4.6. Programozható logikai áramkörök

A programozható logikai áramkörök olyan logikai áramkörök, amelyek programozható memóriaként is használhatók.

Megvizsgálva egy több kimenetű kombinációs hálózat igazságtáblázatát, azt tapasztaljuk, hogy adott feltételek mellett ez megfelelhet egy memóriának is. A **4.16.** ábrán egy kombinációs hálózat igazságtáblázata látható.

A	B	F_0	F_1	F_2	F_3
0	0	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	1
1	1	1	1	1	0

A és B a kombinációs hálózat bemenetei

F_0, F_1, F_2 és F_3 a kombinációs hálózat kimenetei

4.16. ábra. Egy kombinációs áramkör igazságtáblázata

Az áramkör bemenetére adott kombinációk megfelelnek a memória címzésének, a kimenetén megjelenő válaszok pedig az adott címeken eltárolt tartalomnak. Pl. a 01-es címen az 1010 adat található.

Az egyes kimenetek logikai függvényei felírhatók. Ezek a következők:

$F_0 = B$, $F_1 = \overline{A} \cdot \overline{B}$, $F_2 = \overline{A} + B$, $F_3 = A \cdot \overline{B}$.

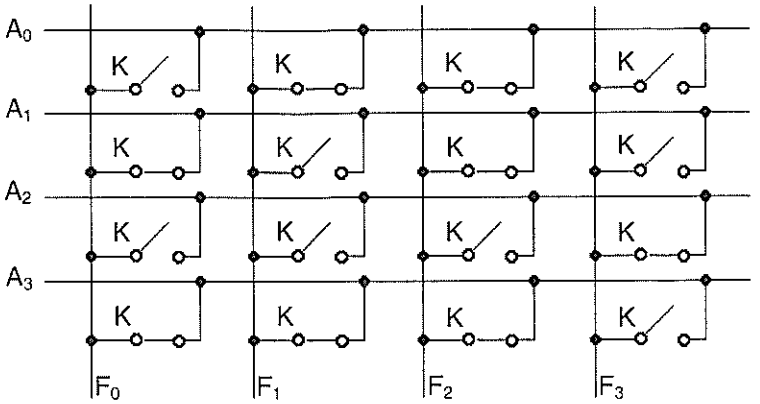
Ezek a logikai függvények ponált és negált bemeneti változókat is felhasználva egyszerűen megvalósíthatók ÉS-VAGY rendszerrel. Ezzel el is készült az adott igazságtáblázat szerint működő kombinációs hálózat. Több bemenet és kimenet esetén a megvalósítás már egyre több kapuáramkört igényel, és ez gondot jelenthet. Ezért a kapuáramkörök helyett ún. **mátrix elrendezésű kapcsolómezőket** használhatunk. Az ebben lévő kapcsolókat mindig a megvalósítandó függvényeknek megfelelően állíthatjuk (programozhatjuk) be.

Egy ilyen mátrix egyszerűsített elvi rajzát láthatjuk a **4.17.** ábrán.

Az egyes vonalak feladata a következő.
A_0–A_3: címvezetékek;
F_0–F_3: kimeneti vezetékek.

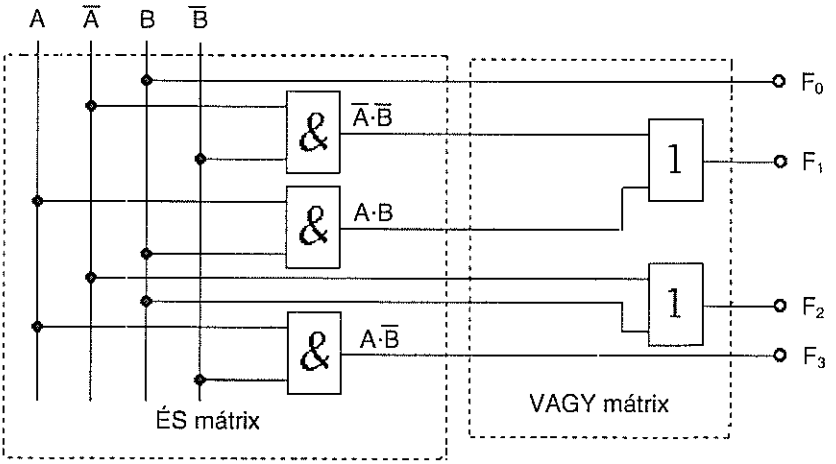
A hálózat működése: Mindig csak egy sor aktivizálható, ugyanakkor a többi nem kap engedélyt. Így pl., ha $A_0 = 1$ (A = 0 és B = 0), akkor F = 0110 információ lesz a kimeneten, ill. ha $A_1 = 1$ (A = 0 és B = 1), akkor F = 1010 információ lesz a kimeneten.

A gyakorlatban az egyszerű kapcsolók helyett MOS tranzisztoros elektronikus kapcsolókat használnak.



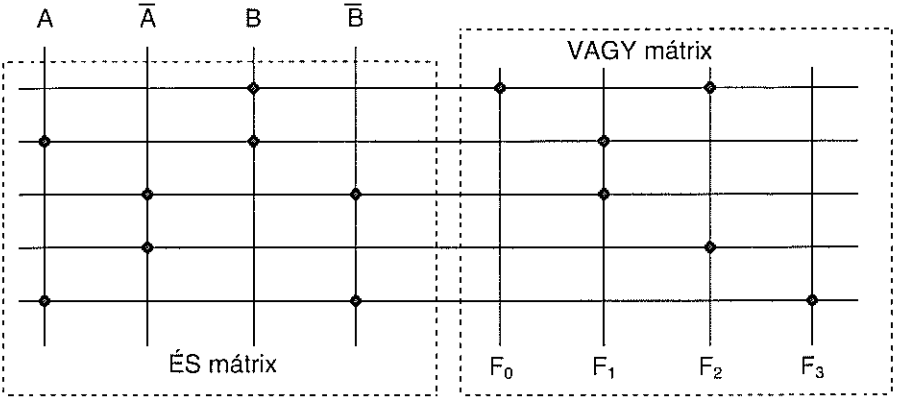
4.17. ábra. Kapcsolókkal felépített PLA áramkör

Ha bemeneti jelként az A és B változókat kívánjuk használni, akkor a függvényeket ún. kétszintű hálózatokkal tudjuk megvalósítani. A bemeneten rendelkezésre áll minden változó ponált és negált alakja is. Ilyen kétszintű hálózat megvalósításához szükség van egy ÉS mátrixra és egy VAGY mátrixra. Az ÉS mátrixszal előállítjuk a szükséges mintermeket, a VAGY mátrixszal pedig ezek VAGY kapcsolatba hozásával a kimeneti függvényeket. Ezzel a mátrix struktúrával bármilyen logikai függvény, ill. memória elkészíthető vagy programozható, és ez adja az alapját a PLA megvalósításoknak. Az előző függvények megvalósítása ÉS–VAGY struktúrában a **4.18.** ábrán látható.



4.18. ábra. ÉS – VAGY struktúrában megvalósított áramkör

Ugyanezeknek a függvényeknek a PLA áramkörös megvalósítása a **4.19.** ábrán látható.



4.19. ábra. Egy PLA-val megvalósított áramkör

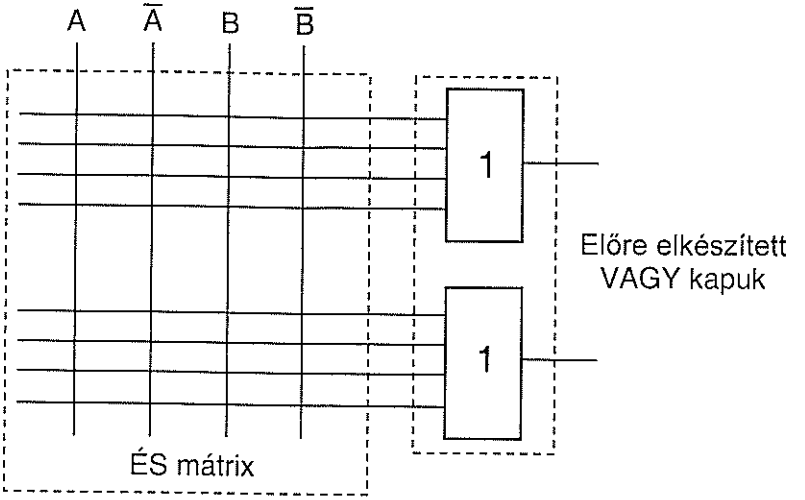
A PLA áramkörös kapcsolás működése: A pontokkal jelölt helyeken kell a kapcsolóknak zárva lenniük (vezetniük), a többi kereszteződési helyen viszont nincs összeköttetés. A PLA áramkörök programozása elég körülményes, és speciális berendezést igényel.

FPLA áramkörök

Az FPLA (Field Programmable Logic Array) áramkörök olyan PLA-k, amelyek a felhasználók által is egyszerűbben programozhatók. Felépítésük annyiban különbözik a PLA-któl, hogy a VAGY mátrixot már a gyártás során kialakítják és így csak az ÉS mátrixot kell programozni. Ezeket rövidítve PAL (Programmably Array Logic) áramköröknek is nevezzük. Vázlatos felépítésük a **4.20.** ábrán látható.

Az eddig bemutatottakon kívül napjainkban még sokfajta programozható logikai áramkört fejlesztettek ki. Ezek közül csak felsorolunk néhányat:
PIA (Programmable Interconnect Array): programozható összeköttetésű mátrix.
MAX (Multiple Array Matrix): többszörözött mátrix-együttes.
CLB (Configurable Logic Block): konfigurálható logikai blokk.
LCA (Logic Cell Array): logikai cellaegyüttes.

Ezeket az áramköröket összefoglaló néven PLD-nek (Programmable Logic Device), programozható logikai eszközöknek nevezzük. Láthattuk, hogy ezek mindegyike univerzálisan felhasználható. Az adott feladat elvégzéséhez a mátrixba beírjuk (beégetjük) a megfelelő működési táblázatot vagy logikai hálózatot. Egy ilyen programozható áramkör (PAL) rajza látható a **4.20.** ábrán.



4.20. ábra. Egy programozható áramkör felépítése

Cellabázisú összetett áramkörök

Ezeknél az áramköröknél a belső programozás mellett lehetőség van a cellák kapcsolatait is beállítani. Ilyen PLD-ket fejlesztett ki a XILINX cég is. Ez a rendszer LCA-n alapul, és a cella-együtteseket többféle elv szerint is konfigurálhatjuk (CLB).

4.7. A számítógépekben alkalmazott memóriák

A következőkben röviden áttekintjük az eddig tárgyalt memóriafajták leggyakoribb felhasználási lehetőségeit.

ROM-BIOS (ROM Basic Input Output System)

A legfontosabb ilyen áramkör a számítógépek alaplapjain található, és az operációs rendszer legfontosabb programrészeit, ill. adatait tartalmazza. A számítógép bekapcsolásakor és újraindításakor ezek a programok futnak le. A ROM-BIOS felépítését tekintve PROM, EPROM, EEPROM vagy újabban flash-ROM áramkör lehet. A PROM és EPROM áramkörök tartalma nem módosítható. Újítás esetén ezeket a tokokat ki kell cserélni. Az EEPROM-ok és a flash-ROM-ok tartalma módosítható.

Operatív tár (RAM)

Ez egy nagyméretű írható-olvasható tár. A számítógép bekapcsolása után ide töltődnek be a legszükségesebb rendszerprogramok (memóriarezidens programok). Egy adott szoftver futtatásakor (használatakor) ide töltődnek be a háttértárból a szükséges programrészek és adatok.

Felépülhet SRAM-ból vagy DRAM-ból, de ma már főként DRAM-okat használnak. Elérési idejük kb. 60–70 ns. Ezeknek is különböző változatait fejlesztették ki, pl. ilyen az SDRAM (Synchronous DRAM). Általa csoportos olvasásra van lehetőség, így az elérési idő 40–50 ns-ra csökkenthető. A DRAM-ok adatszélessége különböző lehet, pl. 8, 9, 32, 36, 64, 72 stb. bit.

Megjegyzendő, hogy bájtonként egy ún. paritásbitet is alkalmazhatnak (így 9, 36 vagy 72 bites szóhosszak jöhetnek létre), amelyek felhasználásával hibafelismerés végezhető.

Ezeket a memóriaegységeket kis kártyákon helyezik el és így csatlakoztathatók az alaplapon a megfelelő helyekre. Ezeknek a kártyáknak a neve SIMM modul (Single In line Memory Module). Ezek 32 vagy 36 bites adatszélességűek és a 486-os alaplapokhoz használatosak. A Pentium alaplapokon a DIMM (Dual In line Memory Module) kártyákat használják. Ezek 64 vagy 72 bites adatszélességűek.

Shadow RAM (árnyék RAM)

Ez az operatív tár egy része. A ROM-BIOS tartalma átmásolható a RAM meghatározott területére és onnan sokkal gyorsabban elérhető, mint a ROM-ból. Hátránya, hogy a RAM-ban helyet foglal.

Video-RAM

A videokártyákon használatos RAM-fajta. Ebben tárolódik a megjelenítendő kép, és a vezérlőáramkör ezt viszi ki a képernyőre. Ez a fajta RAM két adatkapuval rendelkezik, a processzor felől egy párhuzamos bemenettel (gyors képátvitel), a monitor felé pedig egy soros kimenettel. Ezzel a megoldással egy időben lehet írni és olvasni is. Ezeket a memóriaegységeket DRAM-okból készítik és kapacitásuk néhány (1–16) Mbájt.

CMOS RAM

Ezt setup memóriának is nevezik. A rendszer legfontosabb alapbeállításait (paramétereit) tartalmazza. Közvetlenül a gép bekapcsolása után érhető el, fontos, hogy a tartalma a gép kikapcsolása után is megmaradjon, ezért ezt egy, az alaplapon lévő akkumulátor táplálja. Megvalósítására kis fogyasztású, CMOS technológiával gyártott áramköröket használnak. Kapacitása kicsi, kb. 64 bájt.

Ellenőrző kérdések

1. Ismertessük a memóriákkal kapcsolatos alapfogalmakat!
2. Csoportosítsuk a tárakat a fizikai működési elvük szerint!
3. Csoportosítsuk a tárakat az adat hozzáférési módja szerint!
4. Ismertessük a statikus memóriacellák felépítését és működését!
5. Ismertessük a dinamikus memóriacellák felépítését és működését!
6. Ismertessük a félvezetős memóriák fajtáit!
7. Ismertessük a memóriacellák szervezésének módszereit!
8. Ismertessük a ROM áramkörök különböző fajtáit!
9. Ismertessük a RAM áramkörök különböző fajtáit!
10. Ismertessük a memóriamodulok összekapcsolásának lehetőségeit!
11. Ismertessük az asszociatív memóriák felépítését és működését!
12. Ismertessük a programozható logikai áramkörök felépítését és működését!
13. Ismertessük a memóriák gyakorlati alkalmazását!

5. TÁROLÓKEZELÉS

Az előző fejezetben a különböző tárakról, és ezek közül is elsősorban a félvezetős tárakról, a memóriákról volt szó. Most a teljes tárolórendszer működésének megismerése a célunk.

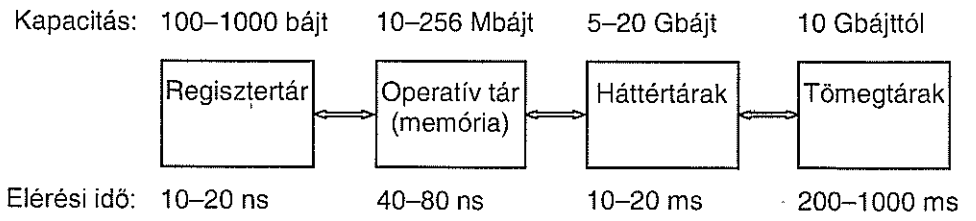
A programok és az adatállományok méreteinek növekedése egyre nagyobb kapacitású tárakat igényel, ezzel sajnos többnyire nő a hozzáférési idő. Az új technológiák bevezetésével a méret szinte állandónak tekinthető, az elérési időt pedig egyéb megoldásokkal (pl. cache alkalmazásával) igyekeznek csökkenteni.

5.1. Tárhierarchia

Az új technológiákkal és áramköri megoldásokkal a mikroprocesszorok (μP) működése egyre gyorsabb lesz. Ez viszont csak úgy használható ki, ha a hozzá kapcsolódó különböző tárak elérési ideje csökken.

A μP minél gyorsabb kiszolgálásának érdekében ún. **tárhierarchia** épült ki.

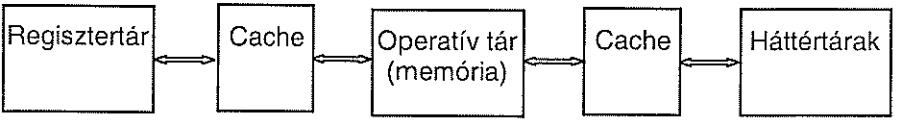
A tárhierarchia a különböző felépítésű és működésű tárak rendszere. Lényege, hogy a μP-hoz logikailag legközelebb eső tárolóegység legyen a leggyorsabb működésű (de ezzel együtt jár, hogy a legkisebb kapacitású is), ebben legyenek tárolva a műveletvégzésekhez szükséges operandusok. Majd logikailag a μP-tól távolodva egyre nagyobb kapacitású tárolóegységek következnek, amiknek sajnos ezzel arányosan növekszik az elérési idejük is. A tárhierarchia az **5.1.** ábrán látható.



5.1. ábra. A tárhierarchia rajza

Az egyes tárak feladata, jellemzői a következők.
Regisztertár: A μP-on belül található nagyon kis kapacitású, gyors működésű tár.
Operatív tár: Ezt főtárnak, memóriának vagy RAM-nak is nevezik. Felépítését és működését részletesen megismerhettük a 4. fejezetben.
Háttértárak: Ide sorolható elsősorban a merevlemezes egység (winchester), a CD és floppy lemez. Kapacitásuk egyre nagyobb, de nagyon fontos, hogy az elérési idejük kb. 1000-szerese az operatív tár elérési idejének.
Tömegtárak: Ide elsősorban a mágnesszalagos tárolási módot soroljuk, amelyet biztonsági mentéseknél, archiválásoknál használunk.

Az adatátviteli sebesség növelésének és a μP munkájának folyamatossá tétele érdekében, gyorsítási és pufferelési céllal, az egyes szintek közé kis kapacitású gyorsítótárakat (cache, ejtsd: kes) helyeznek el (**5.2.** ábra). Ezek a felhasználó számára láthatatlanok (hozzáférhetetlenek).



5.2. ábra. A cache-ekkel kibővített tárhierarchia

A különböző tárak kezelése a tárkezelő egység (Memory Management Unit, MMU) feladata. Ez az egység lehet a μP-on belül, de azon kívül is.

MMU (Memory Management Unit) feladata a különböző táregységek hatékony működtetése, a különböző címzési módokkal (l. a 2.3.2. pontot) megadott címek átalakítása abszolút (fizikai) címekké, valamint a tárkezeléssel kapcsolatos szervezési és védelmi feladatok ellátása.

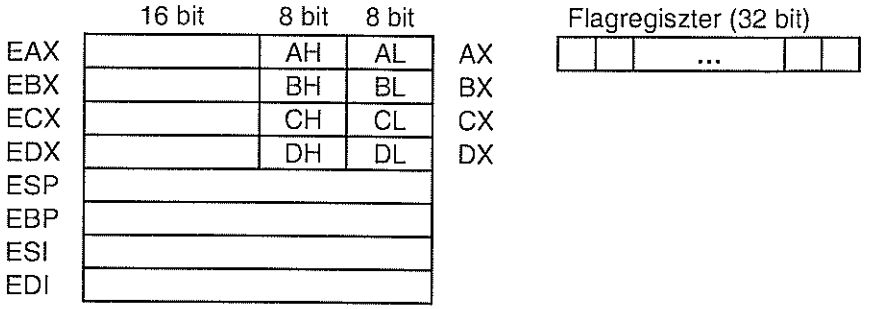
5.2. Regisztertárak kialakítása és alkalmazása

A μP-on belül található viszonylag kis kapacitású (8, 16, 32, 64 bites) tároló. A μP-ok fejlődésével a bennük lévő regiszterek száma is növekszik, és ez gyorsabb műveletvégzést tesz lehetővé.

Megkülönböztetünk általános és speciális célú regisztereket. Ezek együttesen alkotják a regisztertárat. Egy ilyen regisztertár részlete látható az **5.3.** ábrán.

Az első négy regisztert (EAX, EBX, ECX, EDX) általános célú regiszteреknek nevezzük, de mindegyiknek van valamilyen speciális feladata is (pl. címzés, ciklusszámlálás, operandusok átmeneti tárolása stb.).

Látható, hogy az általános célú regiszterek 16 bites, ill. 8 bites egységenként is kezelhetők. Ez azért szükséges, hogy a korszerűbb 32 bites számítógépeken is lehes-



5.3. ábra. Egy regisztertár részlete

sen futtatni az egyszerűbb (8,16 bites) rendszerekre írt programokat (felülről kompatíbilis rendszerek).

A következő négy regiszter (ESP, EBP, ESI, EDI) speciális célú regiszterek.

Ezeket a veremtár kezelésére és indexelt címzésekhez használják.

A veremtár az operatív memória területén kialakított zsákszerű tár, amelyből a beletett adatok fordított sorrendben vehetők ki. A működése azon alapul, hogy az utoljára beletett adat a bennlévők fölé kerül és ehhez lehet hozzáférni először, mert ennek a címe van az ún. veremmutatóban. Ezt a működési módot LIFO (Last In – First Out, az utoljára betett vehető ki először) elvnek is nevezik.

Egy speciális célú regiszter még a **flagregiszter**, ami az egyes különálló jelzőbiteket (pl. zérusbit, előjelbit stb.) tárolja. Elsősorban gépi kódú programozáskor az elágazások kezelésében van nagy jelentősége.

A regisztertár kezelése

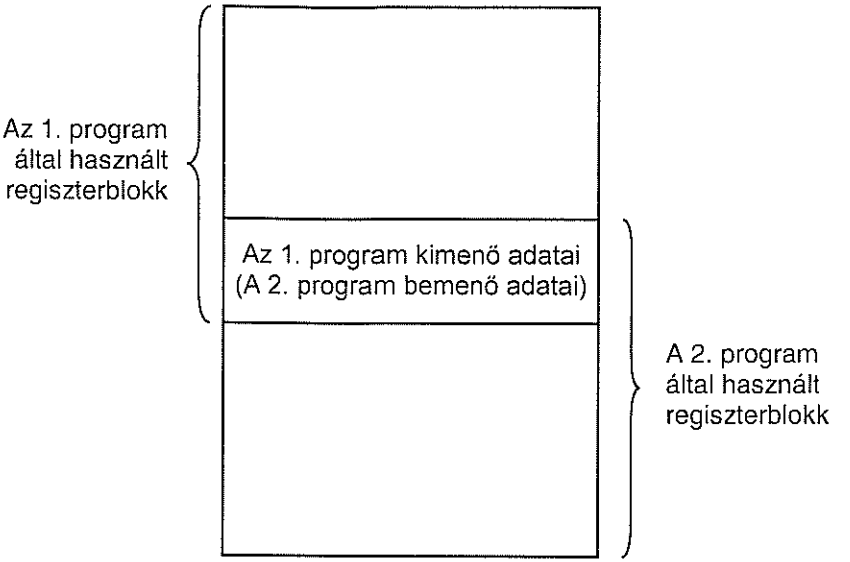
A korszerűbb μP-okban a regisztertárak jelentősége egyre nagyobb. A μP egy feladat futtatása közben a regisztereknek csak egy részét használja (látja). Ezeknek az egységeknek a kialakítására, kezelésére különböző módszereket dolgoztak ki, amelyek közül a legelterjedtebbek a regisztercsoportos, az ablaktechnikás és a blokktechnikás eljárások.

Regisztercsoportos kezelési eljárás

A regisztertárat azonos méretű, nem átlapolódó részekre (csoportokra vagy ún. bankokra) osztja fel. Egy-egy ilyen csoport mérete 2 valamely hatványa. Elég merev felosztás, ma már nem nagyon használatos.

Ablaktechnikás kezelési eljárás

A regisztertárat azonos méretű átlapolható részekre osztja fel. Az egység (ablak) mérete 2 valamely hatványa. Az ablaktechnika elsősorban arra alkalmas, hogy az egyes programrészek között megkönnyítsük a paraméterátadást. Ez a leggyakrabban alkalmazott megoldás, amire egy egyszerű példa látható az **5.4.** ábrán.



5.4. ábra. Az ablaktechnika elvi felépítése

Látható, hogy az 1. program által létrehozott eredményeket (kimenő adatokat) a 2. program mindenféle áthelyezés nélkül tudja használni, a közös rész mindkét programból könnyen elérhető.

Blokktechnikás kezelési eljárás

A regisztertárat tetszőleges méretű átlapolható részekre osztja fel, és ezzel az ablaktechnikánál rugalmasabban használható. Működési módja és használata az ablaktechnikához hasonló, de a tetszőleges blokkméretek miatt bonyolultabb a kezelése.

5.3. A cache-tárak

A cache a μP munkáját folyamatosabbá teszi azzal, hogy a szükséges adatokat viszonylag gyorsan a rendelkezésére bocsátja. Láttuk, hogy a tárhierarchiában több helyen is alkalmazzák, de legfontosabb szerepe a μP és a főtár között van.

A cache-tár lehet a μP-on belül és a μP-on kívül is. A belső cache mérete kb. 8–64 Kbájt, a külsőé tipikusan 64–512 Kbájt. Ezek a tárak a felhasználó részéről nem hozzáférhetők. A μP tárolhat benne utasításokat és adatokat is, de vannak olyan

megoldások is, hogy csak utasításokat (utasítás cache), ill. csak adatokat (adat cache) tárol bennük.

A cache-tárban a memória egy összefüggő részének tartalmát tárolják a memóriabeli hely címének egy részével együtt. A memória és a cache-tár közötti adatátvitel mindig blokkos formában történik (egyszerre több bájt átvitele kerül sorra). Ez azért van így, mert a program futása során nagy a valószínűsége annak, hogy egymás után következő utasításokra és adatokra van szükség, és így kevesebbszer kell a főtárhoz fordulni. A μP előre betölti a memóriából azokat az információkat, amelyekre valószínűleg rövidesen szüksége lesz.

A cache-tár egy CAM felépítésű tár, így az adatok keresése tartalom szerint (aszszociatív módon) történik. Ez konkrétan azt jelenti, hogy a μP az adat keresésekor annak címét hasonlítja össze a cache-ben lévő adatok címeivel, ill. címük egy részével. Ha az megtalálható a cache-ben, akkor találatról (cache-hit), ha nincs bent, akkor hibáról (cache-miss) beszélünk. Ekkor sajnos az adatot a főtárban kell keresni, ami több időt vesz igénybe és lassítja a program futását.

5.3.1. A cache-tárak típusai

A cache-tárakban az adat visszakeresése annak címe (vagy a cím egy része) alapján történik, ezért a cache-tárban az adat mellett a címének egy részét is tárolni kell. Ezt a részt **tag-nak** (ejtsd: teg) nevezzük. Az adatátvitel és a tárolás 4–64 bájtos blokkonként történik. Egy blokkra egy címmel hivatkozhatunk. Minden blokkhoz, esetleg bájthoz két fontos vezérlőbit tartozik.

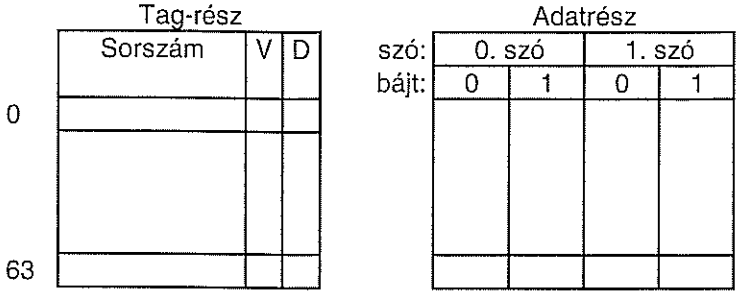
- **Érvényességi jelzőbit.** Azt mutatja meg, hogy az adott blokkban (esetleg bájtban) lévő adat érvényes-e (mert pl. ugyanazon a címen a memóriában már lehet, hogy átírták az információt).
- **Módosítási jelzőbit**. Arról ad információt, hogy történt-e módosítás a blokkban (esetleg bájtban).

A jelzőbitek használatáról a későbbiekben részletesebben szó lesz.

Ezek után vizsgáljuk meg egy egyszerű cache-tár felépítését (**5.5.** ábra).

Az egyszerű érthetőség és áttekinthetőség végett egy kis kapacitású és rövid blokkhosszúságú cache-tárat vizsgálunk meg. Az ábrán a tag-rész tartalmazza a címrészeket, ami alapján az adatok keresése történik. A tag-részhez tartoznak még a különböző vezérlőbitek. Ezek áramkörileg egy egységet alkotnak.

A másik áramköri egységben az adatok találhatók. Az ábra szerint egy adatblokk 4 bájt méretű és egy szó két bájtos. A felvázolt tár most 64 blokkot tartalmaz.



5.5. ábra. Egy cache-tár vázlatos felépítése

A cache-tárak elérhetőség és kezelhetőség szempontjából a következő négy nagy csoportba sorolhatók:

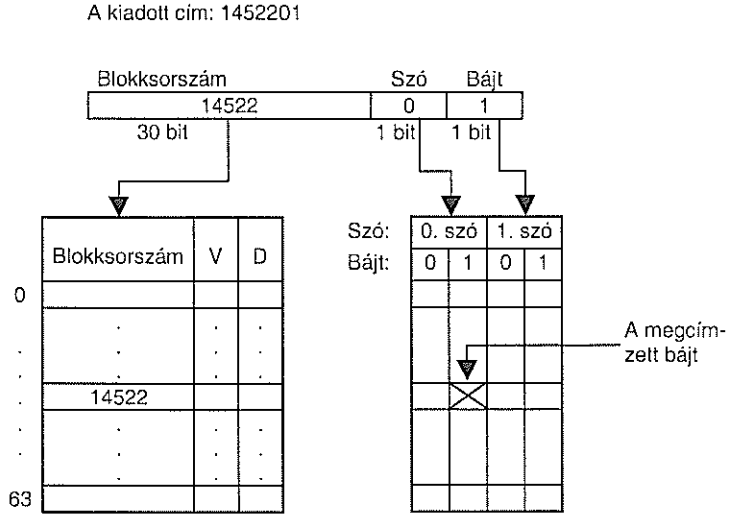
- teljesen asszociatív cache,
- közvetlen leképezésű cache,
- csoport-asszociatív cache,
- szektor-leképezésű cache.

Ezek a cache-tárak az adatok elérési módja alapján más és más felépítésűek.

Teljesen asszociatív (fully associative) cache

A teljesen asszociatív cache-ben a beolvasott blokk bárhová kerülhet. Adatkeresés-kor a kiadott címet a tag-rész minden sorával összehasonlítja. Ez a módszer nagyon rugalmas adatelhelyezést biztosít és nagyon gyors keresést tesz lehetővé. Hátránya, hogy bonyolult összehasonlító áramkör szükséges hozzá, ezért az ilyen felépítésű cache-tárak viszonylag kis kapacitásúak.

Vázlatos felépítésük az **5.6.** ábrán látható.



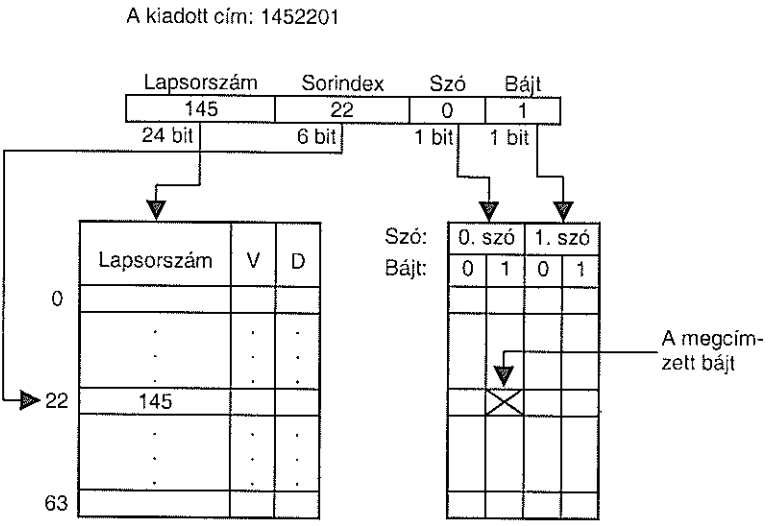
5.6. ábra. A teljesen asszociatív cache-tár felépítése

A cím jelenleg 32 bites. A kiadott cím alapján (az ábrán ez 1452201) a szükséges adatot úgy keressük meg, hogy a cím egy részét (14522) összehasonlítjuk az összes bent lévő blokk sorszámával, és ha az bent van (cache-hit), akkor a cím további része alapján (01) a blokkban megkeressük a megcímzett bájtot. Ha nincs bent az adott sorszámú blokk (cache-miss), akkor a főtárban keresünk tovább, ami jóval időigényesebb művelet.

Közvetlen leképezésű (direct mapping) cache

A közvetlen leképezésű cache-ben a beolvasott blokk csak meghatározott helyre kerülhet. A blokk helyét a cím egy része mutatja meg **(sorindex).** Így minden blokk csak abba a sorba kerülhet, amelyet a sorindexe meghatároz. Ez túlságosan merev tárolási mód, de egyszerű összehasonlító áramkör szükséges hozzá.

Felépítése az **5.7.** ábrán látható.



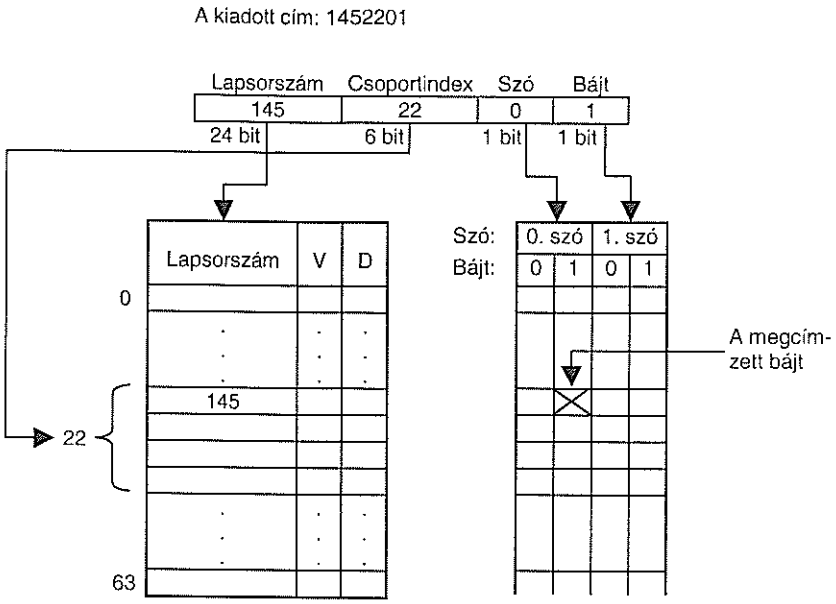
5.7. ábra. A közvetlen leképezésű cache-tár felépítése

A rendszer a kiadott cím sorindexe alapján egy címdekódolóval megkeresi a szükséges sort (viszonylag hosszú művelet), majd összehasonlító áramkörrel megvizsgálja, hogy ott a címnek megfelelő lapsorszámú blokk található-e. Ha igen (cache-hit), akkor a cím utolsó két bitje (01) alapján megkeresi a kért bájtot. Ha nem (cache-miss), akkor a főtárban keresi tovább az adatot.

Csoport-asszociatív (set associative) cache

Ez a fajta felépítés átmenetet képez a teljesen asszociatív és a közvetlen leképezésű cache között. A tár *n* soros csoportokra van osztva és ezek a csoportok úgy működ-

nek, mint a teljesen asszociatív cache-tárak, vagyis az n hely közül bármelyikre kerülhet az adott blokk.
A csoport-asszociatív cache felépítése az **5.8.** ábrán látható.



5.8. ábra. A csoport-asszociatív cache felépítése

A kiadott cím csoportindex része alapján egy címdekódoló megkeresi a szükséges csoportot (viszonylag hosszú művelet), majd az összehasonlító áramkörök megállapítják, hogy az adott lapsorszámú blokk bent van-e a cache-ben. Ha igen (cache-hit), akkor a cím utolsó két bitje (01) alapján a rendszer megkeresi a kért bájtot. Ha nem (cache-miss), akkor a főtárban folytatjuk az adat keresését. Ez a leggyakrabban alkalmazott cache-fajta.

Szektor-leképezésű (sector mapping) cache

A legritkábban alkalmazott cache-fajta. A csoport asszociatívhoz hasonló, de annak éppen a fordítottja, vagyis a csoport helyét a lapsorszám alapján határozzák meg az összehasonlító áramkörök, és a csoporton belül a blokk helye már adott.

5.3.2. A cache-tárak tartalmának karbantartása

A cache-tárban mindig a főtár egy részlete található, és akkor látja el jól a feladatát, ha többnyire azok az adatok vannak benne, amelyekre a μP-nak szüksége van. (Sokszor van cache-hit és ritkán cache-miss). Ehhez több dolognak is teljesülnie kell:

- a tartalom betöltése,
- a megfelelő helyettesítési eljárás alkalmazása,
- az adategyezőség biztosítása (aktualizálás).

Ezek együttes alkalmazását nevezzük a cache-tárak karbantartásának.

A tartalom betöltése

Azt, hogy mit töltünk be a cache-be, az éppen futó program határozza meg, de ezt figyelembe véve is több módszer használatos. Ezek közül a legegyszerűbb az **aktuális igény szerinti betöltési módszer,** amelynek lényege a következő.

A μP kér egy adatot, és ha ez nincs bent a cache-ben, akkor kikeresi a főtárból, és onnan kerül be a μP-ba, de ezzel párhuzamosan a bájtot tartalmazó blokkot áttölti a cache-be is. Így a legközelebb kért adat már valószínűleg a cache-ben lesz.

Ennél több szervezést és előkészítést igényel az a módszer, amely előre „gondolkozik", és azokat a blokkokat tölti be a cache-be, amelyekre nagy valószínűséggel rövidesen szükség lesz (pl. az éppen használt blokkot követő blokkok betöltése).

Helyettesítési eljárások

Ez az eljárás határozza meg azt, hogy az éppen betöltendő blokk melyik helyére kerüljön be (feltételezve, hogy a cache mindig tele van).

Sokféle helyettesítési stratégia terjedt el, ezek közül a fontosabbak:

- a legrégebben bent tartózkodó helyére,
- a mostanában legkevésbé használt helyére,
- véletlenszerűen cserélve.

Mindegyik eljárás alkalmazásához szükséges valamilyen adminisztráció.

Leggyakrabban az ún. LRU (mostanában legkevésbé használt) stratégiát alkalmazzák, vagyis azt a blokkot viszik ki, amelyikre mostanában nem történt hivatkozás

Az adategyezőség biztosítása (aktualizálás)

Cache-tár alkalmazása esetén az egyik legfontosabb feladat az, hogy a cache és a főtár tartalmának (azonos címeken) meg kell egyezniük. Ilyen probléma azoknál a cache-táraknál jelentkezik, amelyekben adatokat is tárolunk, mert ezek az adatok

átírhatók a cache-ben is, ill. a főtárban is. A feladat az, hogy ha egy adatot az egyik helyen átírnak, akkor rövid időn belül a másik helyen is módosítsák.

Probléma kétféle módon keletkezhet.

- Amikor a μP a cache-ben lévő adattal dolgozott, majd azt módosította, és az eredményt át kell írni a főtárba is. (Ez a gyakoribb eset.)
- Amikor a főtárban történt egy olyan adat átírása (pl. külső adatbevitellel), amely a cache-ben is bent van és át kell írni a cache-t is.

A főtárban lévő adat aktualizálására több módszer is használatos, ezek a következők.

Azonnali átírás. A módosított adat azonnal beírásra kerül a főtárba, függetlenül attól, hogy az adott blokk a cache-ben van-e. Ez a módszer lelassítja a program futását, de biztosítja az adategyezőséget.

Visszaírási eljárás. E módszer alkalmazásakor sokféle lehetőség adódhat, amelyek a következők.

- Ha az átírandó adat a cache-ben van, akkora μP ott átírja, a főtárban pedig csak a blokk cseréjekor történik meg a módosítás.
- Ha az átírandó adat nincs a cache-ben, akkor kétféle módon járhat el:
  - betölti a keresett blokkot a cache-be és csak ott módosít, a főtárban pedig csak a blokk cseréjekor,
  - a főtárban módosít és a cache-be nem tölti be a blokkot.

Ennek a módszernek a működése gyorsabb, de nem biztosít minden esetben adategyezőséget.

Egyszeri átírásos módszer. Az adat első módosításakor mindkét helyen megtörténik az átírás, de ha ugyanaz az adat ismét módosul, akkor a főtárban már nem történik meg az aktualizálás, majd csak a blokk cseréjénél. A működése gyorsabb, mint az azonnali átírásosé és biztonságosabb, mint a visszaírásos eljárásénál.

A cache-ben lévő adat aktualizálására a következő módszereket használják.

- Egy logikai áramkör figyeli, hogy a cache-ben bent van-e a főtárban módosított adat. Ha igen, akkor ott is módosítja (aktualizálja), vagy csak érvényteleníti az ún. V (valid) bitet (ezzel jelzi, hogy a cache-ben hibás adat van).
- Ha a főtárhoz adatkiolvasási kérelem érkezik valamelyik perifériától, akkor megvizsgálja, hogy a cache-ben is bent van-e a kívánt adat, ha igen, akkor arról a helyről olvassa ki, ahol érvényes adat van (ehhez szükséges a vezérlőbit vizsgálata).

5.4. A virtuális tárkezelés

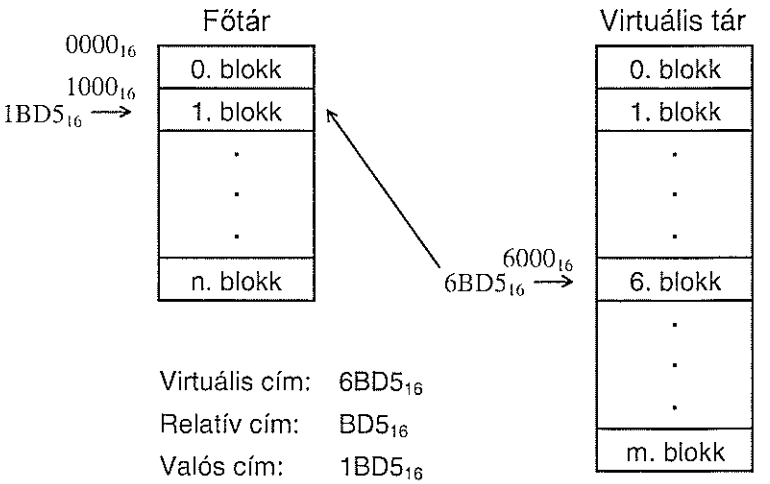
A számítógépekben kialakított főtár (operatív tár) viszonylag kis méretű (ma többnyire 10–256 Mbájt közötti). A ma használatos μP-ok címvezetékeinek a száma általában 32. Ennyi bitet felhasználva elvileg 2^{32} = 4 Gbájtos memóriatartományt lehetne elérni.

Ezt a teljes tartományt nevezzük látszólagos vagy virtuális memóriának. Azért látszólagos, mert ekkora méretű főtár nincs kiépítve és nem is célszerű kiépíteni, viszont a háttértáron (merevlemezen) ez megtalálható. Leegyszerűsítve azt mondhatjuk, hogy a háttértár a virtuális memória, és ez a főtár bővítéseként is felfogható.

5.4.1. A virtuális tár kezelésének elve

A programok csak akkor futnak, ha az operatív tárban vannak, ezért a futtatás előtt azokat oda be kell tölteni. A mai gyakorlatban egyre általánosabbá válik az, hogy a programok (szoftverek) mérete jóval nagyobb, mint a főtár kapacitása, ezért a program futása alatt annak mindig csak egy részlete (az éppen használatos része) található meg az operatív tárban. A szükséges részek háttértárról való betöltését valamilyen módon meg kell oldani. A főtár és a háttértár (virtuális tár) egységes kezelését az ún. virtuálistár-kezelés teszi lehetővé.

Ennek elve az **5.9.** ábrán látható.



5.9. ábra. A virtuális tár kezelésének elve

A rendszer működésének lényege, hogy mindkét tár blokkokra (keretekre) van osztva. Az egyszerűbb kezelhetőség érdekében legyenek ezek egyforma méretűek, pl. 4 Kbájtosak! Így az egyes blokkok kezdőcímei: 0000_{16}, 1000_{16}, 2000_{16}, stb. A főtárban legyen n blokk (keret), a virtuális tárban pedig m blokk, ahol $m > n$!

A felhasználói program virtuális címeket (nevezik ezt logikai címeknek is) használ, és a tárkezelő egység (MMU) feladata, hogy ezeket valós (főtárbeli) címekké alakítsa át.

A példánkban a felhasználói program a $6BD5_{16}$ címre (virtuális cím) hivatkozik. Az 1. számjegy (most a 6) jelenti azt, hogy a kívánt információ (adat vagy utasítás) a 6. blokkban található.

Az MMU megvizsgálja, hogy a blokk a főtárban van-e. Ehhez különböző táblázatokat használ. Ha a főtárban van, akkor a táblázatot felhasználva meghatározza, hogy melyik keretben van az információ, és ennek ismeretében kiszámolja a pontos főtárbeli címet.

Ha a kért adatot tartalmazó blokk nincs a főtárban, akkor az MMU-nak el kell döntenie, hogy melyik bent lévő blokk helyére töltse be a szükségeset. A döntés az ún. **helyettesítési algoritmussal** hozható meg.

Tegyük fel, hogy az 1. számú keretre esett a választás, akkor ennek a blokknak a helyére hozza be a keresett adatot tartalmazó blokkot. A blokk beolvasása a főtárba hosszadalmas művelet és lelassítja a program futását.

A rendszer a virtuális cím ($6BD5_{16}$) felhasználásával meghatározza a relatív címet (a blokk kezdőcíméhez viszonyított cím):

- **relatív cím** = virtuális cím – a blokk címe = $6BD5_{16} - 6000_{16} = BD5_{16}$. Ha ez megvan, akkor kiszámolja a főtárbeli ún. fizikai címet:
- **fizikai cím** = a blokk kezdő címe + relatív cím = $1000_{16} + BD5_{16} = 1BD5_{16}$, majd erről a címről behozza a kért adatot.

5.4.2. A virtuális tárkezelés megvalósítása

Lapozás

A lapok (page-ek) olyan blokkok, amelyeknek a mérete állandó és helyük rögzített. A lapok mérete általában 4 Kbájt. Az **5.9.** ábrán egy blokk egy lapnak felel meg. Ezek a lapok az ún. lapkeretek helyére kerülnek be a főtárba és a háttértárba is. Minden lapkeretnek állandó (kötött) a kezdőcíme. Ezek alapján a címek meghatározásának módja:

- **virtuális cím** = a keret háttértárbeli kezdőcíme (logikai sorszáma) + relatív cím,
- **fizikai cím** = a lapkeret főtárbeli kezdőcíme + relatív cím.

Az MMU-nak a címek meghatározásához szüksége van különböző adatokra és ezeket felhasználva egy- vagy többlépcsős indirekt címzést valósít meg.

Ezeket az adatokat tartalmazó táblázatokat TLB-nek (Translation Lookside Buffer) nevezzük. Felépítésük alapján ezek teljesen asszociatív vagy csoport-asszociatív cache-tárak, és a leggyakrabban használt lapok fő adatait (deszkriptorait) tartalmazzák. Ide tartoznak a lapok kezdőcímei, jelzőbitjei stb. Amikor egy új hivatkozás (címkiadás) során lapcserére van szükség, akkor valamilyen stratégia alapján el kell dönteni, hogy melyik lapot visszük ki a háttértárra.

A döntés elvileg nagyon egyszerű: azt amelyikre nagy valószínűség szerint már nem lesz szükség, ill. csak hosszú idő múlva kell. Ezt sajnos előre nem tudhatjuk, ezért a lapváltásokra ún. helyettesítési algoritmusokat dolgoztak ki. Ezek közül a leggyakrabban használatosak a következők.

- **A legrégebben bent lévő lap cseréje (FIFO = First In Firs Out).** Azt a lapot viszi ki, amelyik a legrégebb óta bent van a főtárban. Viszonylag egyszerűen adminisztrálható módszer, de nem túl szerencsés, mert lehet, hogy azért van bent régóta, mert az ebben lévő adatokat használja a rendszer a legtöbbször. Ekkor viszont rövidesen vissza kell tölteni.
- **A legkevésbé használt lap cseréje (LRU = Least Recently Used).** Azt a lapot viszi ki, amelyre egy meghatározott időtartam alatt a legkevesebbszer történt hivatkozás. Kicsit bonyolultabb az adminisztrációja, mint a FIFO-é, de hosszú távon kevesebb lapcsere szükséges.
- **A legrégebben nem használt lap cseréje.** A legrégebben bent lévő lapok közül azt viszi ki, amelyet a legrégebben nem használtak. Viszonylag egyszerű és eléggé hatékony módszer.

A felsorolt módszereken kívül egyéb eljárásokat is alkalmaznak, de azok jóval bonyolultabbak, ill. kevésbé hatékonyak. Pl. ilyen lehet a véletlenszerű választás, ami semmilyen szempontot nem vesz figyelembe a döntésnél.

Szegmentálás

A szegmensek (segment) olyan blokkok, amelyeknek a mérete nem állandó, de a rendszertől függően létezik egy max. méret. A szegmensek kezdőcíme nem rögzített, ahogy az egyik szegmens befejeződik, a következő címen kezdődhet egy másik szegmens. Sőt a szegmensek átlapolódhatnak, ez különösen a programok írásánál jelenthet előnyt.

Előnye hogy jól kitölthető a rendelkezésre álló tárterület, mert minden blokk teljesen fel van töltve, szemben a lapozásos módszerrel, ahol ha egy program jóval kisebb, mint a lapméret, akkor is elfoglal egy lapkeretnyi területet. Előnye az is, hogy így könnyen védhető egy összetartozó terület, legyen az bármekkora méretű is.

Hátránya, hogy egy kivitt szegmens helyére nagy valószínűséggel egy kisebb méretű töltődik be (mert nagyobb méretű számára nem elég a hely, egyező méretű valószínűsége pedig kicsi), ezzel egy idő után elaprózódott üres helyek jönnek létre. Ennek megszüntetésére meghatározott időközönként a tárterületet át kell rendezni. Hátrányt jelent az is, hogy a lapozással szemben **a nyilvántartás itt több adminisztrációt igényel** (tárolni kell a szegmens méretét és kezdőcímét).
A címek meghatározása hasonló, mint a lapozásos eljárásnál:

- **virtuális cím** = a szegmens logikai sorszáma + relatív cím,
- **fizikai cím** = a szegmens főtárbeli kezdőcíme + relatív cím.

Az MMU-nak a cím kiszámításához itt is szüksége van adatokra, amelyeket az ún. szegmensleíró táblákban rögzítenek. Ezek felépítése és működése a TLB-éhez hasonló.

A szegmensek betöltésével kapcsolatban a következő módszereket alkalmazzák.

- **Betöltés az első szabad helyre.** A főtárat az elejétől vizsgálja és az első olyan helyre rakja a szegmenst, ahol az elfér. Kis méretű szegmensek esetén viszonylag jó kitöltést tesz lehetővé.
- **Betöltés a legjobb helyre.** Megkeresi azt a helyet, ahová betöltve a legkevesebb üres hely marad, és a szegmenst oda helyezi el.
- **Betöltés a következő szabad helyre.** A főtárat onnan kezdi vizsgálni, ahová az előző szegmenst töltötte. Ahol elegendő méretű üres helyet talál, oda helyezi a szegmenst.
- **Betöltés a legrosszabb helyre.** Megkeresi azt a helyet, ahová betöltve a legtöbb üres hely marad szabadon, és oda viszi át a szegmenst. Így van lehetőség arra, hogy a kialakult üres helyre még lehet újabb szegmenst betölteni.

5.5. A címzés általánosítása, a tárkezelés gyorsítása

A számítógépek működtetésének biztonságosabbá, folyamatosabbá és gyorsabbá tételéhez szükség van a címzési módszerek kiterjesztésére (pl. perifériák címzése), új címzési megoldások kialakítására (pl. memóriatömbök kezelése), a program- és adattárolás szétválasztására stb.

A főtárhozfordulás gyorsítása

A μP műveletvégző sebessége jóval nagyobb, mint a főtárból való adatbetöltés sebessége, ezért a gyorsabb betöltést különböző címzési megoldásokkal segítik elő. Ezek közül a két leggyakoribb a következő.

Memóriatömbök használata. Ennél a módszernél a memóriát (főtárat) címzés szempontjából ún. tömbökre bontják. Pl. 32 bites szavak esetében 4 blokkra. Így egyetlen cím kiadásával (annak legalsó két bitje '0' értékű) 4 bájtnyi adatot tudunk kiolvasni.

Átlapolt címzés. Ezt a módszert a DRAM-okból felépülő főtárakban alkalmazzák az adatátvitel gyorsítása érdekében. A DRAM-ok ciklusideje közel kétszerese az elérési időnek, ami azért van, mert a működtetésükhöz hosszú feléledési idő szükséges. Ezt a feléledési időtartamot használják fel arra, hogy a μP egy újabb címzési folyamatot indítson el. Így egyszerre két cím is ki van adva. Mivel két cím nem vonatkozhat ugyanarra a tárolóblokkra, ezért a tárat szét kell osztani legalább két tömbre. Ezt a szétosztást a gyakorlatban a memóriamodulon belül is megoldhatják, mint pl. az EDO (Extended Data Output) DRAM-oknál.

A program- és adattárolás szétválasztása

Mivel egy program futása során az abban lévő utasításokat (általában) nem írják át, az adatokat viszont elég sokszor, ezért a gyorsabb adatfeldolgozás és a biztonság érdekében ajánlatos egy adott programhoz tartozó utasításokat és adatokat különkülön helyen (szegmensben) tárolni. Pl. a kiadott cím egy bitjének változtatásával kijelölhető, hogy az egy adatszegmensre vagy egy utasításszegmensre vonatkozike. Megoldható ugyanez a szegmensleíró (deszkriptor) táblában egy vezérlő- (védelmi) bit átállításával is.

Eszközök címzése

A μP-nak egy perifériához úgy tud hozzáférni, hogy kiválasztja (megcímzi) az eszközt és kijelöli az adatátvitel irányát. Így volt ez a memóriamodulok esetében is (ME, R/W). Az eszköz kiválasztása a tokengedélyező jel aktivizálásával lehetséges (CS), az irány kijelölése pedig az $R/\overline{W}$ vezérlőjellel történik. A programozó ezeket ugyanúgy kezelheti, mint a főtárat, de itt a címterület jóval kisebb, ezért nem szükséges minden címvezeték bekötése.

A különböző eszközök elérésére (címzésére) a következő két módszert használják.

- **A főtárral azonos elérési mód:** A perifériák ugyanúgy címezhetők, mint a memória, így bármelyik memóriára hivatkozó utasítás elérheti a memóriát és a perifériákat is. A legtöbb μP-nál ezt a módszert használják.

- **Az elkülönült elérési mód:** A memória, ill. a perifériák elérésére használt utasítások műveleti kódja határozza meg az átvitel helyét, pl. az M/$\overline{IO}$ vezérlőjel aktivizálásával. Ez a címtartomány megduplázását jelenti, mert ugyanaz a cím kiadható a főtárnak és a perifériának is.

A tárkezelés gyorsítása

A szegmensek, ill. a lapok betöltésénél az adatokat a háttértárról a főtárba kell átvinni, majd azok egy részét a μP-ba. A tárhierarchia felépítésénél láttuk, hogy a háttértárak elérési ideje kb. ezerszerese a főtár elérési idejének, vagyis nagymértékben lelassul az éppen futó program végrehajtása. Ezért törekedni kell a lap- és szegmenscserék számának csökkentésére és a cserék lebonyolításának gyorsítására.

A cserék számának csökkentése jól strukturált programozási technikával (objektum-orientált programozás) érhető el. Ennek lényege, hogy jól meghatározott programegységek jöjjenek létre és az ugrások (vezérlésátadások) száma csökkenjen.

A lebonyolítás gyorsítása úgy lehetséges, ha a lap- és szegmenstáblázatok nagyon gyors működésű cache-tárakban vannak elhelyezve.

5.6. A tárvédelem lehetőségei

Napjainkban egyre fontosabbá válik a tárolt adatok és programok védelme. A korszerű számítógépek és operációs rendszerek lehetővé teszik, hogy egyidejűleg több felhasználó is dolgozzon ugyanazon a gépen (multiuser-üzemmód) és több program fusson párhuzamosan (multitask-üzemmód). Ezzel növekszik annak a valószínűsége, hogy az egyes programok (folyamatok) beleírnak a másikba, elrontva annak működését. Az ilyen problémák megakadályozására feltétlenül szükséges a különböző védelmi módszerek alkalmazása.

A védelem legfontosabb feladatai a következők.

- Az elkülönített memóriaterületek védelme. Egy program csak a saját területét használhatja, vagyis a kiadott címeket meg kell vizsgálni, hogy a megadott területre vonatkoznak-e. Ezen belül is külön kell választani a programterületet (ahol az utasítások vannak) és az adatterületet, mert a programterületről csak olvasás történhet, míg a másik írható-olvasható.
- A rendszerprogramok védelme a felhasználói programoktól.
- A felhasználói programok védelme egymástól.

A védelmi rendszer kialakításában nagy fontosságúak a szegmensek és a különböző deszkriptortáblák.

A tárvédelem kialakításának néhány lehetősége:

- Külön szegmensbe lehet tenni az adatokat és az utasításokat tartalmazó programrészeket, ezzel szabályozva az írhatóságot és olvashatóságot.
- Védelmi bitek rendelhetők az egyes szegmensekhez és ezeket a deszkriptor-táblákban állíthatjuk be.
- Prioritások is rendelhetők az egyes programokhoz (pl. a rendszert működtető programok prioritása a legnagyobb).
- Létrehozhatók olyan szegmensek, amelyeket minden program használhat. Ezek a GDT-táblában (Globális Deszkriptor Táblában) vannak nyilvántartva. Vannak olyan szegmensek, amelyeket csak egy-egy felhasználói program használhat. Ezek az LDT-táblában (Lokális Deszkriptor Táblában) vannak nyilvántartva.

Ellenőrző kérdések, feladatok

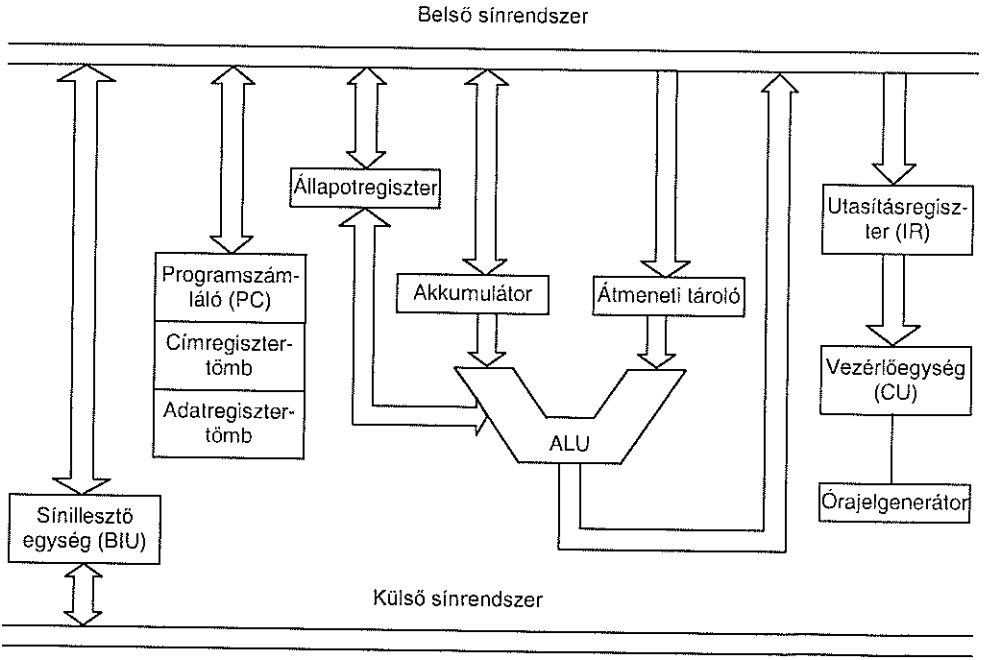
1. Ismertessük a tárkezeléssel kapcsolatos alapfogalmakat!
2. Mutassuk be a regisztertárat és alkalmazási módjait!
3. Mutassuk be a cache-tárak típusait!
4. Ismertessük a cache-tárak karbantartásának módjait!
5. Ismertessük a virtuális tárkezelés elvét!
6. Mutassuk be a lapozást és a szegmentálást!
7. Ismertessük a tárvédelmi lehetőségeket!

6. MIKROPROCESSZOROK

A mikroprocesszor olyan egy vagy több lapkán kialakított VLSI áramkör, amely a digitális számítógép központi egységének (Central Process Unit) alapvető funkcióit hajtja végre. A mikroprocesszor végzi az aritmetikai és logikai műveleteket, valamint a rendszer vezérlését.

6.1. A mikroprocesszor általános felépítése

A processzor meghatározott feladatokat ellátó egységekből, ún. **funkcionális egységekből** épül fel. A processzor funkcionális felépítése a **6.1.** ábrán látható. Az egyes processzorok felépítésében jelentős eltérések tapasztalhatók, azonban mindegyikre jellemző, hogy tartalmazza a következő alapvető funkcionális egységeket:



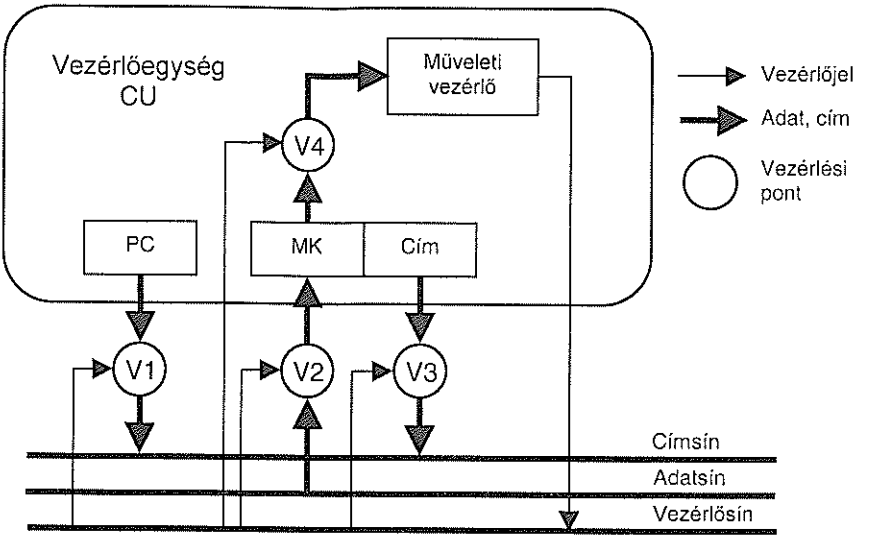
6.1. ábra. A processzor funkcionális egységei

- vezérlőegység, CU (Control Unit),
- aritmetikai és logikai egység (ALU, Aritmetic Logic Unit),
- regisztekészlet (belső munkatárak),
- sínillesztő egység (BIU, Bus Interface Unit),
- címszámító egység (AU, Address Unit),
- belső sínrendszer.

6.1.1. A vezérlőegység

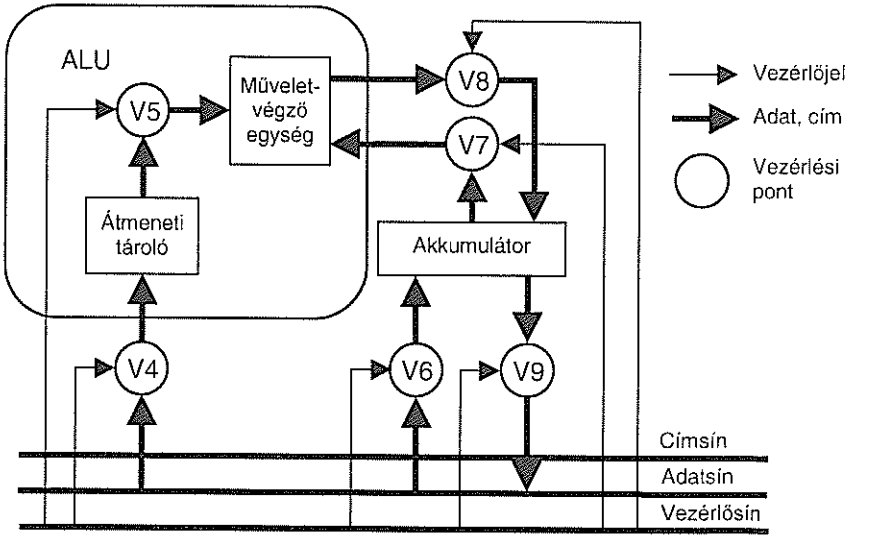
A vezérlőegység az utasítások alapján előállítja a processzoron belüli és a processzorhoz kapcsolt külső egységek működéséhez szükséges vezérlőjeleket. Belső vezérlőjelekkel irányítja az aritmetikai egység működését és a regiszterek, ill. a belső sínrendszer közötti adatátvitelt. Külső vezérlőjelei a processzor és a memória, valamint a processzor és a perifériák közötti adatátvitelt, a sínvezérlést és a megszakításkezelést irányítják. Összehangolja és ütemezi az egész rendszer működését. Tulajdonképpen egy vezérelt szinkron sorrendi hálózat, az órajelgenerátor által szolgáltatott többfázisú órajellel ütemezve.

A vezérlőegység működésének megértéséhez meg kell vizsgálnunk, hogy az utasítások végrehajtását milyen elemi lépésekben hajtja végre a processzor. A 4. fejezetben már megismerkedtünk a gépi ciklus fogalmával, ami a processzor által egy vezérlőjel-sorozattal végrehajtható elemi tevékenység. Ha ennél még részletesebben vizsgáljuk a processzor működését, akkor a vezérlési feladat szerinti legkisebb elemi művelet egy ütemezett vezérlőjel kiadása, vagy megszűntetése a vezérlősín valamely vezetékén. Ezek az elemi lépések adatútvonalak lezárását vagy megnyitását, ill. állapotjelzők beállítását eredményezik. Ezt a folyamatot nevezzük **műveleti vezérlésnek**. A műveleti vezérlést a vezérlőegység vezérlési pontokon keresztül valósítja meg, amelyek vezérelt kapcsolóknak tekinthetők. Az ütemezett cím- és adatforgalom ilyen, a processzoron belüli és kívüli vezérlési pontokon keresztül valósul meg. Az egyes külső és belső vezérlési pontok a **6.2. – 6.5.** ábrákon láthatók.

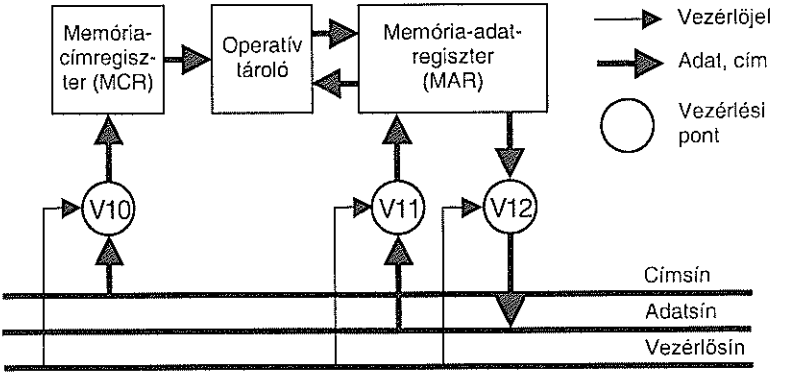


PC: utasításszámláló, MK: műveleti kód

6.2. ábra. A vezérlőegység vezérlési pontjai

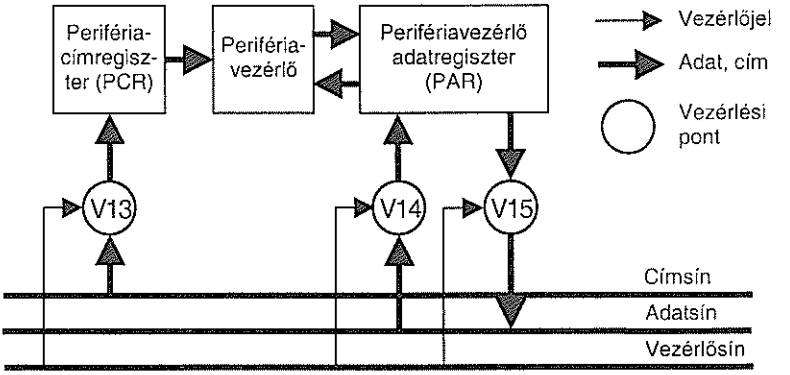


6.3. ábra. Az ALU vezérlési pontjai



MCR: memória-címregiszter, MAR: memória-adatregiszter

6.4. ábra. Az operatív tár vezérlési pontjai



PCR: perifériavezérlő címregiszter, PAR: perifériavezérlő adatregiszter

6.5. ábra. A perifériák vezérlési pontjai

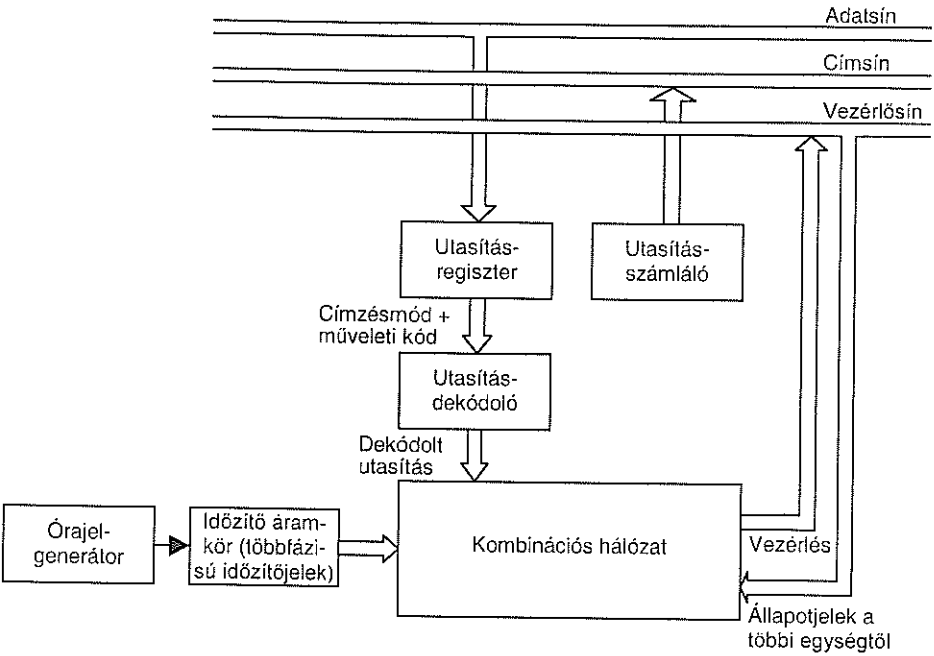
A műveleti vezérlés megértéséhez a **6.2.–6.5.** ábrák alapján tekintsük át az utasításlehívás műveleti vezérlését (**6.6.** ábra)!

Elemi lépés	Érintett vezérlési pontok
Utasításszámláló tartalmának (az utasítás címének) átvitele a memória címregiszterébe	V1, V10
A memória adatregiszterében lévő tartalom (a kiolvasott utasítás) átvitele a processzor utasításregiszterébe	V12, V2
A műveleti kód alapján további vezérlőjelek kiadása	V4

6.6. ábra. Utasításlehívás műveleti vezérlése

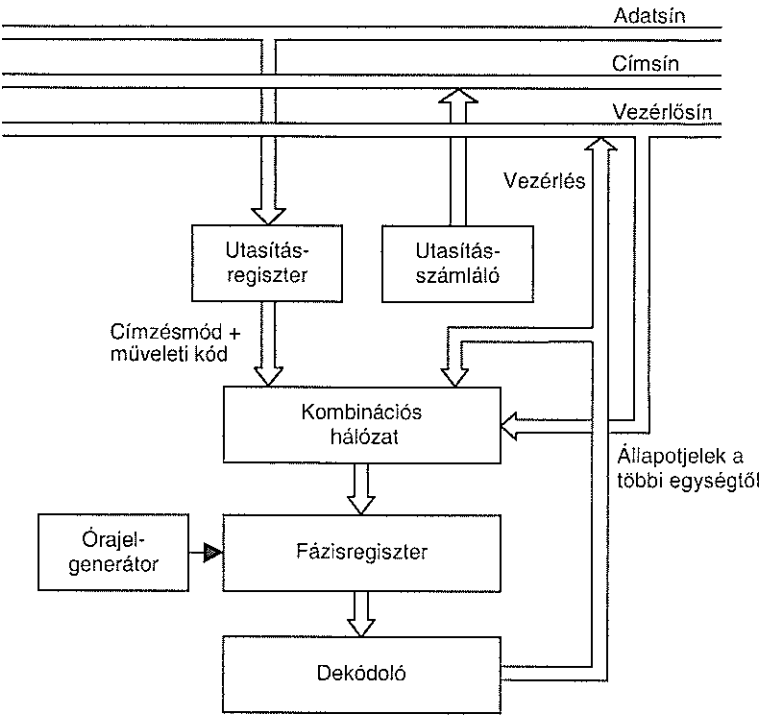
A számítógépekben a műveleti vezérlés huzalozott és mikroprogramozott módon valósítható meg.

- **Huzalozott műveleti vezérlés.** A processzorba beépített hardvereszközök irányítják a műveletvégzés elemi lépéseit. A műveleti kód bitjei vezérlik a műveletet vezérlő áramkört, amely szinkron sorrendi hálózatként órajellel ütemezett vezérlőjeleket állít elő a vezérlési pontok számára. Ez hardveres megoldás, amely gyors műveleti vezérlést eredményez, viszont bonyolult vezérlőáramkört igényel, ezért drága, és a hardveres kialakítás miatt merev (utólag nem módosítható). Huzalozott vezérlés logikai felépítése látható a **6.7.** ábrán. Ilyen műveleti vezérlővel rendelkeznek a RISC processzorok.



6.7. ábra. Huzalozott vezérlőegység felépítése

A huzalozott vezérlőegység egy speciális változata a fázisregiszteres vezérlőegység, amely egyfázisú órajellel is működtethető. A fázisregiszter tulajdonképpen egy tárolósor, amely a vezérlőjelek állapotát tartalmazza kódolt formában. A tárolók kimenetei dekódolót címeznek, amely a kódolatlan vezérlőjeleket álltja elő. Fázisregiszteres vezérlőegység felépítése látható a **6.8.** ábrán.



6.8. ábra. Fázisregiszteres vezérlőegység felépítése

- **Mikroprogramozott műveleti vezérlés.** A műveletvégrehajtás elemi lépéseit a processzorban elhelyezett mikroprogram írja le, amely a mikroprogramtárban (belső ROM) helyezkedik el. A műveleti kód bitjei (makroutasítás) a mikroprogramtárban egy címet határoznak meg, amely címen az adott utasítás műveleti vezérlését leíró mikroprogram található. A mikroprogram mikroutasításokból áll.

Mikroutasítás felépítése

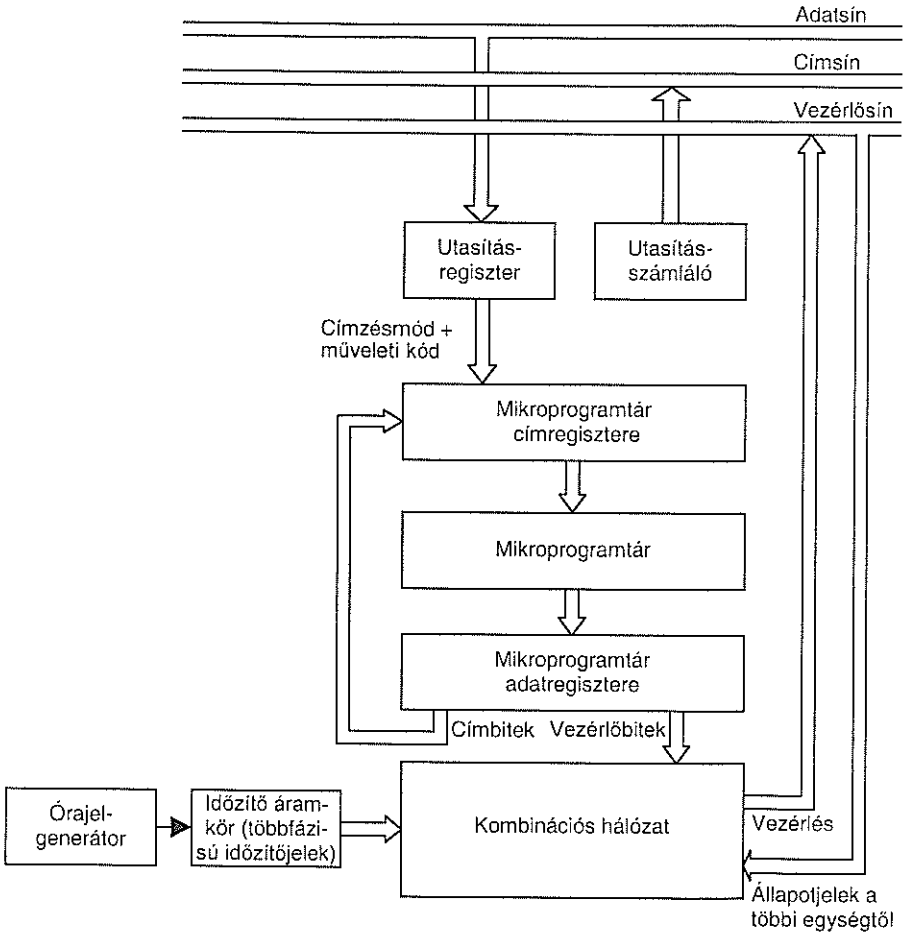
A mikroutasítás három részből áll:

- **mikro-műveletikód:** a vezérlési pontok pillanatnyi állapotát írja le;
- **mikro-címzésmód:** azt írja le, hogy a mikroprogram végrehajtása hogyan folytatódik: sorrendileg (INC), feltétel nélküli vezérlésátadással (JMP) vagy egy külső feltételtől függő vezérlésátadással (JF, ahol F külső feltétel);
- **mikrocím**: vezérlésátadó mikroutasításoknál a következő mikroutasítás címét tartalmazza, sorrendi végrehajtásnál a címrész tartalma lényegtelen. Mikroutasítás felépítése a **6.9.** ábrán látható.

Mikro-műveleti kód	Mikro-címzésmód	Mikrocím

6.9. ábra. A mikroutasítás felépítése

A mikroprogram utasításait sorban végrehajtva történik meg a műveleti vezérlés. Ez szoftveres megoldás, amely lassúbb műveleti vezérlést eredményez, viszont egyszerűbb vezérlőáramkört igényel, ezért olcsóbb, valamint a processzor utasításkészlete nagy számú utasítást tartalmazhat. Mikroprogramozott vezérlő logikai felépítése látható a **6.10.** ábrán. Ilyen műveleti vezérlővel rendelkeznek a CISC processzorok.

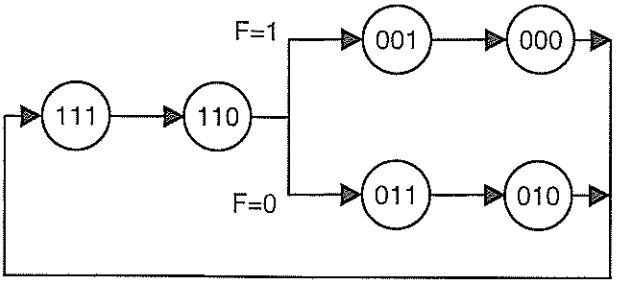


6.10. ábra. Mikroprogramozott vezérlőegység felépítése

6.1.2. Vezérlőegység tervezése

Tervezzünk fázisregiszteres vezérlőegységet, ami a **6.11.** ábra szerinti állapotdiagramnak megfelelően működik (ahol F a vezérlőjel)!

A megvalósításhoz használjunk JK-tárolókat, dekódolókatt és tetszőleges kapuáramköröket!



6.11. ábra. Vezérlőegység állapotdiagramja

Első lépésben meg kell terveznünk a fázisregiszter (JK-tárolók) vezérléséhez szükséges kombinációs hálózatot. A kombinációs hálózat tervezéséhez szükségünk van az állapotdiagram alapján kitöltött állapotátmeneti táblára, amely a feltételes és feltétel nélküli állapotátmeneteket tartalmazza.

Az állapotátmeneti táblába az előírt átmenetekhez szükséges vezérlőjeleket írjuk be, a tároló vezérlési táblája alapján. A JK-tároló vezérlési táblája a **6.12.** ábrán látható.

N. állapot	N+1. állapot	J	K
0	0	0	h
0	1	1	h
1	0	h	1
1	1	h	0

6.12. ábra. JK-tároló vezérlési táblája

A kitöltött állapotátmeneti tábla a **6.13.** ábrán látható.

A feltétel nélküli átmenetek esetén az állapotátmeneti tábla kitöltése egyszerű, csak az adott helyiérték N. és (N+1). állapota alapján a vezérlési tábla J-K oszlopait kell az átmeneti tábla J és K oszlopaiba másolni. A feltételes átmeneteknél a J és K bemenetek vezérlése a vezérlőjel függvénye.

Célszerű egy igazságtáblázatot kitölteni, ami alapján a vezérlőfüggvény könnyen meghatározható. Pl. a 100 állapotból az F feltételtől függően a 001 vagy 011 állapotba kell átmenni. Ehhez a Q_B helyiértékre a következő igazságtáblát tölthetjük ki:

	$Q_{B,N}$	$Q_{B,N+1}$	J_B	K_B
F=0	0	1	1	h
F=1	0	0	0	h

A táblázatból kiolvasható, hogy a feltételes átmenet esetén a helyes átmenethez a K_B vezérlőbemenetre bármilyen jelet adhatunk (h: határozatlan állapot), a J_B vezérlőbemenetre pedig F negáltját kell adnunk. Ezt írjuk be az állapotátmeneti tábla vezérlés oszlopaiba (**6.13.** ábra).

N. állapot				Vezérlés						N+1. állapot								
										Feltételes F=0			Feltételes F=1			Feltétel nélküli		
q	Q_A	Q_B	Q_C	J_A	K_A	J_B	K_B	J_C	K_C	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C
q_7	1	1	1	h	0	h	1	h	1							1	0	0
$\bar{q}_4$	1	0	0	h	1	$\bar{F}$	h	1	h	0	1	1	0	0	1			
q_1	010	0	1	0	h	0	h	h	1							0	0	0
q_0	0	0	0	1	h	1	h	1	h							1	1	1
q_3	0	1	1	0	h	h	0	h	1							0	1	0
q_2	0	1	0	1	h	h	0	1	h							1	1	1

6.13. ábra. A vezérlőegység állapotátmeneti táblázata

A következő lépés az átmeneti táblából a vezérlési függvények kiolvasása. Célszerű a q dekódolt vezérlőjeleket kiolvasni, mert így a vezérlőhálózat egyszerűbben megvalósítható.

Ezt kétféleképpen végezhetjük.

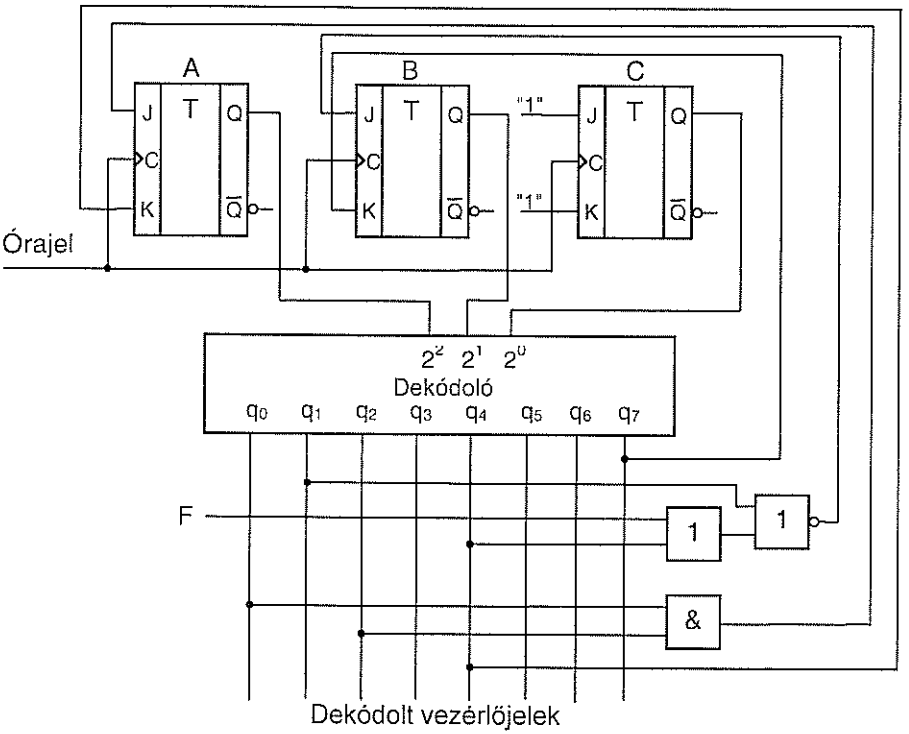
- A táblázat egyeseit és a ponált vezérlőjeleket olvassuk ki a megfelelő dekódoló kimenettel szorozva.
- A táblázat nulláit és a negált vezérlőjeleket olvassuk ki (egyszeres negálás) a megfelelő dekódoló kimenettel szorozva, majd az egész vezérlőfüggvényt negáljuk.

  Célszerűbb azt a kiolvasást választani, amelyik egyszerűbb vezérlőfüggvényt eredményez (nullából vagy egyből van kevesebb).

A táblázatból kiolvasott vezérlőfüggvények:

- $J_A=q_0+q_2$, $K_A=q_4$;
- $J_B = Fq_4+q_1$, $K_B=q_7$;
- $J_C=1$, $K_C=1$.

A vezérlőfüggvények ismeretében a vezérlőegység megvalósítható (**6.14.** ábra).



6.14. ábra. Fázisregiszteres vezérlőegység logikai rajza

Tervezzük meg a **6.11.** ábra szerint működő vezérlőegység mikroprogramozott változatának mikroprogramját is!

Tervezési adatok:

Legyen a mikroprogram utasításainak szerkezete a következő:

- mikro-műveleti kód: 3 bit,
- mikro-címzésmód: 2 bit,
- mikrougrási cím: 3 bit.

Mikro-címzésmódok kódjai:

- INC: 01,
- JF: 11 (F=1 JMP, F=0 INC),
- JMP: 00.

Először mnemonikus formában írjuk meg a mikroprogramot az állapotdiagram alapján (**6. 15.** ábra)!

Mikrocím	Állapot	Mikro-címzésmód	Ugrás
000	q_7	INC	
001	q_4	JF	q_1
010	q_3	INC	
011	q_2	JMP	q_7
100	q_1	INC	
101	q_0	JMP	q_7

6.15. ábra. Mnemonikus mikroprogram

A mnemonikus kódú programot binárisan kódolva kapjuk a mikroprogram tárban elhelyezhető programját (**6.16.** ábra).

Mikrocím	Bináris kód	Hexadecimális kód
000	11101hhh	E8
001	10011100	9C
010	01101hhh	68
011	01000000	40
100	00101hhh	28
101	00000000	00

hhh legyen: 000!

6.16. ábra. A vezérlőegység bináris mikroprogramja

6.2. Aritmetikai és logikai egység (ALU, Aritmetic Logic Unit)

Az ALU feladata matematikai és logikai műveletek végzése. Az aritmetikai műveletek is visszavezethetők logikai műveletekre, ezért az ALU gyakorlatilag egy kombinációs hálózat. A műveletvégzéshez néhány regisztert használ (átmeneti regisztert, akkumulátort), innen veszi a műveletvégzés operandusait. A processzorba integrált műveletvégző egységek fixpontosak vagy lebegőpontosak, BCD vagy bináris kódúak, átvitelgyorsítósak vagy e nélküliek lehetnek. A processzor teljesítményét alapvetően meghatározza a műveletvégző egységek száma és működése. A műveletvégző egységekkel részletesebben a Tankönyvmester Kiadó: Digitális elektronika c. tankönyve foglalkozik.

6.3. Regiszterkészlet

Mivel a regiszterek a processzor belsejében található nagyon gyors tárolók, ezért igen drága tárolást valósítanak meg. Méretük a korszerű processzoroknál 32, 64, 128 bit.

A felhasználói programok szempontjából három csoportra bonthatók:

- rendszerregiszterekre,
- adatregiszterekre,
- címzőregiszterekre.

Rendszerregiszterek

A programozó által nem hozzáférhető, csak a processzor által az utasításvégrehajtás során használt regiszterek.

Utasításregiszter (Instruction Register, IR): Rendszerregiszter, amely a mikroprocesszor által beolvasott, végrehajtás alatt álló utasítást tárolja.

Állapotregiszter (Flags): A jelző- vagy flagbiteket tartalmazza. Ezek a bitek szoros kapcsolatban állnak az ALU-val, értéküket az ALU állítja be az utoljára végzett művelet eredményének függvényében. Értéküktől feltételes vezérlésátadó utasítások végrehajtása függ. Az Intel 80X86 processzorcsaládnál ez a Flags regiszter.

Néhány jelzőbit értékét a programozó is beállíthatja a processzor flagvezérlő utasításaival.

Az állapotregiszter legfontosabb jelzőbitjei:

- **Z**: zéróflag, értéke 1, ha az utoljára végzett művelet eredménye nulla,
- **S**: előjelflag, értéke 1, ha az eredményben a legnagyobb helyiértékű bit 1,
- **P**: paritásflag, értéke 1, ha az eredményben páros számú 1-es van,
- **C**: átvitel bit, értéke 1, ha az eredmény legnagyobb helyiértékű bitje már nem ábrázolható a célregiszterben,
- **AC**: kiegészítő átviteli bit, értéke 1, ha a művelet végrehajtásakor a 3. bitről a 4. bitre átvitel keletkezik,
- **O**: túlcsordulás flag, értéke 1, ha az eredmény nem helyes, mert az előjelbitre túlcsordulás következett be.

Adatregiszterek

Általános célú regiszterek, amelyek a felhasználói programok utasításaiban korlátozás nélkül használhatók. Ez csak az utasítások többségére igaz, hiszen ezeknek a regisztereknek is lehet speciális feladatuk, ami más regiszterrel nem végezhető el, így a vezérlőegység egyszerűbb felépítésű lehet.

Az Intel 80X86 processzorcsaládnál az assembly nyelvű programok négy ilyen regisztert (AX, BX, CX, DX) használhatnak.

Akkumulátor (AX): Általános célú regiszter, az ALU gyakran használja a műveleteihez. Sok utasításnál itt kell elhelyezni a művelet egyik operandusát, majd az eredmény is ebben tárolódik. Az Intel 80X86 processzorcsaládnál ez az AX jelű regiszter.

Címzőregiszterek

A processzor használja a címképzési eljárás során, de tartalmukat a felhasználó is módosíthatja. A címzőregiszterekkel gyakorlatilag adatok és utasítások memóriabeli címeit adhatjuk meg.

Az Intel 80X86 processzorcsaládnál a címregiszterek a következők.

Szegmensregiszterek:

- CS: Code Segment, az utasításterület kezdőcímét tartalmazza,
- DS: Data Segment, az elsődleges adatterület kezdőcímét tartalmazza,
- SS: Stack Segment, a veremtár területének kezdőcímét tartalmazza,
- ES: Extra Segment, a másodlagos adatterület kezdőcímét tartalmazza.

Utasításszámláló (Program Counter, PC): Utasításlehívás alatt a betöltendő utasítás kódszegmensen belüli eltolási címét tartalmazza, majd értéke inkrementálódik, így a következő utasításlehívásnál a következő utasítás címére mutat.

A különféle címzésmódokhoz használnak még egyéb címzőregisztereket:

- BP: bázismutató,
- SP: veremtármutató,
- SI: forrásindex-mutató,
- DI: célindex-mutató.

6.4. Belső sínrendszer

Ezen keresztül bonyolódik a belső egységek közti kommunikáció. Lehet egysínes, kétsínes és háromsínes kialakítású. Gyakorlati jelentősége már csak a két- és háromsínes rendszernek van.

6.5 A mikroprocesszorok alapvető típusai

A napjainkban alkalmazott processzorok igen sokfélék. Szinte nincs olyan terület, ahol nem alkalmaznak valamilyen processzort. A korszerű híradástechnikai berendezések, a háztartási gépek, az ipari mérőműszerek, a robotok vezérlését ma már többnyire mikroprocesszoros rendszerek végzik.

A processzorokat a vezérlőegység felépítése szerint a CISC és a RISC processzorok csoportjába sorolhatjuk.

CISC processzorok (Complex Instruction Set Computer)

Jellegzetes példájuk az I80X86 processzorcsalád. A processzorok között a lassúbb, olcsóbb kategóriát képviselik, tipikusan kis rendszerek, mikroszámítógépek, PC-k építésénél alkalmazzák.

A CISC processzorok jellemzői:

- bonyolult, sok gépi ciklusból álló utasítások,
- az utasítások végrehajtása több gépi ciklust igényel,
- sokféle utasítás és címzésmód létezik,
- sokféle utasítás szintű tárhozzáférés,
- bonyolult fordítóprogramot igényel,
- mikroprogramozott műveleti vezérlés,
- változó hosszúságú utasítások,
- kis számú regiszter.

RISC processzorok (Reduced Instruction Set Computer)

Jellegzetes példájuk a MIPS processzorok. Erős hardvertámogatású, gyors processzszorok, magas áruk miatt többnyire közepes, ill. nagyszámítógépek építésénél alkalmazzák.

A RISC processzorok jellemzői:

- kevés utasítás és címzési mód,
- az utasítások végrehajtása egy órajelciklust igényel,
- közvetlen memóriahasználatra csak két utasítás (STORE, LOAD) alkalmas,
- egyszerű fordítóprogramot igényel,
- huzalozott vezérlőegységgel rendelkezik,
- sok regiszterből álló regisztertárat tartalmaz,
- rögzített az utasítások hossza.

6.6. Az utasítás-végrehajtás párhuzamosítása

A processzor működési sebességének növelését legkönnyebben az órajelfrekvencia növelésével érhetjük el, de ennek határt szab a belső áramkörök működési sebessége.

A gyorsítás másik módszere az utasítások feldolgozásának átlapolása, idegen szóval ezt „pipelining"-nak nevezzük. Az utasítás feldolgozásának egyes lépései másmás erőforrást igényelnek. Ez adta az alapötletet a párhuzamosításhoz: ha egy erőforrás az előző utasítás feldolgozásánál felszabadul, akkor az esetleg igénybe vehető a következő utasítás feldolgozása során. Ez többlet hardvert igényel. A módszer nem tévesztendő össze azzal a megoldással, amikor a rendszerben egyszerre több különálló processzort használunk (multiprocesszoros rendszer).

Egy egyszerű példán keresztül tekintsük át a pipelining lényegét!

Tegyük fel, hogy a processzor egy utasítást a következő három lépésben dolgoz fel:

- utasításlehívás (Fetch): F,
- utasításdekódolás (Decode): D,
- utasítás-végrehajtás.

Ha az egyes lépéseket független, párhuzamosan működtethető vezérlőegységek hajtják végre, akkor az utasítások feldolgozásának lépései esetleg párhuzamosíthatók.

Egy idealizált párhuzamosított feldolgozást szemléltet a **6.17.** ábra. Az ábrán látható, hogy ebben az esetben, a harmadik lépésben már három különböző utasításhoz tartozó tevékenységet hajtunk végre (E1, D2, F3). Ez igen jelentős időmegtakarítást jelenthet a teljes program végrehajtása során.

Utasítás	Párhuzamos lépések				
1.	F1	D1	E1		
2.		F2	D2	E2	
3.			F3	D3	E3

6.17. ábra. Párhuzamos utasításvégrehajtás

A pipelining működés során felléphetnek problémák, mert az egyes elemi lépések zavarhatják egymást. A problémák okai a következők lehetnek.

- Az egyes elemi lépések végrehajtásához szükséges idők eltérőek. Ilyenkor késleltetéseket kell beiktatni, hogy a leglassúbb lépés végrehajtása is biztonságosan befejeződhessen.
- Az utasítások soros végrehajtását a vezérlésátadó utasítás megzavarhatja, mert nem a következő címen lévő utasítást kell végrehajtani. Ez csak az utasítás dekódolása után derül ki, amikor a következő címen lévő utasítás lehívása már megtörtént. Ilyenkor a betöltött utasítást törölni kell.
- A következő utasítás az előző eredményét használná, de az ebben a lépésben még nem áll rendelkezésre. A második utasítás feldolgozását késleltetni kell, pl. a fordítás során olyan utasítást kell ide helyezni, amely ebben a fázisban már végrehajtható. (Ezzel a második utasítás feldolgozását időben eltoljuk.)
- Az erőforrások használata során előfordulhatnak ütközések. A foglalt erőforrásokat nyilván kell tartani és a pipelining-ot fel kell függeszteni, ha egy folyamat ugyanezt az erőforrást kéri.

Ellenőrző kérdések, feladatok

1. Soroljuk fel a mikroprocesszor funkcionális egységeit, ismertessük a feladatukat!
2. Ismertessük a műveleti vezérlés fogalmát!
3. Határozzuk meg a perifériaírás műveleti vezérlését!
4. Ismertessük a huzalozott és fázisregiszteres vezérlés működését!
5. Ismertessük a mikroprogramozott vezérlés működését!
6. Ismertessük a RISC processzorok jellemzőit!
7. Ismertessük a CISC processzorok jellemzőit!
8. Ismertessük a pipelining fogalmát, jelentőségét!
9. Soroljuk fel a pipeling feldolgozásnál előforduló hibákat, ismertessük a hiba megszüntetésének módszereit!

7. AZ INTEL 8086-OS MIKROPROCESSZOR

A könyv eddigi fejezeteiben megismerkedtünk a számítógép mint digitális rendszer általános felépítésével, kialakítási lehetőségeivel. A továbbiakban két konkrét példán, az Intel 8086 mikroprocesszorán és az MCS 51 mikrokontroller-családon keresztül bemutatjuk az eddigi elvek gyakorlati megvalósítását.

A 8086 az Intel első 16 bites processzora. A processzor már magán viseli a korszerű mikroprocesszorok számos jegyét, de nem túlzottan bonyolult felépítésű és működésű, így bemutatásával jól szemléltethetjük a mikroprocesszorok konkrét megvalósítási módját. A 8086 felülről kompatíbilis a korábbi processzorokkal, azaz a 8086 végrehajtja a korábbi processzorok utasításait is. A 8086 mikroprocesszor részletesebb leírása a Tankönyvmester Kiadó: Alkalmazott számítástechnika c. tankönyvében található. A 8086 belső felépítése a **7.1.** ábrán látható.

A 8086 általános jellemzői:

- 16 bites regiszterkészlet;
- 20 bites cím (2^{20} = 1 Mbájt tárlókapacitás) kezelése;
- utasítások előjeles bináris számok és pakolatlan BCD számok szorzására, ill. osztására;
- multiprocesszoros működés lehetősége;
- utasítás-előolvasási sor bevezetése, az utasítás-beolvasás (fetch) és az utasítás-végrehajtás (execution) átlapolása érdekében;
- koprocesszor használatának lehetősége.

A 8086 adatformátumai:

- Bájt: 8 bites bináris szám vagy karakter (karakter esetén ez a karakter ASCII kódja),
- Word: szó, a 8086 által egyszerre kezelt adatmennyiség, ami 2 bájt hosszú,
- Duplaszó: 4 bájton ábrázolt szám,
- Tíz bájt: 10 bájt hosszúságú adatot ábrázol.

A negatív számokat kettes komplemens kódban, fixpontosban, a szám nagyságától függően 1–10 bájton ábrázoljuk. Létezik lebegőpontos ábrázolás is, amely egyenes kódban, előjelbittel ábrázolja a számot, ehhez azonban matematikai koprocesszorra (8087) van szükség.

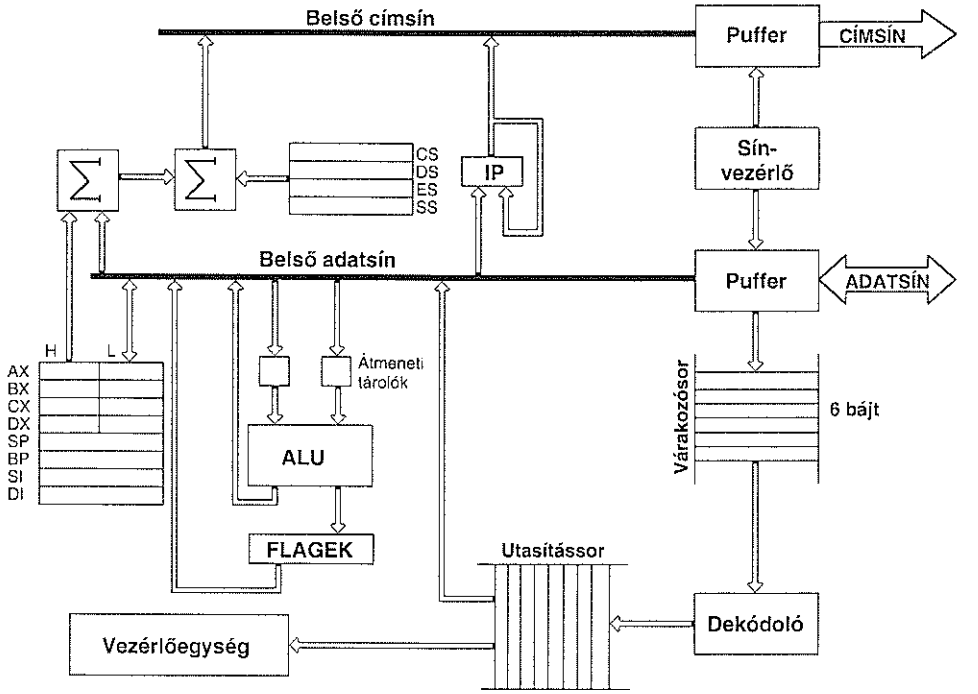
Lehetőség van a tízes számrendszerhez igazodó BCD kódú számábrázolásra.

Ezek tárolása kétféle módon lehetséges:

- pakolt BCD számábrázolással,
- pakolatlan (zónázott) BCD számábrázolással.

7.1. A 8086 belső felépítése

A 8086 belső felépítése a **7.1.** ábrán látható.



7.1. ábra. A 8086 belső felépítése

A processzor négy fő egysége a következő.

- **Sínvezérlő egység (Bus Unit, BU):** Puffereket és sínvezérlő áramköröket tartalmaz.
- **Utasítás dekódoló egység (Instruction Unit, IU):** A beérkező utasításokat kiolvassa az utasítássorból és előkészíti a vezérlőegység számára.
- **Végrehajtó egység (Excess Unit, EU):** A vezérlőegységet, az ALU-t, és a Flagregisztert tartalmazza, valamint rendelkezik egy regisztertárral.

- **Címkiszámító egység (Address Unit, AU):** Összegző áramkörökön kívül négy 16 bites szegmensregisztert is tartalmaz, amelyek a szegmensek kezdőcímének megadására használhatók. Található benne egy utasításszámláló (Instruction Pointer, IP) is, amelyben mindig a következő utasítás címe található.

Várakozó sor

A 8086 órajelfrekvenciája nem túl nagy. A mikroprocesszor gyorsítása érdekében vezették be a tervezők a várakozó sort (Instruction Queue). Így az utasítás-előolvasás (fetch) és az utasítás-végrehajtás (execution) tevékenységek *átlapolódnak*, azaz a végrehajtási szakaszban ebbe a hatbájtos belső memóriába olvassa be a mikroprocesszor a következő utasítást, így annak végrehajtására nem kell sokat várni, hiszen már a mikroprocesszorban van.

7.2. A 8086 utasítás-végrehajtása

A program indításakor az első utasítás címe automatikusan az IP-ben található. A címkiszámító egység (AU) megcímzi ezt a memóriarekeszt. Ez a címsín-puffer útvonalon keresztül a memória címregiszterébe kerül. Erről a címről a processzor kiolvassa az adatot (az utasítást) a memória adatregiszterébe. Ez az adatsín-puffer útvonalon keresztül bekerül a dekódolóba.

Innen az utasítás egy része (műveleti kód és címmódosító rész) a vezérlőegységbe kerül. A vezérlőegység ennek alapján végrehajtja a szükséges műveleteket.

Az utasítás címrésze a címkiszámító egységbe kerül, ami ennek alapján kiszámolja az utasítás végrehajtásához szükséges operandus pontos címét. Ez a cím a címsín-puffer útvonalon keresztül bekerül a memória címregiszterébe. Erről a memóriacímről a processzor kihozza az operandust az adatsínre, amit a pufferen keresztül betölt a végrehajtóegységbe.

7.3. A 8086 regiszterkészlete

Címregiszterek

- Utasításszámláló (Instruction Pointer, IP): Az éppen végrehajtandó utasítás címét tartalmazza.
- Veremmutató (Stack Pointer, SP): A veremtárba utoljára beírt adat címét tartalmazza.

- Állapotregiszter: A mikroprocesszor státuszbitjeit (flagjeit) tartalmazza.
- Bázismutató (Base Pointer, BP): A tár indexelt címzéséhez használjuk.
- Forrásterület-index (Source Index, SI): Adatterületek indexelt címzéséhez használjuk.
- Célterület-index (Destination Index, DI): Adatterületek indexelt címzéséhez használjuk.
- Kódszegmens regiszter (Code Segment, CS): Mindig az aktuális programmodul báziscímét tartalmazza.
- Veremszegmens regiszter (Stack Segment, SS): A veremként használt memóriaterület báziscímét tartalmazza.
- Adatszegmens regiszter (Data Segment, DS): A program fő adatterületének báziscímét tartalmazza.
- Extraszegmens regiszter (Extra Segment, ES): Egy másodlagos adatterület báziscímét tartalmazza.

Adatregiszterek

- **AX** (Accumulator): általános regiszter aritmetikai műveletekhez,
- **BX** (Base register): általános regiszter aritmetikai műveletekhez és címző regiszter bázisregiszteres indirekt címzéshez,
- **CX** (Counter register): általános regiszter aritmetikai műveletekhez és számlálóregiszter ciklusszervezéshez,
- **DX** (Data register): általános regiszter aritmetikai műveletekhez és címzőregiszter I/O műveletekhez.

7.4. A 8086 memóriakezelése

A 8086 minden regisztere 16 bites, címtere azonban 20 bites. A processzor a 20 bites címeket szegmentációval állítja elő. Egy szegmens 2^{16} = 64 Kbájt memóriaterületet jelent. A szegmensből nem lehet kicímezni, mert ha a címképzés során túlcsordulás keletkezik, akkor a cím elvész. A regiszterben tárolt legnagyobb értéket túllépve a regiszter tartalma nullázódik, és a legnagyobb helyiérték elé keletkezik átvitel, de nincs lehetőség eggyel több bit tárolására.

Stack memória (Veremtár):

LIFO szervezésű memória (Last In First Out), az utoljára beírt adat vehető ki először. Adatok átmeneti tárolására használjuk, ill. megszakítás esetén a processzor ide menti el a visszatéréshez szükséges információt (visszatérési cím, flagek). A veremkezelő utasítások automatikusan módosítják az SP veremtármutatót.

7.5. A 8086 I/O lehetőségei

A 8086 I/O tevékenységét portos rendszerben valósítja meg. Egy portot úgy foghatunk fel mint egy regisztert, amelyet speciális utasításokkal tudunk írni és olvasni. A portok a memóriához hasonlóan sorszámokkal címezhetők. Legfeljebb 64 K darab port köthető be, amelyet 0–255-ig közvetlen címzéssel, e felett pedig a DX regiszteren keresztül, regiszteres indirekt címzéssel tudunk megcímezni. A 8086 8 vagy 16 bites I/O portokat tud kezelni. Minden I/O művelet egyik operandusa AL vagy AX regiszter, a másik pedig a megcímzett I/O port.

7.6. A 8086 megszakítási (interrupt) rendszere

A 8086 256 szintű vektoros megszakítási rendszerrel rendelkezik. A memória elején a 00000h címtől kezdődően minden megszakításhoz egy 4 bájt hosszú blokk (ún. megszakítási vektor) tartozik. Az alsó két bájt tartalmazza a megszakítást kiszolgáló rutin ofszetjét (ez kerül be IP-be), a felső kettő pedig a báziscímét (ez kerül be a CS-be).

A 8086-nál programból is hívhatjuk bármelyik megszakítást az INT utasítással (szoftveres megszakításkérés).

7.7. A 8086 címzésmódjai

A mikroprocesszor a műveletvégzés operandusát a címzésmód alapján határozza meg. Az operandust tartalmazhatja maga az utasítás, ill. lehet regiszterben, a memóriában vagy I/O porton. A címzésmódokat részletesen a 4. fejezet tárgyalja. A 8086 a következő címzésmódokat használja:

- közvetlen adatcímzés (literális címzés),
- regisztercímzés,
- közvetlen (direkt) memóriacímzés,
- indexelt címzés,
- bázisrelatív címzés.

7.8. A 8086 utasításkészlete

Utasításkészletnek nevezzük a processzor utasításainak összességét. A 8086 CISC processzor, viszonylag sokféle utasítással rendelkezik, ez jelentősen megkönnyíti a programozását, viszont a fordításához bonyolultabb fordítóprogram szükséges.

A 8086 utasításkészlete a következő csoportokra osztható.

- **Adatmozgató utasítások** (pl. MOV operandus1, operandus2, adatmozgatás: operandus2-t operandus1-be írja).
- **Aritmetikai utasítások** (pl. ADD operandus1, operandus2, összegzés: operandus1+operandus2-t operandus1-be írja).
- **Korrekciós utasítások** (pl. DAA, két BCD szám összeadása után BCD korrekciót végez).
- **Logikai utasítások** (pl. AND operandus1, operandus2, ÉS művelet a két operandus között, bitenkénti ÉS kapcsolatot képez, és az eredményt operandus1-be írja).
- **Léptető, forgató (rotáló) utasítások** (pl. ROL operandus1, CL, forgatás balra, az operandus tartalmának jobbra forgatása a CL regiszter tartalmának megfelelő számú lépéssel).
- **Feltételbit (flag) műveletek** (pl. CLC, a carry flag törlése).
- **Vezérlésátadó utasítások** (pl. JZ címke, feltételes ugrás a címkével megjelölt sorra, ha a zéró flag értéke egy).
- **Ciklusszervező utasítások** (pl. LOOP címke, ugrás a címkére, ha CX regiszter tartalmát inkrementálva annak tartalma nem nulla).
- **Veremkezelő utasítások** (pl. PUSH operandus, az operandus mentése a verembe).
- **Szubrutinkezelő utasítások** (pl. CALL rutinnév, a vezérlés átadása a megadott nevű rutinra).
- **Megszakítások** (pl. INT megszakítási rutin, a megadott sorszámú megszakítási rutin hívása).
- **Sztring (karakterlánc) műveletek** (pl. LOADSB, karakter betöltése a DS:SI által megcímzett bájtból az AL regiszterbe).
- **I/O (bemeneti/kimeneti) utasítások** (pl. IN 078H, olvasás portról, a 078H című port tartalmát az akkumulátorba tölti).
- **Egyéb utasítások** (pl. HLT, a processzor leállítása hardver megszakításig).

Ellenőrző kérdések, feladatok

1. Ismertessük a 8086 általános jellemzőit!
2. Milyen adatformákkal képes műveleteket végezni a 8086?
3. Ismertessük a 8086 belső felépítését!
4. Mit jelent a várakozó sor, mi a feladata?
5. Soroljuk fel a 8086 regiszterkészletét, ismertessük az egyes regiszterek szerepét!
6. Ismertessük a 8086 memóriakezelését!
7. Mi a veremtár, mire használjuk?
8. Ismertessük a 8086 I/O rendszerének kialakítását!
9. Ismertessük a 8086 megszakítási rendszerét!
10. Ismertessük a 8086 címzési módjait!

8. AZ MCS 51 MIKROKONTROLLER-CSALÁD

Az egytokos mikroszámítógépeknek, vagy más néven mikrokontrollereknek nagy jelentőségük van a műszaki alkalmazásokban. Ezeket használják az elektronikus játékokban, háztartási eszközökben, autókban, híradástechnikai eszközökben, ipari folyamatirányító rendszerekben stb. Ezek a mikrokontrollerek a mikroprocesszoros célrendszerekből alakultak ki, a gyakorlatban szükséges funkcionális elemek egyetlen tokba integrálásával. Ennek megfelelően programozásuk, alkalmazásuk megegyezik a szokásos mikroprocesszoros rendszerekével. A mikrokontrollerek alkalmazására, ill. az MCS 48 mikrokontroller-családra vonatkozó további információk a Tankönyvmester Kiadó: Alkalmazott számítástechnika c. tankönyvében találhatók.

8.1. Az MCS 51 mikrokontroller jellemzői

Az egyik legnépszerűbb mikrokontroller-család az Intel cég MCS 51 jelű áramkörcsaládja, amelynek fő jellemzői a következők:

- 8 bites architektúra,
- beépített szorzó- és osztóműveletek,
- beépített ROM és RAM,
- belső órajelgenerátor,
- 64 Kbájt-ra bővíthető adat- és programtároló,
- két beépített időzítő-számláló egység,
- soros duplex adatátviteli port.

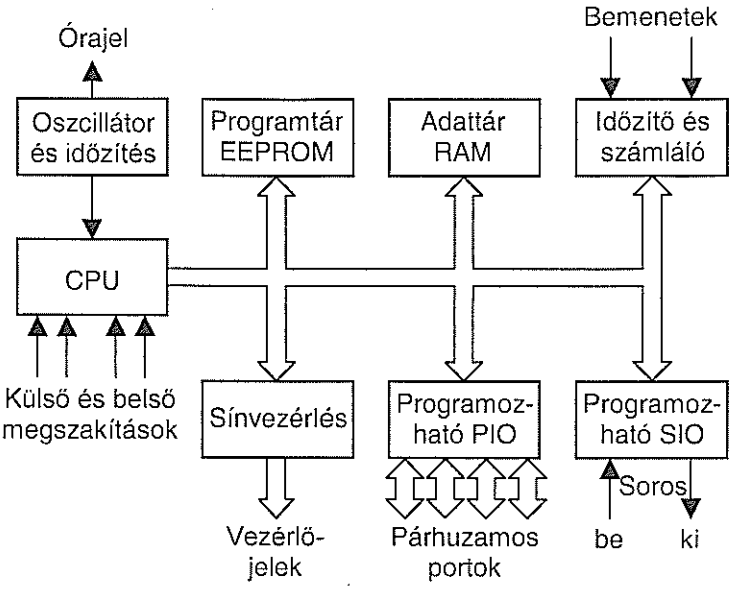
8.2. Az MCS 51 család elemeinek felépítése

Az MCS 51 áramkörcsalád elemeinek elvi felépítése látható az **8.1.** ábrán.

Társszervezés

A mikrokontrollerek egy belső, típustól függően 128 vagy 256 bájtos adattárolót tartalmaznak (ahol változókat tudunk tárolni), amihez különálló 4 Kbájt kapacitású

programtároló és 8 Kbájt méretű adattároló kapcsolható. A tokban két akkumulátor (A és B) és a működést meghatározó regiszterek (állapotregiszter, vezérlőregiszterek, be- és kimeneti regiszterek stb.) kaptak helyet.



8.1. ábra. MCS 51 család elemeinek elvi felépítése

Bemeneti- és kimeneti portok

A felhasználás szempontjából az egyik legfontosabb jellemző, hogy a kontroller mennyi és milyen portokkal rendelkezik. Az MCS 51 család elemei négy kétirányú, nyolcbites, párhuzamos portot tartalmaznak. Ezek közül a Port0 és Port2 külső, bővítő memóriamodul csatlakoztatására alkalmas. A Port1 és Port3 egyes kivezetései kettős funkciót látnak el. Egyrészt külső egységek adatainak kezelését teszik lehetővé, másrészt a külső egységek vezérléséhez szükséges jeleket is továbbítják. A kivezetések funkciója programból választható.

Időzítő- és számlálóegységek

A két beépített időzítő- és számlálóegység bármelyike (Timer0, Timer1) használható időzítőként vagy eseményszámlálóként.

Időzítő üzemmódban a regiszterek tartalma minden gépi ciklus során eggyel nő.

Számláló üzemmódban a regiszterek tartalma a megfelelő bemenetre érkező felfutó él hatására eggyel nő.

A számláló- és időzítőegység ezen kívül még számos feladatra programozható, pl. frekvenciaosztásra, automatikus újraindításra stb.

Soros port

A beépített duplex (egyidőben kétirányú) soros port megkönnyíti a kontroller és a környezete közötti kommunikációt. A port adatátviteli sebessége és formátuma egy vezérlőregiszterrel programozható.

Megszakítások

A mikrokontroller ötszintű vektoros megszakítási rendszerrel rendelkezik (öt különböző megszakításforrása lehet). A megszakításvezérlő regiszter segítségével az öt forrás mindegyike függetlenül tiltható-engedélyezhető. Az INT0 és INT1 külső megszakítások, a soros porthoz egy, az időzítőkhöz két megszakítás tartozik. A megszakítások prioritása (fontossági sorrendje) programozható.

Utasításkészlet

A mikrokontroller utasításkészlete 111 utasításból áll, a fontosabb utasításcsoportok a következők.

Adatmozgató utasítások (pl. MOV, bitek, bájtok mozgatására),

Aritmetikai utasítások (pl. MUL A, B, az A és B regiszter közötti szorzás),

Logikai utasítások (pl. SETB B, a B regiszter valamely bitjének 1-be állítása),

Vezérlőutasítások (pl. SJMP, ugrás a –128 +127 tartományon belül).

A kontroller programozásához kifejlesztettek egy magasrendű programnyelvet az AH BASIC-et, amely egy strukturált programozást támogató BASIC verzió.

Ellenőrző kérdések, feladatok

1. Mi a mikrokontroller, mire hasznnálják?
2. Ismertessük az MCS 51 mikrokontroller-család fő jellemzőit!
3. Ismertessük az MCS 51 mikrokontroller-család társzervezését!
4. Mi az időzítő-számláló egységek feladata?
5. Milyen portokat tartalmaznak az MCS 51 mikrokontroller család egyes tagjai?
6. Ismertessük az MCS 51 mikrokontroller-család megszakítási rendszerét!