művelet vagy az adott konfigurációban, adott rendszerben igényelt műveletek megadására, kiválasztására szolgálnak az utasítások (parancsok). Az *időzítés* és a *vezérlés* mindezen funkcióknak az ellátásához elengedhetetlenül szükséges.

A mikroszámítógép és a külvilág közötti adatcsere lehet:

- programozott adatátvitel,
- ciklusloposos (cycle-stealing) átvitel.

Az első eljárás szerinti átviteleket a mikroprocesszor programja vezérli. Egy-egy programlépés (utasítás) egyidejűleg csak egy adatszónak a mikroszámítógép és a periféria közötti átvitelére vonatkozik. A második csoport interfészáramkörei általában ciklusloposos eljárással valósítják meg a közvetlen tárhozzáférést (DMA: Direct Memory Access). Vezérlésük a periféria oldaláról történik, és ez az eljárás definiált adatblokkok átküldését teszi lehetővé a tár és a periféria között. Az átvitel közvetlenül a periféria és a tár között megy végbe oly módon, hogy egy vagy több adat átvitelének az időtartamára a rendszer a mikroprocesszor tárral való kapcsolatát felfüggeszti, azaz a processzor működésétől „ellopja" az átviteli ciklusokhoz szükséges időt. A program az adatblokkok átvitelében, annak vezérlésében nem vesz részt, csupán kezdeményezheti azt, ill. értesül az átvitel befejeztéről.

6.1. PROGRAMOZOTT ADATÁTVITEL

A programozott adatátvitel megvalósításához viszonylag egyszerű hardverre van szükség. Az adatok átvitele teljes egészében a program felügyelete alatt zajlik, ami sok alkalmazásban egyszerű és rugalmas adatkezelést és adatfeldolgozást eredményez [1]. Hátrány azonban a viszonylag alacsony átviteli sebesség, és az, hogy a programfutási időből a processzor jelentős hányadot szentel az adatok be-, ill. kivitelének. Mindez a programvezérlésből adódik. Éppen ezért ezt az átviteli módszert a mikroszámítógéphez képest lassú perifériák kezelése esetén alkalmazzák. Általában egy adat átvitelének a lebonyolításához 25...200 μs-nyi programidő szükséges, ami 5000-től 40000 karakter/s adatátviteli sebességet eredményez.

A programozott adatátvitel eszközeiként az illesztőegységet (interfész) megvalósító hardvert, valamint az átvitel szervezésében részt vevő utasításokat kell megemlítenünk. A 6.1. ábrán egy be-, ill. kiviteli rendszert mutatunk be. A perifériák illesztéséhez illesztő- és vezérlőáramkörök tartoznak, amelyek a processzor adat- és vezérlősínjére csatlakoznak. A megfelelő periféria kiválasztásához a címvonalakra csatlakozó címdekódoló szolgáltat kiválasztó jeleket (SEL_1, ..., SEL_n).

Az illesztőegységek a mikroprocesszortól vezérlőutasításokat és a kimeneti perifériáknak továbbítandó adatokat kapnak, az állapotkódokat és a bemeneti perifériáktól kapott adatokat pedig a mikroprocesszornak továbbítják.

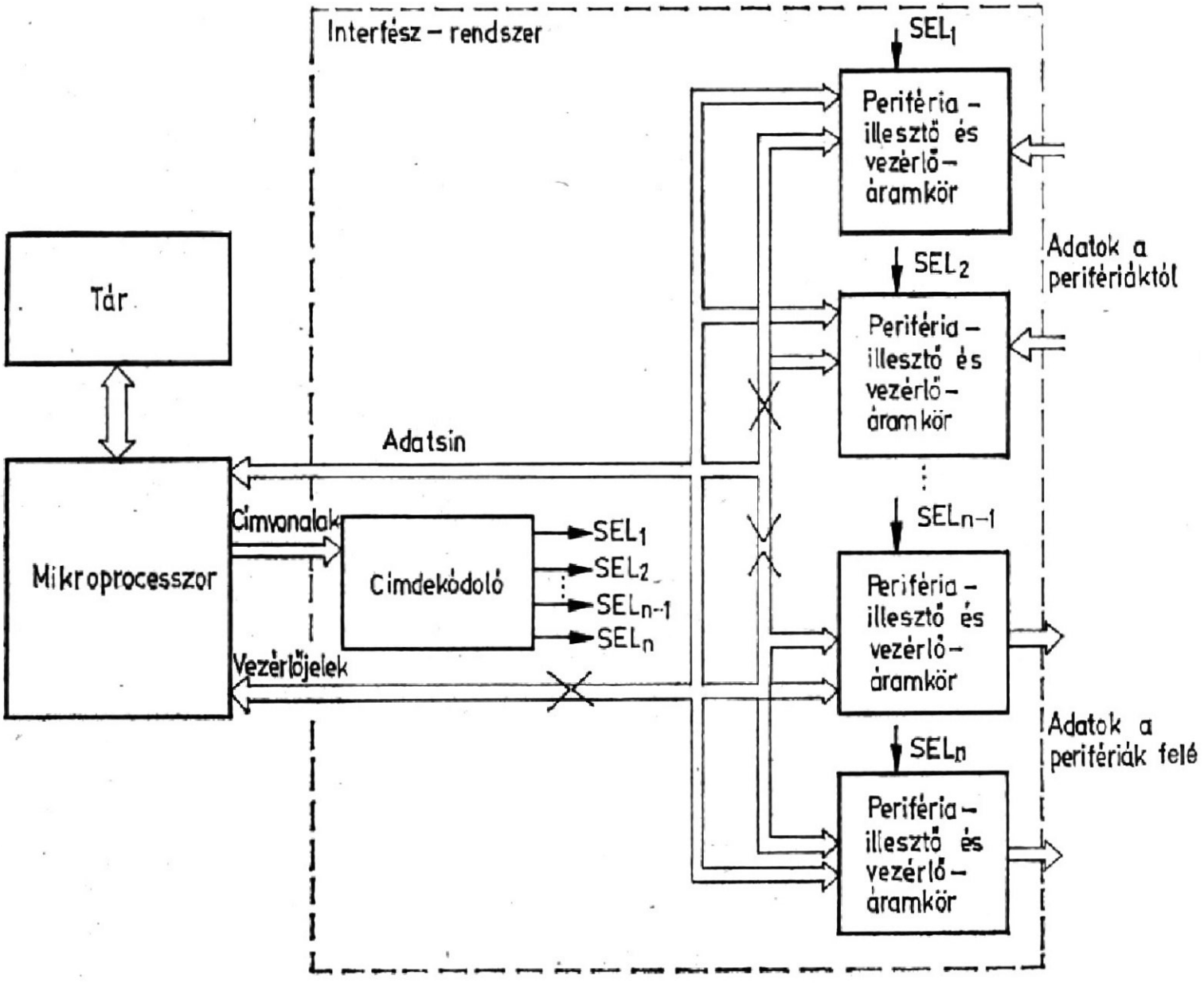
Be-, ill. kimeneti utasításai minden mikroprocesszornak vannak. Az utasításokban a perifériák illesztőegységeinek a megcímzésére címrész (címbitek) található. Az utasítások címbitjeiket tekintve két csoportra oszthatók [2].

Egyes mikroprocesszorok be-, ill. kimeneti utasításainak a műveleti kódja megegyezik a tár–processzor közötti adatátvitelek műveleti kódjával. A címrész alapján különböztethetők csak meg egymástól a tár- és a perifériaműveletek. Az ilyen rendszerekben természetesen a processzor által megcímezhető tármező egy részében tárolóáramkörök nem helyezhetők el, mert ez a rész a perifériákhoz tartozik. Ez esetben a címdekódoló áramkör feladata a perifériák kiválasztásán túl az is, hogy elkülönítse a tár- és a be-, ill. kimeneti műveleteket egymástól. Ezt a fajta utasításmegoldást „tárba ágyazott" be-, ill. kimeneti (memory-mapped input/output) műveleteknek nevezik.

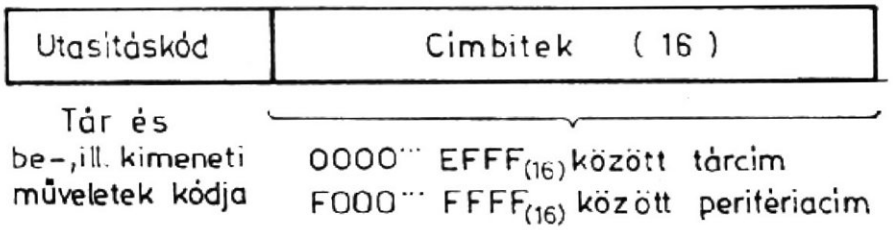
A második csoportba azok a be-, ill. kimeneti műveletek tartoznak, amelyeknek a címmezője nem terheli a tárcímterületet, mert maga az utasításkód különbözteti meg a beviteleket vagy kiviteleket a tárműveletektől. Ennek a megoldásnak az előnye, hogy a mikroprocesszor által megcímezhető tár teljes mértékben kiépíthető. Hátránya csak nagy perifériakiépítettség esetén jelentkezik. Általában a be-, ill. kimeneti utasításoknak lényegesen rövidebb címrésze van, mint a tárműveleteknek, s így kevesebb periféria megcímzésére adnak lehetőséget, mint a tárba ágyazott megoldás.

A 6.2. ábrán példát mutatunk a két utasításmegoldásra. A be-, ill. a kimeneti utasítások kódja általában különbözik egymástól, azaz a műveleti kód megadja az átviteli irányt is. Egyes mikroprocesszorok esetén a nem tárba ágyazott címzésmegoldáshoz csak egyetlen műveleti kód van a kétirányú átviteli parancs céljára, ez esetben a címmező egy részét az egyik irányhoz, másik részét pedig a másik irányhoz rendelik (pl.: Intel 8008).

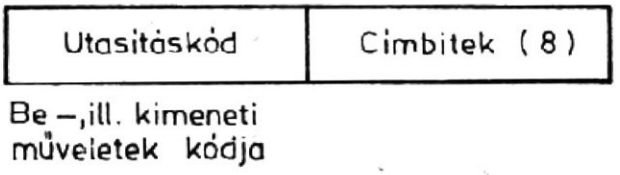
Az átvitelek végrehajtását a program vezérli. A programban való átvitelszervezést tekintve az átvitelek három típusát különböztethetjük meg: szinkron (vagy feltétel nélküli) átvitelek, aszinkron (vagy feltételhez kötött) átvitelek, és programmegszakításos átvitelek [3].



6.1. ÁBRA. Egy tipikus programozott be-, ill. kimeneti illesztőegység blokkvázlata

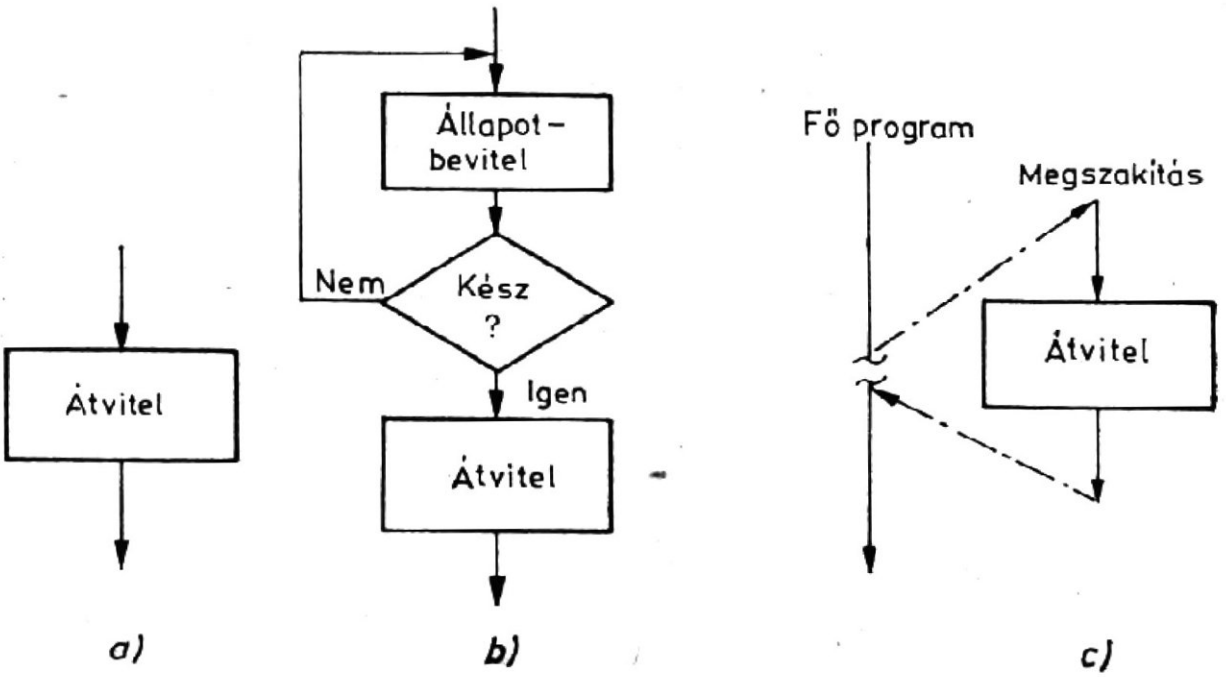


a)



b)

6.2. ÁBRA. A be- és kimeneti utasítások típusai
***a)* tárba ágyazott címzéssel, *b)* utasításkód-megkülönböztetéssel**



6.3. ÁBRA. Programozott átvitel
***a)* feltétel nélküli, *b)* feltételhez kötött, *c)* programmegszakításos [3]**

Feltétel nélküli átvitel

Ritkán alkalmazott átviteli mód, csak olyan folyamatokhoz használják, ahol az átvitel időzítése szigorúan rögzített és ismert. A perifériának az átvitel pillanatában kommunikációra kész állapotban kell lennie. A program egyszerű és „egyenes vonalú", az átvitel miatt bizonytalan idejű hurkok nem szerepelhetnek benne (6.3. ábra). A programban a be-, ill. kimeneti utasítások azokon a helyeken találhatók, amelyeken (és amelyek időpillanataiban) az átvitel szükséges. Az ilyen átvitelek meg-

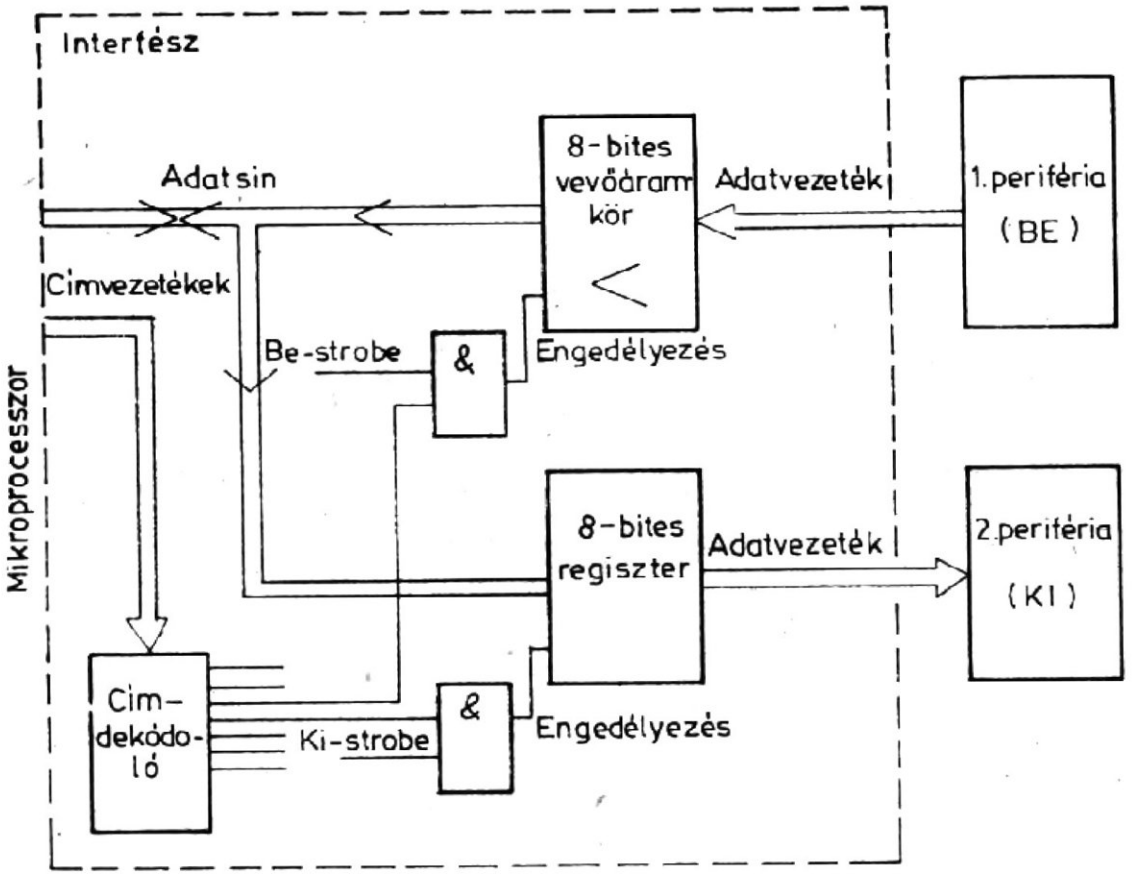
valósításaihoz általában minimális hardvert kell használni. Egy be- és egy kimeneti periféria esetén a program a következő (Intel 8080A utasításokkal):

```
⋮
IN BEVIT   ; adat bevitele
⋮
OUT KIVIT; adat kivitele
⋮
```

Az interfészáramkörök pedig ez esetben csupán kiválasztó áramkört, bemeneti adatvevő áramköröket, valamint kimeneti adatregisztert tartalmaznak (6.4. ábra). A címdekódolónak az ábrán nem használt kimeneteihez egyéb perifériavezérlő-áramkörök csatlakoztathatók.

Feltételhez kötött átvitel

Ezt az aszinkron átviteli módot igen gyakran alkalmazzák. Az átvitel végrehajtása előtt a programnak meg kell győződnie a perifériák állapotáról, arról, hogy készen áll-e a periféria az újabb kommunikáció megvalósítására. Leggyakrabban két utasítás használatos az átvitel lebonyolítására. Az első feladata, hogy az állapotinformációt a processzor lekérdezhesse. Ezután a program megvizsgálja az állapotot. Ha a periféria nincs kész, a program – egy hurokban – további ismételt lekérdezéseket hajt végre. Amint a periféria elkészül, a program rátér a második átviteli utasítás végrehajtására, amely már a kívánt be-, ill. kimeneti művelet létrejöttét eredményezi. Ennek az átviteli módnak a fő előnye, hogy szinkronizálja a periféria és a mikroszámítógép műveleteit, hátránya viszont a kész állapotra várakozással elvesző programfutási idő.



6.4. ÁBRA. A feltétel nélküli átvitelek vezérlőegysége

A periféria kiszolgálása a programban a következő (Intel 8080A utasításokkal):
Bevitel

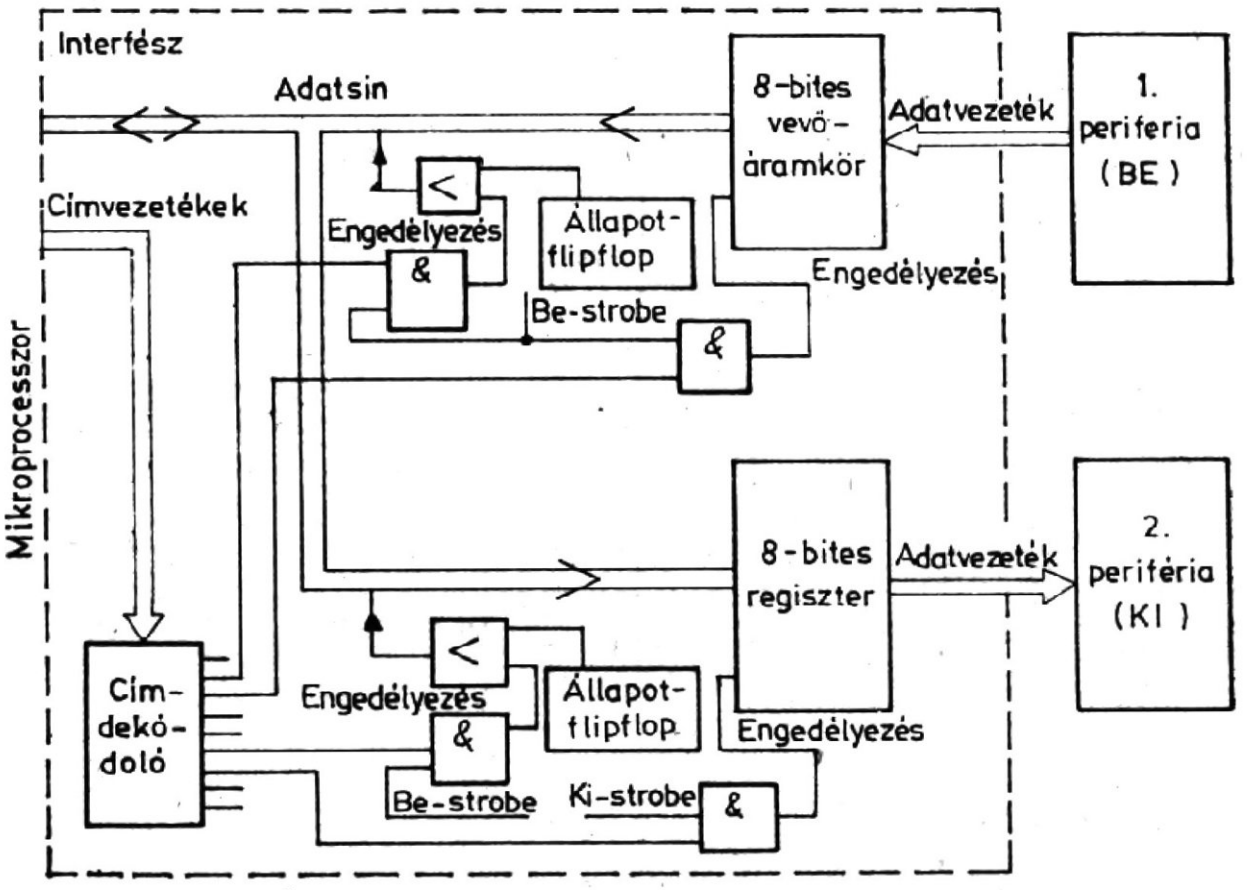
```
        ⋮
TEST1: IN STIN     ; Állapot behozatala az A-ba
       ANI 1       ; Maszk a 0. „KÉSZ" bit számára
       JZ TEST1    ; 0 = FOGLALT, ugorj TEST1-re
       IN BEVIT    ; Az adat behozatala az A-ba
       MOV M,A     ; Az adat tárba helyezése
        ⋮
```

Kivitel

```
TEST2: IN STOUT    ; Állapot behozatala az A-ba
       ANI 1       ; Maszk a 0. „KÉSZ" bit számára
       JZ TEST2    ; 0 = FOGLALT, ugorj TEST2-re
       MOV A,M     ; Adat behozatala a tárból az A-ba
       OUT KIVIT   ; Az adat kivitele az A-ból
        ⋮
```

E programban az állapotszavak beviteli periféria esetén az STIN, és kiviteli periféria esetén az STOUT címmel választhatók ki vagy kérdezhetők le. A KÉSZ/FOGLALT jelzésre a példában a 0. bitet használjuk.

Az interfész megvalósítására példát a 6.5. ábrán mutatunk. A példán az állapotadatok csupán egybitesek. Lekérdezésükre a címdekódoló külön kiválasztó jeleket generál, amelyekkel az állapotbistabilok kimenő jeleit az adatsínre kapuzza az áramkör.



6.5. ÁBRA. A feltételhez kötött átvitelek vezérlőegysége

Programmegszakítás

Az átvitelek programmegszakításos módja lehetővé teszi a mikroprocesszor idejének a jobb kihasználását. Az átvitel a program vezérlésével jön létre, de a mikroprocesszornak mégsem kell ismételten lekérdezni a periféria állapotát. A mikroprocesszor olyan, programmal megvalósított adatfeldolgozási műveleteket hajthat végre, amelyek függetlenek az átviteltől. Ezt a programot főprogramnak (néha háttérprogramnak) nevezik. Amint a periféria egy előző átviteli műveletet befejez, azaz készen áll egy újabb átvitelre, megszakításkérő jelet generál az átvitel kezdeményezéséhez. Ez azt eredményezheti, hogy a processzor abbahagyja a főprogramot, és rátér egy ún. megszakítási szubrutinra. Ez a szubrutin szolgál az átvitel programozására, tartalmazza az átviteli utasításokat. Ilyen műveletet ábrázol a korábbi, 6.3 *c* ábra. Amint a processzor a szubrutint befejezi, rögtön visszatér a főprogram folytatására.

Egy megszakításkéréses átvitel lebonyolításához tehát a következő lépések szükségesek:

1. A periféria megszakításkérő jelet generál.
2. A processzor nyugtázójelet küld, amellyel visszajelzi a megszakításkérés elfogadását. Ez csak a folyamatban lévő utasítás befejezésekor történhet meg.
3. A főprogram végrehajtását tükröző program-utasításszámláló értékét megőrzésre „kimenti" a processzor, majd rátér a megszakítási szubrutinra.
4. Szükség esetén a szubrutin által érintett processzorregiszterek, -állapotok értékét is kimenti a megszakítási szubrutin.
5. Az átvitelek végrehajtása a szubrutinban.
6. A kimentett regiszterek, állapotok értékeinek a visszaállítása, majd visszatérés a megszakított főprogramhoz.

Természetesen lehetnek a programnak olyan részei, amelyeknek a megszakítása nem kívánatos. Pl.: több periféria működtetése esetén nem kívánatos, hogy egyik periféria megszakítsa a másik megszakításkezelő szubrutinját. Ezért a mikroprocesszorok megszakítását általában programmal (utasítással) engedélyezni és tiltani lehet. Sok esetben egy megszakításkérés elfogadása (nyugtázása) után a processzor automatikusan tiltja a további megszakításkéréseket, amíg a program (pl.: a megszakítási szubrutin végén lévő utasítással) ezt a tiltást fel nem oldja.

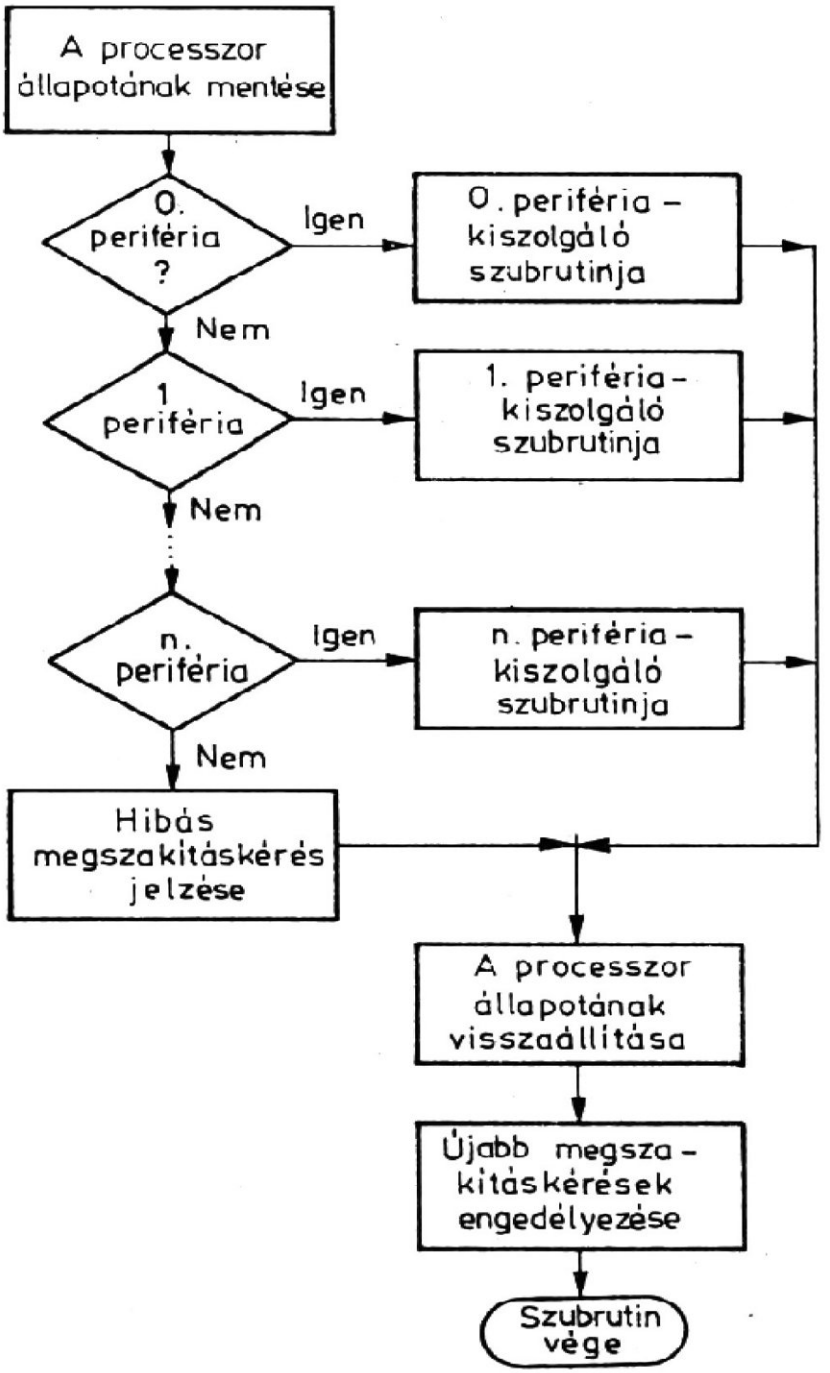
Egyidejűleg több periféria is működhet, több forrásból érkezhet megszakításkérés a processzorhoz. A források azonosítására két módszer terjedt el:

- a lekérdezéses megszakításkérés-azonosítás (polled interrupt) és
- a megszakításkérési vektorok (vectored interrupt) segítségével történő azonosítás.

A *lekérdezéses azonosítás* során a megszakításkérés elfogadása után a működő perifériákat egymás után lekérdezzük – pl.: a *foglaltság-kész* bithez hasonló *megszakításkérés* bit állapotjelüket tartalmazó állapotátvitellel – mindaddig, amíg a kérést beküldő perifériát megtaláljuk. Ezután a program végrehajtását már a megfelelő kiszolgálószubrutinnak (interrupt-service routine) adjuk át, amelyik az átvitelt végrehajtja. A lekérdezés folyamatábráját a 6.6. ábra mutatja.

A megszakításkérések lekérdezésének programja (Intel 8080A utasításokkal):

```
;A mikroprocesszor-regiszterek tartalmának tárba mentése
INTR: PUSH B    ; a B,C regiszterek mentése
      PUSH D    ; a D,E regiszterek mentése
      PUSH H    ; a H,L regiszterek mentése
      PUSH PSW  ; az A és jelzőbitek mentése
```



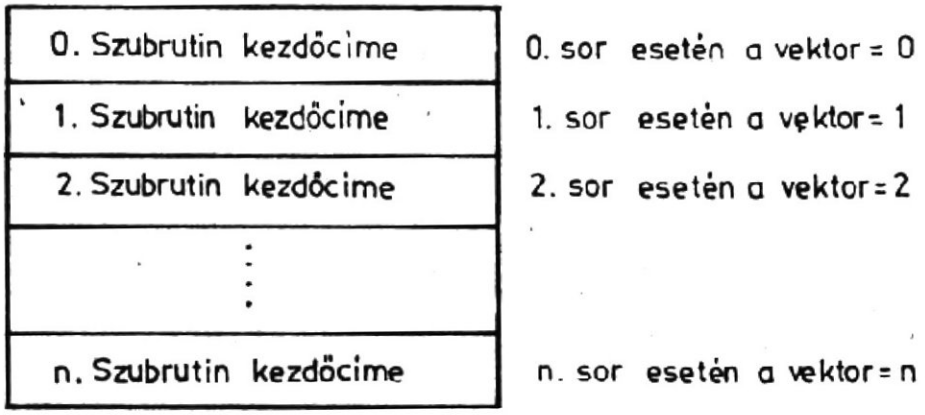
6.6. ÁBRA. A megszakításkérések azonosítása lekérdezéssel

```
;Megszakításkérések lekérdezése
        IN ST0      ; a 0. perifériaállapot adatának behozatala
        ANI 2       ; az 1. megszakításbit maszkja
        JNZ SUBR0; ha MEGSZAKÍT, ugorj a 0. szubrutinra
        IN ST1      ; az 1. perifériaállapot adata
        ANI 2
        JNZ SUBR1
        :
        IN STN      ; az n. perifériaállapot adata
        ANI 2
        JNZ SUBRN
```

```
;Hibás megszakításkérés esetén
            JMP HIBA  ; ugorj a hibajelzésre
            :
;A kiszolgáló szubrutinok
     SUBR0:
            :
            IN DATA0  ; adatátvitel bevitel a 0. perifériától
            :
            JMP ITEND; ugorj a megszakításkiszolgálás végére
    SUBR1:
            :
            IN DATA1  ; adatátvitel az 1. perifériától
            :
            JMP ITEND
    SUBRN:
            :
            OUT DATAN ; adatkivitel az n. perifériának
            :
            JMP ITEND
;A szubrutin befejező része
    ITEND: POP PSW     ; az A és a jelzőbitek visszaállítása
           POP H       ; a H,L regiszterek visszaállítása
           POP D       ; a D,E regiszterek visszaállítása
           POP B       ; a B,C regiszterek visszaállítása
           EI          ; megszakításkérés újbóli engedélyezése
           RET         ; visszatérés a főprogramra
```

A bemutatott példában a perifériák állapotszavának az 1. bitje jelezte a megszakításkérést. A megszakítási rutinban azoknak a regisztereknek a tartalmát kell elmenteni, amelyeket a szubrutin később használni kíván. Az adott példában az INTR rutinra való rátérést **CALL** szubrutinhívó utasítással (esetleg Intel 8080A-nál az egybyte-os **RST** szubrutinhívó utasítással) végezte a mikroprocesszor. Az INTR programrészre ugrással egyidőben a processzor automatikusan tiltja a további megszakításkéréseket, ezért szükséges a főprogramba való visszatérés előtt engedélyezni a későbbi megszakításkéréseket. Az engedélyező utasítás, az **EI** utasítás hatása csak a következő utasítás után érvényesül (jelen esetben a **RET** után), így biztosítható, hogy egy esetleges várakozó megszakításkérés a főprogramra való visszatérés előtt ne szakítsa meg a még nem teljesen befejezett szubrutint. A visszatérést a **RET** utasítás valósítja meg.

A *megszakításkérési vektorokkal történő azonosítás* lényege, hogy az elfogadás után a programvezérlés közvetlenül a megfelelő kiszolgálórutinra tér át. A megszakításkérést kibocsátó periféria illesztőegysége elfogadáskor egy ún. vektort, azonosító kódot küld a processzornak, amelynek az alapján a processzor eldönti, hogy melyik kiszolgálórutin tartozik a megszakításkéréshez. Egyes mikroprocesszoroknál ez a vektor egy utasításkód lehet (pl.: egy programelágazási utasítás, amely a szubrutinra való rátérést biztosítja), más mikroprocesszoroknál a vektor a szubrutin kezdőutasításának a közvetett vagy közvetlen címét vagy a címnek egy részét adja. Közvetett cím esetében a megszakítási rutinok kezdőcímei a tárban egy táblázatban találhatók, s a vektor a táblázat megfelelő sorára mutat rá. A processzor azután ennek a táblázatsornak a kiolvasásával állapítja meg a tényleges kezdőcímet (6.7. ábra). A vektort általában az adatsín közvetítésével kapja meg a processzor.



6.7. ÁBRA. A megszakításkérések azonosítása táblázatra mutató vektorral

Az alábbiakban példaként az Intel 8080A utasításainak a segítségével egy kiszolgálórutint mutatunk be arra az esetre, amikor a szubrutinra való rátérés az azonosítás során beküldött szubrutinhívó utasításkóddal történt (a 0. periféria programja), mely a programvezérlést az INT0 címre adja át.

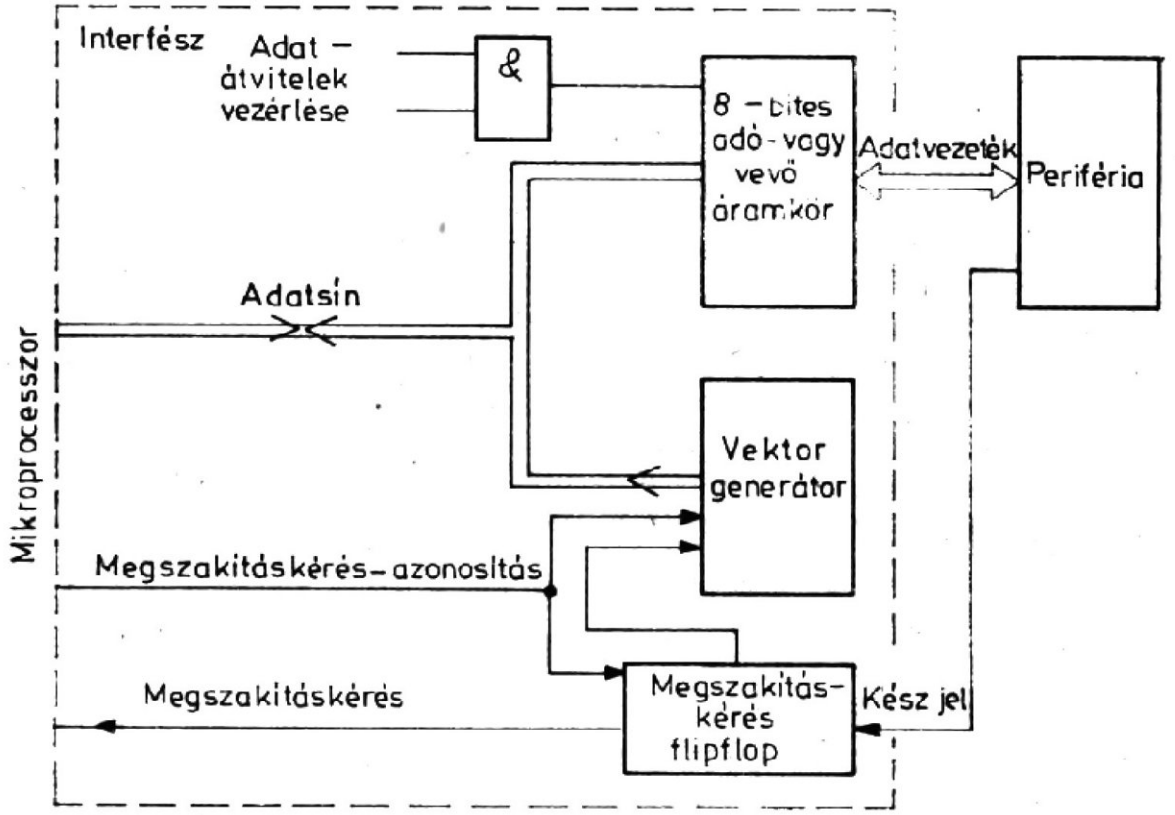
```
INT0: PUSH B      ; a B,C regiszterek mentése
      PUSH D      ; a D,E regiszterek mentése
      PUSH H      ; a H,L regiszterek mentése
      PUSH PSW    ; az A és a jelzőbitek mentése
      :
      IN DATA0    ; a 0. periféria adatának behívása
      :
      POP PSW     ; az A és a jelzőbitek visszaállítása
      POP H       ; a H,L regiszterek visszaállítása
      POP D       ; a D,E regiszterek visszaállítása
      POP B       ; a B,C regiszterek visszaállítása
      EI          ; a megszakításkérés engedélyezése
      RET         ; visszatérés a főprogramba
```

A vektoros megszakításkérés-azonosítás áramköri megvalósítására, egy perifériával, a 6.8. ábra mutat példát.

Több egyidejű megszakításkérés kapcsán felmerül a prioritás kérdése. A megszakítások prioritásával kapcsolatosan prioritási osztályokról és prioritási szintekről beszélhetünk. A magasabb prioritási osztályokba tartozó perifériák megszakításkérései – hacsak külön a program nem tiltja – megszakíthatják egy alacsonyabb osztályú periféria kiszolgálórutinját.

Gyakori eset, hogy egy mikroprocesszorban két megszakítási osztályt alakítanak ki úgy, hogy a magasabb osztály megszakításkéréseit programmal tiltani nem lehet (non maskable interrupt). Ezt a megszakításkérést a processzor mindig, minden utasítás végén elfogadja. Ide köthetők azok a megszakításkérő áramkörök, amelyek kiszolgálása nem tűr halasztást (pl.: hálózatkimaradás esetén elvégzendő teendők). Az alacsonyabb osztály megszakításkérései már programmal tilthatók, ill. a kiszolgálórutinjaik indításakor a további kéréseket automatikusan tiltja a processzor belső logikája. A megszakításkérések tiltása is általában két szinten lehetséges:

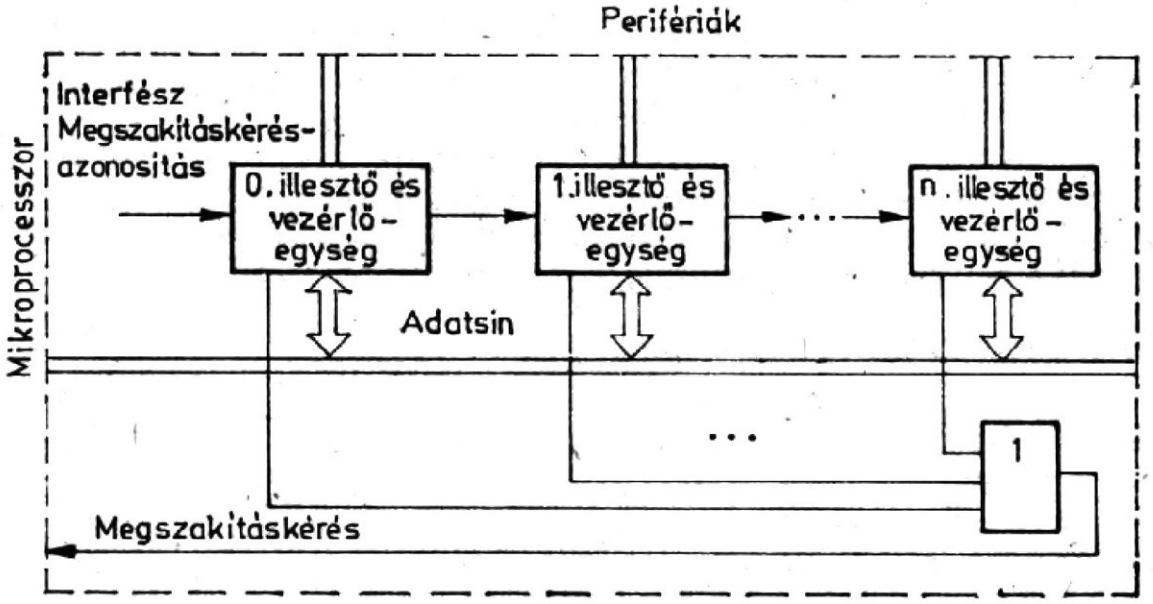
1. A processzor szintjén, ez esetben a perifériaillesztő- és vezérlőáramkörök generálhatnak ugyan megszakításkérő jeleket, a processzor azonban ezeket a jeleket figyelmen kívül hagyja.
2. Az illesztőegységek szintjén, ez esetben a processzor parancs kiküldésével az



6.8. ÁBRA. A megszakításos adatátvitel vezérlőegysége és az azonosítási vektort generáló áramkör

illesztőegység megszakításkérő jelet generáló áramkörét tilthatja le, ill. engedélyezheti.

A prioritásszintek fogalma a megszakításkérés azonosításának a pillanatában jelentkező több egyidejű megszakításkéréssel kapcsolatos. Ilyen esetben először a magasabb prioritási szintű perifériát kiszolgáló szubrutin végrehajtását látja el a processzor. A programmal lekérdezett megszakításkérések (program-polled interrupt) prioritási szintjét a program lekérdezési sorrendje adja meg, hiszen arra a szubrutinra tér rá először a program, amelyikhez tartozó periféria a lekérdezésre válaszolva legelőször jelzi, hogy tőle származik a megszakításkérő jel. A megszakításvektoros rendszerekben rendszerint a megszakítási lánc (daisy-chain) megoldást alkalmazzák. Ennek lényegét a 6.9. ábra szemlélteti. Azonosítás során a processzor



6.9. ÁBRA. A felfűzött prioritási lánc vázlata

felől érkező *megszakításkérés nyugtázása* (interrupt acknowledge) jel a legmagasabb prioritásszintű egység bemenetére jut. Ez az egység csak akkor engedi tovább a jelet a következő egység felé, ha ő maga nem kér megszakítást. Így a nyugtázójel a legnagyobb prioritásszintű megszakításkérő egységig jut csak el. Az az egység, amelyik bemenetén megkapja a nyugtázójelet és megszakítást is kér, az adatsínre kapuzhatja az őt azonosító vektort.

Az eddig elmondott megszakításkérés-kezelő és -azonosító eljárások nagyon sokféle kombinációja és változata fordul elő a különbözá típusú mikroprocesszorok és mikroszámítógépek esetén. Az egyik érdekes s az alkalmazások adott körében igen jól használható megoldás lényege a következő. A mikroprocesszornak a kivezetései között nemcsak egy, hanem több megszakításkérési bemenet van. Ezek bármelyikére egy-egy periféria megszakításkérő jele köthető. A megszakításkéréseket tulajdonképpen a bemeneti pontra kötés azonosítja a processzor felé. A programoknak már a processzor belső áramkörei adnak a kérés azonosításához vektorokat, és ugyancsak a processzor belső kialakítása gondoskodik a kérések közötti prioritási szintek kialakításáról.

Hardvereszközök

Az interfészegységeknek az integrált áramkörökben való megvalósítása, s ezzel az interfész hardverméreteinek a csökkentése, megbízhatóságának a növelése a mikroprocesszorok kidolgozásával párhuzamosan halad. Ahogyan a mikroszámítógépek hardvertervezését egyszerűsíti és megkönnyíti a programvezérelt LSI processzorok alkalmazása, ugyanúgy egyszerűsítik az interfészrendszerek létrehozását az LSI és a VLSI interfészáramkörök. Ezeknek az áramköröknek a kidolgozását és megjelenését nyomon követve, kettős irányzat figyelhető meg. Egyrészt az, hogy egyre több alkalmazási feladathoz és területhez vásárolhatók nagy bonyolultságú kész integrált áramkörök. Ez a fejlődési irány természetesen nagyon megnöveli a kapható különböző áramköri típusok számát. Másrészt azonban megjelentek olyan eszközök, amelyek a mikroprocesszoros rendszer bekapcsolása után programmal ún. vezérlővagy a működési módot meghatározó utasításokkal „konfigurálhatók". Ez azt jelenti, hogy az áramkör lehetséges sokféle működési változata közül valamelyik az adott alkalmazásban kiválasztható, s ennek megfelelően használható az integrált áramkör. Ez azt eredményezi, hogy a felhasználó ugyanazt az LSI típust különböző célokra használhatja (pl.: feltétel nélküli bemeneti áramkörként, megszakításkéréses kimeneti áramkörként, párhuzamos adatátvitelre, de kétállapotú jelek bitenkénti átvitelére is stb.). Az ilyen sokoldalúan és sokcélúan felhasználható integrált áramkörök viszont jelentősen csökkenthetik egy-egy alkalmazásban a felhasznált alkatelemek típusválasztékát, tovább könnyítve ezzel a kidolgozást.

Természetesen az interfész hardver alkatelemei között a sokkal egyszerűbb integrált áramkörökről is kell szólnunk, de ismertetjük azoknak az „univerzálisan felhasználható" elemeknek a típusait is, amelyek belső felépítésüket tekintve bonyolultabbak, mint egy-egy mikroprocesszor.

A címdekódoló és kiválasztó áramkörök

Ezeknek az áramköröknek a feladata kiválasztójelek generálása a perifériavezérlők számára a címvonalakon érkező címkombinációk alapján. A címdekódolók között kis integráltságú logikai kapuáramkörök és közepes integráltságú kódkonverterek, kóddekódolók találhatók. Bonyolultabb rendszerekben, ahol sok kiválasztójelet kell generálni, előnyös lehet a PROM és FPLA áramkörökkel megvalósított címdekódolók alkalmazása.

Adatadó, adatvevő áramkörök, multiplexerek, demultiplexerek

Az LSI mikroprocesszoroknál igen elterjedten használják a háromállapotú jeleket. A legtöbb esetben ilyen háromállapotú az adatsín és a címvezetékek sínje, de a vezérlőjelek közül is több lehet háromállapotú. A multiplexerek legegyszerűbben megvalósítható formái éppen a háromállapotú kapuáramkörökkel, vonalmeghajtó áramkörökkel kialakított változatok. A demultiplexerek a jelek vonali adatcím alapján történő sok vonalra való szétosztásának hasznos eszközei. A kétirányú sínekre, adatsínekre való csatlakoztatás eszközei a kétirányú vonali meghajtó-, ill. vevőáramkörök (bi-directional bus driver/receiver vagy transreceiver).

A MOS technológiával megvalósított LSI és VLSI áramkörök esetén követelmény a csatlakozó áramkörök bemeneti impedanciájára vonatkozó előírások figyelembevétele. Nagyobb rendszerekben meglehetősen sok áramköri alkatelem különböző sínekre való csatlakoztatására van szükség. A sínek terhelhetősége pedig igen korlátozott, ezért nagyon fontos eszközök a kis bemenő áramot igénylő vonali fogadó- és meghajtóáramkörök.

Regiszterek

Az adó-, ill. vevőáramkörök a párhuzamos adatoknak csak a pillanatnyi értékeit képesek bemenetükről a kimenetükre továbbítani. Gyakran azonban – szinkronizálási okok miatt – az adat mintavételezése, majd további vonalra való kiadása is szükségessé válik. Ez esetekben regisztereket vagy több bites bistabil integrált áramköröket használhatunk. A regisztereknek nagyon sokféle változatát kínálják a félvezető alkatelemgyártók. Mikroprocesszorok mellett előnyösen alkalmazhatók a háromállapotú kimenetü áramkörök. Sok alkalmazásban további előnyt nyújthatnak azok a regiszterek, amelyekben az adatokat tároló flip-flopokon kívül egy további, FELTÖLTÖTT/ÜRES állapot jelzésére szolgáló flip-flop is van, amely automatikusan a megfelelő helyzetbe billen az adatregiszterek beírásakor, valamint kiolvasásakor (a háromállapotú kimenetek aktivizálásakor) [4].

Szintátalakítók és vonali meghajtó- és fogadóáramkörök

A különböző technológiájú alkatelemek, valamint a rendszerek különböző részeinél alkalmazott eltérő jelszintszabványok miatt néha szükséges olyan áramköröknek a rendszerbe építése, amelyek a logikai szinteket átalakítják. Az ilyen átalakítók között a leggyakoribbak a TTL–MOS jelszintek átalakítói. Ezek olyan integrált áramkörök, amelyek a TTL oldalon 0...5 V, míg a MOS oldalon 0...30 V közötti jelszintekkel dolgoznak (pl.: Signetics 8T18 és 8T90).

Más típusú integrált áramkörökkel a logikai szintek és a kommunikáció különböző szabványaiban előírt jelszintek közötti konverzió valósítható meg. Példaként a soros adatátvitel CCITT V24 szabványát említhetjük, a +12 V-os és −12 V-os jelszintjeivel (a fogadó- és meghajtóáramkörökre példa: Signetics 8T15 és 8T16).

Időzítő- és késleltetőáramkörök

Ilyen célra leggyakrabban astabil és monostabil áramköröket használunk. A mikroprocesszorok órajel-generátora gyakran rezgőkristállyal stabilizált astabil áramkör.

Érdekes lehetőségeket nyújtanak a bemeneti feszültséggel szabályozható időzítőkörök. Ezek mind astabil, mind pedig monostabil funkciók ellátására alkalmasak. Közöttük található olyan típus is, amellyel μs-tól több órás időzítésig terjedő bármilyen késleltetési érték elérhető (pl.: Signetics 555 timer).

Eddig kis és közepes bonyolultságú integrált áramkörökkel felépített eszközöket

ismertettünk. Tekintve, hogy használatuk és belső felépítésük meglehetősen egyszerű és a gyártók katalógusaiban [4] ... [6] részletes ismertetők találhatók, itt bővebben nem tárgyaljuk ezeket az áramköröket.

6.2. PROGRAMOZHATÓ LSI INTERFÉSZELEMEK

A programozható LSI interfész alkatelemek a mikroprocesszoros rendszerek sokoldalúan felhasználható eszközei. A programozható kifejezés itt nem programvezérlést jelent, tehát nem úgy működnek ezek az alkatelemek, hogy utasítások sorozatát hajtják végre. A programozható eszköz egy olyan interfészelem, amely több feladat ellátására alkalmassá tehető, a feladathoz adaptálható. Ezt a feladathoz való adaptálást a konkrét alkalmazásokban vezérlő, vagy átviteli módot meghatározó parancsokkal érhetjük el. Általában egy rendszeren belül működés közben az átviteli módot nem változtatja meg a program. A módmeghatározó parancs a rendszer inicializálásának a része, tehát a tápfeszültség bekapcsolása után elinduló program az elején, az inicializálási szakaszban tartalmazza ezeket a „konfigurálási" parancsokat. A működés során azután már állandó marad az átviteli mód.

Programozható perifériaillesztő

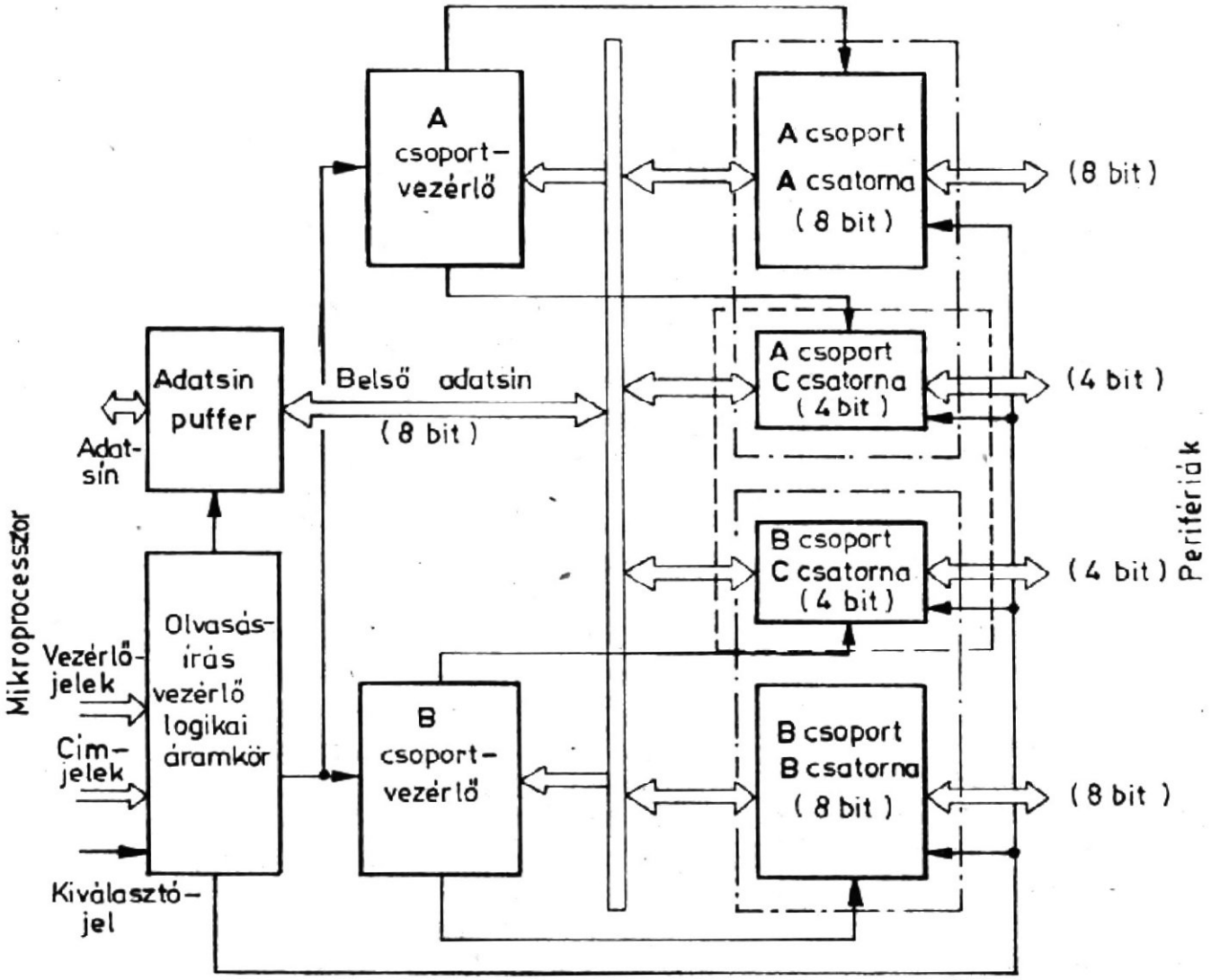
A programozható perifériaillesztő (PPI: Programmable Peripheral Interface) áramkör feladata mikroprocesszoros rendszerekben a párhuzamos be-, ill. kimeneti interfészek perifériaillesztési és vezérlési funkcióinak az ellátása, valamint a kimeneti kétállapotú jeleknek a bitenkénti kezelése. A PPI áramkörök egyik irányban közvetlenül annak a mikroprocesszornak a vezérlő- és adatsínjére köthetők, amelynek az LSI áramkörkészletébe tartoznak. Ez nagyon leegyszerűsíti a mikroprocesszorhoz való csatlakoztatásukat, az esetek nagy részében csak címkiválasztó és esetleg megszakításkérések kezelésére szolgáló egyszerű áramkörökkel kell kiegészítenünk a PPI-ken alapuló interfészrendszereket.

Egy tipikus PPI [7] belső felépítésének a vázlatát mutatja a 6.10. ábra. A bemutatott PPI alkalmas mind a feltétel nélküli (szinkron), mind a feltételhez kötött (aszinkron) átvitelek lebonyolítására, de alkalmas megszakításkéréses átviteli mód szerinti működésre is. A PPI-nek a perifériához csatlakozó pontjain a ki- és a beviteli, valamint a kétirányú 8-bites párhuzamos adatátvitel egyaránt megvalósítható. De a PPI programozható úgy is, hogy a perifériákhoz csatlakozó pontjain az adatok értelmezése bitenként történhessen, és egy kimeneti csatornán belül bármelyik bit értéke külön-külön is programozható legyen.

A mikroprocesszorhoz a PPI az adatsínnel, valamint az írás/olvasást vezérlő jelvezetékekkel, a csatornák megcímzésére és a parancsátviteleknek az adatátvitelektől való megkülönböztetésére szolgáló címvezetékekkel csatlakozik. A PPI működtetésére kiválasztójel szolgál. A mikroprocesszor felőli adat- és parancsátvitelekhez tartozó be-, ill. kimeneti utasításokat és vezérlőjeleket foglalja össze a 6.1. táblázat.

Parancskivitellel a periféria felé csatlakozó 24 kivezetés két vagy három csoporba osztható. A „konfigurálási" parancs kódját a kiviteli utasítás adatrésze tartalmazza. Ha feltétel nélküli adatátvitelre van szükség, a 24 bitnyi kivezetés 3×8 bites csatornává alakítható (**A**, **B** és **C** csatornák), sőt a **C** csatorna további 2×4 bites „félcsatornára" (C_F és C_A) bontható. Bármelyik csatorna, beleértve a 4-bites „félcsatornákat" is, programozható beviteli, vagy kiviteli irányba. Minden csatorna mind a két átviteli irányban adatregisztereket tartalmaz.

Feltételhez kötött adatátviteli, és programmegszakításos átviteli eljárások esetén a perifériákkal való kapcsolattartáshoz nem elegendő az adatsínnel való össze-



6.10. ÁBRA. A programozható perifériaillesztő áramkör belső felépítése [7]

6.1. TÁBLÁZAT. A PPI működtetése

Címjelek		Vezérlőjelek		Kiválasztó-jel	Átviteli utasítás
A1	A0	olvasás	írás		
0	0	igen	nem	igen	A csatorna bevitele
0	1	igen	nem	igen	B csatorna bevitele
1	0	igen	nem	igen	C csatorna bevitele
1	1	igen	nem	igen	Illegális beviteli utasítás
0	0	nem	igen	igen	Kivitel A csatornára
0	1	nem	igen	igen	Kivitel B csatornára
1	0	nem	igen	igen	Kivitel C csatornára
1	1	nem	igen	igen	Parancskivitel
x	x	x	x	nem	Nincs kiválasztva, adatsínkimenetek nagy impedancián

x: közömbös logikai érték

köttetés, az átvitelhez ún. kézfogásjelek (handshake) is tartoznak. Ezen túlmenően az átvitel végén megszakításkérő jel generálása is szükséges. 24 kivezetés felhasználásával tehát nem alakítható ki ez esetben három átviteli csatorna. Ilyenkor a két csoport (A és B) alkotja a két perifériavezérlőt, és a C csatornához tartozó vezetékek jeleiből képezhető a két átvitelvezérlő, valamint a megszakításkérő jel. Ez utóbbi már nem a perifériához, hanem a megszakításkezelő áramkörhöz csatlakozik. A C csatornából megmaradó csoportonkénti egy bit továbbra is úgy programozható, mint a C csatorna része. Természetesen kombinálható is a többféle átvitelváltozat: az A csoportot az adott rendszer feltételhez kötött, míg a B csatornát és a C csatorna megmaradó részét feltétel nélküli átviteli módban használhatja.

A feltételhez kötött átvitelek lekérdezéséhez a processzor a C csatornát megcímezve bekérheti az átvitel lebonyolítását végző két kézfogásjelet, s mint állapotjelet megvizsgálhatja.

A periféria és az interfész kétirányú adatátvitelének egy adatsínen való lebonyolításához 2×2 kézfogásjel tartozik. Ezen túlmenően megszakításkérésre is szükség van. Így csak az egyik csoport (az A csoport) működhet kétirányú party line adatsínnel, hiszen működéséhez a C csatornához tartozó öt kivezetés szükséges. A másik csoport ez esetben a C csatorna fennmaradó három jelét használhatja feltételhez kötött átvitelek lebonyolítására, de működhet feltétel nélküli átviteli üzemmódban is. Ez utóbbinál a C csatorna három bitje önállóan programozható. A kétirányú adatátviteli interfész állapotának a lekérdezése a C csatorna címével kiadott beviteli utasítással történik, hasonlóan az egyirányú átvitelnél elmondottakhoz.

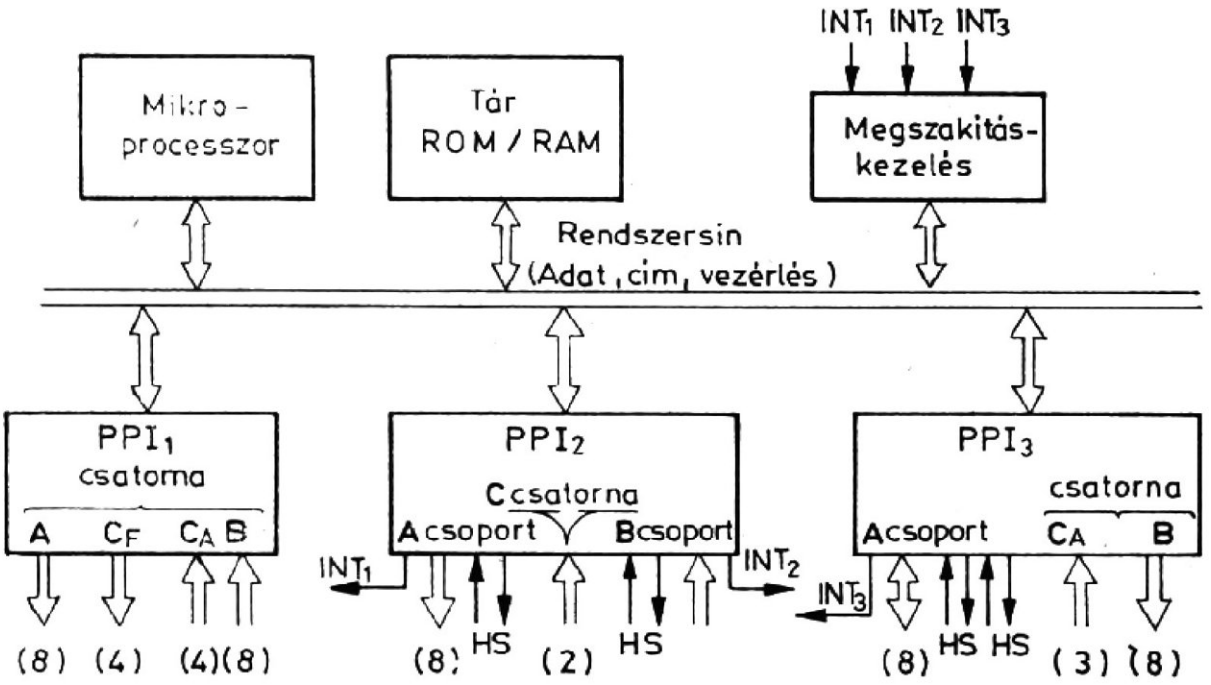
A C csatorna kimeneti pontjairól tehát vezérlőjelek kerülnek az A és B csoporthoz. Sokféle kombináció lehetséges a PPI konfigurálására. Amennyiben az adott konfigurációban a C csatorna összes bitjét nem használjuk vezérlőjelként, a fennmaradó bitek feltétel nélküli átvitel céljára „konfigurálhatók" és használhatók. A C csatorna kiviteli bitjeit bitek szerint is programozhatjuk (ez esetben az utasítás alakja parancsutasításnak felel meg, s a parancskódban található a bitkezelési funkció paramétere).

Megszakításkéréses üzemmódban a PPI belső logikája lehetőséget ad a PPI szintjén való megszakításkérés engedélyezésére és tiltására. Az engedélyező és tiltó parancsok utasításalakja a bitenkénti programozással megegyezik.

A 6.2. táblázatban röviden összefoglaltuk a PPI különböző üzemmódjait, a 6.11. ábrán pedig egy példa vázlatosan mutatja a PPI-k használatát. A PPI_1 minden csatornája feltétel nélküli átviteli módban működik (A és C_F kimeneti, B és C_A bemeneti csatornák). A PPI_2 áramkör A csoportja „kézfogásos" kimenetet, a B

6.2. TÁBLÁZAT. A PPI-vel megvalósítható átvitelek összefoglalása

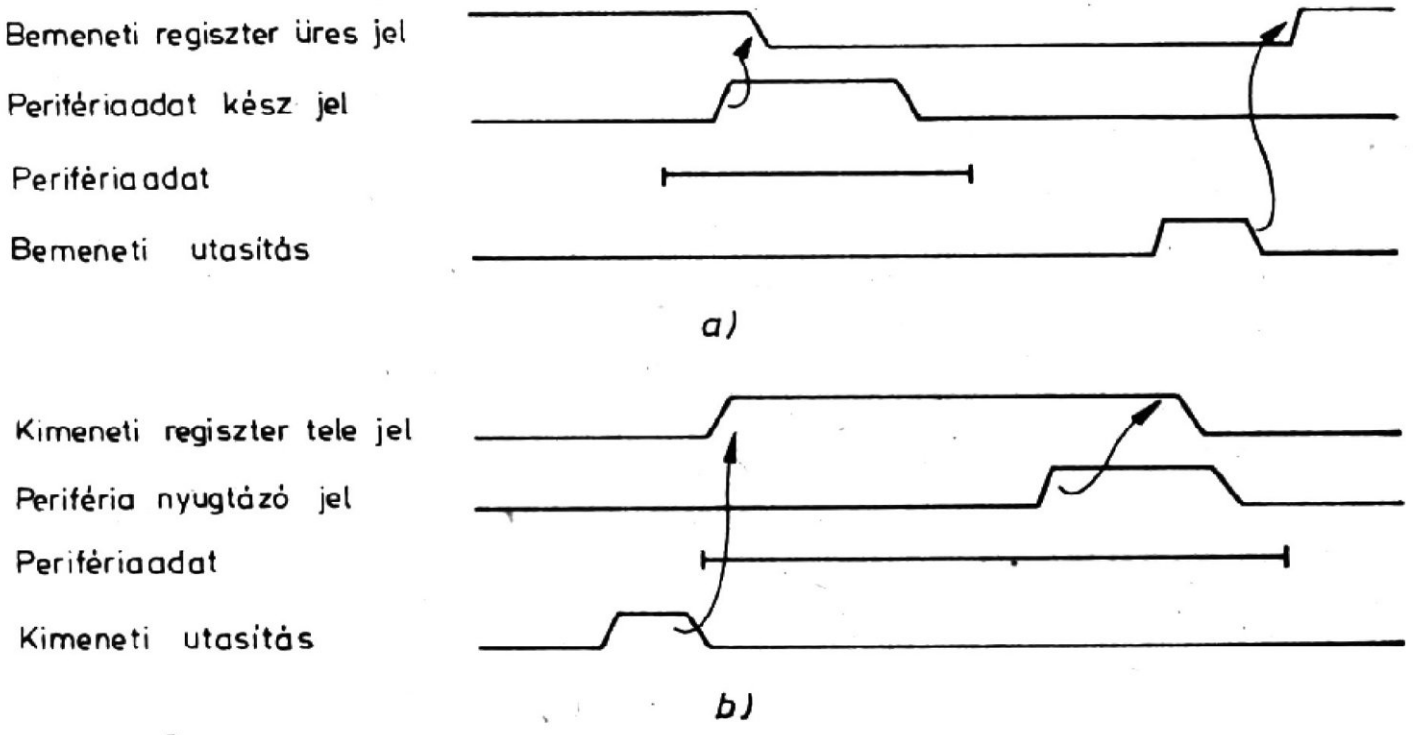
Feltétel nélküli átvitel		Feltételes vagy megszakításos átvitel			
		Egyirányú adatsín		Kétirányú adatsín	
A csatorna	bármelyik csatorna kiviteli vagy beviteli; C csatorna bitenként is programozható	A csoport	bármelyik csoport kiviteli vagy beviteli	A csoport	kétirányú
B csatorna		B csoport		B csoport	kiviteli vagy beviteli
C_A csatorna					
C_F csatorna					



6.11. ÁBRA. Példa a programozható perifériaillesztők használatára

csoportja pedig kézfogásos bemenetet valósít meg. A kézfogás jeleit HS betűkkel, a megszakításkérés jelét INT-tel jelöltük. A PPI_2 a C csatorna két szabad bitjét feltétel nélküli bevitel céljára használja. A PPI_3 áramkör A csoportja kétirányú adatátvitelt végez, B csatornáját és a C_A csatorna három bitjét pedig feltétel nélküli kivitel (B), ill. bevitel (C_A) céljaira használja. A kézfogásos adatátvitel idődiagramja a 6.12. ábrán látható.

A PPI jelen leírásában egy tipikus interfészelemet mutattunk be. A különböző mikroprocesszorok hasonló célú elemeinek a felépítése némileg eltérhet ettől (pl.: a kivezetések mind kézfogásos, mind adatjelként való használatában, a megszakításkérés kezelésében stb.).



6.12. ÁBRA. A PPI és a periféria közötti átvitel vezérlőjelei *a)* bevitel, *b)* kivitel esetén

A programozható soros átviteli illesztőegység

A mikroprocesszorok és a külvilág közötti kapcsolatnak egyik igen fontos módszere a soros adatátvitel. A soros adatátvitelt megvalósító LSI eszközök közé az univerzális szinkron–aszinkron adó–vevő áramkörök (USART: Universal Synchron Asynchron Receiver Transmitter) és az univerzális aszinkron adó–vevő áramkörök (UART: Universal Asynchron Receiver Transmitter) tartoznak. Az USART és UART a soros–párhuzamos adatátalakításon és visszaalakításon kívül bizonyos modemvezérlési funkciókat is elláthat. A mikroszámítógép és az USART–UART áramkör kapcsolatának a vázlatát a 6.13. ábra mutatja.

Az alábbiakban egy USART LSI áramkört [7] ismertetünk röviden. Belső felépítését a 6.14. ábrán láthatjuk. A mikroprocesszorhoz annak vezérlőjeleivel és az adatsínnel kapcsolódik. A működtetéshez kiválasztójel és adat- ill. parancsjel tartozik (ez utóbbit rendszerint a kiválasztójelhez hasonlóan az átviteli utasítás címrésze adja meg). Az USART az adatokat a mikroprocesszortól párhuzamos formátumban kapja, majd ezeket soros adattá alakítja. Az adással egyidőben, de külön is fogadhat soros adatokat a vonali bemeneten. A beérkező adatokat párhuzamos formátummá konvertálja.

Az USART a mikroprocesszorral négyféle utasítás hatására létesít adatkapcsolatot. Ezeket a vezérlőjelekkel együtt a 6.3. táblázatban foglaltuk össze. A bekapcsolást

6.3. TÁBLÁZAT. Az USART-ra vonatkozó átviteli utasítások [7]

Vezérlőjelek				Utasítás típusa
Adat/ parancs	Olvasás	Írás	Kiválasztás	
parancs	nem	igen	igen	Parancs kivitele
parancs	igen	nem	igen	Állapot lekérdezése (bevitele)
adat	nem	igen	igen	Adatkivitel
adat	igen	nem	igen	Adatbevitel
x	x	x	nem	Nincs kiválasztva, adatsín kimenetein nagy impedancia

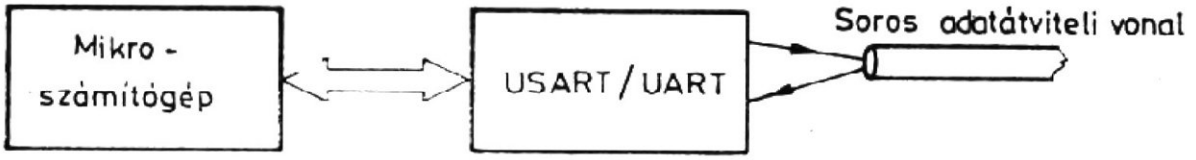
x: közömbös logikai érték

(nullázást) követően az első parancsutasítás az átviteli módot adja meg. Az *átviteli mód* utasítás paramétere az alábbiakat határozhatja meg:

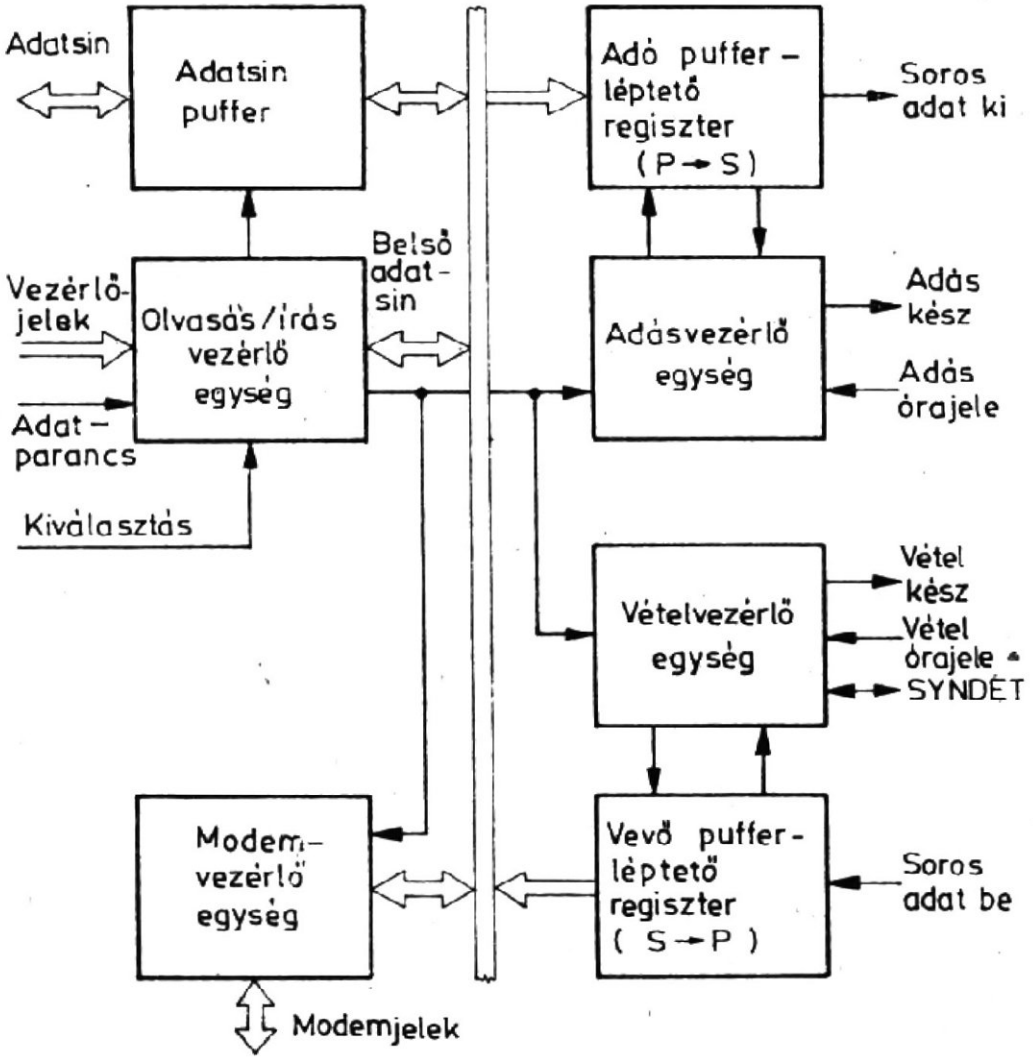
- a szinkron vagy aszinkron adatátvitelt;
- az átviteli sebességet és az adás/vétel órajel arányát (kívánságra az órajeleket az USART alacsonyabb frekvenciára leosztja);
- a paritás-ellenőrzés szükségességét, ellenőrzés esetén a paritás típusát (páros vagy páratlan);
- aszinkron adatátvitel esetén a STOP bitek kívánt számát (1, 1 1/2 vagy 2);
- szinkron adatátvitelnél
  - a külső vagy belső SYNDET jelet,
  - az egykarakteres vagy kétkarakteres szinkronizálást.

Az átviteli mód megadását követő parancskivitelek értelmezése:

- szinkron üzemmódban az első kettő a szinkronkarakterek értékét adja meg (aszinkron üzemmódban erre nincs szükség);



6.13. ÁBRA. Mikroszámítógép csatlakoztatása a soros adatátviteli vonalra USART/UART áramkörön keresztül [8]



6.14. ÁBRA. Az USART funkcionális blokkvázlata [7]

■ a további parancskivitelek tartalma a modemvezérlő jeleknek a megadására, az adó–vevő körök tiltására, a hibajelzések törlésére szolgál, ill. szinkron üzemmódban „keresés" parancs is adható, valamint jelezhető, hogy a következő paparancskivitel újból átviteli mód megadására szolgál.

Az USART-tal természetesen csak feltételhez kötött vagy megszakításkéréses átvitelek megvalósítása lehetséges, a programnak a vevőkörrel való hosszú idejű pontos szinkronizálása csak így oldható meg. Az állapotbevitelek (parancsbevitelek) segítségével a processzor lekérdezheti az adó és a vevő foglalt állapotát, valamint a vevő által vett üzenettel kapcsolatos esetleges hibajelzéseket (paritáshiba, adathosszhiba, és adatelveszés). Megszakításkérésként pedig, az ADÁS KÉSZ és VÉTEL KÉSZ jelek szolgálhatnak.

Az UART–USART LSI elemeknek a PPI-khez hasonlóan sok változatát gyártják a félvezetőgyártók, s más-más változata használható a különböző típusú mikroprocesszorokhoz. Ismeretesek többé-kevésbé processzorfüggetlen típusok, de kap-

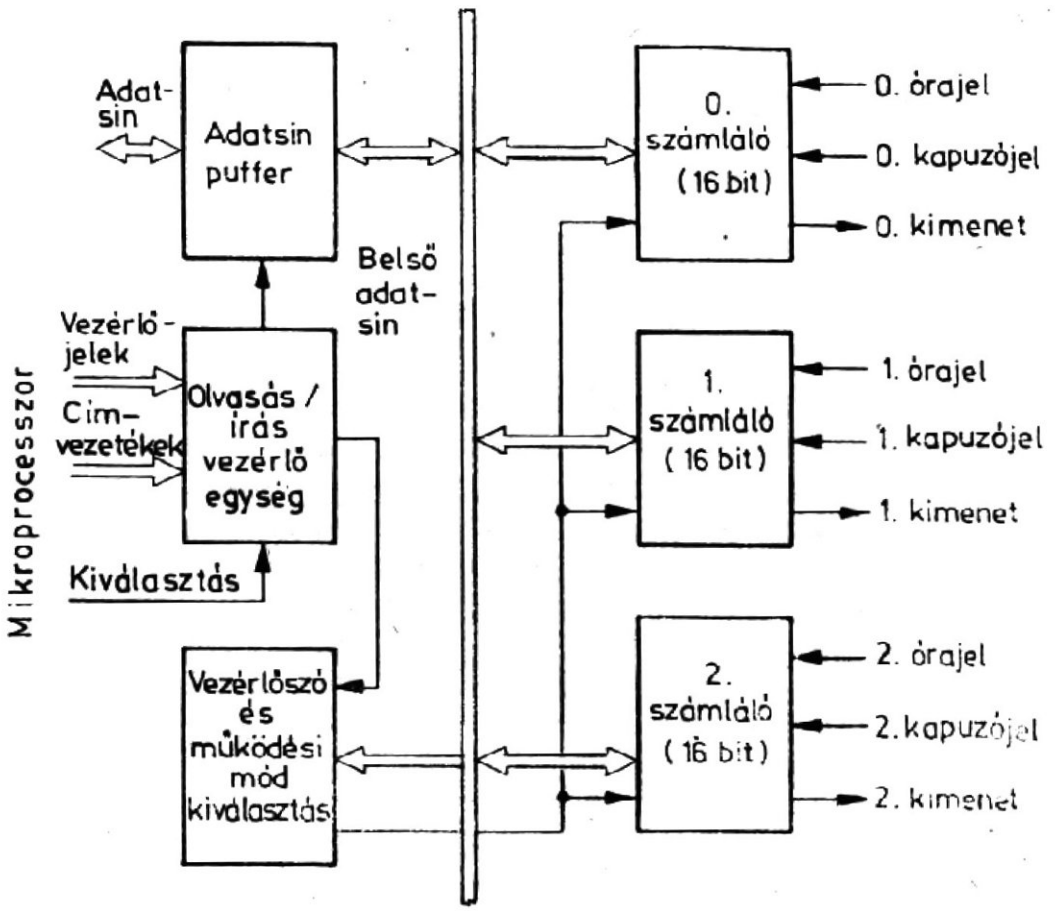
hatók olyanok is, amelyek adott mikroprocesszor-családhoz kialakított megszakításkezelő logikai áramkört tartalmaznak. A felépítési alapelvük azonban ezeknek is hasonló a fent ismertetett USART-hoz.

Programozható időzítő–számláló

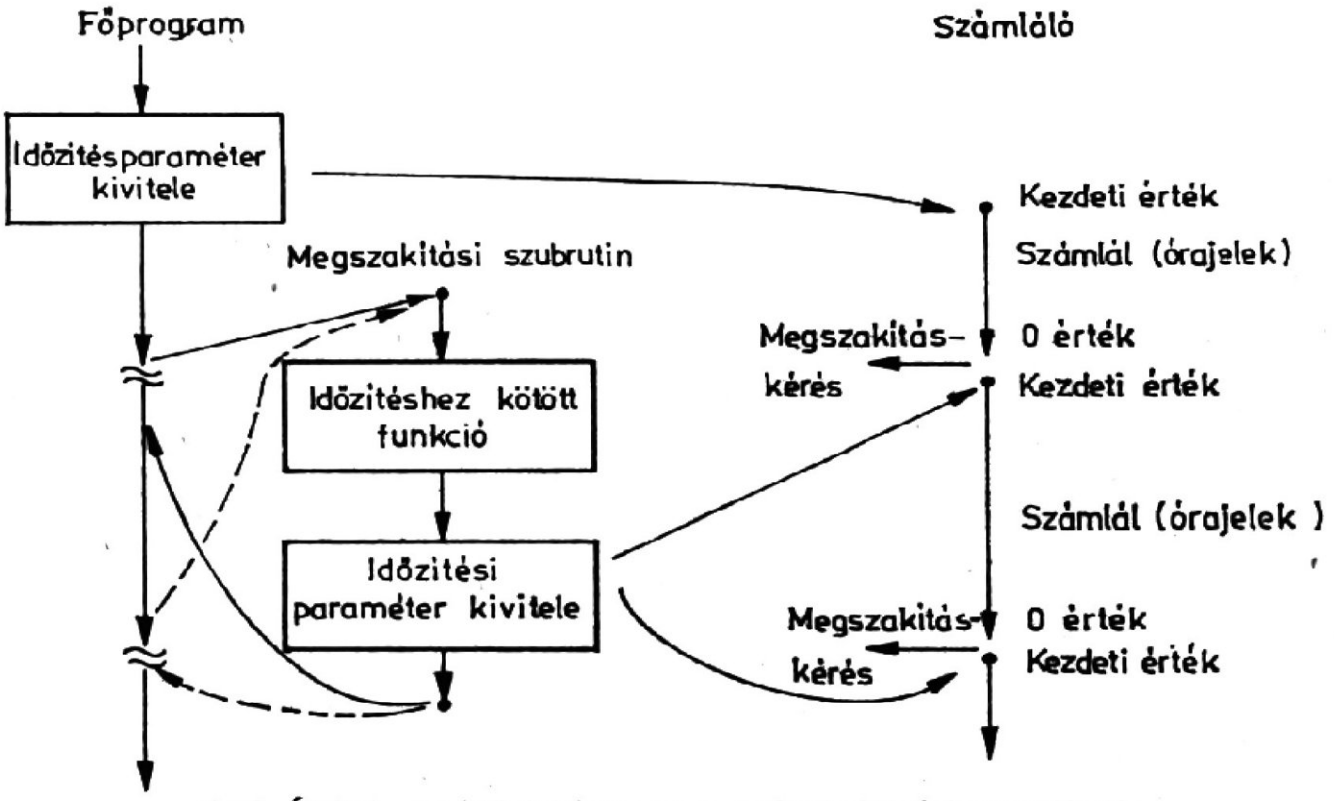
A mikroprocesszoros rendszerekben az egyik gyakran alkalmazott illesztési feladat a mechanikai eszközök illesztése. Ilyen feladat pl.: a nyomtatók vezérlése, léptetőmotorok vezérlése stb. Az ilyen vezérlési feladatokhoz a legváltozatosabb késleltetési és számlálási műveletekre van szükség.

A programozható időzítő–számláló eszközök (programmable counter/timer) segítségével az ilyen funkciók könnyen, időrabló programhurkok használata nélkül valósíthatók meg. A 6.15. ábrán egy tipikus LSI időzítő–számláló belső felépítését mutatjuk be. Lényegében három 16 bites számlálót tartalmaz, amelyek egymástól függetlenül működnek, de közös be-, ill. kimeneti vezérlőegységük van. A program a három számláló bármelyikébe adatot tölthet be, amely a számláló kezdeti tartalmának felel meg. Ezután a külső órajelek a parancsutasításban meghatározott módban (BCD vagy bináris) a számlálót léptetik. A külső órajel tiltható vagy engedélyezhető egy külső kapuzójellel. Amint a számláló tartalma eléri a 0-át, a kimeneten megjelenik az időzítőjel (vagy számlálófunkciók esetén a túlcsordulásjel). Ezt a megszakításkezelő egységgel összekötve a program is értesül az időzítés végéről (vagy a túlcsordulásról). Többszintű késleltetés érhető el egynél több számláló igénybevételével és a megszakításkérések megfelelő alkalmazásával.

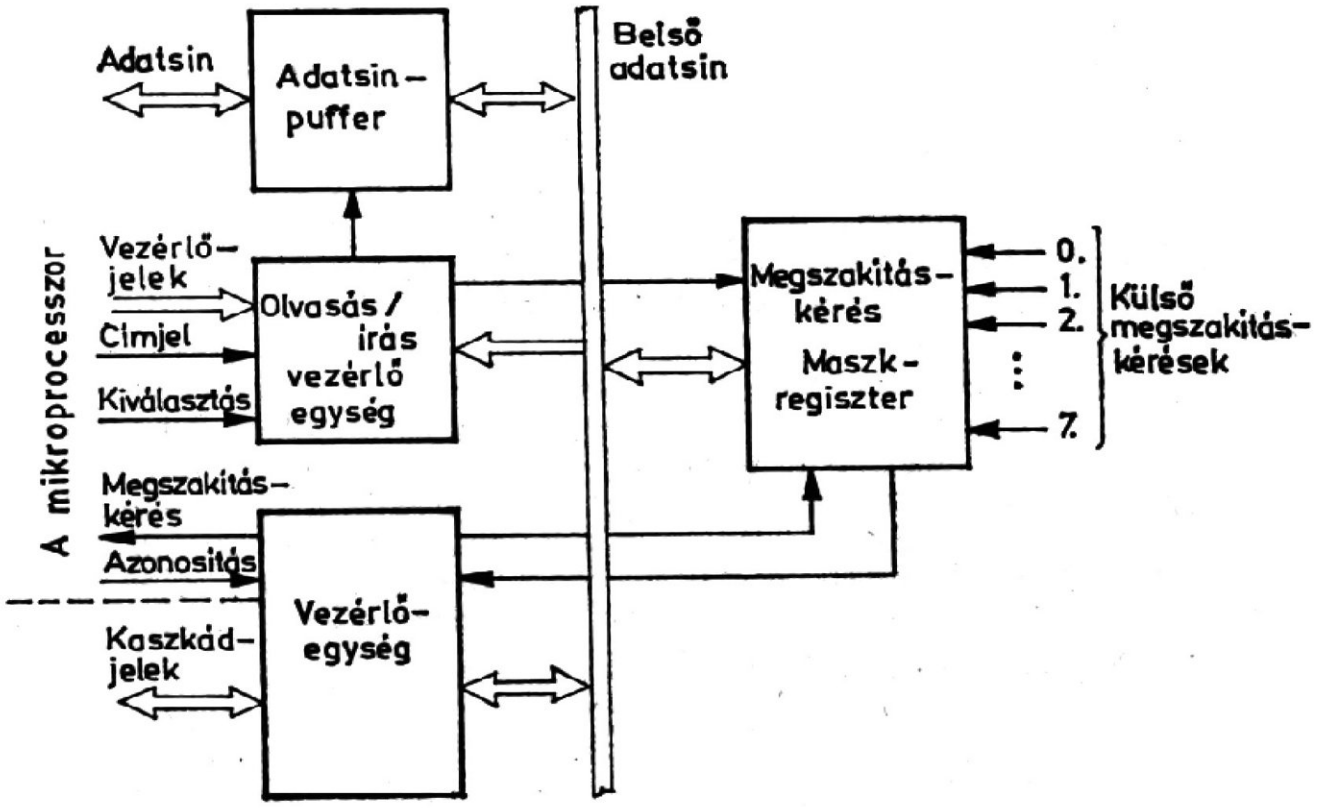
Az időzítő–számláló elem bemutatott típusa be-, ill. kimeneti elemenként kezelhető a mikroprocesszor be-, ill. kimeneti utasításaival. A három számláló három be-, ill. kimeneti egység, amelyekbe adatok írhatók (időzítés kiindulási adataiként), és amelyekből adatok olvashatók ki (a számlálók tartalma). Ez utóbbi művelet ese-



6.15. ÁBRA. A programozható időzítő–számláló áramkör blokkvázlata [7]



6.16. ÁBRA. Az időzítési program és funkciók folyamatábrája



6.17. ÁBRA. A programozható megszakításkérés-kezelő egység blokkvázlata [7]

mények számlálására is alkalmassá teszi az elemet. A negyedik kimeneti csatornaként tekinthető a parancsszó fogadására alkalmas „vezérlőszót és vezérlési módinformációt" tároló regiszter.

Az időzítés folyamatábrája a 6.16. ábrán látható.

Programozható megszakításkérés-kezelő egység

A mikroprocesszoros rendszerekben a mikroprocesszor által kiszolgált perifériák száma, és a kiszolgálási sebesség igen szoros kapcsolatban van a rendszer által elvégezhető feladatok mennyiségével. Minél nagyobb a kiszolgálási sebesség, annál több feladatot végezhet el a mikroprocesszor. A kiszolgálás sebességét a megszakítási rendszer sebessége nagymértékben meghatározza. A kérések hatékonyabb kezelése jelentősen növelheti a mikroszámítógép lehetőségeit, a vezérelt feladatok mennyiségét.

A 6.17. ábrán egy programozható megszakításkérés-kezelő LSI áramkör elvi blokkvázlatát adjuk meg. Ez az LSI áramkör nyolc külső megszakításkérés-jelet fogad. Több áramkör is használható kaszkádjelekkel összekötve, akár 64 megszakításkérés-jel fogadására is alkalmas blokk építhető fel több LSI áramkörből. A prioritási sorrend algoritmusa parancsutasításokkal programozható, és a rendszer működése közben is állandóan változtatható a program szükséglete szerint. A prioritási algoritmus megvalósítására, valamint az egyes kérések tiltására és engedélyezésére szolgál a vezérlő logika áramköre és a maszkregiszter. A megszakításkérések azonosításakor különböző azonosítási vektorgenerálási módban dolgozhat az egység. Így egybyte-os és több byte-os vektor beküldésére is alkalmas megszakításkezelő egység építhető fel több LSI áramkör kaszkádba kötésével.

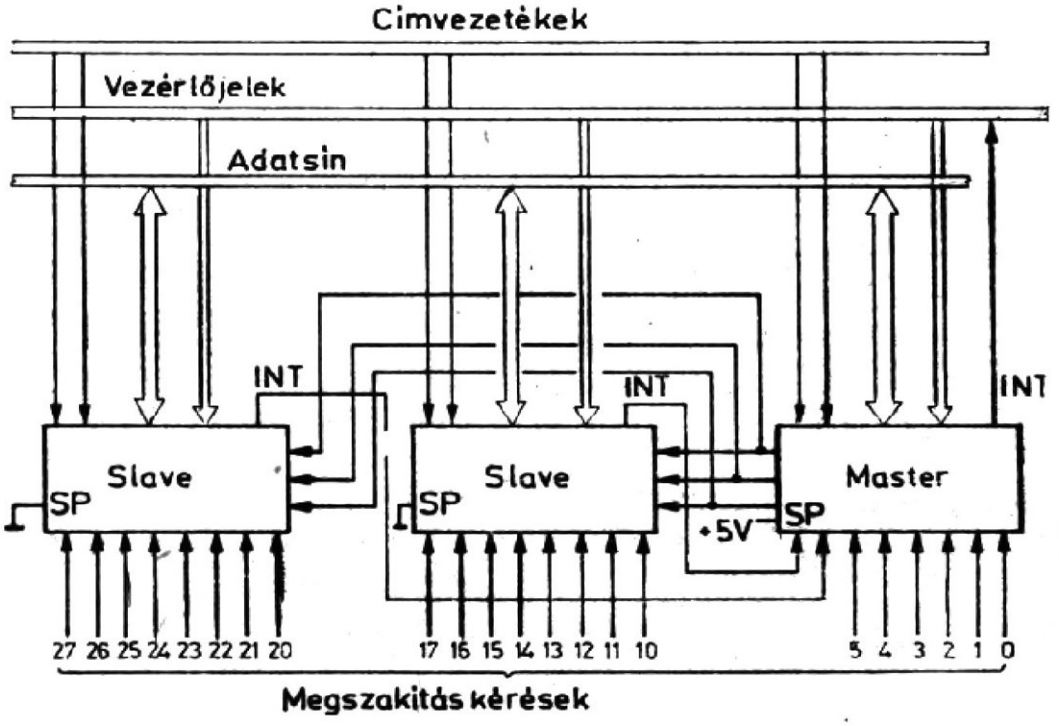
A processzor oldaláról a programozás és az állapotlekérdezés a programozható perifériaillesztő, ill. a számláló- és időzítőegységekhez hasonlóan történik.

A kaszkádkapcsolás, több byte-os vektor beküldése esetén az egyik LSI áramkör masterként, a többiek slave-ként köthetők. A master gyűjti össze a slave-ek által begyűjtött INT megszakításkéréseket és továbbítja a processzornak. Azonosításkor, ha pl.: egy CALL szubrutinhívó utasítás beküldése szükséges, a master küldi be az utasítás műveleti kódját, míg a slave-ek feladata, hogy a bemenetükre kötött megszakításkérésnek megfelelő címet (ami a parancskivitellel beprogramozott algoritmusnak is a függvénye) a CALL utasítás címrészeként beküldje. Azt, hogy az LSI áramkör master vagy slave, nem a program, hanem a hardverbekötés határozza meg. Egy három LSI áramkörből álló megszakításkérés-kezelő egység blokkvázlatára a 6.18. ábra mutat példát. Az LSI áramkörök közvetlenül a minkroprocesszor adat-, vezérlő- és címvezetékeire csatlakoznak, valamint egymáshoz kapcsolódnak a kaszkádjeleikkel. A bemenetükre csatlakoznak a perifériák megszakításkérései, max. 22 jel. Az SP bemenetekre adott logikai jelszinttel jelöltük ki a master, ill. a slave áramköröket. A master a kaszkád kimenetein jelzi a slave-eknek, hogy melyik megszakításkérésének az azonosítására kerül sor.

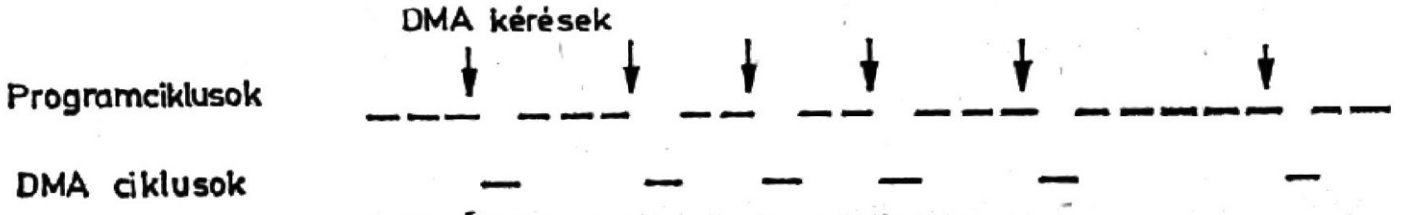
6.3. KÖZVETLEN TÁRHOZZÁFÉRÉS

A közvetlen tárhozzáférés (DMA) adatátvitelei a program zavarása nélkül mennek végbe, azt teljes egészében a hardver vezérli. Ennek köszönhetően a DMA interfészegység sokkal komplikáltabb, mint a programozott átvitelek illesztőegységei. A processzornak és akkumulátorának a megkerülésével az átvitel közvetlenül a tár és a periféria között történik meg. Ez azt jelenti, hogy a mikroszámítógépben párhuzamosan (a processzor működését ciklusloposokkal lelassítva) két művelet zajlik:

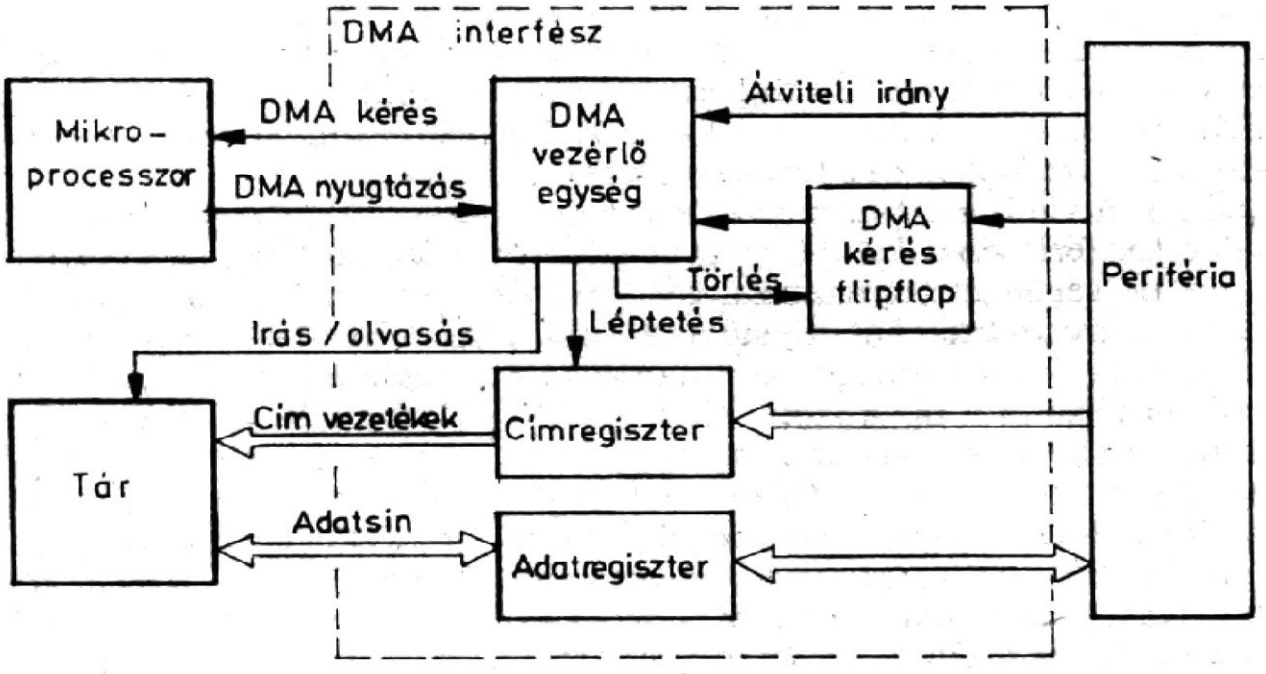
- a program művelete,



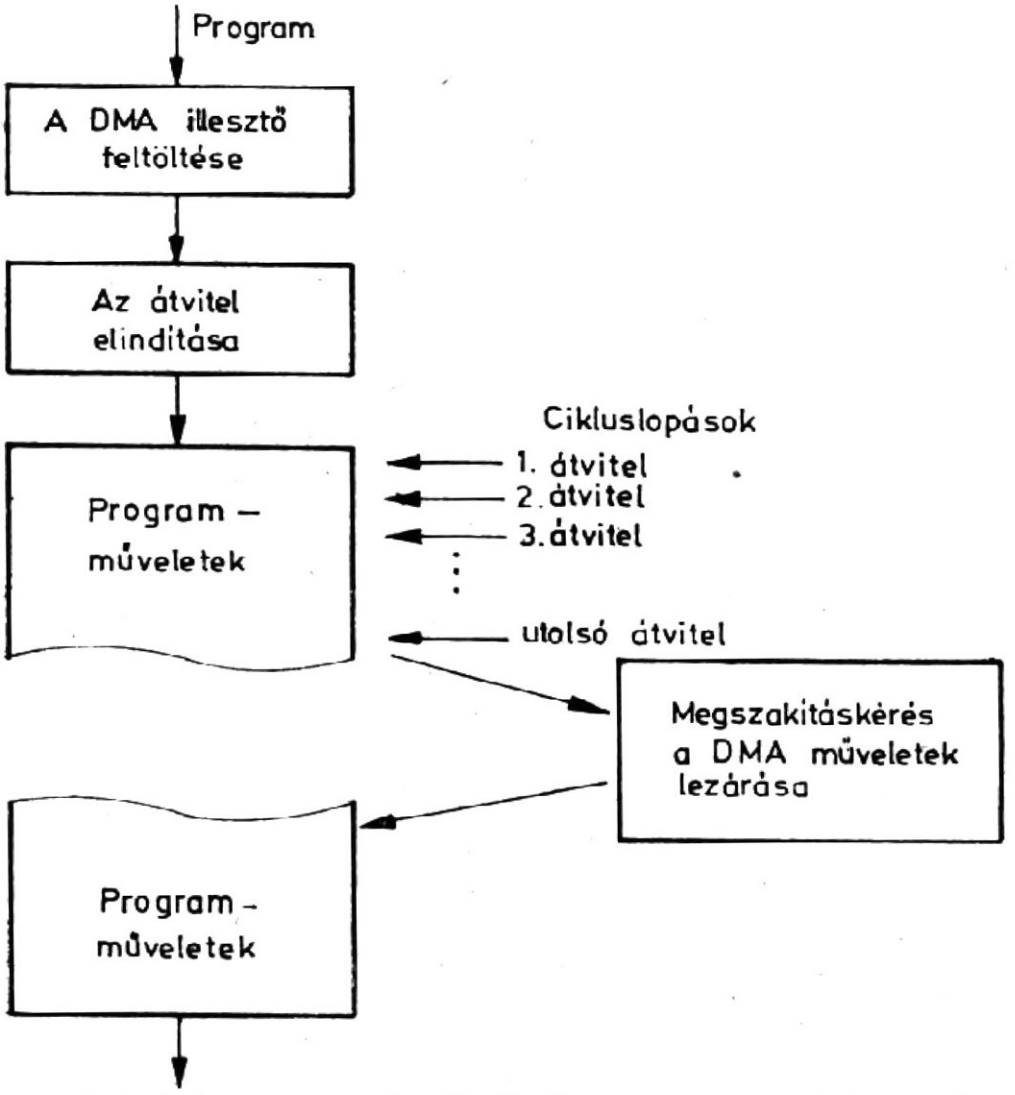
6.18. ÁBRA. A programozható megszakításkésés-kezelő egységek kaszkádkapcsolása [7]



6.19. ÁBRA. A ciklusiopásos átvitelek elve



6.20. ÁBRA. A DMA átviteli illesztőegység blokkvázlata



6.21. ÁBRA. A DMA átvitel programmal való elindításának és végrehajtásának a folyamatábrája

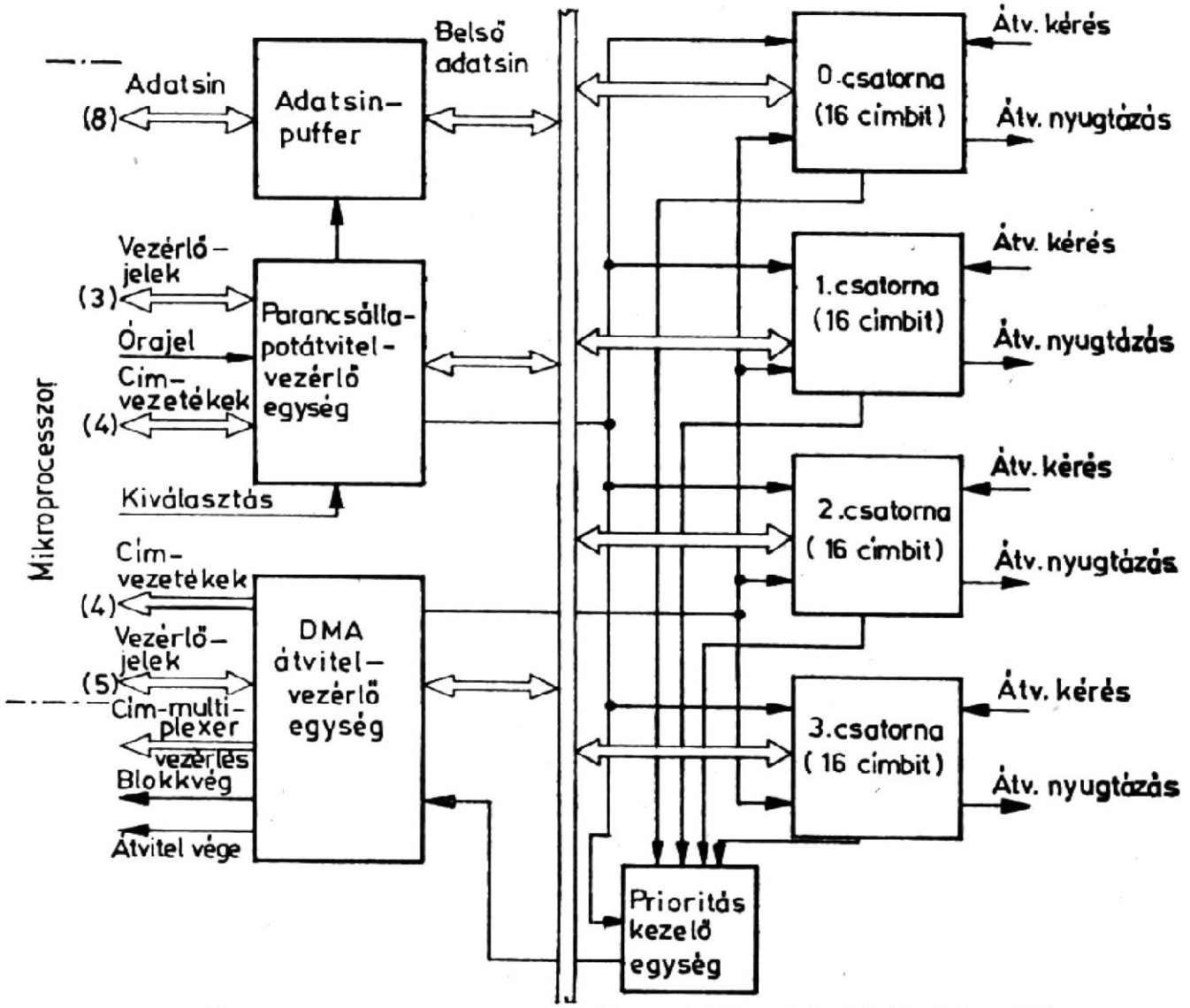
■ a DMA adatátvitel [9], [10].

Tekintve, hogy DMA vezérlő átvitel esetén az átvitel időtartamára a tár adat- és címvezetékeit használja, a processzor tárhozzáféréseit (ciklusait) a DMA átvitelek idejére tiltani kell, a működését fel kell függeszteni. A ciklusiopás elvét a 6.19. ábra mutatja. Az ilyen átvitelszervezés különösen a gyors átvitelt és nagymennyiségű adatot igénylő perifériák (pl.: mágneslemezes tárak) esetében előnyös.

A DMA átvitel hardvermegvalósítására tipikus példát mutat a 6.20. ábra. Az átvitelt a periféria kezdeményezi. Az átvitelkérés-jelet megkapva a mikroprocesszor bármely ciklus végén azt nyugtázza, ezt követően végbemehet az átvitel. A tár az illesztőegység címregiszteréből értesül az átviteli címről, és az írás/olvasás jelnek megfelelően végrehajtja a parancsot. Az adatok az adatregiszter közvetítésével kerülnek átvitelre a periféria és a tár között. Az átviteli blokk indítása előtt a periféria (vagy a processzor) a címregiszterbe tölti a kezdőcímet, az első adat címét. A további adatokat rendszerint a soron következő címekről vagy címekre kell átvinni, ehhez a DMA-vezérlés címléptető (növelő vagy csökkentő) jeleket generál a címregiszternek.

A legtöbb mikroprocesszor az adat- és címvezetékeit háromállapotú kapuáramkörökön keresztül hajtja meg, s a DMA-kérés nyugtázásakor lekapcsolódik ezekről a sínekről. Ez esetben a DMA illesztőegység cím- és adatvezetékei közvetlenül a processzor sínjére köthetők.

A perifériák ugyan maguk vezérlik a DMA adatátvitelt, de az adatblokk átvitelének a megkezdésére általában a program ad parancsot, s ugyancsak értesülnie



6.22. ÁBRA. A programozható DMA vezérlőáramkör blokkvázlata [7]

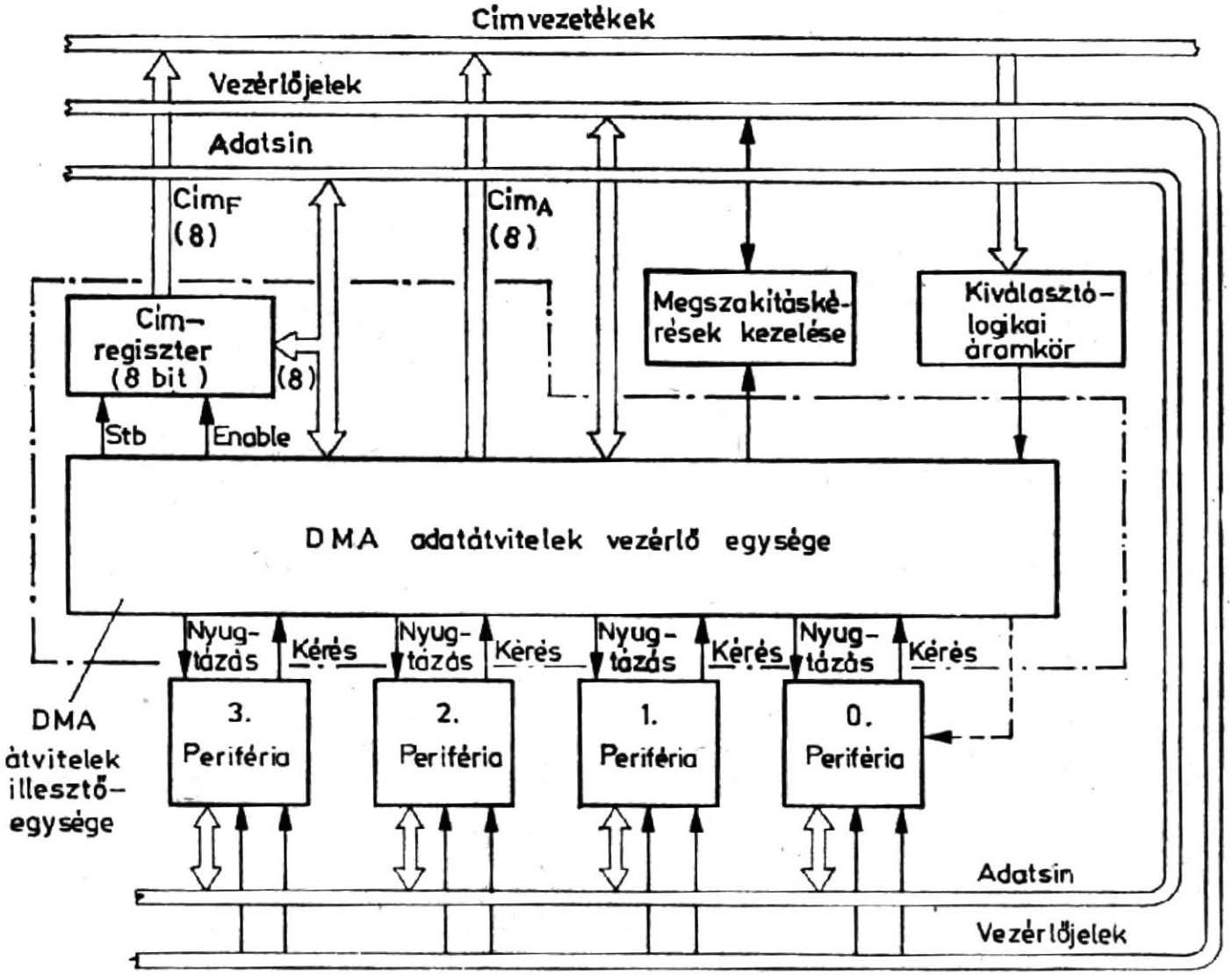
kell a programnak az adatátvitel befejeztéről. Ez utóbbit leggyakrabban megszakításkérő jel generálásával közli az átviteli illesztőegység. (A 6.20. ábrán az egyszerűség kedvéért elhagytuk a programkapcsolat hardverrészeit.)

Így a DMA átvitellel kapcsolatos teljes műveletsor:

- a DMA-illesztőegység kiinduló adatokkal való feltöltése,
- parancsadás az átvitel elindítására,
- ismételt ciklusolopásos adatátvitel,
- az átvitel befejezése, a DMA műveletek lezárása.

Ezeket a lépéseket összefoglalva a 6.21. ábra mutatja. A folyamat programozására példát egy LSI DMA átviteli vezérlőelem ismertetése során adunk.

A 6.22. ábrán egy LSI programozható DMA átviteli vezérlőegység blokkvázlatát láthatjuk. Az LSI egység négycsatornás. Feladata, hogy max. négy periféria DMA átvitelét szervezze meg és vezérelje. A négy csatorna vezérlő áramkörének mindegyike a 6.20. ábrán bemutatott illesztőegységgel megegyező felépítésű, azzal a különbséggel, hogy a vezérlőáramkörök egy, a négy csatornával kapcsolatos közös része a DMA átvitelvezérlés blokkba került, valamint, hogy minden csatorna a címszámlálón kívül egy további 14 bites számlálót is tartalmaz. Az itt ismertetendő LSI DMA átvitelvezérlőnek ui. az egy blokkban átviendő adatok számát is meg lehet adni, s ez a 14 bites számláló használható a blokkhossz megadására, ill. a blokkvég meghatározására. (A korábbi 6.20. ábrán látható egyszerűbb felépítésű illesztőegység



6.23. ÁBRA. A DMA átviteli illesztőegység és a perifériák kapcsolata

működésénél feltételeztük, hogy a periféria kezeli a blokkhosszal kapcsolatos teendőket.)

A csatornalogikai áramkörök a 14 bites byte-számláló mellett 16 bites számláló–címregisztert tartalmaznak. A kiindulási adatokkal való feltöltés során fel kell tölteni mind a byte-számlálót (a blokkhosszal), mind pedig a számláló-címregisztert. Ez utóbbiba az első átviendő adat tárcímét kell betölteni. A vezérlő ezután a periféria átvitelkérés-jelére a mikroprocesszor vezérlősínjén keresztül ciklusloпáshoz szükséges felfüggesztéskérés-jelet generál, majd annak a mikroprocesszor általi elfogadása után indítja az átvitelt. Az átvitelhez a 16 bites címregiszter tartalmát a tár címvezetékeire helyezi, s az átvitel irányának megfelelő tárműveleti (írás/olvasás) és perifériaműveleti (olvasás/írás) jelet generál. A tárba íráshoz a perifériából olvasni kell, és fordítva. A perifériákat ezért a vezérlősínre és az adatsínre is csatlakoztatni kell (6.23. ábra). Az átvitel iránya is programozottan adható meg a kiindulási adatokkal való feltöltés során.

Minden csatorna három átviteli módban működhet a program által megadott parancs szerint:

- a tárból a periféria felé irányuló átvitel,
- a perifériától a tárba irányuló átvitel és
- az ellenőrzés mód.

Több csatorna esetén felvetődik a prioritás kérdése. A prioritáskezelő két prioritási módban működhet:

- a 0. csatorna a legnagyobb és a 3. csatorna a legkisebb prioritású,
- változó prioritási sorrend.

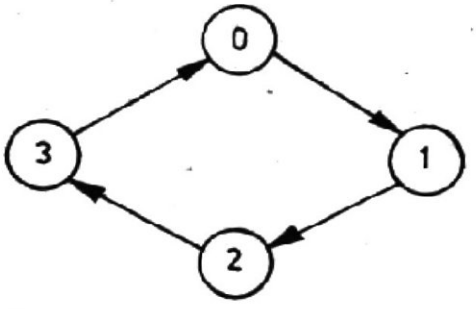
6.4. TÁBLÁZAT. A változó (forgó) prioritási sorrend

Prioritás	Az utolsó kiszolgált csatorna			
	Cs0	Cs1	Cs2	Cs3
Legmagasabb	Cs 1	Cs 2	Cs 3	Cs 0
	Cs 2	Cs 3	Cs 0	Cs 1
	Cs 3	Cs 0	Cs 1	Cs 2
Legalacsonyabb	Cs 0	Cs 1	Cs 2	Cs 3

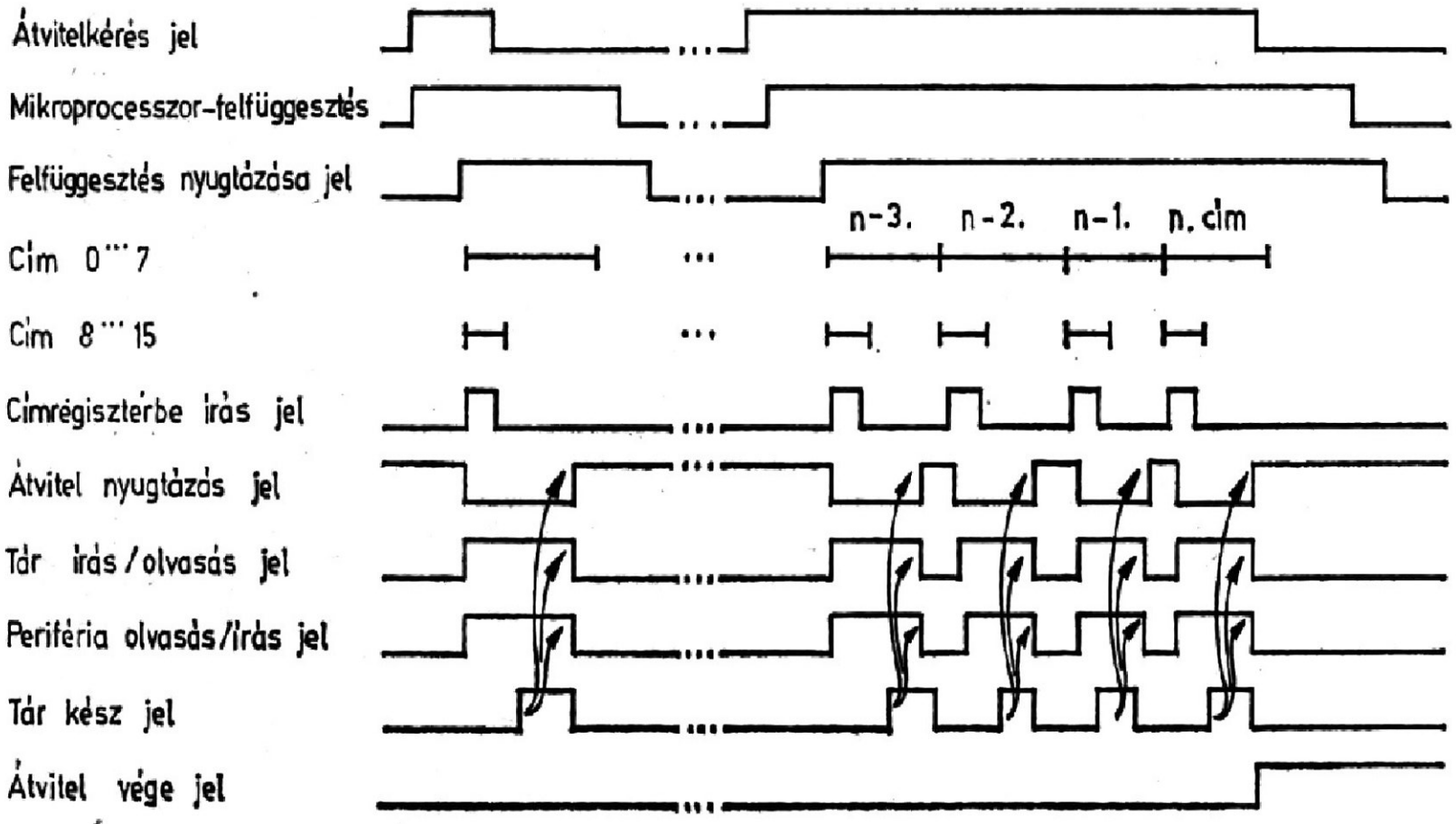
Ez utóbbi esetben minden átvitel után változik a prioritási sorrend, a 6.4. táblázat szerint. A legmagasabb prioritású csatorna sorszámának a változását a 6.24. ábra mutatja.

A DMA átvitelek lebonyolításához 8 bites kiegészítő címregisztert is kell alkalmazni, mert ez a típus a 16 bites címet két részben adja ki a címvezetékekre. Az adatsín 8 bitje és a 8 címbit adja a sín 16 bites címét. Az adatsín további használata miatt azonban a cím felét egy háromállapotú kimenetekkel rendelkező regiszterbe kell megőrizni az átvitel idejére. Természetesen a címnek a regiszterbe tárolásához megfelelő vezérlőjeleket biztosít az LSI DMA átvitelvezérlő.

A megfelelő számú adatátvitel megtörténte után ÁTVITEL VÉGE jelet generál az egység, amely megszakításkérő jelként használható a program felé. Az átvitelek idődiagramját a 6.25. ábrán mutatjuk be. További szolgáltatás a BLOKKVÉG-jel. A hajlékony mágneslemezes tárak esetén használatos 128 byte-os adatblokkok határainak a jelzésére kívánságra a vezérlő blokkvégjel generálására is programozható. A blokkvégjelet pl.: a hajlékony mágneslemezes tár vezérlője az adatok lemezre írásakor használhatja. A blokkos adatátvitelek esetén különböző hibajelzési és ellenőrzési módszerekkel védekezhetünk a hibás blokkok átvitele, a hibás vétel ellen. Az ismertetett LSI áramkörnél hiba-ellenőrzésre az ellenőrzés módot használják. Ez esetben tárolvasás–írás műveletet nem kezdeményez a DMA-illesztőáramkör, csak a perifériáknak ad vezérlőjeleket, ezek hatására a periférián belső adatolvasást hajthatnak végre ellenőrzés, hosszanti redundanciaadatok (pl.: CRC adat) generálása céljából.



6.24. ÁBRA. A prioritási sorrend változása



6.25. ÁBRA. A DMA átvitelek idődiagramja

Érdekes üzemmód az ún. önfeltöltéses üzemmód. Gyakori, hogy egyes alkalmazásoknál ugyanazon tárterületekről ugyanannyi adatot tartalmazó blokkokat kell átvinni. Tekintve, hogy a blokkátvitelek közben a blokkhossz-számláló és a címregiszter tartalma is változik, ez esetben is minden újabb blokk átvitelét megelőzően a programnak fel kell töltenie ezt a két regisztert. Ezt az újabb feltöltést az LSI áramkör önállóan, a program beavatkozása nélkül az önfeltöltéses üzemmódban végzi. Ebben az esetben az egyik nem használt csatorna két regiszterébe tölti be a program a kiindulási adatokat. Az átviteleket a másik csatorna vezérli, s amint átvitelvéghez ér, automatikusan áttöltődik a regisztereibe a nem használt csatorna regisztereiben lévő két kiindulási adat. Így az átviteleket lebonyolító csatorna újabb átviteli blokk be- vagy kivitelére kész.

A kezdeti adatmegadáshoz, ill. az állapotlekérdezéshez a program be-, ill. kimeneti utasításokat használ. Csatornánként 16 bites kezdőcímet és 14 bites blokkhosszadatot, valamint két bitnyi műveletkódot (kivitel, bevitel, ellenőrzés) kell megadni

6.5. TÁBLÁZAT. A programozott DMA illesztőegység műveleti kódja [7]

15 bit	14 bit	Művelet
0	0	Ellenőrzés
0	1	Tárba írás
1	0	Tárból olvasás
1	1	Tiltott kód

(6.5. táblázat). Ezek megadását a program csatornánként kétszer két kimeneti utasítással végzi (a 16 bites címet két, egymást követő kimeneti utasítással, és újabb két utasítással a 14 bites adathossz értéket és a 2 bites parancskódot kell megadni). Bemeneti utasításokkal a program bármikor visszaolvashatja ugyanezeknek a paramétereknek a pillanatnyi értékét. Ezen túlmenően egy parancskiviteli utasítással adható meg a működési mód, és állapotbeviteli utasítással kérdezhető le az LSI illesztőegység állapota. A 6.6. táblázat összefoglalja a parancskód bitjeinek és az állapotkód bitjeinek az értelmezését. A táblázatban a parancskód 5. bitje a nyújtott írási ciklus parancs. Egyes tárakkal való együttműködésnél az írásjel időzítésének a megváltoztatására (megnyújtására) van szükség, ezt lehet előírni ezzel a bittel. A 4. állapotbit jelentése, hogy az önfeltöltés megtörtént, de az újabb blokk átvitelét még nem kezdte el a periféria. Az adott LSI eszköznél önfeltöltés módban csak a 2. csatorna tud működni, s a 3. csatorna regisztereinek a tartalmát tölti át. Ezért az önfeltöltés mód megadására egy bit elegendő.

A 6.7. táblázatban a parancs- és állapotátviteleket foglaltuk össze. Az összetartozó 16 bites adatok (címregiszter és blokkhosszregiszter tartalma) megadásánál és olvasásánál az egy adathoz tartozó két be-, ill. kimeneti utasítás címrésze nem változik, a két adat olvasását és írását a DMA-illesztőegység egy belső flip-flopja vezérli.

Az alábbiakban az Intel 8080A mikroprocesszor utasításainak a felhasználásával egy programozási példát mutatunk be a DMA átvitelek indítására és az átvitel utáni megszakításkérés kezelésére. A példában egy tárba írási DMA műveletet indítunk a 0. csatornán keresztül. A többi csatorna nem működik. Az átvitel végén megszakításkéréssel jelentkezik a DMA-illesztőegység. A megszakításazonosítást

6.6. TÁBLÁZAT. A DMA illesztőegység parancskódja és állapotkódja [7]

Bitsorszám	Parancskód	Állapotkód
7	2. csatorna önfeltöltéses működése	
6	Átvitelvég-generálás engedélyezése	
5	Nyújtott írási ciklus	
4	Változó prioritási sorrend	Önfeltöltés
3	3. csatorna működés engedélyezése	3. csatorna átvitelének vége
2	2. csatorna működés engedélyezése	2. csatorna átvitelének vége
1	1. csatorna működés engedélyezése	1. csatorna átvitelének vége
0	0. csatorna működés engedélyezése	0. csatorna átvitelének vége

az átvitelvégjel lekérdezésével végezzük. A kezdeti feltöltéshez a tárban lévő táblázatból olvassa ki a program az adatokat. Az illesztőegység megcímzéséhez a be-, ill. kimeneti utasítások címparamétere $10_{16}...18_{16}$.

```
      Főprogram:
;a DMA illesztőegység kezdeti feltöltése
;a kezdőcím megadása
     INIT: LXI H,TABL    ; táblázat kezdőcíme a HL-be
           MOV A,M       ; kezdőcím alacsony byte A-ba
           OUT 10H       ; 0. csatornába kivitel
           INR L         ; HL növelése
           MOV A,M       ; kezdőcím magas byte A-ba
           OUT 10H       ; 0. csatornába kivitel
;a blokkhossz megadása
           INR L         ; HL növelése
           MOV A,M       ; blokkhossz alacsony byte A-ba
           OUT 11H       ; 0. csatornába kivitel
           INR L         ; HL növelése
           MOV A,M       ; blokkhossz magas byte A-ba
           ORI 40H       ; tárba írás kódja
           OUT 11H       ; 0. csatornába kivitel
;parancskód kivitele
           MVI A,41H     ; átvitelvég-generálás, 0. csatorna engedélyezése
           OUT 18H       ; parancs kivitele
;a periféria indítása (perifériafüggő program)
;főprogram folytatása
           :
```

6.7. TÁBLÁZAT. A programozott DMA vezérlő programozása [7]

Paraméter		Címbemenetek				Belső flip-flop	Adatbitek jelentése							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0. csatorna-címregiszter (BE/KI)	AB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	FB					1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
0. csatorna-blokkhosszregiszter (BE/KI)	AB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	FB					1	KÓD	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	
1. csatorna-címregiszter (BE/KI)	AB	0	0	1	0	0	ua., mint a 0. csatorna							
	FB					1								
1. csatorna-blokkhoszregiszter (BE/KI)	AB	0	0	1	1	0								
	AF					1								
2. csatorna-címregiszter (BE/KI)	AB	0	1	0	0	0	ua., mint a 0. csatorna							
	FB					1								
2. csatorna-blokkhosszregiszter (BE/KI)	AB	0	1	0	1	0								
	FB					1								
3. csatorna-címregiszter (BE/KI)	AB	0	1	1	0	0	ua., mint a 0. csatorna							
	FB					1								
3. csatorna-blokkhosszregiszter (BE/KI)	AB	0	1	1	1	0								
	FB					1								
Parancs (kivitel)		1	0	0	0	0	Parancskód							
Állapot (bevitel)		1	0	0	0	0	Állapotkód							

AB alacsony helyi értékű byte
FB magas helyi értékű byte

C_0...C_{13} blokkhossz
KÓD műveleti kód (l. a 6.5. táblázatot)

```
A megszakítási szubrutin:
    INT: PUSH H          ; a regiszterek elmentése
         PUSH D
         PUSH B
         PUSH PSW        ; A és a jelzőbitek elmentése
;DMA-illesztő megszakításkérésének azonosítása
         IN  18H         ; állapotkód behozatala
         MOV B,A         ; állapotkód B-be mentése
         ANI 1           ; 0. bitmaszk
         JZ MASINT       ; más megszakításkérés, ugorj
;a 0. csat. megszakításkérés kezelése (alkalmazásfüggő program)
         :
;szubrutin vége
         POP PSW         ; A és a jelzőbitek visszatöltése
         POP B           ; a regiszterek visszatöltése
         POP D
         POP H
         EI              ; megszakításkérés engedélyezése
         RET             ; visszatérés a főprogramba.
```

Az egyéb LSI illesztőegységekhez hasonlóan a DMA illesztőknek is sok változatát dolgozták ki a különböző mikroprocesszorokhoz. Az egyes típusok felépítése, programozása többé-kevésbé eltérő, felépítési és működési alapelvük azonban megegyezik az itt bemutatott típussal.

6.4. PERIFÉRIAVEZÉRLŐ LSI ÁRAMKÖRÖK

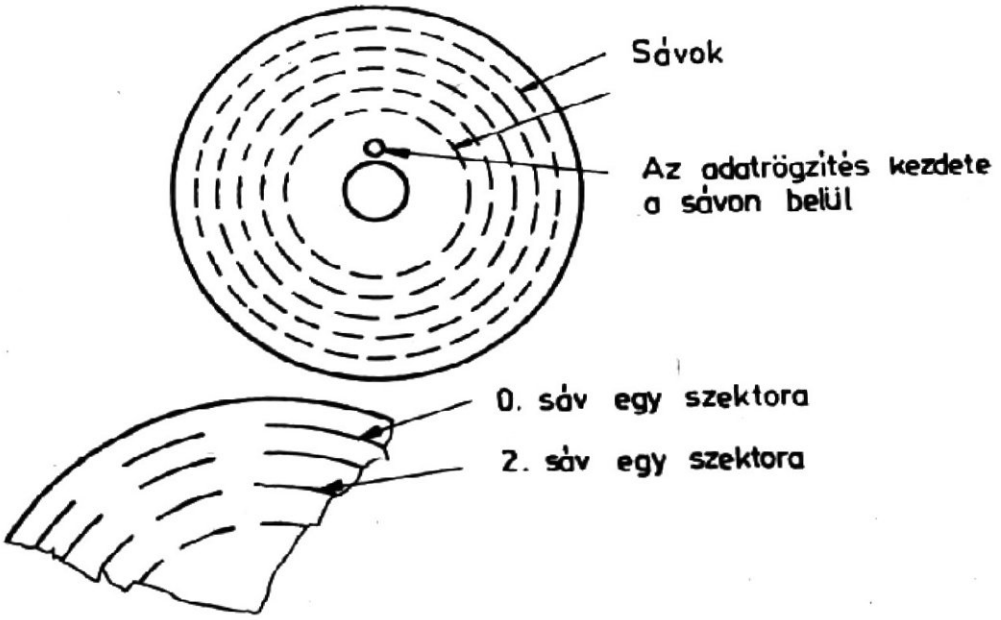
Az általános célú be-, ill. kimeneti vezérlőáramkörök csak az adatátvitel irányításában és lebonyolításában vesznek részt. Az előzőekben a programozott és a közvetlen tárhozzáférés adatátvitelét vezérlő és lebonyolító LSI és VLSI alkatelemek ismertetéséhez mutattunk be tipikus eszközöket.

Egyes általánosan elterjedt perifériák működésének a vezérléséhez az adatátvitelek megszervezésén kívül további bonyolult funkciókat megvalósító áramkörök is szükségesek. A következőkben néhány ilyen LSI és VLSI perifériavezérlőt mutatunk be [11].

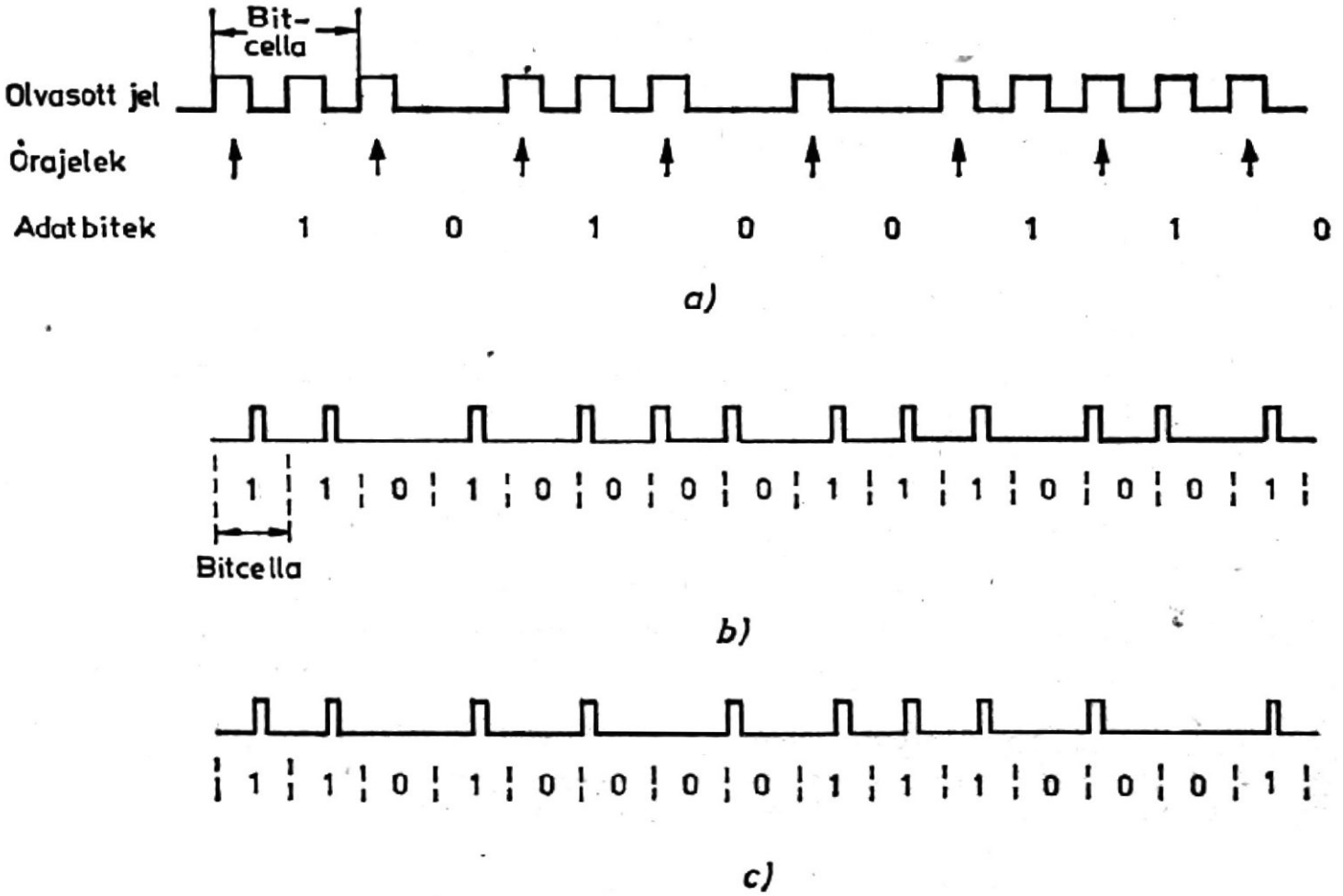
Hajlékony mágneslemezes tárat vezérlő LSI áramkörök

A hajlékony mágneslemezes tárak a mikroprocesszoros készülékek és mikroszámítógépek leggyakrabban használt háttértárai. Az 1972-ben megjelent 256 kbyte-os, a mágneslemezen egyoldali adatrögzítést megvalósító, egyszeres adatsűrűségű tároló-eszközöket egyre inkább kiszorítják az 1Mbyte-os, kétoldali rögzítést megengedő és kétszeres adatsűrűségű tárak. (Itt a kapacitásra vonatkozó adatok természetesen egy cserélhető mágneslemez tárkapacitását jelentik.)

A hajlékony mágneslemezes egységeknek, valamint az adatrögzítési megoldásoknak sokféle változata ismeretes. Itt nincs arra mód, hogy ezekről részletesebben írjunk. Az egyik legelterjedtebb adatrögzítési mód az ún. „soft-sectoros" rögzítés, ezt használják az IBM kompatibilis formátumú mágneslemezek esetében is. Az adatokat koncentrikus ún. sávokra, trackekre írják fel (6.26. ábra). A szabványos IBM kompatibilis mágneslemez 77 sávot tartalmaz. A sávokon belül az adatokat adatblokkok formájában rögzítik. Az adatblokkok hossza többféle lehet, az IBM szabvány



6.26. ÁBRA. A hajlékony mágneslemezes tár adatrögzítés-formátuma (soft-sectoros lemez)



6.27. ÁBRA. A hajlékony mágneslemezen alkalmazott adatkódolási módok
a) FM kódolás, *b)* MFM kódolás, *c)* M²FM kódolás

szerint 128, 256, 512 és 1024 byte-os szektorok használhatók. A szektorok száma a blokkhossz és az adatsűrűség függvénye. A szektorok adatait ún. szektorközök, adatokat nem tartalmazó részek (gap) választják el egymástól.

A szektorokban mind az adatbyte-ok, mind pedig a byte-on belüli bitek soros formában kerülnek felírásra. A rögzítéskor tetszőleges tartalmú adatok bitjei közé szinkronizálás céljából órajeleket is beírnak. A rögzítés az egyszeres adatsűrűségnél FM típusú, a kétszeres adatsűrűségnél pedig a módosított FM (MFM) vagy kétszeresen módosított FM (M^2FM) típusú (6.27. ábra).

Az MFM és M^2FM kódolásban az adatbitek jelentik az órajeleket is, tehát ez az eljárás önszinkronizáló. Az órajelek elhagyása, helyettük adatbitek írása ugyanolyan mágnesfluxusváltási gyakoriság mellett is kétszeres adatsűrűséget eredményez. Természetesen az órajeleknek számító adatbitek még egymás után sok logikai *0*-át vagy *1*-et tartalmazó adatok esetén sem maradhatnak el, ezért a *0*, ill. *1* bit jelölésére nem elegendő a jelhiány, ill. a jel megléte. Egyéb impulzusjellemzők felhasználása is szükséges.

Az MFM kódolási szabály a következő: Az *1* értékű adatbitnek megfelelő impulzus mindig a bitcella közepén jelenik meg. Az *1* bitet követő *0*-nak a bitcellában impulzushiány felel meg, a *0* bitet követő *0*-t pedig a bitcella kezdetén megjelenő impulzus jelöli.

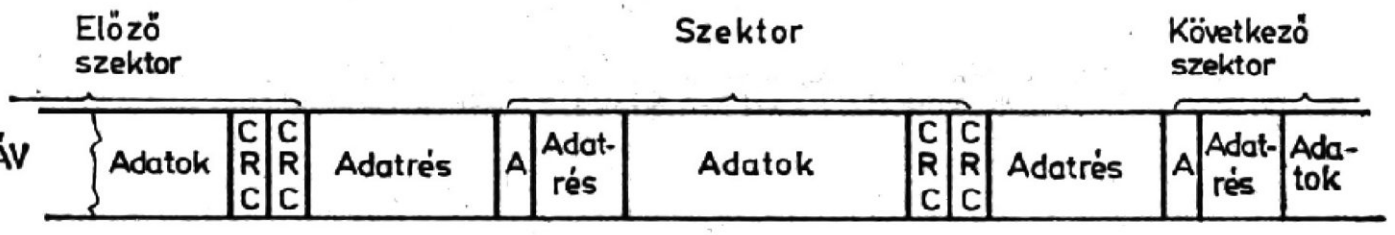
Az M^2FM kódolás szabálya szerint *1* értékű adatbitet továbbra is a bitcella közepén megjelenő impulzussal kódolunk. *0* bitet jelöl a bitcella elején megjelenő impulzus, ha a megelőző bitcellában nem volt sem *0*-t, sem *1*-et jelentő impulzus. Amennyiben a megelőző bitcellában akár az *1*-et, akár a *0*-t impulzus jelezte, és a jelenlegi bit értéke *0*, a bitcellába nem írunk impulzust.

A szoft-szektoros szervezésnél csak a kezdeti szektor, a 0. szektor jelzésére ad a hajlékony mágneslemezes tár meghajtó elektronikája szinkronjelet (a lemezen erre a célra fénysugárral letapogatható lyuk található). A többi szektor megtalálása és azonosítása a rögzített adatok alapján válik lehetővé. Ehhez minden szektor előtt azonosítási mező (ID: IDentification Field) található. Az azonosítási mező tartalmazza a sáv-, ill. a szektorcímet. Ezt követően találhatók az adatbyte-ok, valamint a hibaellenőrzésre szolgáló kódok (ún. CRC: Cyclic Redundancy Code byte-ok). A szektorszervezés a 6.28. ábrán látható.

A hajlékony mágneslemezes tárat vezérlő áramkörök (FDC: Floppy Disk Controller) funkcióit a következőkben foglaljuk össze [4], [12], [13].

1. Vezérlési funkciók

- író/olvasófej működési helyzetbe vitele (head-load),
- sáv kiválasztása, az író/olvasófej sugárirányú léptetése (step és direction),
- az íróáram sávtól függő szabályozása normál vagy csökkentett értékre (low current),
- több lemezmeghajtó egység esetén az egység kiválasztása (unit select),
- a 0. szektor jelének (index) és a 0. sáv jelének fogadása,
- kétoldalas felíráskor az oldal kiválasztása (side select).



A: Azonosítási mező

6.28. ÁBRA. A hajlékony mágneslemezek szektoraiban levő információ szervezése

2. Írási funkciók
- a mikroszámítógéptől érkező párhuzamos adatok fogadása,
- az információ formálása, átkódolása (adat- és órajelek kezelése),
- az adatok soros formába alakítása és a lemezmeghajtó egység felé továbbítása (write data),
- az írásvédelem-jel fogadása (write protect) és írásengedélyező jel generálása (write enable).

3. Olvasási funkciók
- a soros adatok fogadása a lemezmeghajtó egységtől (read data),
- az adatok átkódolása (az adat- és órajelek szétválogatása),
- a párhuzamos adatoknak a mikroszámítógéphez továbbítása.

4. Adatkezelési és ellenőrzési funkciók. Mind az olvasás, mind pedig az írás során szükséges a szektorok kezdetének és az azonosítási adatoknak (adat- és címmarker jeleknek), valamint a hibajelzésre szolgáló CRC kódok kezelése.

Ezeknek a feladatoknak az ellátására LSI bázisú FDC áramköröket is használhatunk. Természetesen az FDC áramkörnek programozhatónak kell lennie. A programozhatóság részben a kezdeti inicializálással összefüggő adatok megadását jelenti (ilyen adatok pl.: a mágneslemezmeghajtó egységgel összefüggő mechanikai időzítések: a fej léptetési sebessége, a fej működési pozícióba helyezésének ideje). Másrészt működés közben is állandóan programozni kell az FDC áramkört az elvégzendő műveletek végrehajtására. Egy tipikus LSI alapú FDC-nek a parancsait a 6.8. táblázatban foglaltuk össze.

6.8. TÁBLÁZAT. A hajlékony mágneslemezes tár vezérlőegység parancsai [12]

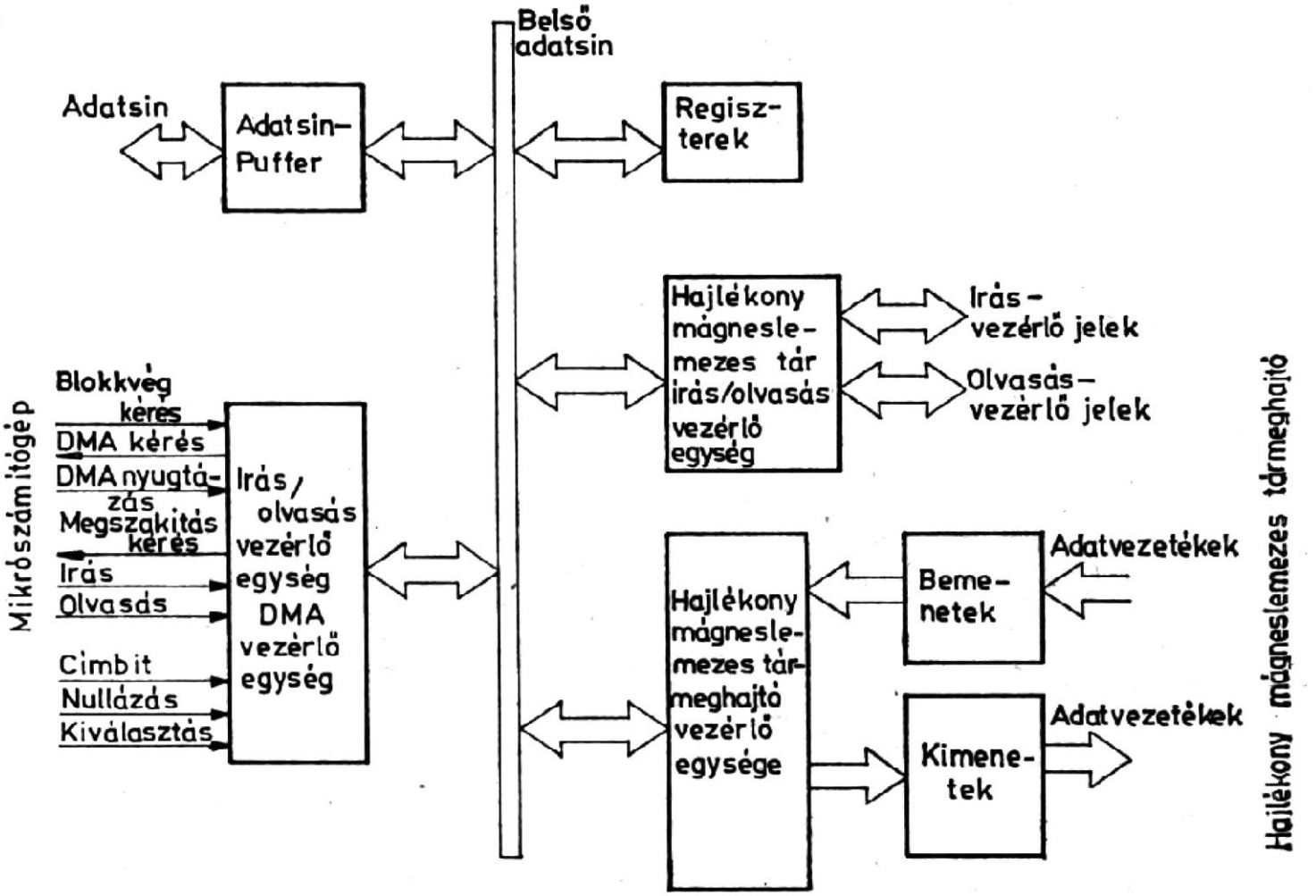
Parancs	Az FDC műveletei
Adat olvasása (READ DATA)	A hajlékony mágneslemezestár-meghajtó író/olvasó fejének működési helyzetbe vezérlése, majd az azonosítási mező olvasása. Amint az azonosítási mező megfelel az olvasandó szektor címének, az FDC kiolvassa a szektor adatokat és a mikroprocesszornak továbbítja. Több szektor olvasására is parancs adható – a parancshoz cím és szektorszám-operandus tartozik. Több szektor olvasásánál a szektorcímet egy-egy blokk befejezése után automatikusan megnöveli az FDC, a szektorszámot pedig csökkenti.
Azonosítási cím olvasása (READ ID)	Az író/olvasó fej pozíciójának a meghatározását szolgálja. A legelső azonosítási címet olvassa be az FDC. Ha címmarkert nem talál két indexjel között (a lemez körbefordulása alatt), a parancsot hibaüzenettel befejezi.
Törölt adatok olvasása (READ DELETED DATA)	Ua., mint az ADAT OLVASÁSA parancs, azzal a különbséggel, hogy a szektor adatait akkor is beolvassa az FDC, ha az adatmarkerben „törölt szektor" jelzést talál.

6.8. TÁBLÁZAT (1. folytatás)

Parancs	Az FDC műveletei
Track olvasása (READ A TRACK)	Egy egész sáv olvasása, az indexjeltől (a 0. szektortól) kezdődően. A többi adatolvasástól eltérően az FDC nemcsak az adatokat olvassa ki és küldi be a mikroprocesszornak, hanem a szektorközi üres adatokat, címeket és CRC kódokat is. Diagnosztikai célokat szolgál ez a parancs.
KERESÉS operandusai: EGYENLŐ, KISEBB vagy EGYENLŐ, NAGYOBB vagy EGYENLŐ (SCAN EQUAL, SCAN LOW OR EQUAL, SCAN HIGH OR EQUAL)	Ez a három parancs lehetővé teszi egy szektor megkeresését. Az FDC leolvassa a lemezről az adatokat, és a megadott adatokkal összehasonlítja őket. Az összehasonlítást byte-onként végzi el. A keresés mindaddig tart, amíg az FDC a szektort megtalálja, a paraméterek feltételeinek megfelelően. A parancsok lehetővé teszik valamely blokk megkeresését.
Adatok írása (WRITE DATA)	A hajlékony mágneslemezes tár meghajtó író/olvasó fejének a működési helyzetbe vezérlése. Ezután az FDC az ID mezők olvasása alapján megkeresi a kívánt szektort. Annak megtalálása után a mikroszámítógéptől adatokat kér és azokat az írófejhez továbbítja a megfelelő időzítések betartásával. Több szektor írására is adható parancs (az ADAT OLVASÁSA parancshoz hasonlóan).
TRACK FORMÁLÁSA (FORMAT A TRACK)	Egy teljes sáv formálása. Az index jel (0. szektor) után erre a parancsra az FDC a mágneslemezre írja a teljes sáv szektorközi órajeleit, cím- és adatmarkereket és a szektoron belüli adatok szinkronizálását szolgáló órajeleket. A formálás egyszeres vagy kétszeres adatrögzítési sűrűségnek felel meg. A mikroprocesszor megadhatja az ID azonosítási adatokat négy byte formájában. Ez lehetővé teszi a szektorok címeinek a fizikai helyüktől független felírását.
A törölt adatok felírása (WRITE DELETED DATA)	Ugyanaz, mint az ADATOK ÍRÁSA parancs, azzal a különbséggel, hogy az adatmarkerek helyére a „törölt adat" markerjelzést írja be az FDC.
Track-re állítás (SEEK)	Az író/olvasó fej léptetése. Az FDC-n belüli sávszámláló tartalma is a fej helyzetének megfelelően változik. Az FDC a léptetés irányát megadó jelet és a szükséges számú léptető jelet állítja elő a hajlékony mágneslemezes tár meghajtó egysége felé.
0. Track-re állítás (RECALIBRATE)	Az író/olvasó fejnek a 0. sávra állítása. Amint a meghajtó egység jelzi a 0. sáv elérését, az FDC nem bocsát ki több léptetőimpulzust. Ha 77 lépés után sem jelenik meg a 0. sáv jelzése, az FDC befejezi a parancsot (hibajelzéssel).

6.8. TÁBLÁZAT (2. folytatás)

Parancs	Az FDC műveletei
A megszakítási állapotadat olvasása (SENSE INTERRUPT STATUS)	Ennek az állapotbyte-nak az alapján a processzor eldöntheti, hogy mi okozta a megszakításkérést (melyik meghajtó, hibás működés, adatátvitel vége stb.).
A meghajtó állapotadatának olvasása (SENSE DRIVE STATUS)	A processzor ennek az adatnak a beolvasásával az FDC és a hajlékony mágneslemezes tár meghajtók állapotának a jeleit kérdezheti le.
Paraméter megadás (SPECIFY)	A meghajtó mechanikától függő időzítések megadása. Megadható az író/olvasó fej működőképes helyzetbe hozásához szükséges bekapcsolási és kikapcsolási idő, valamint a sávokra való léptetés ideje. Továbbá megadható az átviteli üzemmód (DMA) vagy adatonkénti megszakításkérés.



6.29. ÁBRA. Az LSI hajlékony mágneslemezes tár vezérlőáramkörének blokkvázlata [12]

Az FDC blokkjainak a vázlatát a 6.29. ábrán tüntettük fel. Az FDC a mikroszámítógéphez az adatsínnel és a vezérlőjelekkel, valamint a kiválasztójelekkel kapcsolódik. A számítógép felől a parancsok kiküldése a programozott átvitelen keresztül történik. A hajlékony mágneslemezes tár és a mikroszámítógép közötti információátvitel akár a programozott, akár a DMA átviteli csatornák közvetítésével is létrejöhet. Az előbbi feltétele, hogy a processzor kellő sebességgel képes legyen kiszolgálni a minden adat írása vagy olvasása után megjelenő megszakításkéréseket (kétszeres adatsűrűség mellett ezek legrosszabb esetben 13 μs-os időközönként jelentkeznek). A parancsok kivitelét és az állapotszó bevitelét az ilyen módú adatátvitelektől a belső kiválasztó-áramkörök bemenetére csatlakozó címbit különbözteti meg.

Az alkalmazások nagy részéhez a DMA átvitel előnyösebb, mint az adatok programozott átvitele. Ez esetben programozottan csak a parancsok és az állapotinformáció átvitele valósul meg. A parancsok vétele után az FDC áramkör automatikusan elvégzi az előírt funkciókat, vezérli a lemezmeghajtó egységeket, és amennyiben az adatok átvitele is szükséges, azokat a DMA kapcsolaton keresztül szervezi meg. A 6.30. ábrán egy ilyen rendszer vázlata látható.

Az FDC a korábban ismertetett DMA vezérlőn keresztül bonyolítja le az adatok átvitelét a lemezmeghajtók és a tár között. Az átvitelt az ÁTVITELKÉRÉS és az ÁTVITELNYUGTÁZÁS jelek vezérlik. A DMA vezérlő a blokkvég jelét is használja az FDC szektorok írása, ill. olvasása során. A belső áramkörök működtetéséhez, időzítéséhez az órajel-generátor ad jeleket. Üres lemezek formálásához, a szektorórajelek felírásához a szinkronizálást az írásórajel-generátor jelei végzik.

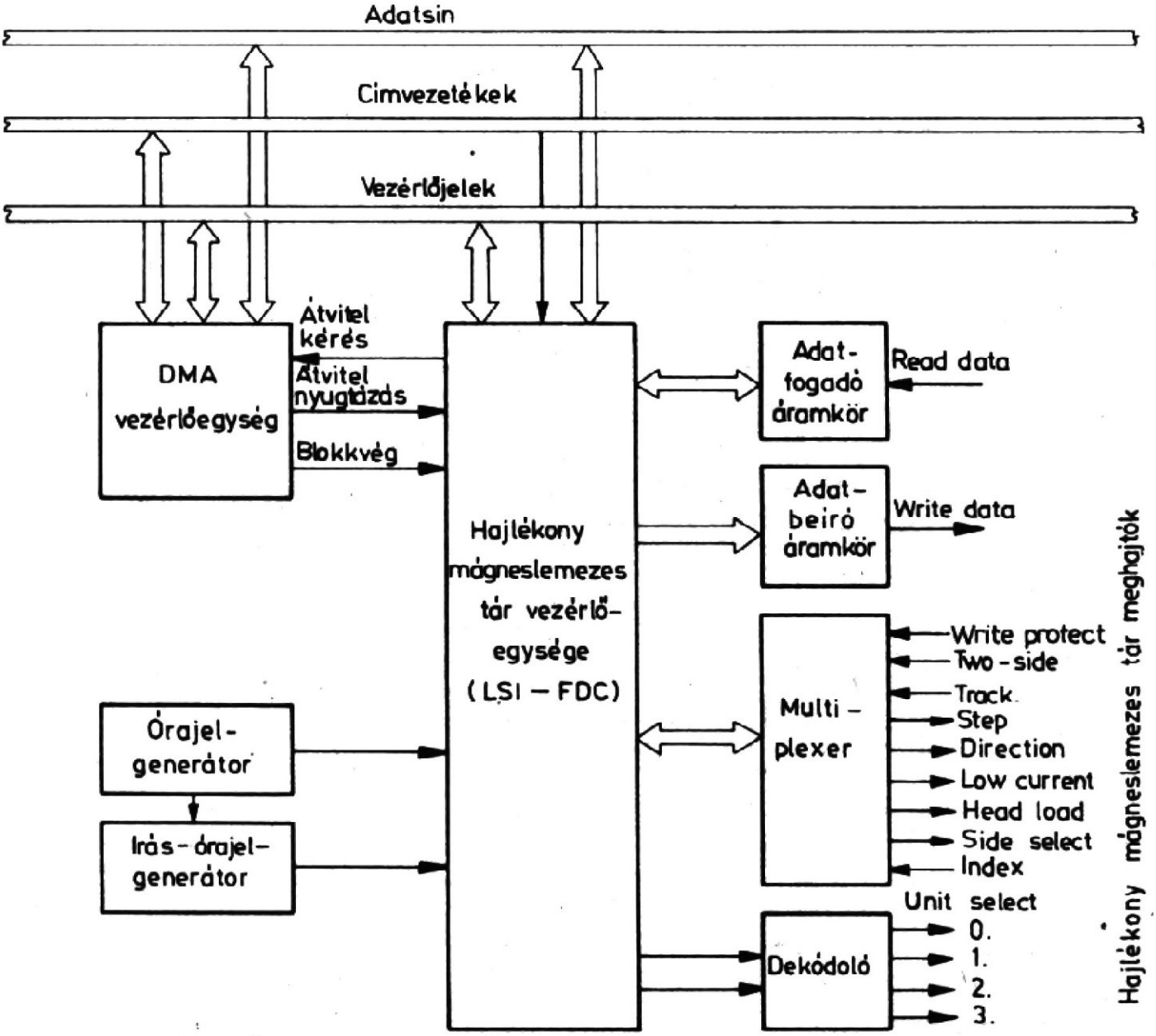
Az FDC egyszerre max négy hajlékony mágneslemezes tár meghajtóvezérlését láthatja el. A meghajtók és az FDC között még a következő egyszerű áramkörökre van szükség: négy meghajtót kiválasztó (dekódoló) logikai áramkörre, valamint négy multiplexerre, a meghajtó és az FDC közötti vezérlőjelekhez. Az írás és olvasás megvalósításához külön író- és olvasójel-formáló áramkörök is szükségesek. Íráskor a belső sávokon a nagyobb adatsűrűség miatt csökkentett íróáramot kell alkalmazni. Egy-egy blokk olvasása, ill. felírása után, valamint a parancsok teljesítése után az FDC megszakításkérés-jelet küld a mikroszámítógépnek. Az ezután az FDC áramkör állapotinformációját kiolvasva értesül a parancs végrehajtásáról, és a hajlékony mágneslemezes tármeghajtók állapotáról.

Az FDC áramkör olvasás és írás közben ellenőrzi az adatok és szektorcímek helyességét, s hiba esetén azt jelzi a processzornak. Automatikusan érzékeli a kétoldalas lemezeket, s ennek megfelelően kezeli a kétoldalt elhelyezkedő író/olvasófejeket.

Az FDC áramkör programozására a 6.31. ábra mutat be egy folyamatábrát. A példa egy szektor írására, ill. olvasására vonatkozik. A hajlékony mágneslemezes tár műveleteket általában bonyolultabb programok, ún. lemezes operációs rendszerek segítségével hajtják végre a mikroprocesszoros vagy mikroszámítógépes rendszerek. Az operációs rendszerek ismertetésével a 8. fejezetben foglalkozunk.

A képernyős megjelenítők vezérlőegységei

A számítástechnikai és irányítási rendszerek leggyakrabban alkalmazott információmegjelenítő eszközei a képernyős megjelenítők. Ezek mind a szöveges (alfanumerikus), mind pedig a grafikus adatok, ábrák kijelzésére alkalmasak. A legolcsóbb megjelenítő eszközök közé a megfelelő vezérlőkkel ellátott televíziómonitorok tartoznak. A vezérlési funkciók megvalósítására, a vezérlők árának a csökkentésére, a hardver tervezésének és megépítésének az egyszerűsítésére a félvezető alkatelemek gyártói programozható LSI képernyős megjelenítő-vezérlőket (VDG: Video Display Gene-



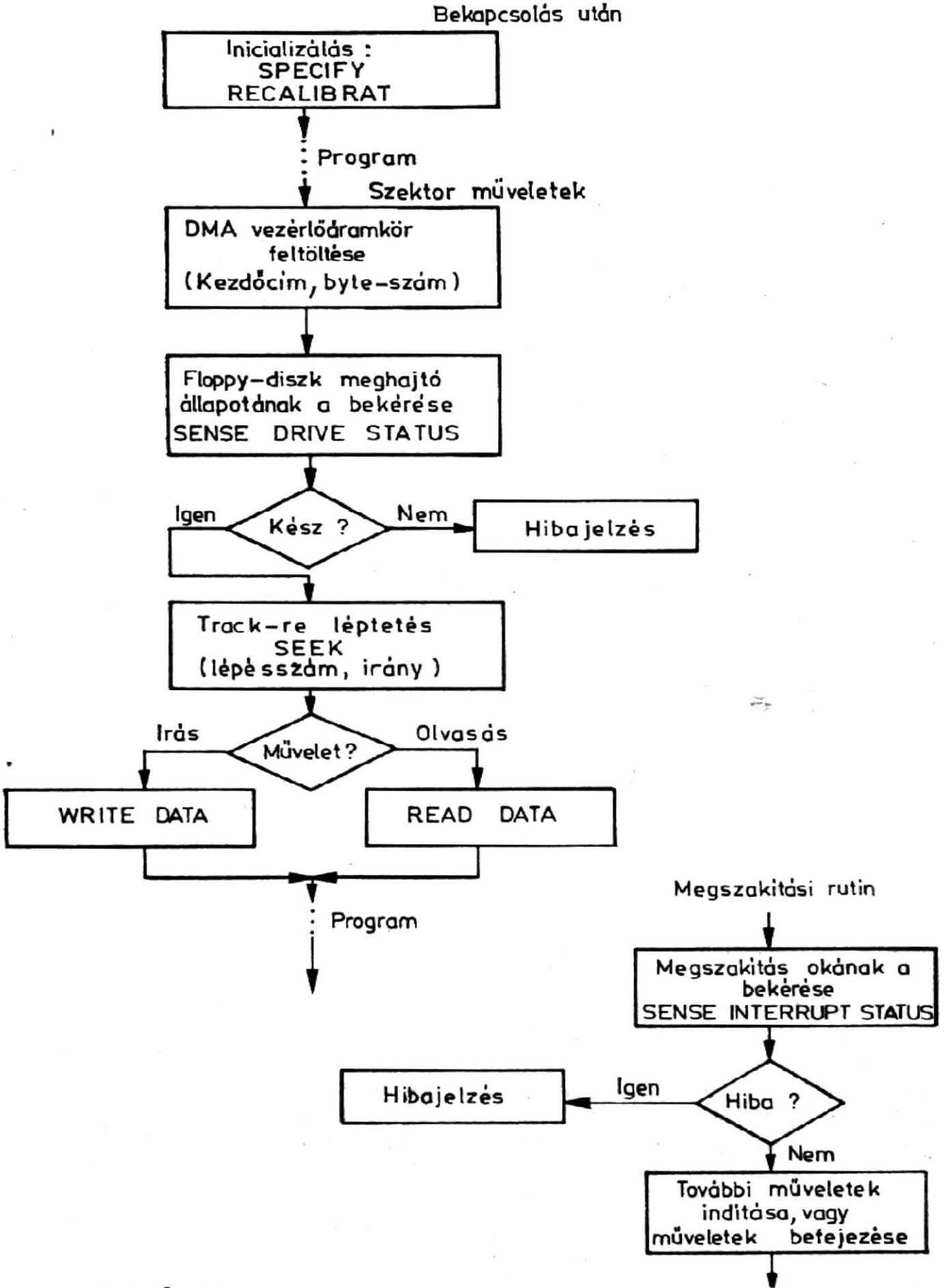
6.30. ÁBRA. Az FDC-nek a mikroprocesszor sínjeivel, valamint a mágneslemezes egység meghajtóival való kapcsolata [12]

rator vagy Programmable CRT Controller) dolgoztak ki és gyártanak [4], [14], [15], [16].

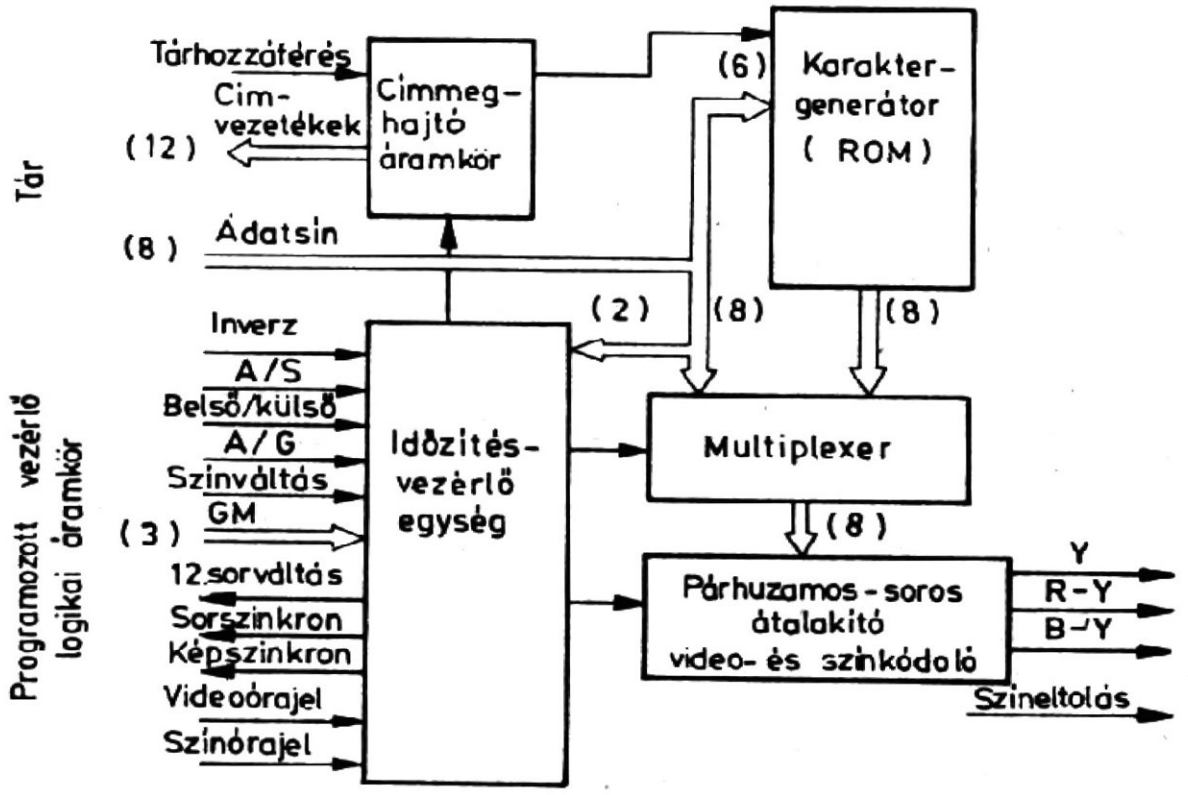
A következőkben egy korszerű, LSI technológiával megvalósított VDG áramkört mutatunk be [15], [16]. Az áramkör feladata, hogy a mikroszámítógép által a megjelenítő tárba beírt képnek megfelelően videojelet generáljon a megjelenítő tv-monitor számára. A videojel fekete-fehér vagy színes monitor bemenetére adható.

A VDG felépítésének vázlatát a 6.32. ábrán láthatjuk. A működési elvét a karaktermegjelenítés elvének a leírásán keresztül mutatjuk be (6.33. ábra). A karaktermegjelenítés azonban csak egy az ismertetésre kerülő VDG különböző üzemmódjai közül. A többi üzemmódot csak röviden tárgyaljuk.

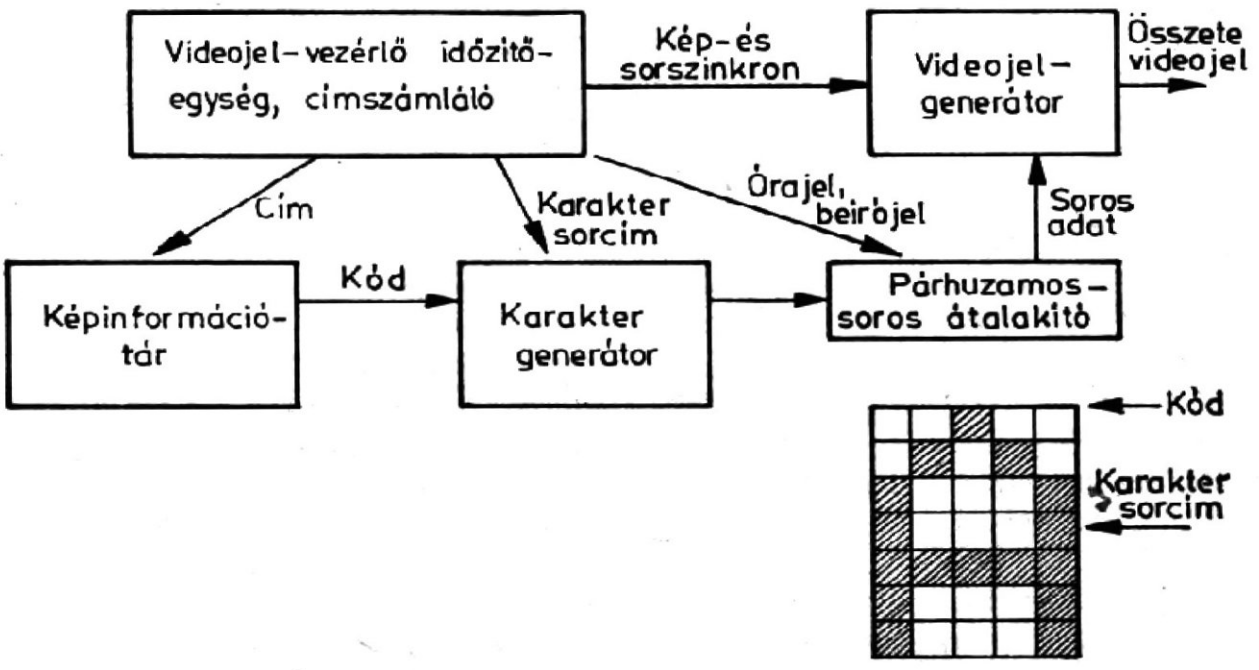
A VDG a megjelenítendő ábra elemeinek a kódját (az alfanumerikus jelek kódját) a hozzá csatlakozó képinformációtárból olvassa ki. A kiolvasási címet a címvezetékre a címmeghajtó áramkörök a belső címszámlálóregiszter tartalma alapján generálják. A címszámláló tartalma a megjelenítendő karakter képernyőn lévő helyével (a pillanatnyi megjelenítési hellyel) szoros kapcsolatban van. A képinformációtár a kiolvasott karakterkódot a karaktergenerátor bemenetére címként kapuzza, majd onnan kiolvasható a karakterkontúr.



6.31. ÁBRA. Az FDC programozása. A folyamatábra a szektorírási és szektorolvasási műveleteket mutatja



6.32. ÁBRA Az LSI fekete-fehér és színes tv-monitos vezérlőáramkörének blokkvázlata



6.33. ÁBRA. A karaktermegjelenítés elvének a vázlata

A videojel generálásához azonban csak a képernyőre az adott pillanatban rajzolandó sor adatai szükségesek, így a karaktergenerátor adatának kiolvasásához a cím az alábbiak szerint képezhető:

KGCÍM = KÓD + KARSORCÍM

A KARSORCÍM-et a képernyőre rajzolandó sorral összefüggésben adja ki a videojel vezérlését végző logikai áramkör. Sorváltáskor ezt a címet megnöveli. A karakterek sorcímének maximális értéke az egy karakter megjelenítésénél alkalmazott rasztertől függ. Az ábrán bemutatott példán 5×7-es raszter látható, a maximális sorszám tehát 7. Az ismertetendő VDG belső karaktergenerátora 5×7-es karaktereket tartalmaz, a külső generátorban 7×9-es karaktereket is használhatunk.

A KGCÍM helyről kiolvasott adat (a karakterkontúr egy sora) a párhuzamos–soros átalakító léptetőregiszterébe kerül. Ebbe egy sor megjelenítése alatt egész képernyősornyi adatot ír be a VDG, azaz annyi adatot, ahány karakter ábrázolható a képernyőn egy sorban. A párhuzamos–soros átalakító léptetőregiszteréből az adatokat a képernyőre rajzolás ütemében (kb. 6 MHz) lépteti ki a vezérlő órajel. A kilépő adatokat kép- és sorszinkronjellel (és színinformációval – ezt a 6.33. ábrán nem tüntettük fel) látja el és erősíti a videojelgenerátor-egység.

A 6.32. ábrán látható VDG üzemmódjai három csoportba sorolhatók:

- alfanumerikus üzemmódok,
- félgrafikus (szemigrafikus) üzemmódok,
- grafikus üzemmódok.

Az *alfanumerikus üzemmódokban* a képernyőn 16 sor és soronként 32 karakter jeleníthető meg. Egy karakterhez 8×12 raszterpontból álló karaktermező tartozik, amelybe a belső karaktergenerátor használatakor 5×7-es mátrix pontjaiból áll össze a karakter (tehát a karakterek között három üres raszterpont, és a sorok között öt üres raszterpont marad). A VDG négyféle alfanumerikus üzemmódban programozható:

- belső karaktergenerátort használó,
- belső karaktergenerátort használó inverz megjelenítésű,
- külső karaktergenerátort használó,
- külső karaktergenerátort használó inverz megjelenítésű.

Az üzemmódok megadásához a BELSŐ/KÜLSŐ és az INVERZ vezérlő bemenetekre kell megfelelő jelet adni (6.32. ábra). Az inverz megjelenítés során a 8×12 raszteremező háttérszíne „világít", s a karakterkontúr pontjai sötétek. A két vezérlőjel karakterenként is váltható. A belső karaktergenerátor 64 karaktert tartalmaz. Így a karakterkód hatbites. A képinformációtár byte-szervezés esetén két szabad bitet tartalmaz. A két szabad bit beköthető a VDG-nek az INVERZ és a KÜLSŐ/BELSŐ bemenetére. Külső karaktergenerátor esetén 64-nél nagyobb karakterkészlet is alkalmazható.

A félgrafikus üzemmód kétféle lehet:

- félgrafikus négy elemrészes és
- félgrafikus hat elemrészes.

Félgrafikus módban az ábramezők száma az alfanumerikushoz hasonlóan 16 sor× ×32 mező. A VDG az ábraelemeket 8×12-es raszterben rajzolja fel, az ábraelemek élei tehát érintkezhetnek egymással, összefüggő vonalas ábra is kialakítható. A négy elemrészes módban a 8×12-es elemmező 2×2 elemrészt tartalmaz (elemrészenként 4×6 pont). Ezek az elemrészek külön „kigyújthatók", de csak azonos színűek lehetnek. Az elemrészek „kigyújtását" a képinformációtárból olvasott adat egy-egy bitje vezéreli. Elemrészen belül azonban a VDG a 4×6 pontot egységesen kezeli. A hat

elemrészes módban az elemmezőt 2×3 elemrészre bontja a vezérlő (elemrészenként 4×4 raszterponttal). Félgrafikus módban a négy, ill. a hat elemrészes váltást az INVERZ bemenetre adott jel vezérli. Az alfanumerikus és a félgrafikus módok közötti váltás karakterenként is megvalósítható az A/S (alfanumerikus/szemigrafikus) bemenetre adott jellel. Ezt a jelet az INVERZ jelhez hasonlóan a külső képinformációtár egyik bitje is előállíthatja. Az alfanumerikus és a félgrafikus módok közötti rugalmas váltás feliratos vonalas ábrák könnyű megjelenítését teszi lehetővé. Ezeket a módokat együttesen minor módoknak nevezik.

A *grafikus módok* csoportosíthatók

- két színt használó módok (egy bit/ábraelem),
- négy színt használó módok (két bit/ábraelem).

A grafikus módok mindkét csoportja négyféle képfelbontást tesz lehetővé (6.9. táblázat). A táblázatban lévő ábraelemek címezhetők, a felbontástól függően egy vagy több raszterpontot jelentenek. A több raszterpontból álló ábraelem pontjait azonban a VDG egységesen kezeli, tehát a pontok vagy azonos színűek, vagy mindegyik sötét. A tár mérete a grafikus módok függvénye. A 6.9. táblázat 6. és 7. grafikus módjához szükséges tár címtartománya nagyobb, mint amit a VDG tizenkét címbittel megcímezhet. A hiányzó címbit külső áramkörökkel állítható elő.

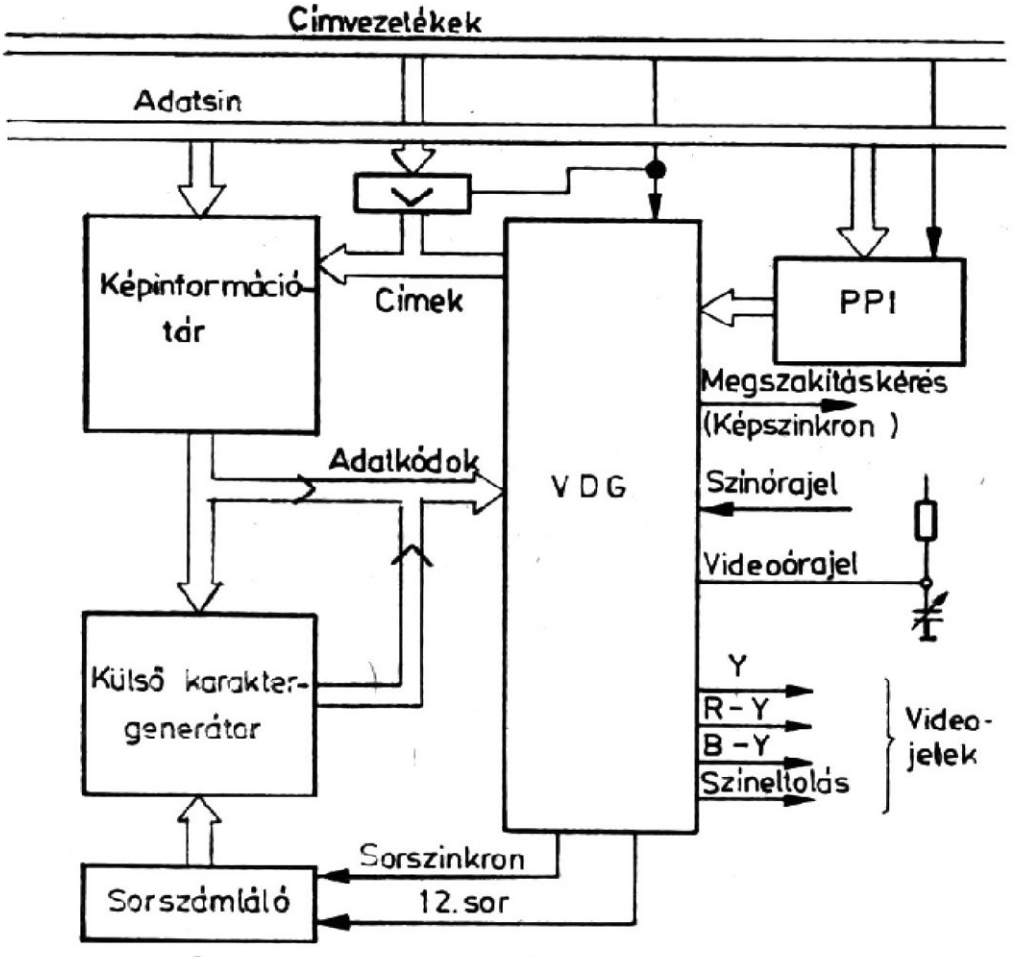
6.9. TÁBLÁZAT. A VDG üzemmódjai

Üzemmód	Rövid jellemzése	Tárkapacitás (képinformációtár)
Alfanumerikus-belső Alfanumerikus-belső-inverz	32×16 rasztermező mező: 8×12 raszterpont karakterek: 5×7 raszterpont	512×(6...8 bit)
Alfanumerikus-külső Alfanumerikus-külső-inverz	32×16 rasztermező mező: 8×12 raszterpont karakterek: 5×7 vagy 7×9 raszterpont	512×(6...8 bit)
Félgrafikus négy elemrész	32×16 elemmező 2×2 elemrész elemenként nyolc szín és fekete	512×(4...7 bit)
Félgrafikus hat elemrész	32×16 elemmező 2×3 elemrész elemenként négy szín és fekete	512×(6...8 bit)
0. grafikus	64×64 elem, négy szín	1 k×(8 bit)
1. grafikus	128×64 elem, két szín	1 k×(8 bit)
2. grafikus	128×64 elem, négy szín	2 k×(8 bit)
3. grafikus	128×96 elem, két szín	1,5 k×(8 bit)
4. grafikus	128×96 elem, négy szín	3 k×(8 bit)
5. grafikus	256×96 elem, két szín	3 k×(8 bit)
6. grafikus	128×192 elem, négy szín	6 k×(8 bit)
7. grafikus	256×192 elem, két szín	6 k×(8 bit)

A grafikus módokat major módoknak nevezik. A major és a minor módok közötti váltás csak akkor engedélyezett, ha a rasztersorszám 12 többszöröse (ez felel meg az alfanumerikus karaktermezők utolsó rasztersorának). A váltás engedélyezésére a VDG kiadja a 12. SORVÁLT jelet. A váltás az A/G (alfanumerikus/grafikus) bemenetre adott jellel vezérelhető. A grafikus módokat a GM bemeneteken lehet megadni.

Az üzemmódok ismertetése után röviden leírjuk a VDG és a mikroszámítógép kapcsolatát (6.34. ábra). A képinformációtárat a mikroprocesszor tölti fel a megjelenítendő adatokkal, a VDG pedig olvassa a tárat. A képinformációtár töltése, a címbitek használata csak a „sorvisszafutás", ill. a „képvisszafutás" alatt, vagyis a katódsugár kioltási ideje alatt engedélyezett. Az ábrán a képvisszafutásról megszakításkérés útján értesül a mikroszámítógép. A képinformációtár töltése alatt a megjelenítést letiltja a TÁRHOZZÁFÉRÉS jel. Erre akkor van szükség, ha a töltést valamilyen oknál fogva nem a katódsugár kioltásának az idején végzi a mikroprocesszor. A VDG az INVERZ, A/S, BELSŐ/KÜLSŐ, A/G, SZÍNVÁLTÁS és a három GM jellel programozható. Ezeknek a jeleknek egy része a képinformációtár szabad bitjeiben helyezhető el, másik részük egy programozható perifériaillesztő egységgel, vagy egy egyszerű programmal beírható regiszterrel adható meg. Külső karaktergenerátor esetén a karaktergenerátor címét a képinformációtárból kiolvasott karakterkód és a sorszinkronjelek számlálását végző sorszámláló kimenetén megjelenő jelek adják. A számláló a 12. SORVÁLT jellel nullázható.

Grafikus üzemmódok esetén a karaktergenerátorra nincs szükség. Ekkor a képinformációtár minden bitje (két szín használata) vagy minden bitpárja (négy szín használata) felel meg egy-egy ábraelemnek. Az ábraelemek adatait a VDG byteonként olvassa be, tehát egy-egy byte nyolc, ill. négy ábraelem adatait tartalmazza.



6.34. ÁBRA. A VDG és a mikroszámítógép kapcsolata

6.10. TÁBLÁZAT. Színmegadás a különböző üzemmódokban

Alfanumerikus és két színt használó grafikus mód	Színváltás jel	Világos ábraelem színe
	0	zöld
	1	égszínkék

Négy elemrészt használó félgrafikus mód	Adatbitek			Világos ábraelem színe
	D4	D5	D6	
	0	0	0	zöld
	0	0	1	sárga
	0	1	0	világoskék
	0	1	1	piros
	1	0	0	kék
	1	0	1	égszínkék
	1	1	0	vörös
	1	1	1	narancs

Hat elemrészt használó félgrafikus mód	Színváltás jel	Adatbitek		Világos ábraelem színe
		D6	D7	
	0	0	0	zöld
	0	0	1	sárga
	0	1	0	világoskék
	0	1	1	piros
	1	0	0	kék
	1	0	1	égszínkék
	1	1	0	vörös
	1	1	1	narancs

Négy színt használó grafikus mód	Színváltás jel	Páratlan bit	Páros bit	Világos ábraelem színe
	0	0	0	zöld
	0	0	1	sárga
	0	1	0	világoskék
	0	1	1	piros
	1	0	0	kék
	1	0	1	égszínkék
	1	1	0	vörös
	1	1	1	narancs

• A SZÍNVÁLTÁS bemenetre adott jellel módosítható az ábra színe. A 6.10. táblázatban foglaltuk össze a színmegadás jeleit.

A VDG két órajelet használ. Az egyik a VIDEOÓRA-jel, ehhez a VDG-nek belső órajel-generátora van. Az órajel kívülről egy *RC* taggal beállítható. A színes tv-megjelenítés a SZÍNÓRA-jellel szemben szigorú stabilitási követelményeket támaszt. Ez az órajel a színinformáció segédvivő jele. Külső, rezgőkvarccal vezérelt órajel-generátorral kell előállítani.

A VDG a kimenetein az összetett videojelet Y, és a színjelek különbségi jeleit (R—Y és B—Y) állítja elő. Kimenetére közvetlenül ilyen bemenő jeleket használó tv-monitor kapcsolható. A vezérlőegység RF modulátor alkalmazásával tv-vivőjellel látható el, így a színes tv-készülékek antennabemenetére is csatlakoztatható. Ebben az esetben a modulátor vezérlésére felhasználható a SZÍNELTOLÁS-jel.

6.5. PERIFÉRIAPROCESSZOROK

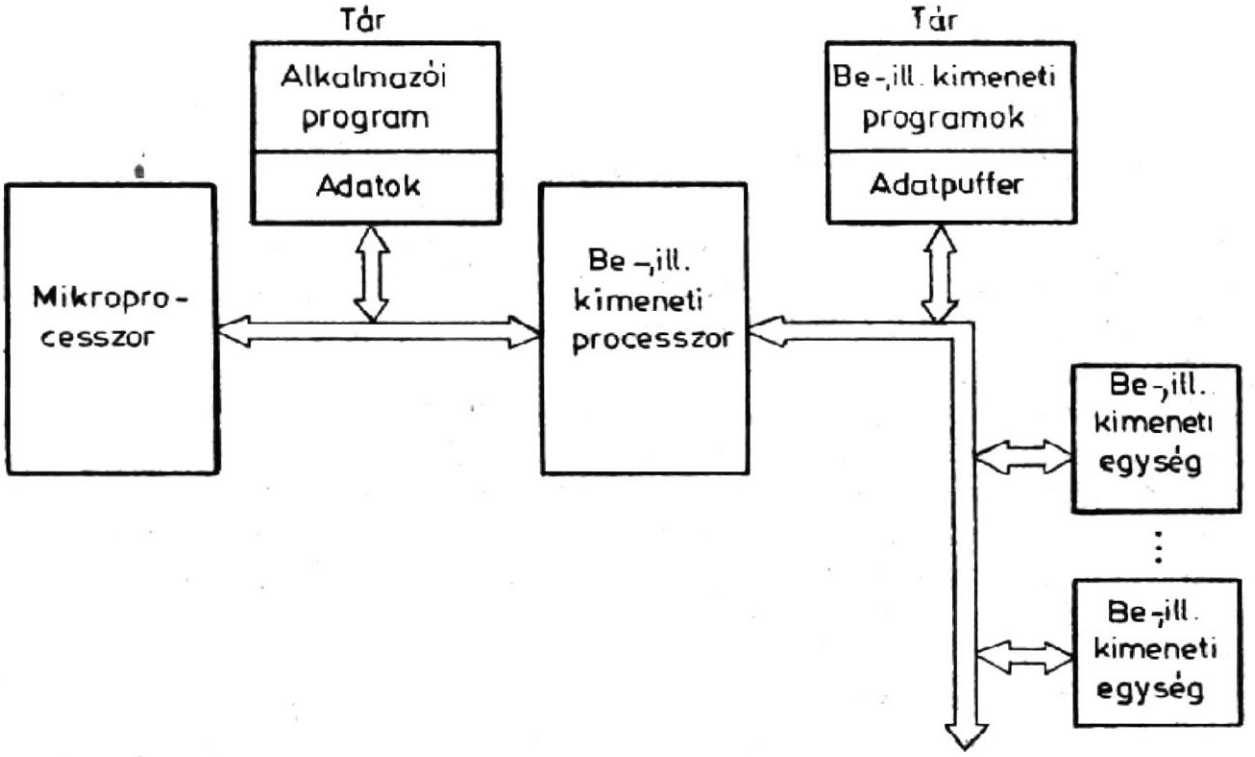
A mikroszámítógépek a lassúperifériák működtetéséhez általában a programozott adatátvitelt, gyorsperifériákhoz a DMA átvitelt használják. Ez utóbbinál ugyan a program és a processzor megkerülésével történik meg az adatok átvitele, mégis sok processzoridő megy veszendőbe, hiszen a gyorsperifériák hosszú időre lefoglalhatják a processzor adat-, cím- és vezérlősínjeit. Másrészt a perifériákat kiszolgáló programrutinok növelik a mikroprocesszorok programjainak bonyolultságát.

Ezeknek a hátrányoknak a kiküszöbölésére ún. „intelligens" be- és kimeneti vezérlőket használnak. Az ilyen be- és kimeneti vezérlőknek vagy processzoroknak LSI és VLSI technológiával megvalósított változatai is megjelentek [17]...[20]. A be- és kimeneti processzorok (IOP: **I**nput-**O**utput **P**rocesszor) jelentősen növelhetik a mikroprocesszoros rendszerek számítástechnikai kapacitását (teljesítményét) azzal, hogy a központi processzort teljesen leválasztják a perifériáktól (6.35. ábra) s csak akkor zavarják működését, amikor az elkerülhetetlen.

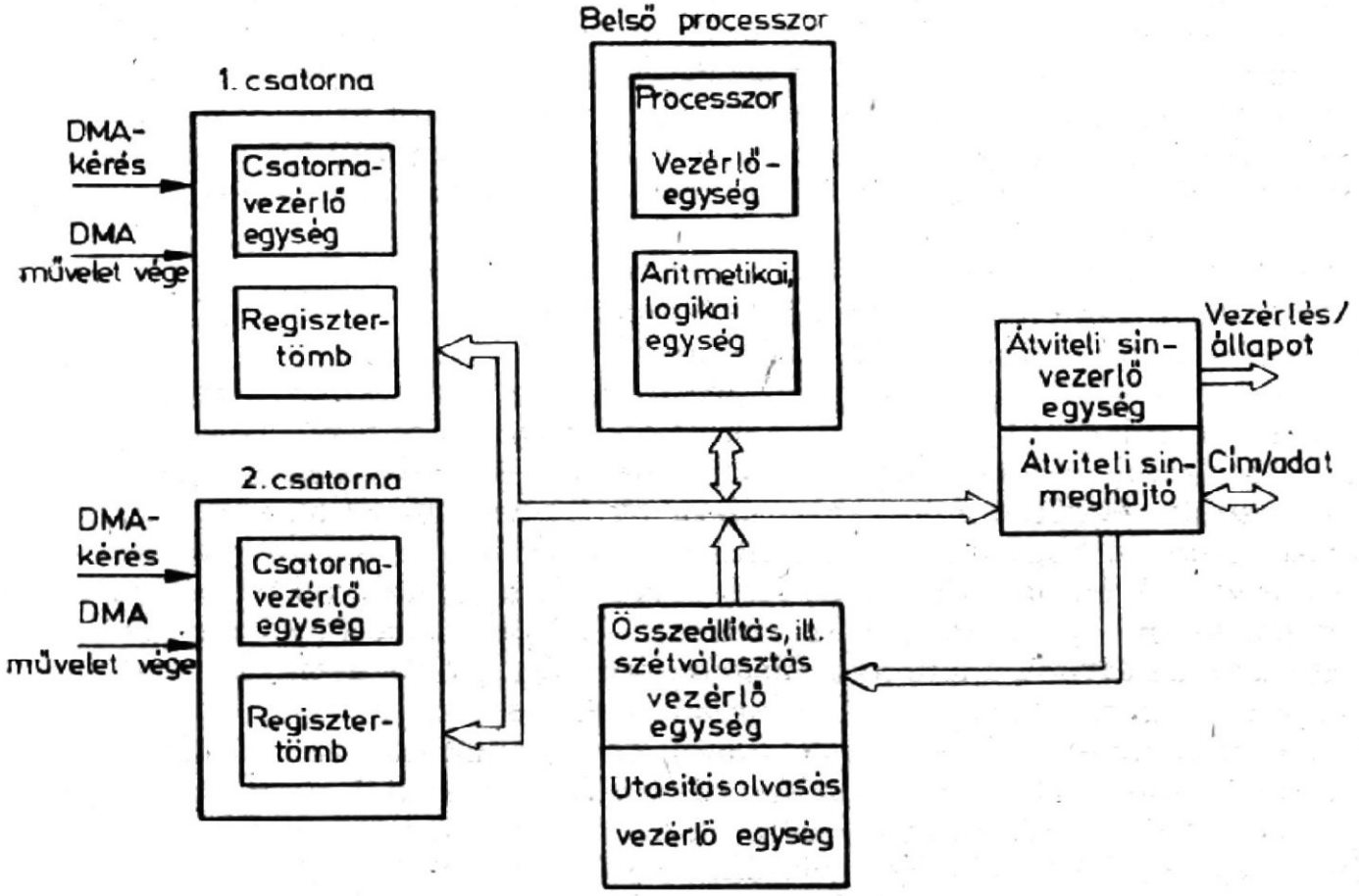
Az IOP-oknak a be- és kivitelekre optimalizált utasításkészletük van. Az utasításaikból – a mikroprocesszorokhoz hasonlóan – programok állíthatók össze. Ezeknek a programoknak a végrehajtását, azaz az utasítások értelmezését, elvégzését és egymás utáni láncolatát az IOP-ok a számítógépek központi egységeihez hasonlóan végzik. Programjukat a központi mikroprocesszor-egységgel szimultán hajtják végre, átviteli műveleteket, adatok előfeldolgozását, ellenőrzését stb. kezdeményeznek és bonyolítanak le. A beérkező és a kiviendő adatokat a perifériaműveletekkel való szinkronizálás céljából közbülső adatpufferelléssel átmenetileg tárolják.

A központi mikroprocesszor és az IOP között nagyon egyszerű és a perifériáktól független, minden perifériára nézve közös szoftverinterfész valósítható meg, mert az IOP programja a perifériáknak megfelelő be- és kimeneti műveleteket végzi el. Az IOP az azonos idejű alkalmazások esetén fontos, azonnali perifériakiszolgálási funkciókat is magára vállalhatja, így meggyorsítja ezek kiszolgálását, és tehermentesíti a központi mikroprocesszort.

A 6.36. ábrán egy LSI és egy VLSI technológiával készült, kétcsatornás IOP blokkvázlata látható [17]. Ez a programvezérelt átviteli egység mind 8-bites, mind pedig 16 bites mikroprocesszorokhoz használható. Alkalmas arra, hogy ún. táv (remote) sínvezérlőként működjön. Ebben az esetben a 6.35. ábra szerint a mikroszámítógép cím-, adat- és vezérlősínjétől elkülöníti az átviteli síneket. Használható azonban helyi (local) üzemmódban is. Ilyen üzemmódban az IOP a mikroprocesszorral közös sínekre dolgozik. A helyi üzemmódban ugyan az IOP alkalmazása nem jár azzal az előnnyel, hogy a ciklusloposos DMA műveletek nem lassítják a processzor műveleteit, sok felhasználói rendszerben mégis előnyösen alkalmazható ez a megoldás. Ez esetben is a be-, ill. kimeneti vezérlő nagy része kiváltható, így egyszerűsödik a hardver és csökkenthető a rendszer mérete. További előnyt jelent a mikroszámítógép programjának az egyszerűsödése, hiszen a sokféle perifériát kezelő különböző kiszolgáló-szubrutint az IOP programjaként lehet megírni. Pl. mágneslemezes tárak működtetésekor az IOP elvégezheti a file-összeállítási, adatlekérdezési, file-nyilvántartási műveleteket. De más perifériák esetén is, hasonló módon az



6.35. ÁBRA. Az IOP kapcsolata a mikroprocesszorral és a perifériákkal



6.36. ÁBRA. Az IOP belső felépítése [17]

IOP-ra háríthatók a vezérlési, átvitelprogramozási műveletek. Az IOP és a programja elvégezheti mindazokat a funkciókat, amelyeket az operációs rendszerek perifériakezelési műveletei igényelnek (l. 8. fejezet).

A helyi és a táv üzemmódokra mutat kapcsolási elrendezést a 6.37. és a 6.38. ábra. Helyi üzemmódban az IOP csak megfelelő vezérlő be- és kimenetekkel ellátott mikroprocesszorokkal képes együttműködni [17], [18]. A vezérlőjelek kialakítása általában az egy családot képező LSI és VLSI alkatelemek között egységes, ezért a közös sínekre kapcsolás feltétele teljesül. Az IOP más elemekkel (mikroprocesszorokkal) is együttműködhet, ekkor azonban nem biztos, hogy teljesülnek a közös sínekre kapcsolás feltételei.

Több IOP is kapcsolódhat közösen ugyanazokra a sínekre. Ilyenkor programmal adható meg az, hogy melyik a master és melyik a slave. A SÍNKÉRÉS, ENGEDÉLYEZÉS kivezetésre a master SÍNENGEDÉLYEZÉS, a slave SÍNKÉRÉS jelet továbbít. A közösen használt egy vezetéken először a slave rövid kérésimpulzust ad, amelyre válaszként szintén rövid impulzussal engedélyezésjel jön a mastertől. A sínműveletek végeztével egy harmadik impulzus, amelyet a slave generál, jelzi az átvitelek végét.

Az IOP-nak közös adat/címsíne van, ezért szükség van címregiszterre és adatsínmeghajtó és -vevő áramkörökre. A perifériák DMA kéréssel kezdeményezik az IOP átviteli műveleteit. Az IOP a programjához tartozó tárátviteleket, ill. az átviteli műveleteket a vezérlő-, cím- és adatsín jeleit használva bonyolítja le. Az átvitel kézfogásjeleit is a vezérlősín vonalai továbbítják. (A DMA kérés jel funkciója az IOP szempontjából tehát hasonló, mint a központi mikroprocesszor megszakításkérései).

Táv üzemmódban (6.38. ábra) az IOP mind a helyi sínekhez, mind pedig a mikroszámítógép vezérlő-, cím- és adatsínjeihez hozzáfér. A helyi sínekre csatlakoztatható a tár is, amelyben mind az IOP programja, mind pedig közbülső és átviteli adatok tárolhatók. A helyi sínekre csatlakoznak a perifériák. Az átviteli csatornák száma második IOP hozzáadásával bővíthető, ekkor az egyik IOP-t kell masternek, a másikat slave-nek programozni.

Az IOP műveleteit két fő részre oszthatjuk:

- programok végrehajtása (nem DMA ciklusok),
- átvitelek lebonyolítása (DMA ciklusok).

Programok végrehajtása során a processzor az utasításoknak megfelelően a tár és a csatornaregiszterek között adatok átvitelét, feldolgozását és vizsgálatát hajtja végre. A program célja egyrészt az, hogy előkészítse a belső regiszterekben (és a tárban) a később végrehajtandó DMA ciklusok lebonyolításához szükséges adatokat, másrészt pedig az, hogy a mikroszámítógépnek jelzéseket, állapotadatokat közöljön.

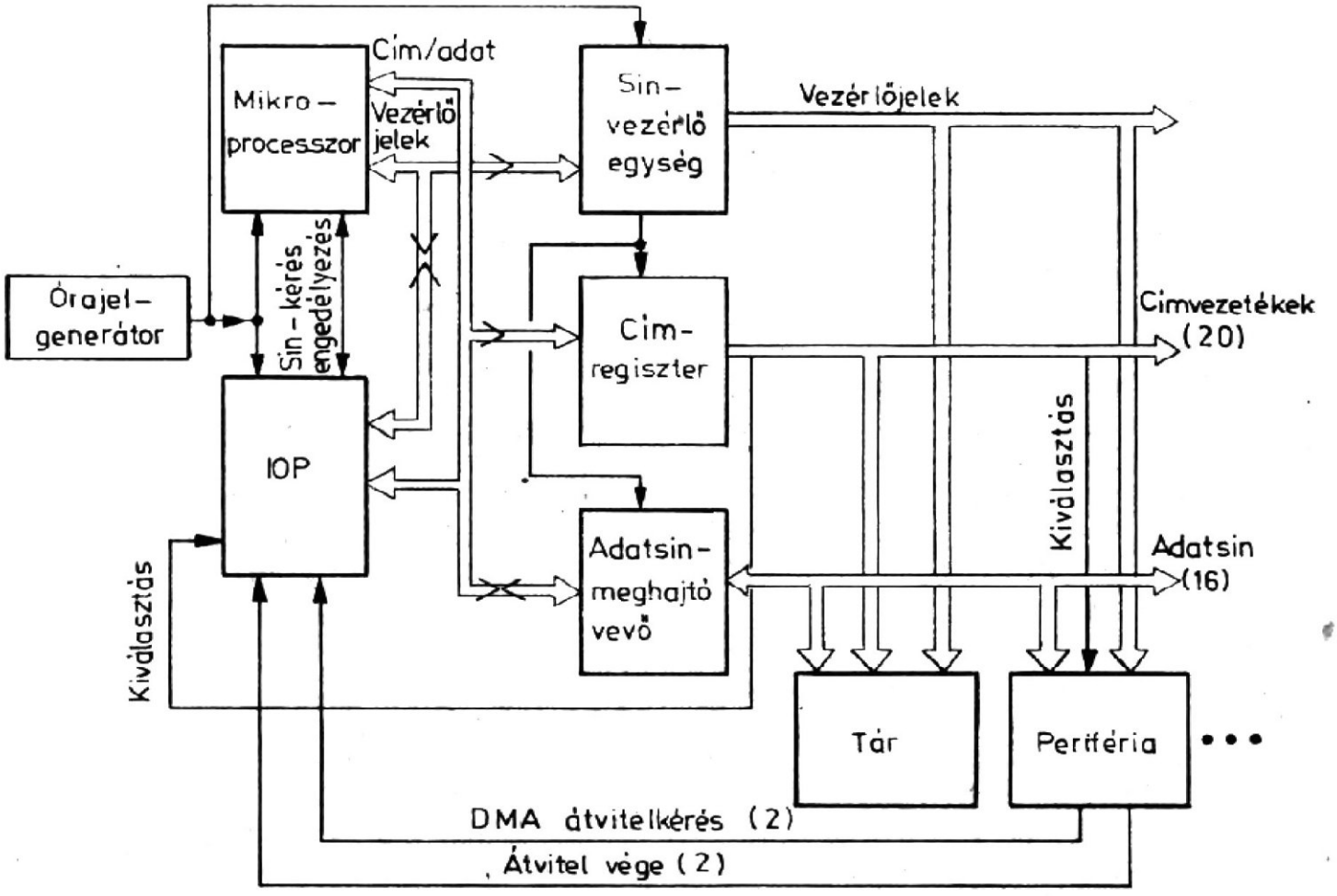
Az átvitelek lebonyolítását az IOP csatornái a regisztereikbe betöltött adatoknak megfelelően végzik. Ezek a regiszterek – később ismertetésükre visszatérünk – tartalmazzák az átvitelek lebonyolításához szükséges főbb adatokat:

- az átviteli parancskódot (CC: Command Code),
- az átviteli címeket (forrás és cél címe), az átviendő adatok számát.

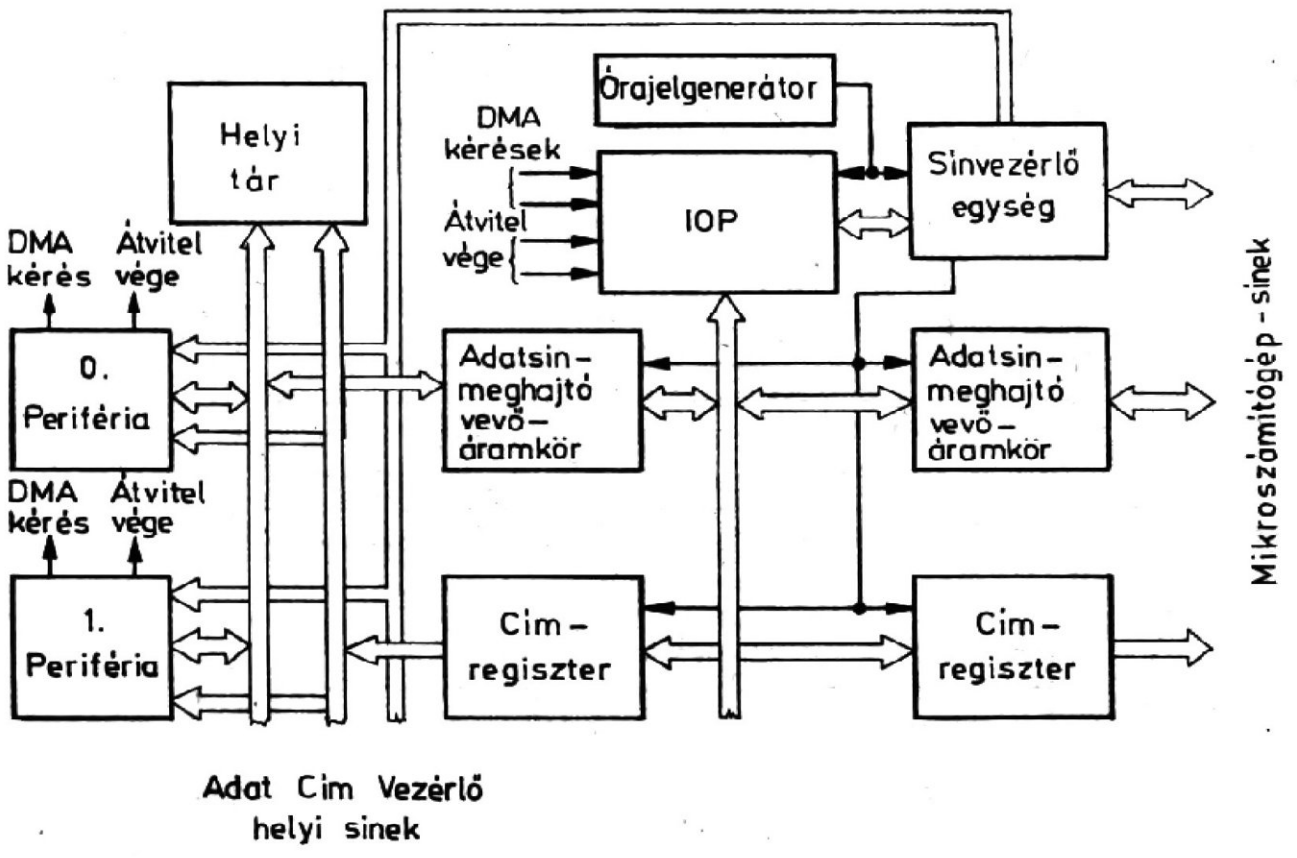
Az átvitelek adatainak a regiszterekbe töltése és az átvitelek indítása a mikroszámítógép által közölt feladatoknak (task), ill. az IOP programjának a kezdeményezésére történik meg. A DMA-műveletek elindítása után a periféria az adatátviteleket a saját sebességének megfelelően az IOP-nak a DMA átvitelkérés bemenetére adott jeleivel ütemezi.

Az átvitelek lebonyolítására az IOP két vezérlőjelet ad ki minden átviteli ciklus során. A vezérlőjeleken megjelenő kódok egyrészt jelzik a csatornaátviteli ciklusokat; ezek a ciklusok:

- 1. csatorna DMA átviteli ciklusa,



6.37. ÁBRA. Az IOP és a mikroprocesszor kapcsolata helyi üzemmódban [17]



6.38. ÁBRA. Az IOP és a mikroprocesszor kapcsolata táv üzemmódban [17]

- 2. csatorna DMA átviteli ciklusa,
- 1. csatorna nem DMA átviteli ciklusa,
- 2. csatorna nem DMA átviteli ciklusa,

másrészt pedig jelzik a helyi, ill. a mikroszámítógép sínjeinek a használatát és a sínen keresztül végrehajtandó műveleteket.

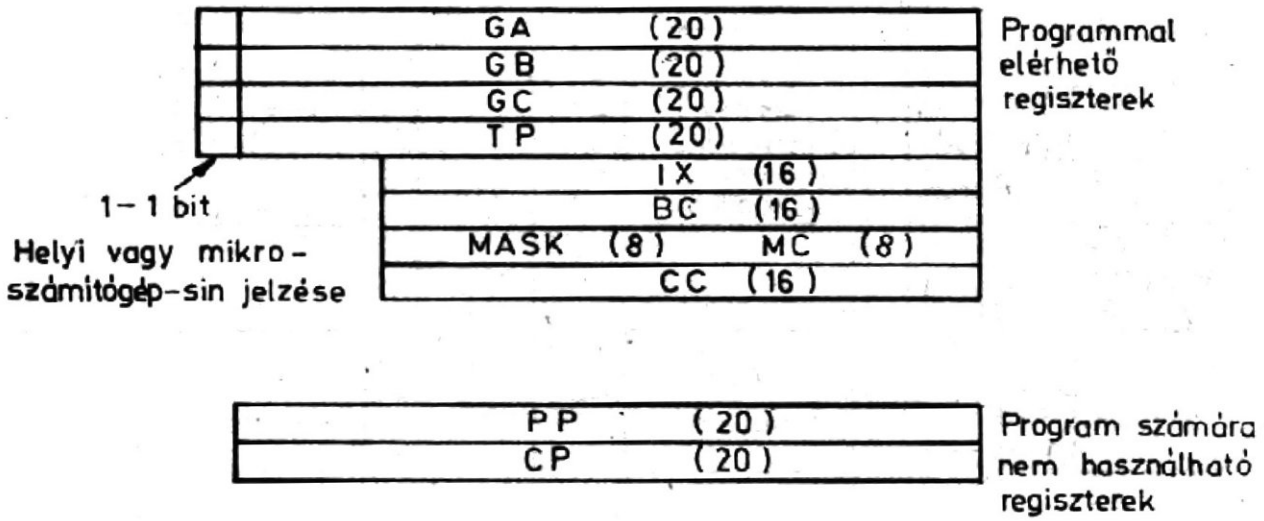
- utasításolvasás a helyi sínen,
- adatolvasás a helyi sínen,
- adatírás a helyi sínen,
- utasításolvasás a mikroszámítógép sínjén,
- adatolvasás a mikroszámítógép sínjén,
- adatírás a mikroszámítógép sínjén.

A vezérlőkódokat a sínvezérlő értelmezi és dekódolja, majd a megfelelő vezérlőjeleket továbbítja a helyi és a mikroszámítógép sínjei felé.

A példaként bemutatott IOP képes mind 8-bites, mind pedig 16 bites adatátvitelek végrehajtására. A mikroszámítógép és a helyi adatsínek szóhossza különbözhet is egymástól, a különböző hosszúságú adatok szükség szerinti összeállítását és szétválasztását az IOP elvégzi.

A két csatorna mindegyike belső regisztertömböt tartalmaz (6.39. ábra). A regiszterek funkcióit a következőkben ismertetjük.

A GA és GB 20 bites regiszterek DMA mutatóként (pointer) használhatók. Tartalmuk a helyi vagy a mikroprocesszor-sín címére vonatkozik (egy-egy bit jelzi, hogy melyik sín) és az átviteli forrás ill. a cél helyét jelzi. A GC regiszter tartalma egy – a tárban lévő- táblázatra mutathat rá, amelynek segítségével a DMA átvitel adatain átalakítások végezhetők, pl.: EBCDIC kód ASCII kóddá alakítása. A feladatmutató (TP: Task Pointer) regisztert a mikroszámítógép és az IOP szoftver közötti interfész használja. A TP tartalma jelzi az IOP-nak a programmal végrehajtandó feladatot. 16 bites regiszterek az IX indexregiszter, a BC byte-számláló regiszter, amelyben az átviteli blokk hossza adható meg, valamint a CC csatornavezérlő regiszter. A CC regiszter tartalma vezérli a DMA műveletek során végzendő átviteleket és átalakításokat. Továbbá, az átviteli adatokon összehasonlítási műveletek is végezhetők, az összehasonlítás paraméterét az MC regiszter, az esetleges maszkparamétert pedig a MASK regiszter tartalmazza.



6.39. ÁBRA. Az IOP belső csatornáinak a regisztertömbje

Az IOP utasításkészlete

Az utasításkészlet regisztereket feltöltő, a regiszterek tartalmát tárba töltő egyszerű aritmetikai és logikai műveleteket, szubrutinhívó, feltételes és feltétel nélküli elágazási utasításokat és bitmanipulációs műveleteket tartalmaz. Speciális utasítások vonatkoznak a megszakításkérések kezelésére, a DMA átvitelek kezdeményezésére, az adatsín bitszélességének a megadására stb. Az utasításkészlet rövid összefoglalását a 6.11. táblázat tartalmazza. Az utasítás operandusában szereplő M betű a megcímzett tárhelyet, a P betű a GA, GB, GC és TP 20-bites regiszterek egyikét jelöli, az R betű pedig bármelyik programmal elérhető regiszterre utal, az adatok azonban abban az esetben, ha R a paraméter, csak 16 bitesek.

6.11. TÁBLÁZAT. Az IOP utasításkészlete

Utasítás típusa	Emlékeztető szimbólum		Az utasítás leírása
Adatátvitelek			
Mutató-(pointer-) utasítások (20 bites adat)	**LPD**	P, M	Mutató töltése a tárból
	LPDI	P, I	Mutató töltése az utasításban lévő adattal
	MOVP	M, P	Mutató tartalmának a tárba írása
	MOVP	P, M	Mutató újratöltése
Regiszterutasítások (16 bites adat)	**MOV**	M, M	Tár–tár átvitel (source – destination)
	MOV	R, M	Regiszter töltése a tárból
	MOV	M, R	Regiszter tartalmának a tárba írása
	MOVI	R, I	Regiszter töltése az utasításban lévő adattal
	MOVI	M, I	Az utasításban lévő adat tárba töltése
Vezérlésátadó utasítások			
Szubrutinhívás	**CALL**		Feltétel nélküli szubrutinhívás
Elágazások	**JMP**		Feltétel nélküli elágazás
	JZ	M	Az ugrás feltétele (0, vagy nem 0) a tártartalom értéke
	JNZ	M	
	JZ	R	Az ugrás feltétele (0, vagy nem 0) a regiszter tartalom értéke
	JNZ	R	
	JBT		Bit vizsgálat az ugrás feltétele (IGAZ vagy HAMIS bit)
	JNBT		
	JMCE		A maszkolt összehasonlítás eredménye az ugrás feltétele (egyenlő, nem egyenlő)
	JMCNE		
Bitvizsgálat és manipuláció	**SET**		A kiválasztott bit *1* értékre, ill.
	CLR		*0* értékre állítása
	TEST		Bitvizsgálat

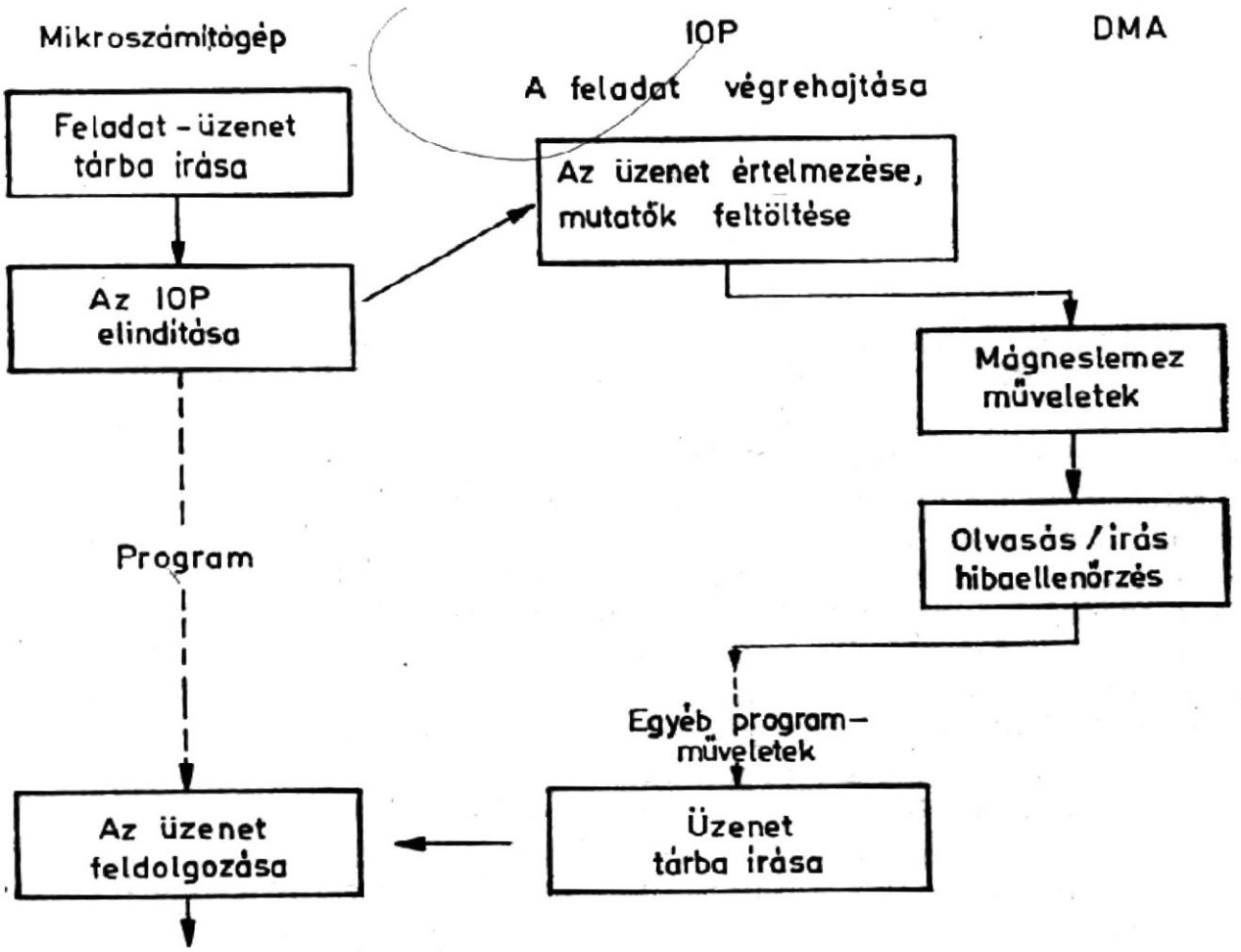
6.11. TÁBLÁZAT (folytatás)

Utasítás típusa	Emlékeztető szimbólum		Az utasítás leírása
Aritmetikai utasítások			
Növelés, csökkentés	**INC**	**M**	A tár tartalmának növelése
	DEC	**M**	A tár tartalmának csökkentése
	INC	**R**	Regiszter tartalmának növelése
	DEC	**R**	Regiszter tartalmának csökkentése
Összeadás	**ADDI**	**M, I**	Az utasításban lévő adatnak a tár tartalmához való hozzáadása
	ADDI	**R, I**	Az utasításban lévő adatnak a regiszter tartalmához való hozzáadása
	ADD	**M, R**	Regiszter tartalmának a tár tartalmához való hozzáadása
	ADD	**R, M**	A tár tartalmának a regiszter tartalmához való hozzáadása
Logikai ÉS	**ANDI**	**M, I**	Az operandusok értelmezése
	ANDI	**R, I**	ugyanaz, mint összeadásnál
	AND	**M, R**	
	AND	**R, M**	
Logikai VAGY	**ORI**	**M, I**	Az operandusok értelmezése
	ORI	**R, I**	ugyanaz, mint összeadásnál
	OR	**M, R**	
	OR	**R, M**	
Invertálás	**NOT**	**R**	Regiszter tartalmának invertálása
	NOT	**M**	A tár tartalmának invertálása
	NOT	**R, M**	A tár tartartalmának invertálása és regiszterbe töltése
Vezérlőutasítások	**HLT**		A csatornaművelet megállítása
	SINTR		„Megszakítás" bistabil bebillentése
	NOP		Nincs művelet, üres utasítás
	XFER		DMA átvitel megindítása
	WID		A „forrássín" és a „célsín" szóhosszának (8 vagy 16 bit) a megadása

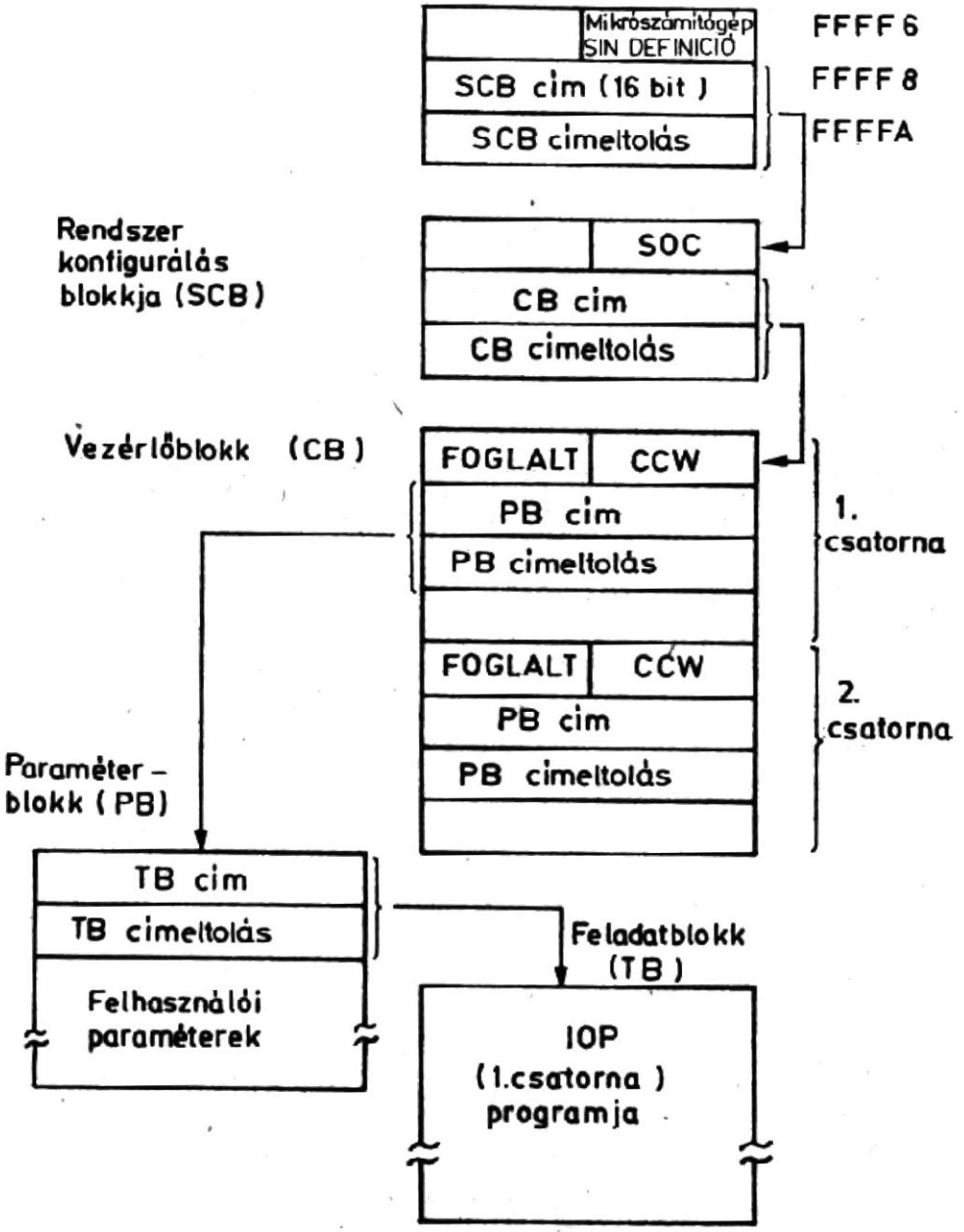
Az IOP és a mikroszámítógép közötti kommunikáció

A két processzor közötti szoftverinterfészt egy rugalmasan kezelhető, tárban tárolt adatstruktúra jelenti. A kommunikáció tehát a tárba beírt adatokkal történik. A mikroprocesszor az IOP programját az IOP által elérhető tárba helyezi végrehajtás céljából, majd egy hardver által kezelt jellel a *csatorna figyelem!* (CHANNEL ATTENTION) jellel kezdeményezi az IOP valamelyik csatornájának a műveletét. Hasonló eljárással üzen az IOP is a mikroszámítógépnek: üzeneteket helyez el a mikroszámítógép tárába. Az üzenet az IOP belső állapotát és a perifériák állapotát jellemzi. Ezután akár megszakításkéréssel, akár pedig a tárba elhelyezett jelzőbittel (-byte-tal) felszólítja a mikroszámítógépet az üzenet megvizsgálására. Az üzenetváltás vázlatát a 6.40. ábrán tüntettük fel. Az üzenetváltás adatstruktúrájára a 6.41. ábrán (amely a mikroszámítógép oldaláról az összes periféria részére egységes lehet) egy lehetséges példát mutatunk be. Az ábrán a kezdeti, bekapcsolás utáni rendszerkonfigurálási eljárást is feltüntettük. Az IOP a nullázás után, a mikroszámítógép *csatorna figyelem!* parancsára meghatározott kezdeti címről kezdi a működést (az FFFF6-os címtől). Itt a mikroszámítógép adatsínjének a szóhosszát megadó utasítást követi a *rendszerkonfigurálás* szoftver blokkjának (SCB: System Configuration Block) kezdőcíme. A cím 20 bites és így két szóban foglal helyet. Az első szó a 16 bites RC relatív címet tartalmazza, a második szó a CE címeltolás (relokálás) értékét. A tényleges címet a 20 bites SCB-t az alábbi összefüggés adja:

$$SCB = 2^4 \cdot CE + RC$$



6.40. ÁBRA. Az IOP és a mikroszámítógép közötti üzenetváltások vázlata



6.41. ÁBRA. A mikroszámítógép IOP üzenet adatstruktúrája [17]

Ezzel a címképzési eljárással a társzegmentálás, a programok tetszőleges szegmensbe töltése válik lehetővé, és minden további blokkhoz fordulásnál hasonlóan megy végbe. Az SCB első adata, a működési kód (SOC: System Operating Code) az IOP adatsínjeinek a szóhosszát (és két IOP esetén a master IOP-t) definiálja. A következő két 16 bites szóban a vezérlőblokk (CB: Control Block) kezdőcímének adatai találhatók. Az SCB-t csak a bekapcsolás utáni rendszerkonfiguráláshoz használjuk, a működés közben többször már nincs rá szükség. A CB adata a csatornavezérlő-szó (CCW: Channel Control Word). Ez határozza meg a csatorna műveleteit. A következő adat egy szoftver jelzőbyte, a csatorna foglalt/szabad állapotának a jelzésére. Az ezt követő két szóban a PB paraméterblokk kezdőcímének adatai találhatók. Az előző blokkoktól eltérően a CB vezérlőblokk a RAM-ban van, s a mikroszámítógép az IOP-nek az adatok átírásával jelölhet ki más és más műveleteket. A RAM-ban lévő paraméterblokk első adata (két szó) a feladatblokk kezdőcímére (TB: Task Block) mutat. A feladatblokk már az IOP által végrehajtandó programot tartalmazza. A paraméterblokk további paramétereket is tartalmazhat az IOP programjának a függvényében. Ide írhatók be az IOP válaszüzenetei is. Az IOP csatornáinak a program által nem használt két 20 bites regisztere a CB és a PB mutatója. Az IOP

a FOGLALT jelzőbyte-ot a működésének befejeztével törli (szabad állapot). A mikroszámítógép ennek a byte-nak a vizsgálatával, vagy a megszakításkéréssel értesül az IOP feladatának a teljesítéséről. Ezután, szükség szerint a mikroszámítógép újraindíthatja a be-, ill. kimeneti processzort a *csatorna figyelem!* hardverjellel.

A TB kivételével a blokkok a mikroszámítógép és az IOP által közösen használt tárrészben vannak. A TB az alkalmazástól függően a helyi sínre csatlakozó tárban (ROM-ban) is elhelyezhető.

Az ismertetett üzenetváltási eljárás nagy előnye, hogy az operációs rendszer részéről teljesen világos be-, ill. kimeneti kezelést tesz lehetővé. A PB adatait perifériáktól függetlenül egységesen kezelheti a mikroszámítógép operációs rendszere.

6.6. NAGYTÁVOLSÁGÚ ADATÁTVITEL

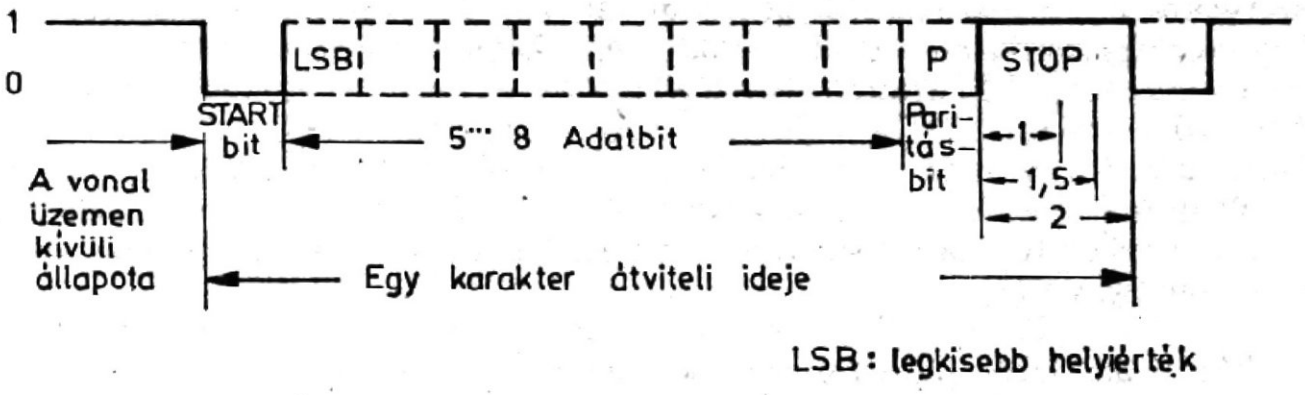
Az elektronikus készülékek és berendezések között gyakran szükséges digitális adatoknak a továbbítása. Az adattovábbítás információhordozói általában áram- vagy feszültségjelek. Az adatok továbbítása sorosan egy vezetékpáron, vagy párhuzamosan sok vezetéken keresztül történhet. Az átvitel lehet szinkron vagy aszinkron. Az előbbinél minden adat és bit megjelenési és továbbítási ideje pontosan előre definiált, míg az utóbbinál nem.

Párhuzamos átvitelek esetén az adatbiteket általában egy mintavételi (strobe) jel kíséri. Ez a jel közli a vevővel az adatok érvényességét. Soros átviteli vonalakon az egy adathoz, karakterhez tartozó biteket időben egymás után továbbítjuk.

A mikroprocesszoros berendezésekben az adatok általában párhuzamosan kerülnek átadásra egymás között. Ez az átviteli forma nagyobb sebességet eredményez. Amint azonban a készülékek közötti távolság, s így az adatátviteli távolság nő, nemcsak a vezetékek és a kábelezés költségei növekszenek, hanem a hosszú vezetéken jelentkező adatátviteli problémák miatt (zajviszonyok, teljesítménynövekedés, jelalaktorzulás stb.) az adó- és vevőáramkörök bonyolultsága is. A továbbiakban az ilyen „távoli" összeköttetéssel foglalkozunk.

A soros adatátvitelt nagyobb távolságok áthidalására használják. Itt az árviszonyok lehetővé teszik, hogy egy-egy meglehetősen bonyolult és drága adót és vevőt alkalmazzunk a vonalak két végén. Az adatokat, természetesen a vonal mindkét végén sorosból párhuzamossá és párhuzamosból sorossá kell átalakítani.

Az egyik gyakrabban használt soros adatátviteli eljárás az aszinkron adatátvitel. Az aszinkron karakterformátumot a 6.42. ábrán láthatjuk. A karakter átvitelét a START bit vezeti be, majd azt követi az 5, 6, 7 vagy 8 bit hosszúságú adat. Az adatot követheti egy paritásbit (páros vagy páratlan), és végül a karakterátvitelt a STOP jel zárja. A STOP jel minimális időtartama szabvány szerint lehet: egy, másfél vagy



6.42. ÁBRA. Az aszinkron adatátviteli karakter formátuma

két bitidő. Újabb karakter esetén csak ezután adható a következő START bit. A vevő és az adó közötti szinkronizálást a START bit megjelenése végzi, a további, adaton belüli együttfutást a mindkét oldalra azonosan megadott átviteli sebesség és adatformátum biztosítja. Az aszinkron átvitel hátránya a viszonylag nagy időveszteség, amit a START és STOP jelek átküldése jelent. Ez nyolcbites adatnál és a legrövidebb STOP jel mellett is 20%-nyi időveszteség.

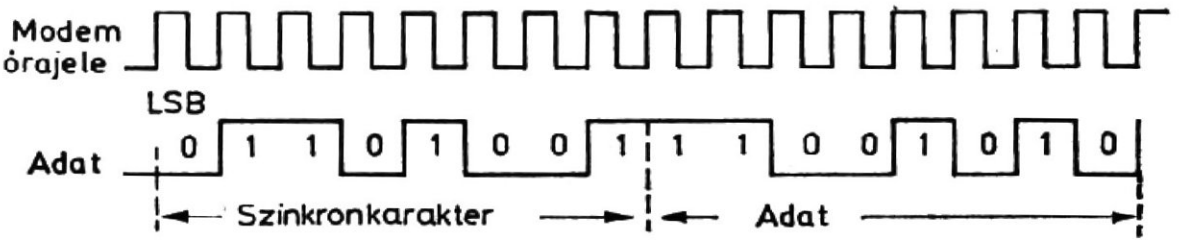
Szinkron adatátvitelnél az adatvezetéken megjelenő adatjel mellett órajel átvitelére is szükség van. Ez az órajel vagy külön vezetéken érkezik, vagy pedig modemek alkalmazása esetén a modulált jelből állítható elő. Az órajel biztosítja az adó és vevő közötti szinkron együttfutást. Az adatok közvetlenül egymást követik, tehát az aszinkron átvitel karakterenkénti időveszteségе így kiküszöbölhető. A karakterek kezdetének és végének a jelzésére itt nincs START és STOP jel, így más módszer szükséges az üzenet kezdetének a jelzésére. Az átviteli blokk elején ezért egy vagy több szinkronkaraktert kell küldeni (6.43. ábra). A szinkronkarakter jelváltásai (kódja) alapvetően különböznek a többitől, nehogy két másik karakter kezdeti és befejező bitjeiből is előálljon ez a jel. A szinkronkaraktert *sync-karakternek* nevezik. Ennek értéke az ASCII-kódban 10010110 (oktális 226). A vevők az átvitel előtt. ún. *sync-keresés* (sync search) módban várják a sync-karaktert. Nagyobb biztonság elérése céljából általában két sync-karakterrel (de néha hárommal is) vezetik be az adatblokkok átvitelét.

A blokkos adatátvitel szabályaira (a vezérlőkarakterekre, a blokkok hosszára, felépítésére stb.) több szabványeljárást dolgoztak ki. Hasonló módon szabványok írják elő a modemek alkalmazására és a modemekben használatos modulációs eljárásokra vonatkozó szabályokat. Ezekkel bővebben a kommunikációs módszereket és eszközöket tárgyaló szakirodalom (pl.: a [8]) foglalkozik.

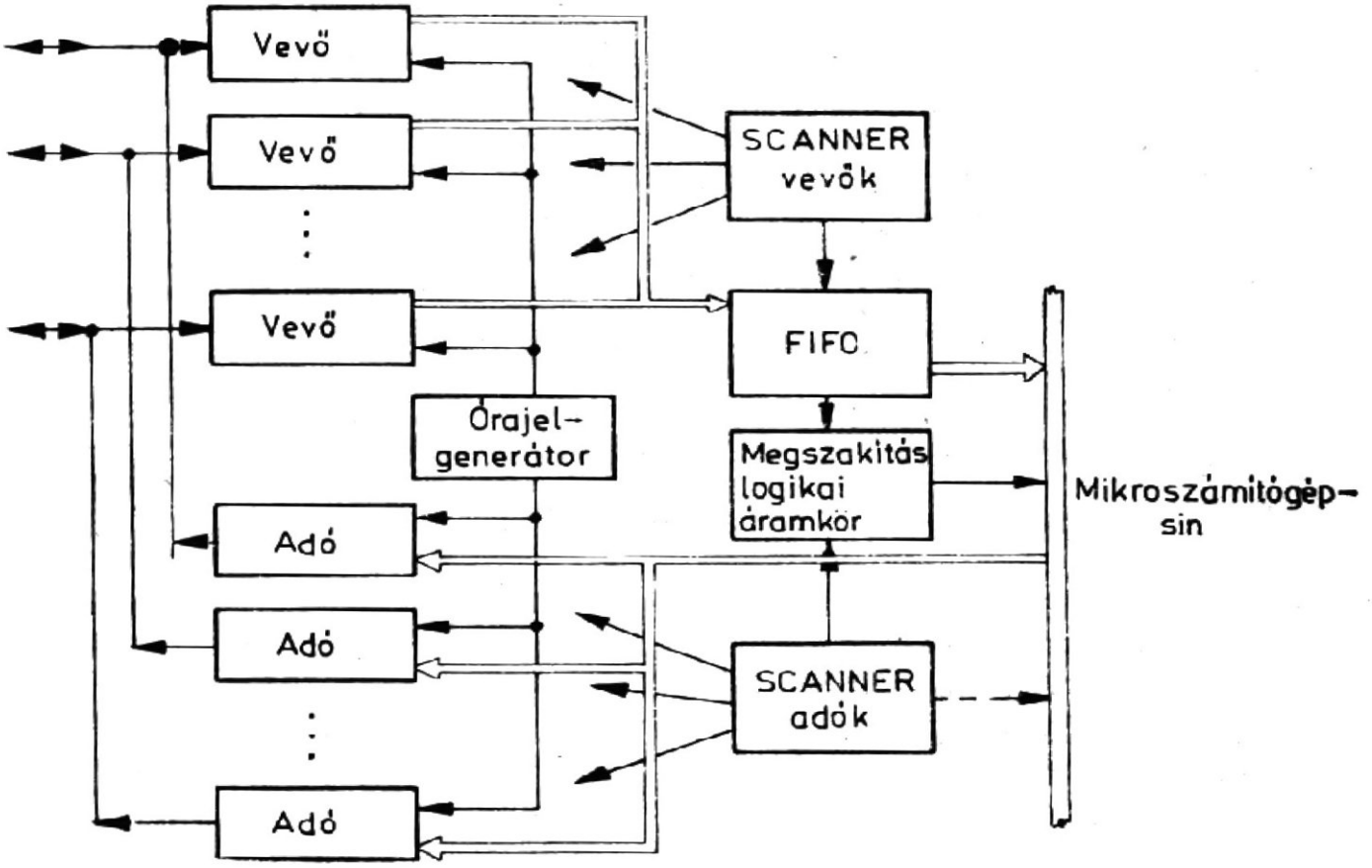
A soros adatátvitelek hardvereszközei között fellelhetők a programozható LSI soros interfészelemek is. Ezek között univerzális szinkron (USRT: Universal Synchron Receiver-Transmitter), aszinkron (UART) és szinkron/aszinkron (USART) üzemmódban is működő elemek ismeretesek. Egy tipikus programozható USART részletesebb ismertetésével a periféria-áramkörök leírása során (l. a 6.2. szakaszt) foglalkoztunk.

Az USART ismertetése során a programozás kérdéseire is kitértünk. Itt érintettük azt, hogy a mikroprocesszor programja hogyan szolgálja ki az USART-ot. Ha a mikroszámítógépes rendszer sok soros adatátviteli vonallal kapcsolódik a külvilághoz, az egyes vonalak kiszolgálását megszakításkérésekkel célszerű megoldani. A megszakításkérések megszervezése, kiszolgálása mind a hardver, mind pedig a szoftver kidolgozása során többletmunkát jelent. További hátrányként jelentkezhet a sok megszakításkérés kezelésével járó időveszteség. Multiplexerek alkalmazásával csökkenthető a hardver (az órajel-generátorok, egyes interfészelemek közösen használhatók, egyszerűsödik a megszakításkezelő logikai áramkör felépítése), és egyszerűsíthető a megszakítások programmal való kiszolgálása is. A 6.44. ábrán egy multiplexer egyszerűsített vázlata látható. Az ábrán az áttekinthetőség kedvéért külön rajzoltuk az egy LSI áramkörben lévő adókat és vevőket. A blokkvázlat új elemei a lekérdező-áramkörök (SCANNER) és a soros puffertár (FIFO). A vevők lekérdező logikai áramköre egymás után végigkérdezi a vevőket, s amelyik kész, azaz adatot tud szolgáltatni, annak az adatát, állapotinformációját és sorszámát a FIFO-ba beírja. A megérkezett adatról ezután a megszakításkérések kezelőegysége értesíti a mikroszámítógépet [8].

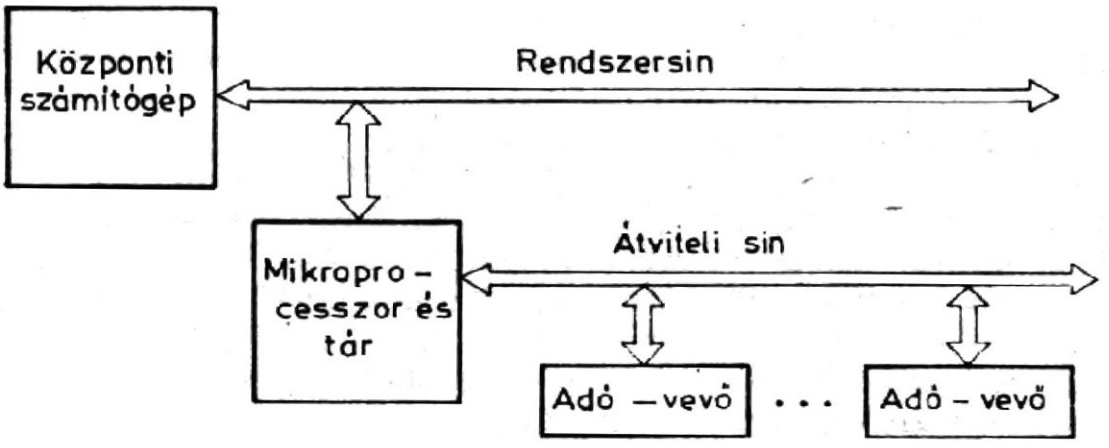
Az adókat a SCANNER hasonlóan lekérdezi. A készen lévő adónál (ez az adó már elküldte az előző adatot) megáll, s megszakításkérést küld a mikroszámítógép-



6.43. ÁBRA. A szinkron adatátvitel



6.44. ÁBRA. Egy soros vonali multiplexer blokkvázlata [8]



6.45. ÁBRA. Egy mikroprocesszorral vezérelt multiplexer vázlata [8]

nek, amelyik a SCANNER címe alapján azonosítja az adásra készen álló adót, és új adatot küld neki.

A multiplexerek vezérlése és a SCANNER-ek nemcsak hardverrel, hanem mikroprocesszoros vezérléssel is megoldhatók. Ez a megoldás nemcsak az adatok adásának és vételének a vezérlését látja el, hanem a mikroprocesszoros egység a beérkezett és kiküldendő adatokat ellenőrizheti, bizonyos előfeldolgozást végezhet rajtuk, ill. blokkokat állíthat össze. Egy ilyen mikroprocesszorral vezérelt multiplexer (intelligens vezérlő) vázlatát a 6.45. ábrán láthatjuk. A mikroprocesszor és a „főgép" közötti kapcsolat akár megszakításkérésekkel és programozott átvitel útján, akár pedig DMA átvitellel valósulhat meg [21], [22]. Az intelligens multiplexer-vezérlőben jól használható az előző szakaszban ismertetett IOP.

A soros adatátvitelekhez fontos kiegészítő műveletként tartozik az átvitel helyességének az ellenőrzése és az átviteli hibák felderítése. A karakterek átvitelének az ellenőrzése legegyszerűbben paritásbit hozzáadásával érhető el. A paritásbit a karakter *1* értékű bitjeinek a számát egészíti ki párosra vagy páratlanra. Előző esetben páros paritásbitről, az utóbbi esetben páratlan paritásbitről beszélünk. Az adó gondoskodik a paritásbit hozzáadásáról, míg a vevőoldalon ellenőrizhető a beérkezett adat *1* értékű bitjeinek a száma. Eltérés esetén az átvitel hibás. Egy paritásbittel az egy bitnyi eltérések (adattorzulások) észlelhetők. Két bitben jelentkező adattorzulás esetén a vevő oldali adat paritása helyes lehet.

További hibavédelmet biztosít, ha egy karaktersorozatot egy további karakterrel látunk el, amelyet a karaktersorozat azonos sorszámú bitjeihez képezett paritásbitek alkotnak.

Például:

001110101	1. karakter
101101100	2. karakter
100010111	3. karakter
110101111	4. karakter
111100010	5. karakter
110111100	Paritáskarakter

Ebben a karaktersorozatban minden karaktert páratlan paritásbittel láttunk el, az öt karakterből álló sorozatot egy hatodik, páratlan paritáskarakter zárja. A karakterenkénti paritásbit neve keresztparitás, a sorozathoz adott kiegészítő karakter neve hosszanti paritáskód. A számítástechnikában gyakori eset, hogy a hosszanti paritáskódot nem a paritásbitnek megfelelő eljárással, hanem az őt megelőző adatok összeadásával képezik (a túlcsordulások mellőzésével). Ez esetben a kód neve ellenőrzőösszeg (vagy kontroll-szumma).

A kereszt és hosszanti paritás-ellenőrzéssel közel megegyező hibavédelmet nyújtanak a ciklikus redundancia kódokat (CRC: Cyclic Redundacy Code) használó ellenőrzések. A CRC-t többrészes, visszacsatolt léptetőregiszterekben állítják elő (6.46. ábra). A különböző szabványok megadják a CRC képzésének a szabályait, a léptetőregiszterek bitjei közötti visszacsatolások számát és helyét. Intelligens multiplexerekben természetesen programmal is előállítható a CRC értéke.

Az adatátvitel különböző funkcióinak az ellátásához, a funkcionális blokkok építéséhez is alkalmazhatók a nagy integráltságú áramkörök:

- kódkonverterek,
- adó/vevőáramkörök,
- modulátor–demodulátor áramkörök,
- aktív szűrők,

6.46. ÁBRA. Egy CRC generátor vázlatos rajza [8]

- multiplexerek, demultiplexerek,
- átviteli sebesség órajelének generátora,
- komparátorok stb.

A kommunikáció LSI eszközeinek a választéka is állandóan bővül [23], [24].

IRODALOM

[1] *Wackerly, J. F.:* Microprocessor input/output architecture. Computer, Feb. 1977 26–33.

[2] *Weissberger, A. J.:* Peripheral interfacing with microprocessors. Computer Design, Feb. 1977. 105–112.

[3] *Soucek, B.:* Microprocessors and microcomputers. Wiley-Interscience Publication, New York, 1976.

[4] Intel data book. Intel Corp., 1978.

[5] Signetics integrated circuits. Data handbook, Signetics, Philips, 1978.

[6] The TTL data book for design engineers (CC 411). Texas Instruments Inc., 1973.

[7] Intel 8080 microcomputer systems. User's manual. Intel Corp., 1975.

[8] *McNamara, J. E.:* Technical aspects of data communication. Digital Equipment Corp., 1978.

[9] *Klingman, E. E.:* Microprocessor systems design. Prentice Hall Inc., Englewood Cliffs, 1977.

[10] *Hilburn, J. L.–Julich, P. N.:* Microcomputers/microprocessors: hardware, software and application. Prentice-Hall Inc. Englewood Cliffs, 1976.

[11] Single chip controllers. Systems international, Jan. 1980. 29–31.

[12] *Weiner, R. J.:* Rely on floppy-disc controller IC to handle „any density" floppy drives. Electronic Design, Feb. 15, 1979. 72–76.

[13] *Santony, A.:* Peripheral chips add functions – and problems – to μP systems. Electronic Design, May 10, 1977. 32–38.

[14] *LeBoss, B.:* Chip programs most CRT functions. Electronics, Feb. 17, 1977. 119–120.

[15] S 68047 Video display generator. American Microsystems Inc., Jan. 1979.

[16] 68047 Video display generator. Application note. American Microsystems Inc., 1979.

[17] 8089 8/16-bit HMOS I/0 processor. Intel Corp., July 1979.

[18] *Stamm, D.–Kopec, S.–McCormick, B.:* Free the μC's CPU from I/0 hassles with a special I/0 processor. Electronic Design, March 29, 1979. 102–106.

[19] *Hyde, J.:* Using the Intel 8741 universal peripheral interface. Microprocessors and microcystems, Aug. 1978. 236–240.

[20] *Philips, D.–Goodman, A.:* Slave microcomputer lightens main microprocessor load. Electronics, July 7, 1977. 109–112.

[21] *Urkumyan, N.:* Cost-effective 64-channel multiplexer system. Computer Design, July 1979. 114–119.
[22] *Lesea, A.–Urkumyan, N.:* Multiplexer system reduces cost of terminal interfacing. Computer Design, Aug. 1977. 109–113.
[23] ICs for communications. Electronic Design, Apr. 12, 1979. 68–143.
[24] *Adlerstein, S.:* LSI building bloks will channel the rising tide of data transmission. Electronic Design, April 26, 1979. 58–63.

7. SÍNRENDSZEREK

Ha egy digitális rendszert egy előlényhez próbáljuk hasonlítani, akkor a processzort és a hozzá csatlakozó tárat a rendszer „agyának", a bemeneteket „érzékszerveknek", a parancsokat végrehajtó kimeneteket pedig „végtagoknak" tekinthetjük. Az egyes funkcionális egységeket és részegységeket összekötő sínrendszer pedig az idegrendszernek, az idegpályáknak felelhetne meg. Még egy kicsit tovább játszva az előbbi hasonlattal, a sínek hibátlan működése az egész rendszer helyes működésének alapfeltétele, zavaraik, „betegségük" pedig nemcsak működési problémákat okoz, hanem – hasonlóan az idegrendszer zavaraihoz – a szimptómák a szervezet bármelyik részén megjelenhetnek és ezért nehéz is őket diagnosztizálni.

Ezért a sínt a mikroprocesszoros rendszer fontos részének, alapvető architekturális erőforrásának kell tekintenünk. Nem megfelelő tervezés esetén, figyelembe véve a rendszer növekvő komplexitását, a sín könnyen lehet az egész rendszerben – annak tulajdonságaiban, modularitásában, megbízhatóságában – korlátozó tényező.

Ahogy a digitális rendszerek és rendszerelemek tervezésénél az ad hoc módszereket a szisztematikus megoldások háttérbe szorították, ugyanúgy a sín tervezésénél – mint a rendszer fontos elemének tervezésénél – is ezek a módszerek kell, hogy a tervezés menetét irányítsák. A mikroprocesszoros rendszersín tervezésének azokat a legfontosabb paramétereit foglaljuk össze, amelyek a rendszerhierarchia bármelyik szintjén előfordulhatnak. A sokféle lehetőség és megoldás – azok alapvető előnyeinek és hátrányainak – áttekintése ugyanakkor kézenfekvővé teszi számunkra azt a már vizsgált tényt is, hogy miért nagyon nehéz és problematikus terület a sínrendszer szabványosítása.

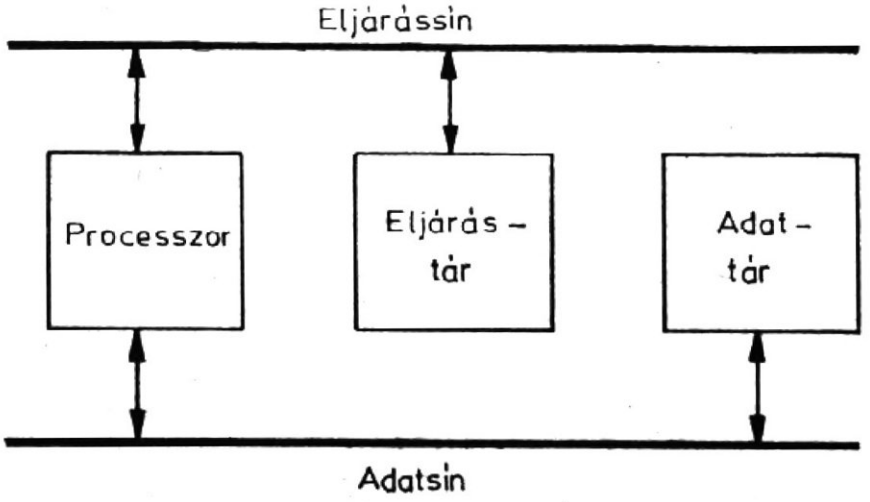
A továbbiakban vizsgálatainkat a sín strukturális paramétereire korlátozzuk. Így nem térünk ki részletesen sem a konkrét áramkörök segítségével való megvalósítás – egyébként jelentős és a tervezés adott szintjén megvalósítandó – problémáira (terhelési, kapacitás, zaj, áthallási stb. viszonyok), sem pedig a konkrét kialakítás (pl. nyomtatott és huzalozott megvalósítás) kérdéseire.

A sínrendszer számos paraméterrel jellemezhető, amelyeket – az előbbiek figyelembevételével – strukturális paramétereknek tekintünk. Ilyen paraméterek a sínek típusa és száma, a vezérlés módja, a sínen való kommunikáció mechanizmusa, az adatátvitel technikája, a sín „szélessége" (vagyis a vonalak száma) stb. Ezen fő paramétereket fogjuk sorba venni, hogy jellemzőik összehasonlításával segítséget nyújtsunk a leggazdaságosabb sínrendszer megtervezéséhez.

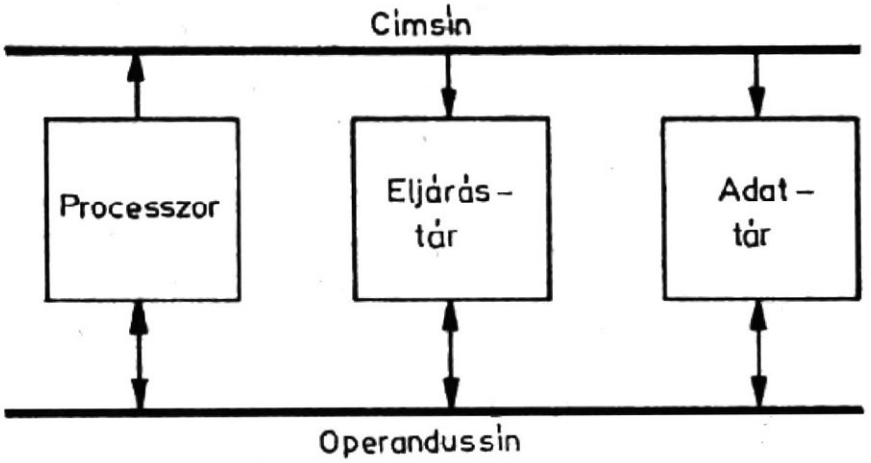
7.1. SÍNTÍPUSOK

A sínek két alapvető típusba sorolhatók:

- a feladathoz, elempárhoz hozzárendelt (dedicated) sínek,
- az univerzális, közös, nem hozzárendelt (nondedicated) sínek.



7.1. ÁBRA. Harvard-osztályú számítógép hozzárendelt típusú eljárás- és adatsínnel



7.2. ÁBRA. Harvard-osztályú számítógép hozzárendelt típusú cím- és operandussinnel

A hozzárendelt sínek

Egy sínt akkor sorolunk ebbe a csoportba, ha vagy egy adott feladathoz (funkcionális hozzárendelés) vagy egy adott fizikai elempárhoz van hozzárendelve (fizikai hozzárendelés). Példáinkat vegyük először a – mikroprocesszoros rendszerhez képest – klaszszikusnak tekinthető számítástechnikából.

A számítástechnikában Harvard-osztályú számítógépnek nevezik az olyan gépet, amelynek két sínrendszere van (7.1. ábra); egyik az eljárásokhoz, a másik pedig az adatokhoz tartozik [1].

Ilyen értelemben mindkét sín a definíció mindkét lehetséges értelmében (funkcionálisan és fizikailag is) hozzárendelt. A funkcionális hozzárendelés azonban csak egy adott hierarchiában igaz. Ha figyelembe vesszük, hogy ebben az esetben az eljárássínen mind címek, mind operandusok lehetnek, tekinthetjük csak fizikailag és nem funkcionálisan hozzárendeltnek.

Ha egy sínrendszert funkcionálisan a címekhez, egy másikat az operandusokhoz rendelünk, a Harvard-osztályú számítógép egy másik formájához jutunk (7.2. ábra). A sínek most azonban nem hozzárendeltek, akár az adat–eljárás szétválasztását, akár a fizikai elempárhoz való tartozást tekintjük [2].

A hozzárendelt sín fő előnye a nagy átbocsátóképesség (thruoghput), mivel egyáltalán nincs vagy csak nagyon kismértékű a sín használatáért folyó „verseny". Ezért a sín elosztását, a versenyhelyzetek megoldását végző sínvezérlő nagyon egyszerű felépítésű lehet. Hasonlóan, az állomások közötti kommunikáció is egyszerűsödik. Pl. címekre sokszor nincs szükség, és a szinkronizáció feltétele is általában automatikusan teljesül.

Ezentúl állomásnak nevezzük azokat az átvitelben részt vevő elemeket, pl. regiszterek, processzorok, tárak stb., amelyek a sínen akár kapnak, akár küldenek információt.

A fő hátrányok a többszörös sininterfészből származó többletköltség (kábelek, csatlakozók, meghajtók stb.), a komplexitásnövekedés következtében jelentkező megbízhatóságcsökkenés. A modularitás is gyenge pontja a hozzárendelt sínrendszereknek, mivel új egységeket a kiépített rendszerekhez sokszor nem lehet hozzáadni.

Közös sínek

A közös, nem hozzárendelt vagy univerzális sínt több funkció vagy állomás között osztjuk meg. A 7.3. ábrán az ún. Princeton-osztályú számítógép blokkvázlatát mutatjuk be, amelynek egyetlen (univerzális) sínrendszere sincs sem fizikai elempárokhoz, sem funkciókhoz hozzárendelve [1], [3].

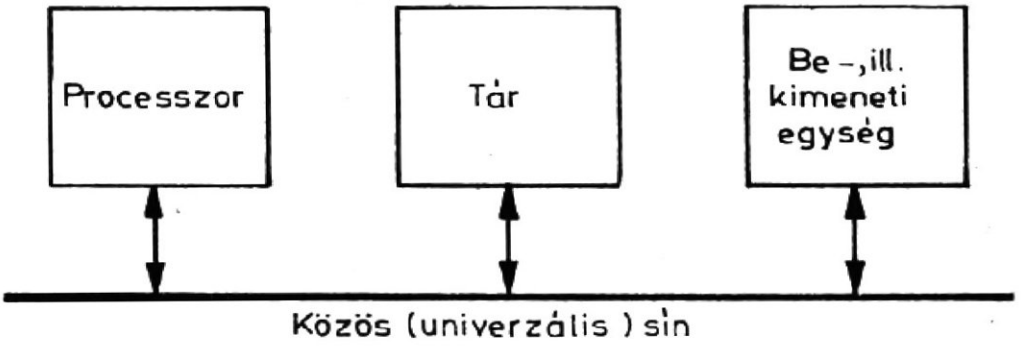
Az ilyen rendszer összes sínt használó állomása közötti kapcsolat többszörös is lehet. Ekkor az egymással párhuzamosan haladó több sín közül bármely állomáspár bármelyiket használhatja. Ez a többszörözés az átviteli sebességet (vagyis az átbocsátóképességet) és a megbízhatóságot is növeli.

Adott állomások (pl. tárak, processzorok) összeköttetésének nagy gyakorlati jelentőségű megoldása a keresztrudas (crossbar) mátrix a telefonközpontoktól származik. Ha nem minden állomást kívánunk minden állomással összekötni (mint a telefonközpontok esetén), hanem csak pl. minden tárat minden processzorral, a vezérlés nem lesz nagyon bonyolult. A „kapcsolók" száma – mint a 7.4. ábrán is láthatjuk – az állomások számának négyzetével nő. Ebben az egyszerűbb formájában fogunk vele még találkozni a multiprocesszoros rendszerek egy csoportjánál (l. a 12. fejezetet). Ennek a megoldásnak komoly hátránya, hogy egy keresztpont (kapcsoló) meghibásodása esetén nincs alternatív út a kapcsolódó állomások számára. Általánosított (a telefonközpontokkal analóg) kód segítségével működtetett változata kiépíthető oly módon, hogy a vezérlés megkeresse a „szabad utat" egyik állomástól a másikig. A régi típusú telefonközpontok mintájára is létesíthetünk sínelrendezést, ahol egy válogató (vagy permutáló) hálózat kapcsol össze n állomást másik n állomással. Ez utóbbi lehetőséget a gyakorlati multimikroprocesszoros rendszer megvalósításához ma még rendkívül ritkán használják. A nem hozzárendelt, univerzális sínrendszer fő előnye a modularitás, így egyszerű a bővítés és az átrendezés (konfigurálás) és gazdaságos a sín kihasználása. Ennek ára, amelyet persze szükségszerűen hátrányosnak kell neveznünk – a közös használatból származó – konfliktushelyzet. Ennek megoldásához idő és speciális logikai áramkör szükséges. Ugyancsak bonyolítja az állomások felépítését, hogy a kommunikációhoz címzésről és szinkronizálásról kell gondoskodni [3].

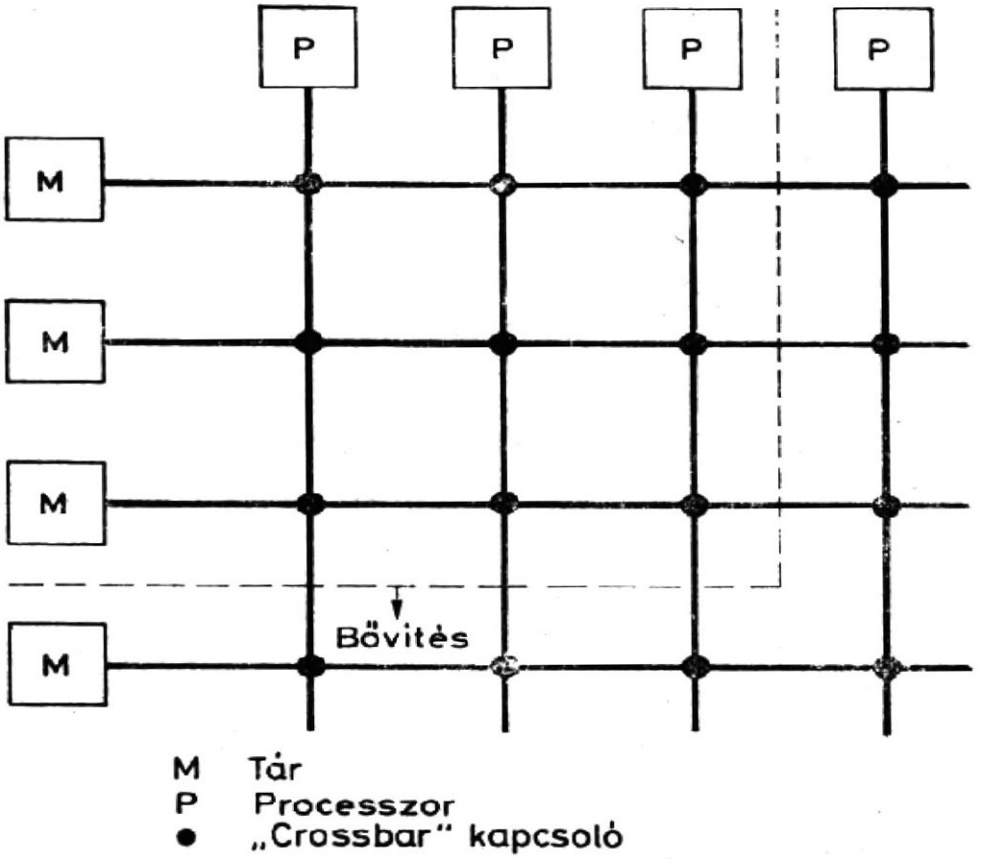
7.2. A SÍNRENDSZER VEZÉRLÉSE

Amikor a sínrendszert több állomás megosztva használja, több problémát kell megoldani, amelyek mind a közös használat következményei. A tulajdonképpeni fő probléma az, hogy hogyan lehet megoldani a sínkérések konfliktusát, vagyis azt, hogy egyszerre csak egy állomás kapja meg a sínt adatok továbbítására.

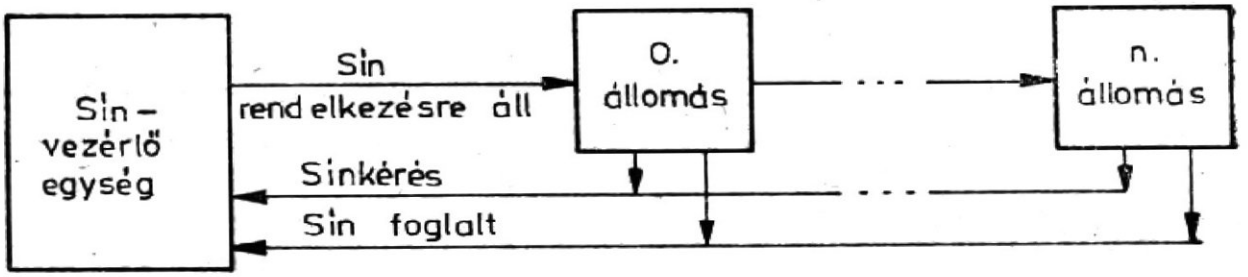
Ezt különböző vezérlőrendszer-elrendezésekkel lehet megoldani. A vezérlőrendszereket két osztályba sorolhatjuk: központosított (centralizált) és elosztott (decentralizált) sínvezérlőkre. Első esetben a sínelosztás feladatát ellátó hardver (amely természetesen esetenként programozható is lehet) külön egység vagy egy adott kitüntetett állomás része. Második esetben az egyes állomásokban a sínvezérlő logikai áramkör funkciói el vannak osztva.



7.3. ÁBRA. Princeton-osztályú számítógép egyetlen univerzális sínnel



7.4. ÁBRA. Keresztrudas sín (az ábrán szaggatott vonallal elválasztva bemutatjuk a bővítés módját is)



7.5. ÁBRA. Központosított soros lánc megoldású sínvezérlő

Ha sokféle megoldás közül kell választanunk, fő szempontunk – a specifikus előnyök–hátrányok mérlegelése mellett – a sín egy kérőállomáshoz való hozzárendeléséhez szükséges idő, ill. az erre a célra felhasznált vonalak számának minimalizálása.

Központosított sínvezérlés

Központosított sínvezérlés esetén egyetlen egység ismeri fel a különböző állomásoktól érkező kéréseket (request) és ad engedélyt a sín használatára. Három alapelrendezést szoktak használni a kérések szétválasztására – mind a központosított, mind az elosztott esetben –, amelyeknek különféle módosított és kombinált megvalósításai is ismeretesek [4], [5], [6]. Ezek a

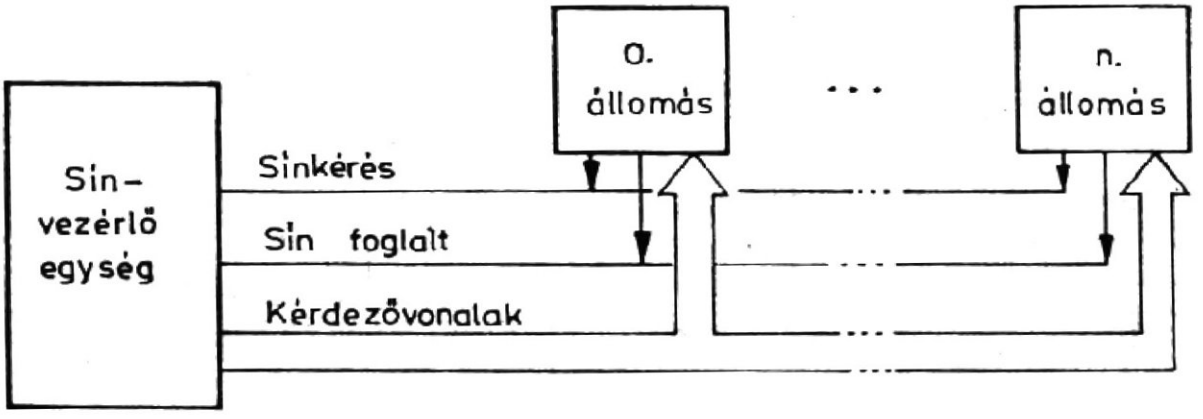
- soros lánc (daisy chain),
- végigkérdezés (polling),
- független kérések (independent request) módszere.

Központosított soros lánc módszerére mutat elrendezést a 7.5. ábra. Az állomások a sín használatára a közös SÍNKÉRÉS (bus request) vonalon keresztül kérhetnek engedélyt. A vezérlő a SÍN RENDELKEZÉSRE ÁLL (bus available) vonal segítségével válaszol, amely az egyes állomásokon keresztül „át van fűzve" oly módon, hogy ha az illető állomás nem akarja a sínt használni, továbbadja a jelet a következőnek. (Ez az ún. huzalozott prioritás vagy elsőbbségi lánc, amelynek sorrendjét a sínvezérlőhöz képest elfoglalt hely szabja meg.) Ha az az állomás kapja meg ezt a jelet, amely kérte a sín használatát, nem adja tovább a következőnek, hanem jelzést ad a SÍN FOGLALT (bus busy) közös vonalra, megszünteti a kérést és elkezdi az átvitelt. A SÍN FOGLALT jel a SÍN RENDELKEZÉSRE ÁLL jelet – a sínvezérlőn keresztül – egészen addig tiltott (ellenkező polaritású) állapotban tartja, ameddig az átvitel zajlik. Ha az illető állomás elengedi a foglalt vonalat, a sín ismét rendelkezésre áll átvitelre, és újabb hasonló folyamat kezdődhet.

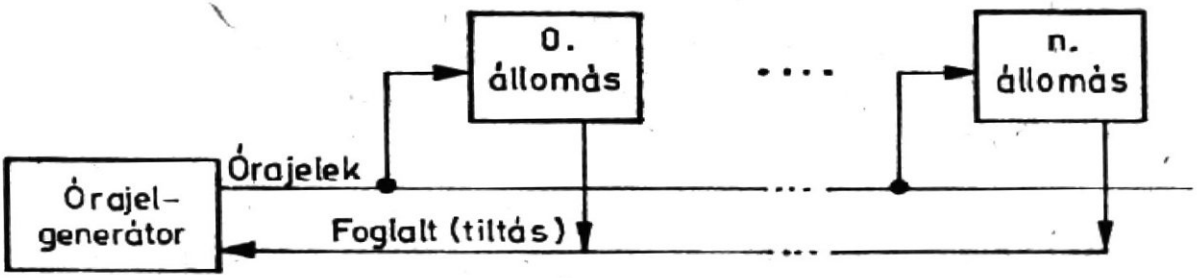
A módszer nyilvánvaló előnye egyszerűsége, mivel kevés vezérlővonalra van szükség, és ezek száma független az állomások számától. További állomások egyszerűen hozzácsatolhatók a rendszerhez (egyszerűen bedughatók az üres helyekre).

A soros láncmódszerek hátránya, hogy a lánc egy elemének meghibásodása (a lánc „megszakadása") az utána következő összes állomás számára megakadályozza a sín használatát. (Mivel azonban a logikai rendszer nagyon egyszerű, megtöbbszörözésével, redundanciával a megbízhatóság növelhető.) Másik hátrány a rögzített prioritás struktúra következménye. A vezérlőhöz közelebb levő állomások gyakori kérései a „hátul" levőket teljesen kizárhatják a sín használatából. A sorbafűzés miatt a sínhozzárendelés folyamata elég lassú is. A sorbafűzés biztosítása azt is szükségessé teszi, hogy minden „megelőző" pozícióban mindig legyen működő állomás.

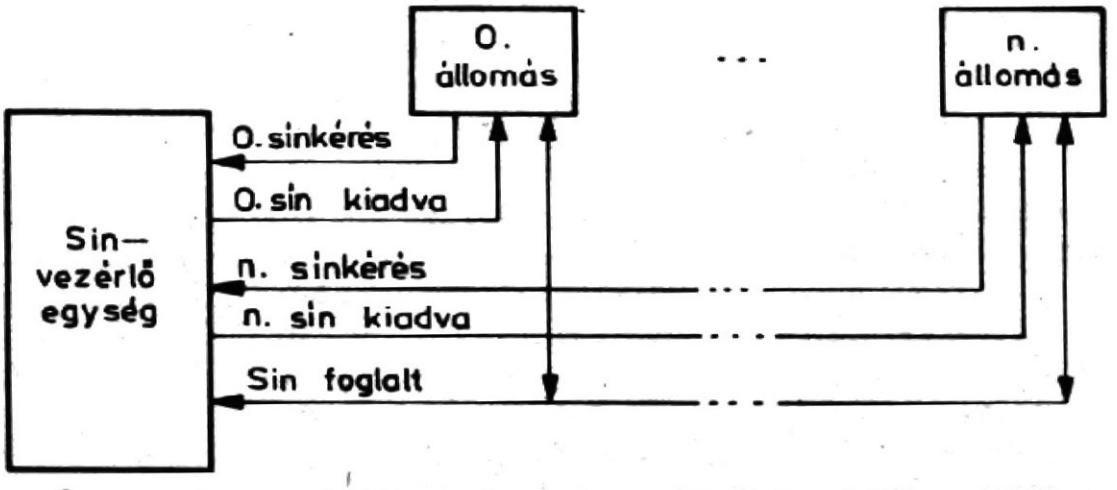
Központosított kérdező- (polling) módszer vázlatát láthatjuk a 7.6. ábrán. A soros lánchoz hasonlóan, minden állomás a sín használatát a SÍNKÉRÉS vonalra adott jellel kérheti. Amikor a sínvezérlő egy ilyen kérést kapott, elkezdi végigkérdezni az állomásokat, így határozva meg azt, hogy honnan érkezett a kérés. Ezt a vizsgálatot a KÉRDEZŐVONALAK (polling lines) segítségével végzik úgy, hogy ezekre a vonalakra egy számláló folyamatosan növelt tartalmát adják. Amikor a kiadott szám megegyezik a kérőállomásba „behuzalozott" értékkel, ez az állomás ezt a SÍN FOGLALT vonal jelszintjének felemelésével jelzi. A vezérlő ekkor abbahagyja a számlálást és így a végigkérdezést is egészen addig, amíg az adott állomás befejezte az átvitelt és megszüntette a foglaltságot. Ha ezután újabb kérés érkezik, a vezérlő zérus tartalommal kezdi újra a számlálást és így a kérdezést, vagy onnan folytatja, ahol abbahagyta. Az első esetben az egyes állomások prioritása a soros lánchoz hasonló (vagyis itt kisebb szám jelenti a nagyobb prioritást), míg a második esetben minden állomás prioritása egyenlő. Az utóbbit körbeléptetett (round-robin) módszernek szokták nevezni. A vizsgálat



7.6. ÁBRA. Központi sínvezérlő-megoldás (végigkérdezés központi számlálóval)



7.7. ÁBRA. Központi sínvezérlő-megoldás (végigkérdezés helyi számlálókkal)



7.8. ÁBRA. Központi sínvezérlő-megoldás (független kérések módszere)

módszeréből következően a prioritás egy adott rendszeren belül nem szükségszerűen állandó, azt könnyű módosítani a végigkérdezés sorrendjének megváltoztatásával.

Az elrendezés egy módosított változatát kaphatjuk, ha a vezérlőben levő számlálót folyamatosan hagyjuk működni, egészen addig, amíg azt egy állomás meg nem állítja. Ekkor nincs szükségünk a SÍNKÉRÉS vonalra, de a módosítás érinti az újraindítási (vagyis prioritás) módot és azt a közepes időt, amely alatt a sínt egy állomáshoz tudjuk rendelni (average bus assignment time).

A végigkérdezés módszerének megbízhatósága jobb, mint a soros láncé, mivel itt egy állomás meghibásodása nem okoz közvetlenül rendszerhibát és az egyes állomások fizikai elhelyezése sem probléma. A kérdezővonalak száma azonban korlátozza a bővíthetőséget, vagyis a maximális állomásszámot. Ezt egyetlen vonalon kibocsátott soros „kérdezőszámokkal" megszüntethetnénk, ekkor azonban a soros szinkron kommunikáció összes nehézségét kellene minden állomásnál vállalnunk.

A 7.7. ábra szerint a központosított végigkérdezés módszerét függetlenné tehetjük az állomások számától, ha minden egyes állomásban elhelyezünk egy számlálót. Ekkor a vezérlő feladata leegyszerűsödik az órajelek kiküldésére, amelyeket minden állomás számol. Amikor a számláló tartalma megegyezik a sínt kérő állomás számlálójába „huzalozott" értékkel, az a FOGLALT (TILTÁS) vonal jelszintjét felemeli és ezzel letiltja az órajelek kibocsátását. Amikor az átvitel befejeződött, megszünteti a foglaltsági állapotot és a számlálás folytatódik. A prioritás ismét többféle lehet és könnyen módosítható. Ha a számlálás egyszerűen folytatódik, azonos – körbeléptetett – prioritás van, ha az újraindulás előtt minden számlálót alaphelyzetbe hozunk, a prioritás az állomások kódjával arányos. Az utóbbi esetben a kód módosításával (pl. átprogramozással) a prioritáskiosztást módosíthatjuk, a rendszert a változó körülményekhez alkalmazkodóképessé (adaptive) tehetjük.

A végigkérdezést és a soros láncot kombinálhatjuk is egymással. Ebben az esetben az egyes állomások között címek vagy prioritásértékek haladnak tovább a SÍN RENDELKEZÉSRE ÁLL jel helyett. Ezzel a soros lánc merev prioritása flexibilisebbé tehető több vonal és logikai áramkör alkalmazásának árán.

A **központosított független kérések módszerét** a 7.8. ábrán láthatjuk. Ebben az esetben minden állomásnak külön SÍNKÉRÉS és SÍN KIADVA (bus granted) vonalai vannak, amelyek segítségével tartja a sínvezérlővel a kapcsolatot. Ha kérés érkezik, a sínvezérlő kiválasztja a következő kiszolgálandó állomást, és kiküldi a számára a SÍN KIADVA jelet. Ez az állomás ekkor megszünteti a kérést, és a sín foglaltságát a többi állomásnak a SÍN FOGLALT vonal jelszintjének felemelésével jelzi. Ha az átvitel véget ért, az állomás megszünteti a foglaltságot. Ennek hatására a vezérlő visszaveszi a SÍN KIADVA jelet, és ezután kíválasztja a következő kérőállomást.

Ennél a módszernél a sín rendelkezésre bocsátásáig a várakozási idő kisebb, mint akár a soros lánc, akár a végigkérdezés alkalmazása esetén, mivel minden kérés párhuzamosan, egyidejűleg kerül a sínvezérlőhöz. Ezen túlmenően a prioritás is teljesen szabadon választható, a sínvezérlő bármelyik kiosztási módot vagy azok kombinációit is alkalmazhatja. Arra is lehetőség van, hogy olyan állomás kérését, amely pl. meghibásodott (és ezért végleg lefoglalná a sint), a vezérlő ne vegye figyelembe.

A módszer fő hátránya, hogy sok vonalra van szükség amely persze a sínvezérlő felépítésének viszonylagos bonyolultságával is együtt jár [7].

Elosztott sínvezérlés

Ebben az esetben a vezérlőrendszer (a vezérlőlogika) a sínállomásokban van elosztva. Ennek megfelelően osztott soros láncot oly módon kaphatunk, hogy a központosított elrendezésből (l. a 7.5. ábrát) elhagyjuk a SÍN FOGLALT vonalat és a közös SÍNKÉRÉS vonalat az 0 jelű SÍN RENDELKEZÉSRE ÁLL bemenetre csatlakoztatjuk.

Az ily módon kialakított elrendezést mutatjuk be a 7.9. ábrán. Az adott állomás SÍNKÉRÉS vonaljelszintjének felemelésével kérheti a sín használatát abban az esetben, ha a bemeneti SÍN RENDELKEZÉSRE ÁLL vonalán engedélyező logikai jelszint van. A sínt nem kérő állomás egyszerűen továbbadja ezt a jelet, az első kérő pedig nem, fenntartva a SÍNKÉRÉS jelét egészen addig, amíg az átvitelt befejezte. Ekkor a SÍN RENDELKEZÉSRE ÁLL jel is visszajut eredeti állapotába.

Az ekkor előálló állapotból jól érzékelhetjük a megoldás fő hibáját: pl. ha a lánc közepén levő állomás befejezte a sínhasználatot, a SÍNKÉRÉS az előző állomás felé, a SÍN RENDELKEZÉSRE ÁLL jel pedig tovább fog haladni a másik irányba. Ha pl. ezen állomás két oldalán levő másik kettő akarja a sínt használni, a „jobb oldali" kapja meg egy időre a sínhasználatot, ami után a „bal oldali" jut végleg szóhoz. Ez a verseny kiküszöbölhető a kijelölés késleltetésével. Ezt akár aszinkron módon, minden állomásba beépített késleltető-áramkörrel, akár központilag szinkronizálva elvégezhetjük.

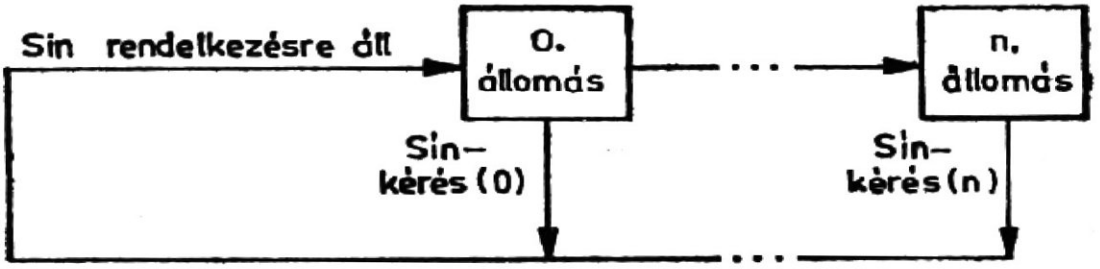
Topológiailag egyszerűbb **osztott soros láncot** mutat a 7.10. ábra. Itt a sín állapotát nem a SÍN RENDELKEZÉSRE ÁLL vonal jelének statikus állapota, hanem egy jelátmenet ezen a vonalon való továbbhaladása jelez. Ha a sín szabad, ez a jelátmenet halad folyamatosan körbe. Az első olyan állomás, amely használni kívánja a sínt, megszünteti a jelátmenetnek ezt a továbbhaladását, majd ha sínre már nincs szüksége, újra elindítja azt.

Ez a megoldás, mivel jelátmenettel dolgozik, a zajra érzékenyebb az előző megoldásnál. Ez az érzékenység csökkenthető kérés/elfogadás típusú megoldással, de lassítja a működést [1].

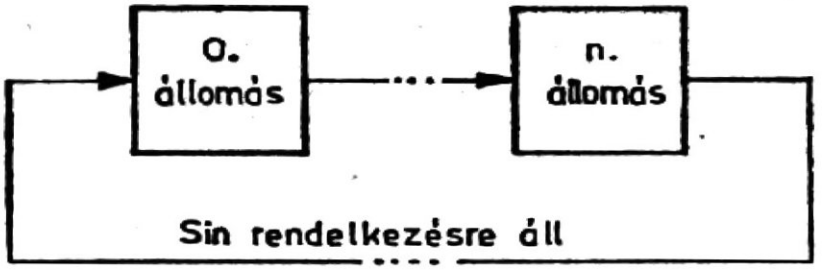
Osztott végigkérdezés a 7.11. ábra szerint végezhető. Ha egy állomás a sínt használni akarja, egy kódot (cím vagy prioritás) ad a KÉRDEZŐVONALAK-ra és a SÍN RENDELKEZÉSRE ÁLL vonal jelszintjét felemeli. Ha a kód megegyezik egy másik olyan állomás kódjával, amely a sínt használni kívánja, ez SÍN ELFOGADVA jellel válaszol. Ekkor az előbbi állomás és a kérdezővonalak közötti, valamint a SÍN RENDELKEZÉSRE ÁLL vonal közötti kapcsolat megszűnik, az utóbbi állomás pedig a SÍN ELFOGADVA vonal jelét alsó logikai szintre viszi és használni kezdi a sínt. Ha kezdeményező (kérdező-) állomás nem kapta meg a SÍN ELFOGADVA jelet (ezt egy külön ELUTASÍTVA vonal is jelezheti,, amely megkülönbözteti a „nem kérést" és sikertelen kérést), megváltoztatja a kiadandó kódot valamilyen algoritmus szerint és ismét próbálkozik. A rendszer helyes működéséhez szükséges, hogy kezdetben csak egy állomás kapja meg a vezérlést. Mivel az osztott sínvezérlők lényegében azonos elemeket tartalmaznak, mint a központosított esetben az egyetlen sínvezérlő, ehhez lényegesen több hardverre van szükség. Ezzel szemben a megbízhatóság lényegesen megnő, mivel egy állomás meghibásodása nem vezet szükségszerűen az egész sín „megbénulásához".

A 7.12. ábra az **osztott független kérések** megoldását mutatja. Minden állomás a prioritásnak megfelelő SÍNKÉRÉS vonala segítségével jelentkezhet. Amikor a sínt éppen használó állomás befejezi az átvitelt, elengedi a SÍN FOGLALT vonalat. Ekkor minden olyan állomás, amelyik átvitelre vár, megvizsgál minden aktív SÍNKÉRÉS jelet. A sínt az az állomás használhatja, amelyik saját prioritását találja a legmagasabbnak. Ezt a SÍN FOGLALT vonal segítségével jelzi ismét a többi állomás számára. Ekkor mindegyik, a sínre váró állomás megszünteti a SÍNKÉRÉS jelet (és tárolja a sínhasználatra kijelölt állomás prioritását, ha a körbeforgatott prioritás algoritmust használjuk).

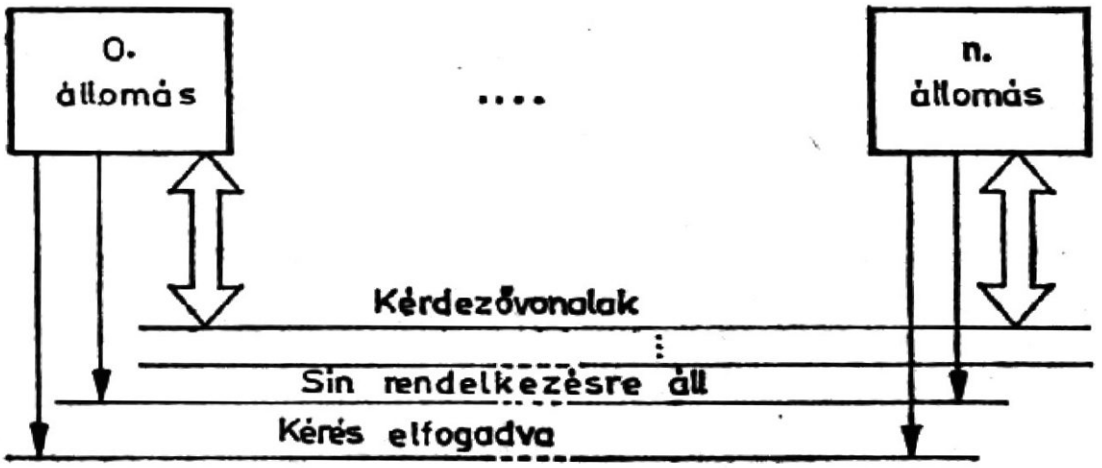
Az egyes állomások prioritásrendszere ebben az esetben egyszerűbb, mint a központosított változatban alkalmazott megoldás, de a vonalak (és csatlakozópontok) száma nagyobb. Ha a prioritás állandó, vagyis dinamikusan nem módosítható,



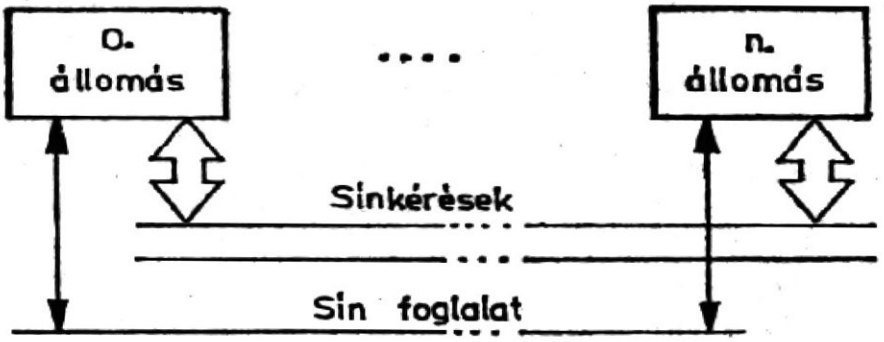
7.9. ÁBRA. Osztott soros lánc megoldású sínvezérlő



7.10. ÁBRA. Egyszerűsített osztott soros lánc megoldású vezérlő



7.11. ÁBRA. Végigkérdezéses módszer osztott sínvezérlő esetén



7.12. ÁBRA. Független kérések módszere osztott sínvezérlő esetén

nem szükséges, hogy minden SÍNKÉRÉS jel eljusson minden állomáshoz, így az osztott változat esetén kb. tíz állomásig kevesebb vonalra van szükség. Az osztottság itt is némi megbízhatóságjavulással jár. Az órajel eltolódás miatt azonban csak kis rendszerek esetén célszerű ennek a megoldásnak a használata, amely nagyon érzékeny a zajra és az órajelhibákra is.

7.3. KOMMUNIKÁCIÓS ELJÁRÁSOK

Ha egy állomás egyszer megkapta a sínt, megfelelő kapcsolatot kell, hogy létesítsen a kívánt célállomással. Az ehhez szükséges információ a következő:

- a forrásállomás címe,
- a célállomás címe,
- az átvitt információ típusa (kommunikációs osztály),
- a megvalósított funkció (akcióosztály).

A forráscím sokszor implicit (beleértett), és adott állomásokhoz hozzárendelt sín esetén a célállomás címe is az lehet.

A kommunikációs osztálynak megfelelően az átvitt információ lehet pl. adat, parancs, állapot, megszakítás stb. Ez is lehet részben vagy egészében implicit, vagy összeolvadhat a megvalósított funkciót meghatározó résszel, amely lehet pl. beviteli, kiviteli egység stb. Ezek után kerülhet sor a tulajdonképpeni információ átvitelére. A sínen keresztül lefolytatott kommunikáció módszere lehet:

- szinkron,
- aszinkron vagy
- félszinkron.

Szinkron átvitel

A szinkron átvitel módszerei jól ismertek, és egész technikája nagyon jól kidolgozott, mivel hosszú vonalakon való információátvitel esetén is jól használható. A sínen való átvitel esetén jelentősége lényegesen kisebb, mint a másik két módszeré, ezért itt röviden csak az alapelveket ismertetjük.

Egy szinkron sín azzal jellemezhető, hogy állandó, egyenlő „szélességű" időszakaszok (time slots) állnak rendelkezésre az átvitelhez, amelyeket általában egy központi időzítő generál vagy szinkronizál.

A központilag (globálisan) időzített sín egy olyan órajelgenerátort tartalmaz, amely minden – a sínt használó – állomás számára órajeleket szolgáltat. A sín logikai struktúrájától és fizikai elrendezésétől függően az órajelek eltolódása jelentős problémát okozhat. Ezen oly módon javíthatunk, hogy egy központilag generált jelet (frame signal: keretjel) bocsátunk ki a minden állomásban meglevő helyi óra szinkronizálására. A helyi órajel azután egy számlálóra kerül, amelynek dekódolt állapotai adják meg az egyes állomások számára kijelölt időszakaszokat. A szinkronjel minden számlálási ciklus kezdetét szabja meg. A szinkronjel külön vonalon is haladhat, de eltolódásának hatása kisebb, ha az adatokkal együtt (amplitudó, fázisjelként, vagy inkább kódolt formában) kerül kiküldésre. Mind a központi, mind pedig a helyi szinkronizáció nagyon érzékeny a zajra.

A szinkron sín egyes időszakaszai vagy adott állomásokhoz vannak (dedicated assignement) vagy nincsenek hozzárendelve (non-dedicated assignement). Szükség esetén a két lehetőség kombinálható is.

Ha egy időszakasz egy állomáshoz hozzá van rendelve, ennek a számára állandóan rendelkezésre áll, függetlenül a tényleges használat frekvenciájától. Ebben az esetben az állomások a sínt időmultiplex alapon használhatják. A prioritás ekkor

csak oly módon valósítható meg, hogy egyetlen állomás számára egynél több ilyen időszakaszt jelölünk ki. (Ezt az eljárást szuperkommunikációnak is szokták nevezni.) Egyetlen szakaszhoz több – lassú vagy felváltva működő – állomást is kijelölhetünk (szubkommunikáció).

Gyakorlatban a kijelölt időszakasz módszer nem gazdaságos. Nem minden állomás kíván ui. egyszerre kommunikálni a sínen, a bővítés is nehéz, és az állomások tényleges átviteli sebességigénye is különböző. Az utóbbi tényező a sín gazdaságos kihasználásának is akadályozója, mivel a leglassabb állomás átviteli sebességéhez kellene az időszakaszokat illeszteni, aminek következtében az átlagos átviteli sebesség nagyon lecsökkenne. Ezen a problémán pufferregiszterek alkalmazásával segíteni lehet, ha azonban az átviteli sebességeltérés nagy, nagyon nagy méretű pufferekre lehet szükség. (Ezen pufferekbe való be- és kivitel egyidejű biztosítását is meg kell valósítani.)

Nem hozzárendelt időszakasz módszerének alkalmazásakor csak azon állomások számára kell időtartományt kijelölni, amelyek ezt éppen igénylik. Ez javítja a sín kihasználását, de az ára egy jóval nagyobb hardver.

Aszinkron átvitel

Az aszinkron átvitel módszerei két csoportba sorolhatók:

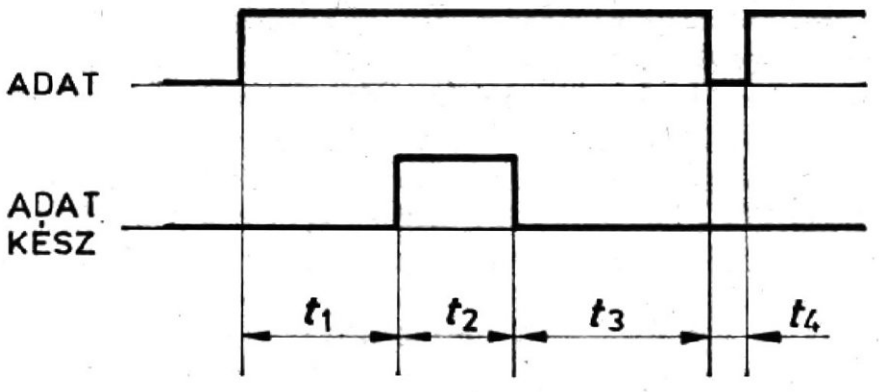
- egy irányból vezérelt rendszerek (OWC: One-Way-Command),
- kérés/visszaigazolás (Request/Acknowledge) típusú rendszerek (amelyeket handshaking eljárásoknak is szoktak nevezni).

(Az aszinkron átvitel bitsoros kommunikáció módszere, amelynél az órajelek időzítését az adatváltozásokból nyerjük, egy további harmadik módszernek tekinthető. Ezzel azonban itt nem foglalkozunk, mivel sínen való átvitelnél nem alkalmazzák.)

Az egyirányból vezérelt elnevezés arra utal, hogy az adatátvitelt teljesen az egyik – a kommunikációban részt vevő – állomás vezérli, és ha az átvitelt megindította, nincs kölcsönhatás a két állomás között (kivéve az adathiba esetét, amikor hibajelzés jöhet vissza).

Az OWC interfészt akár a forrás, akár a célállomás vezérelheti [8].

Forrásvezérelt OWC esetén az átvivőállomásról az adatok és az ADAT KÉSZ (Data Ready) jel a 7.13. ábra idődiagramjának megfelelően kerül a sínre. Az ADAT KÉSZ jel időzítése nagymértékben függ a megvalósítás részleteitől, vagyis attól, hogy a célállomás hogyan használja azt fel. Ha az ADAT KÉSZ jel önmaga írja be közvetlenül az adatot, akkor t_1 idővel késleltetni kell ahhoz, hogy az adat – a sínen való továbbhaladás után – a fogadó- (cél-) állomásnál a beírás pillanata előtt „beálljon". Biztosabb megoldás, ha az adat megérkezése után bocsátjuk csak ki ezt a jelet, de ekkor az átviteli sebesség az állomások közötti fizikai távolságtól fog függeni. Jobb megoldás ezért az ADAT KÉSZ jelet az adatokkal egyenlő hosszúvá választani (vagyis $t_1 = 0$ és $t_3 = 0$), és a beírást végző késleltetést a fogadóállomásnál megvaló-



7.13. ÁBRA. Forrásvezérelt aszinkron OWC kommunikáció idődiagramja

sítani. A t_4 idő a forrásállomás kimeneti regiszterének újra való feltöltéséhez vagy a sínvezérlés átadásához szükséges.

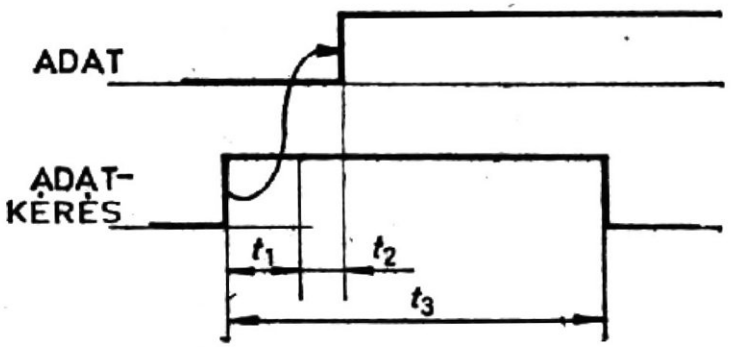
A forrásvezérelt OWC interfész fő előnye egyszerűsége és gyorsasága. A fő hátránya, hogy nincs a célállomástól visszajelzés az adatok megérkezéséről. Ugyanakkor ezen a módon nehéz (és jó hatásfokkal tulajdonképpen nem is lehet) különböző sebességű állomások közötti kommunikációt megvalósítani. Zajimpulzusok az ADAT KÉSZ vonalon könnyen hibát okozhatnak.

A visszajelzés (adatelfogadás) problémája megoldható a célállomás vezérelt OWC interfész alkalmazásával, amelynek idődiagramját a 7.14. ábrán mutatjuk be. Ha a fogadóállomás az ADATKÉRÉS vonaljelszintjét felemeli, ennek hatására a forrás- (küldő-) állomás az adatokat a sínre teszi.

A célállomás most nem tudja egyszerű módon meghatározni, hogy mikor „kapuzza be" az adatvonalak tartalmát, mert az az adott fizikai távolságtól függ. Ha a fogadóállomás hibát észlel (pl. a szó paritásának megvizsgálása után), egy HIBÁS ADAT (Data Error) jelet küld vissza újabb ADATKÉRÉS helyett. Ez a vizsgálati idő persze az átviteli sebességet korlátozhatja. A sebességet az állomások közötti sínátvitel késleltetési ideje is – a kétirányú jelátvitel miatt – a forrásvezérelt esethez képest kétszeresen befolyásolja.

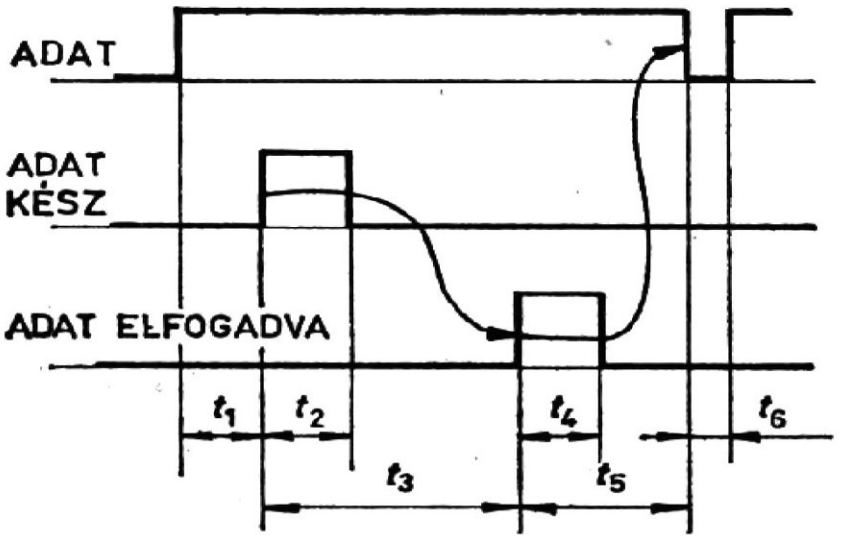
A kérés, ill. visszaigazolás módszeren belül – amely a mikroprocesszoros sínrendszerek esetén használt, legfontosabb, és leggyakoribb eljárás – három alcsoportot különböztethetünk meg. A csoportosítás alapja az adatok, az adatok beírását (bekapuzását) végző jel és az adatok elfogadását (visszaigazolás) jelző impulzus egymáshoz képest elfoglalt helyzete. Ennek alapján megkülönböztetünk:

- nem összekapcsolt,
- félig összekapcsolt és
- teljesen összekapcsolt aszinkron kérés, ill. visszaigazolás módszert.

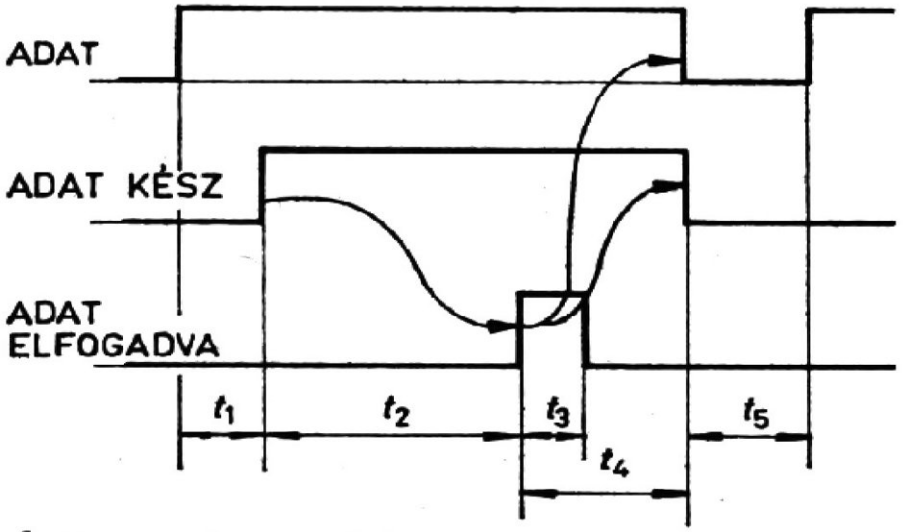


7.14. ÁBRA. Célállomás-vezérelt aszinkron OWC kommunikáció idődiagramja

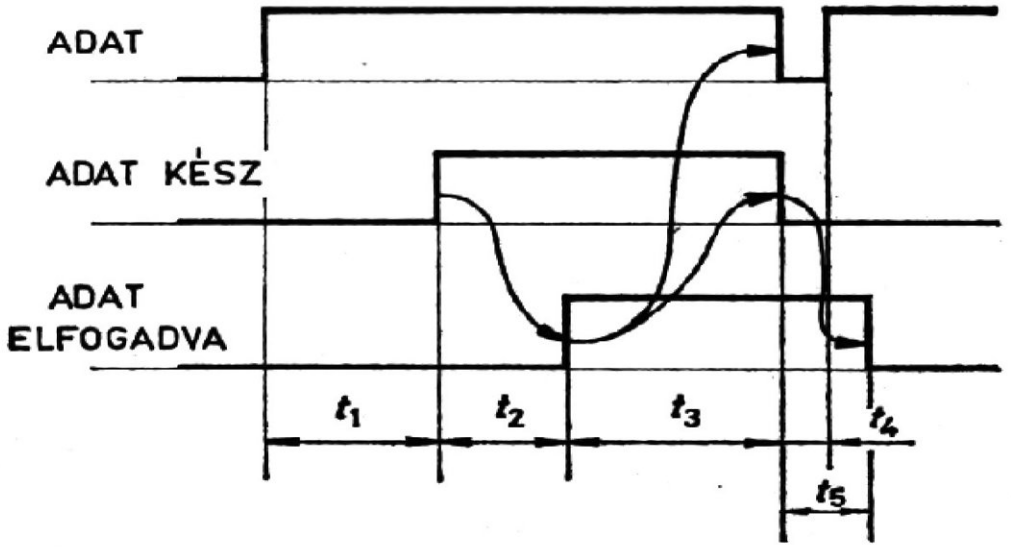
A 7.15. ábra a nem összekapcsolt módszer idődiagramját mutatja. A forrásállomás az adatokat a sínre teszi és az ADAT KÉSZ vonaljelszintjét felemeli. A célállomás tárolja az adatokat és egy ADAT ELFOGADVA jellel válaszol, amelynek hatására az ADAT jel megszűnik, és új adatokat lehet a sínre tenni. Ha a fogadó, ill. célállomás hibát észlel, az ADAT ELFOGADVA jel helyett egy külön vonalon ADATHIBA jel választ küld. Ez a megoldás nem csupán a hiba jelzésére alkalmas, hanem a párbeszéd (más szóval kézfogás) következtében bármilyen sebességű állomások közötti kapcsolatot biztosítja. A jelek pontos időzítése itt is a konkrét implementáció függvénye. A sín késleltetési idejének és az impulzusok szélességének nem megfelelően megválasztott aránya esetén újabb ADAT KÉSZ jel érkezhet, mialatt az ADAT ELFOGADVA jel még az előzőleg küldött adatokra „válaszolva" a felső logikai jelszint értéken van. Ebben az esetben az egész sín működése megakadhat („befagyás").



7.15. ÁBRA. Aszinkron, nem összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja



7.16. ÁBRA. Aszinkron, félig összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja



7.17. ÁBRA. Aszinkron, teljesen összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja

Ezt a hibát elkerülhetjük, ha az ADAT KÉSZ jel egészen addig tart, amíg a forrásállomás megkapta az ADAT ELFOGADVA (vagy ADATHIBA) jelet. A 7.16. ábra szerinti, félig összekapcsolt típusú interfész az újabb átvitelt biztosan késlelteti a válasz megérkezéséig. Az ADAT KÉSZ jel ilyen – változó hosszúságú – megoldása esetén kisebb a zajérzékenység is – az előző esethez képest – nagyon kis mértékű hardver növekedés mellett, mindenféle sebességcsökkenés nélkül.

Egy további időzítési hiba következhet be azonban, ha az ADAT ELFOGADVA jel túlnyúlik a forrásállomás pufferregisztere újra adattal való feltöltésének periódusán, és ezzel kitoldja a következő ADAT KÉSZ felfutó élét. Ezen segít a teljesen összekapcsolt interfész (7.17. ábra), ahol az új ADAT KÉSZ az előző ADAT ELFOGADVA (vagy ADATHIBA) jel lefutóéléig nem kezdődhet el. Így most mind a két, az adatátvitelben részt vevő jel érzéketlenebb a zajra is. Ennek ára és így a megoldás fő hátránya nem a kissé komplexebb hardver, hanem hogy a sínkésleltetés megkétszereződött a félig összekapcsolt esethez képest, amely az átviteli sebesség felső határát közel a felére csökkenti.

Félszinkron átvitel

Azt a sínt, amelyek működésmódja félszinkron, olyan megoldásnak tekinthetjük, ahol az átvitelre nem feltétlenül egyenlő hosszú időszakok állnak rendelkezésre. Másrészről ezt a sínt úgy is definiálhatjuk, hogy az alapvetően egy olyan aszinkron működésű sín, amely ha nincs átvitel, szinkronként viselkedik. Ily módon ez a megoldás megőrzi az aszinkron működés fő előnyét – a különböző átviteli sebességű állomások közötti átvitel gazdaságos megvalósítását –, míg megszabadít ennek hibaérzékenységétől és a szinkronrendszer órajelelcsúszásának probrémájától is.

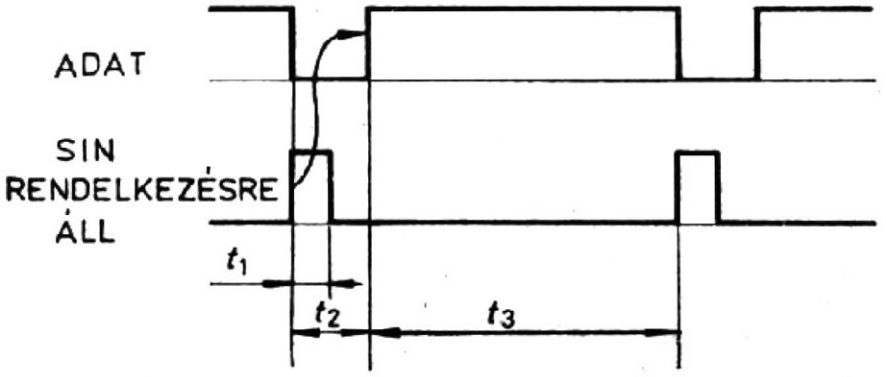
Egy szinkronrendszerben a hibajelzés nem lassítja a sín működését, mivel az *adat hibás* jel akármennyi időszakasszal eltolható. Ez nem lehetséges a szokásos aszinkron sínnél, mivel nincs minden állomás számára hozzáférhető központi időzítőjel. Valóságban ez a megállapítás csak akkor igaz, ha nincs átvitel a sínen, mivel ha van adatátvitel, mindig van egy vagy több olyan jel, amelyet minden állomás megfigyelhet. Így egy aszinkron sín is késleltethetné az ADATHIBA jelet valahány *n* szó átviteli időnek megfelelően, de ha a sínt egyik állomás se használja, nincs jel a sínen, és így az utolsó *n* szó átvitelét végző állomásoknak nincs időreferenciájuk a késleltetett hibajelzéshez. A félszinkron sín oly módon oldja meg ezt a problémát, hogy erre a célra az aszinkron átvitelek közötti időben órajeleket generál. Bár gyakorlatban csak *n* impulzusra lenne szükség, a folyamatos órajel-generálás van, amely a normális sínműködés újraindításához is felhasználható.

Központi vezérlés esetén a sínvezérlő érzékeli, hogy a sínt nem használják és ekkor elkezdi ezeket a speciális órajeleket kibocsátani. Osztott vezérlés esetén ezt a feladatot a sínt utoljára használó állomásnak kell megoldania.

Az aszinkron sínekhez hasonlóan a félszinkron rendszer is lehet

- egy irányból vezérelt (OWC) vagy
- kérés-visszaigazolás rendszerű.

A félszinkron forrásvezérelt sín időzítése (7.18. ábra) hasonlít aszinkron megfelelőjéhez (l. a 7.13. ábrát). A forrásállomás azonban ahelyett, hogy egy ADATKÉSZ jelet küldene ki, jelezve az új adat megjelenését, egy SÍN RENDELKEZÉSRE ÁLL jelet küld ki. Ez definiálja saját időszakának a végét és így a következő kezdetét. Egy adott időszakasz alatt lezajlik a sín kijelölése a következő szakaszban való felhasználáshoz. A SÍN RENDELKEZÉSRE ÁLL jel hatására azután a következő állomás a sínre kiküldi a célállomás címét és az adatokat. A kiválasztott állomás ezután megvárja, amíg a sínen a jelek beállnak az adatoknak megfelelően, beírja az adatokat és egy másik SÍN RENDELKEZÉSRE ÁLL jelet generál.



7.18. ÁBRA. Félszinkron, forrásvezérelt OWC kommunikáció idődiagramja

A félszinkron sínrendszerek esetén lehetőség van az ADAT KÉSZ és a BUSZ RENDELKEZÉSRE ÁLL funkciók kombinálására és így egyetlen vonal segítségével való megvalósítására.

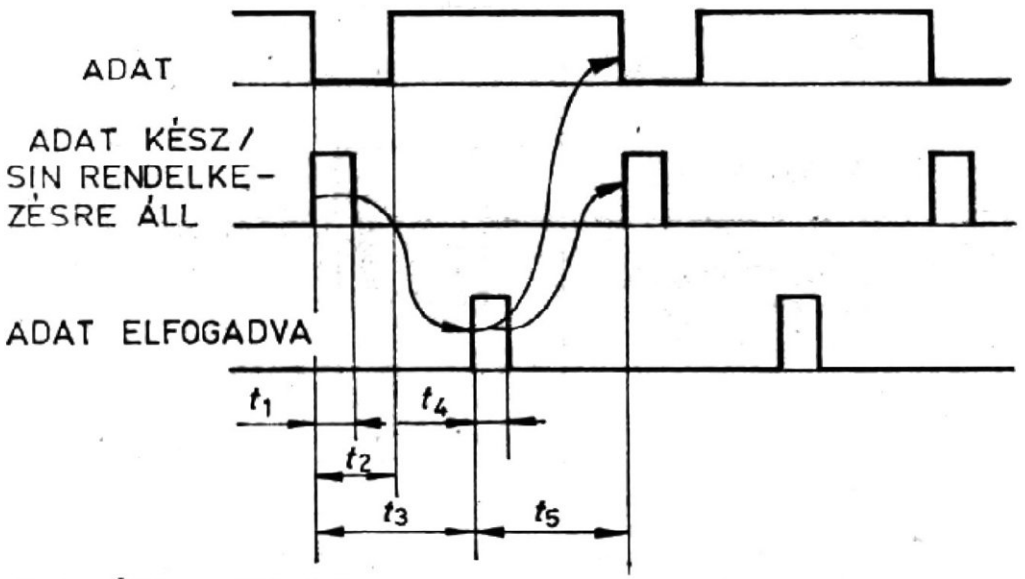
Az egyirányból vezérelt sín kiegészítésével jutunk el a nem összekapcsolt kérés – ill. visszaigazolás interfészhez (7.19. ábra). Ez már képes különféle sebességű állomásokat kiszolgálni, de zajérzékeny, és hajlamos a „befagyásra". A forrásállomás ebben az esetben nem hoz létre SÍN RENDELKEZÉSRE ÁLL jelet addig, amíg vissza nem kapta az ADAT ELFOGADVA jelet, ami „sínsávszélesség" pazarlást jelent, ha egy lassúbb forrást a következő időszakaszban egy gyorsabb követi.

A 7.20. ábra szerint, ha az ADAT ELFOGADVA jelet használjuk SÍN RENDELKEZÉSRE ÁLL jelként, ez a probléma megoldható. Így a sávszélesség aszinkron értelemben – a fogadóállomás sebességének megfelelően – optimalizált. Természetesen a zajérzékenység és „befagyás" problémája változatlanul jelen van.

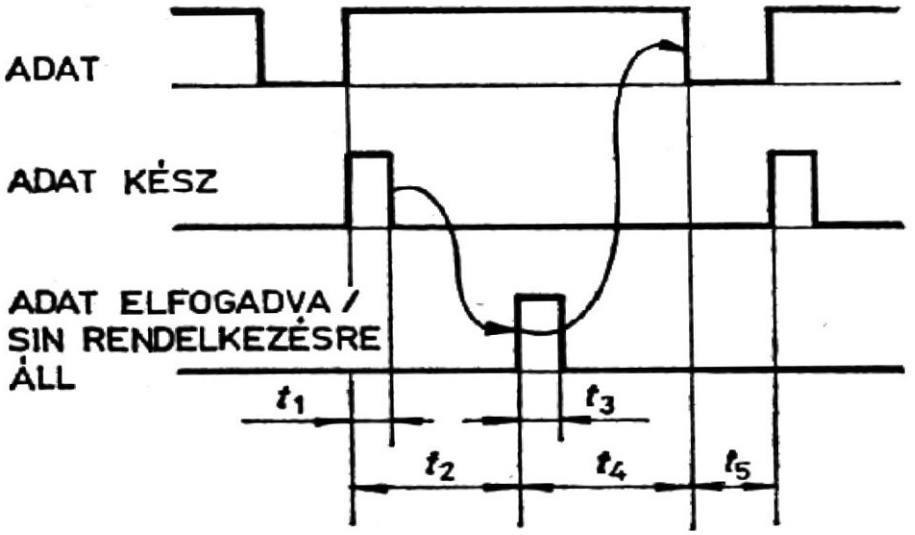
Mint láttuk, az ADAT KÉSZ jel SÍN RENDELKEZÉSRE ÁLL jelként való alkalmazása nem bizonyult jó megoldásnak a nem összekapcsolt esetben, így nem meglepő, hogy a félig összekapcsolt esetben sem optimális.

A 7.21. ábrán is láthatjuk, hogy a t_5 időt elpazaroljuk, mivel az ADAT KÉSZ jelnek az első élét használjuk SÍN RENDELKEZÉSRE ÁLL jelként.

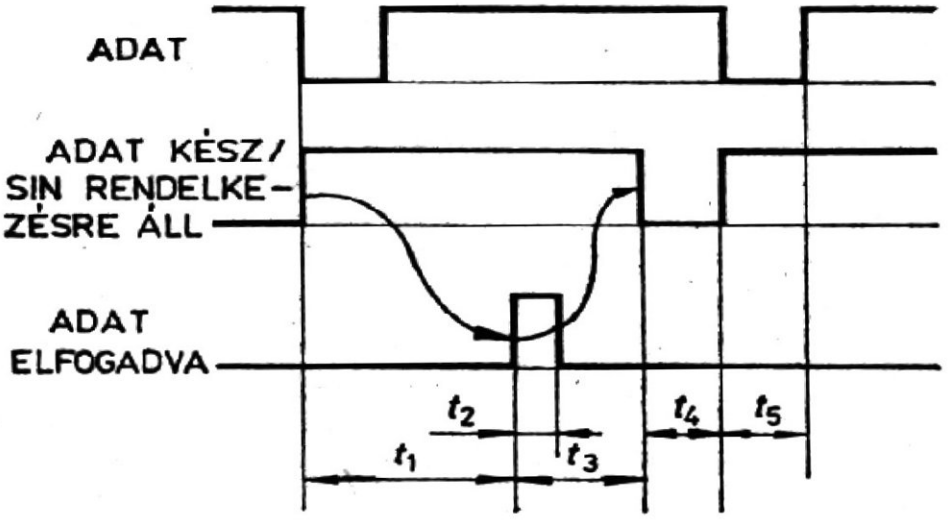
A minden következményt optimálisan kielégítő, félig összekapcsolt megoldás esetén (7.22. ábra) az ADAT ELFOGADVA jelet használjuk SÍN RENDELKEZÉS-



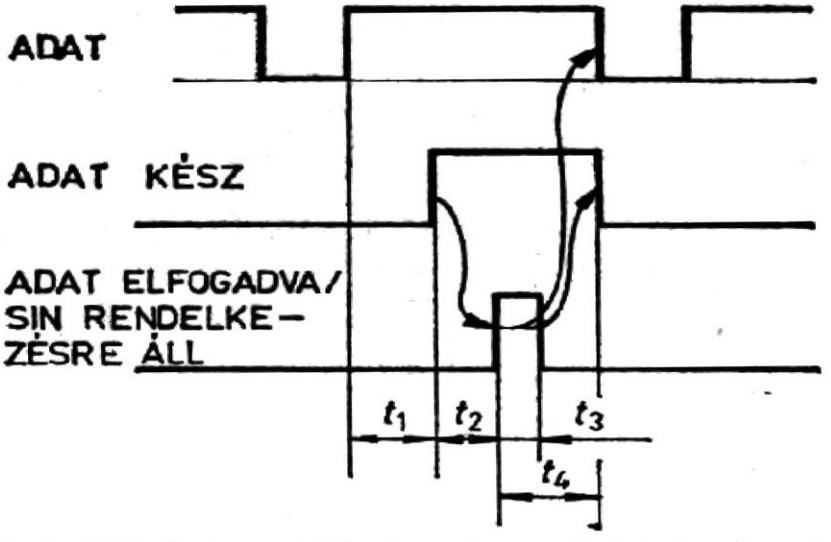
7.19. ÁBRA. Félszinkron, nem összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja



7.20. ÁBRA. Félszinkron, nem összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja



7.21. ÁBRA. Félszinkron, félig összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja



7.22. ÁBRA. Félszinkron, félig összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja

RE ÁLL jelként, hasonlóan a teljesen összekapcsolt interfészhez (7.23. ábra), ahol az ADAT ELFOGADVA hátsó éle szolgál SÍN RENDELKEZÉSRE ÁLL jelként.

Az egyes ábrák idődiagramjain feltüntettük a megfelelő megoldások konkrét tervezésekor figyelembe veendő legfontosabb időzítéseket is.

7.4. AZ ADATÁTVITEL SZERVEZÉSE

A sínen való átvitel az adatátvitel szervezése szempontjának az alapján öt alapcsoportra osztható:

- csak egyedül álló szavak átvitele,
- csak állandó hosszúságú blokkok átvitele,
- csak változtatható hosszúságú blokkok átvitele,
- egyedül álló szavak és állandó hosszúságú blokkok átvitele,
- egyedül álló szavak és változtatható hosszúságú blokkok átvitele.

(Szó itt a sínen való kommunikáció alapegységét jelenti, és a blokk ilyen szavakból áll.)

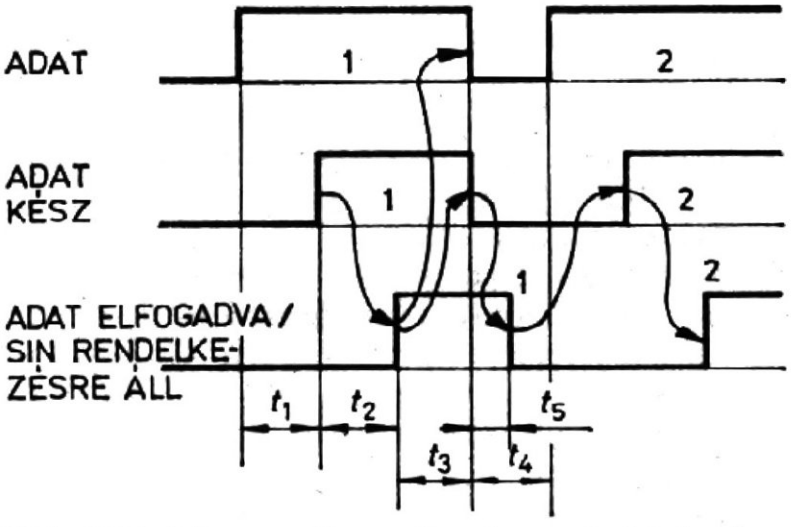
Az adatátviteli mód szoros kapcsolatban van a rendszer három másik jellemzőjével. Ezek

- a sínt használó állomás elérési (címzési) tulajdonságai,
- a sínvezérlés mechanizmusa,
- a sínen való kommunikáció módja.

Természetesen, ha olyan állomásokra gondolunk, mint pl. processzorok vagy tárak, az adatátviteli módot további tényezők (pl. programozás, tárfelosztás és -használat stb.) is nagymértékben befolyásolják [9].

Csak egyedül álló szavak átvitele

Ez a megoldás eleve kizárja blokkorientált állomások (pl. mágneslemezes tárak) effektív alkalmazását. Természetesen, ha a sín sávszélessége, a vezérlő sebessége és a sínhasználatot szabályozó algoritmus ezt lehetővé teszi, lassú állomás blokkjait lehet szavanként is kezelni.



7.23. ÁBRA. Félszinkron, teljesen összekapcsolt kérés–visszaigazolás kommunikációs módszer idődiagramja

Csak állandó hosszúságú blokkok átvitele

A sín sávszélessége ezzel a megoldással növelhető a flexibilitás rovására. Mivel a sín a teljes blokk átvitele idejére az állomás rendelkezésére áll, a vezérlő egyszerűbb és lassabb is lehet. Amíg az ADAT KÉSZ jelnek a blokkon belül minden szó esetén meg kell jelennie, a hibajelzés alapja a teljes blokk, mivel ez a legkisebb olyan egység, ami megismétlésre kerülhet.

Csak változtatható hosszúságú blokkok átvitele

Dinamikusan változtatható blokkhosszúság esetén a blokk hosszúsága az átvitelben részt vevő állomások fizikai és logikai követelményeihez alkalmazkodhat.

Egyedül álló szavak és állandó hosszúságú blokkok átvitele

Különösen alkalmas megoldás olyan esetekben, amikor nagyon nagy prioritású állomás kis gyakoriságú átvitelére és fix hosszúságú blokkok együttes átvitelére van szükség.

Egyedül álló szavak és változtatható hosszúságú blokkok átvitele

Költséges, de a legjobb hatásfokú és legflexibilisebb megoldás.

Egyedül álló szavak a blokkátvitel előkészítéséhez szükséges tevékenység (overhead: többlet) nélkül kezelhetők. A blokk hossza az állomás és az alkalmazás igényeihez illeszthető, így a sín használata optimális.

SÍNSZÉLESSÉG

A sín szélessége (width) az adott feladatra (pl. adatok átvitelére) alkalmazott vonalak száma. A vonalak megvalósításához adó- és vevőáramkörök, csatlakozóhelyek, kábelpárok szükségesek.

A vonalak száma csökkenthető több lépésben való (pl. szavak helyett byte-ok) átvitelével, a vonalak többszörös felhasználásával, vagy esetenként soros–párhuzamos átalakítással.

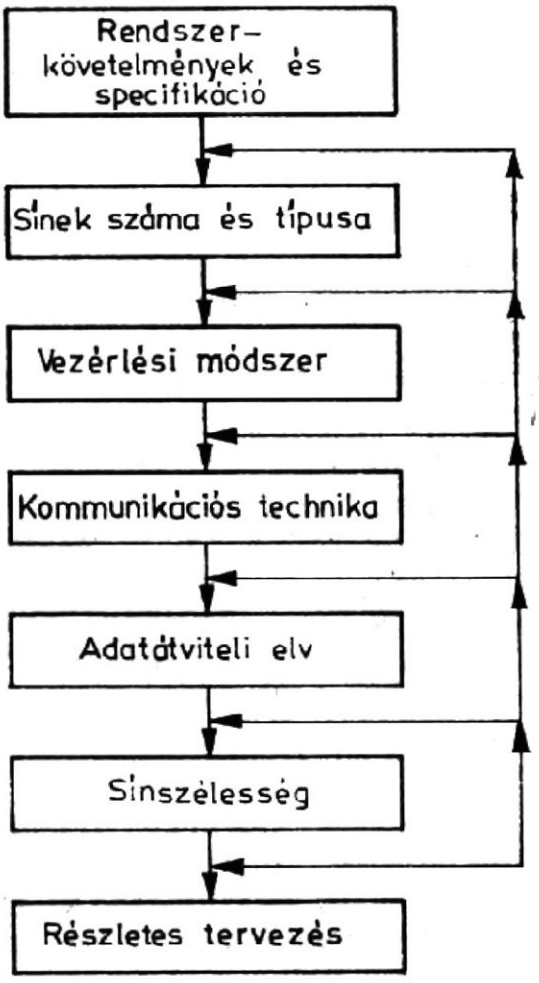
Egy digitális sínrendszer tervezésekor az előzőkben tárgyalt tényezők szisztematikus mérlegelésére van szükség. A 7.24. ábrán feltüntettük a sín strukturális kialakítása során a jellemző paraméterek megválasztásának sorrendjét. Természetesen az eljárás számos kompromisszumon (tradeoff) keresztül vezet a végső soron elfogadható megoldáshoz, ami az egész tervezés vagy azok részei többszöri ismétlése (iteráció) után érhető csak el.

A sín típusának (hozzárendelt vagy nem hozzárendelt) és az alkalmazott sínek számának meghatározása a tervezés legalapvetőbb lépése. A figyelembe veendő tényezők a következők: átbocsátóképesség (throughput), vezérlés, kommunikáció komplexitása, megbízhatóság, modularitás.

A sínvezérlés, a sín használatáért kialakuló versenyhelyzetek megoldásához az adatátviteli módok közül a feladatnak megfelelőt kell kiválasztani. Itt a fő szempontok a következők: a sin „elérésének" ideje, a vezérlés bonyolultsága (ára), modularitás, a vezérlő flexibilitása, az állomások fizikai elhelyezésének korlátai.

A kommunikációs technika szinkron, aszinkron vagy félszinkron lehet a következő fő szempontok figyelembevételével: átbocsátóképesség, megbízhatóság, a különböző állomások átviteli sebességei, a sínkihasználás hatásfoka, adatátviteli filozófia, sínszélesség.

Az adatátviteli módszer megválasztását nagymértékben befolyásolja, hogy van-e szükség blokkorientált állomás használatára. Ezen túlmenően az adatátviteli mód függvénye a vezérlés sebességének és flexibilitásának, az átbocsátóképességnek, a



7.24. ÁBRA. Sínrendszerstruktúra szisztematikus tervezésének menete

kommunikáció sebességének és technikájának, és sokszor a programozásnak (szoftvernek) és a társszervezésnek is.

A sín szélességét majdnem mindig elsősorban a sín hosszúsága vagy az átbocsátóképesség szabja meg. További szempontok: ár, megbízhatóság, kommunikációs technika és átviteli sebesség.

A sínrendszer strukturális kialakítása után kerülhet sor annak részletes megtervezésére, a konkrét áramköri elemek kiválasztására, a logikai és áramköri tervezésre.

7.5. VONALADÓ ÉS -VEVŐ ÁRAMKÖRÖK

A mikroprocesszoros digitális berendezések önálló működésű funkcionális – egymástól helyileg is elkülönülő – egységeit az információcsere biztosítása érdekében jelátvivő vezetékek, más szóval vonalak kötik össze. Mint az előzőkben láttuk, az egyes egységek összekapcsolása különböző módon történhet. A legfontosabb két megoldási mód közül az első a *közvetlen vonalak segítségével való összekötés*, amikor a vonal egyik végére egyetlen adóáramkör, másik végére pedig egyetlen vevőáramkör csatlakozik. Másik a sínen keresztüli összeköttetés, amely esetén több egység használ egy közös vonalrendszert információcsere céljából. A sínrendszer egyirányú vagy kétirányú kapcsolatot biztosíthat. Egyirányú sínre vagy egy adó és több vevő, vagy egy vevő és több adó csatlakozhat. Kétirányú sín esetén bármely modul (állomás) adhat, és az információt bármelyik egység veheti.

Az alkalmazott vonalaknál különbséget kell tennünk aszimmetrikus (unbalanced, single ended line) és szimmetrikus vonal (balanced, differential line) között [12].

Az *aszimmetrikus vonalak* esetén a vonal lényegében egy vezetékből áll, és az áramkör az egyes egységek földpontjait összekötő vezetéken keresztül záródik. Ezzel szemben *szimmetrikus vonalak* esetén a vonal két vezetékből áll, és az információ átvitelére szolgáló jel a földhöz képest szimmetrikus, tehát ebben az esetben egyik vezeték sem földvezeték. A szimmetrikus jel azt jelenti, hogy a vonal két vezetékére adott jel azonos nagyságú, de ellentétes fázisú. Mivel a vonal szimmetrikus, a zavarforrásokkal való csatolás mindkét vezetékre azonos, hasonló okból a vezetékek és a föld közötti impedanciák is azonosak. Ennek következtében a vonal vezetékein létrehozott zavarjelek azonos nagyságúak és azonos fázisúak, tehát ún. közös módusú jelek. Ugyanilyen jelként jelentkezik az egységek földpontjai közötti potenciáleltérés is. Gyakorlatban természetesen a szimmetria nem tökéletes, ezért a zavarjelnek van egy – a közös módusú komponensnél nagyságrenddel kisebb – szimmetrikus komponense is. Mivel a hasznos jel szimmetrikus, a zavarjel pedig közös módusú, vevőáramkörként megfelelően nagy közös jelelnyomású (CMR: Common Mode Rejection) áramkörrel a hasznos jel jól különválasztható [10], [11]. Gyakorlatban a szimmetrikus vonal általában egy összesodort vezetékpárból (twisted pair) áll, míg az aszimmetrikus vezetékekhez kiterjedten használják a szalagkábeleket. Egy adott fiókon belüli síntípusú kapcsolat megvalósításának ma már szinte egyeduralkodó korszerű megoldása a hátlapon (back-panel) nyomtatott formában megvalósított sínrendszer.

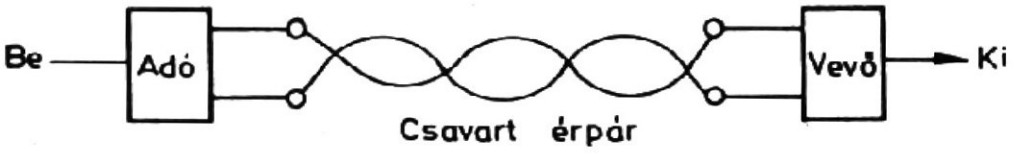
Ha az egyes egységek között nagy potenciálkülönbségek vannak, célszerű galvanikus elválasztásuk. Erre a célra legjobban az ún. optikai csatolók felelnek meg, amelyek egy jól szigetelt (1,5...5 kV), de optikailag átlátszó réteggel elválasztott fénykibocsátó diódából (LED: Light Emitting Diode) és egy fototranzisztorból vagy fotodiódából állnak.

Rövid közvetlen vonalak esetén vonaladó és -vevő áramkörként normál logikai áramköröket is használhatunk. Ilyen célra nagyobb meghajtó képességű TTL puffer kapuáramköröket alkalmazhatunk. Nagysebességű átvitelnél a vonalakat illesztetten kell lezárni. Kis impedanciájú (50...100 Ω) illesztett vonalak meghajtására sok speciális áramkört gyártanak [11]. A különböző adatátviteli eljárások (pl. CCITT V. 24) előírásainak megfelelő TTL-kompatibilis vonaladó és -vevő áramkörök is kaphatók integrált áramkörök formájában.

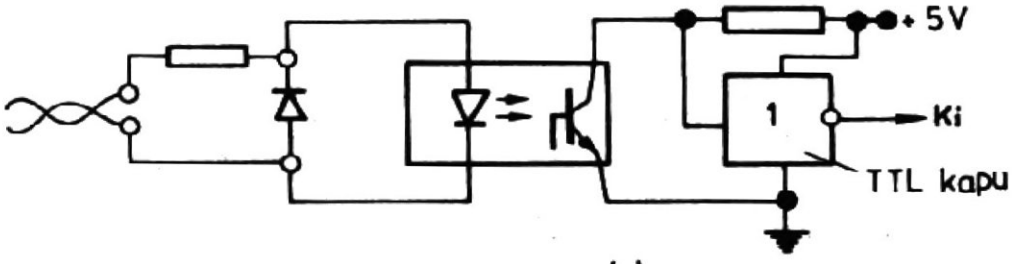
A sínek vevőáramköreivel szemben a követelmény a viszonylag kis terhelés, mivel a vonalra sok áramkör csatlakozhat. Illesztetten lezárt vonalak esetén a vevőáramkör bemeneti ellenállásának a vonal hullámellenállásánál nagyságrendileg nagyobbnak kell lennie a vevő csatlakoztatásánál fellépő reflexiók elkerülésére.

A sínvonalak egyik típusánál – ezek általában a vezérlővonalak – a vonalra dolgozó adóáramköröknek huzalozott logikai VAGY kapcsolatban kell lenniük. Ebben az esetben nyitott kollektoros kimenetű áramköröket alkalmaznak, amelyek közös kollektor-ellenállásra csatlakoznak.

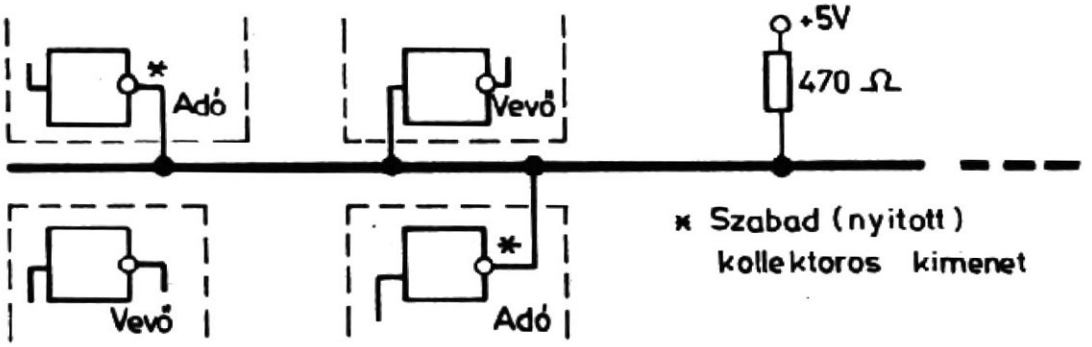
A sínvonalak másik típusa esetén – ezek általában az adat- (és cím-) vonalak – a vonalra kapcsolódó egységek közül egyidőben csak egy adhat információt a vonalra, és ilyenkor a többi adónak a vonalról lekapcsolt állapotban kell lennie. Erre a célra hosszú ideig szabad kollektoros (és szabad emitteres) áramköröket használtak, ahol a közös ellenállásra kapcsolódó kimeneti tranzisztorok közül mindig csak egy volt aktív állapotban, a többi le volt zárva és így lényegében lekapcsolódott a vonalról. Ma azonban szinte kizárólag az erre a célra kifejlesztett vonalmeghajtó áramköröket [12] alkalmazzák, amelyeknek kimenete háromállapotú. Ezek az áramkörök ugyanakkor (szemben a nyitott kollektoros áramkörökkel) mindkét „normális" logikai állapothoz kis kimeneti impedanciát biztosítanak, amely nagyobb átviteli sebességet tesz lehetővé a vonalakon.



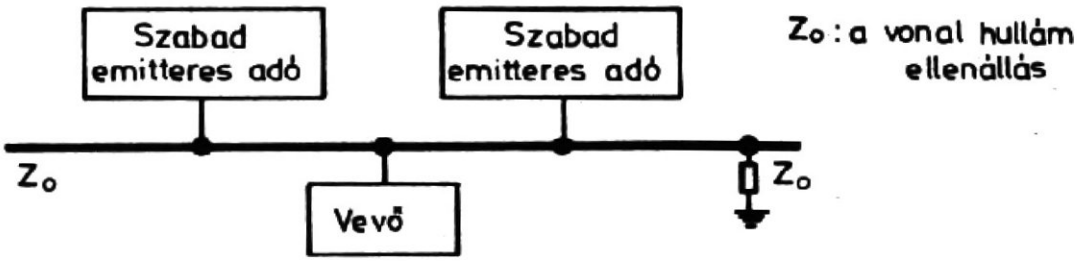
a)



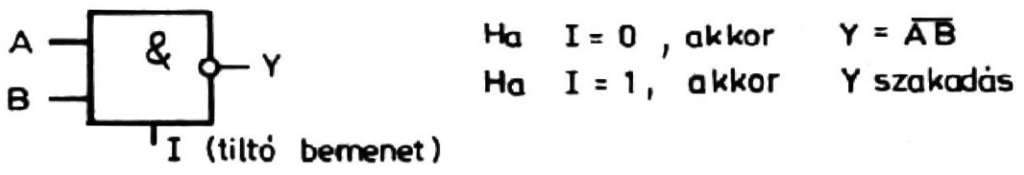
b)



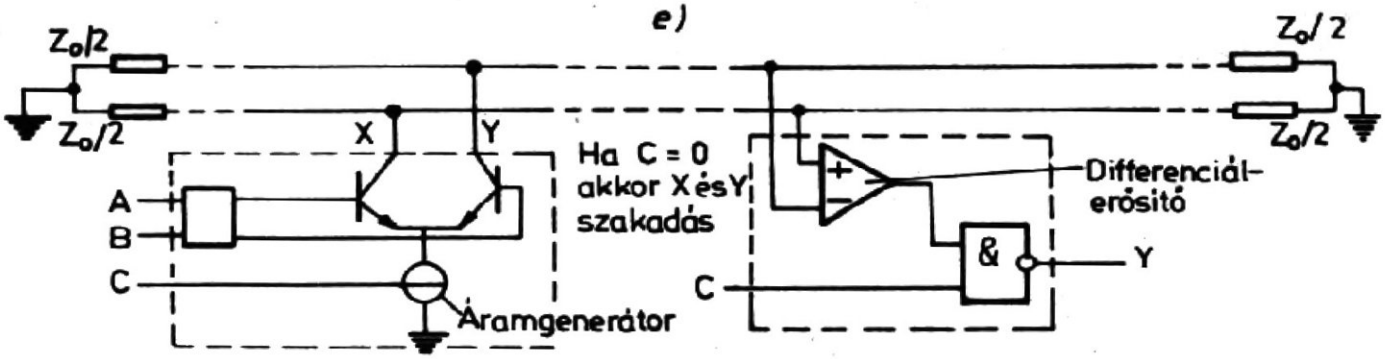
c)



d)



e)



f)

7.25. ÁBRA. Vonaladó- és vonalvevő-áramkörök *a)* szimmetrikus vonal, *b)* optikai csatoló alkalmazása, *c)* szabad kollektoros sínmeghajtás, *d)* szabad emitteres vonalmeghajtás, *e)* háromállapotú kimenettel rendelkező áramkör, *f)* szimmetrikus vonaladó és vonalvevő alkalmazása

Szimmetrikus vonalak esetén gyakran használnak áramgenerátoros kimenetű adóáramköröket. Az áramgenerátoros meghajtás kedvező, mert a vonalra rákapcsolt adó sem befolyásolja az adóponti illesztést, és a vonalra jutó közös módusú zavarjelek alig befolyásolják az adó működését. Ugyanakkor a vonal hullámellenállása is széles határok között változhat, és rövidzár sem okoz meghibásodást. A szimmetrikus vonalak vevőáramköreinek bemeneti fokozata – a nagy közösjel-elnyomás biztosítására – szinte kivétel nélkül egy differenciálerősítő [10].

A vonaladó és -vevő áramkörök megválasztása az alkalmazás körülményeinek (vonalak hossza, információátvitel szervezése, sebessége, zavarjelek nagysága), a vonalak típusának és kivitelezésnek a függvénye. Választás esetén az összes szempont mérlegelésével körültekintően kell eljárni, mivel a modulok közötti kapcsolat gyakorlati megvalósítása az egész rendszer megbízható működésének egyik kulcskérdése. A 7.25. ábrán példaként mutatunk be az említett megoldások közül néhányat.

IRODALOM

[1] *Thurber, K. J. et al.:* A systematic approach to the design of digital bussing structures. Proc. AFIPS, Fall Joint Computer Conference, 1972. 719–740.

[2] *Pirtle, M.:* Intercommunication of processors and memory. Proc. FJCC 1967. AFIPS Press 1967. 621–633.

[3] PDP–11 unibus interface manual. Digital Equipment Corporation, Marynard, Massachusetts, 1970.

[4] *Batcher, K. E.:* Sorting networks and their application. Proc. SJCC 1968. AFIPS Press, 1968. 307–314.

[5] Caprani, O.–Jensen, K. H. – Ongaard, U.: Microprocessors connected to a common memory. Proc. EUROMICRO'77. North Holland Publishing Co. Amsterdam, 1977. 175–182.

[6] *Divham, S. J.:* Fast LSI arbiters supervise priorities for bus-acces in multiprocessor systems. Electronic Design, May 24, 1979. 128–133.

[7] *Siegel, H. J. et al.:* A survey of interconnection methods for reconfigurable paraller processing systems. Proc. AFIPS, National Computer Conference 1979. New York. 529–542.

[8] *Chu, W. W.:* A study of asynchronous time divison multiplexing for time sharing computer systems. Proc. FJCC 1969, AFIPS Press 1969. 669–678.

[9] *Korn, G. A.:* Digital computer interface systems. Simulation, December 1968. 285–298.

[10] *Hainzmann J.–Horváth G.–Szegi A.:* Fejezetek a digitális elektronikából. Tankönyvkiadó, Budapest, 1975.

[11] Signetics Interface Application Memos. Signetics Corporation, 1974.

[12] Line drivers and receivers; (Application Report) Texas Instruments, Inc., Dullas, 1975.

8. OPERÁCIÓS RENDSZEREK

A mikroprocesszoros és mikroszámítógépes rendszerek létrehozása során az egyik legproblematikusabb terület a felhasználói programok gyors és hatékony kidolgozása [1], [2].

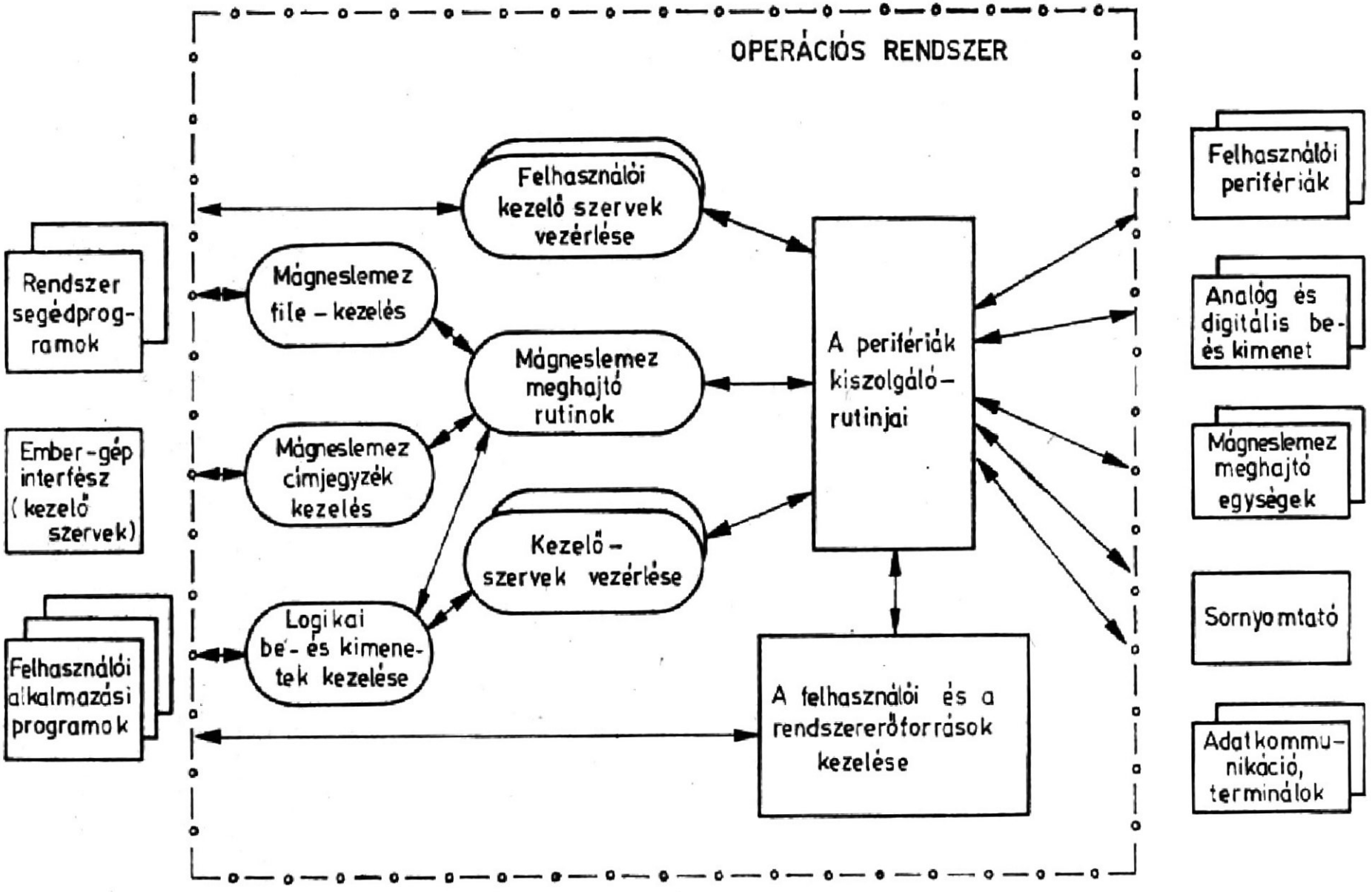
A mikroszámítógépek programozását a magas szintű programnyelvek használata és az általános célú segédprogramoknak a felhasználói programokba való beépítése lényegesen meggyorsíthatja (l. a 10. fejezetet). Ezeknek a programoknak a használata, külső tárakból a tárba való gyors behívása, a tárban való összeállítása, a megfelelő időben és sorrendben történő indításuk, egymás utáni láncolásuk, valamint a szükséges rendszerperifériák vezérlése már komoly feladat. Mindezeket a hardvert és a programokat is irányító szoftverrendszernek: az ún. operációs rendszernek (OS: Operating System) kell ellátnia. Természetesen az operációs rendszerek nemcsak a segédprogramok használatát könnyítik meg, és a magas szintű programnyelvek fordítóprogramjainak a vezérlését látják el, hanem a teljes rendszerhardver és rendszerszoftver (assemblerek, file-kezelő-, szerkesztő-, hibakeresőprogramok, különböző felhasználói programok) irányítását, vezérlését is elvégzik. Az OS indítja a programok végrehajtását, időzíti a be-, ill. kimeneti eszközök működtetését, irányítja az adatfeldolgozás menetét, a tár kezelését (a működő programok igényeinek megfelelő tárgazdálkodást), az adatok kezelését stb. [3].

A számítógépes rendszerek három alapegysége: a processzor, a tár és a be-, ill. kimeneti eszközök. A működés során e három alapegységben, valamint az egységek között utasítások és adatok áramlanak és kerülnek feldolgozásra. Az operációs rendszer feladata az ún. erőforrások kezelése, vezérlése. Az erőforrások közé tartoznak [2]:

- a processzor (a központi és a be-, ill. kimeneti processzor),
- a tár,
- a be-, ill. kimeneti eszközök,
- az információ.

Az OS-ek bonyolult programok, felépítésükkel, részletes ismertetésükkel könyvünkben nem foglalkozhatunk. A továbbiakban elsősorban a fejlesztői rendszerekkel, ezek megfelelő alkalmazásaival kapcsolatos OS-t és annak használatát tárgyaljuk.

Az operációs rendszerek a kezelő személy számára könnyen hozzáférhető, jól definiált illesztő felületet adnak a mikroszámítógéphez és programjához [4] (8.1. ábra). A programozónak a legtöbb gondot talán a háttértárakon levő programok és adatok elérése, valamint a külső átvitelek és megszakításkérések vezérlése jelenti. Nagyobb programok vagy modulárisan felépített programrendszer futtatása esetén a felhasználói szoftvert több lépésben kell végrehajtás céljából a tárban elhelyezni (átfedéses programozás: overlaying).



8.1. ÁBRA. Az operációs rendszer: a felhasználó, valamint a rendszerszoftver és rendszerhardver közötti interfész

8.1. AZ OS HASZNÁLATÁNAK ÁLTALÁNOS ELVEI

A felhasználói programok szempontjából előnyös, ha mind a kidolgozás, mind pedig a futtatás ugyanazon a mikroszámítógépen valósítható meg. Az OS akkor lehet kompatibilis mind a fejlesztési, mind pedig az alkalmazói környezettel, ha moduláris felépítésű és a felhasználói programokra vonatkoztatva „ortogonális" [4]. A modularitás előnye az OS viszonylag egyszerű kezelése, funkcióinak könnyű áttekinthetősége. Az „ortogonális" itt azt jelenti, hogy a modulok egymástól és a felhasználói programtól függetlenek, így pl. a mágneslemezes tár kezelési műveletei a feladatkezelésektől függetlenül végezhetők el. Ezek a tulajdonságok azt az előnyt is nyújtják, hogy a szoftvert is és a hardvert is a fejlesztési követelményeknek megfelelően bővíthetjük, majd a végtermékben lecsökkenthetjük a minimális futási idő és tárkapacitás elérése érdekében. Például a programok perifériakezelő rutinjai (handler) szükség szerint tetszőlegesen bővíthetők vagy csökkenthetők.

A be- és kimeneti műveletek kezelése (Input/Output management)

Az OS-ek be-, ill. kimeneti műveletek kezelését végző programmoduljainak a felépítése általában a fizikai perifériák jellemzőitől független felhasználói programok megírását és programfejlesztést tesz lehetővé. Ez a mikroszámítógépes rendszerekben, amelyek több kül önbözőtípusú perifériához csatlakoztathatók (8.1. ábra), úgy érhető el, hogy a programrészek az átviteli kéréseket (utasításokat) az ún. logikai egységekhez továbbítják. Logikai egységen (logical unit) a továbbiakban a perifériákat és az őket kiszolgáló programrészeket értjük. Például egy logikai egység: a kezelői terminál (írógép, képernyős megjelenítő) és az arról való beviteleket vezérlő programrutin. Egy másik logikai egység: a terminál és a kimeneti programrutinja [5]. A felhasználói program a logikai egységeket egységes elvekkel, egységes adatstruktúrával használja (l. később), függetlenül a tényleges fizikai perifériától. Természetesen az OS csak akkor képes egy adott perifériát kezelni, ha az OS programjai között szerepel az adott kiszolgáló programrutin, valamint ha előzőleg a felhasználói program vagy a gépkezelő az átvitelkéréshez hozzárendelte a fizikai perifériát.

Azt a programot, programmodult vagy programfunkciót, amelyik az átvitelkéréseket továbbítja a logikai egységnek, a következőkben eszköznek vagy eszközkezelőnek (device handler) nevezzük [5]. Maga az OS is tartalmaz a gépkezelő és a felhasználói program számára elérhető eszközöket. Az eszközöket használatuk előtt a tárba be kell táplálni és aktivizálni kell. Az egyszerűbb eszközök, mint pl. a képernyős megjelenítő vagy az írógép vezérlője a tárba töltéssel már aktívvá is válhatnak, a bonyolultabb eszközök részére külön aktivizáló vagy inicializáló parancsok küldésére is szükség lehet.

A perifériákat és a logikai egységeket tekintve megkülönböztethetők ún.:

- címjegyzék (directory) szervezésű logikai egységek,
- nem címjegyzék szervezésű egységek [4].

Az előbbiekre példa a hajlékony mágneslemezes tár. A címjegyzék segítségével a mágneslemezen levő programok és adatok (file-ok) azonosíthatók és jelezhetők. Ez egy definiált helyen található adattömb, amelynek a tartalma jelzi, hogy a mágneslemezen milyen programfile-ok találhatók, és milyen címeken (lemezhelyeken) vannak. A logikai egységek kezelőprogramjai lényegesen bonyolultabbak, ha a periféria adatai címjegyzék szervezésűek. Az OS-en belül kell a kétféle kezelést különválasztani, hogy a felhasználói program a kéréseket már egységesen továbbíthassa minden perifériának.

A tár kezelése (memory management)

A mikroszámítógépek programjait és adatait tárban, valamint a háttértárakon (ha van ilyen a rendszerben) tárolják. A háttértáron levő programok vagy programmodulok együttes mérete többszörösen meghaladhatja a tár kapacitását. Modulárisan felépített programok használhatók úgy, hogy a tárba csak a pillanatnyilag használt vagy a leggyakrabban használt programmodulok kerülnek be, a többi a háttértárban van [2], [6].

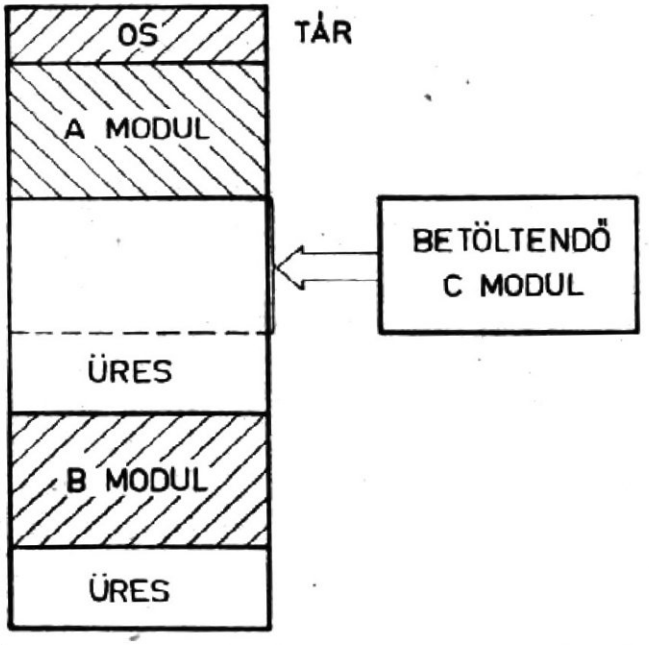
A rugalmasabb programrendszerek lehetővé teszik, hogy a programmodulok áthelyezhetők (relokálhatóak) legyenek. Ez annyit jelent, hogy a programmodul által elfoglalt tárcímterület változtatható az igényeknek megfelelően. A modul a tárban levő többi modul által el nem foglalt területre kerül be (8.2. ábra). Az áthelyezhetőség megköveteli, hogy a programmodulban használt összes tárhivatkozási művelet címrésze módosítható legyen.

Az áthelyezés eljárásai szerint megkülönböztetünk:

- hardverrel támogatott címeltolást,
- szoftverrel megoldott relokálást.

Az előbbi esetben a processzor ún. báziscímregisztert tartalmaz. Ebbe a báziscímregiszterbe kerül be az áthelyezési báziscím a tárba töltés helyének megfelelően. A báziscím megadását az OS tárkezelő rutinja végzi. A tárhivatkozási utasítások címrésze csak a programon belüli relatív cím. Az abszolut címet minden utasítás végrehajtásakor a processzor hardverje képezi a bázis- és a relatív cím alapján. A program tehát a tárba való behelyezési címtől függetlenül mindig ugyanazokat a címadatokat tartalmazza. A báziscímregiszterek alkalmazása a mikroprocesszorokban még nem általános (kivétel néhány újabb típus, pl. MOTOROLA 68000, ZILOG Z–8000 [7], [8], [9]).

A szoftveráthelyezés nagyobb feladatot hárít az operációs rendszerre, de a programozóra is. A programok megírásakor a programozó vagy áthelyezhető, vagy át nem helyezhető programot ír. Az utóbbi esetben a program mindig csak egy helyre tölthető be, és ún. abszolút tárcímekre hivatkozik. Áthelyezhető programok esetén a háttértárban található tárgyprogram-file-ban jelezni kell a tárreferenciák adatait. Az OS ún. csatoló- (linker vagy linkage editor) programja a tárba való be-



8.2. ÁBRA. Az áthelyezhető programmodul tárba való betöltésének a vázlata

töltés során módosítja ezeket az adatokat az aktuális betöltési helynek megfelelően. A tárba a töltés az áthelyező töltőprogrammal végezhető el. Az áthelyezhető programok programváltozóinak (adatainak) négy típusa:

- abszolút című adat,
- belső adat,
- globális adat,
- külső (external) adat.

Az abszolút című adatok megegyeznek a nem áthelyezhető programokkal kapcsolatban elmondottakkal. A belső adatokat csak a programmodulon belül használjuk. A globális és a külső adatok olyan programváltozók, amelyeket több külön-külön áthelyezhető programmodul használ. Globális az adat, ha az adott modulon belül definiáljuk, vagyis itt jelöljük ki a helyét. A külső adatot egy másik programmodul definiálja, de az adott modul is hivatkozik rá. A globális és a külső adat tehát relatív fogalom, egy adat az egyik programmodul globális adata, míg a másiknak a külső adata. A csatolóprogram helyesen csak úgy tudja kezelni ezeket az adatokat, ha a tárgyprogram-file-ban a változók típusát is jelöljük.

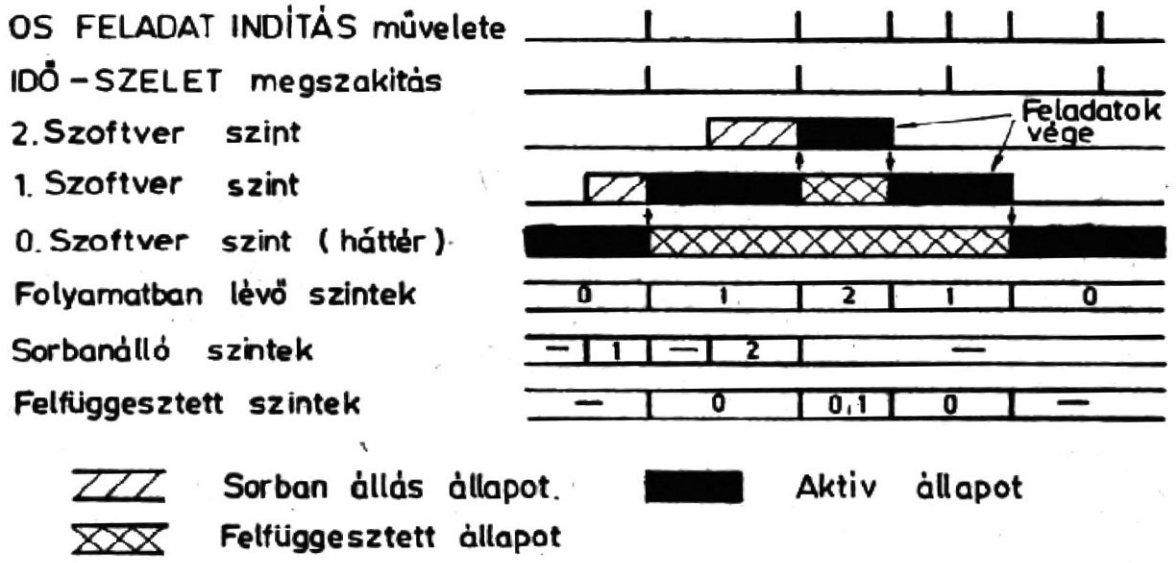
Természetesen az áthelyezhető modulok használatához nemcsak megfelelő csatolóprogram szükséges, hanem megfelelő fordítóprogramok is kellenek, amelyek a fordítás közben készülő tárgyprogramba az áthelyezéshez szükséges jelzéseket is beépítik [10].

Az OS a rendszer- és a felhasználói programok működése közben folyamatosan nyilvántartja a tármezők foglaltságát. Ha nem fér már be a program a tárba, az OS a kezelőnek jelzést adhat. Egyszerűbb operációs rendszerek a programmodulok tárba töltését kezelői parancsra vagy a felhasználói program parancsára végzik. A bonyolultabb rendszerek a programot képesek automatikusan a szabad tárhelyekre betölteni, vagy a már nem szükséges programrészek adatainak a háttértárra való visszahelyezésével felszabadítani a tárrészeket újabb betöltés előtt.

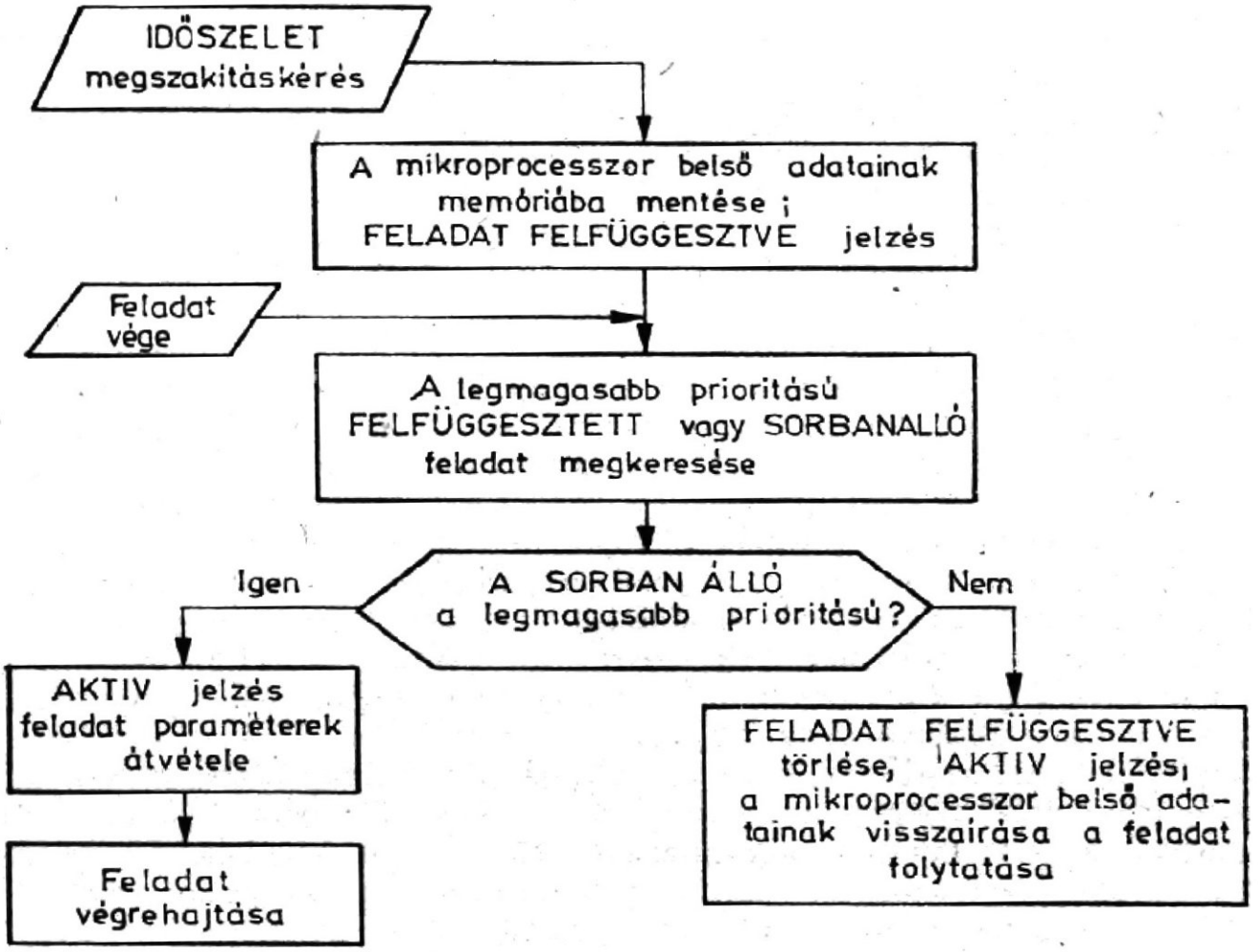
A feladatok kezelése

Sok alkalmazáshoz, de különösen az azonos idejű adatfeldolgozási és vezérlési műveletekhez a mikroprocesszoros rendszerek esetén is szükség van több szoftverfeladat egyidejű ellátására. Egy adott idő alatt a mikroprocesszoros központi egység csak egy feladattal tud foglalkozni [11]. Az azonos idejű alkalmazások operációs rendszerei általában lehetővé teszik több feladat kezelését, több alkalmazói programmodul futtatását. A feladatok egymás utáni időzítését, láncolását feladatütemezésnek (task scheduling) nevezzük. Példaként ismertetünk egy ilyen eljárást, amelynek lényege a processzor futási idejének időszeletekre (time slice) bontása periodikusan generált megszakításkérő jelekkel. Ez a megszakításkérés a szoftverfeladatok váltásának, új feladat indításának az időpillanatát jelenti. Továbbá az új feladatra való rátérés feltétele lehet az aktív feladat programjának a befejeződése. A 8.3. ábrán egy ilyen feladatidőzítési eljárást mutatunk be. A diagram három feladatot tüntet fel, a feladatok száma azonban az alkalmazástól függően több is lehet. A feladatoknak ún. szoftverprioritási szintjük van, a példában a magasabb sorszám magasabb prioritást jelent. Természetesen a szoftverprioritások és az időszeletes megszakításkérés alkalmazása nem zárja ki a gyorsperifériáktól érkező további megszakításkérő jelek használatát. Ezeknek a halasztást nem tűrő perifériáknak a kiszolgálását a feladatoknál is magasabb prioritási szintű programrutinok végezhetik el.

A feladat aktív állapotban van, ha a mikroprocesszor éppen a hozzátartozó programot hajtja végre. Magasabb szintű feladatra való rátéréskor a „félbehagyott" feladat felfüggesztett állapotba kerül. Egy feladat előkészítése kész, vagyis „sorbanállás" állapotban van, ha végrehajtását a processzor megkezdheti, amint az ehhez



8.3. ÁBRA. Az OS feladatkezelő funkcióinak és a feladatok végrehajtásának időDiagramja [11]



8.4. ÁBRA. Az OS feladatkezelő funkciójának a folyamatábrája

szükséges feltételek teljesülnek. Több felfüggesztett vagy sorbanálló feladat esetén az OS feladatkezelő rutinja a várakozók és az aktív közül a legmagasabb prioritású futását engedélyezi. A feladatkezelés döntési folyamatábráját a 8.4. ábrán tüntettük fel.

A feladatoknak a külvilággal való adatcseréken kívül szükségük van az egymás közötti kommunikációra. Az üzenetet küldő programmodul a továbbítandó információt általában a tárba helyezi, majd erről jelzést küld a fogadó programmodulnak. Az egyik gyakran alkalmazott megoldás szerint az üzenetekből összeálló táblázatot a feladatkezelő tartja nyilván. Gyakran két táblázat is szerepel az OS feladatkezelő részében, az egyik az érvényben levő üzeneteket és a hozzájuk tartozó küldő és fogadó feladatok azonosítóit tartalmazza, míg a második táblázatban azoknak a feladatoknak az azonosítói vannak, amelyek üzenetet kérnek [12]. Alkalmazástól függően az üzenetet kérő feladatok lehetnek olyanok, amelyek mindaddig várnak, amíg üzenetet nem kapnak, vagy pedig olyan programrészek, amelyek az üzenetre csak bizonyos ideig várnak.

Információkezelés

A legtöbb információkezelési nehézség a háttértárakon levő programoknak és adatoknak a használatával kapcsolatos. Ezeknek az adatoknak az azonosításához, megkereséséhez bonyolult algoritmusokra és bonyolult programokra van szükség. A mikroszámítógépekkel kapcsolatosan a hajlékony mágneslemezes háttértárak terjedtek el. Az információkezelésre tipikus példa a hajlékony mágneslemezeket kezelő program, a hajlékony mágneslemezes operációs rendszer.

A mágneslemezeken a programokat és az adatokat ún. file-ok formájában tárolják. A bonyolultabb mágneslemezes háttértárak és kezelőprogramjaik megengedik, hogy egy mágneslemezen belül a különböző file-okat különböző hosszúságú rekordok tartalmazzák. Így pl. a rekord mérete lehet 128, 256, 512, 1024, 2048 vagy 4096 byte. Egy file-on belül azonban általában nem változik a rekordok mérete. A tárolási eljárás természetéből adódik, hogy a file-t csak egész számú rekord alkothatja, ezért csak a hosszabb file-ok tárolásához célszerű nagyobb rekordméretet alkalmazni.

A mágneslemezes tárak írható–olvasható tárak. Lehetőség van file-ok törlésére, és a felszabadult helyen egy másik file tárolására. Ez azt is jelenti, hogy a hatékony lemezkihasználáshoz nem megfelelő a file adatainak folyamatos felírása, az összetartozó adatokat tartalmazó rekordok folyamatos egymás utáni elhelyezése. Ezért, valamint az egy mágneslemezen levő sok file használata miatt a mágneslemezen való tároláshoz azonosítókra van szükség. Az azonosítók lehetnek:

- rekordazonosítók (a rekordok címei),
- file-azonosítók (a file-ok nevei).

A rekordok címei a mágneslemezes tárat kezelő program szempontjából hasonlóak az adatok tárcímeihez, csak nagyobb adattömegre vonatkoznak. A file-ok azonosítási folyamata viszont mindazoknak a rekordoknak és a rekordok egymás utáni sorrendjének a megkeresését és meghatározását jelenti, amelyek az adott file összes adatát tartalmazzák.

A file-okra a file-nevekkel hivatkoznak. A mágneslemezen tárolt file-ok eléréséhez a címjegyzékfile-t használják. A file-ok elérését végző program tehát csak a címjegyzék kiolvasása és a tartalmának az értelmezése után tud olvasni, írni vagy módosítani egy-egy file-t. Természetesen a tartalmát is módosítani kell a mágneslemezestár-műveletek után, ha azok valamely file adataiban olyan módosítást eredményeznek, ami a file-hoz tartozó rekordok láncolatát is érinti. A szerteágazó és

bonyolult funkciói miatt gyakran a mágneslemezes tár file-jait kezelő programot is OS-nek vagy mágneslemezes (diszk-) operációs rendszernek (DOS) nevezik.

A DOS tehát az operációs rendszernek a háttértárat kezelő programmodulja. Lényeges, hogy az adatkezelések a DOS-t használó programban függetlenek legyenek a háttértárat jelentő periférián használt file-szervezés típusától. Követelmény, hogy az adatkezelési parancsok megadásához elegendő legyen megadni az átvitel forrásának és végcéljának az azonosítóit (mágneslemezes tár átvitele esetén a file nevét és a tárcímet), valamint az átviteli parancshoz tartozó egyéb információkat (írás, olvasás, aktivizálás stb.). A DOS-t részletesebben a következő részben tárgyaljuk.

8.2. OS FUNKCIÓK ÉS ALKALMAZÁSOK

Az operációs rendszerek alkalmazását, funkcióit és használatukat egy konkrét OS ismertetésén keresztül mutatjuk be [13]. A bemutatásra kerülő operációs rendszer az ún. áthelyezhető modulokat tartalmazó és be-, ill. kivitelek kezelését végző OS, vagy röviden RIO (**R**elocatable Modules and **I/O** Management), fejlesztési célokra készült. Segítségével felhasználói programok készíthetők, ellenőrizhetők, de ez a szoftverrendszer lehetőséget nyújt arra is, hogy áthelyezhető moduljait a felhasználói programba beépíthessük. A programrendszer főbb szoftveregységei [13], [14]:

- PROM-ban elhelyezett minimális töltőprogram (bootstrap loader),
- PROM-ban levő hibakereső programcsomag a fejlesztői rendszerrel való programellenőrzés támogatására,
- áthelyezhető makro-assembler és csatoló- (linker) program,
- szerkesztőprogram (editor),
- az operációs rendszer, és funkcióvégrehajtó moduljai (ide tartozik a mágneslemezes tárkezelő is).

A felsorolt szoftveregységek közül e fejezetben csak az utóbbival foglalkozunk, a többit csak annyiban érintjük, amennyiben ez az OS ismertetéséhez szükséges. A hibakereső program funkcióit bővebben a 4. fejezet, az assemblert, a csatoló- és a szerkesztőprogramot a 9. fejezet tárgyalja.

A RIO hajlékony mágneslemezes tár alapú. Ez azt jelenti, hogy mind az OS moduljai, mind a szoftveregységek hajlékony mágneslemezes táron találhatók, s onnan töltik be őket a tárba. A felhasználói programok fejlesztését és végrehajtását is alapvetően a hajlékony mágneslemezes háttértár biztosítja. A RIO felügyelete alatt készített programokat mágneslemezen tárolja a rendszer.

A RIO file-okkal dolgozik. A file-ok programként végrehajtható modulokat vagy felhasználói programokat jelentenek, de jelenthetnek feldolgozandó adatokat is valamely programfile számára. A be-, kivitelek lebonyolítása a logikai egységek segítségével történik. A logikai egységek használatához egy egységcím (unit number) vagy cimke (label) csatolható az egység programon belüli azonosításához, vagy a reá való hivatkozáshoz. Az egységet kezelő vagy kiszolgáló programrészt eszköznek vagy eszközmeghajtónak nevezzük. Így eszköz a soros vonalat kiszolgáló programrutin, de eszköz például a mágneslemezes operációs rendszer, a DOS is. Egy adott programhoz vagy programfile-hoz ezek az eszközök – és természetesen a logikai egységek is – hozzárendelhetők.

A RIO be-, kimeneti struktúrája az összes eszközmeghajtó számára egységes illesztőfelületet definiál. A RIO az ún. aktív eszközök táblázatában (ADT: **A**ctive **D**evice **T**able) és a logikai file-ok táblázatában (LFT: **L**ogical **F**ile **T**able) található adatok alapján vezérli az átviteleket. Az ADT tartalmazza az aktív eszközök

azonosítóját és az eszközprogramrész belépési pont címét. A logikai egységet és az eszközt az LFT adatai rendelik össze. Az ADT és az LFT adatainak a megadását a RIO parancsok között tárgyaljuk.

Az átvitelkéréseket a RIO egy meghatározott programbelépési pontjára (SYSTEM-cimke) való rátéréssel (CALL utasítással) indíthatjuk. Ezt megelőzően az átvitelkérés paramétereit a tárban elhelyezett ún. paramétervektorral és a vektormutató beállításával adjuk meg. A vektorformátumot a 8.1. táblázat tartalmazza.

8.1. TÁBLÁZAT. Az átvitelkérések paramétervektora

	Tárrekesz	Tartalma
Mutató (IY regiszter)	0	logikai egység sorszáma
	1	átviteli parancs kódja
	2,3	átviteli cím (tárcím)
	4,5	adatblokk hossza (byte-szám)
	6,7	befejezés utáni visszatérési cím
	8,9	hibás befejezés utáni visszatérési cím
	10	befejezés állapotjelző kódja
	11, 12	kiegészítő információ

Az átviteli parancskód az eszköz műveletét határozza meg (pl.: kivitel, bevitel, egy adat átvitele, blokk átvitele, átvitel egy blokkvégjelző kódig stb.). Befejezés után a visszatérési cím a meghívási helyet (CALL SYSTEM utasítást) követő cím, vagy a paramétervektorban megadott cím, az átviteli parancskódban található jelzéstől függően. Átviteli hiba esetén a visszatérési cím eltérhet a helyes működéshez előírt visszatéréstől. Az eszköz az átvitel befejezését követően állapotkódot (pl. hibák jelzésére) közöl az őt meghívó programmal. A kiegészítő információra vonatkozóan példát a DOS ismertetése során mutatunk. Ennek értelmezése a meghívott eszköztől függ.

Az átviteli parancs lehet fiktív átviteli kód is. Ebben az esetben a RIO nem továbbítja a meghívott eszköznek a paramétervektort, hanem a parancskódnak megfelelően bizonyos átalakításokat végezhet a paramétervektorral. Így pl. logikai egység hozzárendelése kérhető, ekkor a RIO és a DOS megkeresi, hogy az átviteli címmel jelölt helyről kiolvasható file-névnek megfelelő file melyik hajlékony mágneslemezes táron található, vagy a RIO a kiegészítő információhoz áttölti a file nevét stb. Ez a parancs a hajlékony mágneslemezes tár file-műveleteinek előkészítését segíti.

A programáthelyezéshez a tár szegmentálását a RIO 128 byte-os egységekben végzi. Egy programmodul csak egymást követő szegmensekbe kerül. A programmodult tárba hívása előtt áthelyezi a kívánt helyre a csatolóprogram, tehát a RIO nem végez automatikus tárkezelést. A RIO a foglalt szegmenseket nyilvántartja, ha foglaltság miatt a betöltésre nem keríthet sort, erről hibajelzéssel értesíti a kezelőt.

A RIO-ban a DOS a legfontosabb eszköz (master device). Az operációs rendszer működésének tárgyalásához elkerülhetetlen a DOS-nak, a DOS funkcióknak a megismerése.

A RIO hajlékony mágneslemezes operációs rendszere

A DOS a mágneslemezen levő információ tárolását és annak a mikroszámítógép részére való elérését biztosítja. A RIO-ban a file-ok azonosításához a következő tulajdonságjelzők (attributumok) tartoznak: a file neve, típusa, altípusa és „fajta" jellemzői (property), valamint a file-t alkotó rekordok hossza (ez a RIO-ban programozhatóan változtatható).

A file-okra elsősorban a file-névvel hivatkozunk, amit a programozó (gépkezelő) adhat meg.

A file-ok típusai a következők:

- címjegyzékfile (directory),
- eljárásfile (procedura). Ez olyan file, amelyet a RIO a tárba való betöltés után rögtön végre tud hajtani,
- bináris file. Ez bármilyen adatot, adatformátumot tartalmazó file,
- szövegfile. Ez az ASCII kódnak megfelelő formátumú adatokat tartalmazó file; ilyent állít elő a szerkesztőprogram.

A file-oknak több különböző altípusa lehet. A felhasználó az azonos célú, de módosított vagy javított file-ok megkülönböztetésére különböző altípuskódokat adhat.

A file-ok fajtái a következők:

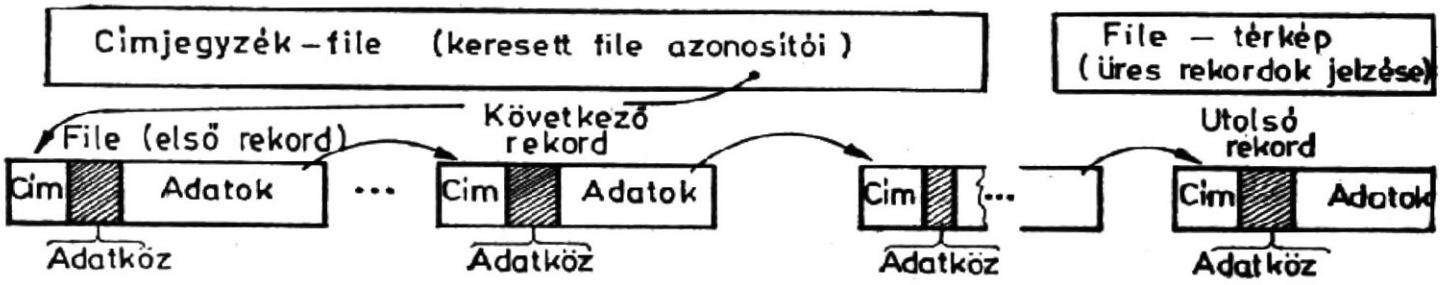
- írásvédett file – tartalma nem változtatható meg,
- törlés-védett file – tartalma nem törölhető,
- zárt attributum file – az attributumok (név, típus, altípus, fajta) nem változtathatók meg,
- titkos – a címjegyzéknek a képernyős megjelenítőre való kiírásakor nem megjelenített file.

A DOS egyidejűleg több hajlékony mágneslemezes egységgel (max 8) és több file-lal tud együttműködni. Több file-lal való együttműködés alatt azt kell érteni, hogy a DOS-t meghívó programok egyszerre több file-t használnak (pl. másolás során egyik file-t olvassák, míg a másik file-t felírják a mágneslemezre), vagyis egyszerre több file található „megnyitott" állapotban. Megnyitott az a file, amelyiknek az olvasásához vagy írásához a DOS a címjegyzék adatait és az összetartozó rekordok azonosítóit (címeit) az elvégzendő művelethez nyilvántartja.

A DOS főbb feladatai röviden a következők:

- Megállapítja, hogy a hajlékony mágneslemezes egységek közül melyek működnek (melyekben van átvitelre kész mágnestárcsa).
- A paramétervektorban közölt file-név, -típus, -altípus stb. alapján megállapítja, hogy szerepel-e ilyen a működő hajlékony mágneslemezes egységek lemezein (a címjegyzékben), s ha igen, melyiken.
- Létező file-nak vagy alfile egy vagy több rekordjának a beolvasását, átírását (módosítását), törlését végezheti el, és új file-t képezhet a mágneslemezen.

A 6.4. szakaszban a 6.28. ábrán bemutattuk, hogy a rekordokat a mágneslemezen elfoglalt fizikai helyük szerint hogyan lehet azonosítani. A 8.5. ábrán vázlatosan tüntettük csak fel a rekordhoz tartozó azonosító adatot (címet) és a file adatait (adatot). Ez az ábra vázlatosan mutatja a kérdéses file rekordjainak megkeresését. A file nevét a címjegyzék tartalmazza, és első rekordjára is „rámutat". A további rekordok azonosításának két módja terjedt el. Az első esetén – ezt használja a RIO DOS programja is – minden rekord tartalmaz ún. csatolóadatokat, amelyek a következő rekord címét megadják. Az utolsó rekord csatolóadata különbözik bármelyik címtől; jelzi, hogy további keresésre nincs szükség. A második szerint a címjegyzék tartalma nemcsak az első, hanem a többi rekord címét is megadja.



8.5. ÁBRA. File-kezelési eljárás és az egy file-hoz tartozó rekordok azonosítása

8.2. TÁBLÁZAT. A RIO mágneslemezes operációs rendszer parancsai

Parancs	Rövid leírása
INITIALIZE	A működő hajlékony mágneslemezes egységek, mágneslemezes egységekben levő mágneslemezek azonosítása
ASSIGN	A file-név és a hajlékony mágneslemezes egység összerendelése. A paramétervektortól függően a DOS megkeresheti, hogy melyik egységben levő mágneslemezen található a file
OPEN	A file címjegyzék adatainak a beolvasása és a file behelyezése az „aktív eszközök" táblázatába
CLOSE	A file-lal végzett műveletek befejezése, a file törlése az aktív eszközök táblázatából. A címjegyzék tartalmát is aktualizálja a DOS
READ	A megnyitott file egy vagy több rekordjának a beolvasása. Többféle olvasási parancs adható, mely az olvasandó rekordra vonatkozik (a soron következő, vagy az előző rekord, vagy az aktuális rekord ismételt olvasása)
WRITE	Egy vagy több rekord írása
DELETE	Egy vagy több rekord törlése
ERASE	A teljes file törlése a címjegyzékből és az általa lefoglalt rekordok üresnek nyilvánítása a file-térképen
SKIP	A file egy vagy több rekordjának olvasás vagy írás nélküli átlépése (l. a 8.5. ábrát), új rekordra pozicionálás. A parancs előre vagy visszalépésre is vonatkozhat
REWIND	A file első rekordjára való visszalépés
RENAME	A file nevének a címjegyzékben való megváltoztatása
UPDATE	A file attributumainak a megadottra való megváltoztatása
QUERRY ATTRIBUTES	A file-attributumok lekérdezése (kiolvasása)

Új file képzésénél a DOS a címjegyzék ún. file-térképét (map file) használja. Ez tünteti fel, hogy mely rekordok foglaltak, s melyek üresek. Természetesen minden file-művelet befejeztével a DOS az aktuális értékeknek megfelelően módosítja a címjegyzék és a file-térkép tartalmát is.

A hajlékony mágneslemezes műveletek elvégzéséhez a paramétervektorban meg kell adni a file nevét, típusát, a hajlékony mágneslemezes egység sorszámát (több, egyszerre működő mágneslemezes egység esetén). Ezeknek a paramétereknek a tárban levő címét általában a paramétervektor kiegészítő információt tartalmazó rekeszeinek tartalma adja meg.

A DOS a műveletek után hibás átvitelek, nem teljesíthető átviteli parancsok, nem létező file keresését előíró műveletek stb. miatt hibaüzenetekkel válaszolhat az őt meghívó programnak. Arra nincs mód, hogy könyvünkben a DOS parancsokat és hibaüzeneteket teljes részletességükben ismertessük, ezért csak röviden és vázlatosan mutatjuk be ennek a példaként választott mágneslemezes operációs rendszernek a műveleteit (8.2. és 8.3. táblázat).

8.3. TÁBLÁZAT. A RIO mágneslemezes operációs rendszerének hibaüzenetei

A hibaüzenetek leírása
Hibásan megadott műveleti (átviteli) parancskód
A kijelölt mágneslemezes egység nem működik (nincs benne mágneslemez)
Írásvédelmi hiba, az írásparancs át nem írható file-ra vonatkozott
Szektorhiba, a rekordokat képező mágneslemez-szektorok azonosítóiban észlelt hiba
Adatátviteli hiba, a hibajelzést szolgáló CRC adat hibája
A megadott nevű file nem létezik a mágneslemezeken
File vége, további olvasási, írási, léptetési parancs nem adható
Nincs megnyitva a file
A rekordazonosítások hibája
Megnyitott file újranyitási kísérlete
Nem létező mágneslemezes egység kijelölése
Már létezik ilyen nevű file (új file felírásakor jelentkező hiba)
Tele a mágneslemez, további írás nem végezhető
Rossz átnevezési parancs (nincs ilyen megváltoztatandó név).
Attributumváltoztatási parancs nem megváltoztatható attributumú file-névvel

A RIO végrehajtóprogram

A RIO végrehajtóprogram (RIO Executive) valósítja meg az egységesített be-, ill. kimeneti műveletek kezelését és nyitott programmodul-rendszerből álló hajlékony mágneslemezes file-kezelési struktúrát képez. A végrehajtóprogram egy tárrezidens részből áll (ez az OS). Ez tartalmazza a be-, kiviteleket kezelő rutinokat, valamint a DOS-sal együttműködő file-kezelő (a hajlékony mágneslemezekről file-

okat behívó és végrehajtó) szubrutint. A képernyős megjelenítőről vagy írógépről néhány parancsot fogad el. Az ezeket a parancsokat értelmező és végrehajtó szubrutinok képezik a file-kezelő részt. A tárrezidens rész által értelmezett parancsokat belső parancsoknak (internal command) nevezzük. A RIO külső parancsait (external command) a hajlékony mágneslemezen található egyéb programfile-ok hajtják végre. Ezeket csak a működésük időtartamára hívja be a tárba az OS, a működésük befejeztével, s újabb parancsra való rátéréssel a tárba az újabb programmodul kerül. A RIO struktúrája nyitott, azaz a külső parancsok készlete bármikor bővíthető a felhasználói file-okkal. Egy-egy parancs végrehajtását a RIO OS-e a parancs kódjának a megadása után (a file nevének a megadása után) automatikusan kezdeményezi. A RIO tárrezidens részen keresztül a programmodulok kommunikálhatnak egymással.

A RIO OS és a modulok elhelyezkedésének a vázlatát a 8.6. ábrán tüntettük fel. A külső modulok mérete és száma igen széles határok között változhat.

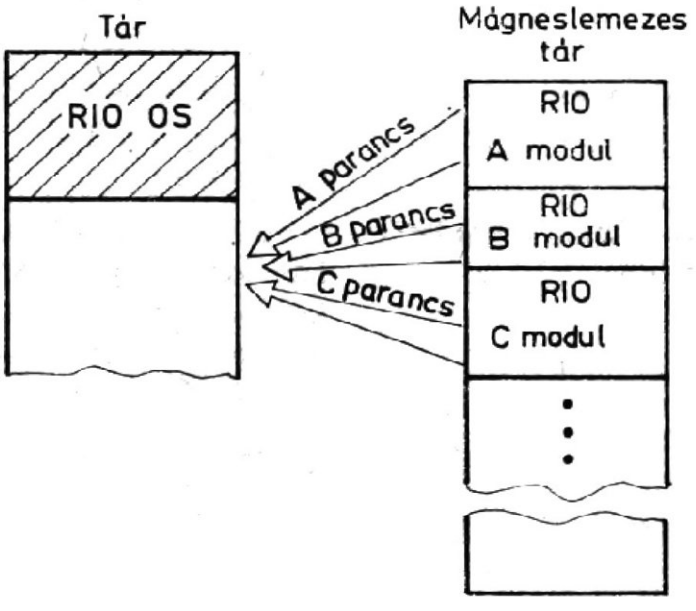
A RIO működésének két alapvető módja:

- a parancsoknak a képernyős megjelenítőn keresztüli (kézi) hívási és végrehajtási módja,
- a felhasználói programba beépített parancsoknak programozott módon való végrehajtása.

A parancsok és a be- és kimeneti eszközök összerendelése tetszőleges lehet. Pl. ha egy olyan file-hoz, melyen a BASIC fordítóprogram van, hozzárendeljük a sornyomtatót (egy képernyős megjelenítőn adott hozzárendelési paranccsal), a kiírások a nyomtatón jelennek meg. Semmi egyéb változtatásra nincs szükség.

A külső parancsoknak a háttértárról történő leolvasásával kapcsolatban az alábbi terminológiát használjuk: a lemezes operációs rendszer, amelynek alapvető szerepe van a parancsok elérésében, a master eszköz, amit röviden eszköznek nevezünk. A parancshoz tartozó programmodul a mágneslemez-meghajtóegységben található. A mágneslemezen levő egyéb program- és adatblokkokat pedig a továbbiakban összefoglalóan file-nak nevezünk.

A file-ok tárba helyezésekor az áthelyezési eljárással kapcsolatos módszer a következő. A program írásakor a fordítóprogramnak (pl. assemblernek) megadjuk a globális és a külső paramétereket (l. a 9. fejezetet). Fordítás után a bináris file (tárgyprogram) tárbeli helyét a csatolóprogram adja meg. Tehát a csatolóprogramot akti-



86. ÁBRA. ARIO-OS és a külső parancsok hívásának a vázlata

vizáljuk, megadjuk a tárgyprogramfile-t és a tárhelyre vonatkozó adatokat. A csatolóprogram ebből eljárás típusú tárgyprogramot képez, amelyet az OS tölt be a tárba a csatolóprogrammal meghatározott helyre. Az OS a parancsok függvényében (l. később) a betöltést a szabad tárhelyre elvégzi, de adható olyan betöltési parancs is, amely más programmodul által elfoglalt helyre is elvégzi a betöltést. Az OS a programmodul működésének a befejeztével (a parancs végén) felszabadítja a parancs részére lefoglalt területet.

Képernyős megjelenítőről történő működési módban a gépkezelő a képernyős megjelenítő billentyűzetével begépeli a kívánt parancsot (8.4. és 8.5. táblázat). Az OS

8.4. TÁBLÁZAT. A RIO belső parancsai

Parancs	Paraméterei	A parancs leírása
ALLOCATE	Tártartomány – alsó határa – felső határa – blokkméret	Ellenőrzi, hogy a tártartományban van-e megadott blokkméretnyi szabad hely (más program által el nem foglalt hely). A programok áthelyezésének megkönnyítésére szolgáló parancs
DEALLOCATE	Tárblokk – kezdőcíme – mérete	„Felszabadítja" a kezdőcímtől a megadott méretű tartományt
RELEASE		Az OS által elfoglalt terület kivételével az összes tárhelyet felszabadítja
BRIEF		A további parancsok esetén a képernyős megjelenítőre csak rövidebb üzenetek kiírását engedélyezi
VERBOSE		A további parancsok esetén a teljes kiírást engedélyezi a képernyős megjelenítőre
FORCE	Parancsfile neve	Foglaltságtól függetlenül beírja a tárterületre a megadott file-t
INITIALIZE	Eszköz neve	Inicializálja (működésre előkészíti) a megadott eszközt (pl. a DOS-t és a hajlékony mágneslemezes tárat)
XEQ	Opcionálisan megadható kezdőcím	Az utolsóként tárba töltött file (parancs) végrehajtását, vagy kezdőcím esetén a megadott kezdőcímen levő program végrehajtását kezdeményezi. A paranccsal ismételten is végrehajtható a tárban levő program
DEBUG		Kilépés a RIO-ból és áttérés a hibakereső programba. A hibakereső program nem része a RIO-nak

8.5. TÁBLÁZAT. A RIO külső parancsai

Parancs	Paraméterek		A parancsok leírása
	kötelező	opcionális	
ACTIVATE	Eszköz neve	Kezdőcím	Az eszközt (pl. a felhasználó által írt új DOS-t, vagy egy perifériakezelő programot) az aktív eszköztáblázatban aktivizálja
DEACTIVATE	Eszköz neve		Az ACTIVATE-tel ellentétes parancs
CAT	File-név	Típus, meghajtó sorszáma	A file-név részlegesen is megadható. A hajlékony mágneslemezes tárban (a kiválasztott mágneslemezestár-meghajtón) levő összes olyan file-t megkeresi, amelyek neve vagy nevének része megegyezik a megadottal, és azoknak a paramétereit (pontos név, típus, jellemzők, file-hossz stb.) kilistázza
EXTRACT	File-név		Az eljárásfile legfontosabb adatait (filehossz, rekordok hossza, kezdőcím, tárhelyigény) kilistázza
COMPARE	File 1 File 2		A két file adatainak az összehasonlítását végzi el, feltüntetve az eltéréseket
COPY	File 1 File 2	Meghajtók sorszáma	A file 1 nevűt file 2 névvel átmásolja. A file-ok nevében típusuk és a mágneslemezestár-meghajtó sorszáma is szerepelhet
COPY. DISK		Meghajtók sorszáma	A teljes mágneslemez tartalmának átmásolására szolgáló parancs
MOVE	File-név	Típus, meghajtó	File-ok másolására (egyik mágneslemezes tárról a másikra) szolgáló parancs. A file-nevek részlegesen is megadhatók (a CAT parancshoz hasonlóan)
DELETE	File-név	Típus, meghajtó sorszáma	A megadott file-ok törlése. A file-nevek részlegesen is megadhatók (a CAT parancshoz hasonlóan)
DUMP	File-név	Kezdőcím, végcím	A megadott file (bináris is) kilistázása a sornyomtatón. Kezdő- és végcím megadása esetén csak a két cím közé eső részt írja ki a parancs
IMAGE	File-név kezdő- és végcím	Programindulási cím, rekordméret	A tárban megadott tartományban elhelyezkedő programnak a hajlékony mágneslemezes tárba való írása

8.5. TÁBLÁZAT (1. folytatás)

Parancs	Paraméterek		A parancsok leírása
	kötelező	opcionális	
RENAME	File régi neve, file új neve	Mágneslemezes egység meghajtók sorszáma	A megadott file nevének a megváltoztatása
STATUS		Mágneslemezes egység-meghajtók sorszáma	A mágneslemezesegység-meghajtóban levő mágneslemez állapotának (neve, teleírt rekordok és üres rekordok száma) a kiiratása
FORMAT	Mágneslemezes egység meghajtó, mágnestárcsa neve		Ez a parancs a mágneslemezesegység-meghajtóba behelyezett mágneslemez tartalmát formálja (órajelekkel, szektorcímekkel ellátja)
DATE		Dátum	Dátumot kiíró és a mágneslemezre felíró parancs. A mágneslemezeken szereplő dátum a programok aktualitását jelzi
DEFINE	File-név, eszköznév	Logikai egység neve	A file-okhoz eszközt, logikai egységet hozzárendelő parancs. Ezzel változtathatók meg a be-, ill. kimeneti kezelőprogramok egy programrendszerben
MASTER	Eszköz neve		A DOS helyett másik programrendszert (pl. felhasználói mágneslemezes tárat kezelő rendszert) master eszközként megadó parancs
LADT			Az aktív eszközök nevét, kezdőcímüket és a hozzájuk tartozó logikai egységeket kilistázó parancs
DISPLAY			A tárhozzárendelést kijelző parancs
DO	Parancs-file	paraméterek	A parancsokból álló parancsfile végrehajtását kezdeményező parancs.
PAUSE			Végrehajtási szünetet iktat a **DO** parancscsal kezdett parancsok közé. A szünet végét képernyős megjelenítő billentyűzetén leütött karakterrel jelezhetjük
ECHO	Szöveg		A megadott szöveget kiirató parancs

8.5. TÁBLÁZAT (2. folytatás)

Parancs	Paraméterek		A parancsok leírása
	kötelező	opcionális	
SET	Paraméterek		Az OS több kezelési paramétert tartalmaz és file-paramétert vezérel (pl.: a gépkezelő által begépelt szövegben a betűtörlést, sortörlést megadó kód vagy tabulátor állandót, ill. file típust, jellemzőt). Ezek változtathatók meg a SET paranccsal.
EXPRESSION EVALUATION	Algebrai kifejezés		Algebrai kifejezéseket kiszámító parancs. Pl.: a begépelt EE25–3525/40 (hexadecimális) kifejezést kiszámítja, s eredményként kiírja a 2E4-et.

a sor végéig fogad el parancsokat, majd a sorvég észlelésekor rátér azok végrehajtására. Egy sorban több parancs is begépelhető parancselválasztó jelek (;) alkalmazásával. Lehetőség van file tárba töltésére úgy, hogy annak végrehajtását az OS nem kezdeményezi. Ebben az esetben más parancselválasztó jel (,) használatos, amely az előtte levő parancs végrehajtását tiltja le. Pl. felhasználói programok különálló részei parancsokként így tölthetők be a tárba későbbi, együttes végrehajtás céljából.

Programozott módon is végrehajtható bármelyik RIO parancs. Ebben az esetben a parancsot úgy építhetjük be a felhasználói programba, hogy a parancs nevét tárba helyezzük, és ennek a helynek a címét az indexregiszter tartalmával adjuk meg. Ezt követően a felhasználói program egy megadott címmel (SYSTEM-címke az OS-ben) szubrutinhívást hajt végre, s ekkor a RIO a kívánt parancsot betölti a tárba, végrehajtja azt, majd visszatér a felhasználói programba. Ily módon egy vagy több file vagy programmodul láncolható össze. A felhasználói program mentesül attól, hogy gondoskodjék a modulok futás közbeni tárba töltéséről, a mágneslemezes egységek kezeléséről. Ez különösen előnyös olyan nagy felhasználói programok esetén, amelyek számára nincsen elegendő operatívtár-hely, és mágneslemezes egységről kell a programmodulokat végrehajtásra bevinni. A programmodulok ilyen szerkezetét átfedéses struktúrának (overlay structure) hívjuk.

A programba beépítendő parancs (programmodul) nevét és paramétereit tartalmazó szövegrészre való rámutatáshoz a RIO be-, ill. kiviteleknek a szabványos hívási eljárását kell követni. A paramétervektor (l. a 8.1. táblázatot) tartalmát híváskor az alábbiak szerint kell megadni:

- logikai egység sorszáma = 0,
- a parancskódnak nincs jelentése,
- az átviteli cím tartalma jelzi a parancsszöveg kezdőcímét,
- az adatblokkhossznak nincs jelentése, a parancsszöveg végét a sorvégkarakter („kocsi vissza") jelzi,
- a befejezés utáni visszatérési címet nem kell megadni,
- a hibás befejezés címét meg kell adni, ide tér vissza az OS, ha a parancs teljesítése során hiba jelentkezik,
- a befejezési állapotjelző kód jelentése nem változik,
- a kiegészítő információ jelentése a hívott parancsnál előírtak szerint alakul.

A RIO a 0. logikai egységnek fiktív jelentést ad, ez nem perifériaművelethez tartozó egység, hanem a programok egymás közötti kommunikációjának az eszköze.

A továbbiakban példákat mutatunk néhány RIO parancs használatára, és a felhasználói programba való beépítésére:

1. **DEFINE** 3 $PRINTER

A parancs a 3. logikai egységet a PRINTER eszközhöz (file-hoz) rendeli.

DEFINE 7 $FELHDOS/FELHFILE

A parancs a 7. egységet a felhasználói DOS-hoz (FELHDOS) rendeli. A DOS több be-, ill. kimeneti kérésjelet generálhat, több logikai egység is lehet a DOS-hoz rendelve. Jelen esetben a felhasználói DOS-on belül a felhasználói file-hez (FELHFILE) rendeli a parancs a 7. egységet.

2. A **DO** parancs használatát egy parancsfile-lal ismertetjük. A PFILE parancsfile alakja:

PFILE: **COPY** # 1 # 2; **ECHO** MASOLAS KESZ; **PAUSE**; **COMP** # 1 # 2;
ECHO OSSZEHASONLITAS KESZ

A **DO** parancs alakja:

DO PFILE PROGR1 PROGR2,

és hatására a rendszer a # 1 helyén levő PROGR1 file-t a # 2 helyen levő PROGR2 néven átmásolja (**COPY** parancs) egy új file-ba. Ezt követően a képernyős megjelenítőre kiírja a MÁSOLÁS KÉSZ szöveget (**ECHO** parancs), majd szünetet tart (**PAUSE** parancs). A szünet végét a gépkezelő egy betű leütésével jelzi. Ezek után a két file-t ellenőrzésként összehasonlítja (**COMP** parancs), és annak befejeztével kiírja a képernyős megjelenítőre az ÖSSZEHASONLÍTÁS KÉSZ szöveget (**ECHO** parancs).

3. **LADT**

Erre a parancsra válaszként megjelenik:

DEVICE	ADDRESS	SIZE	UNITS
DOS	2500	1600	0 5 6 8 10 12 13 14 15 16 17 18
CONSOLE	4ABC	0200	1 2
PRINTER	3215	0100	3

ahol az aktív eszköz neve a DEVICE oszlopban, a kezdőcíme az ADDRESS oszlopban a programmérete (eszközmérete) a SIZE oszlopban van, míg a hozzárendelt logikai egységeket a UNITS oszlopban írja ki a RIO.

4. **MOVE** LIN* T=P S=1 D=0

Ennek a parancsnak az eredményeképpen az 1. hajlékony mágneslemezről (S: Source: forrás) a 0. hajlékony mágneslemezre (D: Destination: cél) másol a RIO minden P (procedúra) típusú (T: típus) file-t, amelyiknek a neve LIN-nel kezdődik. A * arra utal, hogy a file-név többi része nincs megadva, a * helyett bármilyen betű vagy betűsorozat előfordulhat a file nevében.

5. **XEQ***

az utolsó betöltött file kezdőcímére ugrató, s azt végrehajtató parancs.

XEQ 4700

a 4700 címen kezdődő programmodul végrehajtását kezdeményező parancs.

Több parancs egymás utáni végrehajtására példa:

6. **ASM** PROG;**LINK** $=5000 PROG (E=5050);DATA,PROG

Ez az összevont parancssor behívja az ASM nevű assembler fordítóprogramot, amely lefordítja a PROG nevű felhasználói forrásfile-t (l. a 9. fejezetet). A fordítás befejezésével azonnal indul a csatolóprogram (LINK: linker), amely a PROG nevű áthelyezhető tárgyprogramot az 5000 címtől kezdődően elhelyezi, ez lesz a PROG nevű eljárásfile a mágneslemezen. Az eljárásfile (program) végrehajtási kezdőcíme (E:

entry point) az 5050. A csatolóprogram működését követően a RIO betölti a DATA nevű file-t, de nem indítja el. A file nevét követő vessző az indítást letiltja. (A példa szerint ez a DATA file a PROG file által feldolgozandó adatokat tartalmazza.) Az adatok betöltését követi a PROG file betöltése az 5000 címtől kezdődően, majd a program végrehajtása az 5050 belépési ponttól.

A RIO-nak a felhasználói programba való beépítésére az alábbi programok mutatnak példát. A példákat a Z80 mikroprocesszor utasításkódjaival és emlékeztető szimbólumaival írjuk meg [10], [15].

1. programpélda: szövegkiiratás a programban. A felhasználói program a RIO által definiált képernyős megjelenítő logikai egységet használja. Ennek a példa szerint a DISPLAY szimbólum az azonosítója. Az átviteli parancs kódja a SORKIV szimbólummal megadott érték (fordításkor egy kódszámot jelent).

```
                    :
;A kiíró program hívása:
                LD      IY,PARVEK  ; IY – a paramétervektor mutatója
                CALL    SYSTEM     ; áttérés az OS-be
;A felhasználói program folytatása
                    :
;A paramétervektor:
    PARVEK:     DEFB    DISPLAY    ; a képernyős megjelenítő (logikai
                                   ; egység) száma: DISPLAY szimbólum
                DEFB    SORKIV     ; sorkiíratási parancskód: SORKIV
                DEFW    SZOVEG     ; a kiíratandó szöveg kezdőcíme két byte
                DEFW    SORHOSZ    ; a szöveg karaktereinek száma két byte
                DEFW    0          ; visszatérési cím, nem kell megadni. A pa-
                                   ; rancskód a hívás utáni helyre definiálja a
                                   ; visszatérést
                DEFW    HIBA       ; hibás művelet utáni visszatérési cím
                DEFB    0          ; visszatérési kód, a RIO írja be
;A kiíratandó szöveg:
    SZOVEG:     DEFM    'MUVELET VEGE'
                DEFB    0DH        ; kocsi vissza kódja
    SORHOSZ:    EQU     $-SZOVEG   ; a karakterek száma
                    :
```

2. programpélda: RIO parancsok hívása a tárban elhelyezett parancsszóval, a 0. logikai egységen (fiktív egység) keresztül. A hívott parancsok a **COPY** modul, majd a MÁSOLÁS KÉSZ szöveg kiírásához az **ECHO** parancs:

```
                    :
;A parancs hívása:
                LD      IY,PARVEK  ; IY (a paramétervektor mutatója)
                CALL    SYSTEM     ; áttérés a RIO-ra
;A felhasználói program folytatása
                    :
```

```
;A kommunikációs paramétervektor parancshíváshoz:
    PARVEK: DEFB  0          ; 0 jelzi, hogy nem be-, kimeneti eszköz, hanem
                             ; parancshívás
            DEFB  0          ; közömbös értékű rekesz
            DEFW  PARNEV     ; a parancsszöveg címe (két byte)
            DEFW  0          ; nem használt két byte
            DEFW  0          ; nem használt két byte
            DEFW  HIBA       ; hibás művelet utáni visszatérési cím
            DEFB  0          ; a visszatérési kód
;A parancs leírása:
    PARNEV: DEFM 'COPY PROG1 PROG2; ECHO MASOLAS KESZ'
            DEFB  0DH        ; kocsi vissza kódja
```

IRODALOM

[1] *Payne, J. L.:* A software development workshop for programmable microelectronics. The Radio and Electronic Engineer, March 1978. 122–126.

[2] *Donovan, J. J.:* Systems programming. McGraw-Hill Book Company, New York, 1972.

[3] *Wester, J. G.:* Software design for microprocessors. Texas Instrument Learning Center, 1976.

[4] *Isaak, J.:* Microcomputer software satisfies conflicting programming needs. Electronics, Dec. 20, 1979. 104–107.

[5] ZDS 1/25, 1/40. PROM user's manual. Zilog Inc., 1977.

[6] *Varga L.:* Rendszerprogramok elmélete és gyakorlata. Akadémiai Kiadó, Budapest, 1978.

[7] Motorola 68000. Motorola Inc., 1979.

[8] Zilog Z–8000. An architectural overview. Zilog Inc., 1978.

[9] Zilog Z–8001 CPU, Z–8002 CPU. Product specification. Zilog Inc., 1979.

[10] Z–80 RIO relocating assembler and linker. User's manual. Zilog Inc., 1978.

[11] *Townzen, D. A.:* A task scheduling executive program for microcomputer systems. Computer Design, June 1977. 194–202.

[12] *Burgett, K.–O'Neil, E. F.:* An integral real-time executive for microcomputers. Computer Design, July 1977. 77–82.

[13] Z–80 RIO operating system. User's manual. Zilog Inc., 1977.

[14] Z–80 RIO Text editor. User's manual. Zilog Inc., 1977.

[15] Z–80 microcomputer system. Technical manual. Mostek Inc., 1977.

9. PROGRAMFEJLESZTÉS ASSEMBLER NYELVEN

A mikroprocesszorok működését a számítógépekhez hasonlóan programok vezérlik. A programok írásához és előkészítéséhez különböző programnyelveket használunk [1]. Az egyik legalacsonyabb szintű programnyelv az assembler nyelv. Ez az utasításkészlettel való szoros kapcsolata miatt mikroszámítógép- vagy mikroprocesszorfüggő. Hasonlóan típusfüggőek az ezen a nyelven megírt felhasználói programok is.

E fejezetben röviden áttekintjük a programíráshoz általánosan használt eljárásokat, majd az assembler nyelvű programírásra adunk néhány példát.

9.1. A PROGRAMÍRÁS ELŐKÉSZÍTÉSE

A programok kidolgozásának a célja az, hogy meghatározzuk az utasítások megfelelő sorozatát, majd azt végrehajtás céljából bevigyük a mikroszámítógép tárába. A megfelelő utasítássorozatot végrehajtva a mikroszámítógép a programmal megvalósítandó feladatot és a hozzá szükséges funkciókat látja el.

Bonyolult szoftvervezérelt feladatok kidolgozását a feladat részekre bontásával kezdjük. Az egyik leggyakrabban használt célravezető módszer a feladat, majd a részfeladatok fokozatos bontása. Először a feladatot megvalósító funkciók összességét állapítjuk meg. Ez a fázis a szoftver-rendszerspecifikáció. Majd a feladatot néhány részre: programmodulokra bontjuk. A programmodulokat további részekre osztjuk, és így tovább. A lebontás végcélját a program elemi lépéseinek, az utasításoknak a meghatározásáig folytatjuk. A programrészek elnevezésére – feladatoktól és alkalmazásoktól függően – a programrendszerek, az alrendszerek, a programmodulok, a programblokkok, az alprogramok, a szubrutinok fogalmát használjuk.

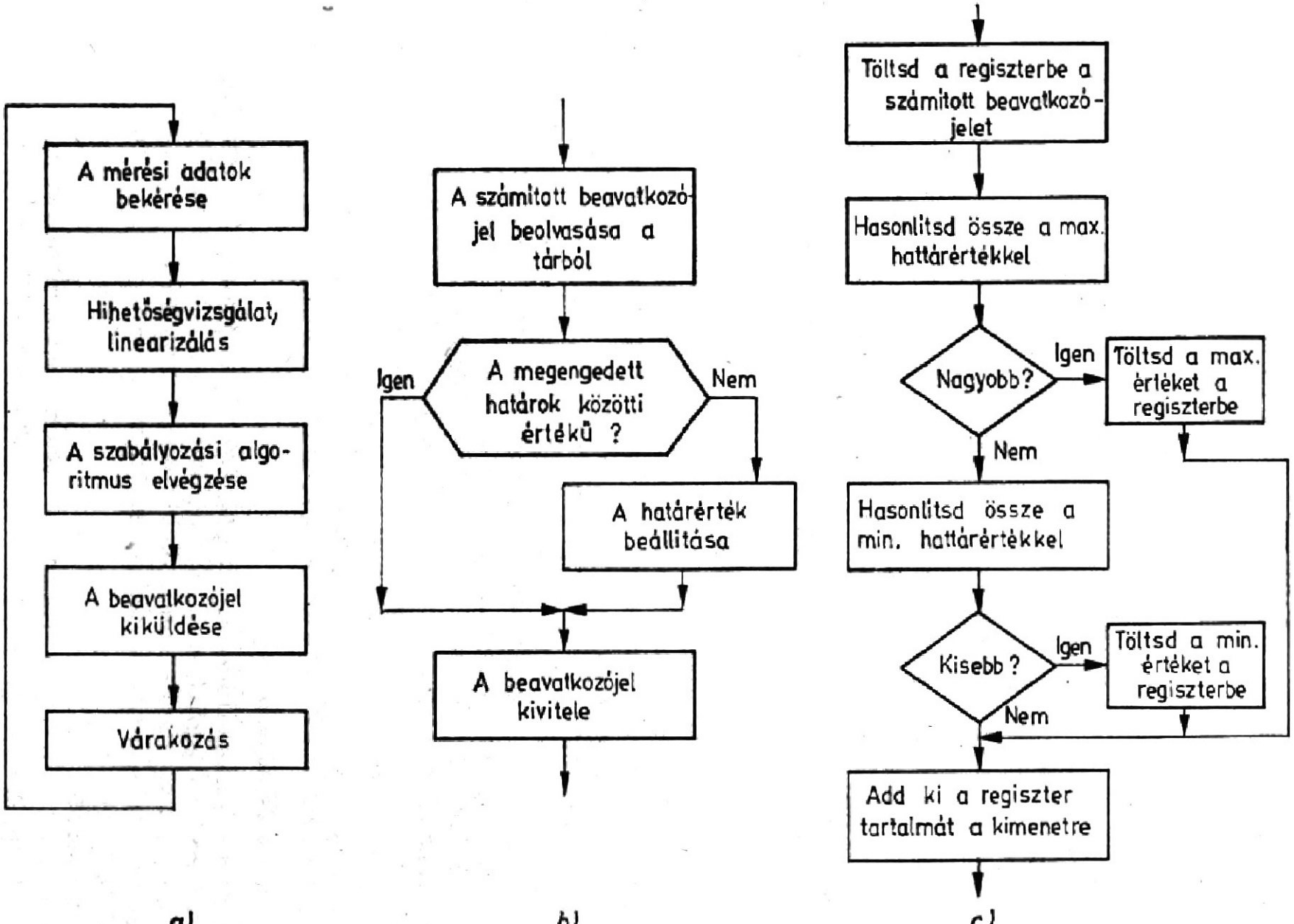
Algoritmusok és folyamatábrák

A feladatot leíró algoritmus lépéseinek, a részfeladatoknak a sorrendjét és az információk áramlását grafikusan a folyamatábrák (flow chart) mutatják [2]. A folyamatábrák készítésének három szintjét különböztetjük meg [3]:

1. feladatmegfogalmazási szint (koncepciószint),
2. algoritmusszint,
3. utasításszint.

Az első két szint folyamatábráinak az elkészítéséhez általában még nem szükséges a mikroszámítógép típusának és utasításkészletének az ismerete. A feladatmegfogalmazás szintje átfogó állításokat tartalmaz arra vonatkoztatva, hogy mit kell tenni. Az algoritmusszint már részleteiben is tükrözi, hogy a teendőket hogyan kell megvalósítani. Végül az utasításszintű folyamatábrák már figyelembe veszik a mikroszámítógép felépítését, regisztereit, utasításait, és mintegy térképként használhatók a szoftver kódolásához, az utasítások megírásához.

A 9.1. ábrán a folyamatirányítás területéről vett feladat folyamatábráira mutatunk be példát. A 9.1*a* ábrán a feladatmegfogalmazási szint folyamatábrája látható.



9.1. ÁBRA. Példa a különböző szintű folyamatábrákra *a)* feladatmegfogalmazás módja, *b)* algoritmusszint, *c)* utasításszint

A 9.1*b* ábra az előző folyamatábra egy részletét, a „beavatkozójel kiküldése" művelet algoritmus szintű folyamatát mutatja. A példában olyan beavatkozójelet tételeztünk fel, amelyet mindkét irányban korlátozni kell. A megengedhető maximális, ill. minimális értéket túllépő beavatkozójel ui. hibás működést okozna. Ugyanezt a részletet mutatja a 9.1*c* ábrán feltüntetett utasítás szintű folyamatábra is. A folyamatábra készítésekor Intel 8080A típusú mikroprocesszort tételeztünk fel. Látható, hogy még az utasítás szintű folyamatábra sem tükröz minden utasítást ábraszimbólummal, egy-egy szimbólum több utasítást is jelenthet.

A folyamatábrák nemcsak a programok kidolgozásának hasznos eszközei, hanem a programok ellenőrzése, kipróbálása, esetleges módosítása és felhasználása során dokumentációként is használhatók. Nem minden feladat és nem minden programozó igényli a folyamatábrák három felsorolt szintjének a kidolgozását. Egyszerűbb feladatokhoz és gyakorlott programozók számára kevésbé szükséges a folyamatábrák többszintes bontása, bonyolultabb programok megírásakor azonban komoly segítséget nyújtanak a helyesen, körültekintően folyamatábrákkal támogatott előkészítési munkák.

Szerkesztőprogramok

A folyamatábrák alapján készül a forrásprogram. Az assembler nyelvű forrásprogram általában a mikroszámítógép által ismert utasításokból áll. Az utasítássorokat az angol irodalomban statementnek nevezik. A folyamatábráknak a programrészekké való átalakítására gyakran használják a programkódolás kifejezést. A programokban megadandó utasítássorok tartalmát és formáját a 9.3. szakaszban ismertetjük. Most a forrásprogram adathordozóra vitelének, vagyis az assembler program számára való előkészítésének a folyamatát mutatjuk be.

A forrásprogram betűkből, számokból, írásjelekből álló „szöveges" információ. A kódolt és papírra leírt programot írógéppel vagy egyéb alfanumerikus billentyűzetű eszközzel legépelhetjük. Gépeléskor a formátumra vonatkozó szabályok figyelembevételével a program utasításait hibátlanul kell az adathordozóra felvinni. Az esetleges gépelési hibák kijavításához, a kívánt formátum eléréséhez a számítógépek is segítséget adhatnak. Azt a számítógépes programot, amelyik a forrásprogram begépelését (adathordozóra rögzítését) segíti, szövegszerkesztő programnak (text editor), vagy röviden szerkesztőprogramnak (editor) nevezzük. A forrásprogram adathordozójaként a leggyakrabban lyukszalagot, kazettás mágnesszalagot vagy hajlékony mágneslemezt használunk. A szerkesztőprogram nemcsak a gépelés során használható, hanem segítséget nyújt a már korábban leírt tárgyprogramok kijavításához, módosításához, új, javított tárgyprogram készítéséhez is.

A szerkesztőprogram adatot kaphat:

- írógépről vagy képernyős megjelenítő billentyűzetéről, amely az ember–gép közötti kapcsolattartás eszköze,
- javításkor a régi forrásprogramból (lyukszalagolvasóról, kazettás mágnesszalagos vagy mágneslemezes egységről).

A szerkesztőprogram az adatokat a következőkkel rögzíti:

- az írógép nyomtatószerkezete vagy a képernyős megjelenítő képernyője, amelyen az éppen feldolgozás alatt levő szövegrészen végzendő műveletek kijelzése látható,
- forrásprogram-file az adathordozón.

A program gépelése és javítása munkaigényes folyamat. Egyes becslések szerint a mikroprocesszoros programok kidolgozói munkaidejükből közel egyharmadnyi részben a szerkesztőprogrammal dolgoznak [7]. Nem mindegy, hogy ezt az időt mennyire hatékonyan tölthetjük ki. A hatékonyságot lényegesen befolyásoló ténye-

zők közé sorolhatjuk a szerkesztőprogramok kényelmes használatát, jól átgondolt és könnyen kezelhető parancskészletét, és nem utolsósorban azokat a perifériákat, amelyek a forrásprogram rögzítését végzik.

A szerkesztőprogram két alapüzemmódban dolgozik:

- szövegbeviteli üzemmód,
- parancs üzemmód.

Szövegbeviteli üzemmódban a billentyűzetről bekerülő szöveget a szerkesztőprogram a forrásprogram utasításaiként helyezi előbb egy munkatárterületre, majd a kezelő parancsára – vagy a munkaterület teljes feöltöltése után automatikusan – kiküldi az adatrögzítést végző perifériának. A szerkesztőprogramok általában soronként kezelik a begépelt információt. Az aktuális sor kezdetét, a sor első betűjének a címét egy sormutató (pointer) rekesz tartalma jelzi. Kijelölt vezérlőbillentyűkkel az utoljára begépelt sor betűi (utolsótól visszafelé) és a teljes sor is törölhető.

Parancs üzemmódban a szerkesztőprogramot vezérlő parancsok billentyűről való megadására nyílik lehetőség. A két üzemmód közötti átváltást az erre kijelölt vezérlőkarakterek bevitele eredményezi. Általában nyomtatott szövegben nem megjelenő kódok a vezérlőkarakterek (pl. az ESC, NAK, ALT MODE ASCII-vezérlő kódok).

Parancs üzemmódban megadható legfontosabb parancsok [8]:

Bemeneti parancsok

- a munkatár feltöltése a forrásfile-ból (ha a forrásfile javítására van szükség),
- szövegmódban sorok bevitele a billentyűzetről,
- egy másik forrásfile hozzátoldása a munkatárban levő programrészhez.

Kimeneti parancsok

- a forrásprogramnak a kimeneti perifériára való kiküldése, adatrögzítés (lyukszalagkészítés, mágnesszalagon vagy mágneslemezen való file-készítés),
- a munkatár tartalmának (egy vagy több sornak) az írógépen vagy a képernyős megjelenítőn való megjelenítése.

Sormutató manipulációs parancsok

- a mutatót a munkatár elejét vagy végét jelző értékre állító parancsok,
- a mutató tartalmának *n* sorral való előre- vagy visszafelé mozgatása.

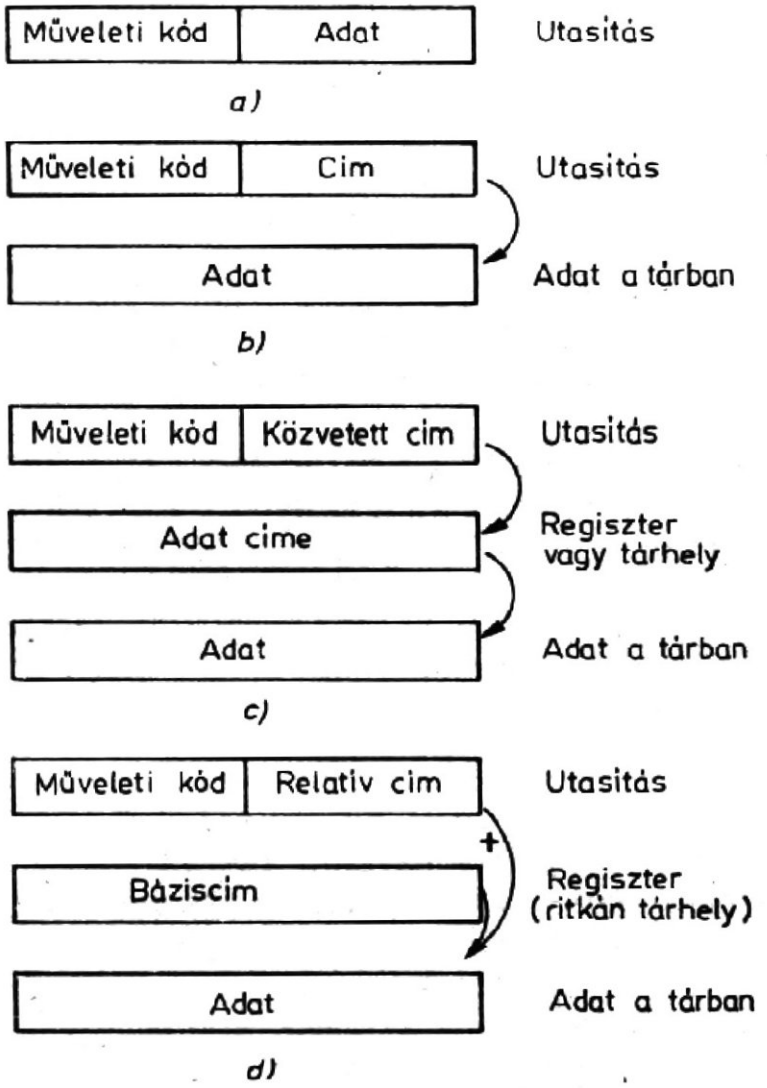
Adatmanipulációs parancsok

- *n* számú sor törlése,
- a sor tartalmának a megváltoztatása: betűcsoportok törlése, hozzáírása, kicserélése,
- új sorok beiktatása,
- megadott szövegrész (betűcsoport) megkeresése, és az előfordulásához tartozó sor kijelzése.

9.2. AZ ASSEMBLEREK ALAPFUNKCIÓI

A mikroprocesszorok utasításkészlete általában nagyszámú különböző adatátviteli, aritmetikai és logikai, feltételes és feltétel nélküli elágazási, valamint vezérlési utasítást tartalmaz. A különböző utasítások hossza egy mikroprocesszoron belül is különböző lehet. Ha megvizsgáljuk a programokat, a programban levő és a programhoz kapcsolódó információ típusait az alábbiak szerint csoportosíthatjuk:

- utasításkódok,
- tár-, regiszter- és bemeneti vagy kimeneti címek,
- adatok.



9.2. ÁBRA. Az alap címzési módok vázlata
a) **közvetlenül csatolt,** *b)* **közvetlen,** *c)* **közvetett,** *d)* **relatív**

Az utasításokban az utasításkódok mellett címek és adatok is lehetnek. Ezeket a továbbiakban közös névvel operandusoknak hívjuk. Az utasításokban előforduló operandusokkal kapcsolatban beszélhetünk címzési módokról. A címzési módok az utasítással érintett adatok elérésének a módját jelentik. A mikroprocesszorok esetén a címzési módok sokasága terjedt el. A négy alap címzési mód a következő (9.2. ábra):

- közvetlenül csatolt (immediate),
- közvetlen (direct),
- közvetett (indirect),
- relatív.

Közvetlenül csatolt címzési mód esetén az adat az utasításban található, az „aktuális" adat az utasításkódot követi, függetlenül attól, hogy az utasítás a tár melyik részén található. **Közvetlen címzési mód** esetén az utasításkódot az adat címe követi. Tehát az utasításban nem maga az adat, hanem az a cím (tár-, regiszter-, vagy be-, ill. kimeneti cím) található, amelyik az adat helyére mutat rá. **Közvetett címzés** esetén az utasításban annak a regiszternek (ritkábban tárhelynek) a címe található, ahol az adat címe helyezkedik el. Az adat eléréséhez tehát nem elegendő az utasítás operandusrésze, hanem annak alapján kell megkeresni az adat címét, majd az adat címének az ismeretében már megtalálható az adat is. **Relatív címzési mód** esetén az utasítás címrésze (amely általános esetben lehet közvetlen vagy közvetett cím is) nem tartalmazza a teljes címet, csak a cím valamely báziscímhez képesti elté-

rését. Ez a relatív cím. A teljes cím a relatív cím és a báziscím alapján található meg (rendszerint a kettő összege). Báziscím leggyakrabban az utasítás tárbeli címe (a programszámláló tartalma) vagy valamelyik regiszter tartalma.

A címzési módok tovább kombinálhatók aszerint, hogy az utasítással érintett cím tárra, regiszterre vagy be-, ill. kimeneti egységre vonatkozik, ill., hogy a közvetett és a báziscím hol helyezkedik el. Egyes esetekben a közvetett címnek a tartalma egy-egy utasítás után meg is változik (növekszik vagy csökken). A címzési módok újabb alcsoportját jelenthetik a *bitcímzési eljárások*. Egyes mikroprocesszor-típusoknak olyan utasításai is vannak, amelyek nem byte-okat, párhuzamos szavakat vizsgálnak és módosítanak, hanem byte-okon belül vagy szavakon belül is képesek egy-egy bitet elérni.

A címzési módot az utasításkód határozza meg. Az utasításkód tehát nemcsak az adatátviteli, aritmetikai, logikai stb. műveletet határozza meg, hanem a címzési módot is kijelöli. Egyes mikroszámítógép-típusok esetén az utasítás műveleti kódjának bitjei elkülönülnek a címzési mód bitjeitől, mások esetén nem. Ez utóbbi azt jelenti, hogy a kód értéke együtt jelöli ki a műveletet és a címzési módot is. Gyakori, hogy egy utasítás két címet is kezel, egyiket kódolt, implicit formában, a másikat az előbbiekben felsorolt címzési módok egyikével. Ilyen utasítások pl. az aritmetikai és logikai utasítások, amelyeknek egyik operandusát (az akkumulátorban levőt) az utasításkód jelöli ki.

Az adatok és az utasítások a tárban helyezkednek el. A tárban elhelyezkedő információt két paramétere jellemzi: a tartalma (az adat vagy az utasításkód számszerű értéke) és az elhelyezkedési címe. A tárgyprogram elkészítéséhez mindkettőre szükség van. Az utasítások kódrészét és címzési módját az emlékeztető szimbólumok adják meg, amelyeket a programozó könnyen memorizálható alakjuknak köszönhetően viszonylag jól kezel. Az adatok lehetnek konstansok vagy változók. A konstansokat és az egyes kiindulási változókat a programozó adja meg. A többi változót vagy a bemeneti egységekről kéri be a program, vagy pedig működés közben maga állítja elő (a program írásakor tehát nem kell meghatározni őket). A be- és a kimeneti műveletekhez tartozó címeket a programozó adja meg a hardver ismeretében. Általában nem okoz nagyobb nehézséget ezeknek a konstansoknak, kiindulási adatoknak és a be-, ill. kimeneti címeknek a megadása, hiszen a legtöbb esetben a számuk nem nagy.

A tárcímek, az utasítások és a tárban levő adatok elhelyezkedésének a meghatározása már nehezebb feladat. Szerencsére ezeknek a címeknek csak egy részét – általában csak néhányat – kell megadni számszerű formában, a többit már a fordítóprogram: az assembler kezeli. A számértékükkel megadandó adatok és címek deklarálásához az utasítások emlékeztető szimbólumaihoz hasonló ún. pszeudoutasítások (direktívák) tartoznak.

A fordítóprogram által kezelt utasítás- és adatcímek között vannak olyanok, amelyeket ugyan nem számértékükkel, de azonosítanunk kell. Az azonosításra azért van szükség, mert a program sok helyen hivatkozhat rájuk. Pl. ha a programozó nem adja meg számszerűen a mérési adat tárbeli helyét, azt a fordítóprogram a fordítás során jelöli ki. Később a tárgyprogram futásakor erre a kijelölt helyre kerülnek a mérési adatok. A megfelelő utasításban csak úgy lehet azonosítani az adatot, ha a programozó már a forrásprogramban jelzi, hogy melyik utasításhoz melyik adat címe tartozik. A jelzés a címazonosítóval, a programozó által megadott szimbólummal végezhető. A címazonosító a forrásprogramban általában két helyen fordul elő: operandusként a megfelelő utasításban és az adat mellett. Az adat vagy az utasítás melletti helyen a címkemezőben deklaráljuk a címazonosítót vagy címkét, azaz jelezzük az assemblernek, hogy ezzel a szimbólummal azonos az adat címe. Ha

az utasításban a címazonosító az operandusmezőben van, akkor az utasítás a címazonosítóval jelölt adatokra hivatkozik. Természetesen címkét nemcsak adatok, hanem utasítások mellett is deklarálhatunk, vagyis az utasítás címével azonos értéket rendelhetünk hozzá. A cím- és adatazonosítót peszudoutasításokkal is dekradálhatjuk, erre a pszeudoutasítások tárgyalásakor visszatérünk.

A program írásakor egy sor az alábbi részekből állhat:

Címke *Műveleti kód* *Operandus*

A műveleti kódon a megfelelő emlékeztető szimbólumot értjük. Az operandusban szimbólumok és számok lehetnek. A programsor egy további negyedik résszel is kiegészíthető. Ez a megjegyzés (comment). A megjegyzést az assembler figyelmen kívül hagyja. A megjegyzésnek csak a programozó és a programot használó ember számára van jelentése. Általában szöveges információt tartalmaz, és a program dokumentálására, könnyebb megértésére alkalmas. A további példákban mi is ilyen célra használjuk.

A következőkben az Intel 8080A mikroprocesszor utasításkészletének és a [4]-ben ismertetett emlékeztető szimbólumoknak a felhasználásával a 9.1*c* ábra folyamatdiagramjához tartozó programrészletet mutatjuk be.

```
;RAM-helyek definiálása
BEAVJ:    DB   0          ; a beavatkozójel helyének a megadása. Az
                          ; értéket az előző programrész tölti be. A DB
                          ; pszeudoutasítás
BEAVMX:   DB   200        ; a max. határérték megadása, értéke 200
BEAVMN:   DB   40         ; a min. határérték megadása, értéke 40
          :
;A határértékek túllépését vizsgáló programrész
LIMEX:    LDA  BEAVJ      ; a beavatkozójel A regiszterbe betöltése
          LXI  H,BEAVMX   ; a max. határérték megcímzése
          CMP  M          ; a felső határértékkel való összehasonlítás. Ha
                          ; a beavatkozójel nagyobb vagy egyenlő, az
                          ; átvitelbit értéke 0 lesz
          JNC  MAXLD      ; ha az átvitelbit 0, ugorj max. érték töltésére,
                          ; ellenkező esetben itt folytasd
          INX  H          ; a min. határérték megcímzése
          CMP  M          ; az alsó határértékkel való összehasonlítás. Ha
                          ; a beavatkozójel kisebb, az átvitelbit értéke
                          ; 1 lesz
          JC   MINLD      ; ha az átvitelbit 1, ugorj min. érték töltésére,
                          ; ellenkező esetben itt folytasd
          JMP  KIVIT      ; ugorj a beavatkozójel kivitelére
MAXLD:    MOV  A,M        ; töltsd A regiszterbe a max. értéket, HL regisz-
                          ; terpár címzi meg
KIVIT:    OUT  BJKI       ; beavatkozójel kivitele a BJKI címre
          :
```

```
;A MINLD programrész megegyezik MAXLD-dal, tekintve, hogy itt már HL a
;min értéket címezi meg. Ezért pszeudoutasítással jelezhetjük, hogy a két címke
;értéke egyenlő: MINLD = MAXLD
    MINLD: EQU MAXLD        ; EQU (értéket adó pszeudoutasítás)
;A BJKI kiviteli cím megadása:
    BJKI:  EQU 18           ; példakénti értéke 18.
           :
```

A példában bemutatott assembler nyelvű programrész nemcsak a különböző műveletekhez használ eltérő emlékeztető szimbólumokat, hanem ugyanahhoz a típusú művelethez is, ha eltérő a címzési mód. Erre példa az **LDA** és a **MOV A,M** szimbólum, de az **LXI** is. Az előző kettő operandusa 8-bites. Az utasítások az A regiszter töltését eredményezik közvetlen, ill. regiszter-közvetett címzéssel. A harmadik utasítás a 16 bites HL regiszterpár közvetlenül csatolt címzésű töltésére vonatkozik.

Ismeretes, hogy az Intel 8080A mikroprocesszor gépi kódjával megadott programot a Zilog Z80 mikroprocesszor is végrehajtja (fordítva már nem igaz, mert a Z80 utasításkészlete bővebb, az Intel 8080A számára ismeretlen utasításokat is tartalmaz). A fenti programpéldát tehát közvetlenül utasításról utasításra haladva átültethetjük Z80 utasításszimbólumokra [5], [6]:

```
    Címke     Műveleti kód  Operandus          Megjegyzés
;RAM-helyek definiálása, megegyezik az Intel 8080A-ra írt programban látottakkal
    BEAVJ:    DEFB 0                  ; a DEFB pszeudoutasítás azonos DB-vel
    BEAVMX:   DEFB 200
    BEAVMN:   DEFB 40
              :
;A határértékek túllépését vizsgáló programrész
    LIMEX:    LD   A,(BEAVJ)    ; a beavatkozójel A regiszterbe töltése
              LD   HL,BEAVMX    ; a max határérték megcímzése
              CP   (HL)         ; a felső határértékkel való összehasonlítás
              JP   NC,MAXLD     ; ha túllépi a határértéket, ugorj a max érték
                                ; töltésére
              INC  HL           ; a min határérték megcímzése
              CP   (HL)         ; az alsó határértékkel való összehasonlítás
              JP   C,MINLD      ; ha túllépi a határértéket, ugorj a min érték
                                ; töltésére
              JP   KIVIT        ; ugorj a beavatkozójel kivitelére
    MAXLD:    LD   A,(HL)       ; töltsd A regiszterbe a HL-lel megcímzett
                                ; értéket
    KIVIT:    OUT  (BJKI),A     ; a beavatkozójel kivitele a BJKI címre
              :
;A MAXLD és a MINLD a példa szerint megegyezik
    MINLD:    EQU  MAXLD
;A BJKI cím értékadása
    BJKI:     EQU  18
              :
```

Az utasításszimbólumok ugyan eltérőek a [4]-ben leírtaktól, fordítás után a gépi kód mégis megegyezik. Ez egyben azt is mutatja, hogy az emlékeztető szimbólumok készlete nemcsak a mikroprocesszor típusától és annak utasításkészletétől, hanem a fordítóprogramtól is függ. Az [5], [6] szerinti emlékeztető szimbólumok egy-

ségesebbek. Pl.: a korábban említett három regisztertöltési műveletet azonos szimbólum – az **LD** – jelöli. A megkülönböztetést az operandusmező módosítói végzik. Ezért az assembler a gépi utasításkód meghatározásához nemcsak az emlékeztető szimbólumot, hanem a címzési módra utaló operandusmódosítót (modifier) is figyelembe veszi. A (címke) vagy a (szám) közvetlen címzést jelöl, azaz azt jelzi, hogy a zárójelek közötti érték adja meg a címet. Pl.: **LD A, (BEAVJ)** utasítás. Ha a zárójelben regiszter neve vagy regiszterpár neve szerepel – pl.: **LD A, (HL)** –, ez regiszter-közvetett címzést jelöl, vagyis azt, hogy a **HL** tartalma adja a címet. A zárójel nélküli címke vagy szám – pl.: **LD HL, BEAVMX** – a közvetlenül csatolt címzésnek a jele.

Mind az Intel 8080A-ra írt, mind pedig a Z80-ra írt példákban látjuk, hogy az operandusok között regiszterek nevei is előfordulnak. A regiszterek és a regiszterpárok nevei az assemblerbe beépített ún. rögzített szimbólumok. Ezeket a szimbólumokat a címkemezőben nem használhatjuk, nem deklarálhatunk számukra a rögzítettől eltérő értéket. Az utóbbi példában a regiszterek és regiszterpárok operandusként szerepeltetése is módosíthatja a gépi utasításkód értékét, hiszen az **LD A, ...** pl. 8-bites adattöltést jelöl, mivel az A 8-bites regiszter, míg az **LD HL, ...** 16 bites adattöltésre vonatkozik, mivel a HL regiszterpár 16 bit hosszú.

Eddig az assembler nyelv alaptulajdonságait tárgyaltuk. Az assemblerek a fentieken kívül egyéb segítséget is nyújtanak a programok kidolgozásához. A fordítás során az assembler a tárgyprogramon kívül a programról kérésre ún. listát (listing) is készít. A listával a program dokumentálható. A listán szerepelnek az előbb közölt sorrendben a forrásprogram összes adatai, valamint a tárgyprogram adatai: az utasítások címei és adatok címei, valamint a címeken található értékek (műveleti kód, adatok, operandusok értékei). A legtöbb assembler a forrásprogram sorait sorszámmal is ellátja. A sorszám a programok további szerkesztéséhez, javításához nyújt segítséget.

Az előbbiekben bemutatott Z80-as program dokumentált alakja (a megjegyzés rovat rövidítésével) a következő:

Tárcím	*Tárgykód*	*Sorszám*	*Címke*	*Műveleti kód*	*Operandus*	*Megjegyzés*
		127	;RAM-helyek definiálása			
1611	00	128	BEAVJ:	**DEFB**	0	
1612	C8	129	BEAVMX:	**DEFB**	200	
1613	28	130	BEAVMN:	**DEFB**	40	
			:			

```
                                    ;A határértékek túllépését vizsgáló programrész
1880  3A 11 16   187   LIMEX:  LD    A,(BEAVJ); a beavatkozójel
                                    ;A-ba
1883  21 12 16   188          LD    HL, BEAVMX; az A max
                                    ;határérték címe
1886  BE         189          CP    (HL)      ;az A határér-
                                    ;tékkel
1887  D2 92 18   190          JP    NC,MAXLD; ha túllépi,
                                    ;ugorj
188A  23         191          INC   HL        ;az A min ha-
                                    ;tárérték címe
188B  BE         192          CP    (HL)      ;az A határér-
                                    ;tékkel
188C  DA 92 18   193          JP    C,MINLD   ;ha túllépi,
                                    ;ugorj
188F  C3 93 18   194          JP    KIVIT     ;ugorj a be-
                                    ;avatkozójel
                                    ;kimenetére
1892  7E         195   MAXLD: LD    A,(HL)    ;töltsd A-regisz-
                                    ;terbe
1893  D3 18      196   KIVIT: OUT   (BJKI),A  ;a beavatkozó-
                       :                      ;jel kivitele
                   234   ;a MAXLD és a MINLD...
                   235   MINLD: EQU   MAXLD   ;tárcím és -tar-
                                    ;talom nem tar-
                                    ;tozik hozzá
                   236   ;a BJKI cím értékadása
                   237   BJKI:  EQU   18      ;tárcím és -tar-
                       :                      ;talom nem tar-
                                    ;tozik hozzá
```

Az utasítások címei a változó utasításhossznak megfelelően növekszenek. A tárgykód tartalmazza a műveleti kódot és az operandus értékét. Kétbyte-os operandus esetén előbb a kis helyiértékű byte két számjegye, majd a nagy helyiértékű byte következik (számok írásmódjához képest fordítva, de a tárhelyekre való elhelyezési sorrendnek megfelelően).

A listákon túlmenően a dokumentálást segíti elő a kereszthivatkozási (cross reference) táblázat elkészítése is. Ebbe a táblázatba az assembler kigyűjti és betűrendbe szedi az összes szimbólumot, majd közli az értékét (a definiálási cím- vagy adatértéket), és a forrásprogram azon sorainak a sorszámát, ahol a szimbólum megjelent.

Szimbólum	*Értéke*	*Deklarálás*	*Hivatkozások sorszámai*
BEAVJ	1611	128	187
BEAVMN	1613	130	
BEAVMX	1612	129	188
BJKI	0018	237	196
KIVIT	1893	196	194
LIMEX	1880	187	
MAXLD	1892	195	190 235
MINLD	1892	235	193

A programozó a program írása közben az assembler nyelv „nyelvtani szabályaiba" ütköző hibákat követhet el. Ezeket az assembler észreveszi, és hibaüzenettel jelzi a hibás sort. A hibaüzenet alakja leggyakrabban szöveges (többnyire angol nyelvű) jelzés, amely a hiba típusára is utal. A tipikus nyelvtani (szintaktikai) hibák:

- nem létező emlékeztető szimbólum a műveleti kód helyén,
- operandusok helyén előforduló hiba (pl.: olyan címzési mód, amelyik az adott műveleti kóddal együtt nincs a mikroszámítógép utasításkészletében),
- nem deklarált szimbólumok használata az operandusmezőben,
- több helyen is deklarált (többszörösen és ellentmondásosan használt) szimbólum.

Az operandusok értékét gyakran különböző aritmetikai műveletek adják. Pl., ha programelágazási címként egy olyan helyre kívánunk ugrani, amelyhez nem rendeltünk címkét, de egy tőle nem távol álló utasításnak van címkéje, az elágazási utasítás operandusaként megadható ennek a címkének, valamint a címnek a címke értékétől való eltérésnek (pozitív vagy negatív irányban) az összege. Az assemblerek lehetővé teszik, hogy az operandusmezőben levő paraméterek (számok és szimbólumok) között aritmetikai és logikai műveletekkel kifejezéseket hozzunk létre. Ezeknek a kifejezéseknek az értékét a fordítóprogram kiszámítja. (Pl.: KIVIT szimbólum helyett a 194 sorszámú sor operandusában használhattuk volna a MAXLD+1 kifejezést is.) A kifejezésekben megengedett aritmetikai és logikai műveletek választéka assemblerenként eltérő lehet. Egyes fordítóprogramok a négy airtmetikai műveleten túlmenően hatványozási, abszolutérték-képzési, modulóképzési és relációs aritmetikai műveleteket, valamint logikai összeadási, szorzási, negálási műveleteket is engedélyeznek. A szimbólumok és adatok szám szerinti megadása esetén a számértékeket általában több számábrázolással gépelheti be a programozó (pl.: bináris, oktális, decimális és hexadecimális számokkal).

Az assemblerekkel kapcsolatban beszélhetünk ún. menetekről. Menet alatt a forrásprogram egyszeri beolvasását értjük. Az assemblerek első menetben (first pass) a forrásprogram címkéit gyűjtik össze és állapítják meg a szimbólumok értékét. A második menet (second pass) során történik meg a tárgyprogram és a programlista generálása (a szimbólumok számértékeinek a felhasználásával). A harmadik menet (third pass) általában a kereszthivatkozási táblázat elkészítéséhez szükséges. Hajlékony mágneslemezes tárban tárolt forrás- és tárgyprogramok esetén a felhasználó számára nincs különösebb jelentősége az egyes meneteknek, hiszen a forrásprogram ismételt olvasását a hajlékony mágneslemezes tárról automatikusan elvégezheti a fordítóprogram. Lyukszalagos perifériákon történő programbevitel esetén azonban az assembler minden menet után leáll. Ekkor a felhasználónak a lyukszalagot ismét a forrásprogram elejére kell „visszahúznia", s csak ezután adhat jelet a következő menet elindítására.

A programíráshoz az utasítások és operandusaik megadásán kívül még egyéb műveletek is tartoznak. Ezeknek a műveleteknek az assembler számára való jelzésére az emlékeztető szimbólumokhoz hasonló szimbólumokat használnak. Ezek a szimbólumok nem a mikroszámítógép, hanem az assembler számára jeleznek „utasítást", ezért is nevezzük pszeudoutasításoknak.

A leggyakrabban előforduló pszeudoutasítások típusai az alábbiak:

- adatok megadását végző (**DB, DEFB, DW, DEFW, DEFM**),
- program futása közben keletkező adatok részére tárhelyet kijelölő (lefoglaló) (**DS, DEFS**),
- azonosítónak, szimbólumnak értéket adó (**EQU**),
- a program tárban való elhelyezését végző (**ORG**: origin).

Adatok megadását és a tárba való elhelyezését az előbbi programpéldában a **DB** és **DEFB** pszeudoutasítások (**DB:** Define Byte) végzik. Ezek operandusát az assembler a tárgyprogramba behelyezi. Ezek a pszeudotutasítások többféle adat: byte, szó (**DW:** Define Word) stb. értékének és szöveges információ írásjelkódjainak (**DEFM:** DEFine Message) a megadására és tárgyprogramba helyezésére nyújtanak módot.

A fordítóprogram a tárgyprogram címeit folyamatosan és kihagyások nélkül növeli. Ha a programnak adatok tárolására és a tárba helyezésére van szüksége, de ezek az adatok nem kiindulási adatok, akkor használjuk a tárhelyeket kijelölő pszeudoutasításokat. Egy ilyen direktíva, pl. a **DS** 10 (**DS:** Define Storage) hatására a fordítóprogram tízzel megnöveli a következő utasítás címét, s így 10 tárhelyet szabadon hagy az adatoknak.

Szimbólumok definiálására is láttunk példát a programrészben. Ez az **EQU** pszeudoutasítás, amely a címkemezőben levő szimbólumhoz hozzárendeli az operandusában található értéket.

A tárgyprogramnak vagy részeinek a tárban elfoglalt helyét pszeudoutasításokkal határozhatjuk meg. Pl. az **ORG** 1000 pszeudoutasítás azt jelzi, hogy az ezt követő információk programba való fordítását az 1000. címtől kezdje az assembler.

9.3. PROGRAMFELÉPITÉSI MÓDSZEREK

Végrehajtásra kész programok

A mikroprocesszorok programjainak egyik általános jellemzője, hogy a berendezés feszültség alá helyezése után működésre kész állapotban vannak. A bekapcsolás után a mikroprocesszorokban törlési művelet indul, amelynek során az utasításszámláló tartalma egy meghatározott értéket vesz fel. A program működése erről a címről indul el. A program kezdetén az ún. inicializáló utasítások találhatók, amelyek a hardver-építőelemek szükséges törlési műveleteit végzik el, ill. előkészítik a rendszert a felhasználói programok futásához. A programnak ezt a részét feltétlenül ROM-ban kell tárolni.

Programhurkok

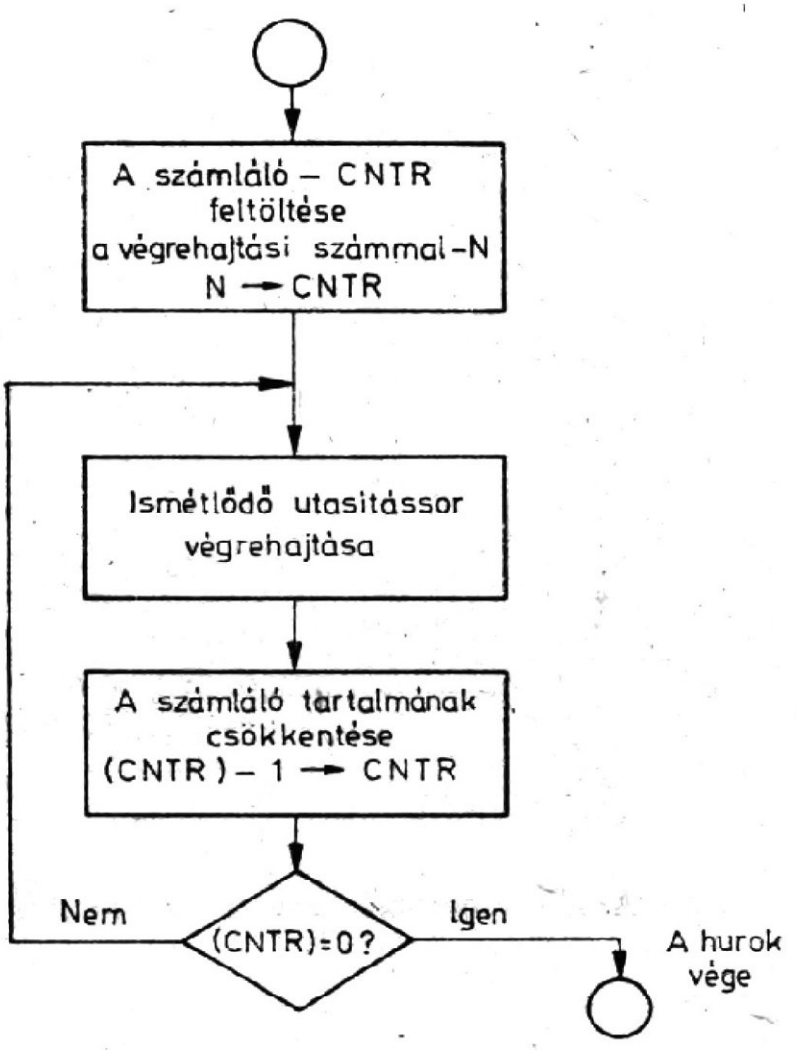
A programok gyakran tartalmaznak többször ismétlendő utasítássorozatot. A legegyszerűbb példa erre egy paraméter szorzása egy konstans számmal. A szorzás visszavezethető ismételt összeadásra. Példaként a következő programot mutatjuk be (Intel 8080A utasításokkal), feltételezve, hogy a *B* regiszterben található a 20-szal megszorzandó paraméter, és az eredményt az *A* regiszterben kapjuk meg (a túlcsordulásoktól eltekintünk a példában):

```
        ⋮
XRA  A   ; A regiszter törlése
ADD  B   ; B és A tartalmának összeadása
ADD  B
ADD  B
  ⋮
ADD  B
ADD  B
  ⋮
```

Az ismétlődő utasítássorozat többszöri leírása nem eredményez jó tárkihasználást (gondoljunk csak arra, ha nemcsak egy utasítás ismétlődik és azok is sokkal többször). Ebben az esetben rövidebb program írható, ha az ismétlődő részt csak egyszer írjuk le, de elágazási utasításokkal többször hajtjuk végre. Ezt a többször végrehajtandó részt programhuroknak nevezzük. A kellő számú végrehajtáshoz szoftverszámlálóra is szükség van, amelynek a tartalma jelzi, hogy hány ismétlés lesz még szükséges. Az előző programrészt átalakítva, és a C regisztert számlálóként felhasználva a következő programot kapjuk:

```
        ⋮
        XRA A          ; A törlése
        MVI C,20       ; számláló C regiszter feltöltése a végrehajtási
                       ; számmal
HUROK:  ADD B          ; a programhurok kezdete
        DCR C          ; számláló tartalmának csökkentése
        JNZ HUROK      ; ha még kell ismételni, ugorj a hurok kezde-
        ⋮              ; tére (Z ≠ 0)
```

Egy másik számlálási megoldás lehet a számláló tartalmának a növelése, ekkor azonban eredetileg a végrehajtási szám negatív értékét (kettes komplemensét) kell a számlálóba betölteni. A hurkok általános folyamatábráját a 9.3. ábra mutatja.



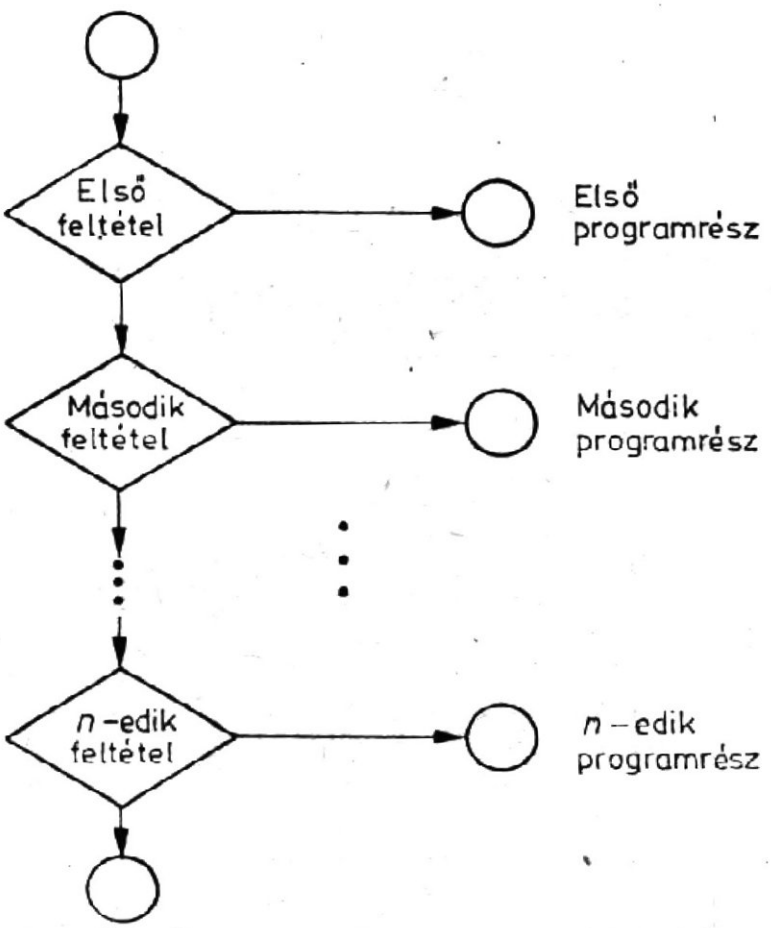
9.3. ÁBRA. A programhurok folyamatábrája

Programelágazások

Az adatok és programállapotok függvényében egy algoritmusban különböző lépések elvégzésére lehet szükség. A különböző algoritmuslépésekre feltételvizsgáló és programelágazási utasításokkal tér rá a program. Sok feltétel esetén az elágazások egyik módja az egymás utáni feltételvizsgálatok és ugrási utasítások alkalmazása (9.4. ábra). Egy másik módszer, amely sok elágazási feltétel esetén hatékonyabb lehet, az elágazási táblázatok használata. Ez a tárban levő táblázat tartalmazza a különböző algoritmuslépésekhez tartozó programrészek kezdőcímeit, amelyekre közvetett címzéssel térhetünk rá. Az alábbi programpéldában az egyik regiszter (C) tartalma a feltételek numerikus értéke, amely szerint a 0-dik, első, ... *n*-edik programrészre kell rátérni (a táblázat egy „tárlapon" foglal helyet):

;az elágazási táblázat, ha $n = 21$

```
TABL: DW PROG0        ; a 0. programrész kezdőcíme
      DW PROG1        ; az első programrész kezdőcíme
      :
      DW PROG20       ; a huszadik programrész kezdőcíme
      :
```



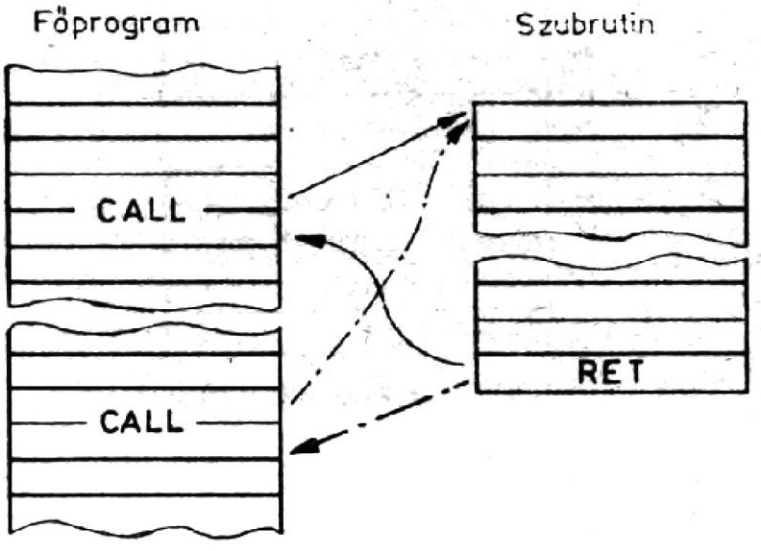
9.4. ÁBRA. Programelágazások sok feltétellel

```
;feltételvizsgálat, elágazás
;C tartalmazza a feltételszámot
FELTV: MOV  A,C
       CPI  21          ; 20-nál nem nagyobb C tartalma?
       JNC  HIBA        ; ha igen, hibajelző programrészre ugrás
       RLC              ; kettővel megszorozni a C tartalmát,
                        ; a táblázatban a címek kétbyte-osak
       LXI  H,TABL      ; a táblázat kezdőcíme HL-ben
       ADD  L           ; A-ban a táblázaton belüli cím
       MOV  L,A         ; HL-ben a feltételhez tartozó programrész
                        ; táblázaton belüli címe
       MOV  E,M         ; a programrész kezdőcímének a betöltése
       INX  H
       MOV  D,M         ; kezdőcím DE-ben
       XCHG             ; kezdőcím HL-ben
       PCHL             ; PC-ben a programrész kezdőcíme, közvetett
         :              ; címzésű ugrási utasítás
HIBA:                   ; hibajelző programrész
```

Szubrutinok, veremtár

Gyakran előfordul, hogy egy programban egy-egy több utasításból álló adatfeldolgozási folyamat többször megismétlődik. A programot úgy is felépíthetjük, hogy a folyamathoz tartozó utasítássorozatot csak egyszer írjuk meg, csak egy helyen helyezzük el a tárban. A program végrehajtása során azonban többször is rátérünk, átadjuk a vezérlést erre a programrészre. Az ilyen utasítássorozatot szubrutinnak nevezzük. A szubrutin végén azután a program visszatér az azt követő helyre, ahonnan a szubrutinra rátért. A szubrutinra rátérő programot főprogramnak (main program) hívjuk, a szubrutinra való rátérést szubrutinhívásnak (subroutine call), a főprogramba való visszalépést visszatérésnek (return) nevezzük. A szubrutinhívást és visszatérést a **CALL** és a **RET** utasítások vezérlik a 9.5. ábra szerint.

A szubrutinok hívásának és a főprogramba való visszatérésnek a legelterjedtebb módja az ún. veremtárat (stack) használó eljárás. A veremtár lehet belső, mikroprocesszorba beépített és korlátozott „mélységű", és lehet tárba kihelyezett. Ez utóbbi terjedt el a mikroprocesszorok többségénél, előnye, hogy mind a veremtár táron belüli helyét, mind pedig a méretét a programozó adja meg. A méretet csak a

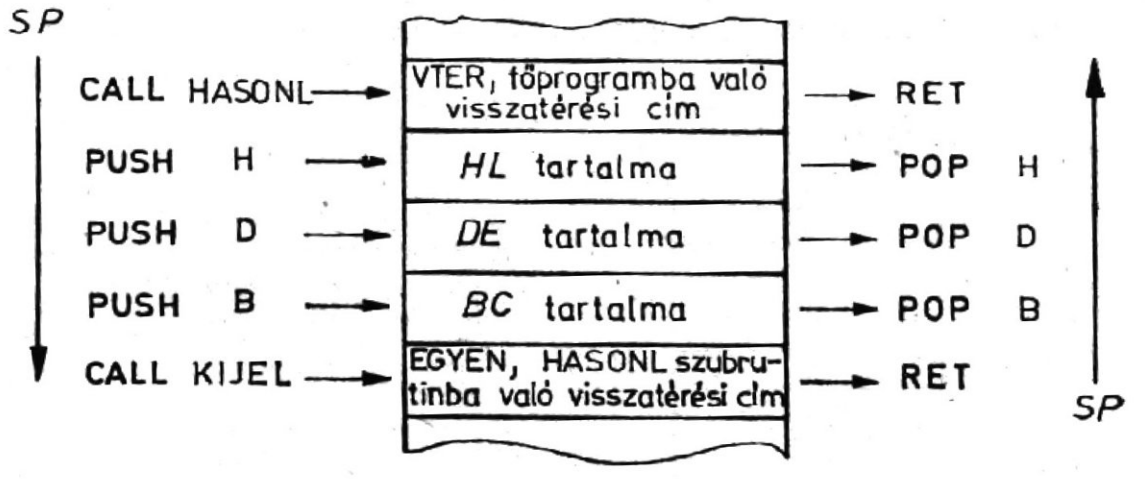


9.5. ÁBRA. Szubrutin hívása és visszatérés a főprogramba

szabad RAM-terület korlátozza. A mikroprocesszor ez esetben egy veremtármutatót (SP: Stack Pointer) tartalmaz, amely a veremtárban levő utolsó adatra mutat rá. A **CALL** utasítás automatikusan betölti a veremtárba a visszatérési címet, majd lépteti a veremtármutató tartalmát, hogy egy újabb veremtárművelet ne rontsa el a visszatérési címet, és ezt követően rátér a szubrutinra. A szubrutinon belül további szubrutinhívások is elvégezhetők, hiszen az újabb **CALL** a következő helyre tölti be a visszatérési címet. Csak a veremtár mérete korlátozza az egymáson belüli (egymásba skatulyázott) szubrutinhívásokat. A **RET** utasítás kiolvassa a **CALL** által beírt visszatérési címet, visszafelé lépteti a mutató tartalmát, majd visszatér a főprogramba.

A szubrutinokban más célra használja a program a mikroprocesszor belső regisztereit. Ha a főprogram értékes részeredményeket tárol ideiglenesen a regiszterekben, és közben hívja a szubrutint, amely használja ezeket a regisztereket, a részeredmények elveszhetnek. Ez ellen az adatok tárba mentésével lehet védekezni. Az egyik lehetőség az, hogy a regiszterek mentésére a veremtárat használjuk. A regiszterek mentésére két utasítás használható: az adat veremtárba helyezése (**PUSH**) és a veremtárból való kiemelése (**POP**). Címzésre a veremtármutató szolgál, és tartalma ugyanúgy változik ennek a két utasításnak a során, mint ahogyan ezt a **CALL** és a **RET** utasítások kapcsán leírtuk. Tekintve, hogy a veremtármutató címtartalma automatikusan növekszik, ill. csökken, a regiszterek tartalmát ellenkező irányú sorrendben kell kiemelni, mint ahogyan azt oda elmentettük (l. a következő példát).

A szubrutin használatára egy olyan, Intel 8080A-ra írt programot mutatunk be, amely két *n* byte hosszú adatblokk tartalmát összehasonlítja egymással. Az összehasonlítás során az eltérő adatokat egy a szubrutinba ágyazott újabb szubrutin (KIJEL kezdőcímmel) kijelzi. Ezt a kijelző szubrutint a példában nem tüntettük fel, csak a hívása látható. A kijelzésen kívül az összehasonlító szubrutin visszatérésekor A-ban jelzi a főprogramnak az eltérések számát. A szubrutin hívása előtt a főprogram HL-ben és DE-ben adja meg a blokkok kezdőcímét, A-regiszterben pedig az *n* számot, azaz a blokkok hosszát. A főprogram eltérések esetén a LEALL nevű programrészre tér rá (ezt már nem írtuk meg a példában). Ha nincs eltérés, akkor folytatódik a főprogram. A veremtár használatát és az ezzel kapcsolatos utasításokat a 9.6. ábrán tüntettük fel.



9.6. ÁBRA. Példa a veremtár használatára

```
                    :
;főprogramrész, amely az összehasonlítást kezdeményezi
                LXI   H,BLOKK1  ; első blokk kezdőcíme HL-regiszterpárban
                LXI   D,BLOKK2  ; második blokk kezdőcíme DE-ben
                MVI   A,DARAB   ; az A regiszterben az n szám
                CALL  HASONL    ; szubrutinhívás
VTER:           AND   A         ; A regiszter tartalma (eltérés) = 0?
                JNZ   LEALL     ; van eltérés, ugorj leállásra
;a főprogram folytatása
                    :
;az összehasonlító szubrutin
HASONL:         PUSH  H         ; HL mentése a veremtárba
                PUSH  D         ; DE mentése
                PUSH  B         ; BC mentése
                MOV   B,A       ; B a darabszámláló
                MVI   C,0       ; C az eltérésszámláló
HUROK:          LDAX  D         ; DE-vel megcímzett BLOKK2 adata A-ban
                CMP   M         ; HL-lel megcímzett BLOKK1 adatával össze-
                                ; hasonlítás
                JZ    EGYEN     ; ha egyenlők, ugorj (Z = 0)
                INR   C         ; ha nem egyenlők, számlálj
                CALL  KIJEL     ; eltérés esetén kijelzés hívása
EGYEN:          INX   D         ; BLOKK2 címregiszter (DE) növelése
                INX   H         ; BLOKK1 címregiszter (HL) növelése
                DCR   B         ; darabszám csökkentése
                JNZ   HUROK     ; vége a blokknak? ha nem, ugorj
                MOV   A,C       ; eltérésszám A-ban
                POP   B         ; BC regiszter tartalmának visszaírása
                POP   D         ; DE visszaírása
                POP   H         ; HL visszaírása
                RET             ; visszatérés a főprogramba
;kijelzés szubrutin
KIJEL               :
                RET
```

9.4. MAKROUTASITÁSOK ÉS A FELTÉTELES FORDÍTÁS

A makroutasításokkal az egy programban használható utasításkészlet növelhető [9]. A programozó a programjában az alaputasításokból olyan makroutasításokat (makrót) készíthet, amelyek az adott alkalmazáshoz kényelmesebben használhatók. A makroutasítás nem más, mint az assembler (makro assembler) fordítóprogramjának definiált és névvel ellátott utasítássorozata, amelyet a hivatkozások helyén az assembler beiktat a tárgyprogramba. A makroutasítások használata – ellentétben a szubrutinokkal – nem jelenti azt, hogy a makrót képező utasítássorozatot csak egyszer kell a tárba elhelyezni, és a program különböző helyeiről hívható. Makróval tehát nincs mód a tárhelyek gazdaságosabb kihasználására. A makrókkal programírási munkát, időt takaríthatunk meg, amennyiben a többször előforduló utasítássorozatot csak egyszer kell definiálni (leírni) az assembler számára. Minden makrohivatkozáskor az assembler fordítóprogramja automatikusan beépíti a tárgyprogramba az egész utasítássorozatot.

A makroutasítások egy vagy több paraméterrel is elláthatók. A paraméterszimbólumok számát és sorrendjét a makró definiálásakor adjuk meg. Hivatkozáskor a paraméterszimbólumok helyett a konkrét paraméterek értékét vagy az ezt reprezentáló szimbólumokat helyezzük el a megfelelő sorrendben.

A definiálásra példaként a tárban elhelyezett két byte összeadását és az eredmény egy harmadik tárhelyre való beírását végző makrót adunk meg. A makrót nevezzük **ADD2B**-nek. Az összeadandó byte címeit és az eredmény címét paraméterenként definiáljuk, hogy a makró hívásakor mindhárom címet tetszőlegesen megadhassuk. A definiálásra a **MACRO** pszeudoutasítás ad parancsot az assemblernek, a nevet címkeként, a paramétereket pedig operandusként kell megadni:

```
;makrodefiníció
ADD2B:  MACRO P1,P2,P3   ; definíció és formális paraméterek
        LDA P1           ; töltsd A-ba a P1 című byte-ot
        LXI H,P2         ; HL-be a P2 címét töltsd
        ADD M            ; összeadás
        STA P3           ; az eredmény tárba írása (P3 címre)
        ENDM             ; a makró vége
```

Az **ENDM** pszeudoutasítás jelzi a makró végét. A felsorolt paraméterek formális paraméterek (dummy parameters), hiszen P1, P2 és P3 szimbólumokat a program nem tartalmaz, helyettük a makró hívásakor az ugyanebben a sorrendben szereplő értékeket fordítja be az assembler. A formális paraméterek csak azt jelzik, hogy a makrón belüli utasítások operandusaként mely paramétereket kell használni. A **MACRO** és **ENDM** pszeudoutasítások közötti programállítások sorozata alkotja a makrotörzset (macro body).

A makróra hivatkozást makrohívásnak (macro call) nevezzük. A makrohíváskor a paraméterek behelyettesítését paraméterkifejtésnek nevezzük. Az alábbi példában az:

```
ADD2B  SZAM1,  SZAM2,  ERED
```

típusú makrohívás az alábbi utasítássorozat tárgyprogramba való beépítését eredményezi

```
LDA  SZAM1
LXI  H,SZAM2
ADD  M
STA  SZAM3
```

Kifejtéskor a formális paraméterek helyén bármilyen legális assembler kifejezés lehet (címke, emlékeztető szimbólum, szám, regiszternév).

Példaként definiáljuk a következő makrót:

```
SHIFT:  MACRO  UTAS,REG
        MOV    A,REG
        UTAS
        MOV    REG,A
        ENDM
```

Ez a makró valamely regiszter tartalmával az UTAS nevű utasításparaméter műveletét végzi el (pl.: jobbra vagy balra tolás). Hívjuk ezt a makrót a D regiszter tartalmának a jobbra tolásához az alábbiak szerint:

```
        SHIFT    RRA,D
```

A makró hívása a következő utasítássorozat befordítását eredményezi:

```
MOV   A,D
RRA
MOV   D,A
```

Más paraméterek, mint pl.:

```
SHIFT   RLC,B
```

a B regiszter jobbra tolását eredményezik az alábbi utasításokkal:

```
MOV   A,B
RLC
MOV   B,A
```

Makrók egymásba ágyazott hívására is van mód. Az alábbi példában a **MOVIN** makró valamelyik regiszter közvetett címzésű betöltésére szolgál. A **MOVIN** formális paraméterei: a cím : ADR és a regiszter : R.

```
MOVIN:  MACRO R,ADR
        LHLD  ADR     ; HL-ben a cím
        MOV   R,M     ; R-ben a HL-lel címzett adat
        ENDM
```

Az ATLAG nevű makró négy különböző tárcímen levő adat *a, b, c, d* átlagát képezi. Formális paraméter: a négy adat címe: PAD1, PAD2, PAD3, PAD4. Az átlag az *A* regiszterben marad (a kerekítésektől eltekintünk a példában):

Az átlagszámítási algoritmus: $\left(\frac{c+d}{2}+\frac{a+b}{2}\right):2$

```
ATLAG:  MACRO PAD1, PAD2, PAD3, PAD4
        MOVIN B,PAD1  ; első adat B-ben
        MOVIN C,PAD2  ; második adat C-ben
        MOVIN D,PAD3  ; harmadik adat D-ben
        MOVIN A,PAD4  ; negyedik adat A-ban
        ADD   D       ; 3. és 4. adat összege
        RRC           ; osztva 2-vel
        MOV   D,A     ; 3. és 4. átlaga D-ben
        MOV   A,B     ; első adat A-ban
        ADD   C       ; első és második adat összege
        RRC           ; osztva 2-vel, első és második átlaga A-ban
        ADD   D       ; első kettő + második kettő átlaga
        RRC           ; osztva 2-vel, átlag A-ban
        ENDM
```

Az átlagképző hívásakor, ha a négy adatbyte címe: CIM1, CIM2, CIM3 és CIM4:

```
ATLAG  CIM1,  CIM2,  CIM3,  CIM4
```

az assembler az alábbi utasítássorozatot generálja:

```
LHLD  CIM1      ; első MOVIN
MOV   B,M
LHLD  CIM2      ; második MOVIN
MOV   C,M
LHLD  CIM3      ; harmadik MOVIN
MOV   D,M
LHLD  CIM4      ; negyedik MOVIN
MOV   A,M
ADD   D
RRC
MOV   D,A
MOV   A,B
ADD   C
RRC
ADD   D
RRC
```

Egyes makro-assemblerek lehetővé teszik, hogy a makrók törzsén belül programhurkokat képezzünk, és a programhurkok definiálásához a makrón belül is címkét definiálhassunk. Ez a címke eltér a makrókon kívüli címkéktől, hiszen a makróra való többszöri hivatkozás esetén sem szabad az assemblernek „többszörösen definiált címke" hibajelzést adni. Az egyik megoldás, hogy a makrón belüli, ún. implicit címke szimbólumában csak a betűk egy részét választhatja meg tetszés szerint a programozó, a további betűk jelzik, hogy ez implicit címke. Az assembler minden makrohíváskor meghagyja a címkének a programozó által szabadon választott részét, és sorszámot rendel az „implicit jelzés" helyett. A sorszám minden hívásnál változik, így a makró kifejtésekor nem keletkezik többszörösen definiált szimbólum.

A következő példában a BLATL makró a tárban levő két adatblokk (max 256 byte hosszú) azonos sorszámú byte-jainak az átlagát képezi. Az átlagokat a második adatblokk helyére teszi. Az adatblokkok címeit a C1, C2 formális paraméterek, a blokkhosszat pedig a DRB formális paraméter adja. Az implicit címke jelzése a XX#$YM, ahol XX-et a programozó szabadon adhatja meg.

```
;Makrodefiníció
BLATL:    MACRO C1,C2,DRB
          LXI   D,C1      ; az első blokk címei DE-ben
          LXI   H,C2      ; a 2. blokk címei HL-ben
          MVI   B,DRB     ; blokkhossz B-ben
HR#$YM:   LDAX  D         ; első blokk adata
          ADD   M         ; plusz a második blokk adata
          RRA             ; osztva 2-vel
          MOV   M,A       ; visszaírás a tárba
          INX   D         ; DE címregiszter növelése
          INX   H         ; HL címregiszter növelése
          DCR   B         ; blokkvég?
          JNZ   HR#$YM    ; ha nem, ugorj HR#$YM-re
          ENDM            ; makró vége
```

A BLATL makró első hívásakor a címkemezőben és az ugrási utasítás operandusában az implicit szimbólumnak a HR0000 címkét, második híváskor a HR0001 címkét és így tovább felelteti meg az assemblerprogram.

A makró második hívása:

```
        BLATL  BL1,BL2,128
```

amelynek a kifejtéskor az assembler fordítóprogram a következő forrásprogramrészt és az ennek megfelelő tárgyprogramot generálja:

```
        LXI    D,BL1
        LXI    H,BL2
        MVI    B,128
HR0001: LDAX   D
        ADD    M
        RRA
        MOV    M,A
        INX    D
        INX    H
        DCR    B
        JNZ    HR0001
```

A makrók és a makró-assemblerek nemcsak a felhasználói programozást könnyítik meg azzal, hogy a programozó az alkalmazáshoz adaptált utasításokkal bővítheti az utasításkészletet. További előny, hogy makrók használatával a programok dokumentálása is áttekinthetőbb, könyebben kezelhető. Készültek olyan makró-assemblerek, amelyek „univerzálisak", mikroprocesszor-függetlenek [9], ugyanakkor nem kötelező ún. szabvány emlékeztető szimbólumok használata (amelyek esetleg idegenek a programozó számára), mint más univerzális assemblereknél [10], [11]. Egyes makró-assemblerek segítségével olyan forrásprogramok írhatók, amelyekben könnyen olvasható és megérthető emlékeztető szimbólumokat használhatunk, ugyanakkor az írás módja és a fordítási eljárás a magas szintű programnyelvekhez hasonló [12].

Feltételes assemblerek

Egyes – sorozatban gyártott – felhasználói rendszerek gyakran egymástól kismértékben eltérő alkalmazási területeken működnek. Ebben az esetben célszerű a működtető programot úgy megírni (a forrásprogram szintjén), hogy ne legyen szükség annak a kissé eltérő alkalmazások miatti változtatására. Ilyen megoldást tesznek lehetővé a „feltételes assemblerek". Ezekben az assemblerekben pszeudoutasításokkal a programozó letilthatja a program egyes részeinek a fordítását.

A forrásprogramot úgy írja meg a programozó, hogy az az egymástól eltérő alkalmazásokhoz szükséges összes programrészletet tartalmazza. Az alkalmazásfüggő programrészeket azonban ilyen pszeudoutasítások közé kell beírni. A pszeudoutasítások paramétereket is tartalmazhatnak. A paraméter értéke a feltétel, és engedélyezi vagy tiltja az adott programrésznek a tárgyprogramba való beépítését. A fordítás előtt mindössze ezt a néhány feltételparamétert kell a programozónak megadnia – az alkalmazás ismeretében –, és az assembler elkészíti a kívánt tárgyprogramot.

Következőkben példát mutatunk a fordítást vezérlő pszeudoutasításokra [6]. Ilyen a **COND** pszeudoutasítás és a hozzá tartozó paraméterekből álló kifejezés, amely azt jelenti, hogy a következő programrész fordítását csak akkor kell elvégezni, ha a kifejezés értéke logikai IGAZ, azaz $\neq 0$.

Az **ENDC** pszeudoutasítás jelzi, hogy itt végződik a feltételes fordítás alá eső rész, eddig tart a **COND** pszeudoutasítás hatása. Pl.:

```
COND SZIMBOLUM
...          ; ezek az utasítások csak akkor kerülnek a tárgy-
...          ; programba, ha SZIMBÓLUM ≠ 0
ENDC
```

A feltételes fordítás és a makrók együttes használatával olyan makroutasítások építhetők fel, amelyeknek nemcsak a formális paraméterei változnak híváskor, hanem a makrotörzsben generált kódok, utasítások sorrendje és mennyisége is változhat. Feltételekhez kötötten egy makró hívhatja saját magát is közvetlenül vagy egy másik makrón belül. Ezeket a makrókat *rekurzív makróknak* nevezzük. Az ilyen esetekben a ciklikus, végtelen számú hívás megakadályozható a feltételes fordítás pszeudoutasításaival.

A rekurzív makrókkal kapcsolatos még egy pszeudoutasítás. A 9.2. fejezetben tárgyaltunk az értéket adó **EQU** pszeudoutasításról. Egy szimbólumnak az **EQU**-val egy programon belül csak egyszer adhatunk értéket, ellenkező esetben az assembler hibaüzenetet generál (többszörösen definiált címke). A **DEFL** (DEFine Label) pszeudoutasítás [6] feladata ugyancsak az értékadás, ez a direktíva azonban egy szimbólummal többször is megismételhető. Ennek a direktívának a hatása a szimbólumhoz tartozó értékre csak addig terjed ki, ameddig a szimbólumnak újabb **DEFL**-lel új értéket nem adunk. Pl.:

```
CIMKE1: DEFL 9              ; értékadás, CIMKE1 = 9
        :
CIMKE1: DEFL 6              ; új értékadás, CIMKE1 = 6
        :
CIMKE1: DEFL CIMKE1+2       ; új értékadás, CIMKE1 = 6+2 = 8
        :
```

A következő példa azt mutatja, hogy rekurzív makrohasználattal, hogyan lehet változó számú utasításcsoportot beépíteni a programba. Rekurzív számlálóként egy makrót és a **DEFL** pszeudoutasítással megadott változót használjuk. Az ismétlendő utasításcsoport pedig egy másik makróban két **INC** és egy **DEC** utasítás legyen, amelyek operandusát a makró formális paramétereiként adjuk meg, az utasításcsoport ismétlődési számához hasonlóan. Ebben a példában a [6]-ban leírt makróassembler szabályaihoz alkalmazkodva a formális paraméterek elé # jelet írunk (Z–80-as assembler):

```
;A számlálómakró definiálása
SZAML:   MACRO   # RP1, # RP2, # RP3, # ISM
SZCMKE:  DEFL    # ISM
         REK     '# RP1', '# RP2', '# RP3'
         ENDM
```

REK a következő rekurzív makró neve, a formális paraméterek ' '-k közé helyezése azt jelenti, hogy a behelyettesítésnél nem azonosítókként kezelendők a formális paraméterek, hanem szöveginformációként, azaz ASCII kódban kell a formális paramétert kifejteni.

```
;A rekurziv makró definiálása:
REK:      MACRO    # RP1, # RP2, # RP3
          COND     SZCMKE
SZCMKE:   DEFL     SZCMKE-1
          REK      '# RP1', '# RP2', '# RP3'
          INC      '# RP1'
          INC      '# RP2'
          DEC      '# RP3'
          ENDC
          ENDM
```

A makrót, ha pl. kétszer kívánjuk megadni az **INC HL, INC DE, DEC BC** utasítássorozatot, az alábbiak szerint hívjuk:

```
          SZAML    HL, DE, BC, 2
```

és a makrót az assembler az alábbiak szerint fejti ki:

```
SZCMKE:   DEFL  2                ; címke értéke: 2
          REK   'HL', 'DE', 'BC' ; REK 1. hívása SZAML-ból
          COND  SZCMKE           ; feltétel IGAZ
SZCMKE:   DEFL  SZCMKE-1         ; címke értéke: 1
          REK   'HL', 'DE', 'BC' ; rekurzív REK, 2. hívás
          COND  SZCMKE           ; feltétel IGAZ
SZCMKE:   DEFL  SZCMKE-1         ; címke értéke 0
          REK   'HL', 'DE', 'BC' ; rekurzív REK, 3. hívás
          COND  SZCMKE           ; feltétel HAMIS, ne fordíts!
          REK   'HL', 'DE', 'BC' ; nem fordítja, nem hívja REK-et
          INC   HL               ; nem fordítja, 3. REK hívás
          INC   DE               ; nem fordítja, 3. REK hívás
          INC   BC               ; nem fordítja, 3. REK hívás
          ENDC                   ; feltétel vége! tovább fordíts!
;itt folytatódik a 2. REK hívás kifejtése fordítással
cím    23     INC   HL  ; a 2. REK híváshoz tartozó
cím+1  13     INC   DE  ; utasítások
cím+2  0B     DEC   BC
               ENDC     ; hatástalan pszeudoutasítás
cím+3  23     INC   HL  ; 1. REK híváshoz tartozó
cím+4  13     INC   DE  : utasítások
cím+5  0B     DEC   BC
               ENDC     ; hatástalan pszeudoutasítás
```

A REK első hivatását a SZAML makró végezte. A REK második hívása rekurzív, majd a harmadik hívás is rekurzív. A harmadik hívás kifejtésekor a fordítást kikapcsolja a COND pszeudoutasítás, hiszen itt a SZMCKE értéke *0* lesz (logikai HAMIS). Ezért a 4. REK hívás már nem érvényesül, és a harmadik híváshoz tartozó utasításoknál, az INC, INC, DEC kifejtésekor a fordítás még mindig tiltott. A fordítás tiltását feloldja az első ENDC pszeudoutasítás. Ezt követően hátra van még a második, ill. az első REK híváshoz tartozó lefordítandó utasítások kifejtése, ami a rekurzív hívások miatt elmaradt. A programlistán a *cím*, és ezt követő *cím+1*, ... az érvényes utasítások címét, a 23, 13 és 0B értékek pedig az utasítások kódját jelölik (feltüntetésük a programlistát szimbolizálja, ez mutatja, hogy az assembler mit fordít be a tárgyprogramba).

9.5. ÁTHELYEZHETŐ PROGRAMOK IRÁSA

A 8. fejezetben az operációs rendszerekkel kapcsolatban tárgyaltuk az áthelyezhető programok jelentőségét, használatuk előnyeit. Az áthelyezhető programok írásához és lefordításához megfelelő assemblerek használata szükséges. Az assembler a fordítás befejeztével ún. áthelyezhető tárgyprogramot készít.

Az ilyen tárgyprogram tárba töltéséhez megfelelő töltőprogramot, ill. csatolóprogramot használunk. A továbbiakban a töltőprogrammal külön nem foglalkozunk. Ismertetjük az assembler áthelyezéssel kapcsolatos funkcióit és egy korszerű hajlékony mágneslemezes rendszerben használatos csatolóprogramot [6].

Az assembler a tárgyprogramhoz rögzített tárhelyeket nem rendel. Ez a csatolóprogram feladata. A felhasználó viszont egy-egy alkalmazás kapcsán több áthelyezhető programmodul együttes működését is kérheti. Az együttes működéshez a modulok közötti kommunikációra is szükség lehet (pl. ha egyik modul használja a másik szubrutinjait). Ezért az assembler a tárgyprogramhoz olyan információkat is hozzárendel, amelyet a csatolóprogram felhasznál a betöltendő programmodulok közötti helyes kommunikáció biztosításához. A modulok közötti kommunikációhoz a programozónak meg kell adnia, hogy melyik operandusokat (adatok vagy utasításcímek) használja közösen több programmodul. Ezek megadásához két pszeudoutasítást használhat:

- **GLOBAL** – a paramétereiben levő szimbólumokat az adott modulon belül deklaráljuk,
- **EXTERNAL** – paraméterei olyan szimbólumok, amelyeket az adott programmodul használ, de másik modulban deklaráltuk őket.

Az assembler ezeket a szimbólumokat az áthelyezhető tárgyprogramfile-ban jelzi. A csatolóprogram feladata, hogy az összes globális (**GLOBAL**) és külső (**EXTERNAL**) szimbólumot figyelemmel kísérje, amikor az áthelyezhető programokhoz a konkrét tárcímeket hozzárendeli.

A pszeudoutasítások használatát bemutató példában tételezzük fel, hogy az egyik programmodul a főprogram, ez tartalmazza a be-, ill. kimeneti adatmezőket, a másik modul pedig a perifériák kezelőrutinjait tartalmazza. A be-, ill. kimeneti mezők címkéi: BEMEM és KIMEM, a be-, ill. kimeneti szubrutinok kezdőcímei pedig az ADATBE és ADATKI címkékkel jelöltek.

A főprogram részlete a következő (Z–80 assembler nyelven):

```
;modulok közötti kommunikáció címkéi a főprogramban
            GLOBAL     BEMEM, KIMEM
            EXTERNAL ADATBE, ADATKI
;a főprogram
                :
            CALL       ADATBE   ; bemeneti szubrutin hívása
                :
            CALL       ADATKI   ; kimeneti szubrutin hívása
                :
BEMEM:      DEFS       100      ; bemeneti adatmező: 100 byte
KIMEM:      DEFS       100      ; kimeneti adatmező: 100 byte
                :
```

A szubrutinok moduljának részlete:

```
;modulok közötti kommunikáció címkéi a szubrutinoknál
            GLOBAL     ADATBE, ADATKI
            EXTERNAL   BEMEM, KIMEM
ADATBE:     LD   HL,BEMEM  ; HL-ben a tármező címe
             :
            IN   A, (PERC1)  ; perifériától behozatal
            LD   (HL), A     ; az adat tárba írása
             :
            RET              ; visszatérés a főprogramba
ADATKI:     LD   HL,KIMEM  ; HL-ben a tármező címe
             :
            LD   A,(HL)      ; adat A-ban
            OUT  (PERC2),A   ; kivitel a perifériára
             :
            RET
```

Az ily módon megírt forrásprogramokat az assemblerrel külön-külön lefordíthatjuk. Az így kapott tárgyprogramok azonban futtatásra alkalmatlanok. Futásra alkalmassá a csatolóprogrammal tehetjük őket. A csatolóprogram funkciói:

- a tárcímekhez hozzárendeli az áthelyezhető programokat,
- lehetővé teszi, hogy kezdőcímet adjunk meg,
- feldolgozza a modulok közötti kommunikáció szimbólumait,
- nem létező vagy többszörösen definiált szimbólumok esetén hibajelzést ad,
- betöltési táblázatot készít, amelyben az egyes programmodulok tárhelyfoglalása, kezdőcíme és a globális szimbólumok nevei és értékei találhatók.

A csatolóprogram használatát és az áthelyezési paraméterek megadását az alábbiakban foglaljuk össze. A programozó a képernyős megjelenítő vagy írógép billentyűzetéről ad parancsot az alábbi formában [6]:

```
LINK  $ = 1000  PROG1  PROG2  $ = 4000  PROG3
```

Ez a parancs azt jelzi, hogy az 1000 tárcímtől kezdődően kell a PROG1 és PROG2 nevű, hajlékony mágneslemezes táron tárolt tárgyprogramokat elhelyezni, a PROG3-nevűt pedig a 4000 címtől kezdődően. Ha a programfutás kezdőcímét nem adja meg a programozó, a csatolóprogram a legelső betöltési helyet (a példában az 1000 cím) jelöli meg a programfutás kezdeteként. A következő példában:

```
LINK  $ = 5000  PROG  (E = 5080)
```

az (E = 5080) paraméterrel jelzi a kezelő a programfutás kezdőcímét (E: Entry point: belépési vagy kezdőcím).

A csatolóprogram az áthelyezhető programokról a hajlékony mágneslemezre ún. eljárásfile-okat készít. Ezek már olyan tárgyprogramfile-ok, amelyeket az operációs rendszer parancsaival a tárba be tudunk tölteni végrehajtáshoz. Az áthelyezhető programok használatának összes előnyét csak jól felépített operációs rendszerekben tudjuk kihasználni.

IRODALOM

[1] *Donovan, J. J.:* Systems programming. McGraw-Hill Book Company, New York, 1972.

[2] *Hilburn, J. L.–Julich P. M.:* Microcomputers, microprocessors: hardware, software and applications. Prentice-Hall Inc., Englewood Cliffs, 1976.

[3] *Wester, J. G.–Simpson, W. D.:* Software design for microprocessors. Texas Instruments Learning Center, 1976.

[4] Intel 8080. Assembly language programming manual. Intel Corp., 1975.

[5] Z–80 microcomputer system. Technical manual. Mostek Inc., 1977.

[6] Z–80–RIO Relocating assembler and linker. User's manual. Zilog Inc., 1978.

[7] *Clark, T.:* Systems design flexibility. Systems International, March 1979. 19–22.

[8] *Csákány A.–dr. Vajda F.:* Mikroszámítógépek. Műszaki Könyvkiadó, Budapest, 1976.

[9] *Cohen, H. A.–Francis, R. S.:* Macro-assemblers and macro-based languages in microprocessor software development. Computer, Febr. 1979. 53–64.

[10] *Marshall, M.:* Assembly language plan raises dust. Electronics, January 17, 1980. 98–100.

[11] *Tiberghien, J.–Taeymans, J.:* CASS8: a cross assembler suitable for most of the 8 bit microprocessors. Proc. of Euromicro, 1977. North Holland Publication Co., Amsterdam, 80–91.

[12] *Krieger, M.:* Structured assembly language suits programmers and microprocessors. Electronics, Jan. 17, 1980. 118–122.

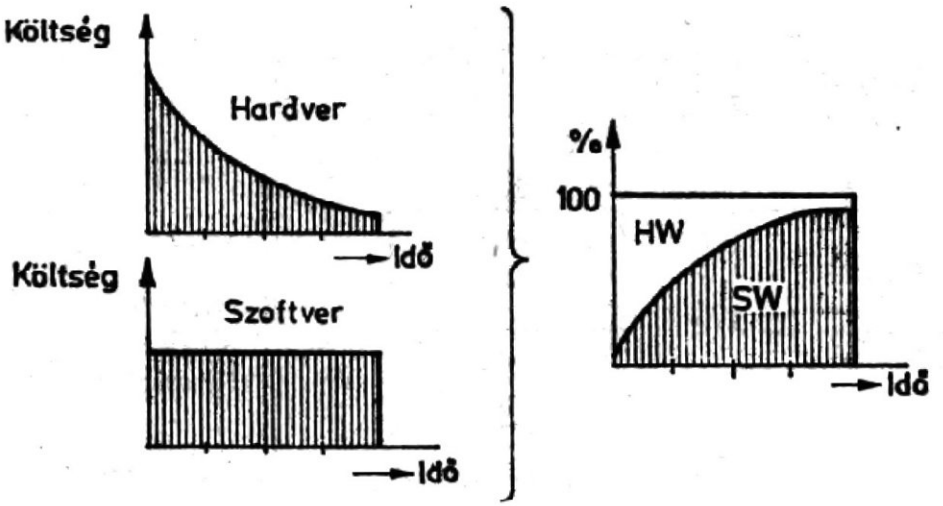
10. MAGAS SZINTŰ PROGRAMNYELVEK ALKALMAZÁSA

Az elmúlt időszakban a mikroprocesszoros rendszerek programjai majdnem kizárólag a gépi kóddal megegyező felépítésű géptől függő assembler nyelven készültek (l. a 9. fejezetet). Ennek következtében míg a hardver ára folyamatosan csökkent, a szoftverköltségek lényegében nem változtak. Így a szoftverköltségek egyre nagyobb részét teszik ki a teljes rendszer költségeinek (10.1. ábra). Különösen igaz ez, ha figyelembe vesszük a fejlődés irányát, amely szerint:

- egyre fejlettebb, az „igazi" számítógép-kategóriába eső rendszerprogramokra (l. a 8. fejezetet) van szükség,
- a mikroprocesszoros rendszerek segítségével megoldott feladatok is egyre összetettebbek.

Érthető, hogy ebben a helyzetben a szoftverfejlesztés gazdaságossága, a programozás hatásfokának javítása nagy jelentőségűvé vált.

Ezen kívül a programok megbízhatóságának a javítása és karbantartásának az egyszerűsítése is előtérbe került és szerepet játszik. Mindezek megnövelték a „strukturált programozás" [2] módszerei iránt megnyilvánuló érdeklődést, és megnövelték a moduláris programozás jelentőségét. Valamennyi említett tényező a magas szintű programnyelveknek – mikroprocesszoros rendszerekben való – egyre elterjedtebb alkalmazása irányába hat. Ezek a nyelvek jobb hatásfokú programozást tesznek lehetővé, közvetlenebb kapcsolatot mutatva a megoldások algoritmusai, a program folyamatábra elemei és a program utasításai között. Ugyanakkor a programírás, -módosítás és -karbantartás is egyszerűbbé válik, mivel a programok moduljai egymástól függetlenül készíthetők el és így módosíthatók is. Az egyes nem szükséges részletek elrejtése (hiding) [3], a modulok „dobozonként" való kezelése (encapsulating: bedobozolás) a megértést és a dokumentálhatóságot is elősegítik.

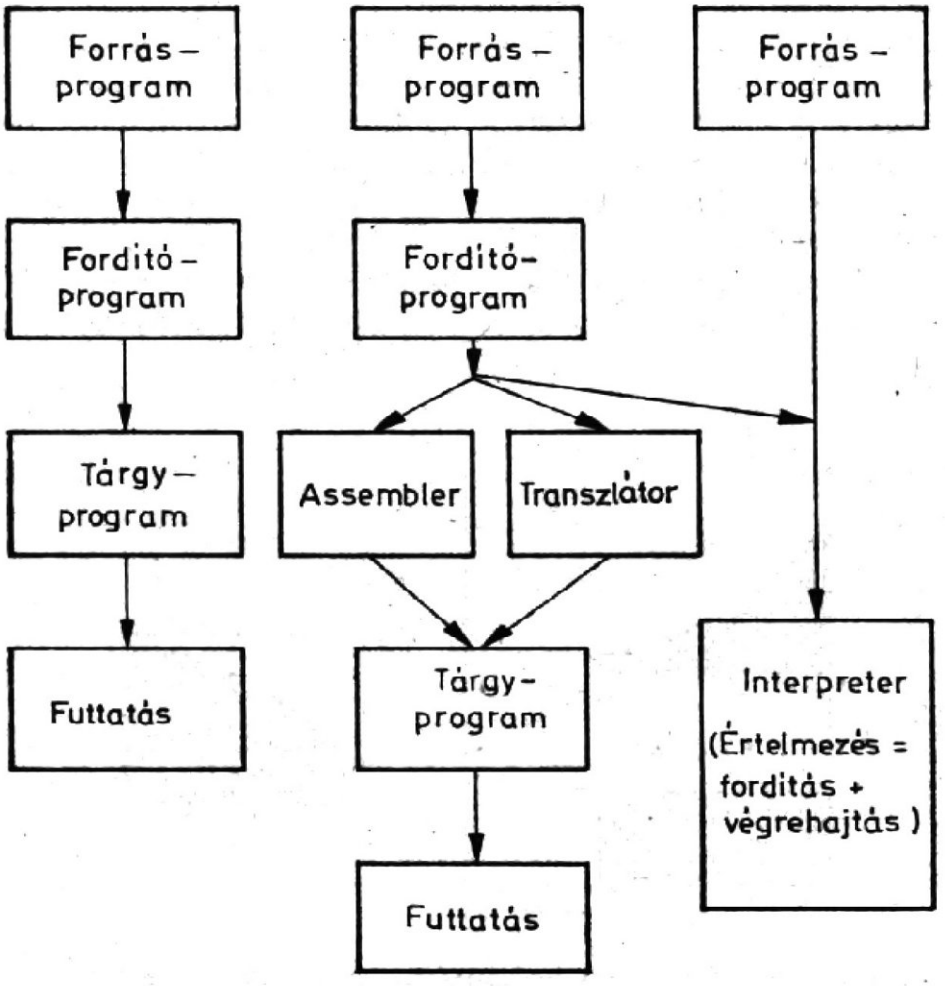


10.1. ÁBRA. Hardver–szoftver költségek és ennek következtében a HW–SW költségarányok változása

A magas szintű programnyelvek külön csoportját képezik a rendszerprogramozási nyelvek (pl. a CDL), amelyek algebrai jelölést és magas szintű vezérlő struktúrákat valósítanak meg olyan feladatok számára, mint pl. folyamatok sorbaállítása vagy erőforrások kezelése. Ezek fő alkalmazási területe a rendszerprogramok (operációs rendszerek, magas szintű nyelvek fordítóprogramjai stb.) megvalósítása.

A fordítás egyik eszköze a kompilátor (compiler), amely a magas szintű nyelven írt forrásprogramot gépi kódra, assembler nyelvre vagy egy közbenső (intermediate) nyelvre fordíthatja le. Ha a kompilátor gépi kódot generál – amit úgy is szoktunk mondani, hogy valódi (real) kódot hoz létre –, végrehajtható kódot kapunk közvetlenül. Ha a kimenet assembler nyelvű, végrehajtás előtt ezt egy további fordítóprogram, egy assembler segítségével (l. a 9. fejezetet) kell gépi kódra fordítani. A közbenső nyelvre (pl. PASCAL p-kód, l. később) fordított programot vagy egy további fordítóprogram segítségével alakítjuk végrehajtható formára vagy a fordítás helyett a másik módszer, az értelmezés segítségével dolgozhatjuk fel (10.2. ábra).

Az értelmező- (interpreter) program a magas szintű – vagy közbenső – nyelvű program utasításait egyenként „fejti meg" és rögtön végre is hajtja. Vagyis, ha az értelmező program egy magas szintű parancsot kap, egy előre meghatározott gépi utasássorozatot működtet, amely megfelel a magas szintű utasításnak. Az értelmező nagyon hasznos a gyors programozói beavatkozást biztosító párbeszédes (interactive: egymásra ható) programozás és programjavítás esetén, mivel az utasítások egyenként kerülnek lefordításra és rögtön végrehajtásra is. Az értelmezőprogram lényegesen lassabban hajtja végre az adott programot, mint ahogy a kompilátor által lefordított tárgyprogram fut a számítógépen. Mivel azonban a programozó a párbeszédes rend-



10.2. ÁBRA. A programfordítás különböző módszerei

szerben az írógépen (vagy a képernyőn) közben írja, ill. javítja a programot, fel sem tűnik neki, hogy a végrehajtás lassú. Ha a feladathoz gyors megoldási sebességre van szükség, akkor persze nincs akadálya, hogy az ilyen módon kijavított programot kompilátor segítségével lefordítsa és ezt a változatot használja.

A teljesség kedvéért meg kell említenünk, hogy az ismertetett módszereken kívül van még egy lehetőség a magas szintű programnyelven írt programok utasításainak feldolgozására. Mivel az eljárást első ízben matematikai formulák feldolgozására hozták létre, az ily módon megvalósított rendszereket formulavezérlésű számítógépeknek nevezik [7]. Az ilyen számítógépek processzorai a magas szintű program minden egyes alapjelét egy utasításnak tekintik, és – általában mikroprogramozás segítségével (l. a 11. fejezetet) – közvetlenül, a program által meghatározott dinamikus sorrendben végrehajtják. (Pl. Pascal Microengine [8]).

A fordítást végző programok futhatnak egy, a programfejlesztésben felhasznált másik számítógépen (keresztmódszer) vagy az adott alkalmazói – vagy fejlesztői – rendszernek lehetnek „saját programjai".

Ha a rendszer-, az alkalmazói program vagy mindkettő, a háttértárban (pl. hajlékony mágneslemezes tárban) van, és onnan olvassuk be a RAM típusú operatív tárba végrehajtásra, akkor ezt szoftvermódszernek nevezik. Ettől eltérő az az eset, amikor a rendszerprogram vagy a végrehajtható felhasználói program ROM típusú tárban „integrálva" van (vagy PROM-ba, ill. REPROM-ba be van égetve).

A magas szintű programnyelven írt programokhoz egy adott feladat megoldása esetén több gépi utasításra – tehát nagyobb kapacitású tárra és több végrehajtási időre – van szükség. Bár egyre nagyobb tárkapacitású elemek kerülnek forgalomba és ezek ára is csökken, valamint a processzorok műveleti sebessége is nő (l. az 1. fejezetet), az idő és a tárhely megtakarítására sokszor mégis nagyon nagy szükség lehet. Gondoljunk pl. különböző azonos idejű alkalmazásokra, ahol az idő, vagy az egyetlen tokos mikroszámítógépekre, ahol a tárkapacitás lehet a korlátozó tényező. Általában elfogadhatjuk azt, hogy az előbbiek szerint minél kisebb darabszámban készítünk el egy rendszert, annál valószínűbb, hogy a magas szintű nyelvű programozása gazdaságosabb megoldás. Ez a „darabszám tényező" a magas szintű programnyelven írt programok fordítása hatásfokának (hatásosságának) javulásával egyre feljebb tolódik. Ezzel szemben a nagy tömegben gyártott mikroprocesszoros rendszerek esetén szinte kizárólag assembler nyelvű programokat célszerű alkalmazni.

A magas szintű programnyelvek fordító programjai által előállított kód – szükség esetén – javítható, optimalizálható. Lehetőség van a legkritikusabb részek assembler nyelvű „kézi" kódolására. A korszerű, magas szintű programnyelvek implementációi azt is lehetővé teszik, hogy magas szintű és assembler nyelvű modulokat egymással összekeverve használjunk, azokat a futtatásnál kapcsoljuk egymáshoz. Így pl. a legidőigényesebb modulokat (pl. perifériakezelés, adat be-, ill. kivitel) assembler nyelven írhatjuk meg és „építhetjük be" a magas szintű programnyelven megírt feladatmegoldó programba [9].

Ha a magas szintű programnyelvek nem felelnek meg követelményeinknek, az assembler szint pedig minket nem elégít ki, további lehetőség a makroprocesszorok használata. Ezek legegyszerűbb formája a makroassembler nyelv (l. a 9. fejezetet), amely csupán az assembler nyelv bővítésének tekinthető. Lehetőség van azonban blokkstruktúrájú nyelvek definiálására is „előre gyártott" makrokönyvtárak alkalmazásával (pl. BSAL), különböző alkalmazási területek igényeinek megfelelően [5].

Természetesen nem feladatunk a különböző magas szintű programnyelvek ismertetése, amelyek kézikönyvekben találhatók meg (pl. [10], [11], [12], [13], [14]). Csupán néhány, a mikroprocesszoros rendszerek alkalmazása szempontjából fontosabb olyan magas szintű programnyelv alapvető tulajdonságával foglalkozunk rö-

viden, amelyek bizonyos esetekben versenyképesek az assembler nyelvű programozással. A különböző mikroprocesszor-családok, mikroszámítógépek és mikroprocesszor-fejlesztői rendszerek sokféle magas szintű programnyelv alkalmazásához nyújtanak feltételeket, ill. fordítóprogramokat [5], [15].

10.1. A BASIC PROGRAMOZÁSI NYELV

A BASIC, amely nagyon könnyen megtanulható nyelv [17], volt az első olyan magas szintű programnyelv, amelyet a legtöbb mikroprocesszor-típusra implementáltak, így több tucat dialektusa létezik [16].

A nyelv általános célú, ami azt jelenti, hogy bár kevésbé hatékony, mint az ALGOL, a FORTRAN vagy a PL-1, jól használható olyan összetett feladatok esetén is, amelyek az ügyvitelgépesítés vagy folyamatszabályozás esetén jelentkeznek [18].

Minden BASIC sor egy sorszámmal kezdődik, amely egyrészt azonosítja a sort, másrészt meghatározza a végrehajtás sorrendjét is. A sorszámokkal lehet hivatkozni a logikai és a vezérlőutasítások megfelelő soraira. Tárhelyeket (azonosítókat), táblázatokat és állandókat szimbolikusan jelölhetnek betűkkel vagy egy betűvel és egy számmal. A kulcsszavak és a parancsok vagy emlékeztető szimbólumszerű rövidítések, vagy teljes angol szavak, amelyek egyes implementációkban csak a kezdőbetűjükkel is helyettesíthetők (rövid forma) [19].

A nyelv eredetileg párbeszédes alkalmazásra készült, azonban ma már nemcsak értelmező, hanem fordításos változatai is vannak. A BASIC szabad formátumú nyelv, ami azt jelenti, hogy egy soron belül nincs megadott formátum, amely szerint az utasításokat és a kiegészítő információkat írni kell. A programok beépített szerkesztőprogram-funkciókkal egyszerű módon módosíthatók. (Az újra megadott sorszámhoz rendelt tartalom az eredeti sorszámú sort felülírja). Ez azt is jelenti, hogy a sorok bármilyen sorrendbe írhatók, a végrehajtás sorrendjétől függetlenül.

Röviden összefoglaljuk egy adott implementáció fő tulajdonságait [20]. Természetesen a konkrét programozáshoz szükséges információt megfelelő kézikönyvekből szerezhetjük meg.

A parancsokat (command), amelyek előtt nem áll sorszám, a rendszer azonnal végrehajtja (a kocsi vissza karakter lenyomása után). Ezek közül bemutatunk néhány alap fontosságut:

LIST	kilistázza a programot,
SAVE	megőrzi a már tárolt programot,
OLD	visszahozza a régi programot,
NAME	új nevet ad az éppen tárolt programnak,
SCRATCH	törli a tárolt programot
RESEQ	a program sorai újra számozhatók.

Az utasításokat, a beépített és a felhasználó által definiálható függvényeket stb. a kulcsszavakkal (keyword) lehet megadni.

Azonosítóknak és állandóknak (amelyeket egy betű, vagy egy betű és azt követő szám jelezhet) értéketa következő utasításokkal lehet megadni:

- **A LET utasítással** (amely opcionális) a következő formában:
  sorszám LET X = kifejezés
  ahol X a változó, a kifejezés pedig lehet számérték, másik változó vagy aritmetikai kifejezés.
- **INPUT utasítással.** Ez esetben az adatot a terminálbillentyűzetről kell begépelni a megadott sorrendben, „?" megjelenése esetén. Ennek formája:

sorszám **INPUT** X1, X2, ... Xn

ahol X_1, X_2 ... a változók nevei (amelyeket vesszővel kell elválasztani).

■ **READ** utasítással, amely a **DATA** utasítással létrehozott adatmezőből rendel adatokat a változókhoz. Pl.:

sorszám **DATA** X1, X2, ... Xn esetén

a vesszővel elválasztott adatokat a megadott sorrendben a

sorszám **READ** X1, X2, ... Xn

utasítás hatására a változókhoz rendeli. Az egyes **DATA** utasításokat sorszámuk sorrendjében veszi, amíg minden változó megkapja az értékét. Egy következő **READ** utasítás esetében a hozzárendelést onnan kezdi, ahol az előbbiekben abbahagyta. (**RESTORE** utasítás hatására ehelyett ismét a **DATA** első adatától olvas.)

A BASIC nyelvben ötféle művelet van: összeadás (+), kivonás (—), szorzás (*), osztás (/), és hatványozás (* * vagy ↑). Az *aritmetikai műveletek* végrehajtási sorrendjét az ún. precedencia-szabály szabja meg. Eszerint elsődleges a hatványozás, másodlagos a szorzás és osztás, harmadlagos az összeadás és kivonás. Ha azonos precedenciájú műveleti jel van az operandus jobb és bal oldalán, akkor az operandus a tőle balra lévő műveleti jelhez fog tartozni. Ez az ún. balról jobbra szabály.

Két aritmetikai kifejezés összehasonlítására a következő relációjelek engedhetők meg: egyenlő (=), kisebb, mint (<), kisebb, vagy egyenlő (=< vagy <=), nagyobb, mint (>), nagyobb, vagy egyenlő (=> vagy >=), nem egyenlő (<>). A relációműveleteket a feltételes vezérlésátadó utasításokban használjuk. Ezek az **IF-THEN** vagy **IF-GOTO.** Ha a reláció (amely két számot, változót, ill. kifejezést hasonlít össze) teljesül, akkor az utasításban megadott sorszámú utasítással folytatódik.

sorszám **IF** reláció **THEN** n

sorszám **IF** reláció **GOTO** n

ahol *n* a feltétel teljesülése esetén végrehajtandó utasítás sorszáma.

A vezérlés a **GOTO** utasítás segítségével feltétel nélkül is átadható az *n* sorszámú utasításra.

sorszám **GOTO** *n*

A különböző BASIC implementációkban számos *beépített függvény* van, amelyeket külön definíció nélkül közvetlenül használhatunk. Ezek pl.: SIN(X), COS(X), ATN(X), EXP(X), LOG(X), SQR(X), ABS(X), INT(X), SGN(X), RND(X), amelyek színusz, koszinusz, arkusztangens, e^x, természetes logaritums, négyzetgyök, abszolút érték, egész rész, előjelképzés, véletlenszám képzését végzik. Az utóbbi *0* és *1* között állít elő ún. álvéletlenszámokat (pseudo-random number). Ha a RND(X) függvényhez tartozó utasítás előtt megjelenik a **RANDOMIZE** utasítás, akkor a program ismételt futtatáskor különböző eredményeket ad, mert az RND(X) függvény más kezdőértékekről indul. A felsoroltakon kívül a felhasználó maga is definiálhat függvényeket, amelyeket az **FN** betűket követő harmadik betűvel különböztethet meg egymástól. Ezeket a függvényeket a **DEF** (definíció) utasítással adhatja meg a következő formában:

sorszám **DEF FNA**(X) = kifejezés(X),

ahol az **A** helyett bármilyen betű állhat..

A programba magyarázó megjegyzéseket helyezhetünk el **REMARK** vagy **REM** utasítással.

A **FOR** ciklusnyitó és **NEXT** cikluszáró utasítások segítségével ún. ciklust szervezhetünk. Ekkor a ciklus belsejében (a **FOR** és a **NEXT** közötti sorokban) levő utasítások ismételten végrehajtásra kerülnek. A V ciklusváltozó X1 kezdőértéke a **TO** utasítással megadott X2 ciklusvégértékig, minden végigfutás kezdetén a **STEP** utasí-

tással megadott X3 lépésköznek megfelelően (amely negatív szám is lehet) megváltozik. Ciklust tehát a következő módon szervezhetünk a BASIC nyelvben:

sorszám **FOR** V = X1 TO X2 **STEP** X3
⋮
sorszám **NEXT** V

Szubrutinokat a **GOSUB** és **RETURN** utasítások segítségével használhatunk. A szubrutinra a vezérlés a **GOSUB** utasítással kerül:

sorszám **GOSUB** n

ahol *n* a szubrutin első utasításának a sorszámát jelenti. Itt a végrehajtás a sorszám szerint addig folytatódik, amíg a rendszer **RETURN** utasítást nem talál. Ekkor a vezérlés átadódik a **GOSUB** utasítást követő utasításra.

Használhatunk egy- vagy kétindexes változókat is, amelyeket tömbváltozóknak nevezünk. Ezeket **DIM** (dimenzió) segítségével deklarálni kell. Ennek formája a következő:

sorszám **DIM** V1(n_1), V2(n_2, m_2)

ahol V jelenti a tömbváltozó nevét, az n_1, n_2, m_2 egész számok pedig a tömb méretét határozzák meg. Vagyis, ha van nulla indexű elem is, egy (n_1+1) méretű vektort és egy (n_2+1, m_2+1) méretű mátrixot kapunk. Ha a tömbváltozókat nem definiáljuk a **DIM** utasítás segítségével, azokat a BASIC automatikusan tízes méretűekként értelmezi.

A korszerű BASIC implementációk karaktersorozatokat (string) is képesek kezelni. Ezek a karaktersorok betűkből, számjegyekből, betűközökből és egyes írásjelekből állhatnak. A kezelés stringállandók és stringváltozók segítségével történik. A stringállandó idézőjelek közé zárt karaktersorozat, pl. „1980". A stringváltozók karakterinformációt tárolnak, jelük a dollárjel (pl. A$). Egyszerű stringváltozó 8 karakternyi információt tud tárolni, míg a stringvektor maximálisan 72-t. (A „méretét" a **DIM** utasítással kell „dimenzionálni".) Az értékadás hasonló a normál változók értékadásához (az **INPUT**, a **READ** és a **DATA** utasításokkal). A karakterek ASCII kódjának numerikus értékét összehasonlíthatjuk az **IF ... GOTO** utasítások segítségével.

A numerikus változók és a karaktersorok adatmezője között az ún. stringkonvertáló függvényekkel létesíthetünk kapcsolatot. A LEN(A$) függvény megadja az A$ karaktereinek számát, az ASC(A$) függvény az ASCII kód decimális értékét, a CMR$(X) függvény pedig egy decimális számból ($0 \leqq X \leqq 63$) a megfelelő ASCII karaktert állítja elő. A VAL(A$) függvény egy karaktersorozatot számmá konvertál, az STR$(X) függvény pedig az argumentumában levő számot stringállandóvá alakítja át. A POS(X$, Y$, Z) függvény megadja, hogy az X$ Z-edik karaktere az Y$ hányadik karakterével egyezik meg először, az SEG$ (X$, Y, Z) függvény pedig megadja X$ karaktersorozat azon részét, amely az Y-adik és Z-edik karakterek között van.

A számítások eredményei és a karaktersorozatok kiírása a **PRINT** utasítással végezhető:

sorszám **PRINT** kifejezés,

ahol a kifejezés lehet állandó, változó szám, aritmetikai kifejezés, stringállandó, idézőjelbe tett karaktersorozat, TAB(X) tabulátor (ahol $1 \leqq X \leqq 72$ a sor elejétől számított karakterpozíció) vagy PNT(X) függvény (amelyet nem kinyomtatható karakterek, pl. csengő, kocsi vissza stb.) megadására használunk. Ha a változókat vesszővel választjuk el egymástól, a nyomtatás ún. zónázott formátumú lesz (egy sor 5 zónából áll, amelyekben max. 14 karakter fér el).

Nagy információmennyiségek kényelmesen file-rendszerekkel kezelhetők. Az összetartozó információmennyiséget tartalmazó file-hoz a programozó egy nevet rendel. A file hossza blokkokban vagy rekordokban van megadva, ahol pl. 1 blokk vagy rekord = 256 szó. Négyféle file-típus van, ezek: FILE (fix hosszúságú ASCII), FILEV (változó hosszúságú ASCII), FILEN (fix hosszúságú numerikus) és FILEVN (változó hosszúságú numerikus). A file kezelésére, használatára, részleteire itt nem térünk ki (l. a 8. fejezetet).

A korszerű BASIC programmegvalósítások lehetőséget adnak assembler szintű függvények beépítésére is az UDEF függvény segítségével, a következő módon:

sorszám UDEF függvénynév argumentum

A függvény (amelynek argumentuma maximálisan négy numerikus és két stringváltozó lehet [20]) definiálása után úgy használható, mint bármely más BASIC függvény.

A BASIC nyelv legfőbb előnye egyszerűségében van (amit az előbbi rövid ismertetés is jól jellemez) és így rendkívül gyorsan megvalósítható feladatmegoldást tesz lehetővé. A másik fő előnye, a párbeszédes forma, amely a lépésenként való programkészítés és kipróbálás hatékony eszköze. Ugyanakkor számtalan BASIC program található különféle feladatok megoldására, amelyeket közvetlenül, vagy kis átalakítással (a különböző BASIC dialektusok tulajdonságainak figyelembevételével) felhasználhatunk.

A legkorszerűbb BASIC változatok (pl. [19], [21]) lehetővé teszik gyors assembler nyelvű szubrutinok közvetlen hívását is. Pl. a **CALL** (N, E, F) segítségével egy olyan N számú, assembler nyelven megírt szubrutint hívhatunk, amely egy megadott F (file) számú és E (hibajelzés) számú adatmezőt olvas be a hajlékony mágneslemezes tárról az adatpufferbe. A BASIC program működését nagymértékben meggyorsíthatjuk, ha szoftver lebegőpontos műveleteket megvalósító programcsomag helyett ilyen műveleteket realizáló LSI hardver aritmetikai tokot (vagy mikroprogramozott aritmetikai egységet, l. a 11. fejezetet) alkalmazunk. A mikroprocesszoros alkalmazások szempontjából jelentős azonos időtartományban működő adatgyűjtési és vezérlési feladatok esetén a normál **PRINT** és **INPUT** utasításokhoz szükséges nagyobb idő megakadályozná a gyors működést. (A szokásos rendszerek esetén ez az idő 100 ... 200 ms karakterenként.) A **PRINT-AND-PROCEED** és az **INPUT-AND-PROCED** módosított utasítások nem zavarják a gyors programfutást, mivel csak a szabad (100...200 ms) időtartományokban írnak ki vagy olvasnak be egyszerre egy karaktert. Pl. a következő esetben az

INPUT [A] B, C$, D

esetén az egyetlen különbség a szokásos utasításhoz képest az A vezérlőváltozó. Ha A = *0*, az utasítás megfelel a normál **INPUT**-nak, míg ha A = *1*, az utasítás elindítja a bevitelt, majd átadja a vezérlést a következő utasításnak, amely a gyors vezérlő–adatgyűjtő programhurokban van.

A BASIC nyelv bonyolult feladatok (pl. egy adott mikroprocesszor számára egy keresztassembler programjának [22] vagy egy másik magasszintű programnyelv fordítóprogramjának [23] elkészítése) gyors és így jó hatásfokú megoldására is alkalmas. Az adott feladatok megoldása során a fordítóprogramokat sokkal hatékonyabbá tehetjük az ún. bootstrap (csizmahúzó) módszer alkalmazásával. Így a fordítóprogram írásához nem nagyon alkalmas BASIC nyelv korlátai csökkenthetők. Magát a bootstrap elvet talán majdnem mindenki ismeri a programbetöltő programokkal kapcsolatosan, amikor egy általános célra nem alkalmas egyszerű programbetöltő segítségével olvassuk be a tulajdonképpeni „fő programbetöltő programot". Először

csak a lefordítandó nyelv egy kis „részhalmazára" írunk egy egyszerű fordítóprogramot. Ez a részhalmaz azután már alkalmas arra, hogy ugyanazon nyelv most már nagyobb része számára írjunk fordítóprogramot. A nagyobb fordítóprogramot most már az első fordító segítségével fordíthatjuk le, ill. írása során ebben megvalósított utasításokat használhatunk fel. Így továbblépve a gyakorlatban általában három fokozatban valósíthatjuk meg a teljes fordítóprogramot, amelynek eredő hatásosságát nem nagyon fogja befolyásolni az első (BASIC-ban megírt) rész. Ily módon tehát a BASIC lényegében rendszerprogramnyelvként is használható.

10.2. A PLZ PROGRAMOZÁSI NYELV

Újabban egy másik magas szintű programozási nyelv is széles körben elterjedt, amelyet különböző cégek az IBM PL/1 alapján hoztak létre. Bár valamennyi változat a strukturált programozás [24] alapkoncepcióját követve a moduláris programozás megvalósítását tűzte ki célul, lehetővé téve valamilyen formában az assembler nyelvű rutinok alkalmazását is optimalizálás céljából, a létrehozott nyelvek (10.1. táblázat) egymással mégsem kompatibilisek.

10.1. TÁBLÁZAT. PL/1-szerű nyelvek

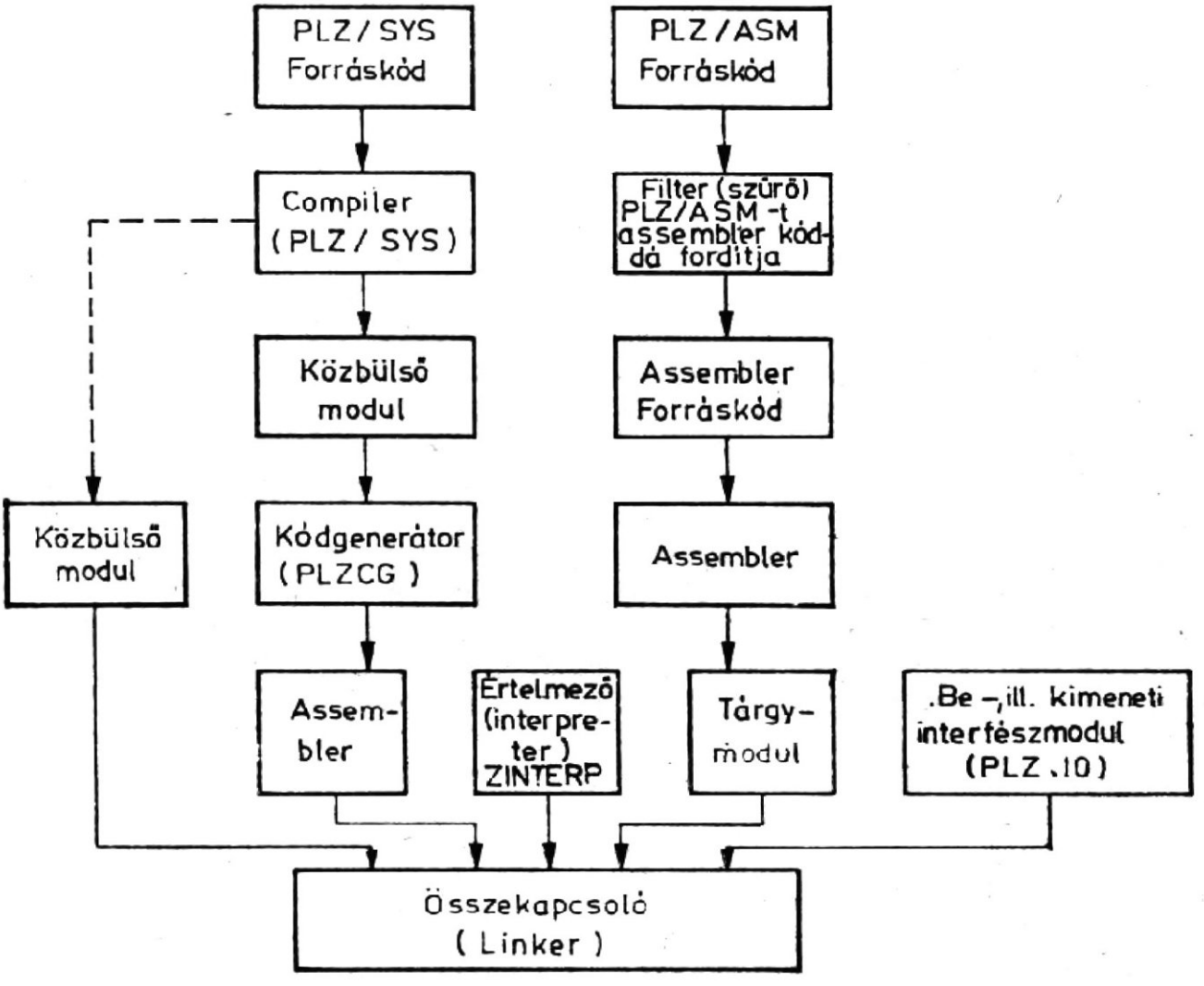
Nyelv neve	Támogatott processzor-típusok	Létrehozó forgalmazó	Megjegyzés
PL/M	8008 8080 8085 8086	Intel	Rezidens compiler (Intellec fejlesztőrendszerekben, ISIS–II operációs rendszer alatt fut)
PLZ (PLZ/SYS és PLZ/ASM)	Z8 Z80 Z8000	Zilog	Rezidens compiler (MCZ sorozatú mikroszámítógépek és ZDS sorozatú fejlesztőrendszerekben a RIO operációs rendszer alatt fut) PLZ/SYS: rendszerprogramnyelv PLZ/ASM: strukturált assembler nyelv
PL/65	6500	Rockwell	Rezidens compiler (assembler nyelvű programot szolgáltat a System 65 assembler számára). Keresztcompiler és assembler is van a PDP–11 gépcsaládra (az RT–11 operációs rendszer alatt fut)
MPL	6800 6809 68000	Motorola	Rezidens compiler (Exorcizer fejlesztőrendszerben) Assembler nyelvű kódot szolgáltat
SMPL	IMP–16 PACE	National Semiconductor	
PL μS	2650	Signetics	

Röviden bemutatjuk a PLZ programnyelvet [12], [25], [26], amely lényegében több, összehangolt tulajdonságú programozási nyelvet tartalmazó család.

A moduláris programozást a PLZ programok felépítése teszi lehetővé, mivel egy PLZ program modulokból áll, amelyek egymástól függetlenül is fordíthatók. Egy modul adatdeklarációt és végrehajtható részeket: ún. eljárásokat (procedure) tartalmaz. A moduláris felépítés ugyanakkor elősegíti az átfedéses (overlay) technika megvalósítását. Ez lehetővé teszi, hogy egyes programrészek, amelyek végrehajtás alatt nem szükséges, hogy állandóan a főtárban legyenek, azonos főtártartományt használjanak fel.

A továbbiakban a PLZ család két tagját, a PLZ/SYS magas szintű rendszerprogramozási nyelvet és a PLZ/ASM magas szintű assemblert mutatjuk be. (Utóbbi azonban szintén egy strukturált assembler nyelvet valósít meg, számos PLZ/SYS vezérlési funkció felhasználásával).

A 10.3. ábrán bemutatjuk a PLZ család különböző programjai közötti kapcsolatot [5], [12], [25]. A PLZ/SYS forráskódot az adott RIO operációs rendszer alatt futtatható szerkesztőprogram segítségével hozhatjuk létre (l. a 9. fejezetet), kialakítva ezen a módon egy – a forrásnyelvű programot tartalmazó – file-t. (A file névnek S névkiegészítővel kell végződnie.) A fordítóprogram ebből egy ún. programlista-file-t hoz létre (a forrással azonos néven csak L kiegészítővel), és egy közbenső kód (ún. Z-kód) formában lefordított tárgyfile-t (Z névkiegészítővel). Ez azután kétféle módon alakítható tovább, vagy egy értelmezőprogrammal (ZINTERP interpreter)



10.3. ÁBRA. A PLZ család

közvetlenül, vagy a PLZCG kódgenerátor-programmal. A kódgenerátor áthelyezhető assembler nyelvű programot készít, amelyet – mielőtt végrehajtható programot kapnánk – le kell fordítani az assembler fordítóval, és a csatolóprogram (PLINK linker) segítségével összekapcsolni más modulokkal. A PLINK össze tudja kapcsolni egymással az assembler fordító által előállított gépi kódot és a kompilátor által előállított Z közbenső kódot.

A 10.3. ábra másik „ága" szerint az assembler nyelven írt forrásprogram először egy szűrőprogramon halad át, amely lényegében egy előfeldolgozó program. Erre azért van szükség, mert – ahogy ez nagyon gyakori – az eredeti assembler nyelvet később a moduláris programozáshoz, logikai adatstruktúrák létrehozásához szükséges eszközökkel továbbfejlesztették. Az ily módon létrehozott forrásfile-t lehet az assembler fordítóval lefordítani, létrehozva egy tárgyprogramfile-t (és egy listafile-t). Ez a tárgymodul most már a csatolóprogram segítségével hozzákapcsolható más programokhoz. A be-, ill. kimeneti (I/O) kapcsolatot létrehozó modul a RIO operációs rendszer adatátviteli eljárását teszi használhatóvá (l. a 8. fejezetet). Pl. az áramlás mód (stream) tetszőleges számú byte-ból álló file mozgatását végzi a file-kijelölő (cursor) által kijelölt helyekre. A RIO operációs rendszer tárkezelő része a file megnyitásakor automatikusan kijelöl egy szükséges kapacitású puffert, ezzel „átlátszóvá" teszi a felhasználó számára bizonyos fizikai eszközöknek (pl. mágneslemezes tár) azt a tulajdonságát, hogy adott nagyságú rekordokat képesek egyszerre átvinni. Ennek a modulnak a feladata a PLZ programok és a RIO operációs rendszer között az illesztés, amely elsősorban a programparaméterek átadásának megoldásán keresztül valósul meg. (Erre a proglémára itt nem térünk ki, hanem utalunk [26]-ra, ahol a programok konkrét használatához szükséges parancsokat is részletesen megtaláljuk.)

Mint minden algoritmus vagy számítógépprogram, a PLZ nyelvű program is két fő részből áll. A végrehajtandó műveletek leírása, és az ezen műveletek kapcsán felhasznált adatok. Általában a definíció egy azonosítót specifikál, amely megadott értékű vagy típusú. A deklaráció egy olyan azonosítót vezet be, amely egy változót jelöl és egy hozzárendelt típust is. Egy adattípus implicite definiál egy értékkészletet és azokat a műveleteket is, amelyeket e készlet elemein végrehajthatunk (l. később). Az adattípus vagy a változódeklarációban, vagy egy külön típusazonosítóra való hivatkozással van leírva.

Minden adatnak értéket kell adni. Az utasításban előforduló értéket vagy egy literális állandó (l. később), vagy egy azonosító, amelyet változóként kell deklarálni, vagy egy értékeket tartalmazó kifejezés adhatja meg. Minden változó azonosítóját deklarálni kell a megfelelő adattípussal együtt (és esetenként az azonosító kezdőértékével is). Minden állandóazonosító az állandónak a definíciójában kell, hogy megjelenjen, amely az azonosítóhoz egy értéket rendel.

A PLZ adattípusai lehetnek *egyszerű típusúak*. Ez lehet szabványos vagy felhasználó által definiált. A szabványos egyszerű adattípusok a következők: byte (BYTE); szó (WORD), amelyek 8 és 16 bites előjel nélküli adatoknak felelnek meg; egész (INTEGER) és rövid egész (SHORT INTEGER), amelyek 16 és 8 bit hosszúságú előjeles értékeket jelölnek; valamint mutató, amely a megvalósítástól függő címértéket jelent.

Az ún. *strukturált* típusok esetén viszont a definícióban kell leírni az elemek típusát és az összekapcsolás módját is. Tömb (array) esetén minden elem azonos típusú és a kiválasztásuk egy vagy több tömbindex vagy tömbszelektor segítségével történik. Rekord esetén annak elemei (ezeket mezőknek is nevezik) nem szükségszerűen azonos típusúak. A rekordszelektor nem kiszámítható érték, hanem egy azonosító azért, hogy a kiválasztott elem típusa nyilvánvaló legyen a program végre-

hajtása nélkül is. Ezeket a mezőazonosítókat a rekordtípusú definícióban kell deklarálni.

Azokat a változókat, amelyeket közvetlen formában deklarálunk, közvetlen (direkt) változóknak nevezzük, mivel a deklaráció a változóhoz egy azonosítót rendel, amellyel erre a változóra hivatkozni lehet. Ezzel szemben változókhoz közvetve is hozzáférhetünk mutatóváltozókon keresztül.

A legalapvetőbb utasításokat egyszerű utasításoknak nevezik. Az értékadó utasítás egy változóhoz (vagy egy változó egy eleméhez) egy kifejezés kiértékelésével kapott új értéket rendel. A kifejezések olyan változókat, állandókat, generátorokat és eljárásokat (l. később) tartalmazhatnak, amelyek egyetlen értéket adnak. Az operátorok az operandusokból eredményt hoznak létre. Ezek a következők:

- Aritmetikai operátorok: összeadás (+); kivonás (−); szorzás (*); osztás kerekítéssel (/); modulus vagyis a maradék kiszámítása (MOD); abszolút érték (ABS); növelés (INC); csökkentés (DEC); tömb elemeinek a száma (ASIZ); változó címe (#).
- Logikai operátorok: logikai ÉS (AND); logikai VAGY (OR); logikai komplementálás (NOT); logikai kizáró VAGY (XOR).
- Relációoperátorok: egyenlő (=); nem egyenlő (<>); kisebb, mint (<); nagyobb, mint (>); kisebb, mint vagy egyenlő (<=); nagyobb, mint vagy egyenlő (>=).
- Az értékadó utasítás formája:
  változó := kifejezés
  Pl.: TERULET := SZEL * HOSSZ
- Az elfogadásutasítás hasonló az értékadóutasításhoz, azzal a különbséggel, hogy egyik értéktípusról a másikra való átalakítást is lehetővé tesz, vagyis a kifejezés típusa fogja a változó típusát megszabni.
  Az elfogadásutasítás formája:
  változó **ACCEPTS** kifejezés
- Az eljárásutasítás az értékadást oly módon végzi, hogy a megjelölt eljárást (l később) végrehajtja és az ebből kapott értéket rendeli a változóhoz.
  Az eljárásutasítás formája:
  változó := eljárásazonosító (aktuális paraméterlista).
  Pl.: NAGY := MAX (X, Y).
- A **RETURN** utasítás hatására az eljárás végrehajtása ezen utasítás helyén befejeződik és az eljárás hívását követő utasításon folytatódik.
- Az **EXIT** utasítás a kétféle programhurokvezérlő utasítások közül az egyik, amely lezárja egy hurok végrehajtását.
- A **REPEAT** utasítás a hurok elejére adja a vezérlést. Ha ez szükséges, pl. egymásba skatulyázott program hurkok esetén, mindkét utasítás kiegészíthető címre utaló címkeazonosítóval is.
  Formájuk ebben az esetben a következő:
  EXIT FROM címkeazonosító,
  REPEAT FROM címkeazonosító.
- Összetett utasítások az egyszerű vagy összetett utasítások sorozatának feltételes, kiválasztott, ismételt végrehajtását teszik lehetővé.
- Feltételes utasításban az **IF**-et követő kifejezés értékelése dönt a **THEN** és **ELSE** kulcsszó közötti, ill. az **ELSE** és **FI** (fordított **IF**), más szóval „**IF** zárójel" közötti utasítások végrehajtásáról. (Az **ELSE** el is maradhat.) Ha a kifejezés kiértékelésekor IGAZ (nem zérus) értéket kapunk, az első utasítás, ha HAMIS (zérus), akkor a második utasítás kerül végrehajtásra.

Feltételes utasítás formája:

IF kifejezés **THEN** utasítás [**ELSE** utasítás] **FI**.

Itt van az egyetlen olyan eset, amikor egy kifejezés relációoperátort is tartalmazhat, mivel ennek kiértékelése is vagy IGAZ vagy HAMIS eredményt ad.

A feltételek egymásba is skatulyázhatók, mivel összetett utasítás, azaz feltételes utasítás is lehet az utasítás helyén.

Pl.
```
IF i <= 5
THEN PROCESS (1)
ELSE IF i <= 8
  THEN PROCESS (2)
  ELSE IF i <= 13
    THEN PROCESS (3)
    ELSE PROCESS (4)
    FI
  FI
FI
```

■ A kiválasztóutasításban a kiválasztást végző kifejezés értékétől függően választhatjuk meg a végrehajtandó utasítást.

Az **IF** után a kiválasztást végző kifejezés következik, majd a **CASE** után ennek értékei (a kiválasztóelemek) és ezt követő **THEN** után az ezekhez tartozó utasítások.

```
IF kiválasztó kifejezés (selector).
  CASE kiválasztóelem (1) THEN utasítás (1)
  CASE kiválasztóelem (2) THEN utasítás (2)
      ⋮
  ELSE utasítás
FI
```

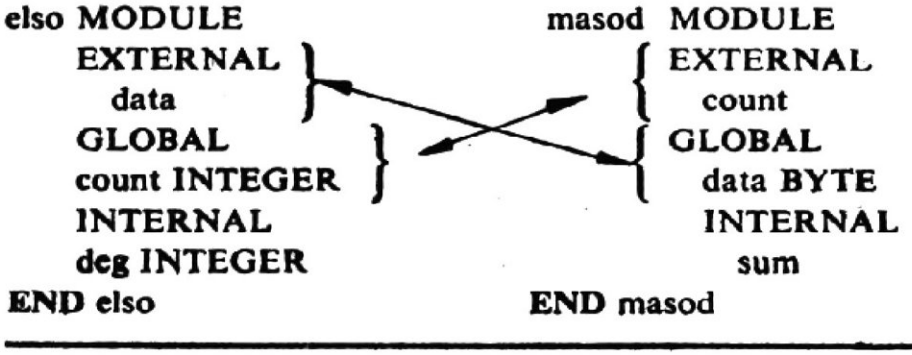
■ A hurokutasítás az ismételt végrehajtást vezérli. A **DO**-tól az **OD**-ig (fordított **DO**) tartó utasítások ismételten végrehajtásra kerülnek, amíg a végrehajtás menetét egy (általában feltételes) **EXIT, REPEAT** vagy **RETURN** utasítás meg nem szakítja.

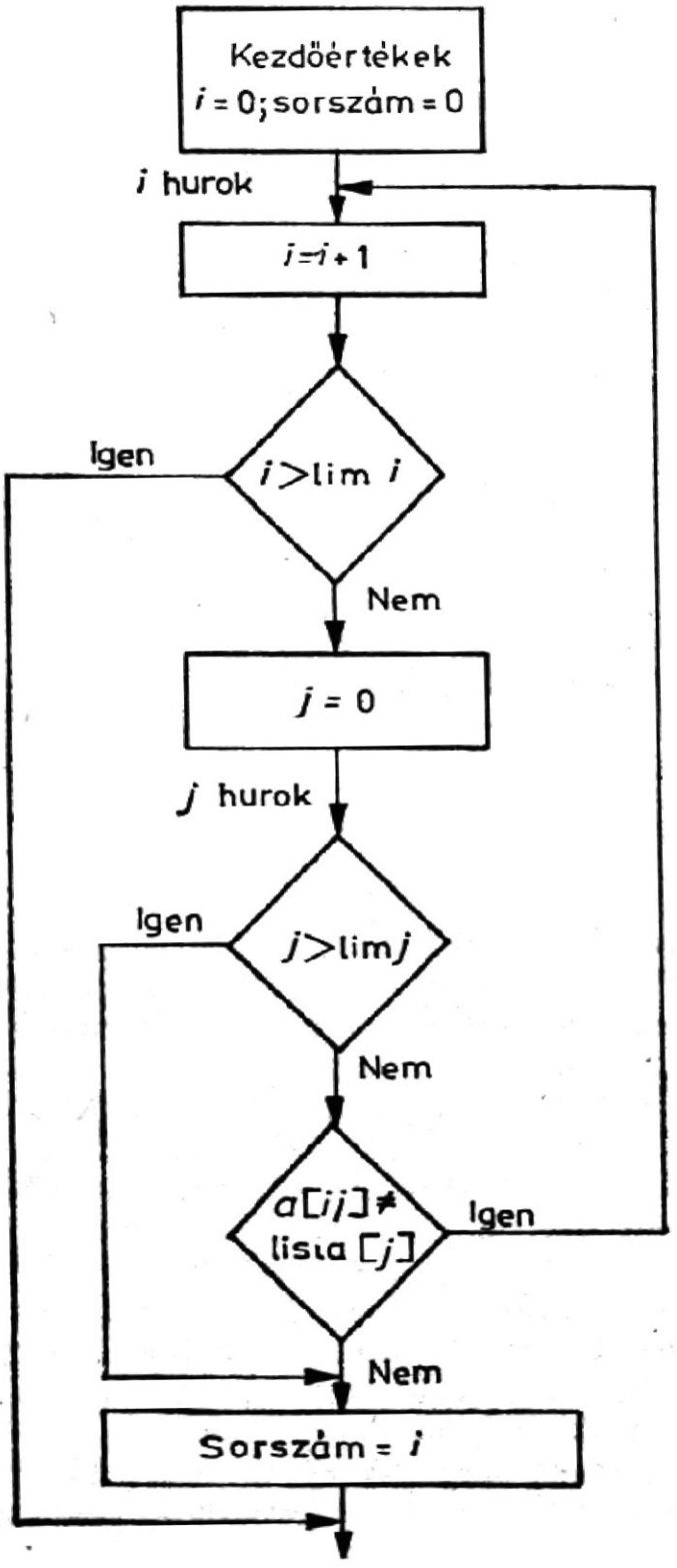
Pl. a 10.4. ábrán megadott folyamatábra szerinti programrészlet a következő lesz:

```
  i := 0   sorszám := 0
iloop:
  DO
    i := i+1
    IF i > limi
         THEN EXIT
    FI
    j := 0
    jloop:
    DO
      j := j + 1
      IF j > limj
         THEN EXIT
      FI
      IF a [i, j] <> lista [j]
         THEN REPEAT FROM iloop
      FI
    OD
  sorszám := i
  EXIT
OD
```

Egy program egy vagy több önállóan fordítható részből: modulból áll. Egy modul végrehajtható részét, amelyet egy névvel is azonosíthatunk, eljárásnak nevezik és ennek megadása az eljárás deklarációja. Az eljárásfej adja meg az eljárás nevét, valamint az azonosítók formális paramétereit (bevitt „in" paraméterek), és azokat az azonosítókat, amelyek értékeit az eljárásnak vissza kell adnia (kiadott „out" paraméterek, más néven RETURNS lista). LOCAL változók, amelyek nincsenek deklarálva helyiek, és csak az adott eljárásban, annak az ENTRY és END közé eső törzsében használhatók. Egy változó az egész modulban használható, ha explicit formában szerepel egy GLOBAL vagy INTERNAL deklarációban vagy az adott modulba EXTERNAL deklarációval került. Ennek megfelelően az EXTERNAL deklaráció azokat a változókat tartalmazza, amelyeket más modulokban definiáltak, a GLOBAL változó- vagy eljárásdeklaráció azokat, amelyeket majd más modulokban is használni akarunk, míg az INTERNAL a változóknak csak a deklaráció helyén való használatát engedi meg. Az egyes változók deklarációját, ill. használatát a 10.2. táblázatban mutatjuk be, ahol a típus megadásmódját is láthatjuk.

10.2. TÁBLÁZAT. Változók deklarációja és használata





10.4. ÁBRA. Programhurok folyamatábrája

Az eljárás formális paramétereinek értéket kell adni, ezek az aktuális paraméterek. Paramétereket az eljárásnak átadni csak érték formájában lehet (ez az eljárás által kibocsátott paramétereknél is igaz).

Az eljárás alakja tehát a következő:

eljárásazonosító PROCEDURE
formális paraméterek listája
RETURNS formális paraméterek listája
LOCAL az eljáráson belül deklarált változók
ENTRY
utasítások
END eljárásazonosító

A PLZ/ASM assembler nyelv magas szintű vezérlőutasítások (feltételes elágaztató, hurokutasítások) alkalmazását teszi lehetővé az adott mikroprocesszor (esetünkben Z80 [26]) assembler utasításain felül. Így sokkal hatékonyabb a programkészítés, áttekinthetőbb a program, az eredmény pedig megegyezik az „egyszerű" assembler nyelven írt programmal. Az eddigi vázlatos ismertetés kiegészítéseként a 10.3. és 10.4. táblázatban két mintaprogramot mutatunk be. Mindkettő azonos feladatot old meg, egy adott számsort növekvő sorrendbe rendez, de az egyik PLZ-SYS nyelven, a másik pedig PLZ/ASM nyelven készült [25].

10.3. A PASCAL PROGRAMOZÁSI NYELV

Az utóbbi időben a PASCAL programozási nyelv a mikroprocesszoros rendszerek területén is rendkívül népszerűvé vált [32].

E programnyelv nagyfokú és növekvő népszerűsége valóban kiemelkedő. Sokak szerint ez egyedülálló tulajdonságaiban rejlik, amelynek következtében mikroprocesszoros rendszerekben is egyaránt jól használható mind rendszer-, mind alkalmazói programozáshoz. Ez prof. Wirth híres, a PASCAL tulajdonságaival foglalkozó könyve [34] címében is tükröződik:

Algoritmusok+Adatstruktúrák = Programok.

A PASCAL tehát mind az adatstruktúrák definiálásában, mind a strukturált programozáshoz szükséges vezérlőstruktúrák megvalósításában talán a legjobb magas szintű programozási nyelv.

Előnyei közül megemlíthetjük [35], hogy

- szinte kényszeríti a programozót, hogy jó hatásfokú (kis tárkapacitás-igényű, gyors futású, nagy műveleti sebességű) kódot írjon,
- a programdokumentáció maga a program (self-documenting: öndokumentáló),
- megkönnyíti a nagy programok felülről lefelé [28] való strukturált felbontását.

A PASCAL-ban írt programok végrehajtási sebességére tájékoztatásul a következőkben megadjuk egy adott fordítóprogram és tipikus feladatcsoport segítségével nyert adatokat [36]. A PASCAL-ban írt változat kb. 16-szor gyorsabban futott, mint a BASIC-ben írt, és 4-szer gyorsabban, mint a FORTRAN-ban megírt azonos program. Az assembler nyelven írt változat is alig kétszer volt gyorsabb a PASCAL-ban írtnál.

A PASCAL használhatóságát, különösen a programok átvihetőségét egyik rendszerről a másikra nagymértékben elősegítette, hogy egy közbenső absztrakt, veremtár-orientált nyelvet, az ún. P-kódot is definiálták. Először erre a formára fordítanak (azonos vagy más számítógép-rendszereken, vagyis rezidens vagy kereszt módon), majd a P-kód dolgozható fel különböző módon. A 10.5. ábrán négy lehet-

```
bubble sort MODULE
  CONSTANT
    true:=1
    false:=0
  INTERNAL
    y ARRAY [10 INTEGER] !rendezetlen tömb!
      :=(33 10 2000 400 410 3 3 33 500 199)
    sort PROCEDURE (n BYTE) ! n a csoportosítandó elemszám!
  LOCAL
    i j limit BYTE              ! tömb indexei!
    temp INTEGER                ! átmeneti tároló!
    switched BYTE               ! befejezést jelzi!
  ENTRY
    limit:=n-2                  ! indexmegállási pont!
    DO
      switched:=false           ! jelzés kezdőértékének beállítása, nincs változás!
      i:=0                      ! kezdés az első tömb elemnél!
    DO
      IF i>limit                ! kilépés, ha a teljes tömbön végighaladt!
        THEN EXIT
      FI
      j:=i+1                    ! szomszédos elem indexe!
      IF y[i]>y[j]              ! szomszédos elemek összehasonlítása!
        THEN
          temp:=y[i]            ! elemek megcserélése!
          y[i]:=y[j]
          y[j]:=temp
          switched:=true        ! jelző kapcsoló!
      FI
      i:=i+1                    ! következő pár!
    OD
      IF NOT switched           ! visszatér, ha nincs változás!
        THEN RETURN
      FI
    OD
END sort
main PROCEDURE                  ! fő eljárás, a „sort" kivéve n=10 aktuális értékkel!
  ENTRY
    sort (10)
    RETURN
  END main
END bubble sort
```

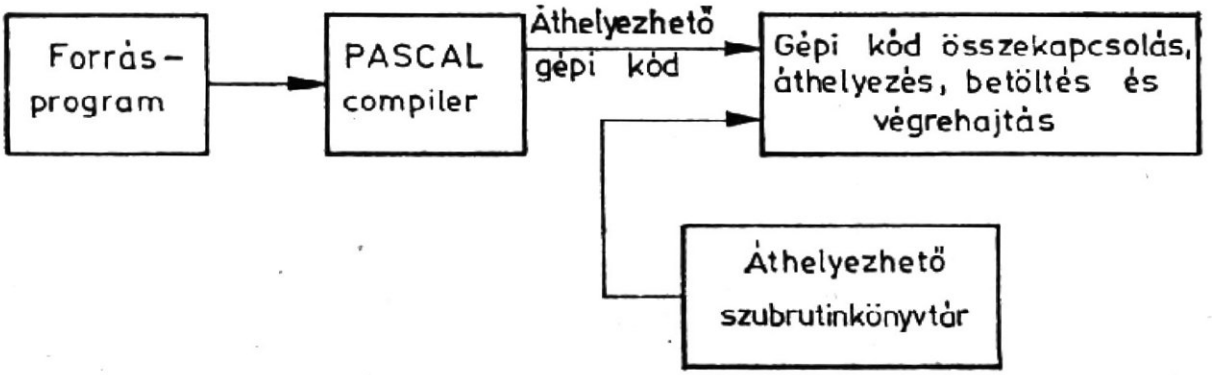
```
bubble sort MODULE
  CONSTANT
  flag:=0                    ! kapcsolójelzőbit pozíciója!
  INTERNAL
    y ARRAY [10 BYTE]
      :=(33 10 200 40 41 3 3 33 50 199)
    sort PROCEDURE
  ENTRY
    DO
      res flag, c            ! szemafor :=hamis!
      ld ix, y               ! i=0 indexű y[i] címe!
      ld d, b
      dec d                  ! limit :=n-1!
      DO
        IF ZERO              ! i>limit vizsgálat!
          THEN EXIT
        FI
        ld e, (ix)
        ld a, (ix+1)
        cp e                 ! y[i]>y[i+1] vizsgálat!
        IF CARRY
          THEN
            ld (ix), a       ! megcserélés!
            ld (ix+1), e     ! y[i] és y [i+1]!
            set flag, c      ! szemafor := igaz!
        FI
        inc ix               ! i:=i+1!
        dec d
      OD
      bit flag, c
      IF ZERO                ! szemaforvizsgálat!
        THEN RETURN
      FI
    OD
  END sort
  main PROCEDURE
    ENTRY
      ld b, 10               ! n:=10!
      sort
      RETURN
    END main
  END bubble sort
```

séges megoldást tüntettünk fel. Az *a* ábra szerint a fordítás közvetlenül gépi kódra történik, a *b* szerint a P-kódot értelmező program dolgozza fel, a *c* ábra szerint a P-fordító egy utófeldolgozóként működik, míg a *d* szerinti megoldás esetén a mikroprocesszor a P-kódot hajtja végre közvetlenül (az utóbbira példa a mikroprogramozott Western Digital PASCAL Microengine speciális áramköri készlet [8]). A módszereket a PASCAL program fejlesztésénél kombinálva is használhatjuk. A program javításának munkáját ugyanis sokkal hatásosabbá tehetjük, ha a fordítási hibákat először P-kódban javítjuk ki, majd a programot az értelmezőprogram segítségével „lőjjük be" és csak azután hozzuk létre a fordító segítségével a gyorsabban futó változatot [23]. Ezen eljárás szerinti programfejlesztés menetének a folyamatát a 10.6. ábrán mutatjuk be.

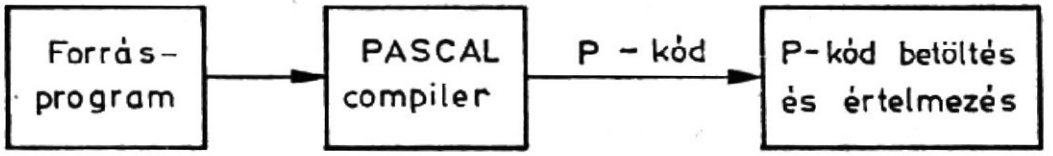
A PASCAL nyelv részletes ismertetése meghaladja könyvünk lehetőségeit, de célkitűzéseit is. Az alapvető ismertetések [13], [14] mellett ezért az érdeklődők figyelmét felhívjuk a sok bevezető tankönyvre is (pl. [35], [37]). Itt csak néhány jellemző tulajdonságot emelünk ki röviden, és a nyelv alapvető tulajdonságait foglaljuk össze.

A PASCAL programok moduláris felépítése, blokkstruktúrája hasonló a PLZ programok felépítéséhez. (Valóságban a helyzet „fordított", mivel a PLZ készült a PASCAL alapján.) Az alapelv itt is az egyes részprogramok nem szükséges jellemzőinek „elfedése" vagy más szóval „átlátszóvá" tevése, ami azt jelenti, hogy az eljárások azon azonosítóinak (változó, függvény, címke stb.), amelyek csak az adott részprogramban fordulnak elő, az alprogramon kívül nincs jelentőségük. Más azonosítók azonban általános (globális) értelmezéssel egyszerűvé teszik az egyes eljárások közötti kapcsolat megkeresését, a programrészek összekapcsolását. A 10.5. táblázatban foglaltuk össze a PASCAL nyelv (a mikroprocesszoros rendszernél legelterjedtebb UCSD: University of California at San Diego, azaz a kaliforniai egyetem San Diego-ban változatának) alapelemeit. Az adattípusokat a 10.7. ábrán külön is bemutatjuk. A standard skaláris adattípusokon kívül egy előzőleg deklarált adathalmaz alcsoportjai is megadhatók, ill. a felhasználó által is bővíthető az adattípusok köre. A strukturált adattípusok közül a rekord a tömb olyan általánosítása, amely a struktúra mezőiként megenged különböző típusú elemeket is. Az egyes mezőkhöz a rekordon belül az ún. adatútnevek segítségével férhetünk hozzá, amelyek definiálják a hozzájuk rendelt mezőket. A készlet a bitsorozatok koncepciójának emlékeztető szimbólumokkal való megvalósítása. Az adott változónévhez tartozó érték bináris nullákat és egyeseket tartalmaz attól függően, hogy az illető bitpozícióhoz tartozó elem része-e az adott készletnek. A file a PASCAL-ban korlátozott értelmű, és csak homogén elemek egyértelműen adott sorrendű gyűjteménye. Így egyszerre csak egyetlen elemet, az ún. pufferváltozót használhatja a felhasználó. Az egymással összekapcsolt adatstruktúrák (listák, fák, gráfok stb.) esetén a mutató feladata az elemek közötti kapcsolat megvalósítása, amelyekkel érdekes összetett adatstruktúrák hozhatók létre [39].

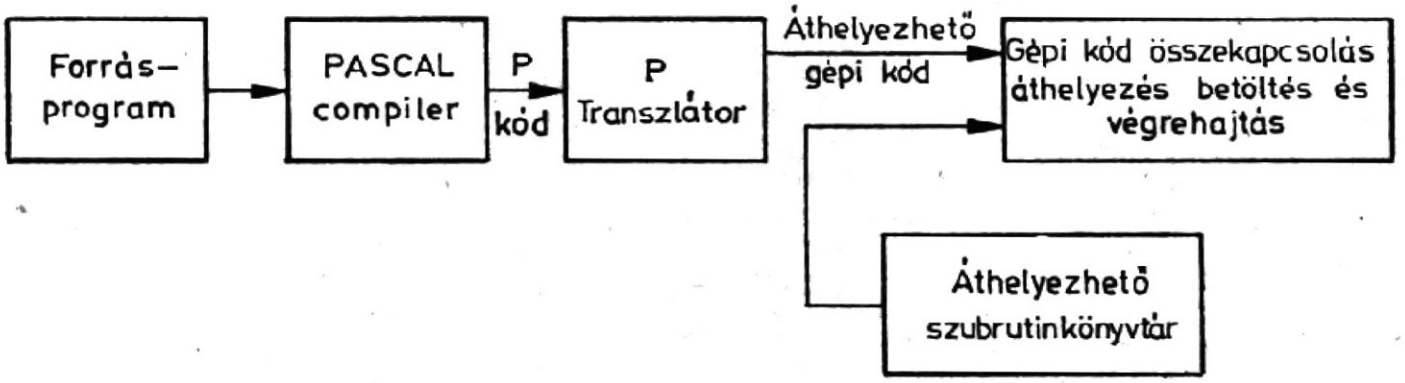
A „hagyományos" magas szintű programozási nyelvek (FORTRAN, COBOL stb.) mellett egyre újabb ígéretes nyelvek (pl. C nyelv [11], Forth [29]) is feltűnnek. Ezek elterjedéséhez (vagy esetleg eltűnéséhez) még hosszabb időre van szükség, így ulajdonképpeni jelentőségüket még nem lehet objektíven megítélni.



a)



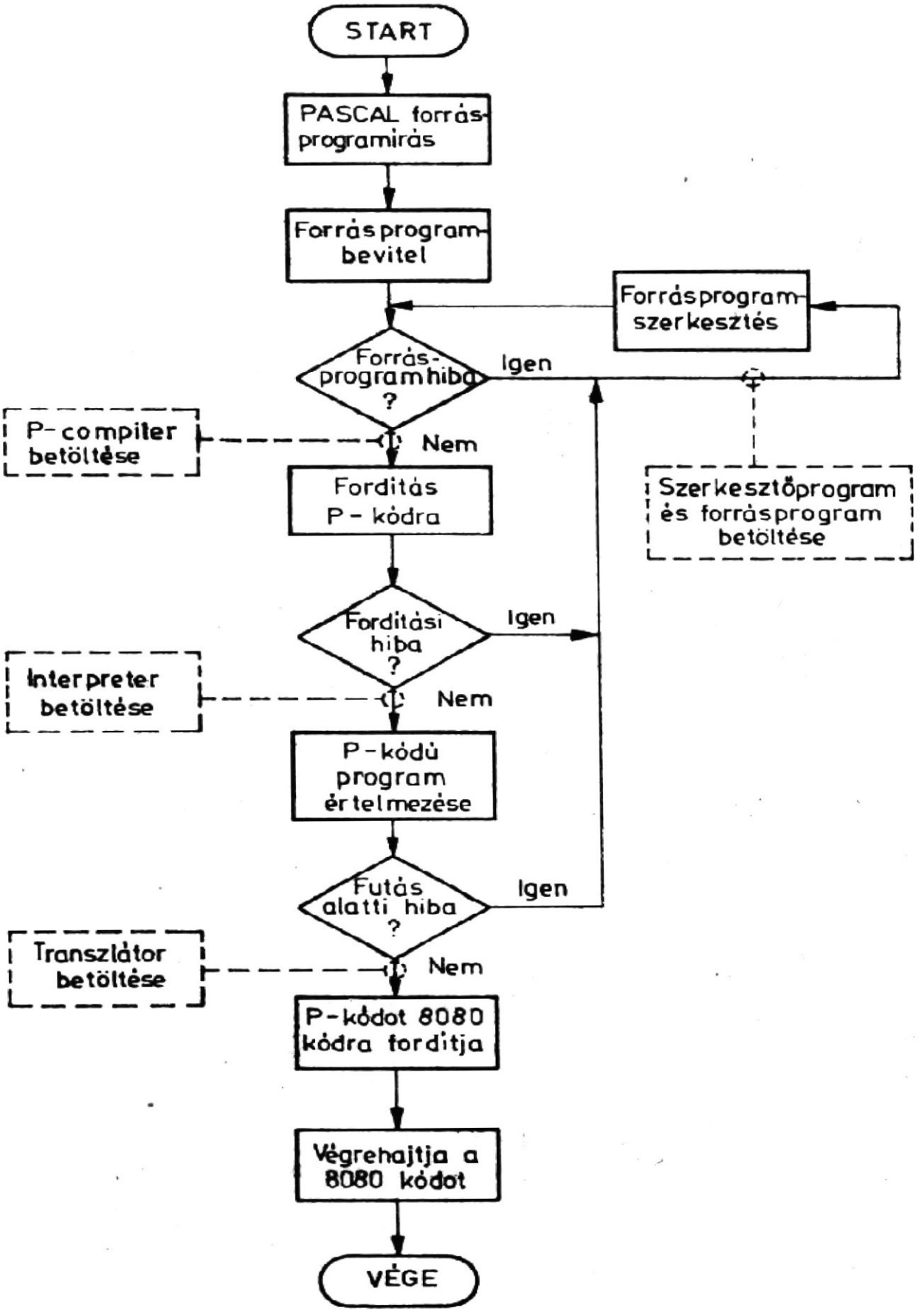
b)



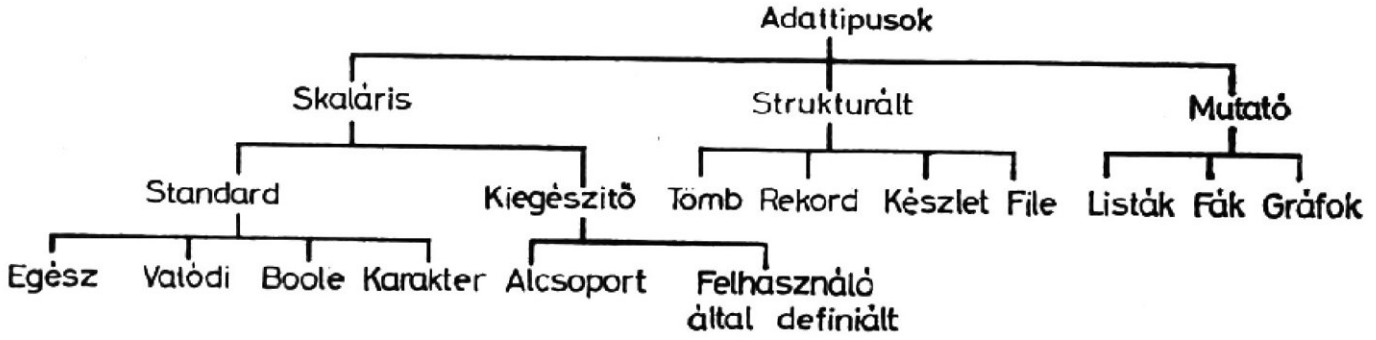
c)

d)

10.5. ÁBRA. A PASCAL P-kód feldolgozásának módszerei



10.6. ÁBRA. PASCAL nyelvű program fejlesztésének folyamata



10.7. ÁBRA. PASCAL adattípusok

10.5. TÁBLÁZAT. A PASCAL nyelv alapelemei

Adatmegadás (Data Specification)

Állandók (Constants)

CONST név 1 = érték 1 Pl. Pi = 3,14159

.

.

.

név n = érték n

Típusok (Types)

Skaláris (scalar) típusok

Standard típusok	**Például**
INTEGER (egész)	32,767
REAL (valódi)	27.9E–15
BOOLEAN (Boole)	TRUE (igaz) FALSE (hamis)
CHAR (karakter)	ASCII karakter
STRING [N]	N db ASCII karakter

Kiegészítő típusok

Felhasználó által definiált	TYPE név = értéklista TYPE DAY = (MON, TUES, WED, THUR, FRI, SAT, SUN)
Alcsoport (subrange) típus	TYPE név = legalsó érték... legfelső érték TYPE WEEKDAY=MON...FRI WEEKEND=SAT...SUN

Strukturált típusok

ARRAY (tömb)

RECORD (rekord)

SET (készlet)

FILE (file)

Mutató (pointer)

POINTER

Változók (variables)

VAR változónév: típus

Adat szervezés (Data Organization)

Tömbök (arrays)

változónév: ARRAY [indexváltozó-lista] OF típus

Készletek (sets)

változónév: SETOF típus

10.5. TÁBLÁZAT (1. folytatás)

Rekordok (records)
változónév: RECORD
változónév 1:típus 1
.
.
.
változónév n:típus n
END (rekordnév)
File-k (files)
változónév: FILE OF típus

Adatkezelés (Data Manipulation)

Aritmetikai (Arithmetric)
+, –, *,/ Real vagy Integer változók esetén az eredmény Real
DIN, MOD Integer változók Integer eredményt adnak

Logikai (Logical)
Reláció operátorok

=, <>, >, >=, <, <=	Két aritmetikai kifejezés között (akár real, akár integer) az eredmény Boole-kifejezés
Kombinatorikai operátorok	
AND, OR	Két Boole-kifejezés, amely Boole-eredményt ad
Negáció operátor	
NOT	Boole-kifejezés inverze

Hozzárendelés (assignment)

változónév:=kifejezés	A kifejezésnek egy olyan típusú értéket kell adnia, amely megegyezik a változónévre specifikálttal

Összetett struktúrához való hozzáférés
WITH rekordnév **DO** utasítás

Készletkezelés (sets manipulation)	Készletekre alkalmazva készletet eredményez
+: egyesítés (union)	
*: metszet (intersection)	
–: különbség (difference)	
Relációk (relational)	Készletekre vagy elemeikre alkalmazva Boole eredményt szolgáltat
=, <>	Egyenlőség-(egyenlőtlenség-) vizsgálat
<=, >=	Beletartozás vizsgálat
IN	Elemvizsgálat, hogy tagja-e a készletnek

Végrehajtás vezérlés (Execution Control)

Feltételes végrehajtás (Conditional Execution)
Egy lehetőség:
IF feltétel **THEN** utasítás
Két lehetőség:
IF feltétel **THEN** utasítás **ELSE** utasítás

Több lehetőség:
CASE kifejezés **OF**
állandó 1: utasítás 1
.
.
.
állandó n: utasítás n
END
Ismételt végrehajtás (Iterative Execution)
Előre adott feltétel (Pre-Conditioned)
WHILE feltétel **DO** utasítás
Utófeltétel (Post-Conditioned)
REPEAT (hurok név) .
utasítások
UNTIL feltétel
Indexezett (Indexed)
FOR index változó := kezdeti **TO/DOWNTO** végső **DO**
BEGIN (huroknév)
utasítás 1
.
.
.
utasítás n
END (hurok név)
Közvetlen vezérlésátadás (Direct Transfer)

COTO címke	Az adott eljáráson belül
EXIT eljárásnév	Az adott eljárást (és azon belül minden eljárást) befejezi és kilép belőle

10.4. A SZIMULÁCIÓ ALKALMAZÁSAI

A szimuláció egy olyan módszer, amely lehetőséget ad modellek kidolgozására és azok segítségével létező vagy feltételezett rendszerek vizsgálatára. A digitális számítógépek elterjedésével a szimulációs módszer is nagymértékben terjed, és szinte minden tudományág mindennapi eszközévé vált. Ezzel egyidejűleg a tervezés és megvalósítás különböző fázisaiban a modellalkotáson keresztül lehetőség nyílt olyan műveletek megvalósítására, amelyeket a valóságos rendszerben (vagy elemeken) túl drága, célszerűtlen, vagy esetleg lehetetlen lenne végrehajtani [40].

A számítógép mint „háttér" a mikroprocesszoros rendszerek megvalósításának minden részműveleténél jelen van (l. az 1. fejezetet) az elemek kidolgozásától a rendszerek létrehozásáig, beleértve a rendszer által megoldott alkalmazási feladatokat is. Ha most ismét vázlatosan végigtekintjük ezt a folyamatot a szimulációs módszerek alkalmazása szempontjából, azt találjuk, hogy majdnem minden fázisban lehet szimulációs módszert alkalmazni, persze sokkal kisebb azon problémák száma, ahol érdemes is ezzel a lehetőséggel élni, néhány esetben azonban e módszer alkalmazása szükségszerű.

Lássunk akkor néhány kiragadott példát ebből a folyamatból. Az LSI és VLSI alapú elemkészlet kidolgozása során a tervezési folyamat részeként a logikai tervek

ellenőrzésére szimulációs módszereket alkalmaznak. Felhasználásuk szinte elkerülhetetlen, mivel egy hiba vagy módosítás rendkívül költséges és hosszadalmas folyamat megismétlését teszi szükségessé. A diszkrét SSI, MSI integrált elemekből készített rendszerek számára kidolgozott, kapu szintű logikai szimuláció módszereit [40] az LSI és a VLSI környezethez továbbfejlesztették [41].

Ennek a nagyon érdekes és fontos területnek a problémáira (idő- és állapotreprezentáció-leírás és hozzárendelt adatstruktúra, modellépítő és strukturális vizsgálatok, dinamikus szimuláció stb.) azonban könyvünkben nem térünk ki.

A másik jelentős terület, ahol a szimuláció, a modellezés alapvető fontosságú: az ellenőrzés. Itt nem csupán az ellenőrzési módszer kidolgozásánál használható a szimuláció (l. az 5. fejezetet), hanem a meghibásodás folyamatának vizsgálatában (diagnosztika)is [42].

A szimulációs vizsgálatok esetén fontos kérdés a modell kialakítása, mind a belső paraméterek, mind a vizsgálati feltételeket leíró külső adatok figyelembevételével. Amennyiben ezek nincsenek helyesen megválasztva, akkor nem várhatunk helyes eredményeket még a leggondosabban elkészített szimulációs programtól sem. Erre vonatkozik a szimuláció egyik „alaptörvénye", az ún. GIGO törvény [43] (Garbage In-Garbage Out), amely szerint, ha szemetet adunk be, úgy szemetet is kapunk vissza eredményül. A modell verifikálása, ill. érvényességének vizsgálata azonban sokszor nem egyszerű feladat [44]. A szimulációs modell kidolgozása során mindenkor megfelelő kompromisszumot kell kialakítani a valóságos rendszer bonyolultsága és a szimulációs módszer kidolgozásának bonyolultsága (ára) között.

A szimuláció eszközként használható a mikroprocesszoros rendszerek tervezéséhez is, mind az architektúra, mind a program kidolgozása, így mind a hardver, mind a szoftver alternatívák vizsgálata során. A szimuláció segítségével lehetőség van a végső tervezés és megvalósítás előtt a várható jellemzők vizsgálatára, a különböző megoldások összehasonlítására és ezen keresztül optimalizálására [45].

A szimulációs módszerek felhasználhatók mind az egyes modulok tervezésében [45], mind a rendszer kialakításában – szemben az LSI és VLSI elemek szimulációjánál használt kapu szintű leírással –, csak a vizsgálat szempontjából lényeges funkciók szintjén kell definiálnunk az egyes egységeket. Ezek persze eltérőek, amikor egy modulon belüli részek kapcsolatát vagy e modulokból kialakított rendszer tulajdonságait vizsgáljuk. Az alkalmazott módszer csupán a kapcsolatok adott szintje szempontjából szükséges lényeg kiemelése miatt az egyes egységeket fekete doboz (black-box) formában definiálja. Ehhez azonban sokszor (pl. bizonyos kommunikációs módszerek modellezése esetén) a „dobozba való belelátásra" is szükség van (szürke doboz: gray-box leírási szint [46]).

A hagyományos szimulációs nyelvek (amelyek közül a legismertebb a CSL, GPSS, SIMSCRIPT, SIMULA [40]) nem alkalmasak mikroprocesszoros rendszerek szimulációjának megvalósításához.

Erre a célra – tehát az elemek definiálására, azaz „bemenő nyelvként" – a különböző CHDL nyelveket (Computer Hardware Description Language: hardverleíró nyelv) használják. A rendkívül nagy számú ilyen nyelv [47], [48] közül a legjobban elterjedtek a: CDL (Computer Design Language: számítógép-tervező nyelv) [49]; a DDL (Digital Design Language: digitális tervező nyelv) [50], az AHPL (A Hardware Programming Language: egy hardverprogramozási nyelv) [51], az ISP (Instruction Set Processor: utasításkészlet processzor) és a PMS (Processor–Memory–Switch: processzor–tár–kapcsoló) [52] nyelv (utóbbi kettőt együtt szokták ISPS-nek is nevezni).

Egy jellemző korszerű szimulációs rendszer [53] alapparancsait röviden összefoglalva a következők:

- Építő parancsok, amelyek – előre megadott – elemek alkalmazását és azok összekötését deklarálják. Pl.:

  DECLARE 8085 [2], **MEMORY** [1] < 4,8 >, **MEMORY** [2] < 8,8 >

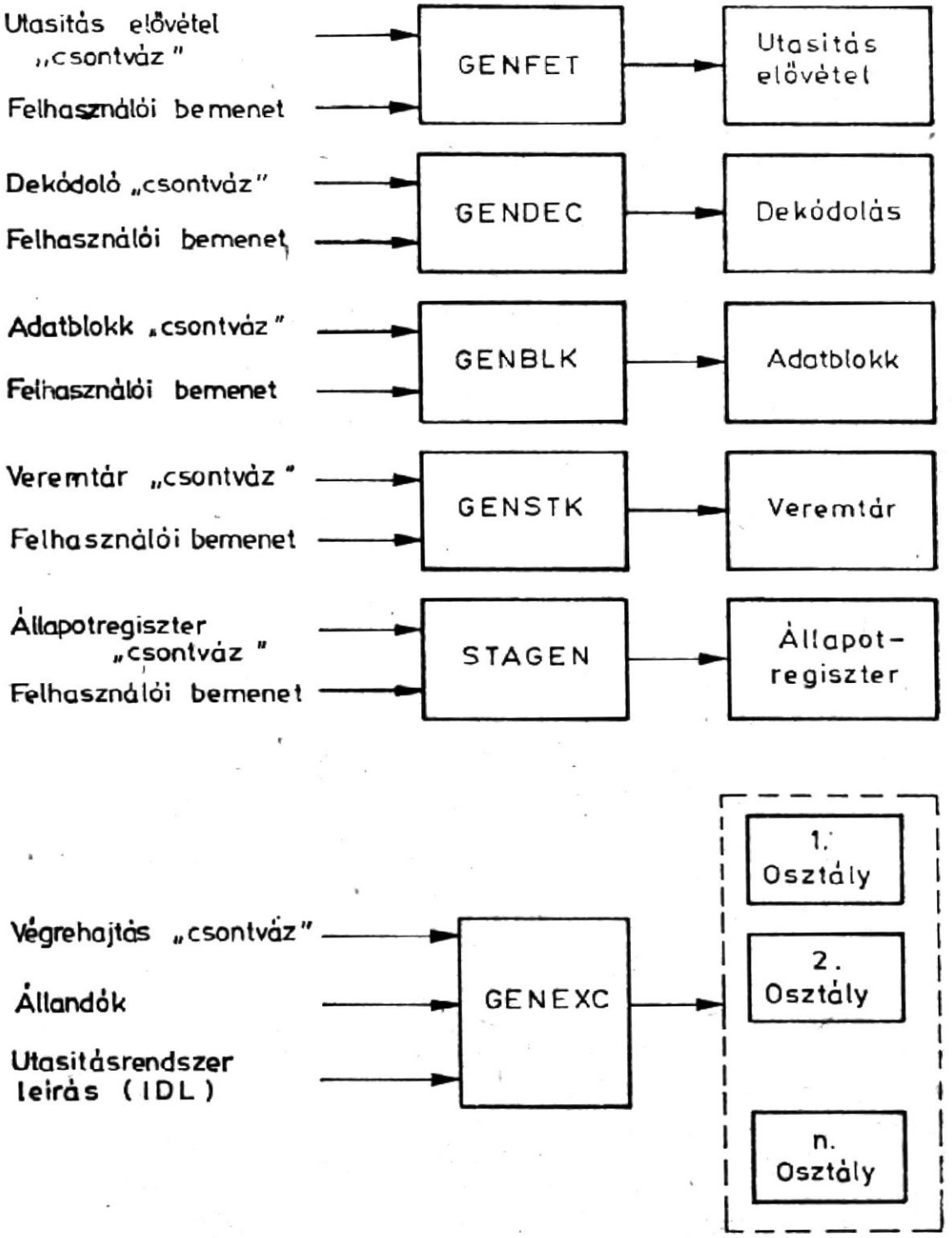
  Ez két 8085-ös processzort, egy 4-bites címmezőjű és 8-bites adatmezőjű tárat és két 8-bites című és 8-bites adatmezőjű tárat definiál.
- Az elemek összekötését végző parancsokat – a szabványos bekötést figyelembe véve – a következő módon lehet megadni:

  CONNECT 8085 [2] < 12–15 >, **MEMORY** [1] < 0–3 > **TO WIRE** < 0–3 >

  Ez a parancs négy, 8085-ös címvezetéket köt össze az 1. tár megadott címvezetékeivel a hátlap (back-plane) 0...3 vezetékei segítségével. Ennek hatására olyan közös táregység jön létre, amelyet mindkét processzor megosztva használhat. Egy vezetéket kiiktathatunk a **DISCONNECT** parancs segítségével. Az említett parancsok dinamikusak, tehát a szimuláció „futása" alatt bármikor kiadhatók.
- A **CLOCK** parancs segítségével a processzorok órajele egymástól függetlenül megadható.
- A vezetékek kapcsolatát, ill. állapotát pl. a **FLOAT** lebegés vagy a **WIREDOR** huzalozott VAGY paranccsal specifikálhatjuk.
- A tár betöltését és kiiratását, megadott sínek logikai értékének beállítását és kiolvasását végzi a második parancscsoport. A felhasználó bármikor „befagyaszthatja" a rendszert és a **SAVE** paranccsal megőrizheti annak aktuális állapotát, amelyet később a **RESTORE** paranccsal visszahozhat.
- A nyomkövető parancsok lehetővé teszik töréspontok definiálását és a hátlapon levő csatlakozókat összekötő sín átviteleinek regisztrálását egy file-ban. A **TRACE** parancs lehet helyi is, ami pl. egy adott processzoron belüli műveletek (pl. regiszterek közötti átvitelek) nyomon követését is végzi. Egymással párhuzamosan, adott időpillanatokban rögzített állapotfile-ok összehasonlításával lehetővé teszi versenyhelyzetek detektálását is.

A szimulációs rendszerek külön kategóriáját jelentik az ún. *utasításszimulátorok*. Ezek lehetővé teszik különböző mikroprocesszor-családok utasításrendszereinek összehasonlítását egy adott feladat szempontjából, a megvalósítás előtt egy mikroprogramozott rendszer (l. a 11. fejezetet) számára definiált utasításokban rejlő lehetőségek vizsgálatát, és így az utasításrendszer optimalizálását vagy pl. egy adott feladat megoldásához felhasználható algoritmusok „kipróbálását". Külön felhívjuk a figyelmet arra, hogy az említett feladatok mindegyikénél csak akkor várhatunk megfelelő eredményt a szimulációs módszertől, ha azok aritmetikai–logikai műveletekkel írhatók le, és a környezeti hatások (pl. megszakítások, a valódi időtartománynak megfelelő be- és kivitelek) nem jelentősek. Utóbbi esetben ui. az utasításszimulátorunkat ki kell egészítenünk e hatásokat utánzó környezetszimulátorral is, ami sokszor nagyon bonyolult feladat, és az egész szimulációs módszer gyakorlati alkalmazhatóságát kérdésessé teheti.

Ha sokféle utasításrendszer szimulálására kell felkészülnünk, érdemes egy generátorprogramot készíteni, amely a fontosabb feladatokat (10.8. ábra) megoldó programrészek „csontváza" (skeleton) felhasználásával, egy erre a célra definiált nyelv az IDL (**I**nstruction **D**efinition **L**anguage: utasításdefiníciós nyelv) segítségével leírt, utasításkészlet szerint hozza létre a szimulációs programot [55]. Az utasításszimulátor segítségével végrehajtott program bármely helyén definiálható töréspontokban a „státusz" (regiszterek tartalma, sínek állapota, tárak tartalma, végrehajtott utasítások száma és felsorolása, eltelt idő stb.) kiíratható; ami más szóval azt is jelenti, hogy a szimuláció „nyomkövetésre" is használható.



10.8. ÁBRA. Utasításszimulátor generátorprogramjának felépítése

IRODALOM

[1] *Knuth, D.:* Computer programming as an art. Com. of ACM, Dec. 1974. 667–673.

[2] *Dahl, O. J.–Dijkstra, E. W.–Hoare, C. A. R.:* Structured Programming. Academic Press, London, 1972.

[3] *Kildall, G. A.:* High-level language simplifies microcomputer programming. Electronics, June 24, 1974. 103–109.

[4] *Gibbons, J.:* When to use higher-level languages in microcomputer-based systems. Electronics, August 7, 1975. 107–111.

[5] *Posa, J. G.:* Programming microcomputer systems with high-level languages. Electronics, January 18, 1979. 105–112.

[6] *Bass, C.–Brown, D.:* A perspective on microcomputer software. Proc. IEEE, June 1976. 905–909.

[7] *Myers, G. J.:* Advances in computer architecture. John Wiley and Sons, New York, Chichester, Brisbane, Toronto, 1978.

[8] *Hemenway, J.–Teja, E.:* Pascal. EDN, April 5, 1979. 79–83.

[9] *Candill, P.:* Using assembly coding to optimize high-level language programs. Electronics, February 1979. 121–124.

[10] *Lőcs Gy.:* A BASIC programozási nyelv. Műszaki Könyvkiadó, Budapest, 1974.

[11] *Kernaghan, B.–Richie, D.:* The C-programming language. Prentice-Hall, 1978.

[12] Report on the programming language PLZ. Zilog, 1978.

[13] *Jensen, K.–Wirth, N.:* Pascal: User manual and report. Springer Verlag, Berlin, 1974.

[14] *Conway, R.–Gries, D.–Zimmerman, E.:* A Primer on Pascal. Winthrop, 1976.

[15] *McLeod, J.:* μC development systems have the hardware, software 16-bit μPs need. Electronic Design, September 13, 1979. 92–95.

[16] *Michelman, E.:* Basic's full power and versatility emerge with a μC development system. Electronic Design, June 7, 1979. 92–95.

[17] *Mayer, R. E.:* A psychology of learning BASIC. Communications of the ACM, November 1979. 589–593.

[18] *Maples, M. D.–Fisher, E. R.:* Real-time microcomputer applications using LLL-Basic. Computer, January 1977. 45–50.

[19] *Maples, M. D.:* Lawrance Livermore Labs' Basic–II-aimed right at real-time applications. Electronic Design, July 19, 1979. 94–98.

[20] Az OS/i BASIC interaktív programozási nyelv. KFKI, Budapest, 1972.

[21] *McIntire, T. C.:* Software interpreters for microcomputers. John Wiley and Sons, New York, Santa Barbara, Chichester, Brisbane, Toronto, 1978.

[22] *Johnson, G. R.–Mueller, R. A.:* Automated generation of cross-system software for microcomputers. Computer, January 1977. 23–31.

[23] *Chung, K. M.–Yuen, M.:* A „tiny" Pascal compiler. Byte, September, 1978. 58–65, 149–155.

[24] *Parnas, D.:* A technique for software module specification. Com. of ACM, May 1972. 330–336.

[25] *Bass, C.:* PLZ: A family of system programming languages for microprocessors. Computer, March 1978. 34–39.

[26] Z80–PLZ User's Guide. Zilog, 1978.

[27] *Brown, W.:* Modular programming in PL/M. Computer, March 1978. 40–46.

[28] *Varga L.:* Rendszerprogramok elmélete és gyakorlata. Akadémiai Kiadó, Budapest, 1978.

[29] *Hicks, S. M.:* Forth's forte is tigher programming. Electronics, March 15, 1979. 114–118.

[30] *Schindler, M.:* Software's main challenger is a „hybrid" that can be mass-produced. Electronic Design, May 24, 1979. 78–83.

[31] *Wecker, D. B.–Krutz, R. L.–Tuma, D. T.:* High-level design language develops low-level microprocessor-independent software. Computer Design, June 1979. 140–149.
[32] *Schindler, M.:* Microcomputers are going Pascal, but most need a development system. Electronic Design, August 2, 1979. 36–38.
[33] *Schneider, G. M.:* Pascal: An overview. Computer, April 1979. 61–66.
[34] *Wirth, N.:* Algorithms + Data Structures = Programs. Prentice Hall, Englewood Cliffs, 1976.
[35] *Bowles, K.:* Problem Solving Using Pascal. Springer–Verlag, Berlin, 1977.
[36] *Krouse, T.:* Pascal language. Electronic Design, September 13, 1978. 82–86.
[37] *Schneider, G. M.:* Weingart, S. and Perlman, D.: Introduction to Programming and Problem Solving with Pascal. John-Wiley and Sons, New York, 1978.
[38] *Pasman, W. J. A.:* Implementation of PASCAL on an 8080 microcomputer. Euromicro Journal, November 1979. 363–369.
[39] *Ravenel, B. W.:* Toward a Pascal standard. Computer, April 1979. 68–82.
[40] *Jávor A.–Benkő T.né:* Diszkrét rendszerek szimulációja. Műszaki Kiadó, Budapest, 1979.
[41] *Teets, J. J.:* The role of simulation in LSI design. Proc. AFIPS, Spring Joint Computer Conference, 1972. 1065–1070.
[42] *Roth, J. P.–Bouricius W. G.–Schneider, P. R.:* Programmed algorithmus to compute test to detect and distinguish between failures in logic circuits. IEEE Trans. on Electronic Computers, October 1967. 567–580.
[43] *Reitman, J.:* Computer simulation applications. (Discrete-event simulation for synthesis and analysis of complex systems). John Wiley and Sons, Inc. New-York, London, Sydney, Toronto, 1971.
[44] *Nutt, G. J.:* A case study of simulation as a computer system design tool. Computer, October 1978. 31–36.
[45] *Kangley, F. J.–Sheehan, J.–LaGro G.:* Simulating modular microcomputèrs. Simulation, May 1979. 141–154.
[46] *Su, S. Y. H.:* Hardware description language applications: An introduction and prognosis. Computer, June 1977. 10–13.
[47] *Lipovski, G. J.:* Hardware description languages: Voices from the Tower of Babel. Computer, June 1977. 14–17.
[48] *Su, S. Y. M.:* A survey of computer hardware description languages in the U.S.A. Computer, December 1974. 45–51.
[49] *Chu, Y.:* Introduction to Computer Organization. Prentice Hall, Englewood Cliffs, 1970.
[50] *Dietmeyer, D.:* Logical design of digital systems. Allyn and Bacon, Boston, 1971.
[51] *Hill, F.–Peterson, G.:* Digital systems: Hardware organization and design. Wiley, Englewood Cliffs, 1973.
[52] *Bell, C. G.–Newell, A.:* Computer structures: Readings and examples. McGraw Hill, New York, 1971.
[53] *Klein, D.:* MMPS – A reconfigurable multi-microprocessor simulator. Proc. AFIPS, National Computer Conference, 1979. New York, 199–203.
[54] *Unger, B. W.:* Programming languages for computer system simulation. Simulation, April 1978. 101–110.
[55] *Mueller, R. A.–Johnson ,G. R.:* A generator for microprocessor assemblers and simulators. Proc. of the IEEE, June 1976. 921–931.
[56] *Parnas, D. L.:* A technique for software module specification with examples. Commun. of the ACM, May 1972. 330–336.
[57] *Табацка, Д.—Згрызек, М.:* Диалоговая система производства и симуляции программ для микро-ЭВМ.

11. MIKROPROGRAMOZOTT RENDSZEREK

A fix utasításrendszerű MOS technológiával készült mikroprocesszoroknál kb. egy nagyságrenddel nagyobb sebességű, bipoláris technológia segítségével megvalósított elemcsaládok, az ún. bitszeletelt (bit-sliced) mikroprocesszorok [1], [2], [3], [4] és nagyon nagy sebességű félvezetős tárolóelemek (l. az 1. fejezetet) megjelenése is új lehetőségeket biztosított a digitális rendszerek fejlesztésében. Ebben az esetben a számítógéptechnika egy adott területe, a mikroprogramozás [5], [6], [7], [8] (mégpedig a felhasználó, és nem a gyártó által megvalósított formában) eredményeit hasznosítják újabb célok, alkalmazások szolgálatában.

A bitszeletelt mikroprocesszor-elemek felépítése eltér a MOS mikroprocesszorok architektúrájától, amelyek egy adott – fix, nem változtatható – utasításrendszert – az ún. gépi utasításokat – valósítanak meg.

A mikroprocesszor-architektúra két fő része: az utasításrendszernek megfelelő feldolgozóegység (és az operandusokat tároló regiszterek) és a vezérlőegység – elsősorban az utasítások értelmezése, dekódolása és az ennek megfelelő vezérlőjelek előállítása – egy tokon belül valósul meg.

Ezzel szemben a bipoláris bitszeletelt mikroprocesszor-családok esetén ezek a funkciók szétválnak, külön tokok formájában valósulnak meg. A „szelet" név is abból származik, hogy az adott funkciókat (feldolgozóelemet, regisztereket stb.) csak adott – pl. 2,4,8 – bitnek megfelelő részre (ez a „szelet") valósítják meg egy tokban, és ha ennek többszörösére van szükségünk, akkor e szeletek párhuzamos működtetésével valósítható meg a kívánt architektúra. Az, hogy mint említettük, ezen elemek mikroprogramozhatók, mindjárt utal egy másik alapvető különbségre is. Az utasítások ui. a MOS mikroprocesszoroktól eltérő és annál egyszerűbb, alacsonyabb szintű – a hardverelemekhez közvetlenebbül kapcsolódó – műveleteket valósítanak meg. Így ezen elemek segítségével a tervező nem csupán különböző rendszerarchitektúrákat és szóhosszúságot hozhat létre, hanem a rendszer utasításkészletét is saját maga definiálhatja.

Ezen utasításkészlet utasításait (amelyeket ismét gépi utasításoknak, vagy ebben az esetben makroutasításoknak is szoktak nevezni) a hardverelemekhez és funkciókhoz közvetlenül kapcsolódó utasítások: az ún. mikroutasítások sorozata formájában valósítjuk meg. Ezen utasításokat egy újabb tárban: az ún. vezérlő- (control) tárban tároljuk. Ilyen módon megvalósított rendszerünk tehát két szinten programozható, a gépi (makró) utasítások szintjén „hagyományos" módon, és a mikroutasítások (mikroprogram) szintjén, amely az architektúra és az utasításkészlet (általában egyidejű) meghatározását, ill. módosítását jelenti. Ez egyben azt is jelenti, hogy mikroprogramozott rendszerek sokkal rugalmasabbak és a bipoláris technológia alkalmazása következtében az elemi (mikro) utasítások szintjén sokkal nagyobb műveleti sebességűek is. Ezen előnyök azonban persze most sincsenek „ingyen". A mikroprogramozás megvalósításához jelentős „beruházás" szükséges, amely elsősorban megfelelő fejlesztőrendszer megvalósítását – megvásárlását – jelenti. A tervezőnek ebben az esetben a fix utasításrendszerű mikroprocesszorok alkalmazásához sokkal

nagyobb mértékben, magasabb szinten kell ismernie a szoftvert és a hardvert, hogy az egész rendszer tervezését, az egymással összefüggő architektúra–mikroutasítás rendszer megvalósítását elvégezhesse. A mikroutasítások ugyanakkor sokkal összetettebbek a makroutasításoknál, így a mikroprogram elkészítése (mikrokódolás) sok időt vesz igénybe. Különösen nehezíti a hibák felderítését, a rendszer működésének vizsgálatát, hogy a hardverrendszert (architektúrát) és az utasításokat (mikroutasításokat) egyszerre definiáljuk, és hibák esetén általában nem egyértelmű, hogy mi a hibajelenség tulajdonképpeni oka (pl. dinamikus működési hiba az architektúra valamelyik elemében vagy rossz mikroprogramrész).

Ennek ellenére a mikroprogramozáshoz és a megvalósításához alkalmazott korszerű LSI elemeknek jelentősége az egyes alkalmazási területeken egyre növekszik. Ilyenek a jelfeldolgozás (pl. digitális szűrés, FFT: Fast-Fourier Transzformation, azaz a gyors Fourier-transzformáció), adatkommunikáció, folyamatszabályozás, nagy sebességű vezérlőegységek (pl. speciális perifériavezérlők, grafikus megjelenítők stb.). Minden olyan algoritmus esetén, ahol a párhuzamos hardvermegvalósítás lehetséges (pl. a digitális képfeldolgozás különféle feladatai), és ugyanakkor a nagyobb műveleti sebességre is szükség van, különösen alkalmasak ezek a rendszerek.

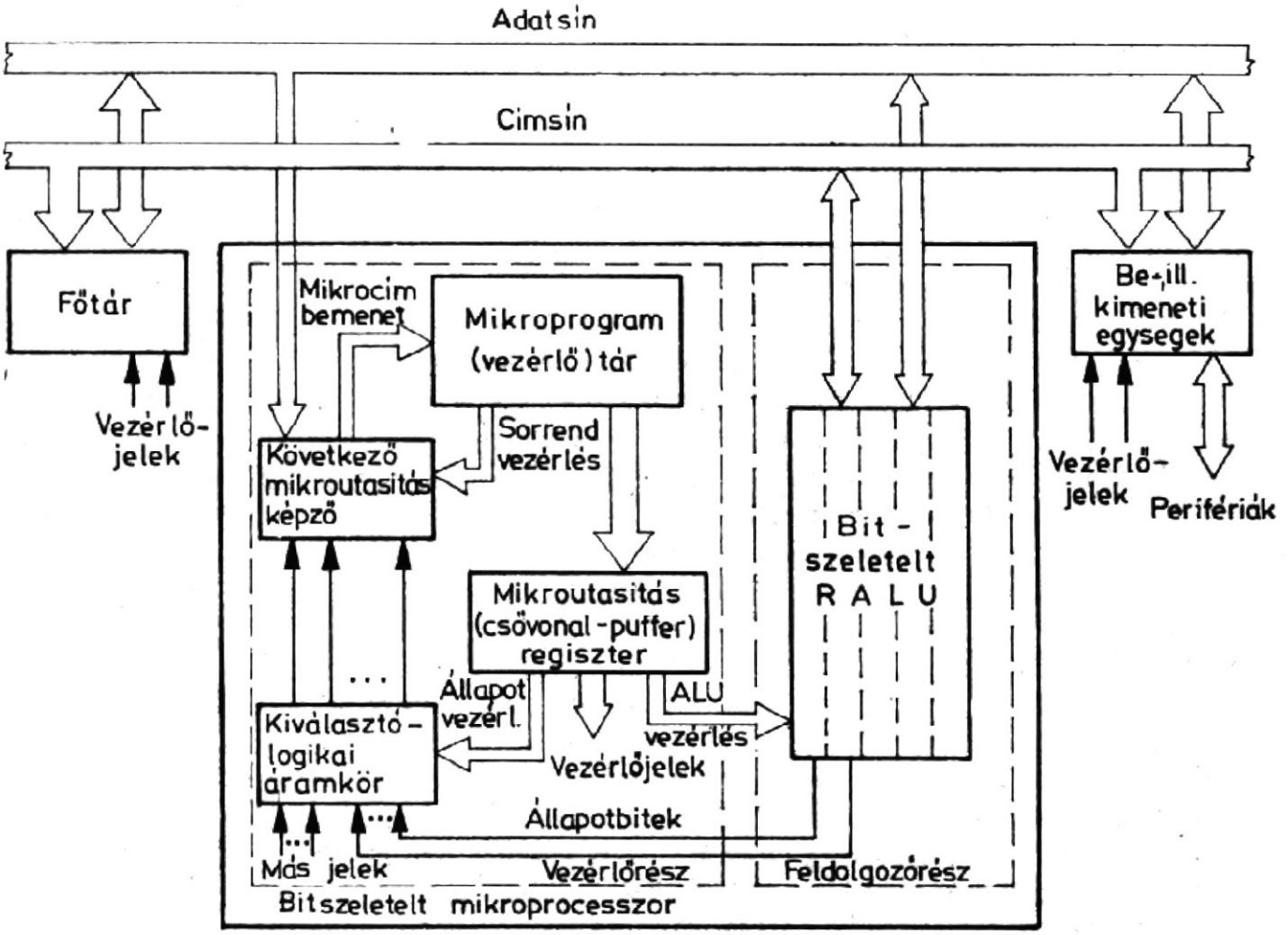
Fontos alkalmazás az emuláció, amely a számítógépek utasításrendszereivel (teljes architektúrájával) megegyező rendszerek megvalósítását teszi lehetővé. Ez a módszer lehetővé teszi – az azonos célra kifejlesztett MOS mikroprocesszorok mellett– a rendkívül sok és így nagy értékű szoftver – mind rendszer és mind felhasználói-programok – „átmentését" az LSI elemek nyújtotta lehetőségek egyidejű felhasználásával. Az ilyen mikroprogramozott rendszerek ezután a mikroprogramozás egyéb korszerű felhasználására (pl. mikrodiagnosztika [9]) vagy a feladatok mikroprogram és hagyományos szoftver megfelelő „keverékével" való megoldására, vagy egyes rendszerprogram funkciók mikroprogrammal való megvalósítására (melyet vándorlásnak: migration neveznek) [10]) alkalmazhatók.

11.1. MIKROPROGRAMOZOTT ARCHITEKTÚRÁK

Nem lehet célunk a mikroprogramozás szerteágazó hardver–szoftver problémakörének [5], [11] ismertetése, azonban a bitszeletelt mikroprogramozható elemek alkalmazásának megértéséhez szükségünk van néhány alapelv, főbb szervezési és felépítési elv és hozzájuk kapcsolódó terminológia rövid összefoglalására.

A bitszeletelt mikroprogramozható elemek alkalmazásával kialakítható digitális rendszerek általános blokkvázlatát láthatjuk a 11.1. ábrán.

A 11.1. ábra szerinti elrendezésben a mikroprocesszor makro- (gépi) utasítást vesz elő a rendszer főtárából a mikroprogram (vezérlő-) tárából kiolvasott mikroutasítások segítségével, majd a makroutasítások műveleti kódját értelmezi oly módon, hogy a kódnak megfelelő mikroprogram-kezdőcímet állít elő. Ezután egy – a makro-operációnak megfelelő - mikroutasítás-sorozat kerül végrehajtásra. A makroutasítás operandusrésze a RALU-ba (Register-, Arithmetic-Logic-Unit: regiszter és aritmetikai–logikai egység) kerül, hogy ott akár az előírt művelet kiszámításához, akár a főtárcím kiszámításához (pl. ugrási szubrutin vagy megszakításkezelő makroutasítások esetén) felhasználásra kerüljenek. A rendszer további erőforrásait pl. (be- és kimeneti egységeket) is a mikroutasítások vezérlik. A mikroutasítás-képző (sequencer) szabja meg a következő végrehajtandó mikroutasítás címét az éppen végrehajtott mikroutasításból származó információ és feltételes elágazást megszabó állapotbitek segítségével. A mikroprogramozott rendszerben ily módon legalább két szintű vezérlés van, amelyet kétszintű programozás – a makro- és mikroszint - valósít meg.



11.1. ÁBRA. Bitszeletelt mikroprogramozható elemek alkalmazásával kialakítható rendszer blokkvázlata

Gyakran a mikroprogramtár kimenetére egy külön mikroutasítás-regisztert (amelyet funkcionális szerepe alapján pipeline: csővonalregiszternek is neveznek) iktatnak be. Így gyorsabb működés érhető el az aktuális mikroutasítás végrehajtásának és a következő mikroutasítás elővételének (fetch) átlapolásával (overlapping).

A feldolgozóegység legfontosabb része a szeletekből felépített aritmetikai–logikai egység (ALU), amely az összes aritmetikai és logikai műveleteket végzi, és a szükséges regisztereket (utasításregiszter, programszámláló, átmeneti tárolóregiszterek stb.) tartalmazza. A rendszer szóhosszának – bitszámának – megfelelő számú RALU szelet valósítja meg mind az ALU-t, mind az egyes regisztereket. Ezért a szeletszám egyenlő a szóbitszám/szeletbitszám. A szeletekből felépített aritmetikai–logikai egység működését gyors átvitelképző (carry look-ahead) elemek alkalmazásával gyorsíthatjuk. A szokásos RALU elemek bizonyos belső – művelet eredményétől függő – állapotok (pl. zérus eredmény, túlfolyás, előjel stb.) kijelzésére is képesek, és az ALU művelet és a regiszterkiválasztást végző összetett (kódolt) vezérlés dekódolását is elvégzik.

Ily módon a makrorendszer felépítése nagyon egyszerű lehet, és a felhasználónak nem szükséges tudnia arról, hogy mikroprogrammal valósult meg a feladata. Természetesen szükség esetén lehetőség van arra, hogy a mikroprogram megváltoztatásával az adott alkalmazás szempontjából optimalizáljuk a rendszer jellemzőit.

A mikroprogramozás a számítógépek tervezésében és alkalmazásában is hosszabb ideje használt, jól kidolgozott módszer [5], [11], amelyet most a félvezetőelemek fejlődése sokkal szélesebb körben tesz hozzáférhetővé.

A mikroutasítás felépítése az adott mikroprogramozott vezérlő kialakítása szempontjából meghatározó, mivel ez egyben az architektúra kialakítását is meghatározza. Egyetlen mikroutasítás általában a hardvererőforrások különböző csoportjait is vezérli, kihasználva a párhuzamos végrehajtásból származó lehetőségeket. Egy-egy adott feladatcsoport (pl. regiszterkijelölés, ALU műveletkiválasztás, adott vonalakon való átvitel megengedése stb.) vezérlését a mikroutasítás egy része végzi, amelyet mezőnek szoktak nevezni. Az adott mezőhöz rendelt műveletet mikrooperációnak (microoperation) nevezik.

Vertikális mikroutasítás felépítés esetén a mikroutasításban csak egyetlen mikrooperáció van (a hozzá kapcsolódó operanduskijelölésekkel és a következő mikroutasítás kiválasztását vezérlő mezővel). *Horizontális felépítés* esetén a mikroutasítás számos, párhuzamos végrehajtást vezérlő mezőt tartalmaz. (Ennek megfelelően a vertikális felépítésű mikroutasítás szóhossza kisebb, a horizontálisé nagyobb.) A horizontális mikroutasítások felépítésük miatt szorosabban kapcsolódnak a végrehajtó hardverelemekhez, ezért a segítségükkel megvalósított rendszert kemény (hard) vagy strukturális (structural) mikroprogramozásnak is nevezik. Ezzel szemben vertikális mikroutasítások esetén, ahol a hardver jobban eltávolodik a konkrét mikroutasításoktól, „puha" (soft) vagy funkcionális mikroprogramozásnak nevezik.

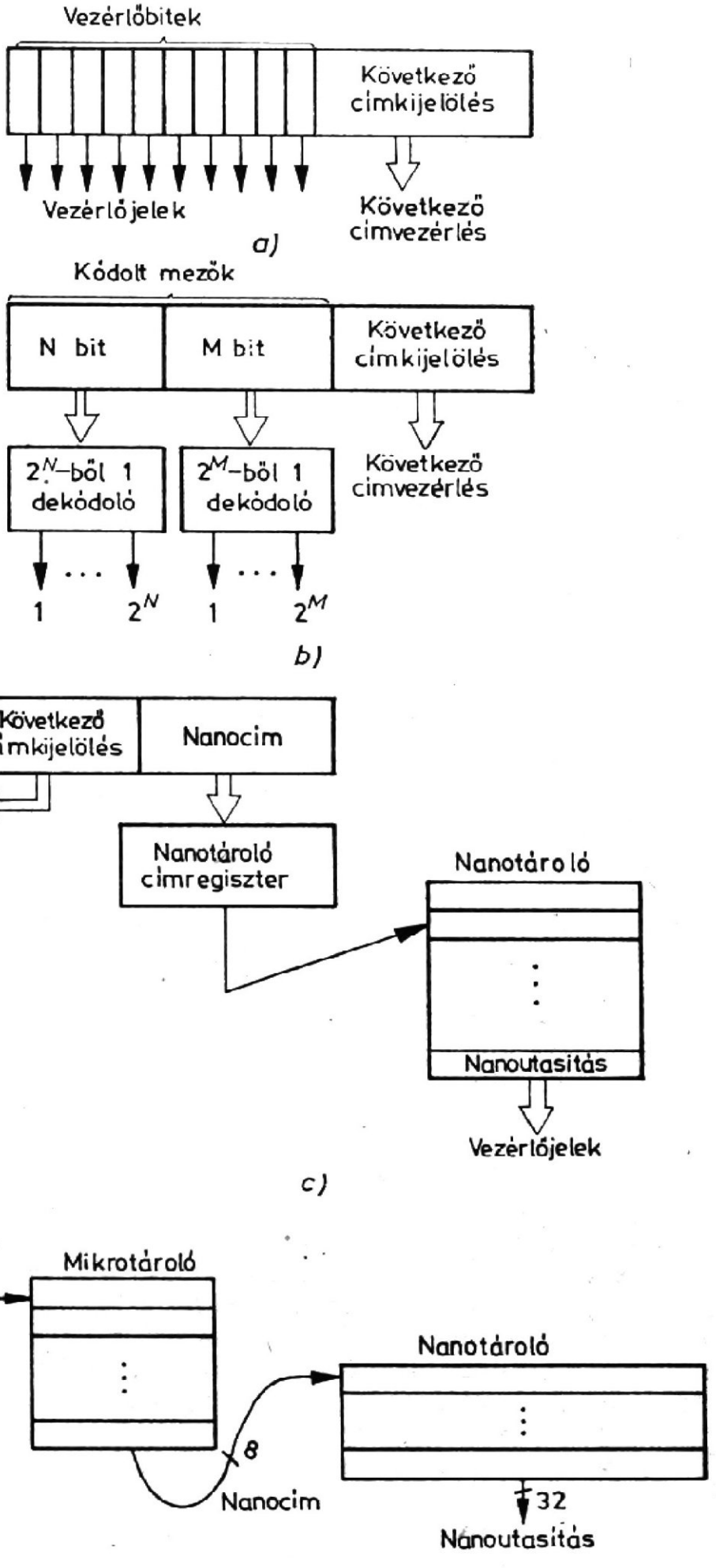
A mikroutasítás formátuma lehet állandó vagy változó. Az első esetben az adott bitek (bitcsoportok) mindig azonos módon értelmezendők, míg változó formátum esetén az egyes bitek (mezők) jelentése változik. Utóbbi esetben rövidebb mikroutasításokat kapunk, de kiegészítő logikai áramkörök (és idő) szükségesek a formátumvezérlés külön utasítások segítségével való megvalósításához.

Ha a mikroutasítás minden egyes bitjét adott vezetékek vagy rendszerpontok vezérlésére külön használjuk, a mikroutasítás *nem kódolt* formájú (unpacked: nem összepakolt, vagy exploded: szétbontott) (11.2*a* ábra). Ez ugyancsak nagyobb mikroutasítás-szóhosszat eredményez, de ugyanakkor nagyobb működési sebességet, nagyobb mértékű „párhuzamosságot" is megvalósít. A másik megoldás az egymást kölcsönösen kizáró vezérlővezetékek, ill. állapotok (pl. ALU művelet, egy-egy regisztertömbből való regiszterkijelölés) egy kódolt mikroutasítás mezővel való vezérlése. Ily módon n-bites mezővel 2^n vezérlési állapot állítható elő, amelyet azután dekódolni kell. Az ilyen mikroutasítást *kódolt formájúnak* (packed: pakolt formájúnak) nevezik (11.2*b* ábra). Alkalmazhatunk olyan megoldást is, ahol a mikroutasítás egy további „szinten" levő újabb tárat címez meg, az ún. *nanotárat* (nanostore), amely a nanoutasításokat (nanoinstruction) tartalmazza (11.2*c* ábra). A 11.2*d* ábra egy másik *kétszintű* megoldást mutat be. A mikroprogram nagyszámú (pl. 1 k szó) rövid (pl. 8 bit), szóból áll, míg a nanotár kevesebb (pl. 256 szó), de hosszabb (pl. 32 bites) szót tartalmaz.

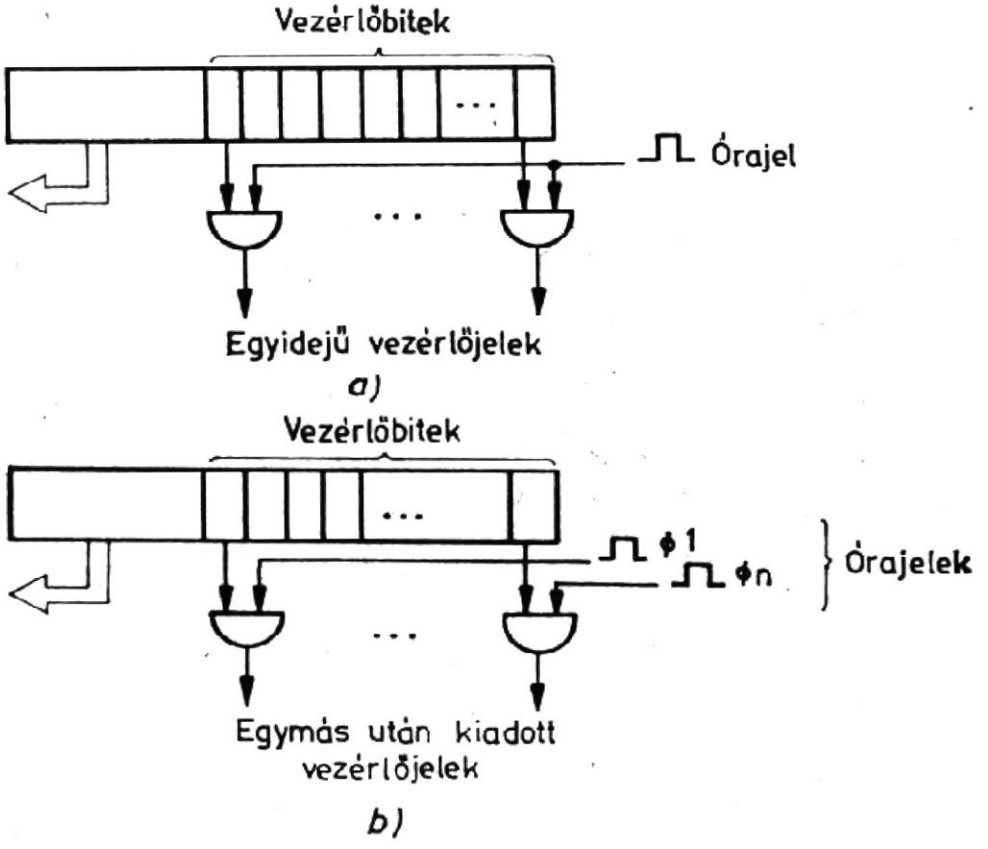
A mikroutasítás időzítése vagy egyfázisú (11.3*a* ábra), vagy többfázisú (11.3*b* ábra). A többfázisú időzítés lehet szinkron, amikor is a fázisok száma állandó, vagy aszinkron, amikor ez a szám változtatható. Az egyes mezőkhöz az időzítés-hozzárendelés vagy külön megadható (explicit) vagy a mezőbe beleértett (implicit).

A mikroutasítás formátumának megválasztásakor a tervezőnek a következőket kell mérlegelnie: tárméret, időzítés, végrehajtási sebesség, az elágaztatás flexibilitása, dekódolás. Általában a mikroutasítás jellemzőire leginkább a feldolgozórész felépítése (a választott elemkészlet [1], [2], [3], [4]) és a megvalósítandó gépi (makro-) utasításkészlet gyakorol hatást.

Ha egy digitális rendszer vezérlése az ismertetett módon (mikroprogramtárban tárolt mikroutasítások segítségével) történik, a rendszert mikroprogramozottnak nevezzük. Ha a felhasználó mikroprogramot akar készíteni, akkor *mikroprogramozható* (microprogrammable) rendszerre és megfelelő hardver–szoftver eszközökre van



11.2. ÁBRA. Mikroutasítás-formátumok
a) **nem kódolt,** *b)* **kódolt,** *c)* **nanoprogramozott,** *d)* **kétszintű**



11.3. ÁBRA. Mikroutasítás időzítése *a)* egyfázisú, *b)* többfázisú

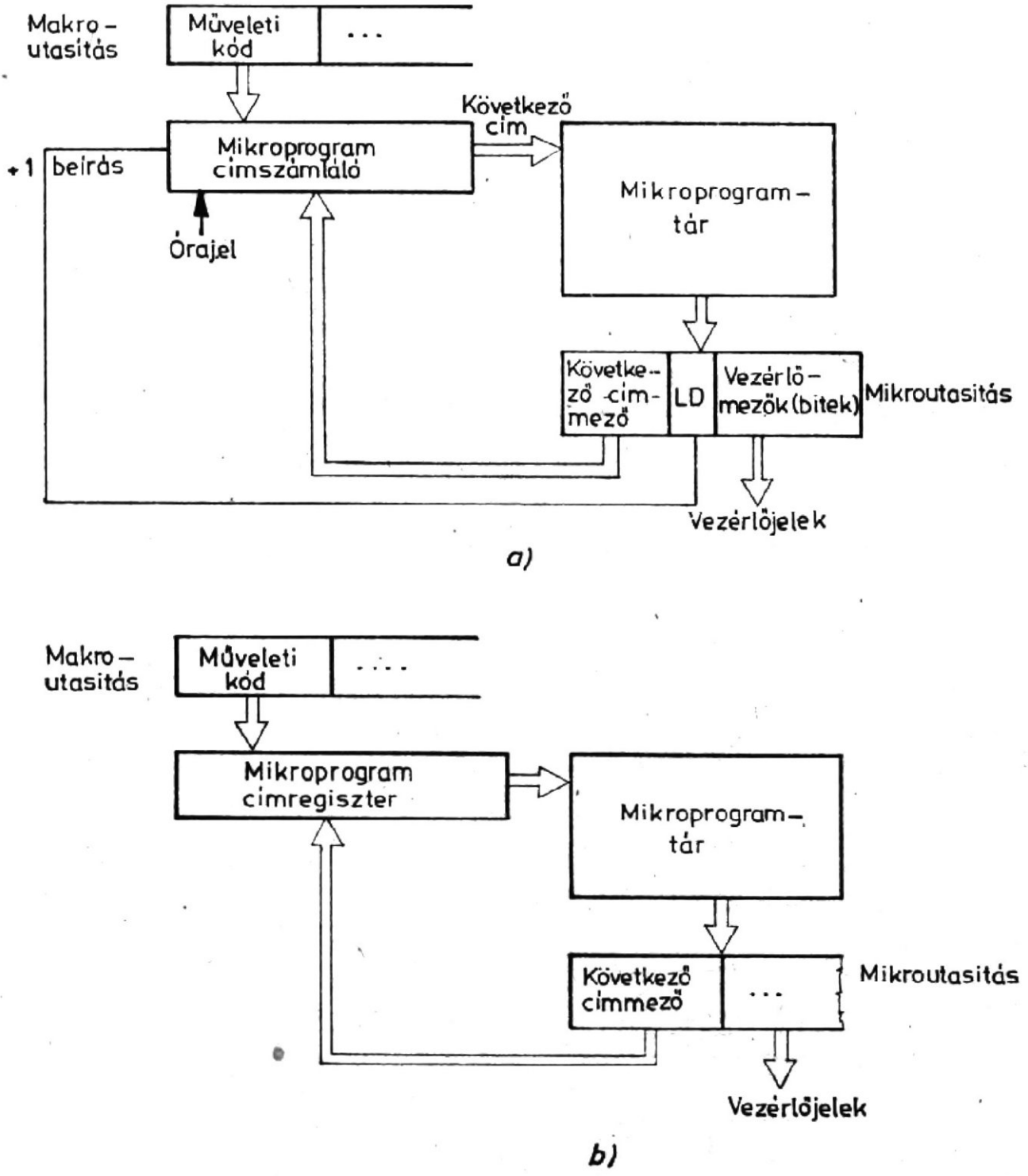
szüksége (l. később). Ha a mikroprogramozás célja az adott feladatnak megfelelő mikroprogram elkészítése, „statikus mikroprogramozhatóságról" beszélünk. Ez esetben az elkészített mikroprogram általában PROM mikroprogramtárba kerül. Dinamikusan mikroprogramozható az a rendszer, amelyben a mikroprogram a rendszer használata közben változtatható, a mikroprogramtár tartalma (vagy legalábbis egy része) a program vezérlése alatt újra tölthető. Ekkor a mikroprogramtárnak RAM típusúnak kell lennie, és gondoskodni kell a dinamikus újratöltés (módosítás) hardver–szoftver eszközeiről is (megfelelő adatutak, tárak, vezérlőprogramok stb.). Ez a vezérlőtár, a WCS (Writable Control Store: írható vezérlőtár).

A következő mikroutasítás címének meghatározásához a következő kétféle címképzési módszert szokták használni:

1. mikroprogramcím (utasítás-)-számláló alkalmazása (11.4*a* ábra), amelynek tartalmát minden egyes végrehajtott mikroutasítás után eggyel megnöveljük. Ez a megoldás a 11.5. ábra szerint kiegészített formában egyszerű feltételes elágazás megvalósítására is képes. Az LD jelű vezérlőbit értékétől függően a címszámláló tartalma vagy megnövelésre kerül, vagy a következő címmező tartalma módosítja a címszámláló tartalmát. Ennek egy változataként a vezérlőbit helyett alkalmazható egy teljes, n bit széles vezérlőmező is, amely 2^n-féle feltétel szerinti elágazást biztosít.

2. mikroprogramcím-regiszter alkalmazása (11.4*b* ábra), amelynek új tartalma (módosítása, ill. módosításának módja) az éppen végrehajtott, aktuális mikroutasítás egy mezejéből kapható. Ez egyben feltétel nélküli ugrási utasítások megvalósítását is jelenti, mivel a mikroprogram végrehajtása lényegében ilyen műveletek sorozataként valósul meg.

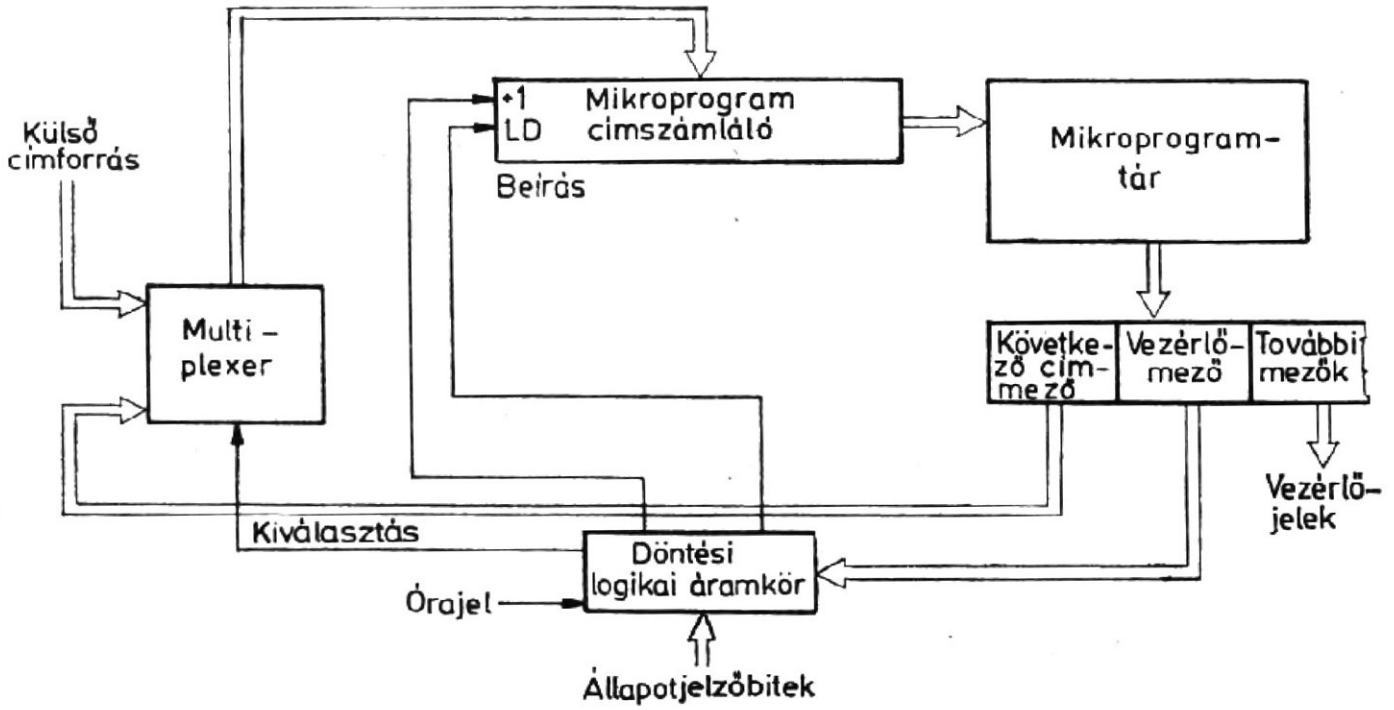
A mikroprogram végrehajtásának menetét külső feltételek alapján elágazások (ugrások) megvalósításával módosíthatjuk. A 11.5. ábrán láthatóan a döntési logikai áramkör megvizsgálja a feldolgozórészből származó állapotjelző-biteket, a mikroutasítás megfelelő vezérlőmezőjének bitjeit, és ebből a következő mikroutasítás



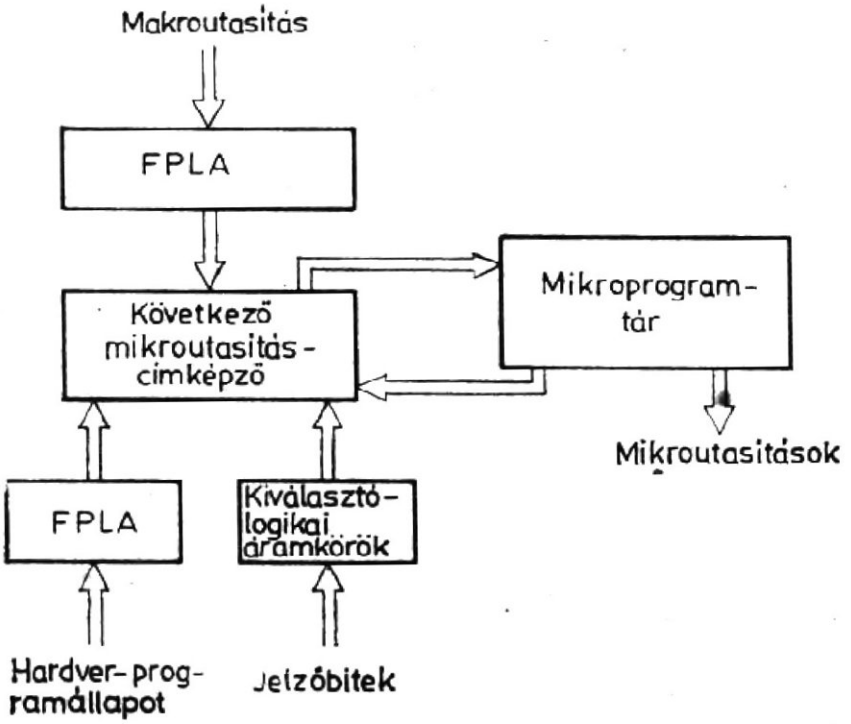
11.4. ÁBRA. Következő címképzés *a)* **mikroprogram címszámláló alkalmazása,** *b)* **mikroprogram címregiszter alkalmazása**

címére vonatkozóan az adott feltétel teljesülésétől függő döntést hoz. Vagy sorrendben a következő mikroutasítást veszi, vagy az adott mikroutasítás következő címmezője szerinti ugrást hajtja végre, vagy pedig a külső forrásból származó (pl. makroutasítás műveleti kódjának megfelelő) címmel cseréli ki a mikroprogramcím-számláló tartalmát.

Az FPLA elemeket (l. a 2. fejezetet) olyan PROM-nak is tekinthetjük, amelynek a címe (címdekódolója) is programozható, s ezek igen hatékonyan használhatók a mikroprogramozott vezérlőkben makroutasítás dekódolására, vagyis minden olyan esetben, amikor egy hosszú kódszó lehetséges kombinációi közül csak viszonylag kevés szó fordul elő. A 11.6. ábra egy olyan elrendezést mutat be, ahol FPLA elemeket alkalmazunk makroutasítás és programállapot-feltételek dekódolására.

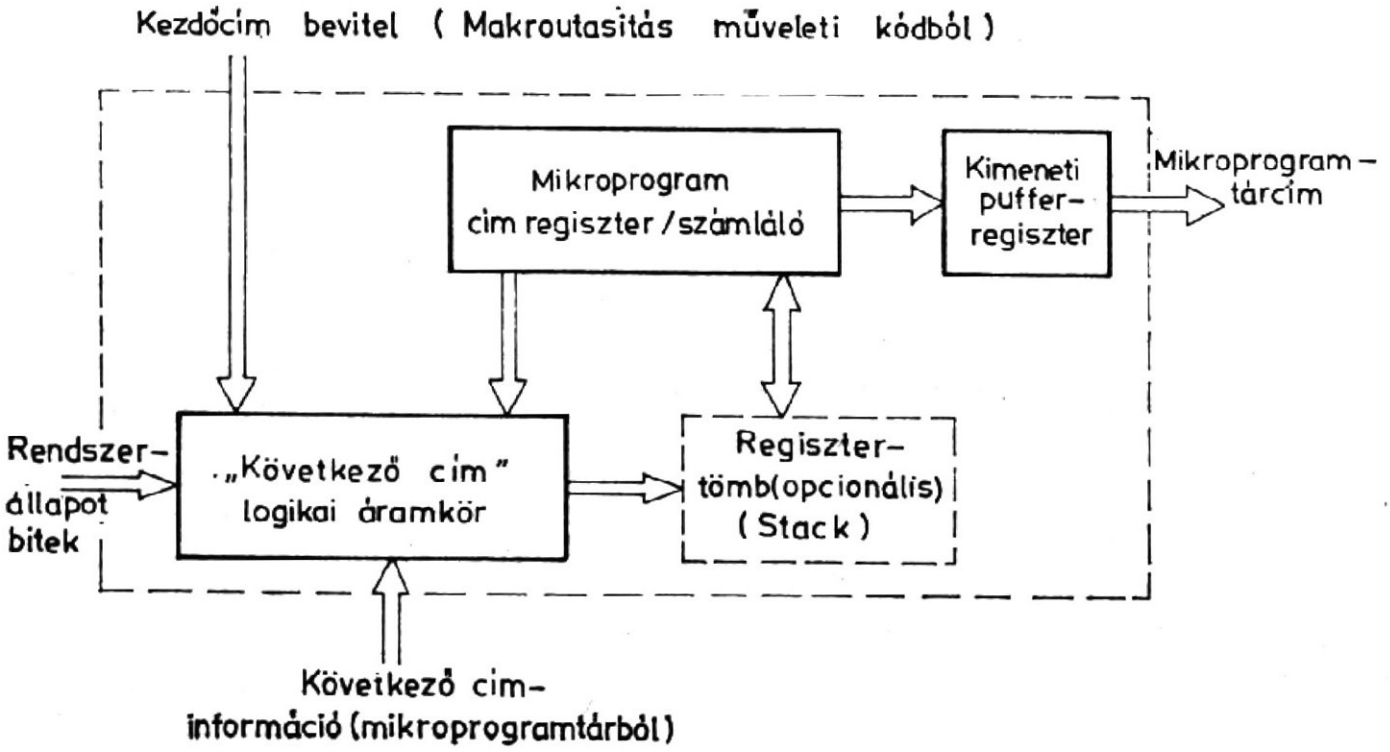


11.5. ÁBRA. Elágaztatás megvalósítása mikroprogram címszámlálóval

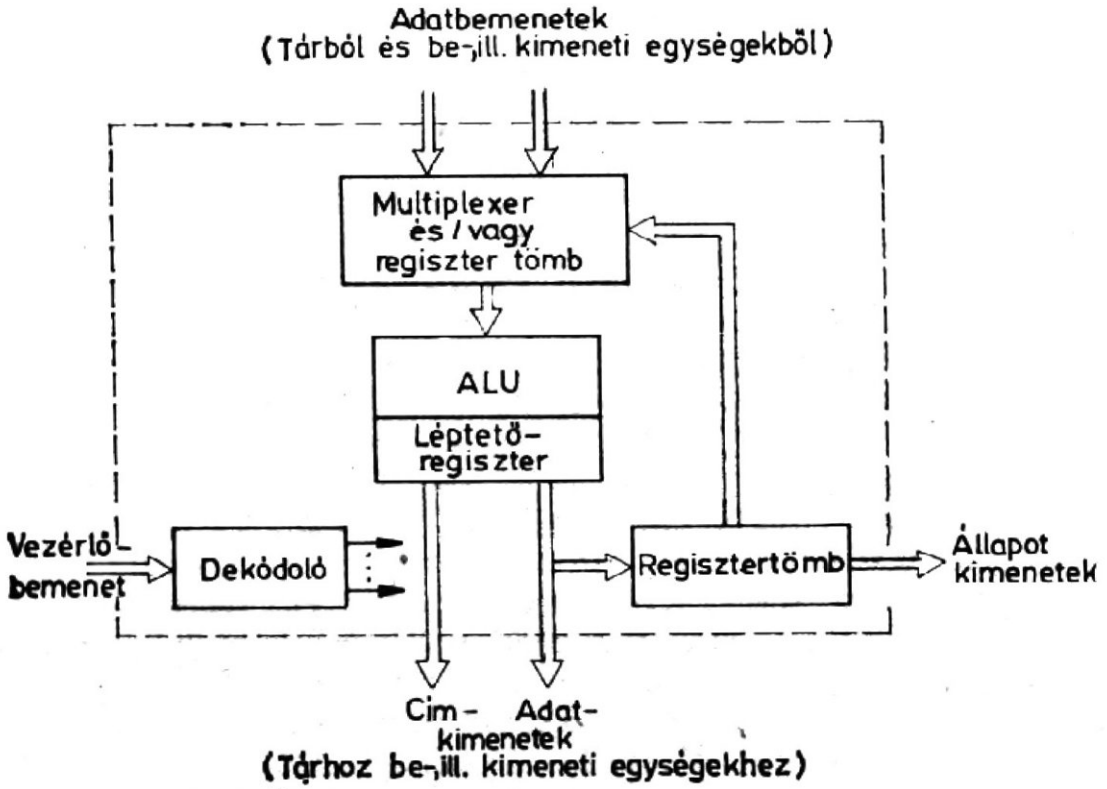


11.6. ÁBRA. FPLA elemek alkalmazása dekódolásra

Nagy számú, különböző LSI következő mikroutasítás-képző és bitszeletelt feldolgozó- (RALU) elemet gyártanak [1], [2], [3] [4], amelyeket a „családelvnek" megfelelően különböző interfészelemekkel is kiegészítenek. A következő mikroutasítás-képző LSI elemek, amelyek blokkvázlatát a 11.7. ábrán mutatjuk be, két alapcsoportba sorolhatók. Az elsőbe tartozók adott címtartományt (tárhelyet) képesek kezelni, míg a második csoport elemei sorba kapcsolható bitszeletelt áramkörök. Az opcionálisnak feltüntetett veremtár adott – általában négy – mélységig képes cím-



11.7. ÁBRA. Tipikus következő címképző LSI elem blokkdiagramja



11.8. ÁBRA. Tipikus RALU szelet blokkvázlata

megőrzésre és visszatárolásra. A LIFO szervezésű veremtár a hurokban használt mikroprogramoknak, mikroszubrutinoknak a futtatását és a megszakításkezelést segíti elő. (A veremtárba való be-, ill. az onnan való kivitelt **PUSH** és **POP** utasítások vezérlik.)

A feldolgozórész funkciót megvalósító RALU szelet, amelynek blokkvázlatát a 11.8. ábrán láthatjuk, szokásosan nem funkcionális módon megosztott elem, hanem 2, 4, 8 bitnek megfelelő szelet, amely mind az ALU-ból, mind a regiszterekből a bitszámnak megfelelő részt valósítja meg.

11.2. MIKROPROGRAMOZOTT RENDSZEREK FEJLESZTÉSE

Egy mikroprogramozott rendszer kidolgozása, a mikroprogramozható LSI elemek alkalmazása sokkal összetettebb, bonyolultabb feladat, mint a fix utasításrendszerű mikroprocesszorelem-családok alkalmazása, azokkal való rendszer kialakítása. A különbség és így a nehézség forrása egyrészt az elemkészletek tulajdonságainak következménye. Az utasítások alacsonyabb szintje több lehetőséget ad a tervező számára, de ugyanakkor sokkal több lehetőség közüli választást is jelent. A mikroutasítások (mikrooperációk) és az azokat közvetlenül végrehajtó hardvererőforrások felépítésének kölcsönhatása miatt a szoftver–hardver- mikroprogram csak egyszerre, együtt tervezhető, az egyes részfeladatok megoldása során a megvalósítás legmegfelelőbb formáját csak az összes követelmények és lehetőségek mérlegelésével, összehasonlító elemzéssel (trade-off) lehet megtalálni. Ezenkívül a mikroprogram-készítés (Firmware Engineering: a mikroprogramozás tudománya, ill. technikája) sokkal bonyolultabb [12], mint a hagyományos programírás (Software Engineering: programozás tudománya, ill. technikája) [11].

A mikroprogramozott rendszer kidolgozása a következő fő részfeladatokra bontható [12], [13], [14]:

1. Tervezés és specifikálás.
2. A hardverrendszer kidolgozása.
3. A mikroprogram elkészítése.
4. A rendszer, ill. a mikroprogram verifikálása, ellenőrzése és a hibák kijavítása.
5. Karbantartás.

A tervezés és specifikálás

Ez az első lépés, amely azonban számos, ismét csak erősen összefüggő részfeladat megoldását jelenti. E fázisban kell dönteni a hardver részletes, blokkvázlat szintű kidolgozásával egyidejűleg a kapcsolódó mikroutasítások felépítéséről (l. az előző architektúrával foglalkozó részt), a mikroutasítás egyes mezőinek feladatáról, ill. az egyes mezőkhöz rendelt mikrooperációkról. Figyelembe kell venni a konkrét implementációhoz választott báziselemek visszahatását, mivel mind a hardver-blokkvázlat, mind a mikroutasítás-struktúra kialakításával kapcsolatosan tulajdonképpen most kell mérlegelnünk azt, hogy a rendelkezésre álló elemkészletek [1], [2], [3], [4] közül melyik a feladathoz legjobban illeszkedő és leggazdaságosabb. Ez a választás sokszor nem egyszerű feladat. A legfontosabb szempontok a megvalósítandó makroutasításokhoz legjobban illeszkedő feldolgozóelem (ill. mikrooperációs készlet) megválasztása, a működési sebességviszonyok mérlegelése, mikroszubrutinok szükségessége és kezelésük, eszközeik kiválasztása stb. Gyakran a döntéshez az alternatívák valamilyen szintű kidolgozására (pl. a legfontosabbnak ítélt mikroprogram „mag" több elemkészlet alapján való elkészítése útján) is szükség van. Ez a benchmark (alappont) módszer [15].

A hardverrendszer kidolgozása

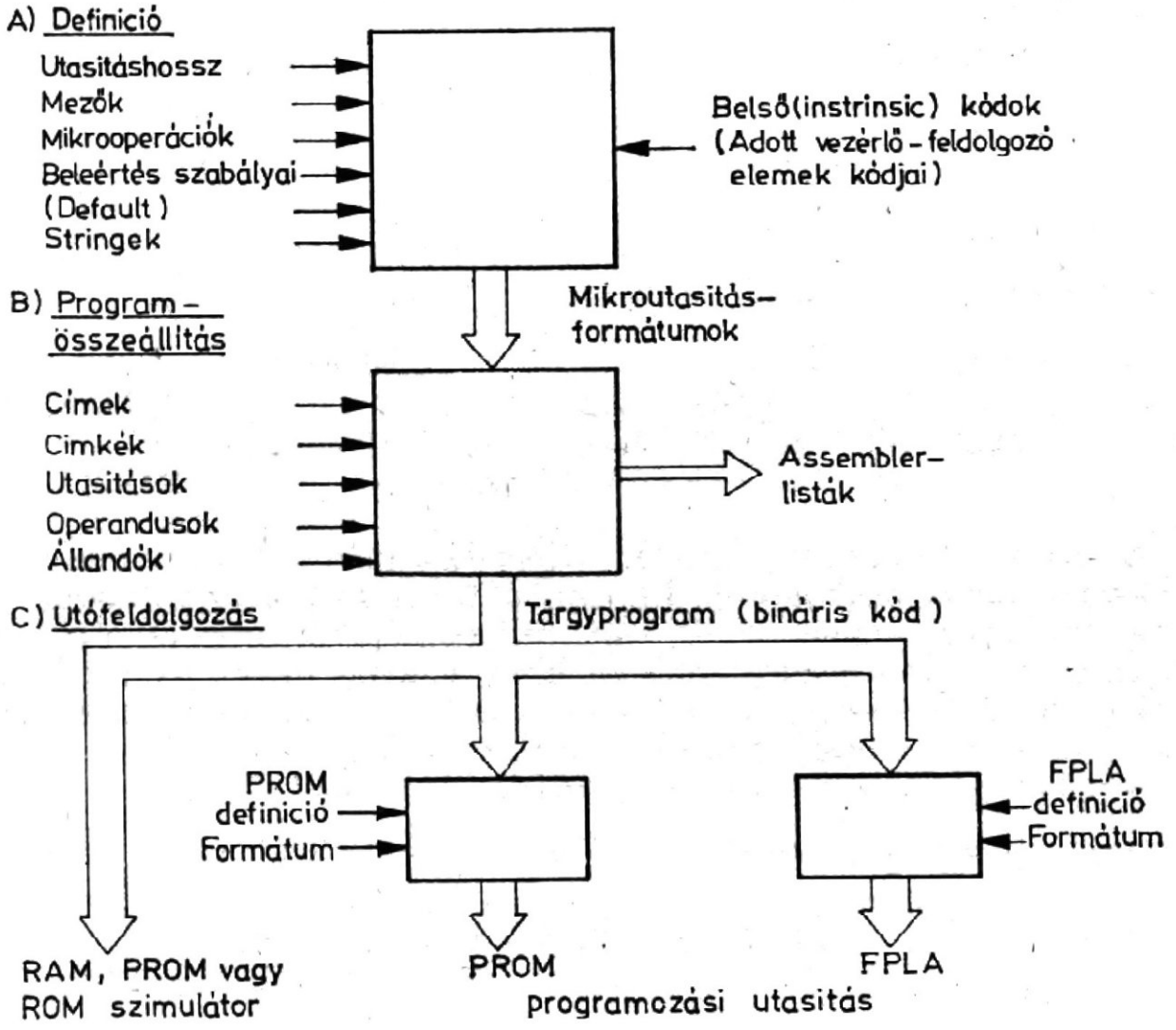
A tervezési fázis végeredménye a részletes hardverblokkvázlat és a kapcsolódó, pontosan definiált mikroutasítás. A *hardverrendszer* részletes kidolgozása ezt követi. A hardver kidolgozása során gondolni kell a rendszer (és ezen belül a mikroprogram) kipróbálása feltételeinek megtervezésére is. A kipróbáláshoz, a mikroprogram verifikálásához, ellenőrzéséhez és kijavításához szinte kivétel nélkül szükség van valamilyen külső eszközre, amelyet csak ezekben a fázisokban csatlakoztatunk mikroprogramo-

zott rendszerünkhöz. Így a hardverrendszer kidolgozásakor ezen eszköz (l. később) csatlakoztatásának lehetőségéről is gondoskodnunk kell, ezzel egyidejűleg tulajdonképpen megtervezve az ellenőrzés és javítás teljes folyamatát is [16].

A mikroprogram elkészítése

Ezután kerülhet sor a legfontosabb feladatra, a mikroprogram megírására. Bár jelentős munka folyik jól használható, rendszerfüggetlen mikroprogram írásra alkalmas magas szintű nyelv megvalósítására [5], [12], ma gyakorlati szempontból csak a különféle mikroprogram assemblerekre [17] támaszkodhatunk. Az assembler programok szinte kivétel nélkül magas szintű nyelven (pl. FORTRAN) készültek, így nagyon könnyen használhatjuk őket a számunkra rendelkezésre álló számítógép (kis- vagy nagy számítógép) vagy szolgáltatás (osztott idejű számítógép terminálja) segítségével. A forrásnyelvű mikroprogram-íráshoz ugyancsak jó szolgálatot tesz a választott rendszer körülményei között rendelkezésre álló szerkesztőprogram, ill. a különböző fázisú mikroprogramok tárolásához a file-kezelő rendszer (l. a 8. fejezetet).

A mikroprogram első részében (11.9. ábra) definiálni kell a mikroutasítást, és minden olyan információt meg kell adni, amelyre a program összeállításához szükség van. Az egyes specifikus paramétereket kulcsszavakkal lehet megadni, amelyek fenntartott szavak, így más célra (pl. valamilyen azonosítóként) nem használhatók. A korszerű assemblerek szabad formátumuak, ami azt jelenti, hogy a megadás sorrendjére, az egyes jellemzők megadásának soron belüli helyére vagy nincsenek kor-



11.9. ÁBRA. A mikroprogram-készítés folyamata

látozások, vagy csak nagyon egyszerű szabályok (pl. adott elválasztó karakterek) vannak. A specifikációs rész a következőket tartalmazza [18], [19], [20]:

- a mikroutasítás bitszámát,
- minden mező nevét,
- a mezők méretét (bitszámát),
- a mezőkhöz tartozó, azok adott értékét jelentő emlékeztető szimbólumokat (mikrooperációk),
- a mezőhöz rendelendő értéket, ha az nem szerepel a megadásnál (ez a default: elmaradás),
- adott típusszámú elemekhez tartozó mikrooperációk használatát (ez az intrinsic: belső mikrooperáció),
- adott mikrooperációkhoz rendelt más mezők mikrooperációinak implikációit, (beleértési) szabályait.

Pl. a mikroutasítást a következő formában, az adott fenntartott szavak segítségével adhatjuk meg [19]:

Utasítás	*Szélesség*		*Megjegyzés*
INSTRUCTION	**WIDTH**	48 *Szám* WORD SIZE *Szimbólum* START+1 *Kifejezés*	

A szimbólumoknak előzőleg definiált értékűeknek kell lenniük. A szám lehet: decimális (D-t vagy semmit se írva a szám után), hexadecimális (H), oktális (O), vagy bináris (B).

A mikroutasítás definíciónak tartalmaznia kell az összes mező definícióit és azt az

Utasítás vége	*Megjegyzés*
END INSTRUCTION	

kell, hogy lezárja.

Egy mező definíciónak a formája a következő:

Mező	*Mezőnév*	*Szélesség*		*Hiányzó érték*		*Megjegyzés*
FIELD	ADDR	**WIDTH**	0FH *Szám* BUFL *Szimbólum* 2+OPR *Kifejezés*	**DEFAULT**	00004 *Szám* NO OP *Szimbólum* START+1 *Kifejezés*	

A mezők előbbiekben ismertetett definiálása tulajdonképpen elég ahhoz, hogy a mikroprogramot megírhassuk. Egyszerűen felsoroljuk az egyes mezők numerikus értékeit. Ez azonban mind az írás, mind pedig a későbbi „olvasás" (dokumentálás)-szintjén nagyon kényelmetlen módszer. Ezért az egyes mezők adott értékeit mikrooperációként definiálhatjuk, és mikroprogram írásakor ezeket az emlékeztető szimbólumokat használhatjuk a numerikus értékek helyett.

A mikrooperációk a következő formában definiálhatók:

Mikrooperációs kulcsszó	*Mikrooperáció név*	*Kijelölés*	*Mezőnév*		*Megjegyzés*
MICROP	ADI	ASSIGN	BUS	21H *Szám* WRITE *Szimbólum* READ-TRUE *Kifejezés*	

A belső kódcsoport valamilyen adott eleme a gyártó által definiált (a microassemblerbe „beépített") mikrooperációk – definíció nélküli – használatát biztosítja.
Pl.:

Belső kulcsszó	*Elemtípus*	*Megjegyzés*
INTRINSIC	2901	

Lehetőség van az egyes mikrooperációkhoz olyan mezőértékeket hozzárendelni, amelyek más mezőkhöz tartoznak. Ebben az esetben ezek megadása automatikusan implikálja a hozzárendelt mezőértékeket is.
Pl.:

Mikrooperáció	*Kulcsszó implikáció*	*Mezőnevek és értékek*	*Megjegyzés*
RUN	**IMPLY** FI = 2	MEM = 0AH KB = 0FH	

Az implikáció „magasabb hierarchikus szintű" specifikációval hatálytalanítható. Az egyes szintek – csökkenő értékkel – a következők [18]:

1. Explicit megadás (mikrooperáció vagy mezőnévvel megadott számérték).
2. **IMPLY**: implikációmegadás.
3. **FIELD**: hiányzó érték megadása a meződeffinícióban.

A definíciókat a mikroprogram specifikációs része követi.

Ez a kezdősortól

Kulcsszó	*Programnév*	*Szélesség*	*Bitszám*	*Hossz*	*Szószám*
PROGRAM	8080 EMULÁTOR	WIDTH	48	LENGHT	512

a terminátorig:

Vége	*Megjegyzés*
END	

tart.

A mikroutasítások specifikációs formája a következő:

Címke	*Elválasztó karakter*	*Mezőnév (Szimbólum)*	*Mezőnév (Számérték)*	*Mezőnév (Kifejezés)*
FETCH	:	F = NOP	MEM = 0BH	REG = R1+1

Mikrooperációk	*Megjegyzés*
NEXT CODE	

Összetartozó mikrooperációk karaktersorozatként is identifikálhatók és használhatók a specifikációkban. Pl.:

Karaktersor-azonosító	*Védett szó*	*Mikrooperációk*
DONE	STRING	'READ REN MASK,

String zárójelek

Értéket a következő formában rendelhetünk szimbólumokhoz (azonosítókhoz):

Szimbólum	*Értékkulcsszó*	
MASK 2	VALUE	OFH *Szám* NOT MASK 1 AND 0FFH *Kifejezés*

Mint az előző példa is mutatja, a kifejezésben logikai műveletek is megengedettek.

További bővítési lehetőség az egyes mikrooperációkhoz argumentumok rendelése. Ily módon összetett mikrooperációkat hozhatunk létre, amelyek pl. egy feldolgozóelem műveletének és műveletben részt vevő regisztereknek a kijelölését egyszerűsítik. Pl. a definíciót a következő formában adhatja meg:

Védett szó	*Mikrooperációnév*	*Argumentumok*
MICROP	MOVE	(REG SOURCE, REG DEST)

Védett szó	*Hozzárendelés*
ASSIGN	OP = 20H SOURCE = REG SOURCE DEST = REG DEST;

Ha egy mikroutasításban használjuk a **MOVE** mikrooperációt a következő formában:

MOVE (2,11) akkor az a megadott argumentumok sorrendjében rendeli hozzá az értékeket vagyis OP = 20H, SOURCE = 2 és DEST = 11.

Zárójelbe tett számmal, szimbolikus névvel vagy kifejezéssel egy mikroutasításhoz közvetlenül címet rendelhetünk. Pl.:

(80H): címke...

(ADDR): címke...

(ADDR+40H): címke...

A mikroassemblerek még számos összetett programozási lehetőséget is biztosítanak [18], [19], [20] (pl. feltételes mező-hozzárendelés vagy mikrooperációra való hivatkozás, mezőrészek definiálása, többszörös mikroutasítás-formátum használata, mikrooperációk egymásba skatulyázása stb.), amelyekre itt most nem térünk ki.

A mikroassembler két kimenetet ad: listakimeneteket (a mikroprogram dokumentációjaként és a további ellenőrzési, hibajavítási munkához) és egy tárgyprogramot, vagyis a lefordított mikroprogramot egy közbülső formában, ami még utólagos feldolgozásra, átrendezésre kerül.

A listák közül szokásosan az első a forrásprogramot és a hozzá tartozó lefordított tárgyprogramot tartalmazza; utóbbit az egyes mezőkhöz rendelt, jól áttekinthető formában. A második egy keresztreferencia-lista (Cross Reference Listing), amely a szimbólumok nevét, típusát (címke, programnév, mezőnév, mikrooperációnév, argumentumnév), azok definiálásának helyét (forrásprogram-sorszám), és a hivatkozások helyét (forráslista-sorszámok) adja meg.

Az assemblerfordító a közbenső tárgyprogramfile-t valamilyen választható – pl. F, P, I, D vagy E rekord [19] – formátumban adja ki.

A mikroassemblerek működését vezérlő pseudoutasítások és a jelzett hibák típusa és a hibajelzés módja a szokásos assembler programokhoz (l. a 9. fejezetet) hasonló.

A rendszer, ill. a mikroprogram verifikálása

Mielőtt a mikroprogramot a végleges formának megfelelően „beégetnénk" a PROMba és az FPLA elemekbe, el kell végeznünk annak kipróbálását, ill. a hibákat ki kell javítanunk. Sok esetben jól használhatunk ezekben a fejlesztési fázisokban egy erre a célra kidolgozott – lehetőleg interaktív – szimulátorprogramot (l. a 10.4. szakaszt). Az esetek többségében (amikor a szimulátor elkészítése a sok speciális mikroprogrammal vezérelt rész miatt nagyon nagy feladat, vagy a külső feltételek nehezen szimulálhatók és a tényleges működési sebességgel is összefüggenek) az egyetlen lehetséges megoldás a mikroprogramozott rendszer kiegészítése – az ellenőr-

zés és hibajavítás idejére – valamilyen külső, támogató rendszerrel. Adott típusú következő mikroprogramcím-képző elemekhez létrehoztak a felhasználói rendszeren belül használható emulátorokat [7], [21], ezek azonban csak korlátozott lehetőségeket biztosítanak, mivel feladatuk csak a címképzésben van.

Legjobban használható megoldást valamilyen „intelligens" elemet (tehát processzort) tartalmazó rendszer és a fejlesztés alatt álló mikroprogramozott rendszer közötti interfész létrehozása jelenti. A támogatást nyújtó rendszer lehet valamilyen mikro- (mini- vagy kis-) számítógép, amelynek van hardverje (operatív és háttértár, ember–számítógép kapcsolat) és szoftverje (operációs rendszer, l. a 8. fejezetet).

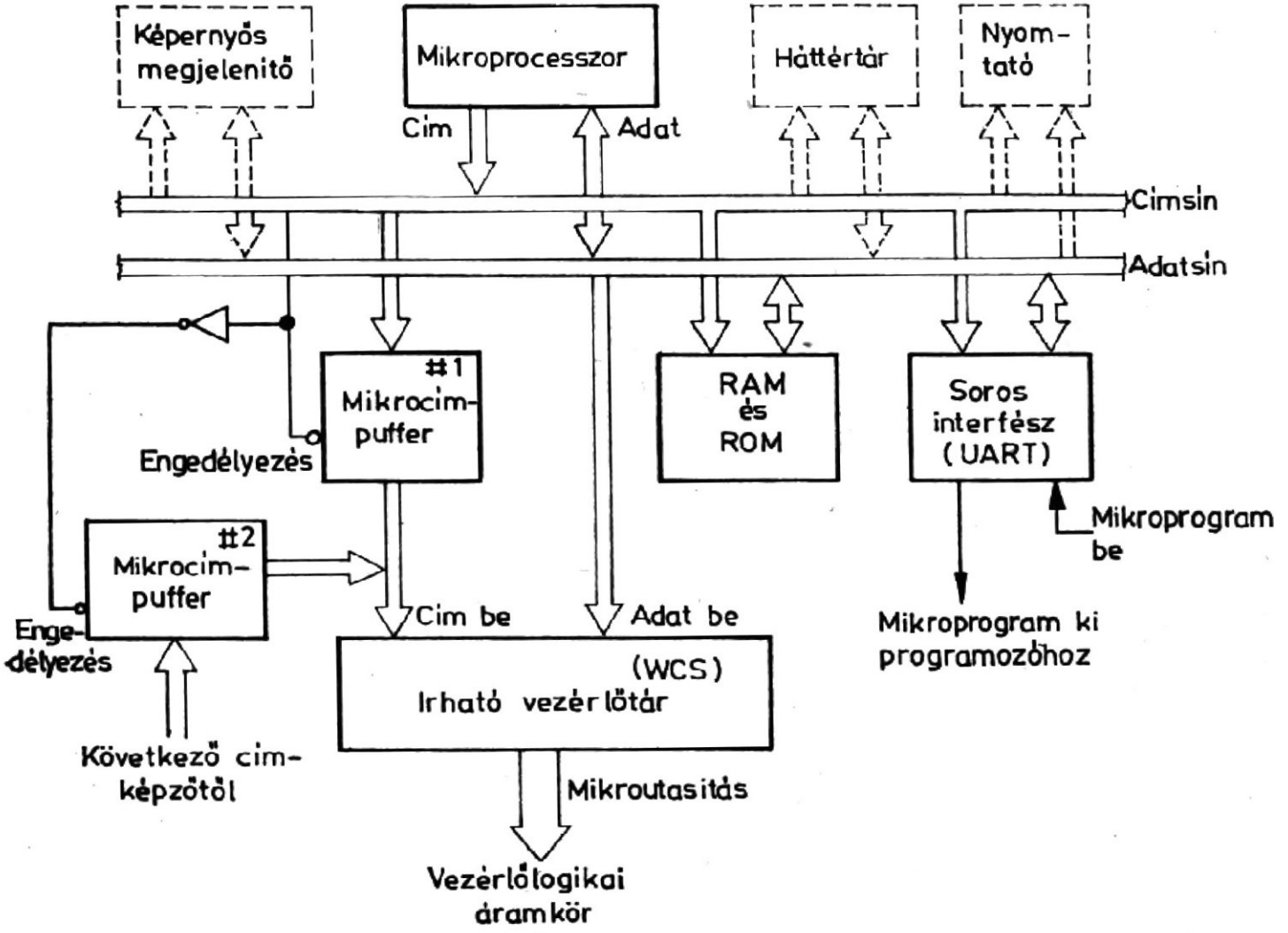
Ezt a számítógéprendszert a 11.10. ábra szerint kiegészíthetjük a mikroprogramozott rendszer fejlesztéséhez szükséges alapvető hardvermodulokkal. A mikroprogram (vagy annak külön vizsgálható része) az ellenőrzés–hibakeresés és javítás idejére egy megfelelő szóhosszúságú és hozzáférési idejű WCS írható vezérlőtárába kerül. Abban az esetben, ha ugyanezt a rendszert használjuk a program összeállítását végző program (mikroassembler) futtatására, a lefordított program már a háttértár (pl. hajlékony mágneslemez) tárgyprogramfile-ban van, és innen tölthető át a WCS-be ellenőrzött futtatás céljából. Erre a célra szükségünk van egy mikroprogrambetöltő programra. Ez a program végzi a mikroprogrammoduloknak a betöltését a vezérlőtár adott címtartományába. Így módunk van ennek részenkénti cserélésére is. A 11.10. ábra szerint kétféle forrásból (a címsín egy bitjének segítségével kiválasztott módon) lehet címezni, futtatás esetén a vizsgált rendszer következő címképzőjéből, és feltöltés, ellenőrzés, módosítás esetén pedig a rendszer belső címsínjének a felhasználásával. Ha az írható vezértármodul pl. a 11.11. ábra szerinti felépítésű, a belső sínnel való kapcsolat 8 bitenként (byte-onként) tesz lehetővé műveletet, amely jól illeszthető a mikroutasítás felépítéséhez (mezőfelosztásához) is.

Az ellenőrzött mikroprogram-futtatást és a szükséges módosításokat a támogatást végző számítógép (operációs rendszere alatt futtatható) speciális, erre a célra készített programja segítségével végezhetjük. Egy ilyen program, amelyet szoktak megfigyelő- (monitor), nyomozó- (tracer), hibajavító (debugger) programnak is nevezni, fő feladatai a következők:

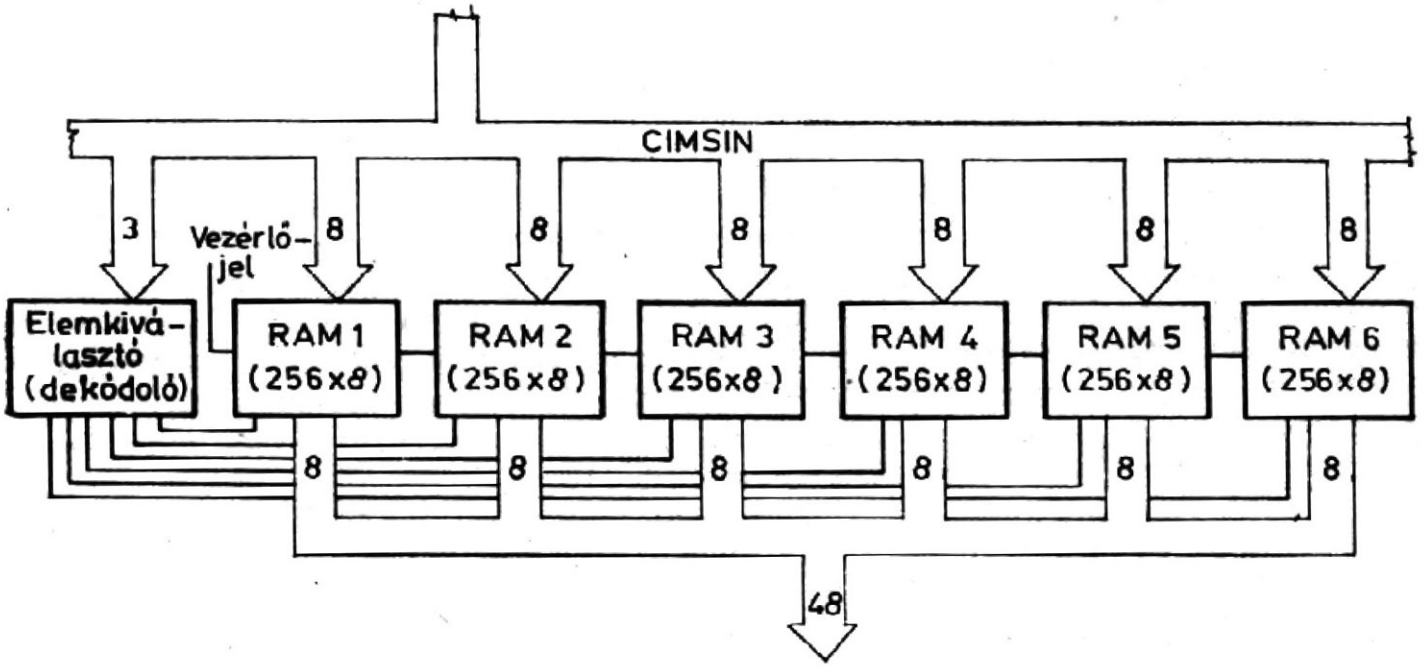
- a mikroprogramozható rendszer erőforrásai (pl. regiszterek, vezérlőtár, feltétel flip-flopok stb.) állapotának a kijelzése (megjelenítése),
- a megfelelő rendszerrészek (pl. a regiszterek, vezérlőtár stb.) tartalmának módosítása,
- előre definiálható pontokban, a töréspontokban a futás megállítása (és a mikrovezérelt rendszer állapotának az ellenőrzése).

A program által megoldandó feladatok ugyan nagyon egyszerűnek látszanak, de mivel a mikroprogramozott rendszer belső erőforrásaihoz közvetlenül nem tudunk hozzáférni, a programot részben a fejlesztett rendszer saját mikrokódja felhasználásával tulajdonképpen mikroprogram formájában, részben pedig a fejlesztéshez használt számítógép assembler nyelvén kell megírni. Azért, hogy fogalmat alkothassunk egy ilyen hibajavító program által nyújtott szolgáltatásokról, röviden összefoglaljuk egy adott rendszer [20] programellenőrzési parancsait.

Egy töréspontón a program megállítja a mikroprogram futását, a mikroprogramozott rendszer állapotát (regiszterek, állapotjelzők stb. tartalma), egy ún. mikroállapot-táblázatban (microstate table) megőrzi, és mielőtt a futást újra elindítaná, az egyes erőforrásokat ebből a táblázatból ismét visszatölti. A parancsokat védett szavakkal: kulcsszavakkal adhatjuk meg, amelyek adott szintaxisnak megfelelő-eleket használnak.



11.10. ÁBRA. A mikroszámítógépes mikroprogramozott rendszer fejlesztését támogató rendszer blokkvázlata



11.11. ÁBRA. Irható vezérlőtár (WCS) felépítése

A parancssor felépítése a következő:

$$\left\{módosító\right\}_0^1 \qquad \left\{egész\ szám\right\}_0^1 \qquad \text{parancs} \left\{címspecifikáció\right\}_0^1$$

A szokásoknak megfelelően a kapcsos zárójelben levő részeket a parancs típusától függően vagy meg kell adnunk, vagy elhagyjuk, de természetesen parancsnak mindig lennie kell egy parancssorban.

A **címspecifikáció** vonatkozhat:
főtárcímre # *n*
mikrotárcímre *n*
regiszterre $*n*
és kiegészíthető bittartomány-megjelöléssel is:

< bal oldali bit : jobb oldali bit >

A **módosító** a parancs értelmét a következő jelekkel változtathatja:
— az értelmét megfordítja (pl. töréspont-beállítás helyett törlés),
? a parancsra vonatkozó (pl. töréspont) lista kiadását jelzi.

Az **egész szám** (integer) mező a címzés egy lehetséges alternatív megoldása, amely a későbbi ismételt hivatkozást egyszerűsíti. A programozó (vagy a program) által hozzárendelt egész szám formájú azonosító segítségével hivatkozhatunk ezután pl. egy definiált töréspontra.

A **parancsok** a következők:

■ Nyitóparancsok (Open)

Az **O** (és megfelelő címspecifikációval kiegészített) parancsra a program megismétli a címet (ha nem adtunk meg bittartományt, az illető elemre jellemző teljes bittartománnyal kiegészítve), és azt követően az illető címhez tartozó tartalmat kiírja és vár.
Pl.:

> **O** # 2034
2034 < 15:0 > 14332 (vár)

A programozó ekkor megadhat egy új tartalmat.

2034 < 15:0 > 14332 14322 (elválasztókarakter)

(Az oktális 14332 értékét 14322-re módosítottuk, vagyis a negyedik bitet 0-ra változtattuk.) Az elválasztókarakter (separátor) megválasztásával a programozó kétféle folytatást írhat elő:

1. ha szeparátorként a kocsi vissza (CR: Carriage Return) billentyűt üti le, a program újabb parancsot vár;
2. ha az új sor (LF: Line Feed) billentyűt üti le, a sorrendben következő címet nyitja meg.

Tehát pl.:

> **O** # 2034 (CR)
2034 < 15:0 > 14332 14333 (LF)
2036 < 15:0 > 1010 (LF)
2040 < 15:0 > 23322 (CR)
>

További lehetséges címmódosítások a szeparátor megválasztásával:
∧ : előző tárhely,
@: közvetett címzés (a tárhely tartalma a cím). Lehetőség van a mikroutasítás cím-bittartomány helyett megfelelő mezőnévvel is hivatkozni.

■ Töréspontparancsok:

Ezek segítségével a felhasználó töréspontokat állíthat be a főtárban vagy a vezérlőtárban (WCR), folytathatja a végrehajtást a töréspontról, szelektíven törölhet egyes töréspontokat, és kérheti a töréspontok listáját. A program a töréspontok jellemzőit egy – maximálisan 16 elemet tartalmazó – listában tartja. Ennek a listának az indexét automatikusan hozzárendeli azonosítóként az egyes töréspontokhoz (mindig a legkisebb még „üreset"), ha a felhasználó nem ad meg explicit formában ilyen jellemzőt.

A töréspont-beállítás parancsjele **B**, amely előtt opcionálisan megadható az azonosító, pl.:

> **B** # 10200
> 2**B**6420

A program futásának – az adott törésponttól való – folytatására a **P** (Proceed: továbbmenés) parancs segítségével adhatunk utasítást. A parancsjel előtt opcionális ismétlési számmal megadhatjuk, hogy az adott töréspontón hányszor haladjon át a program megállás nélkül.

Pl. a

> 200 **P**

a megálláshoz tartozó töréspontnak csak a 201. előfordulása esetén áll meg.

Töréspontot a listáról törölhetünk a **—B** paranccsal, amelyet vagy azonosítóval, vagy címinformációval kell kiegészíteni (ellenkező esetben az összes aktív töréspontot törli).

A töréspontlistát a következő parancs hatására kaphatjuk meg: **?B**

A lista fejléce a következő:

Töréspont száma	*Cím*	*Ismétlést szám*
BREAKPOINT NUMBER	ADRESS	REPEAT-COUNT

■ Megjelenítésparancsok:

E parancsok segítségével a felhasználó címspecifikációkat adhat (törölhet) egy listában, amelyet megjelenítéslistának hívnak. A megjelenítéslista hasonlít a töréspontlistához, ebben is 16 elem lehet, amelyek címspecifikációkat és hozzájuk tartozó értékeket tartalmaznak.

A listához újabb címet a **D** (add: hozzáadás) paranccsal rendelhetünk. A cím mellett megjelenítésazonosító számot is írhatunk a **D** elé (ha nem adunk azonosítót, a program rendeli hozzá a következő szabad értéket).

Listából való törlést a **—D** paranccsal (és hozzáadott azonosítóval vagy címmel) végezhetünk. (Azonosító nélkül az egész lista törlődik.)

Töréspont elérésekor a megjelenítéslista aktuális tartalma kiírásra kerül.

Pl. a

> $ 7 < 15:0 > 1012
> $ 2 < 15:0 > 5421
> $ PSW < 15:0 > 340
> 6430 < ALU > 12

lista a 7. és 2. regiszter, a PSW (Program Status Word: programállapotszó), valamint a 6430 mikroutasításszó ALU jelű mezőjének címét (azonosítóját) és tartalmát jelenti.

A ?D parancs hatására az egész megjelenítéslista kiírásra kerül, a következő fejléccel:

Azonosító szám	*Cím*	*Mező*
NUMBER	ADDRESS	FIELD

■ Vezérlőparancsok

A G parancs hatására a vizsgált program végrehajtása megkezdődik a megadott címről. Ennek a parancsnak a segítségével lehetőség van egy adott töréspontón való megállás után a végrehajtást egy másik helyen folytatni. A megadható cím vagy mikrocím (szám külön kiegészítés nélkül) vagy makrocím (# jel előzi meg a számot).

Az utóbbi esetben a címet át is helyezhetjük egy relokációregiszter megadásával, aminek aktuális tartalma ekkor előjelhelyesen hozzáadódik a megadott címhez.

A program a WCS vezérlőtár tartalmát megőrzi a főtár nem használt részében. Így lehetőség van az L (Load: betöltés) paranccsal visszahelyezni a vezérlőtár eredeti tartalmát.

Az R (Reset: alaphelyzet) parancs törli mind a töréspont-, mind a megjelenítésistákat.

Így az > R parancs egyenértékű a
> –B
> –D

parancsok együttes hatásával.

A mikroprogram ellenőrzése és kijavítása után most már végleges helyére a felhasználó által programozható PROM-ba akarjuk azt „égetni".

Ez a mikroprogram egyes részeinek az adott tárolóelemekhez való hozzárendelését teszi szükségessé. Más szóval szükség van egy olyan utófeldolgozó programra (l. a 11.9. ábrát), amely a mikroprogram-formátum és a PROM elemek definíciója alapján elkészíti a PROM (vagy FPLA) elemek programozóberendezésének vezérlőprogramját. A PROM (FPLA) tartalom beírást is valamilyen „intelligens" elemet (pl. mikroprocesszort) tartalmazó berendezés segítségével végezzük, amely rögtön ellenőrzi is a beírás folyamatát, ill. eredményét. A 11.10. ábra szerinti rendszer is illeszthető a programozóberendezéshez (pl. soros interfésszel). Ebben az esetben a teljes folyamat – beleértve most már a „beégetést" is – az operációs rendszer szolgáltatásainak (pl. file-kezelés, automatikus programbetöltés) felhasználásával végezhető. Lehetőség van az egyes tárolóelemekbe írt tartalom kilistázására is a megfelelő mikroutasításrész (mezőnév és bitpozíció) feltüntetésével, a csatlakoztatott nyomtató (pl. mátrixnyomtató) felhasználásával.

Az ismertetett feladatokat végző utófeldolgozó program [18] számára pl. a következő formában adhatjuk meg a PROM elemek és a mikroprogram jellemzőinek egymáshoz rendelését:

```
ROM 512 BY 8
WORDS 0 TO 511
BITS (CPE(6), CPE(5), CPE(4), CPE(3)
      CPE(2), CPE(1), CPE(0), FI(4)
```

Ahol a **ROM** kulcsszóval adjuk meg a tár méretét, szervezését (a **BY** előtt a szószám, mögötte a bitszám áll), a **WORDS** (szavak) adja meg a mikroprogramcímek hozzárendelését (a **TO** jelentése-ig), a **BITS** (bitek) után pedig zárójelben szerepelnek a bitsorrendnek megfelelően az egyes bitekhez rendelt mezőbitek (mezőnév

és zárójelben a mező bitszáma). Ily módon egyértelműen megadható minden egyes tárolóelemre vonatkozóan a függőleges (szavak szerinti) és vízszintes (bitek szerinti) hozzárendelés.

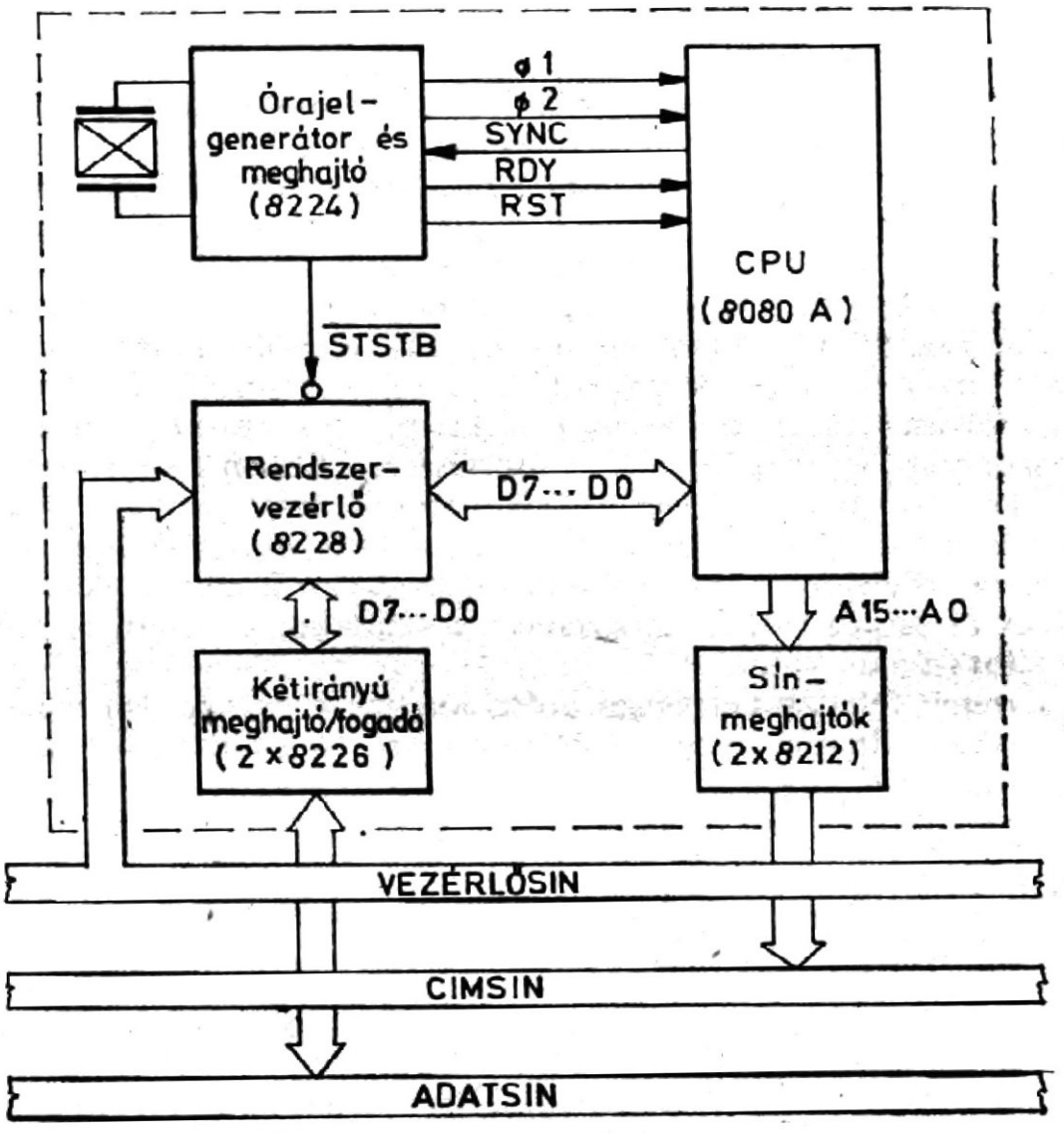
Karbantartás

Az utolsó fázis, a mikroprogram karbantartása (maintenance) a már egyszer lezárt rendszer mikroprogramjának a módosítását jelenti. A karbantartás két csoportra osztható [12]: javításra és korszerűsítésre. A módosítás a gyakorlatban a mikroprogramtárnak (ill. egyes elemeinek) – általában az alkalmazás helyén való – kicserélését, majd az egész rendszer újravizsgálatát (retest) jelenti.

11.3. PÉLDA EGY MIKROPROGRAMOZOTT RENDSZERRE

Példaként egy olyan kidolgozott rendszert [22] mutatunk be röviden, amelyet a mikroprogramozott rendszerek alkalmazásának egyik legérdekesebb területéről vettünk. Emellett ennek a példának a segítségével arra is lehetőségünk van, hogy érzékeljük, hogyan használhatjuk a mikroprogramozásban rejlő lehetőségeket feladatok megoldására, a rendszer különböző részegységeinek vezérlésére.

Az alapfeladat a 11.12. ábra szerinti 8080 alapú rendszer emulálása. Az emuláció, hasonlóan a szimulációhoz (l. a 10.4. szakaszt) a tulajdonságok, jellemzők



11.12. ÁBRA. Az emulált rendszer blokkvázlata

valamilyen fajta utánozását jelenti. Egy számítógéprendszer egyik jellemzője az utasításrendszere. Az utánozandó rendszert szokták cél- (target) rendszernek nevezni, míg a szimuláció/emuláció megvalósítása számára hardver/szoftver erőforrásokat nyújtó rendszert gazda- (host) rendszernek nevezik. Ha a cél az utasításrendszer utánozása, utasításszimulátorról vagy utasításemulátorról beszélünk. Szimuláció esetén szoftvereszközöket (tehát programot), míg emuláció esetén mikroprogramot használunk fel. Szimuláció esetén a megvalósítandó utasítások értelmezése számítógépprogram, míg emuláció esetén mikroprogram segítségével történik. Ha most már a két módszerrel elérhető eredmény jellemzőit hasonlítjuk össze, legfontosabb tulajdonságként a végrehajtási sebességet tekinthetjük. Általában a szimuláció segítségével való utasításrendszer megvalósítása esetén az eredeti utasításvégrehajtási időnél sokkal (egy-két nagyságrenddel) nagyobb időket kapunk, míg az emulátor sebessége az eredeti utasítás-végrehajtási időnél lényegesebb jobb is lehet. Az ok nyilvánvaló. Szimulátor megvalósításakor csak a gazdarendszer (a célrendszer utasításaitól általában nagymértékben eltérő) gépi utasításait használhatjuk, vagyis egy gépi utasítás megvalósításához sok gépi utasítás szükséges. Ezzel szemben az emulátor esetén magukat a célrendszer utasításait valósítjuk meg a mikroutasítások (és a segítségükkel vezérelt elemi hardvererőforrások) segítségével.

Az emuláció célja nem csupán az utasítások realizálása lehet, hanem a rendszer egyéb jellemzőinek (pl. rendszerinterfész jeleinek) megvalósítása is. Ebben az esetben rendszeremulációról beszélünk. Esetünkben is ez a helyzet, vagyis a mikroprogramozott gazdarendszer be-, ill. kimenő jeleinek, vagyis a cím-, adat- és vezérlősín jeleinek meg kell egyezniük a 8080 célrendszer azonos jeleivel. További kikötéseket is tehetnénk, pl. az időarányos (time-scaled) emuláció esetén az egyes utasításvégrehajtási idők aránya a két (host és target) rendszerben meg kell, hogy egyezzen egymással. Esetünkben ezt az utóbbit nem kívánjuk meg.

Intel 3000-es bitszeletelt elemcsalád felhasználásával megvalósított 8080 emulátor utasítás-végrehajtási ideje az eredeti 8080 család elemeivel megvalósított rendszer (11.12. ábrát) utasítás-végrehajtási időinek [15] – az utasítások típusától függően – fele-nyolcada [21].

A 8080 emulátor egyszerűsített blokkvázlatát a 11.13. ábrán mutatjuk be. Az emulátor a következő három fő részre osztható:

1. Mikrovezérlő egység.
2. Aritmetikai–logikai egység és regiszterek.
3. Be-, ill. kimenethez és a tárhoz tartozó interfész.

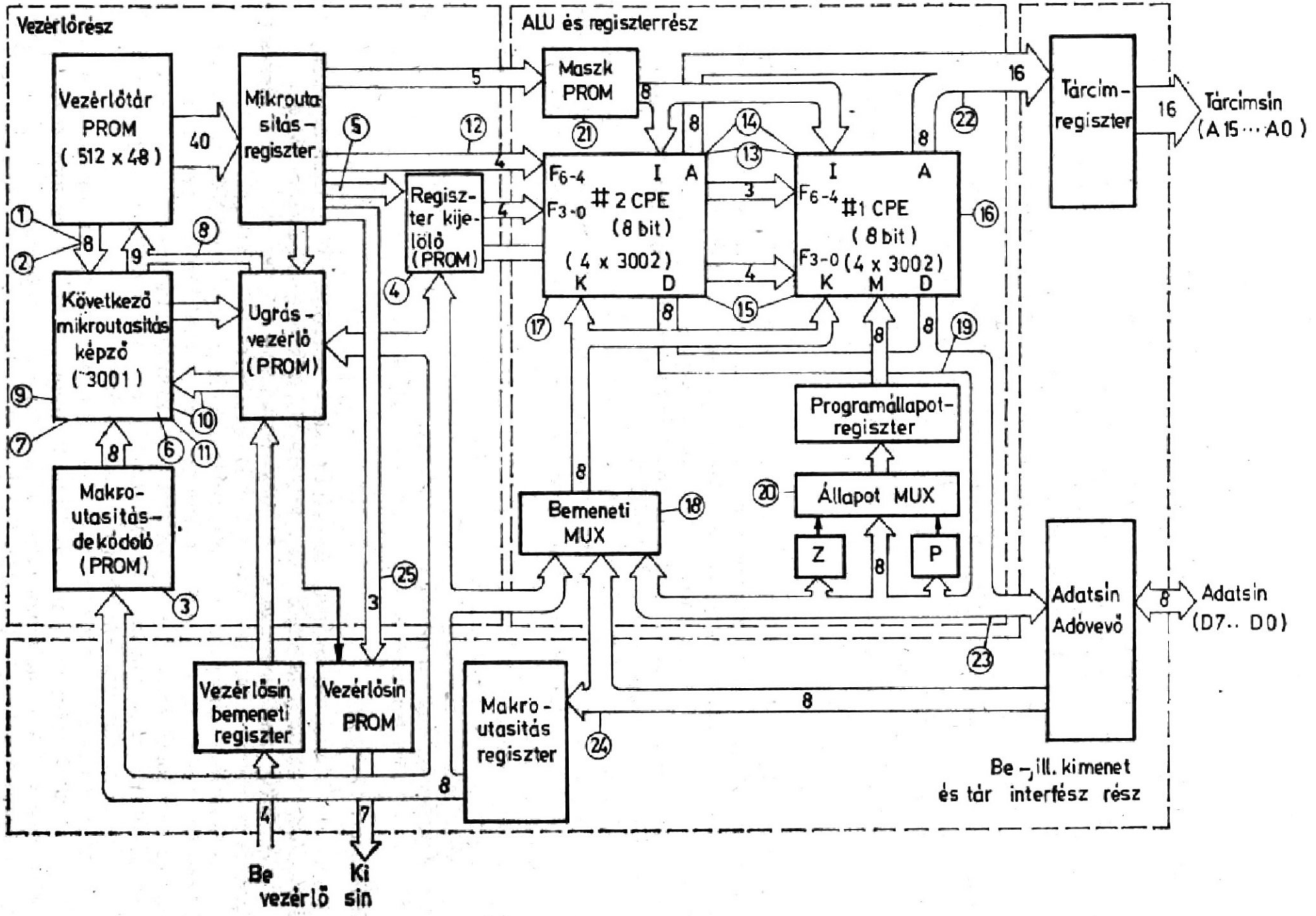
Mikrovezérlő egység

A mikrovezérlő egység biztosítja az egész rendszer vezérlését. A mikroutasítások (felépítésüket l. később) 48 bitesek, és a vezérlő 512 mikroutasítást képes tárolni.

(A 8080 utasításrendszerének megvalósításához – beleértve két kiegészítő szorzó- és osztóutasítást is – csak 345 mikroutasításra volt szükség.)

A vezérlőtár címzését a mikroutasítás-címképző végzi (egy 3001 típusú MCU: Micro Control Unit, azaz mikrovezérlő egység). A címképzéshez vagy a makroutasítás-dekódolóból származó kezdőcímet vagy a mikroutasítás erre a célra szolgáló mezőjéből származó mikrooperációt használja fel. Utóbbi esetben az ugásvezérlőből származnak az elágazás feltételei (feltételes elágazás).

A makroutasítás-dekódoló egy olyan PROM, amely az utasításregiszterből származó 8080 műveleti kódból előállítja az adott utasítást megvalósító mikroutasítás-sorozat első utasításának címét. Lehetőség van az utasításregiszterben megőrzött műveleti kód második dekódolására is, előállítva egy újabb címet a vezérlőtár-



11.13. ÁBRA. Emulátor egyszerűsített blokkvázlata

ban. Ezzel az utasításcsoportok közös részei (pl. regiszterkijelölés) közös mikroutasítás-sorozattal valósíthatók meg.

Az ugrásvezérlő áramkör segítségével lehet feltételes ugrásokat megvalósítani a következő mikroutasítás-cím meghatározásához. Ez valósítja meg a 8080 vezérlősín-bemenetek közül a RDY (Ready: kész) és HOLD (felfüggesztés) vezérlőjeleket a vezérlőtár legkisebb helyiértékbitjének felhasználásával dinamikus hurkot képezve. (Az egyszer végrehajtott mikroutasítás egészen addig saját magát jelöli ki következő végrehajtandó mikroutasításként, amíg a bemenő feltételek – legkisebb helyiértékű bit megváltoztatásával – tovább nem viszik egy következő végrehajtandó mikroutasításra. Hasonló módon történik a megszakításkezelés is).

A 8080 programállapotszó (PSW) és az utasításregiszter egyes bitjeivel megcímzett PROM segítségével is megvalósítanak feltételes elágaztatást.

A mikroutasítás-regiszterben található az éppen végrehajtás alatt álló mikroutasításnak a következő mikroutasítás címképzésében közvetlenül részt nem vevő része. Ez lehetővé teszi átlapolt mikroutasítás-végrehajtás és következő mikroutasítás-elővétel megvalósítását, ezzel meggyorsítja az emulátor működését.

A regiszterek kijelölése az aritmetikai–logikai egység regiszterelemeiben két PROM felhasználásával valósul meg. Így egy további, a mikroutasításból származó jel függvényében a regiszterkijelölést vagy egy mikroutasítás-mezőből megcímzett vagy pedig a makroutasításregiszter megfelelő bitjeivel címzett PROM elem tartalma végzi. Ez a megoldás a közvetlen – mikroutasítás-mezővel való – regiszterkijelölésnél gazdaságosabb, mert minden egyes mikroutasításban kevesebb bitet igényel.

Aritmetikai–logikai egység és a hozzá tartozó regiszterek

Ez a rész valósítja meg az összes számítási feladatot, tehát az adott 8080 aritmetikai és logikai műveleteken túlmenően a szükséges címszámításokat (pl. feltételes és feltétel nélküli ugrási és szubrutinhívó utasítások esetén, vagy pl. megszakítás- és szubrutinkezelésnél a külső veremtár címzését). Ezért a 3002 típusú CPE (központi feldolgozóelem) szeletekből kiképzett ALU két részből áll, és képes két párhuzamos 8-bites műveletet megvalósítani, vagy – a két részt mikrooperáció vezérlése alatt összekapcsolva – egyetlen 16 bites aritmetikai–logikai egységet megvalósítani. Ugyanezen elemekben implementálták a 8080 összes regisztereit (8-bites regiszterek, amelyek 16 bites regiszterpárokká is összekapcsolhatók, valamint a címkezelés 16 bites regiszterei) valamint a mikroprogram által igényelt átmeneti tárolóregisztereket. A két párhuzamosan működő 8-bites részben a regiszterkijelölő PROM különböző regisztereket jelölhet ki, és a végrehajtott mikroművelet is, amelyet a mikroutasítás egy erre a célra szolgáló mezője szab meg, eltérő lehet.

A bemeneti multiplexer vagy a külső (makró) tárból származó, vagy az ALU kimenetén az előző mikroművelet eredményeként megjelenő adatokat, vagy az utasításregiszter tartalmát kapcsolja az ALU-ra. (Képes ezen kívül – ismét természetesen mikroutasítás-mező vezérlése alatt – invertálni a bemenő adatot, vagy pedig csupa zérus vagy csupa egyes értéket adni a bemenetre.)

A programállapot-regiszter tárolja a 8080 PSW értékét, amit a CPE elemeken keresztül tud – szükség esetén – a külső veremtárba kiküldeni (ugyanezen az úton érkezik vissza a PSW érték a veremtárból). Az aktuális vagy visszaírt állapotszó közül az állapotmultiplexer választ.

A maszkolást – a CPE elemekkel – a maszk PROM tartalma végzi, amelyet a megfelelő mikroutasítás-mező címez meg. (A maximálisan szükséges 32 féle 8-bites maszk 5-bites mező segítségével kiválasztható, amely a mező segítségével való maszkoláshoz képest mikroutasításonként 3 bitet takarít meg.)

Be-, ill. kimeneti egységhez és a tárhoz tartozó interfész

A (külső) tárcímregiszter a tárcímsín meghajtásán túlmenően meg is őrzi az éppen megcímzett mező címét. Ez ismét átlapolt működést biztosít, de most a makroutasításokra vonatkozóan. A következő végrehajtandó makroutasítás címét a CPE elemekben implementált programszámláló regiszter tartalmazza. A mikroprogrammal kidolgozott cím egy – ugyancsak az CPE elemekben levő – belső címregiszterbe kerül, és innen „készen" kapuzható át a külső címregiszterbe.

Az adatsín adó-vevő valósítja meg a kétirányú adatsínt, amely háromállapotú, tehát képes nagy impedanciájú állapotba is kerülni (pl. amikor a 8080 processzor felfüggesztett állapotban van, és ezért a sínt fel kell szabadítani egy másik processzor vagy közvetlen tárhozzáférés számára).

A külső adatsínről a 8080 műveleti kód a makroutasítás-regiszterbe íródik. Ennek kimenete – mikroprogram-vezérléssel választható módon – a makroutasítás-dekódoló, az ugrásvezérlő vagy a regiszterkijelölő PROM, ill. az ALU bemenetére csatlakozhat.

A 8080 vezérlősín számára a kimenő jeleket egy vezérlősín-PROM állítja elő. Ennek a hét jelét – ismét jelentős vezérlőtár-megtakarítást elérve – 3-bites mezővel választjuk ki, amelybe egy további bemenet csatlakozik az ugrás vezérlőből.

A 8080 vezérlősín bemenő vezérlő jeleit a vezérlő sínpufferregiszter szinkronizálja össze a mikroprogrammal.

A 11.1. táblázatban foglaltuk össze az emulátor egyes mezőit, azok rövidítéseit (amelyek mezőnévként, ill. mikrooperációként a mikroprogram assembler segítségével emlékeztető szimbólumként is használhatunk), valamint a mező által végzett funkciókat. Az egyes mikroutasítás-mezők „kapcsolatát" a végrehajtást végző modulokkal az egyszerűsített blokkvázlaton (11.13. ábra) mezőszámok segítségével is telöltük.

A 8080 rendszert megvalósító mikroprogramot a [24] irodalom részletesen jartalmazza.

11.1. TÁBLÁZAT. 8080 emulátor mikroutasítás-formátuma

A mező száma	A mező neve	A mező bitszáma	Mikrooperációk	A funkció leírása
1.	AC Address Control: cím vezérlés	7	MCU mikroutasítás cím parancsok	A következő mikroutasítás-cím kidolgozás módját adja meg
2.	LD Load: betöltés	1	LD	A makroutasítás dekódolóból írja be a mikroutasítás címet
3.	SJM Secondary Jump: másodlagos ugrás	1	SJM	A makroutasítás dekódoló címének kiegészítésével azonos makrooperációs kódból egy második mikroutasítás címet képez

11.1. TÁBLÁZAT (1. folytatás)

A mező száma	A mező neve	A mező bitszáma	Mikrooperációk	A funkció leírása
4.	RRE Register ROM Enable: regiszter ROM engedélyezés	1	RRE	Regisz erkijelölést végző PROM-ok közül vagy a mikroutasítás vagy a makroutasítás-regiszter által megcímzettet jelöli ki
5.	RGP Register Group: regisztercsoport	5	R00, R11, R22 R33, R44, R55 R66, R77, R88 R99, RAA, RBB RCC, RDD, REE RFF, R3C, RC3 RCB, RCF, RAC R3D, RDC, RCD RFE, RFF, RFF	A két 8-bites ALU regiszter kijelölését 0...F hexadecimális kódok segítségével a makrooperációk szerinti kombinációknak megfelelően végzi
6.	IST Interrupt Strobe: megszakítás engedélyezés	1	IST	Az INT és HOLD 8080 jelek számára engedélyezi a megszakítást
7.	LD2 Load2: második beírás	1	LD2	A beírás (LD) módosításával másik mikroutasítás-címet jelöl ki
8.	EXT External Test: külső ellenőrzés	1	EXT	Az MA0 mikroutasítás címbit segítségével dinamikus váróhurok megvalósítását engedélyezi
9.	IER Interrupt Enable Register: megszakítás engedélyező regiszter	1	IER	A 8080 megszakításengedélyező jel (INTE) képzését engedélyezi
10.	FI Flag Input: jelzőbit-menet	2	SCZ STZ STC HCZ	Az MCU-ban lévő jelzőbitek (C és Z) beírását vezérli (vagy mindkettő vagy egyik, vagy egyik sem veszi fel a jelzőbit bemenetlogikai értékét)

11.1. TÁBLÁZAT (2. folytatás)

A mező száma	A mező neve	A mező bitszáma	Mikrooperációk	A funkció leírása
11.	FO Flag Output: jelzőbitkimenet	2	FF0, FFC FFZ, FF1	Az MCU-ban lévő jelzőbitek (Z és C) egyikét, logikai nullát vagy egyet ad a jelző bitkimenetre
12.	FGP Function Group: funkciócsoport	4	A CPE mikrooperációi	A 3-bites CPE mikrooperáció kijelölés mellett a negyedik bit teszi lehetővé, hogy a két 8-bites ALU különböző mikrooperációt hajtson végre
13.	CS Carry Select: átvitelkiválasztás	3	CSO, CN, BN AN, CS1, NCN NBN, NAN	8 vagy 16 bites műveletet határoz meg, a bevitt áthozat komplementált vagy nem, léptetéshez multiplexeli az átvitelt, az átvitelt és a félátvitelt beírja a PSW-be
14.	DBY Double Byte: kettős byte	1	DBY	Gyors átvitelképzés 8-bitenként vagy 16 bites ALU-ra
15.	CCR Conditional Clock Ready: feltételes órajelképzés kész	1	CCR	Órajel feltételes kitiltásával a CPE-k regisztereinek tartalma nem változik művelet esetén (csak az állapot jelzés jelenik meg)
16.	NC1 No Clock Array 1: órajeltiltás az 1. ALU-ban	1	NC1	Az 1. ALU szelektíven kikapcsolható
17.	NC2 No Clock array 2: órajeltiltás a 2. ALU-ban	1	NC2	A 2. ALU szelektíven kikapcsolható

11.1. TÁBLÁZAT (3. folytatás)

A mező száma	A mező neve	A mező bitszáma	Mikro-operációk	A funkció leírása
18.	KS K-bus Select: K-sínkiválasztás	3	K0, KD, KM, KIR, K1, KND KNM, KNIR	Az ALU D kimenetet, tárból származó adatot, az utasításregiszter tartalmát, ezek komplemensét csupa zérus vagy csupa egy értéket ad az ALU bemenetére
19.	ED1 Enable Data 1: első adatengedélyezés	1	ED1	A két ALU D kimenet közül az egyiket engedélyezi a belső sín meghajtójaként
20.	FRW Flag Register Write: jelzőbit-regiszter írás	1	FRW	Az újonnan kiszámított jelzőbiteket beírja a PSW regiszterbe
21.	IMB I Mask Bus: I maszk sín	3	I00, IBF, I99, IC7, IFF	Az emlékeztető szimbólumokban a második két hexadecimális számjegy szerinti maszkot generálja a regiszterkijelölő PROM segítségével
22.	ADL Address Load: címbeírás	1	ADL	Beírja a (külső) tárcímregiszterbe a CPE-k (belső) címregiszterének tartalmát
23.	DOE Data Out Enable: adat ki engedélyezés	1	DOE	Az ALU kimenetét a kétirányú adatsínre engedi tárírás és be-, ill. kimeneti írás műveletek esetén
24.	IRW Instruction Register Write: utasítás-regiszter-írás	1	IRW	A kétirányú adatsínről érkező bemenő adatot beírja az utasításregiszterbe
25.	SWPA Status Word PROM Address: státuszszó PROM cím	3	HLTA, INTA, IOW, IOR, MEMW, MEMR, RTRAP, NSTAT	Az egyes 8080 műveleti módoknak megfelelő vezérlősín-állapotokat generálja a vezérlősín PROM segítségével. (RTRAP speciális állapot jelzi, hogy a műveleti kód nincs a 8080 utasításrendszerében)

IRODALOM

[1] *Alexandridis, N. A.:* Bit-sliced mikcroprocessor architekture. Computer, June 1978. 56–80.

[2] *Adams, P. M.:* Microprogrammable mikcroprocessor survey. SIGmicro Newsletter, 1978. Part 1. No. 1.: 23–49. Part 2. No. 2.: 7–38.

[3] *Adams, W. T.–Smith, S. M.:* How bit-slice families compare. Electronics, 1978. Part 1. August 3, 91–98. Part 2. August 17, 96–102. Part 3. August 31, 138–139.

[4] *Chu, P.:* ECL accelerates to new system speeds with high-density byte-slice parts. Electronics, August 2, 1979. 120–125.

[5] *Agrawala, A. K.–Rauscher, T. G.:* Foundations of Microprogramming. Architecture, Software and Applications. Academic Press, New York, 1976.

[6] *Sawin, D. H.:* Basic principles of microcontroller design. MicroScope, 1977 (10) 1–7.

[7] *Kong, J.–Proske, T.:* Microcoprogramming boosts computer power – especially if you can do it yourself. Electronic Design, March 15, 1979. 84–88.

[8] *Mrazek, D.–Boron, M.:* Boost bit-slice microsequencer speed: choose from eight different circuits. Electronic Design, May 24, 1979. 118–123.

[9] *Ramamoorthy, C. V.–Chang, L. C.:* System modeling and testing procedures for microdiagnostics. IEEE Trans. on Computers, November 1972. 1169—1183.

[10] *Stockenberg, J.–van Dam, A.:* Vertical migration for performance enhancement in layered hardware/firmware/software systems. Computer, May 1978. 35–50.

[11] *Boehm, B. W.:* Software Engeneering, IEEE Trans. on Computers, December 1976. 1226–1242.

[12] *Davidson, S.–Shriver, B. D.:* An overview of firmware engineering. Computer, May 1978. 21–23.

[13] *Conway, J. C.:* Hardware approaches to microprogramming with bipolar microprocessors. Computer Design, August 1978. 83–91.

[14] *Blood, B.:* Microprogram for fast processing using bipolar LSI bit-slice circuits. Electronic Design, September 13, 1978. 114–115.

[15] *Csákány A.–Vajda F.:* Mikroszámítógépek. Műszaki Könyvkiadó, Budapest, 1977.

[16] *Gilb, T.:* Software Metrics. Winthrop Publishers Inc. Cambridge, Massachusetts, 1977.

[17] *Powers, V. M.–Hernandez, J. H.:* Microprogram assemblers for bit-slice microprocessors. Computer, July 1978. 108—120.

[18] CROMIS (Serics 3000 Cross Microprogramming System) reference manual. Intel Corp. Santa Clara, California, 1975.

[19] Micro assembler reference manual. Signetics Corp., Sunnyvale, California, June 1977.

[20] The PDP–11/60 microprogramming tools reference manual. Digital Equipment Corporation, Maynard, Massachusetts, 1977. October.

[21] *Mick, J. R.–Schopmeyer, R.:* MOS support microprocessor teams with bit-slice prototypes for easier microprogram debugging. Electronics, September 15, 1977. 127–130.

[22] *Oberzeir, J. A.:* Writable control store saves microprogramming time and expense. Electronics, June 21, 1979. 121–125.

[23] *Stetson, R. J.:* Build a flexible firmware generator to control system-timing sequences. Electronic Design, September 13, 1978. 130–132.

[24] 8080 emulator manual. Signetics Corp. Sunnyvale, California, March 1977.

[25] *Balph T.–Blood, W.:* Assembler streamlines microprogramming. Computer Design, December 1979. 79–84.

[26] *Ambrózy G.–Miskolczi J.–Vajda, F.:* Small computer built with Intel 3000 bit-slice microprocessor system. Proc. MIMI'78 (Zürich), Acta Press, Anaheim, Calgary, Zürich 1978. 3–8.

[27] *Ambrózy G.–Miskolczi J.:* Development of a microprogrammable microprocessor system by means of a PDP—8E minicomputer. Euromicro Journal, 1978. 92–104.

[28] *Кирпичников, В. М.—Скляров, В. А.:* К методам описания и синтеза одного класса микропрограммных автоматов. Известия АН СССР, Техническая кибернетика, январь—февраль 1979. 127—137.

12. MULTI-MIKROPROCESSZOROS RENDSZEREK

A „hagyományos" számítástechnika és az LSI mikroprocesszorok alkalmazástechnikájának kölcsönhatása fontos szerepet játszik a multi-mikroprocesszoros rendszerek fejlődésében. Egyrészt a mikroprocesszorok alkalmazásával újra „feltaláljuk a kereket", vagyis a nagyobb számítógépek multiprocesszoros rendszereinek tapasztalatait vesszük át, másrészt azonban a mikroprocesszor segítségével elvileg új számítástechnikai elrendezések is kialakíthatók. Ugyanakkor a nagyszámú processzor alkalmazása nagyteljesítményű párhuzamos architektúrák megvalósítását is lehetővé teszi [1], [2], [3].

Több mikroprocesszor alkalmazásával megjavíthatjuk rendszereink jellemzőit (elsősorban működési sebességét), növelhetjük azok megbízhatóságát, „hibatűrőkké" tehetjük rendszereinket, ezzel nagymértékben fokozva azok rendelkezésre állásának (availability) valószínűségét. Erre a két legfontosabb módszer az újraelrendezés és többszörözés. Az első esetben meghibásodás esetén a rendszer a működő modulok között újra elosztja a részfeladatokat, ezzel csökkentett jellemzőkkel való továbbműködést biztosítva. A második esetben két vagy több (általában három) funkcionális egység vagy egész rendszer működik párhuzamosan. Természetesen mindkét módszer alkalmazása esetén szükség van megfelelő hibajelzésre és a meghibásodott elem azonosítására is.

Maga a multi gyűjtőnév számos – alapvetően eltérő – elrendezést és így alkalmazást is magába foglal. Ezért, mielőtt a multimikroprocesszoros rendszerek tervezésének problémáival foglalkoznánk, röviden áttekintjük ezek felépítését, megadva szervezésük alapvető definícióit.

12.1. MULTIPROCESSZOROK ARCHITEKTÚRÁJA

A multiprocesszorokat (párhuzamos processzorok) többféleképpen is csoportosítják [1], [4], [5], [6]. Ezek közül az utasítás, ill. adatfolyam szerinti felosztást mutatjuk be. Ennek alapja, hogy egy adott időintervallumban a párhuzamos működésben azonos utasítások különböző adatokkal, azonos adatok különböző utasításokkal vagy különböző utasítások különböző adatokkal vesznek-e részt. Ezek alapján a következő csoportok lehetségesek:

- SISD (Single Instruction, Single Data: egy utasítás, egy adat) egyetlen szokásos processzor.
- MISD (Multiple Instruction, Single Data: több utasítás, egy adat) csővonal (pipeline) processzor.
- SIMD (Single Instruction, Multiple Data: egy utasítás, több adat) párhuzamos processzorrendszer.
- MIMD (Multiple Instruction, Multiple Data: több utasítás, több adat) elosztott vagy több processzoros rendszer.

A továbbiakban a SIMD és MIMD szervezését nézzük meg egy kicsit közelebbről.

SIMD architektúra

A 12.1. ábra szerinti egyetlen vezérlőegység (processzor) veszi elő a tárból az utasítást és értelmezi, ill. bizonyos esetekben – pl. ugrás vagy feltételes elágazás esetén – végre is hajtja azt. Aritmetikai utasítások esetén a vezérlő az utasítást átadja végrehajtásra a feldolgozóelemeknek (processzoroknak). Ezeknek közös táruk (12.1. ábra) vagy pedig külön táruk (12.2. ábra) van, amelyek egy nagy adatátviteli sebességet biztosító sínhez csatlakoznak.

A SIMD elrendezések a következő alcsoportokra oszthatók:

- Mátrix- vagy vektorprocesszorok, ahol az adatok cím szerint – a mátrix- vagy vektorelrendezésnek megfelelően – párhuzamosan kerülnek feldolgozásra.
- Asszociatív processzorok, ahol a párhuzamosan feldolgozott adatokat tartalom szerint választják ki.
- Feldolgozó együttesek, ahol egy teljes számítógép a vezérlőegység és a processzorok ezen keresztül küldött üzenetekkel kommunikálnak egymással.

Multimikroprocesszoros rendszerekkel való megvalósítás esetén legnagyobb jelentősége a mátrix- (vektor-) processzoroknak van. Ezek elsősorban olyan speciális alkalmazások esetén gazdaságosak, ahol

- az elvégzendő műveletek különböző adatokkal egyidejűleg mátrix- (vektor-) műveletek formájában írhatók le,
- a processzorok között nagy adatátviteli sebességű adatutak állnak rendelkezésre,
- az egyidejűleg kezelt operandusokat egyidejűleg lehet elővenni.

Ilyen művelettípusok jelentkeznek parciális differenciálegyenletek, lineáris egyenletek megoldásánál, mátrixok szorzásánál stb. Az alkalmazások specifikus területei pl. a jelfeldolgozás vagy képfeldolgozás. Bitszeletelt mikroprocesszor-elemek alkalmazásával mikrovezérléssel is megvalósíthatunk LSI mátrix- (vektor-) processzorokat (l. a 11. fejezetet).

MIMD architektúra

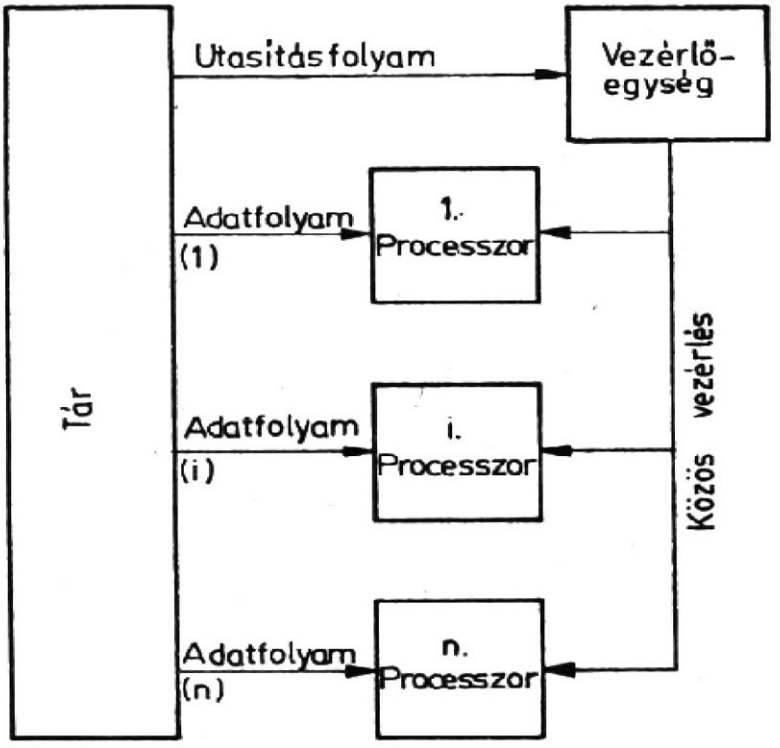
A MIMD rendszer általános felépítési elvét mutatja a 12.3. ábra. A 12.4. ábrán levő blokkvázlaton már egy szokásos megoldás részleteit is feltüntettük [7]. Az egyes processzorok független részfeladatokat oldanak meg különböző adatokkal, majd ezek eredményeit együttesen használják fel. A processzorok az erőforrásokat általában együttesen használják, azonban lehet saját külön erőforrásuk (tár, be- ill. kimeneti egység) is. A feladatok ilyen típusú architektúrával való jó hatásfokú megoldásának a feltétele egyrészt a feladat megfelelő felosztása és processzorhoz való rendelése (dedication), másrészt pedig a processzorok vagy részfeladatok szinkronizálása. Az utóbbi feladatot a processzorok megszakításrendszere segítségével valósítottuk meg a 12.4. ábrán bemutatott megoldásában.

Az MIMD rendszerek két alcsoportja: az elosztott és multiprocesszor-rendszerek.

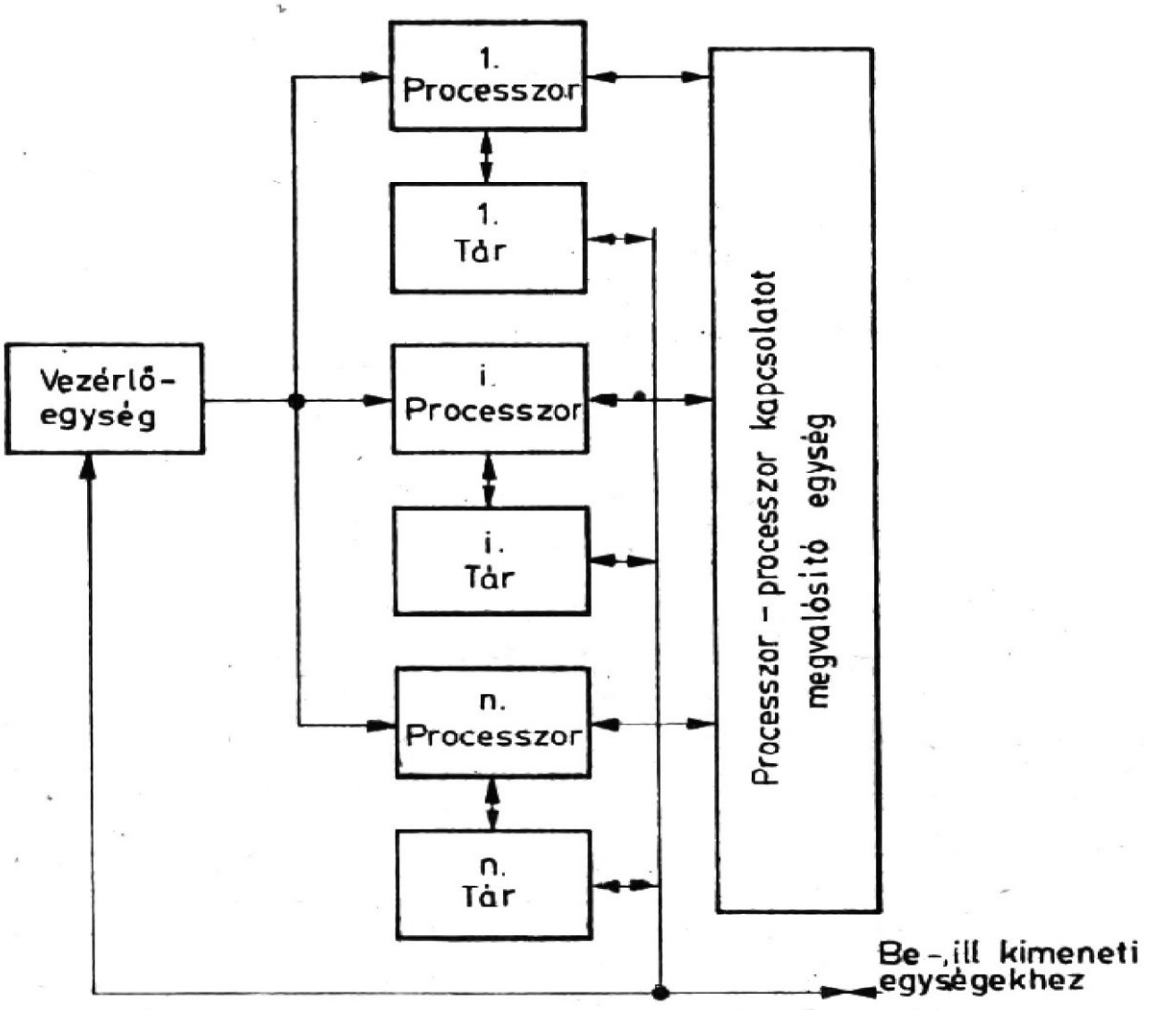
Elosztott rendszerek

A processzorok ebben az esetben is hozzájuk rendelt funkciókat hajtanak végre, és ezen túlmenően földrajzilag különböző helyen lehetnek (kommunikációs hálózat).

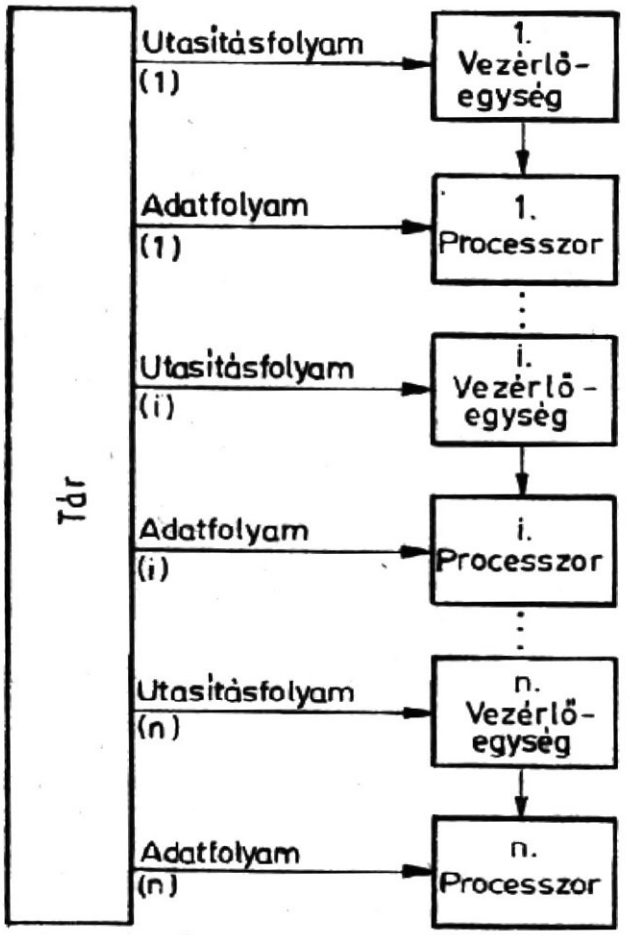
Az egyes részfeladatokat előre jól kell ismerni, és hozzájuk tartozó programokat kell az egyes processzorokhoz rendelni. A hozzárendelés ebben az esetben statikus (nem változik) és oly módon kell elvégezni, hogy az egyes részprogramok lehetőleg külön-külön elkészíthetők és kipróbálhatók (kijavíthatók) legyenek. A processzorok közötti kommunikáció megosztva használt perifériák, vagy soros adatcsatorna segít-



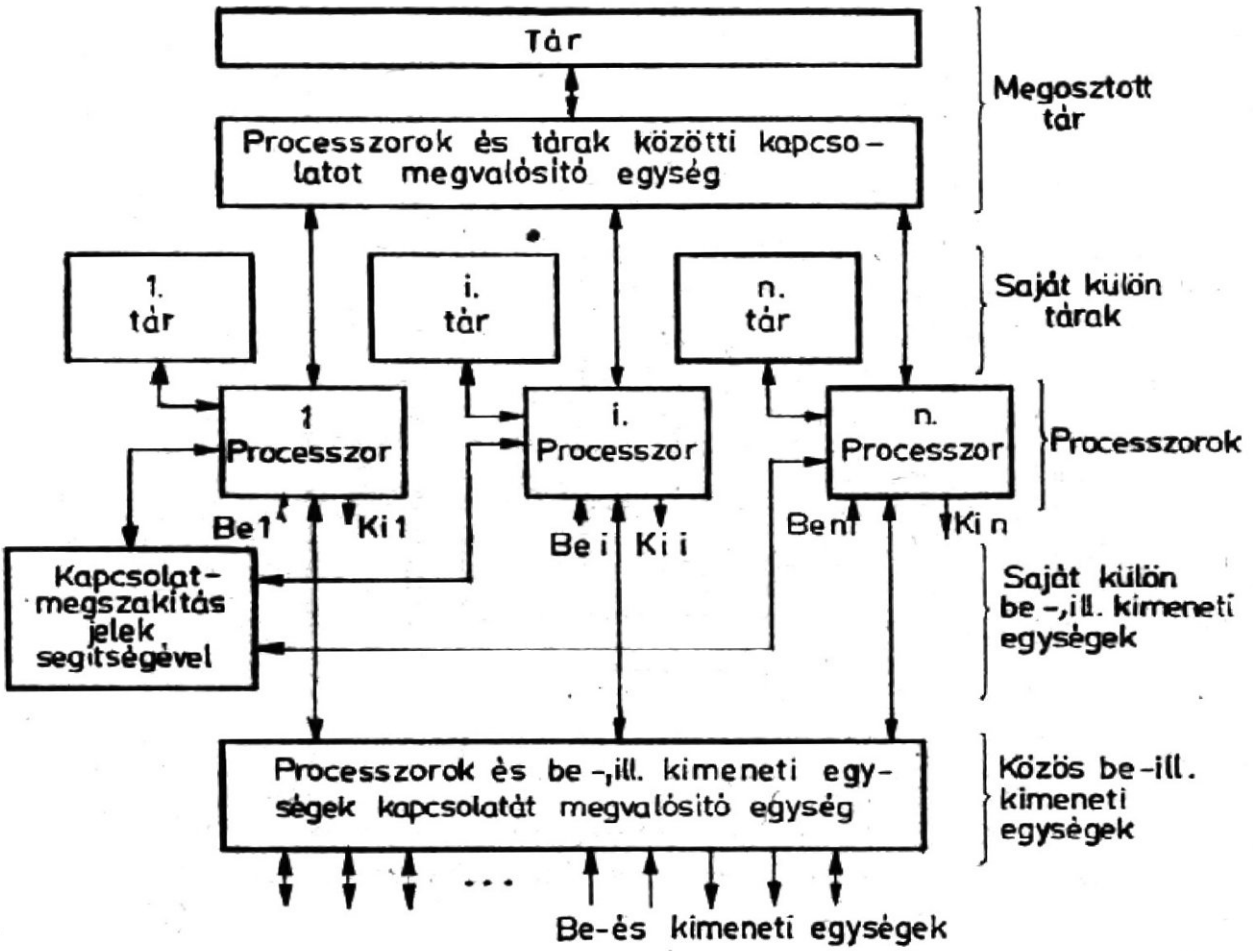
12.1. ÁBRA. SIMD multiprocesszor blokkvázlata (közös tár)



12.2. ÁBRA. SIMD multiprocesszor blokkvázlata (külön tárak)



12.3. ÁBRA. MIMD multiprocesszor



12.4. ÁBRA. MIMD multiprocesszor funkcionális blokkvázlata

ségével átadott üzenetek, vagy adatblokkok formájában valósul meg. Ha az üzeneteket háttértár (pl. mágneslemezes tár) közvetíti, a rendszert multi-mikroszámítógépnek szokták nevezni.

Elosztott rendszerek legfontosabb hátrányai a következők:

- A processzorok terhelése általában nem egyforma, és így a rendszer nem jó hatásfokú.
- Nehéz a programot megfelelő részekre bontani.
- Bővítés nagymértékű átrendezés nélkül nem biztosítható.
- Bármely processzor meghibásodása általában üzemképtelenséghez vezet, mivel a rendszer nem képes dinamikusan feladatokat átadni a meghibásodott processzortól egy másik processzornak.

Multiprocesszoros rendszer

A multiprocesszoros rendszereknek (MPS: MultiProcesszor System) olyan operációs rendszerük van, amely a feladatokat érkezésükkor a processzorokhoz hozzárendeli. Az operációs rendszer, vagy minden processzor modulon futhat (lebegő executiv), vagy egy erőforrás-elosztó (master) processzor rendelheti a feladatokat az egyes (slave) processzorokhoz. A processzormodulok lehetnek azonos felépítésűek, azaz szimmetrikus felépítésűek, vagyis bármely feladat elvégzésére képesek, vagy adott funkcióra orientáltak, azaz aszimmetrikus felépítésűek. Ha valamennyi processzor ilyen felépítésű, ismét egy elosztott rendszert kapunk.

A multiprocesszoros rendszerek tovább csoportosíthatók a processzormodulok közötti kölcsönhatás alapján is, amit gyakran csatolásnak is neveznek. Ennek alapján beszélhetünk szorosan és lazán csatolt processzorokról.

Szorosan csatolt processzorok a vezérlő által szigorúan meghatározott módon működnek. Általában csak az egészen nagy számítógépek kategóriájába eső rendszereket valósítják meg ily módon.

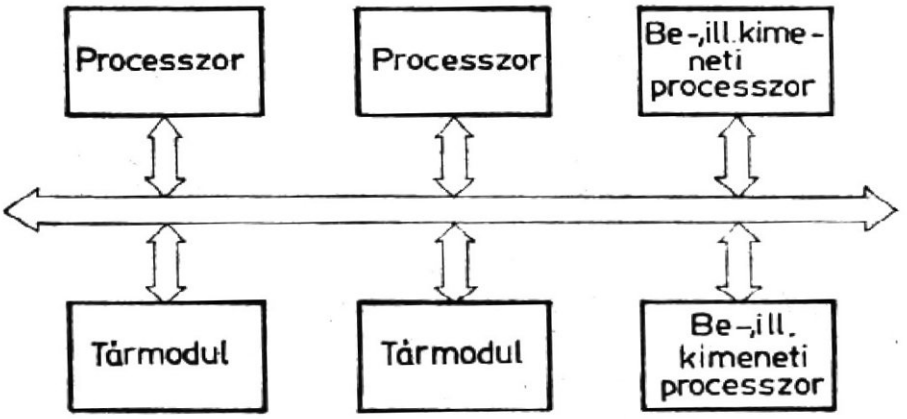
Lazán csatolt processzorok között nincs programkölcsönhatás, csupán tárat (vagy más erőforrást, például be-, ill. kimeneti egységet) használnak közösen, hogy információt adjanak át egymásnak. Minden olyan rendszer, ahol a feladat ily módon felbontható (pl. adatbázisrendszerek, ipari vezérlőrendszerek stb.), megvalósítható mikroprocesszorokból felépített multiprocesszoros formában.

12.2. FUNKCIONÁLIS MODULOK CSATLAKOZTATÁSA

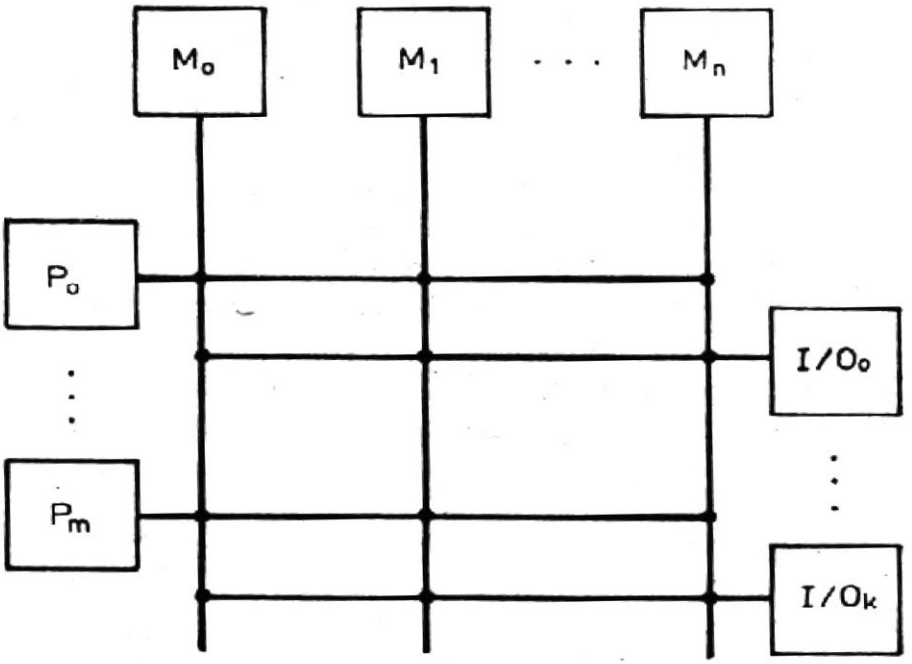
A funkcionális modulok (processzorok, megosztott tár és be-, ill. kimeneti egységek) szervezésének, a csatlakoztatás topológiájának nagy jelentősége van a multiprocesszoros rendszer által biztosítható párhuzamosság, ill. a megvalósítható egyidejű működés (concurrency) szempontjából. Vizsgálatainkat először a legfontosabbra, az adatcserére koncentráljuk. Ennek figyelembevételével a legnagyobb gyakorlati jelentőségű csatlakoztatási módok a következők:

Osztott idejű közös sínrendszer

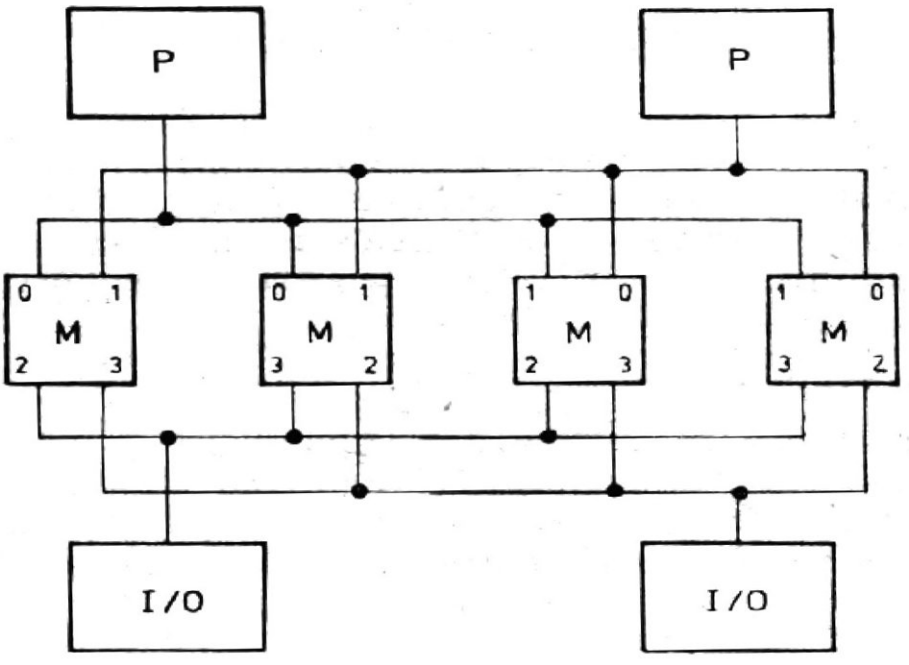
Ez a legegyszerűbb és így megvalósítás szempontjából is a legolcsóbb megoldás (12.5. ábra). A megvalósított sínrendszer jellemzői (l. a 7. fejezetet) fogják a rendszer legfontosabb jellemzőit is megszabni. Mivel a sín átviteli sebessége a rendszer működési sebességét korlátozza, a gyakorlatban az egyes processzoroknak sokszor saját táruk és be-, ill. kimeneti csatlakozásuk is van.



12.5. ÁBRA. Osztott idejű közös sínrendszer



12.6. ÁBRA. Keresztrudas (crossbar) kapcsolórendszer



12.7. ÁBRA. Modulok csatlakoztatása többszörös hozzáférésű tárak segítségével

Crossbar (keresztrudas) kapcsolórendszer

Ha az osztott idejű működést megvalósító sínek számát szaporítjuk, elérhetünk egy olyan állapotot, hogy – a 12.6. ábra szerint – minden egyes táregységnek egy különkülön adatútja legyen. Így az egyidejű átvitelek számát ez szabja meg. A keresztpontoknál levő „kapcsolóknak" kell megoldaniuk a több egyidejű kérésből származó konfliktushelyzeteket is, valamilyen előre eldöntött prioritási sorrend alapján (l. a 7. fejezetet). Így e kapcsolók elég összetett felépítésűek lesznek. Figyelembe véve, hogy a kapcsolók által kezelt vonalak (cím-, adat- és vezérlőjelek) száma is nagy (32...64 vezeték a gyakorlatban megvalósított rendszerekhez), az egész rendszer nagyságát a kapcsolók fogják meghatározni.

Többszörös hozzáférésű tárakat alkalmazó rendszer

Ez egy olyan elrendezés, amely több egyidejű átvitelt valósít meg. A tármodulok vezérlésének természetesen olyannak kell lennie, hogy több egyidejű hozzáférést tegyen lehetővé. A prioritást nagyon gyakran a 12.7. ábra számozása szerint az egyes fizikai csatlakozópontokhoz rendelik oly módon, hogy a processzoroknak és a be-, ill. kimeneti egységeknek magasabb prioritásuk legyen egy adott (primer) tármodulhoz való hozzáférés esetén. Nem szükséges azonban, hogy minden tármodul minden processzorhoz és be-, ill. kimeneti modulhoz csatlakozzon.

Hasonlítsuk össze az előbbi három rendszer tulajdonságait az ár, megbízhatóság, flexibilitás, és a rendszer „átbocsátóképessége" szempontjait figyelembe véve [7], [12], [13], [14]:

Osztott idejű közös sínrendszer

- Legolcsóbb rendszerhardver.
- Egyszerű hardver konfiguráció, módosíthatóság.
- A rendszer nagyságát korlátozó sín átviteli sebesség.
- Sínmeghibásodás esetén katasztrofális rendszerhiba.

Keresztrudas kapcsolórendszer

- A legkomplexebb rendszer.
- A funkcionális egységek a legegyszerűbbek.
- A legnagyobb potenciális átviteli sebesség és rendszerhatásfok.
- A rendszer bővítése a jellemzőket mindig javítja.
- A rendszer, ill. a kapcsolók felépítésével a megbízhatóság növelhető.

Többszörös hozzáférésű tárokat alkalmazó rendszer

- A legbonyolultabb tárfelépítés és a legdrágább.
- Viszonylag nagy átviteli sebesség.
- A rendszer méretét a tárhozzáférések száma határozza meg.
- Nagyszámú csatlakozópont és vezeték.

A processzorok közötti kommunikáció alapvető eszköze az ún. postaláda tár (mailbox). Ebben az értelemben processzornak tekintjük a központi feldolgozóegységeket (CPU) és a be-, ill. kimeneti egységek processzorait is (I/O processzor pl. a közvetlen tárhozzáférést vezérlő DMA csatorna). A postaláda tár egy olyan – a processzorok által – megosztva használt erőforrás, amely üzeneteket, adatokat, kéréseket stb. tartalmaz. A küldő processzor elrendezi az információt, és elhelyezi azt a postaládában. A címzett fogadóprocesszor vagy időnként „belenéz" a postaládába, vagy pedig a küldő „megnyomja a csengőt", jelezve, hogy valami van a postaládában a fogadó számára. Így a gyakorlatban végigkérdezés (polling), jelzőbitek, ugrásfelté-

telek, megszakítás vagy speciális utasítások használhatók a postaláda segítségével kommunikáció esetén.

Különösen alkalmas a postaláda állapotának jelzésére egy állapotbit. A jelzőbit állapota – amelyet a küldő beállít és az információra váró ellenőrizhet (TAS: Test-And-Set, azaz ellenőrzés és beállítás eljárás) – mutatja, hogy a postaládában van-e hasznos információ.

12.3. OPERÁCIOS RENDSZEREK

Ahhoz, hogy általános célú multiprocesszoros rendszerről beszélhessünk, nem elegendő, hogy a rendszerben több processzor legyen, és hogy ezek a processzorok a rendszer erőforrásait közösen és megosztva használják, hanem a rendszer működtetéshez megfelelő operációs rendszerre is szükség van. Egy ilyen operációs rendszer fő feladatai a következők [1], [15]:

1. Beviszi és tárolja a rendszer által megoldandó feladatokat (task).
2. Irányítja a feladatok végrehajtását.
3. Különleges körülmények (pl. aritmetikai vagy géphiba, megszakítások, illegális vagy fenntartott utasítások végrehajtásának megkísérlése, védett mezők címzésének megkísérlése, nemlétező címekből származó fennakadás, a rendszer befagyása stb.) esetén végrehajtja a szükséges intézkedéseket.
4. Elosztja a hardvererőforrásokat.
5. Hozzáférést biztosít a szoftvererőforrásokhoz (pl. szerkesztő-, fordítóprogramok, szubrutinkönyvtárak, segédprogramok stb.).
6. Védelmet (esetleg titkosítást) és hozzáférés-vezérlést biztosít az információ számára.
7. Üzeneteket közvetít az egyes részek (feladatok) között.

Multiprocesszorok operációs rendszerei számára az általános operációs rendszer feladatain túlmenően meg kell oldania a processzorok és a be-, ill. kimeneti egységek terhelésének megfelelő szinten tartását, és szükség esetén a rendszer újjárendezését (Reconfiguration).

A párhuzamos feldolgozás, a több folyamat szimultán – egyidejű – végzését biztosító multifeldolgozás (multiprocessing) és a több program (feladat) egy rendszeren való futtatását biztosító multiprogramozás specifikus problémáival kapcsolatosan az irodalom igen gazdag [9], [16], [17], [18].

A multiprocesszorok operációs rendszere a következő három szervezési mód egyikét használja [1]:

1. Mester–szolga (master–slave) operációs rendszer, amely mindig egy adott (master) processzort használ.
2. Minden processzor számára külön operációs rendszer, amely annak igényei szerinti szolgáltatásokat nyújt.
3. Szimmetrikus operációs rendszer, ahol az egyes feladatok (és hozzájuk rendelt erőforrások) el vannak osztva az egyes processzorok között, ill. a „mester" pozíciót hol az egyik, hol a másik processzor játssza.

A multi-mikroprocesszoros rendszerek jövője vitathatatlan akkor is, ha még nem állt elegendő idő rendelkezésre az elvek olyan gyakorlati alkalmazására, amely kereskedelmileg forgalmazott általános célú rendszerek formájában valósult volna meg.

Ilyen rendszerek megvalósításakor ui. számos jelentős probléma merül fel [9]. Ezek pl. a következők:

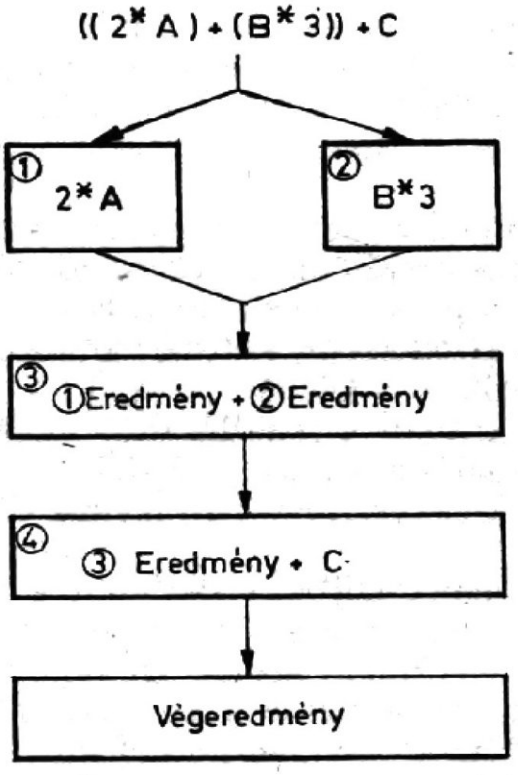
1. A feladat részekre bontása (decomposition).

2. Az összekötés struktúrája (interconnection).
3. A címelhelyezés módszere (mapping).
4. A rendszerprogram (szoftver) felépítése.
5. A processzorok közötti kölcsönhatás (interference).
6. A „befagyás", holtpont elkerülése (deadlock).
7. A hibatűrés (Fault tolerance).
8. Be-, ill. kimeneti berendezésekhez a kapcsolat.

12.4. PÁRHUZAMOS FOLYAMATOK PROGRAMOZÁSA

Egy feladat megoldása kapcsán gyakran felmerül, hogy egy programlépést csak két (vagy több) megelőző lépés eredményének felhasználásával lehet „megtenni". A 12.8. ábrán egy ilyen számítás folyamatábráját láthatjuk, ahol az 1. és 2. részfeladat – a harmadik előtt – logikailag egyidejűleg végrehajtandó. A két részfeladat független is egymástól, mivel nem használ közösen erőforrásokat, és egyik eredménye nincs hatással a másikra. A logikai párhuzamosság (concurrency) egyetlen processzor alkalmazása esetén is egy általánosan megoldott feladat (l. a 8. fejezetet az operációs rendszerekkel kapcsolatosan). Különösen igaz ez az egyetlen (egyprocesszoros) rendszeren több feladat osztott idejű (time-sharing) megoldását biztosító multiprogramozást megvalósító számítógép-rendszerek esetén. E területek számára kidolgozott alapvető módszereket [18] használhatjuk fel abban az esetben, ha a rendszerben több processzorunk van, amelyek nem csupán logikai, hanem fizikai párhuzamos végrehajtást is lehetővé tesznek.

Párhuzamos programozási folyamatok megoldása során a számítás folyamatában egy szekvenciális részt (alapegységet) részfeladatnak (processnek) is szoktak nevezni. Ezt egy adatstruktúra, az ún. PCB (Process-Control Block: részfeladat-vezérlő blokk) definiálja, amely egy tárterületet tartalmaz a központi egység regisztereinek megőrzésére megszakítás esetén, egy prioritásmezőt, valamint a „sorban"



12.8. ÁBRA. Példa a párhuzamos részfeladatokra

következő PCB-re vezető mutatót. A párhuzamos programozás megvalósításához minden ilyen részfeladatnak aszinkronnak és megszakíthatónak kell lennie. A részfeladatok közötti kommunikáció vagy egy rendszerprogramon keresztül, vagy megszakítások segítségével valósítható meg.

A megosztott erőforrások közötti megosztást, ill. hozzárendelést a sorban várakozó részfeladatok (vagy processzorok) számára szokásosan a felügyelőprogram egy külön programmodulja végzi. Ezt szokták „magnak" (kernel) nevezni [18], [19] [20].

Az erőforrás-kezelés és a hozzárendelés bonyolult számítástechnikai feladat, amely különböző algoritmusok szerint valósítható meg [21], [22].

A párhuzamos végrehajtás leírására több programnyelvekben használt eszköz áll rendelkezésre [27]. Az ALGOL–60 nyelv kibővítéseként használt **parbegin** (párhuzamosság kezdete) és **parend** (párhuzamosság vége) utasítás-zárójeleket (amely helyett gyakran a **cobegin** és **coend** formát is találjuk) elterjedten használják a párhuzamosság leírására. A **parbegin** és **parend** közötti programrészletet egy összetartozó egységnek lehet tekinteni. Ez azt jelenti, hogy:

- a „zárójelek" között lévő utasítások egyidejűleg végrehajthatók,
- a **parend** utasítás zárójelet követő utasítás csak akkor hajtható végre, ha már a megelőző összes utasítás végrehajtása befejeződött.

Pl.:

begin S_1; **parbegin** S_2; S_3; S_4 **parend** S_5 **end** azt jelenti, hogy S_1 (utasítás) végrehajtása után egyidőben kezdődhet S_2, S_3 és S_4 végrehajtása, és mikor mindegyik befejeződött, akkor kezdődhet S_5 végrehajtása.

Egy másik elterjedt módszer, amikor programelágazást a FORK művelettel definiálnak [27].

Pl. **FORK** *A* elindít egy részfeladatot az *A* címkénél és folytatja a kiadó részfeladatot a következő utasításnál. A **JOIN** (egyesülés) utasítás a párhuzamos ág befejeződését vezérlő utasítás. A **QUIT** (abbahagyás) utasítás az egyik ág befejezését végzi anélkül, hogy a csomóponthoz jutna.

A párhuzamos működést, az egyes folyamatok együttműködését, szinkronizációját különböző eszközökkel valósíthatják meg. Az egyes közösen használt erőforrások egyidejű használatából származó konfliktushelyzetek megoldásának legegyszerűbb módja a kölcsönös kizárás (interlock). Ez megvalósítható oly módon is, ha csak azt tételezzük fel, hogy a tár írás és olvasás művelete (LOAD és STORE) oszthatatlan. Ezt nevezzük tárkizárásnak. Ez annyit jelent, hogy a tár egy szavához csak egy hozzáférés lehetséges, ha más akarja a szót felülírni vagy vizsgálni, várnia kell.

Ha a végrehajtás alatt a közös adatterület elemeit (vagy elemét) módosítják vagy vizsgálják, kritikus szakaszról beszélünk. A kölcsönös kizárás biztosítja, hogy egyidejűleg csak egy program legyen a kritikus szakaszban. A kritikus szakasz egy bezárás **(LOCK)** és kizárás **(UNLOCK)** utasításpár közé téve nem enged meg újabb belépést, amíg nem következik be a kilépés.

Az előző példákban a kritikus szakaszba való belépésre a részfeladat „hurokban" várakozik. Másik megoldás a várakozó részfeladat „blokkolása" (alvó állapot) és a belépés esetén annak „felébresztése": (wakeup). Ehhez tartozó utasításpár a **BLOCK** és **WAKEUP.**

Ugyanerre a célra definiálta Dijkstra 1965-ben a szemafor műveleteket [27]. A szemafor egy egész típusú változó, amelyek értékét két művelettel, a *p* és *v* operációkkal, az ún. szinkronizációs primitívekkel lehet változtatni.

A párhuzamos program végrehajtása előtt a szemaforokat megfelelő kezdőértékre kell beállítani (inicializálás). Minden szemaforhoz tartozik egy várakozó sor, amelyre a részfeladatok kerülhetnek *p* (*sem*) művelet hatására.

A *p* (*sem*) csökkenti a szemafor értékét eggyel, és a kiadó részfeladat fut tovább. Ha a csökkentés eredményeként a szemafor negatívvá válik, akkor a *p* (*szem*) műveletet kiadó részfeladatnak várakoznia kell, azaz a szemafor várakozólistájára kerül.

A *v* (*sem*) növeli a szemafor értékét eggyel. Ha a várakozólistán van részfeladat, akkor ennek eredményeként azok valamelyike futtathatóvá válik.

A *p* és *v* operációk oszthatatlanok, vagyis egy szemaforra egyidejűleg csak egy *p* vagy *v* művelet kerülhet.

Ha a szemafor értéke nem pozitív, akkor abszolut értéke a várakozólistán levő elemek számával egyenlő. A szemafor műveletekhez hozzátartozik egy ütemezési algoritmus is, ami meghatározza, hogy a várakozólistáról milyen algoritmus szerint választódik ki az elindítandó részfeladat. Egy futó részfeladat csak akkor függesztődik fel, ha *p* operációt hajt végre egy nem pozitív szemaforon [29].

Aszinkron események közötti kapcsolat megoldásához jól illeszkedő megoldás a megszakítás alkalmazása.

Egy multi-mikroprocesszoros rendszerben megszakítást négy szinten használhatunk: [17]

1. Adott processzorokon belüli kommunikációra és vezérléshez (pl. illegális utasítás vagy illegális operandus, vagy nullával való osztás stb.).
2. Rendszeren belüli (intrasystem) kommunikációra és vezérléshez (pl. rendszerperifériák kéréseinek kiszolgálása).
3. Operációs rendszerrel (operating system vagy executive program) való kommunikáció és vezérlés. (Az egyes részfeladatok megoldási sorrendjét, ill. a rendszer erőforrásaiért folyó versenyt a szoftver vezérlése alatt dinamikusan változtatható prioritásszintek szabják meg.)
4. Processzorok közötti (interprocessor) kommunikáció és vezérlés. (Adatok vagy állapotbitek átvitelének megindítása, vagy vezérlésátadás.)

Megszakításvezérelt (interrupt driven) rendszerek esetén az egyes feladatok végrehajtását a hozzájuk rendelt prioritásszint szabja meg. Pl. egy jellemző prioritásstruktúra [23] a következő megszakításszintekből áll.

- NPR (NonProcessor Request: nem processzortól jövő kérés) nagysebességű közvetlen tárhozzáféréssel (DMA) működő perifériák számára.
- Négy sínkérés (BR: Bus Request) szint periféria-megszakításkérések számára.
- Hét szoftvermegszakítási szint program-megszakításkérések számára (PIR: Program Interrupt Request).

A részfeladatok és processzorok közötti koordinációt korszerű szinten CHLL (Concurrent High Level Language: konkurrens magas szintű nyelv) típusú nyelvek (pl. concurrent PASCAL [19]) alkalmazásával oldhatjuk meg. E megoldások esetén ugyanakkor arra is törekszenek, hogy a koordinációs folyamat automatikusan menjen végbe és a felhasználó számára ne jelentsen külön problémát (átlátszóság). Ennek részleteivel kapcsolatosan azonban csak az irodalomra utalunk [15], [19], [20], [24].

IRODALOM

[1] *Enslow, P. M.:* Multiprocessors and parallel processing. John Wiley and Sons, New York, London, Sydney, Toronto, 1974.

[2] *Burton, T.:* Multi-μP systems combine the efficiency of dedicated microcomputers with the throughput of minis. Electronic Design, August 2, 1977. 68—71.

[3] *Thurber, K. J.:* Parallel processor architectures. Computer Design, January 1979. 89–97.

[4] *Enslow, P. H.:* Multiprocessor organization – A survey. Computing Surveys, March 1977. 103–129.

[5] *Smith, A. J.:* Multiprocessor memory organization and memory interference. Commun. of the ACM, October 1977. 754–761.

[6] *Hughes, P.–Doone, T.:* Multi-processor systems. Microelectronics and Reliability, 1977. 281–293.

[7] *Weissberger, A. J.:* Analysis of multiple-microprocessor system architectures. Computer Design, June 1977. 151–163.

[8] *Baker, K. D.:* Functional decomposition on multi-microprocessor systems. The Radio and Electronic Engineer, November 1977, 497–504.

[9] *Fuller, S. M. et al.:* Multi-microprocessors: An overview and working example. Proc. of the IEEE, February 1978. 216–228.

[10] *Adams, G.–Rolander, T.:* Design motivations for multiple processor microcomputer systems. Computer Design, March 1978. 81–89.

[11] *Davis, R. L.–Zucker, S.–Campbell:* A building block approach to multiprocessing. Proc. AFIPS, Spring Joint Computer Conference, 1972. 685—703.

[12] *Gilbert, P.–Chandler, W. J.:* Interference between communicating parallel processes. Commun. of the ACM, June 1972. 427–437.

[13] *Reyling, G.:* Performance and control of multiple microprocessor systems. Computer Design, Marc 1974. 81–87.

[14] *Bowra, J. W.–Torng, H. C.:* The modeling and design of multiple function-unit processors. IEEE Trans, on Computers, March 1976. 210–221.

[15] *Korin, H.:* Parallelism in hardware and software: Real and apperant concurrency.Prentice Hall, Englewood Cliffs, 1972.

[16] *Jones, A. K. et al.:* Programming issue raised by a multiprocessor. Proc. of the IEEE, February 1978. 229–237.

[17] *Jaswa, R.:* Designing interrupt structures for multiprocessor systems. Computer Design, September 1978. 101–110.

[18] *Baer, L.:* A survey of some theoretical aspects of multiprocessing. Computer Surveys, March 1973. 31–80.

[19] *Hemmenway, J.–Teja, E.:* Advanced software systems design course. EDN, October 20, 1979. 293–336.

[20] *Buhr, R. J. A.:* Toward transparent coordination mechanisms for multiple microprocessor systems. Dept of System Eng. and Computing Science, Carleton Univ. Ottawa. Ont. Canada, 1978.

[21] *Dijkstra, E. W.:* Solution of a problem in concurrent programming. Commun. of the ACM, September 1965. 569–570.

[22] *Lamport, K.:* A new solution of Dijkstra's concurrent programming problem. Commun. of the ACM, August 1975. 453–454.

[23] PDP–11 unibus interface manual. Digital Equipment Corporation, Marynard, Mass. 1975.

[24] *Brown, W.–Buhr, R. J. A.:* A concurrent high level language approach to the design of multiple microprocessor system. Proc. MIMI'77 (Zürich), Acta Press, Anaheim, Calgary, Zürich, 1977. 82–87.

[25] *Бронштейн, И. И.—Генинсон, Б. А.—Трахтенгерц, Э. А.:* Правило минимизации числа конфликтов в многопроцессорных вычислительных системах. Автоматика и телемеханика, март, 1979. 143—149.

[26] *Петров, П.—Николов, А.:* Об архитектуре мультимикропроцессорных вычислительных систем для управления процессами в режиме реального времени.

[27] *Balogh K.–Knuth E.–Salamon M.–Szlankó J.:* Párhuzamos programozás alapjai. KFKI report 1978–18.

[28] *Eufinger, R. J.:* Integrating peripherals into processing Systems. Computer Design, December 1978, 77–83.

[29] *Conway, M. E.:* A multiprocessor System design. Proc. AFIPS, FJCC 1963, Spartan Books, New York, 1963, 139–146.

TÁRGYMUTATÓ

A, Á

abszolút című adat 259
– címzés 25
ad hoc módszer 55
adatadó áramkörök 185
adatdimenziós megjelenítési mód 102
adatgyűjtő 100
adatmozgató utasítások 25
adattípusok 25
„adatutak kontúrjai" kirajzolásával való megjelenítési mód 102
adatútnevek 318
adatvevő áramkörök 185
ADT 262
AHPL nyelv 324
aktív eszközök táblázata 262
alapművelet 37
alárendelt processzor 37
algoritmus alapú ellenőrzés 153
algoritmusszint 275
állandó hibák 149
állapotadatok 173
állapotanalizátor indíthatósága 105
– szekvenciális indíthatósága 105
– – újra indíthatósága 106
– szelektív nyomkövetése 106
állapotátmenetek számlálásán alapuló módszer 163
állapotátmenet-számláló 160
állapotdiagrammok 71
állapotok számának megállapítása állapotanalizátor esetén 106
állapotregiszter 35
általános célú áramköri kártyák 89
álvétetlen bináris jelsorozat 161
– sorozatot előállító generátor 160
átlapot felfrissítés 38
analizátor adatfeldolgozó rész 101
analizátor értelmező rész 101
– kijelző egysége 102
analizátortár 101
ANSI 19
APU 36
áramköri emuláció módszere 78, 122, 123
– emulátormodul 130
áramkörkijelző-kereső 84
áramkörök, adatadó 185
–, adatvevő 185
–, hajlékony mágneslemezes tárat vezérlő 306
–, időzítő- 185
–, interfész- 173
–, késleltető- 185
–, kiválasztó 184
–, lekérdező 228
–, PPI 186
–, univerzális aszinkron adó-vevő 190
–, – szinkron adó-vevő
–, vonali fogadó- 185
áramnyomkövető 86
aritmetikai processzor 37
áruvizsgáló berendezések 160
ASCII 19
assembler nyelv 275
– program 74
assemulátoros miniszámítógép 96
asszimetrikus vonalak 252
aszinkron átvitel 174, 243
– forgásos prioritási technika 91
– nagytávolságú adatátvitel 227
asszociatív tároló elem 32
ATE rendszer 150
átfedéses programozás 255
– struktúra 271
áthelyezhető program 258
– programmodul 258
– tárgyprogram 298

átlátszó 36
– felfrissítés 38
átvitel, aszinkron 174, 243
–, – nagytávolságú 227
–, blokk 36
–, blokkos adat- 228
–, ciklusloposos 35, 173
–, csak állandó hosszúságú blokkok 250
–, – egyedül álló szavak 249
–, – változtatható hosszúságú 250
–, egyedül álló és változtatható hosszúságú blokkok 250
–, feltétel nélküli 176
–, feltételhez kötött 177
–, szinkron 174
–, visszajelzéses aszinkron 36
átviteli mód utasítás 190
átlóeltolás minta 156
átviteli parancskód 263
átvitt információ 242
automatikus felfrissítés 31
azonosítási mező 206

B

balról jobbra szabály 305
BASIC nyelv 304
battery back-up 31
BAU 36
báziscímregiszter 258
BCD 25
beépített függvény 305
be- és kimeneti műveletek kezelése 257
– – – processzorok 218
– – – regiszter 35
behelyettesítéses ellenőrzési módszer 152
belső adat 259
– felfrissítés 31
– parancs 267
benchmark módszer 338
BEP 38
BHLF 48
bináris file 267
bipoláris tárak 28
bitcímzési eljárások 280
bitorientált protokollok 38
bitszeletelt mikroprocesszor-elemek 329
blokkátvitel 36
blokkos adatátvitel 228
Bool-kifejezés feldolgozása 27
bootstrap módszer 307
BPNF 48
BSI 20
buboréktár 32

C

CAD 12
CAM 32
CAS 30
CB 226
CC 220
CCD 32
CCITT 19
CCW 226
CE jel 30
CDL nyelv 324
célállomás vezérelt OWC interfész 244
célrendszer 349
centralizált sínvezérlő 235
CHDL nyelvek 324
CLL 369
ciklikus redundancia kód 230
cikluslopásos átvitel 35, 173
címazonosító 280
címdekódolók 184
címjegyzék szervezésű logikai egység 257
címjegyzékfile 261
címke 280
címmező kezelés programozóberendezés esetén 143
címzési módok 279
CMOS 10
CMR 252
CODEC 35
compiler 74
CRC 37, 119, 160, 230
CRCC 165
crossbar kapcsolórendszer 365
CRT 165
CS jel 30
cursor 310
– jel 104

CS

csak állandó hosszúságú blokkok átvitele 250
– egyedül álló szavak átvitele 249
– változtatható hosszúságú blokkok átvitele 250
csapda 27

csapdautasítás 27
csatolóadatok 264
csatolóprogram 299
csatornavezérlő-szó 226
„csizmahúzó" rendszerű ellenőrzés 151
csoportos felfrissítés 31
– programozás 143
csővonal processzor 359
csővonalregiszter 331

D

D operátor 162
DDL nyelv 324
debugger 343
decentralizált sínvezérlő 235
dedikált mikroprocesszoros-analizátorok 100
diagnosztikai eszközök 121
– vizsgálat 149
DIMOS 12
dinamikus hibák 150
– RAM 29
dinamikusan mikroprogramozható 334
DMA 35, 194
– illesztőegység 194
– interfészegység 194
„dobozolt szoftver" 21
döntéshozatal 76
DOS 262
D-típusú bistabil multivibrátor 161
DUT 152

E, É

EAROM-elem 29
ECL 32
EEROM-elem 29
egy irányból vezérelt rendszerek 243
– lapkás mikroszámítógép 22
– lépés üzemmód 31
– mikroprocesszor-típushoz kidolgozott segédeszközök 138
egyedi hiba 150
egyedül álló szavak és állandó hosszúságú blokkok átvitele 250
– – – – változtatható hosszúságú blokkok átvitele 250
egyetlen szokásos processzor 359
egyirányú sín 251
egymás helyére dugható elemek 141
egyszerű adatkezelés 142
– típusú adatok 310
– utasítások 311
eljárásfej 313
eljáráshívás 28
eljárások 310, 313
ellenőrzés előre tárolt igazságtáblázat segítségével 154
– összehasonlítással 153
– programozóberendezés esetén 143
ellenőrző berendezés 144
ellenőrzőösszeg 230
– programozó berendezés esetén 143
ellenőrzött programfuttatás módszere 78, 123, 131
elosztott felfrissítés 31
– processzoros rendszer 359
– rendszerek 360
– színvezérlés 239
– sínvezérlő 235
elsőbbségi lánc 237
emuláció 348
emulálandó mikroprocesszorsín 130
EPROM 18, 28
erőforrások 255
értelmezőprogram 302
esemény tároló modul 137
eseményszavak 132
eseménytár 132
eszköz 257, 262, 267
eszközkezelő 257
eszközmeghajtó 262

F

FAMOS-elem 28
FDC 206
feladatmegfogalmazási szint 275
feladatmutató regiszter 222
feladatok kezelése 259
feladatütemezés 259
felfrissítés, átlapolt 38
felfrissítés, átlátszó 38
–, automatikus 31
–, belső 31
–, külső 31
–, csoportos 31
–, elosztott 31
–, versenymentes 38
felfrissítési ciklus 31
felhasználói program állapot 27

félszinkron sín 246
feltétel nélküli átvitel 176
feltételdetektáló modul 131
feltételes assemblerek 295
– fordítás 137
feltételhez kötött átvitel 177
felügyelőprogram állapot 27
fémbiztosítékok 140
fémezőmaszk 28
FET 10
FFT 37, 330
FIFO 32
file 261, 267
file-azonosítók 261
file-kezelő szubrutin 267
file-ok tulajdonságjelzői 264
file-térkép 266
Firmware Engineering 338
fizikai hiba 150
„fix" programok 63
flexibilis töréspontfeltételek 137
FM 37
FPGA 43
FPLA 43
FPLP 40
FPLS 45
FPRP 51
fordítóprogram 74
fokozatos rendszerépítés módszere 122
folyamatábra 72, 275
– szimbólumai 72
főprogram 179, 288
formulavezérlésű számítógépek 303
forráscím 242
forrásprogram 74, 277
forrásprogramfile 74
forrásvezérelt OWC interfész 243
funkcionális mikroprogramozás 332
– vizsgálat 149

G, Gy

gap 206
galopp minta 157
gazdarendszer 349
generátor jellemző polinómja 162
gépi utasítások 329
GIGO törvény 324, 349
globális adat 259
gyártás gazdaságossága 12

H

hajlékony mágneslemezes tárat vezérlő áramkörök 306
hand-shaking eljárások 243
hardver kidolgozása 64
hardver blokkon belüli hiba 76
hardver- és szoftverfunkciók együttes ellenőrzése 64
hardver kidolgozást elősegítő fejlesztési segédeszközök 81
hardverrel támogatott címeltolás 258
hardvertervezés 17
Harvard-osztályú számítógép 234
háttérprogram 179
helyi üzemmódban dolgozó IOP, 218, 220
hiba, állandó 149
–, dinamikus 150
–, egyedi 150
–, fizikai 150
–, hardverblokkon belüli 76
–, időszakos 149
–, keretezési 35
–, logikai 150
–, parametrikus 150
–, program- 76
–, statikus 152
–, véletlenszerű 149
hibafelderítő vizsgálat 149
hibajavítóprogram 343
hibakereső modul 130
– program 129, 133
hibrid programozók 49
HMOS 12
horizontális mikroutasítás felépítés 332
hosszanti paritáskód 230
hozzárendelt sínek 234
hurok megvalósítása 27
huzalozott prioritás 237

I

ID 206
IEC 19
időanalizátorok 100
idődimenziós megjelenítési mód 102
időmérés állapotanalizátor esetén 106
időnyújtás 104
időszakos hibák 149
időszeletes megszakításkérés 259
időzítőáramkörök 185

implicit címke 294
impulzusszámlálók 87
indexelt címzés 25
– – programszámláló segítségével 25
inicializáló utasítások 286
információkezelés 261
integrált áramkörökre csíptethető sínkijelző 87
– fejlesztői rendszer 121
intelligens be- és kimeneti vezérlők 218
interfészáramkörök 173
IOP 218
– utasításkészlete 223
írásra visszatérő minta 158
– – galopp minta 157
ISO 19
ISP nyelv 324
ISPS nyelv 324
jelátvivő vezetékek 251
jelzőszám 160
jelzőszám-analízis 160

K

kapcsolási rajz 72
karakterorientált protokollok 38
karakter sorok 306
kártyavizsgáló 160
„kemény" mikroprogramozás 332
képernyős megjelenítő-vezérlő 210
képinformációtár 216
keresési művelet 38
kérés/visszaigazolás típusú rendszerek 243, 244
„kereszt" jelző 95
keresztassembler 95
kereszthivatkozási táblázat 284
keresztparitás 230
keresztreferencia-lista 342
keresztrudas kapcsolórendszer 365
– sín 235, 236
keretezési hiba 35
keretjel 242
késleltetőáramkörök 185
kétirányú sín 251
kétszintű programozás 330
kézfogásjelek 188
kidolgozási költség 68
kihozatal 12
kijelölt időszakasz módszer 242, 243
kiválasztó áramkörök 184
kódolt formájú mikroutasítás 332
kompilátor 302
„konfigurálási" parancsok 133, 180
kontaktmaszk 28
kontrollerek 22
kontroll-szumma 230
korai beírás 31
körbeléptetett prioritási módszer 237
korlátozott pontosságú aritmetika 27
közös modusú jelek 252
– sínek 233, 235
központilag időzített sín 242
központosított független kérések módszere 239
– kérdező módszer 237
– soros lánc módszere 237
– sínvezérlés 237
– sínvezérlő 235
közvetett címzési mód 279
– regisztercímzés 23
– – előzetes címcsökkentéssel 23
– – utólagos címcsökkentéssel 23
közvetlen címzés 25
– címzési mód 279
– regisztercímzés 23
– tárhozzáférés 194
– változók 311
– vonalak segítségével való összeköttetés 291
közvetlenül csatolt címzési mód 279
kritikus szakasz 368
kulcsszavak 304
külső adat 259
– felfrissítés 31
– parancs 267
kvázistatikus RAM 29

L, Ly

lapműködési mód 30
lazán csatolt processzorok 363
LD 283
LFT 262
lekérdezéses megszakításkérés-azonosítás 179
lekérdező áramkörök 228
lépésenkénti programvégrehajtás módszere 77
léptetőregiszter 161
léptetőregiszter-elrendezés 162
LIFO 32

logikai állapotanalizátorok 100
– analizátorok 100
– diagramkijelzők 100
– egység 257
– file-ok táblázata 262
– hiba 150
– impulzusadó 84
– jelszint kijelző 84
– komparátorok 87
– programozók 49
LSI 9

M

mag 368
mágneslemezes operációs rendszer 129
major módok 216
makró 291
makrodiagnosztikai program 151
makroprocesszorok 303
makroutasítás-dekódoló 349
makroutasítások 137, 291, 329
manipulációs utasítás 37
másolás 264
– programozóberendezés esetén 143
maszkprogramozható ROM 28
MC 6800 utasítás készlete 26
Mealy-gép 46
Mealy-állapotdiagramok 71
megfigyelőprogram 343
„megfoltozás" 51
megjegyzés 281
megnyitott file 264
megosztott helyi tár 90
megszakítási lánc 183
– szubrutin 179
megszakításkérési vektorokkal történő azonosítás 181
megszakítható összeköttetés 40
megszakított összeköttetés 40
memorizáló képernyős oszcilloszkópok 100
menet 285
menetelő minta 155
mester processzor 37
mező 310
MFM 32
– kódolási szabály 202
mikroállapot-táblázat 343
mikrodiagnosztika 151
mikrokódolás 330
mikrooperáció 332
mikroprocesszor független makro-assemblerek 295
mikroprocesszor-analizátorok 113
mikroprocesszorok 22
mikroprocesszor-szonda 113
mikroprogram karbantartása 348
mikroprogramcím-regiszter 334
mikroprogramcím-számláló 334
mikroprogram-készítés 338
mikroprogramozott mikroprocesszorok 67
– rendszer 332
mikroszámítógép típusú rendszerek 62
mikroszámítógépek 11, 92
mikroutasítás-képző 330
mikroutasítás-regiszter 331, 351
mikroutasítások 329
mikrovezérlő egység 349
MIMD 359, 360
miniszámítógép – mikroszámítógép csatolóegysége 96
minta, átlóeltolás 156
–, galopp 157
–, írásra visszatérő 158
–, menetelő 155
–, pillangó 158
–, sétáló 155
–, oszlop galopp 157
minor módok 215
mintaérzékenység 150
mintavételi jel 227
MISD 359
MMU 36
mnemonic 21
mnemonikok 114
MODEM 35
módmeghatározó parancs 186
modul 313
moduláris multiprocesszoros rendszer 90
monitor 343
monitorprogram 99, 129
Moore-állapotdiagramok 71
Moore-gép 46
Moore-törvény 10
MOS 10
MOSFET 140
MOS-tárak 28
MPS 263
MSC ellenőrzési módszer 151
MSI 9
MTFB 14
működési algoritmus 72

multifeldolgozás 366
multimikroprocesszoros rendszerek 359
multimikro-számítógép 363
multiplexerek 185
multiprocesszoros rendszerek 363
multiprogramozás 366
műveleti kód 281
M²FM kódolási szabály 206

N, Ny

n készülék programmal való ellátásának költsége 68
n_k kritikus darabszám 69
nagy impedanciájú összeköttetés 140
nanotár 332
nanoutasítás 332
NBS 38
nem áthelyezhető program 258
– címjegyzék szervezésű logikai egység 257
– hozzárendelt időszakasz módszer 242, 243
– – sínek 233, 235
– kódolt formájú mikroutasítás 332
NMOS 10
nyelv, AHPL 324
–, BASIC 304
–, CDL 324
–, CHDL 324
–, DDL 324
–, ISP 324
–, ISPS 324
–, PASCAL 315
–, PL/1-szerű 308
–, PLZ 309
–, PLZ/SYS 309
–, programösszeállító 301
–, rendszerprogramozási 302
–, veremtár-orientált 315
nyomkövetés módszere 78
nyomkövető 99
– tár 132
nyomozóprogram 343

O, Ö

OE jel 30
olvasás szempontjából egyenértékű elemek 141
operációs rendszer 255
operandusok 279
optikai csatolók 252
OS 255
OS csatolóprogramja 258
oszcilloszkópok 83
oszcilloszkóp-szinkronjel 132
oszlop galopp minta 157
oszlopcím 30
osztott független kérések 240
– idejű közös sínrendszer 363
– soros lánc 240
– végigkérdezés 240
OWC 243
önellenőrzés 150
önfeltöltéses üzemmód 201
összetett művelet 37

P

PASCAL 315
PAL 43
PALASM 48
paramétervektor 263, 264
parametrikus vizsgálat 149
parancs üzemmód 278
párbeszédes programozás 302
paritásbit 230
párhuzamos processzor rendszer 359
PB 226
PCB adatstruktúra 367
PERCOM 20
perifériák 173
personality board 30
personality-modul 115...118
P-kód 315
pillangó minta 158
pipeline-regiszter 331
PLL 38
PLZ programnyelv 309
PLZ-ASM assembler 309
PLZ/SYS programozási nyelv 309
PL/1-szerű nyelvek 308
PMS nyelv
PPI 19
– áramkör 186
PRBS 160
precedencia-szabály 305
Princeton-osztályú számítógép 235
prioritásszintek 183
postaláda tár 365
processzor, alárendelt 37
–, be- és kimeneti 218
–, csővonal 359

processzor, egyetlen szokásos 359
–, lazán csatolt 363
–, szorosan csatolt 363
–, társ- 37
program 72
–, áthelyezhető 258
–, csatoló- 299
–, „fix" 63
–, hibajavító- 343
–, hibakereső 129, 133
–, makrodiagnosztikai 151
–, monitor- 99, 129
–, szerkesztő- 74, 95, 277
–, szövegszerkesztő 277
–, szűrő- 310
–, teszt- 78
–, háttér- 179
programdokumentáció 74
programelágazások 288
programhiba 76
programhurok 54, 287
programírás költsége 68
programkódolás 277
programlistafile 309
programmal lekérdezett megszakításkérések prioritási szintje 183
programmegszakításos átvitel 179
programnyelv tárkapacitásigény növekedési tényezője 68
programok kidolgozása 64
programozható eszköz 186
– időzítő-számláló 192
– logika 40
– megszakításkérés–kezelő egység 194
– perifériaillesztő 186
– soros átviteli illesztőegység 190
programozóberendezés 142
programozott adatátvitel 173, 174
– logika 11, 40, 62
programösszeállító nyelv 301
programsor 68
programstatisztikai funkciók 137
PROM 18
– -elem 29
prototípust emuláló eljárás 75
pszeudoutasítások 285
pufferváltozó 318
puffertárelemek 32
„puha" mikroprogramozás 332

R

RALU 330
– szelet 331
RAM 28
RAS-only mode 30
redundancia ellenőrzéskód 160
referencia csipesz 87
regiszterek 185
regisztertükör 99
rejtjelző egység 38
rekordazonosítók 261
rekurzív makrók 296
relatív bázisregiszteres címzés 25
– cím 280
– címzés mód 279
– programszámlálón keresztüli címzés 25
relokálható programírás 137
– programmodul 258
rendszer befagyása 170
– funkcióinak meghatározása 64, 65
– specifikációja 64, 65
rendszer-definiálás 17
rendszeremuláció 349
rendszerprogramozási nyelvek 302
rendszertervezés 64, 65
REPROM 18
RIO 262
rögzített szimbólumok 283
– utasításkészletű mikroprocesszorok 67
ROM 28
ROM-szimuláció 96

S

SA 160
SBMOS 12
SCANNER 228
scanning 34
SCB 225
sétáló minta 155
SIMD, 359, 360
sín szélessége 250
sínen keresztüli összeköttetés 251
sínkijelzők 87
sínrendszer paraméterei 233
sínrendszer-analizátorok 118
sínvezérlők 87
SISD 359
SOC 226
„soft-sectoros" rögzítés 204

Software Engineering 338
sorcím 30
soros vonali jelek analizátorai 118
SP 290
SSI 9
SSP 37
START bit 228
statement 277
statikus mikroprogramozhatóság 334
– RAM 29
státuszregiszter 35
string 306
stringállandó 306
stringkonvertáló függvények 306
stringváltozók 306
strobe jel 227
strukturális mikroprogramozás 332
strukturált programozás 301
– típusú adatok 310
sync-karakter 228
sync-keresési mód 228

SZ

szektor 206
szektorközök 206
szekvenciális rész 367
szeletenkénti jó termékek száma 13
szemafor 368
szerkesztőprogram 74, 95, 277
szervizrutin 99
szilárd mag ellenőrzési módszer 151
szilícium „biztosítékok" 140
szimmetrikus jel 252
– vonalak 252
szimuláció 323
szimulátoros eljárás 75
szinkron nagytávolságú adatátvitel 227
– sín 242
szinkronjel 242
szinkronkarakter 228
szinkronizációs primitívek 368
szintátalakítók 185
szisztematikus hiba 150
szó 249
szoftver „készítés" 17
szoftverrel megoldott relokálás 258
szoftvermódszer 303
szoftver-rendszerspecifikáció 275
szoftverszámláló 287
szondák 100
szorosan csatolt processzorok 363
szövegbeviteli üzemmód 278
szövegszerkesztő program 277
szubkommunikáció 243
szubrutin 289
– hívás 289
szuperkommunikáció 243
szürke doboz leírási szint 324
szűrőprogram 310

T

tár ára 68
– kezelése 258
„tárbaágyazott" be-, ill. kimeneti műveletek 174
tárgykód 284
tárkizárás 368
tárolócsöves oszcilloszkópok 83, 100
tárrezidens rész 266
társprocesszor 37
társzegmentálás 226
tárvizsgáló algoritmus 155
TAS 366
távsínvezérlő 218
TB 226
teljes cím 280
tesztprogramok 78
totemoszlop 18
több processzoros rendszer 359
– „puha" mag ellenőrzési módszer 151
többszörös hozzáférést lehetővé tevő tárak 32
– hozzáférésű tárakat alkalmazó rendszer 365
tömbkezelés 27
töréspont 132, 136, 343
TP 222
TPU-egység 36
tracer 343
transparent 36
TRC 160
trigonometrikus művelet 37
TTL 9
tűágyas ellenőrző berendezések 150
tüskejel-érzékelő 101

U

UART 19, 190, 228
UCSD 318

univerzális sínek 233, 235
– aszinkron adó-vevő áramkörök 190
– fejlesztői rendszerek 138
– mikroprocesszor-analizátorok 100
– szinkron–aszinkron adó–vevő áramkörök 190
UPC 38
USART 190, 228
USRT 228
utasítás, adatmozgató 25
–, *átviteli mód* 190
–, inicializáló 286
–, makró- 329
–, manipulációs 37
–, mikró- 329
–, pszeudó- 285
utasításemulátor 349
utasításkód 280
utasításszimulátorok 325, 349
utasításszint 275

V

VDG 210
– alfanumerikus üzemmódja 214
– félgrafikus üzemmódja 214
– grafikus üzemmódja 215
vektor 181
vektormutató 263
véletlenszerű hibák 149
veremtár 289
veremtárelemek 32
veremtármutató 27, 290
veremtár-orientált nyelv 315
versenymentes frissítés 38
vertikális mikroutasítás felépítés 332
vezérlő parancsregiszter 35
vezérlőmaszk 28
vezérlőtár 329
vezetett hibafelderítés 150
visszafelé nyomozás módszere 150
visszajelzéses aszinkron átvitel 36
visszatérés 289
VLSI 10
vonalak 251
vonali fogadóáramkörök 185
– meghajtóáramkörök 185
– protokollok 38

Z

zónázott formátum 306

W

watch-dog alkalmazása 170
WCS 334
Winchester típusú diszkek 38
wire-wrap kötés 89

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató

3337 Szegedi Nyomda
Felelős vezető: Dobó József igazgató

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Mózer István
A borítót és a kötést tervezte: Székely Edit
A könyv ábráit rajzolta: Remete Katalin
A könyv formátuma: B5 — Ívterjedelme: 33,625 (A5)
Ábrák száma 220 — Példányszám: 3800
Papír minősége: 80 g ofszet — Betűcsalád és -méret: New Times gm/gm
Azonossági szám: 61 221 — MŰ: **3281-i-8385**
Készült az MSZ 5601 és 5602 szerint
Ez a könyv az 1981. évi kiadás 2. javított utánnyomása