Értékünk az ember
Humánerőforrás-fejlesztési Operatív Program

Benyó Balázs

SZÁMÍTÓGÉPEK ARCHITEKTÚRÁJA

SZÉCHENYI ISTVÁN EGYETEM
GYŐR

Magyarország célba ér
NEMZETI FEJLESZTÉSI TERV

Készült a HEFOP 3.3.1-P.-2004-09-0102/1.0 pályázat támogatásával.

Szerző: dr. Benyó Balázs
egyetemi docens

Lektor: Németh Gábor
egyetemi docens

A dokumentum használata

Mozgás a dokumentumban

A dokumentumban való mozgáshoz a Windows és az Adobe Reader megszokott elemeit és módszereit használhatjuk.

Minden lap tetején és alján egy navigációs sor található, itt a megfelelő hivatkozásra kattintva ugorhatunk a használati útmutatóra, a tartalomjegyzékre, valamint a tárgymutatóra. A ◀ és a ▶ nyilakkal az előző és a következő oldalra léphetünk át, míg a Vissza mező az utoljára megnézett oldalra visz vissza bennünket.

Pozícionálás a könyvjelzőablak segítségével

A bal oldali könyvjelző ablakban tartalomjegyzékfa található, amelynek bejegyzéseire kattintva az adott fejezet/alfejezet első oldalára jutunk. Az aktuális pozíciónkat a tartalomjegyzékfában kiemelt bejegyzés mutatja.

A tartalomjegyzék használata

Ugrás megadott helyre a tartalomjegyzék segítségével

Kattintsunk a tartalomjegyzék megfelelő pontjára, ezzel az adott fejezet első oldalára jutunk.

Keresés a szövegben

A dokumentumban való kereséshez használjuk megszokott módon a Szerkesztés menü Keresés parancsát. Az Adobe Reader az adott pozíciótól kezdve keres a szövegben.

Tartalomjegyzék

Előszó

A jegyzet műszaki informatikus hallgatók számára készült, tartalmában a Széchenyi István Egyetem műszaki informatika BSc szakának tantervében szereplő Számítógép architektúrák tárgy tematikájához igazodik. Az egyes fejezetek egymásra épülve, könnyen érthető módon mutatják be a számítógép működését, a modern számítógépben alkalmazott megoldásokat, ezért más szakot hallgató diákok, ill. az informatika és a számítógépek iránt érdeklődők számára is jól használható irodalom.

A jegyzet az alapoktól indulva mutatja be a számítógépes rendszerek működését, kevés számítógépes alapismeretre épít. Egyes részek feltételeznek jártasságot a matematika, a boolean algebra és a digitális technika alapjaiban.

A jegyzetben igyekeztünk azokra a számítógép architektúrák tárgykörébe tartozó témákra fókuszálni, amelyeket a műszaki informatika szakon tanuló leendő rendszer-tervezők, informatikusok, programozók, rendszergazdák közvetlenül fel tudnak használni majdani munkájuk során. A jegyzet egyes fejezeteiben az adott területhez kapcsolódó elméleti ismeretek és elvek bemutatása után a gyakorlatban is alkalmazható tudás átadására helyeztük a hangsúlyt, a BSc szak követelményeinek megfelelően.

A jegyzet ábráinak jelentős része Irv Englander The Architecture of Computer Hardware and Systems Software: An Information Technology Approach című könyve alapján készült. A jegyzet írásakor azért választottuk ezt a forrást, mert ennek tematikája illeszkedik leginkább a jegyzet írásakor kitűzött célokhoz.

Köszönöm közvetlen kollégáimnak, elsősorban Kovács Katalinnak és Horváth Zsoltnak a jegyzet megírásában nyújtott hathatós segítségét. Köszönöm a hallgatók visszajelzéseit, melyek sokban segítettek a jegyzet mostani formájának kialakításában.

Remélem hasznos olvasmány lesz mindenki számára ez a jegyzet, és minden Olvasója számára eredményesen segíti a számítógép architektúra ismeretek elsajátítását!

A Szerző

I. RÉSZ

SZÁMÍTÓGÉPES RENDSZEREK MŰKÖDÉSÉNEK ÉS FELÉPÍTÉSÉNEK ÁTTEKINTÉSE

1. Számítógépes rendszerek felépítése

Számítógép segítségével számos feladatot tudunk megoldani. Ezek a feladatok a számítógép felhasználója számára egymástól nagyon eltérőek lehetnek. A feladatokat végrehajtó számítógépek első látásra – az elvégzett feladatokhoz hasonlóan – egymástól nagyon különböznek. Alaposabban szemügyre véve azonban a számítógépek felépítésük és működésük szempontjából számos olyan tulajdonsággal rendelkeznek, melyek minden számítógépes rendszerre jellemzőek. Ebben a fejezetben ezeket a jellemzőket mutatjuk be.

A számítógépes rendszerek működésének megértéséhez és általános leírásához nézzük meg, hogyan történik egy-egy számítógéppel támogatott szolgáltatás végrehajtása.

Vegyünk néhány egyszerű példát. Az első példánk egy számítógépes átutalás indítása legyen. A felhasználó elindít egy böngésző programot, megkeresi a bankja által készített számlavezető oldalt, ott megadja a rendszerbe történő belépéshez szükséges azonosítási információkat. Amennyiben sikeres lesz az azonosítás, a böngészőben a számlavezető-rendszer megjeleníti a felhasználó számlainformációit. A megfelelő menüpontot kiválasztva a felhasználó átutalást kezdeményez, melyhez meg kell adni az átutalás adatait. Az átutaláshoz megadott adatok jóváhagyása után a rendszer elvégzi az utalási tranzakciót, majd a tranzakció eredményéről tájékoztatást küld. Az átutalás befejezése után a felhasználó kilép a banki rendszerből.

A második példa egy SMS elküldése. A felhasználó kiválasztja a telefon menüjéből a SMS-küldés menüpontot. Begépeli az üzenet szövegét, majd megadja a címzett telefonszámát és elküldi az üzenetet. Az üzenet sikeres vagy sikertelen küldéséről a felhasználó a rendszertől visszajelzést kap.

Harmadik példánk egy bank-automatából történő pénzfelvétel. A felhasználó behelyezi a bankkártyáját az automatába, majd megadja az azonosító információit, ami általában a PIN kódja. Ha a felhasználó azonosítása sikeres, a rendszer kéri a felvenni kívánt összeget, ill. esetlegesen további információkat (pl. kért-e nyugtát stb.). Majd ha minden, a pénzkiadáshoz szükséges feltétel teljesül, akkor az automata visszaadja a bankkártyát, kiadja a pénzt és a nyugtát.

Példáinkból jól látszik, hogy a számítógépes rendszerek megjelenési formájukat és a felhasználó számára nyújtott szolgáltatásaikat tekintve igen

különbözőek, azonban minden esetben használatukhoz adatokat kellett a számítógépes rendszerbe bevinni, az adatokat a rendszer feldolgozta, majd az eredményt megjelenítette a felhasználó számára értelmezhető formában. A működés során bevitt adatok feldolgozáshoz adott esetben további, korábban tárolt adatokat is felhasznált a rendszer.

Az első két példában az információ bevitele grafikus felhasználói felületen történt, a harmadik példában a bevitt információ részben a bankkártyán tárolt információ volt. Az első és a harmadik példa esetében biztosak lehetünk abban, hogy a szolgáltatás megvalósításához, vagyis a feldolgozás lépéshez a rendszernek korábban tárolt adatokat is fel kellett használni: a bankszámla adatait, azon levő egyenleget, különböző – felhasználó által korábban megadott – készpénz-felvételi, ill. utalási limitet stb.

Az információ bevitelére használt eszközök a példáinkban igen változatosak: a billentyűzet és az egér az első példában, mobil telefon billentyűzete a második példában, érintő képernyő és kártyaolvasó a harmadik példában. Az eredmény megjelenítésére minden esetben a képernyőt használták a különböző rendszerek, a harmadik példában a képernyőn kívül a nyugta-nyomtatót és a készpénzkiadó eszközt is.

1.1. Input-Process-Output (Beolvasás–Feldolgozás–Kiírás) Modell

A számítógépes rendszerek működése három fontos lépésre bontható fel:

- Input (Beolvasás): adatok bevitele a számítógépbe.
- Process (Feldolgozás): különböző műveletek végzése a számítógépben tárolt adatokon.
- Output (Kiírás): a feldolgozás eredményének megjelenítése a szolgáltatást igénybevevő felhasználó által értelmezhető formában.

Bármilyen számítógépes rendszer működése felosztható ezekre a lépésekre, legyen az egy számítógépes játék, egy robotokkal támogatott automatizált ipari összeszerelő-sor, vagy egy autó blokkolásgátló rendszerének a vezérlője. Az egyes lépések végrehajtása időben egymással átlapolódhat, ez adott esetben megnehezítheti a lépések szétválasztását. Az interaktív alkalmazások használatakor – például egy szövegszerkesztő használatakor – az adatbevitel/feldolgozás/adatmegjelenítés jellemzően időben párhuzamosan történik.

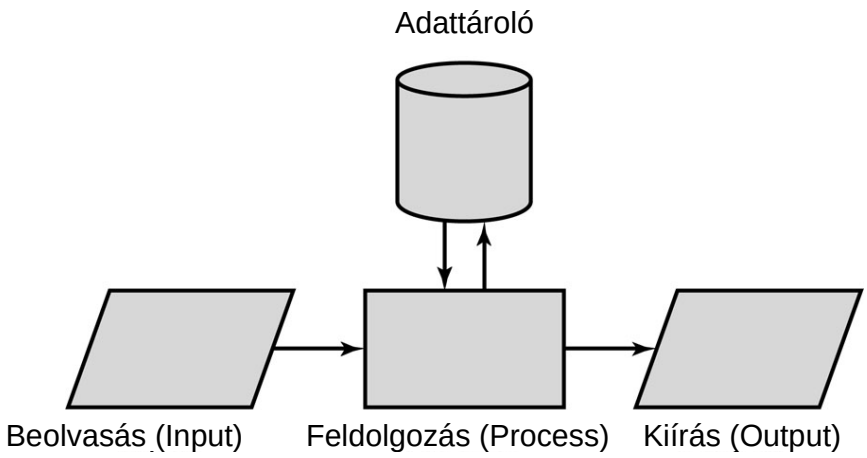
Az egyes számítógépes rendszerek között a különbséget a lépések végrehajtási sorrendje, a beolvasáshoz/feldolgozáshoz/megjelenítéshez használt eszközök fogják jelenteni.

Általános célú számítógépes környezetben a leggyakrabban használt eszközök az egyes lépések végrehajtására a következők:

- Input (Beolvasás): billentyűzet, egér, lapolvasó (scanner).
- Process (Feldolgozás): CPU, memória.
- Output (Kiírás): monitor, nyomtató, fax.

A számítógépes rendszerek működéséhez, a feldolgozás lépés megvalósításához a számítógépbe bevitt adatok átmeneti és hosszútávú tárolására van szükség. A számítógépben leggyakrabban használt tároló eszközök: merevlemez, optikai lemez, hajlékony-lemezek, mágnes kazetta.

A számítógépes feladatok végrehajtásának ezen leegyszerűsített leírását nevezzük Input-Process-Output (röviden IPO) modellnek, mely modell áttekintését az 1.1. ábra mutatja.



1.1. ábra. Input-Process-Output modell

A beolvasás, feldolgozás és kiírás lépések további részfeladatokra bonthatók, például adatállományok mozgatása, konvertálása, keresés bennük stb. A számítógép programok készítésekor ezeket a részfeladatokat kell tovább pontosítani, lebontani a számítógép által közvetlenül végrehajtható műveletekre. A számítógép programozás nem más, mint annak a számítógép

által végrehajtható műveleteket tartalmazó műveletsornak a definiálása, mely a felhasználó által kívánt feladatot megvalósítja.

A számítógép által közvetlenül végrehajtható műveletek igen egyszerűek, ezek típusait a 7.11. fejezetben meg fogjuk ismerni. Ezen műveletekből összeállítani egy számítógépes programot igen időigényes lenne, ezért a gyakorlatban ún. magas szintű programozási nyelveket használunk. Ezen nyelvek utasításaihoz bonyolultabb működés tartozik. Magas szintű programozási nyelvek jellemző utasításai a következők:

- adatbeolvasási/kiviteli utasítások, adatállomány írása, olvasása;
- aritmetikai műveletek, logikai műveletek, értékadó műveletek különböző típusú adatelemeken;
- feltételes elágazás utasítások;
- ciklus utasítások.

A magas szintű programozási nyelvek utasításaiból álló programokat az ún. fordítóprogramok „fordítják le” a számítógép által közvetlenül végrehajtható műveletekre, vagyis állítják elő a magas szintű programozási nyelven megírt programból a számítógépen futtatható, a számítógép által közvetlenül végrehajtható műveletsorozatot. Ezt azért lehet megtenni, mert egy-egy magas szintű programozási nyelvi műveletet egyértelműen meg tudunk feleltetni egy számítógép által közvetlenül végrehajtható műveletsorozatnak.

A magas szintű programozási nyelvek utasításait a különböző számítógépeken más és más, az adott számítógép architektúrájától függő műveletsorozat fogja végrehajtani. A magas szintű programozási nyelvek alkalmazásának egyik fontos előnye, hogy használatukhoz, vagyis egy program elkészítéséhez, nem szükséges az adott hardver részletes ismerete. A fordítóprogram a magas szintű programozási nyelven írt programot lefordítja adott hardverkörnyezetben végrehajtható utasítássorozattá.

A számítógép programok készítésében az ún. objektum-orientált (OO) programozás lényeges változást hozott. Az egyes programok utasítássorozatait OO környezetben már utasításblokkonként fejlesztik és tárolják. Ennek számos előnye van, a kód-újrafelhasználás lehetősége, áttekinthetőbb programszerkezet az összetartozó adatelemek és utasításblokkok együttes kezelése miatt, vagy az egyszerűbb program karbantartás és továbbfejlesztés lehetősége. Fontos kiemelni, hogy az OO szemlélet lényeges változást hozott a szoftverfejlesztésben, de nem változtatott az IPO modell érvényességén. Az OO környezetben fejlesztett programok is ha-

sonlóan hajtódnak végre, mint a más programozási környezetben fejlesztett társaik, a különbség a mi szempontunkból talán annyi, hogy az OO forráskódban nehezebben követhető az egymás után végrehajtott utasítások sorozata.

1.2. Számítógépes rendszerek általános felépítése

A különböző számítógépes rendszereket feloszthatjuk négy fontos komponensre:

1. Számítógép hardver. Ezek az elemek biztosítják az adatbevitel, adattárolás, adatfeldolgozás és adatmegjelenítés fizikai megvalósítását. Jellemzően több hardver komponensből áll egy számítógép, és hozzá tartozik az a vezérlő elektronika is, mely az elemeket külön-külön, ill. azokat közösen vezérli.
2. Számítógép szoftver. A számítógép szoftver utasítások formájában definiálja, hogy egy-egy szolgáltatás megvalósításához (egy adott feladat végrehajtásához) a számítógép hardver milyen műveleteket és milyen sorrendben kell, hogy végrehajtson.
3. A számítógép által tárolt és feldolgozott adatok. Különböző típusú (egész szám, lebegőpontos szám, karakteres szöveg, grafikus kép stb.) információ ábrázolása olyan formában, melyet a számítógép közvetlenül fel tud dolgozni.
4. Számítógép kommunikációt lehetővé tevő elemek. Ezek az elemek hardver és szoftver részeket is tartalmaznak. Lehetővé teszik, hogy a számítógép más számítógépekkel kapcsolatba léphessen, azokkal adatcserét folytasson. Ezeket az elemeket a többi elemtől nem az elemek jellege (szoftver, hardver stb.), hanem azok funkciója különbözteti meg a többi elemtől.

A számítógépes szakirodalom a rendszereket hagyományosan az első három komponensre osztja. A negyedik komponens különböző részeit – mint azt említettük is – a szoftver és hardver elemek közé sorolja. A modern számítógép-tudomány azonban az elosztott rendszerek rohamos terjedése miatt előszeretettel különbözteti meg ezt a negyedik komponensét a számítógépes rendszereknek, annak növekvő fontossága miatt. Ezen negyedik komponensbe sorolt elemek teremtik meg a kapcsolatot különböző számítógépek között és teszik lehetővé az elosztott – több számítógépet igénybevevő – feldolgozás lehetőségét.

A számítógép architektúráját az első két komponens, vagyis a számítógép hardver és a számítógép szoftver alkotja. Amikor a számítógép architektúrájáról beszélünk, akkor ennek a két komponensnek a felépítéséről, részeiről és működéséről beszélünk. A számítógép architektúráját meghatározó komponensek közé a szoftver elemek közül csak az ún. operációs rendszerhez tartozó komponenseket soroljuk. Ezek a szoftver komponensek működtetik a hardver elemeket, teszik lehetővé további alkalmazások futtatását. A felhasználótól erősen függő, ún. alkalmazói szoftverekkel a számítógép architektúrája kapcsán nem foglalkozunk. A számítógép szoftver komponenseinek erről a felosztásról a következő két alfejezetben még lesz szó.

Az alkalmazói szoftverek és a számítógép által tárolt és feldolgozott adatok a számítógépes rendszerek működéséhez alapvetően szükséges elemek, azonban azok felépítése, ill. jellemzői elsősorban a felhasználó igényeit tükrözik, nem függenek a számítógép architektúrájától.

1.2.1. Számítógép hardver elemei

A számítógépek hardver elemei közül a felhasználó az adatmegjelenítő, ún. output (kimeneti) és az adatbevitelt lehetővé tevő, ún. bemeneti (input) eszközökkel találkozik közvetlenül. Ezeket az eszközöket közös elnevezéssel szokták még perifériáknak, vagy perifériás eszközöknek nevezni, mely név megkülönbözteti őket a számítógép magját alkotó elemektől.

A számítógép magját a központi feldolgozó egység (Central Processing Unit, CPU) valamint a memória képezi, mely mag egy külső interfész egységen keresztül lehetőséget ad a perifériák csatlakoztatására. Ez a mag képes elvégezni a programokban definiált adatfeldolgozó, adatmozgató műveleteket, vagyis az IPO modell feldolgozás lépésének műveleteit.

A számítógép magjának műveletvégzés szempontjából legfontosabb elemét, a CPU-t működés szempontjából további három részre bonthatjuk:

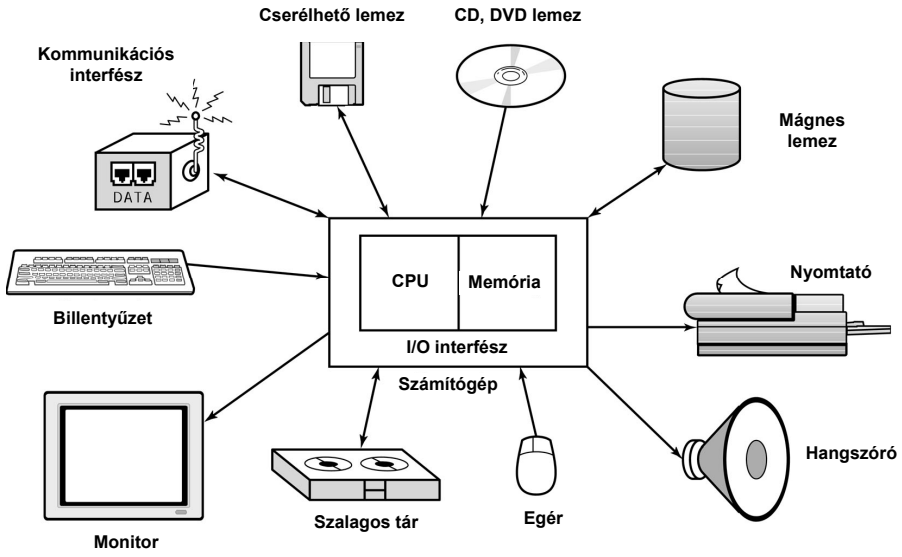
1. Aritmetikai/logikai művelet-végrehajtó egység (Arithmetic/Logic Unit, ALU). Ez az egység végzi az aritmetikai és Boolean logikában definiált adatmódosító műveleteket az adatokon.
2. Vezérlő egység (Control Unit, CU). Ez a komponens felelős a CPU vezérléséért és gondoskodik az utasítások CPU-n belüli helyes végrehajtásáról, az adatok CPU-n belüli mozgatásáról.
3. Interfész egység. Ez az elem felelős az adatoknak a CPU és a többi hardver elem közötti mozgatásáért.

A CPU ALU és CU egysége a CPU interfész egységén keresztül kapcsolódik a számítógép további részeihez. Az összeköttetés jellemzően ún. adat sínen (bus-on) keresztül valósul meg. A sín (bus) a számítógépekben általánosan használt kapcsoló elem. A sín elektromos jeleket továbbító kábelek együttese a kommunikációt lehetővé tevő vezérlő logikával kiegészítve. A sín egy vagy több számítógép komponens között teremt kapcsolatot, tesz lehetővé egy vagy kétirányú adatcserét a sínre kapcsolt egységek között. A sínen szállított jelek lehetnek adatértékek, a kommunikáció irányításához használt vezérlőjelek vagy tápfeszültség. Sín kapcsolatot használunk – mint említettük – perifériák csatlakoztatására, de például a CPU-n belüli komponensek összekapcsolására is.

A memóriáknak több típusa létezik. Van csak olvasható ROM (Read Only Memory) típusú memória, és RAM (Random Access Memory) írható/olvasható memória. A RAM memória jellemzője, hogy az adatok tárolásához szükséges elektromos feszültség, tehát a számítógép kikapcsolásával a benne tárolt adatok törlődnek, viszont az adatok elérése a többi tárolóval összehasonlítva gyors. A ROM valamivel lassabb adatelérést tesz lehetővé, azonban a benne tárolt adatok nem vesznek el a számítógép kikapcsolásakor.

A különböző típusú memóriákat közös elnevezéssel nevezik még elsődleges tárolónak (primary storage) megkülönböztetve őket a másodlagos tárolóktól (secondary storage). A másodlagos tárolók a háttértárak (mágneslemezek, mágnesszalag, optikai tárolók [CD, DVD] stb.), melyek az elsődleges tárolóknál lassabb adatelérést tesznek lehetővé. Cserében viszont tároló kapacitásuk általában jóval nagyobb és a bennük tárolt adatok nem vesznek el a számítógép kikapcsolásakor. A másodlagos tárolók a számítógép interfész egységen kapcsolódnak a számítógép magjához, tehát a másodlagos tárolók a perifériák közé tartoznak.

A perifériák többi része – mint korábban említettük – elsősorban ki- és bemeneti eszköz. Ezeknek egy csoportja a kommunikációt teszi lehetővé különböző számítógépek között. A számítógépek általános felépítésének blokksémáját az 1.2. ábra mutatja.



1.2. ábra. Számítógépek általános felépítése

1.2.2. Számítógép szoftver elemei

A szoftver fogalmának pontos meghatározására a számítógép-tudomány számos definíciót készített. Számunkra elegendő az az egyszerű meghatározás, hogy a szoftver a számítógépes rendszer azon része, mely meghatározza a számítógép hardverje által végrehajtott utasítások sorozatát.

A szoftver tehát egy olyan adatállomány, mely a számítógép által végrehajtható utasítások sorozatát tartalmazza. (A szoftver fogalom tágabb meghatározásaival itt nem foglalkozunk.) Egy számítógépen számos ilyen állomány található.

A szoftvereket két nagy csoportra osztjuk az általuk megvalósított szolgáltatások, funkciók alapján:

- Operációs rendszer. Az operációs rendszer, vagy másnéven rendszer szoftver olyan alapfunkciókat valósít meg, mely minden számítógép felhasználó számára szükséges a számítógép használatához.
- Alkalmazói szoftverek. Az alkalmazói szoftverek, vagy más néven alkalmazások, felhasználói szoftverek olyan programok, melyek egy-egy felhasználói kör speciális igényeit elégítik ki. Vannak közöttük olyanok, melyeket sok felhasználó használ, például egy kisvállalati könyvelést végző szoftver, de van olyan is, mely egészen egyedi, például egyetlen bonyolult tudományos számítás elvégzésére kifejlesztett egyedi program.

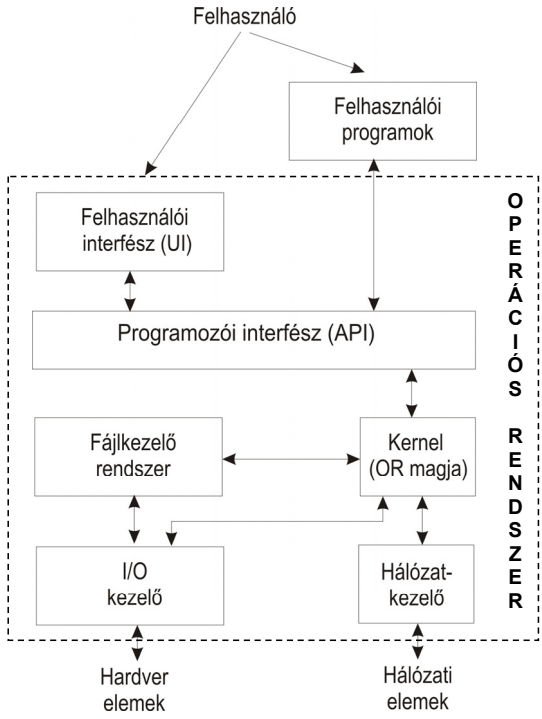
A két szoftver kategória között a határ nem szigorú, a számítógépek fejlődésével újabb és újabb szolgáltatások kerültek bele az operációs rendszerekbe.

Az operációs rendszer – mint említettük – szolgáltatásokat nyújt. Egyrészt szolgáltatásokat nyújt közvetlenül a felhasználóknak, másrészt szolgáltatásokat nyújt az adott számítógépen futó alkalmazásoknak. Ennek megfelelően az operációs rendszereknek két fontos interfésze van:

- Felhasználói interfész. A felhasználói interfész segítségével a felhasználók közvetlenül tudják az operációs rendszer szolgáltatásait aktivizálni. Például adatállományokat tudnak létrehozni, tartalmukat megjeleníteni, esetleg azokat módosítani, alkalmazásokat indítani, a számítógépen futó alkalmazásokat kilistázni stb. A felhasználói interfész lehet egyszerű parancssoros, szöveges interfész (pl. UNIX), vagy grafikus, ahol menük és ablakok segítik a munkát (pl. Windows XP).
- Programozói interfész. A programozói interfész a programok futtatását teszi lehetővé az adott számítógépen. Az operációs rendszer tartalmazza a hardver elemek közvetlen kezelését megvalósító funkciókat. Az operációs rendszer ezeket a funkciókat elérhetővé teszi a programok számára, megkönnyítve ezzel az alkalmazások készítését a programozók számára. A programozói interfész ezen szolgáltatások (függvények) elérését teszi lehetővé ún. rendszerhívások formájában.

A programozói interfész használatát az teszi lehetővé, hogy a modern számítógépes rendszerek megengedik, hogy egy végrehajtható program utasításai ne egyetlen adatállományba legyenek eltárolva. Létrehozhatunk ún. programkönyvtárakat tartalmazó adatállományokat, melyek nem tartalmaznak önállóan futtatható programot, de a bennük levő utasításrészleteket, utasítássorozatokat ún. függvényeket más programok használhatják, hivatkozva rájuk. Ha több állományban vannak a program utasításai, akkor természetesen minden programrészletet tartalmazó állománynak elérhetőnek kell lenni a program futatásakor. Mint említettük az operációs rendszerek tartalmazzák azokat az utasítássorozatokat függvények formájában, melyek a hardver elemek közvetlen kezelését valósítják meg. Az alkalmazói programoknak nem kell mást tenni, mint ezekre az operációs rendszer által biztosított függvényekre, ill. az őket tartalmazó függvénykönyvtárakra hivatkozni, így könnyen végre tudják hajtani a hardver kezelést igénylő feladatokat.

Rendszertechnikai szempontból sokkal fontosabbak az operációs rendszer azon szolgáltatásai, melyet nem közvetlenül a felhasználónak nyújt, hanem az alkalmazások számára biztosít. A szoftver komponensek felépítését és együttműködését az 1.3. ábra mutatja. Látható, hogy a felhasználói interfészt megvalósító szoftver komponens is a programozói interfészen keresztül éri el az operációs rendszer szolgáltatásait, és teszi egyszerűen elérhetővé azokat a felhasználó számára.



1.3. ábra. Számítógépek szoftver komponensei és azok kapcsolatai

Fontos megemlíteni, hogy az operációs rendszernek fontos vezérlő feladatai is vannak. Össze kell hangolnia a hardver elemek működését. Egy modern rendszerben a hardver elemek jellemzően párhuzamosan dolgoznak, gyakran több program is fut egymással időben átlapolva. Ez teszi az operációs rendszerek készítését igen bonyolult feladattá.

Az operációs rendszer sem homogén, több részre osztható. Mint említettük az operációs rendszer különböző funkciókat, szolgáltatásokat nyújt a felhasználók, ill. az operációs rendszerben futó programok számára. Az operációs rendszerek komponenseit általában úgy határozzuk meg, hogy

az azonos funkciót megvalósító részek egy komponensbe kerüljenek. A hardver elemek elérését biztosító operációs rendszer komponenseket hardver meghajtóknak ritkábban I/O kezelőknek (angolul driver-eknek) nevezzük. Az operációs rendszer alapfunkcióit megvalósító, jórészt állandóan futó komponenst a számítógép magjának, vagy más néven kernelnek nevezzük.

Az operációs rendszer egyes szolgáltatásait gyakran külön komponensben valósítják meg. Így általános, hogy egy operációs rendszerben létezik file rendszer komponens, mely az adatállományok kezelését, tárolását valósítja meg, vagy van a hálózati kapcsolat elérést lehetővé tevő komponens az ún. hálózati alrendszer. A gyakorlatban használt operációs rendszerek általában más komponenseket is tartalmaznak. Az 1.3. ábra az operációs rendszer komponenseinek egy lehetséges jellemző felosztását mutatja.

II. RÉSZ

ADATOK TÁROLÁSA SZÁMÍTÓGÉPES RENDSZEREKBEN

2. Adatábrázolás különböző számrendszerekben

A számítógépek legfontosabb alapfunkciója adatok tárolása és feldolgozása. A tárolt adatokat különböző típusba sorolhatjuk annak alapján, hogy milyen típusú információt tárolunk bennük, vagyis a tárolt adatokat milyen típusú információnak feleltetjük meg.

Számítógépekben általánosan használt adattípus például az egész szám, a szöveg (karakterek sorozata), vagy a valós számok (tört és egész értékek) ábrázolására szolgáló ún. lebegőpontos szám. Ezeknél az egyszerű adattípusoknál összetettebb információt leíró adatokat is tárolunk számítógépeken, így álló és mozgóképek, hangok, dokumentumok tárolására szolgáló adatállományok is léteznek. Amikor különböző típusú adatok számítógépes tárolásáról beszélünk, akkor elsősorban az egyszerű adattípusok tárolását értjük ez alatt. Mi is az egyszerű adattípusok tárolásával fogunk foglalkozni először ebbe a fejezetben. A hangok, képek, dokumentumok tárolási módszerei bonyolultabbak, ezeket az egyszerű adattípusok után tekintjük át.

A számítógépen történő adattárolásnál meg kell különböztetni az adatok fizikai tárolását a tárolt adatok értelmezésétől. A számítógép minden adat tárolásakor egy jelsorozatot rögzít, melyet később vissza tud olvasni. Ezt nevezzük fizikai tárolásnak. A korábban eltárolt, majd visszaolvasott jelsorozatot a számítógép az eltárolt adat típusától függően értelmezi, ami azt jelenti, hogy megfelelteti az adott adattípus esetén használt értéknek.

A különböző típusú adatok fizikai tárolása a számítógépen igen hasonlóan történik, a különbség a tárolt adatértékek értelmezésében van. Az adatok számítógépen történő tárolásának legfontosabb kérdése, hogy a fizikailag tárolt adatértékeket hogyan rendeljük össze az eltárolni szándékozott információ elemeivel.

A tárolás logikájának megértéséhez szükséges, hogy megismerkedjünk a számok kettes számrendszerben történő ábrázolásával. Ebben a fejezetben először röviden áttekintjük, hogy milyen elméleti módszerek léteznek különböző típusú adatok ábrázolására, majd megismerkedünk a kettes számrendszerben történő számábrázolással.

2.1. Számértékek ábrázolása, számrendszerek

A számok, ill. számértékek ábrázolásán azok valamilyen jelsorozattal történő leírását, fizikai rögzítését értjük. A számok ábrázolásakor mindig valamilyen jelölésrendszert használunk. A jelölésrendszer meghatározza, hogy mely értékhez mely jelsorozat fog tartozni, vagyis megadja az értékek és a jelsorozatok egyértelmű és kölcsönös megfeleltetését.

Ugyanannak az értéknek két különböző jelölésrendszerben általában különböző jelsorozatot feleltetünk meg. A tizenkettő értéknek például római számokat használva a „XII" jelsorozat, míg arab számokat és decimális számrendszert használva a „12" jelsorozat felel meg.

A modern – elsősorban európai hagyományokra épülő – kultúrákban a számértékek ábrázolására az ún. helyiértékes jelölést használjuk.

A helyiértékes jelölés szisztematikus jelölésrendszer. Használatához kiválasztunk egy alapszámot (értéket), ez lesz a jelölésünk alapszáma. Egy adott alapszámot alkalmazó helyiértékes jelölésrendszert számrendszernek nevezzük, a kiválasztott alapszámot pedig a számrendszerünk alapjának. Minden számrendszerben annyi különböző számjegyet használunk az értékek ábrázolására, amennyi a számrendszer alapja. Hagyományosan ha b a számrendszer alapja, akkor a *0..b-1* számjegyeket használjuk. Kettes alapú (bináris) számrendszer esetén a használható számjegyek: 0 és 1; nyolcas alapú (oktális) számrendszer esetén a használható számjegyek: 0, 1, 2, 3, 4, 5, 6, és 7; tizenhatos alapú (hexadecimális) számrendszer esetén a tíz decimális számjegy, valamint az A, B, C, D, E és F.

Az értékeket jelölő számokat számjegyek sorozataként írjuk le. Egy k+1 számjeggyel tartalmazó számot következő alakban írjuk fel:

$$a_k a_{k-1} a_{k-2} \ldots a_0$$

ahol a_i a szám számjegyeit jelöli.

Ehhez a számhoz (jelöléshez) tartozó számértékeket a következő képlet alapján kapjuk meg:

$$a_k b^k + a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + \ldots + a_0 b^0$$

Ahol:

- b számrendszer alapját,
- a_i a szám számjegyeit jelöli.

Tehát például a tízes számrendszerbeli 4357 szám a következő értéket jelöli $(\mathbf{4}\times10^3) + (\mathbf{3}\times10^2) + (\mathbf{5}\times10^1) + (\mathbf{7}\times10^0)$. Fontos látni, hogy ugyanaz a szám (számjegysorozat) különböző számrendszerekben más és más értéket jelöl. Ezért a számok használatakor – amikor az nem egyértelmű a szövegkörnyezetből – meg kell adnunk a számábrázolás alapját. Ezt a szám után indexként szoktuk jelölni. Tehát helyesen a tízes számrendszerbeli 4357 pontos jelölése: 4357_{10}.

Az előző példánkat folytatva a hexadecimális számrendszerben a 4357 – helyesen jelölve tehát a 4357_{16} – a következő értéket jelenti: $4357_{16} = (\mathbf{4}\times16^3) + (\mathbf{3}\times16^2) + (\mathbf{5}\times16^1) + (\mathbf{7}\times16^0) = 17232_{10}$.

A különböző számrendszerbeli számok átváltásával a későbbiekben részletesen is foglalkozunk.

2.2. Aritmetikai műveletek végrehajtása

A helyiértékes jelölés egyik legfontosabb előnye, hogy használatával könnyű az egyes aritmetikai műveletek elvégzése. Az egyes műveletek eredményét elég az egyes számjegyeken végrehajtott művelet esetén definiálni, majd ez alapján – néhány egyszerű szabály figyelembevételével – azok elvégezhetőek több számjegyet tartalmazó számokon is. A tízes alapú számrendszerben használható összeadó táblát a 2.1. ábra mutatja. Két számjegy összegét (az ábrán a 3-at és a 6-ot jelöltük) a táblán egyszerűen kikereshetjük. Ha a számjegyek összege egy számjegyen nem ábrázolható, akkor az eredmény két számjegyű lesz.

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

2.1. ábra Tízes számrendszerben használható összeadó tábla

Több számjegyű számok összeadása esetén a műveletet számjegyenként végezhetjük el. Amennyiben az eredmény nem ábrázolható egy számjegyen, úgy az eredmény magasabb helyiértékű számjegyét a következő számjegyeken végrehajtott művelet kiszámításakor kell figyelembe venni. Ekkor az eggyel nagyobb helyiértékre ún. átvitel keletkezik. Ezt jól láthatjuk a következő példán, ahol két bináris számot adunk össze: 1101101_2-t és 10110_2-t. Az átvitelt a legfelső sorban jelöltük.

1	*1*	*1*	*1*	*1*			
	1	**1**	**0**	**1**	**1**	**0**	**1**
+			**1**	**0**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Szorzás esetén a művelet végrehajtását lépésekre bontjuk szét. A szorzandót a szorzó számjegyeivel egyenként szorozzuk meg úgy, hogy a számjegyenként kapott szorzás eredményét (részletszorzatot) mindig egy helyiér-

tékkel arrébb toljuk, amely eltolás megfelel a számrendszer alapjával történő szorzásnak. A végén az eltolt értékeket összeadjuk.

			1	1	0	1	x	1	0	1
			1	1	0	1				
		0	0	0	0					
	1	1	0	1						
1	0	0	0	0	0	1				

A számítógépek működése és egyszerű megvalósítása szempontjából igen fontos kiemelni, hogy a bináris számrendszerben a számjegyeken végzett aritmetikai műveletek logikai (boolean) műveletekkel helyettesíthetőek. Vegyük szemügyre az összeadó táblát:

+	0	1
0	0 0	0 1
1	0 1	1 0

XOR	0	1
0	0	1
1	1	0

AND	0	1
0	0	0
1	0	1

A művelet alsó helyiértékű eredménye az XOR (kizáró vagy) művelettel, míg a felső helyiértékű eredménye az AND (és) logikai művelettel állítható elő.

Szorzótáblánál hasonlóan, a művelet eredményét megkaphatjuk AND (és) logikai művelettel:

*	0	1
0	0	0
1	0	1

AND	0	1
0	0	0
1	0	1

2.3. Számrendszerek közötti konvertálás

A számrendszerek közötti konvertálás egy adott számrendszerbeli szám másik számrendszerbeli alakjának meghatározását jelenti. A konvertálás legegyszerűbb módja a maradékos osztással történő átszámítás. A konvertálandó értéket az adott számrendszerben maradékos osztással elosztjuk a másik számrendszer alapszámával. A maradékokat egymás mellé írva megkapjuk az osztóként használt szám számrendszerében használható

alakját a konvertált számnak. Erre adunk egy példát a 42_{10} kettes számrendszerbeli alakjának kiszámításával:

10-es alap: **42**

	maradék	
42/2	**0**	*legkisebb helyi-értékű bit*
21/2	**1**	
10/2	**0**	
5/2	**1**	
2/2	**0**	
1		*legnagyobb helyi-értékű bit*

2-es alap: **101010**

A konverzió egyszerűbb, ha a konvertálandó és konvertált számrendszerek alapjai egymás hatványai. Ebben az esetben az egyes értékek adott számrendszerekben felírt számjegyei megfeleltethetők szisztematikusan egymásnak, és a konverziót elvégezhetjük számjegyenként (ha a konvertálandó szám számrendszerének alapja nagyobb a konvertált szám számrendszerének alapjánál), ill. a számjegyek csoportjain (ha a konvertálandó szám számrendszerének alapja kisebb a konvertált szám számrendszerének alapjánál):

16-os alap:	**1**	**F**	**6**	**7**
2-es alap:	**0001**	**1111**	**0110**	**0111**

2.4. Tört értékek ábrázolása, konverzió

Tört értékek ábrázolására továbbvihetjük az egész számok leírásakor megismert logikát. Ha egy ábrázolt számérték megadásakor bevezetjük az egyes helyiérték után a pont jelölést, majd megengedjük a további számjegyek használatát kapjuk a következő alakot:

$$a_k a_{k-1} a_{k-2} \ldots a_0 \, . \, a_{-1} a_{-2} \ldots a_{-n}$$

Ahol a_i a szám számjegyeit jelöli.

Ehhez a számhoz (jelöléshez) tartózó a számértékeket a következő képlet alapján kapjuk meg:

$$a_k b^k + a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + ... + a_0 b^0 + a_{-1} b^{-1} a_{-2} b^{-2} + a_{-3} b^{-3} + ... + a_{-n} b^{-n}$$

Ahol:

- b számrendszer alapját,
- a_i a szám számjegyeit jelöli.

A tört értékek ilyen módon történő ábrázolása hasonló előnyös tulajdonságokkal bír, mint az egész értékek helyiértékes leírása, tehát egyszerű az aritmetikai műveletek végrehajtása.

A tört ponttal történő számábrázolás számítógépekben történő alkalmazása kapcsán egy fontos problémába ütközünk. Bizonyos tört értékek csak ún. végtelen tört formában írhatók le – tízes számrendszerben ezeket nevezzük végtelen tizedes törteknek –, vagyis a pontos ábrázolásukhoz végtelen számú számjegy tárolására lenne szükség. Ez a számítógépekben lehetetlen, ezért az ilyen értékek pontos tárolását nem lehet megoldani. A számítógépek ezért ezen értékek helyett azok közelítő értékét tárolják.

A probléma abból adódik, hogy az egyes számrendszerekben más és más tört értékek nem ábrázolhatók pontosan véges számábrázolást feltételezve. Tehát lesznek olyan tört számok, melyeket egyik számrendszerből a másikba konvertálva, pontosan nem fogunk tudni tárolni, csak azok közelítő értékét.

Ez a probléma leggyakrabban a tízes számrendszerből kettes számrendszerbe történő konvertáláskor okoz problémát, hiszen a számítógépes alkalmazások általában a tízes számrendszert használják az adatok bevitelére és a kettes számrendszert az adatok tárolására.

és a kettes számrendszert az adatok tárolására.

3. Különböző típusú adatok tárolása a számítógépen

3.1. Adatok tárolása a számítógépen

A Neumann elven működő számítógépek bitsorozatokat, ill. bitsorozatok csoportjait (byte-ok sorozatát) tárolják minden típusú adat fizikai tárolásakor. A különböző típusú adatok, ill. azonos típusú adatok különböző adatábrázolást használó tárolása között abban van különbség, hogy hogyan történik a bináris jelsorozatok megfeleltetése az adott adattípus értékeinek. Az adatok számítógépes tárolásakor az aktuálisan használt adatábrázolás szabályai határozzák meg a számítógép által fizikailag tárol bináris jelsorozat (bináris értékek) és az ábrázolt adatok értékei közötti megfeleltetést.

Annak érdekében, hogy az egyes számítógépek közötti adathordozás egyszerű legyen, ill. a számítógépek hasonló módon működjenek, a különböző típusú adatok adatábrázolási szabályait egységesítették, általánosan elfogadott ajánlásokba vagy szabványokba foglalták. Ebben a fejezetben a legfontosabb, általánosan használt adatábrázolási szabvánnyokkal ismerkedünk meg.

3.2. Szöveges információ tárolása

A szöveget karakterek sorozataként tárolja a számítógép. A szövegek tárolásához az egyes karaktereket feleltetjük meg – a kiválasztott ábrázolási szabványtól függően – egy vagy két byte-os, számítógép által tárolt bináris kódnak.

Szövegek ábrázolására különböző szabványok léteznek:

- ASCII (American Standard Code for Information Interchange): a legkorábban kidolgozott és máig a legszélesebb körben használt szabvány. Várhatóan a Unicode terjedésével veszít a jelentőségéből.
- EBCDIC (Extended Binary Coded Decimal Interchange Code): Az IBM cég által kidolgozott, elsősorban nagyszámítógépes környezetben használt szövegábrázolás.
- Unicode: közelmúltban kidolgozott szabvány, mely figyelembe veszi a különböző nyelvek által használt különböző ábécéket, ill. az ábécékben definiált különböző karaktereket.

Az ASCII szövegábrázolás az Amerikai Szabványügyi Hivatal (American National Standards Institute) által elfogadott szabvány. Az eredeti ASCII kódtáblázat (
3.1. ábra) a karaktereket hét bites kódoknak felelteti meg, így összesen 128 karaktert lehetett kódolni. Az ASCII szabványt később kiterjesztették. Bevezették a nyolc bites karakterkódolást és az újonnan definiált további 128 karakter helyen már nem csak az angol ábécé betűit, hanem más ábécék betűit is lehetséges volt kódolni. Attól függően, hogy a 128 újonnan definiált karakter milyen ábécé (vagy ábécék) betűit kódolja különböző kiterjesztései, változatai jöttek létre az ASCII táblázatnak. Ennek az a hátránya, hogy ha olyan szöveget jelenítünk meg, amely tartalmaz 127-es kód fölötti kódokat, akkor a megjelenítés fog függeni, attól, hogy az ASCII melyik kiterjesztett táblázatát használjuk. Európában az ISO 8859-1 kódszámon szabványosított ún. Latin-1 ASCII kódtáblát használják a legtöbbször, mert ez tartalmazza a legtöbb nyugat-európai nyelvben használt karaktert. A magyar nyelvben szereplő összes betűt az ISO 8859–2 kódszámon szabványosított ún. Latin-2 ASCII kódtáblát tartalmazza.

MSD LSD	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	NUL	DLE	SP	0	@	P	1.	p
1	SOH	DC1	!	1	A	Q	a	W
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACJ	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

3.1. ábra Az ASCII kódtáblázat

Az EBCDIC (Extended Binary Coded Decimal Interchange Code) konverziós táblázatot az IBM dolgozta ki, és a mai napig is főként IBM, ill. IBM kompatibilis nagyszámítógépek körében használatos. Karakterkódok sajnos eltérnek az ASCII kódtáblázatban definiált kódoktól, ezért ha csak ASCII kódolású szövegek megjelenítésére alkalmas szoftver áll rendelkezésre az EBCDIC kódolású szövegeket konvertáló program segítségével át kell kódolni. Az EBCDIC nyolc bites kódokat használ.

A Unicode karakterkódolást egy non-profit szervezet, a Unicode Consortium dolgozta, ill. dolgozza ki. A szervezet a szöveges állományok egységes számítógépes ábrázolására szolgáló szabvány kidolgozása céljából jött létre. 1991 óta számos szabványt publikáltak, a legutóbbi, a Unicode 5.0 2006. júliusában jelent meg.

A Unicode kódolás megengedi a két vagy négy bájtos, tehát 16 vagy akár 32 bites karakter kódok használatát. A karakterek és szimbólumok egymásnak történő megfeleltetésének definiálására két különböző leírási formát (UTF – Unicode Transformation Format és UCS – Universal Character Set) és azokon belül különböző kódtáblák használatát engedi meg. A következő Unicode kódtáblák használtak:

- UTF-7 – hét bites karakterkódolást használó, ma már nem alkalmazott kódtábla.
- UTF-8 – nyolc bites, ill. változó hosszúságú kódokat megengedő táblázat, ez a leginkább kompatibilis az ASCII kódolással, ASCII Latin-I kódjai 0-tól 255-ig szerepelnek ebben a kódtáblában.
- UTF-EBCDIC – nyolc bites, ill. változó hosszúságú kódokat megengedő táblázat, ez a leginkább kompatibilis az EBCDIC kódolással.
- UCS-2 – 16 bites fix hosszúságú kódokat használó kódtáblázat, ma már nem alkalmazott.
- UTF-16 – 16 bit hosszúságú kódokat használó kódtáblázat.
- UCS-4 és UTF-32 – melyek lényegében azonosak –, 32-bites fix hosszúságú kódokat használó kódtáblázat.

Nem véletlenül beszéltünk korábban betűk és kódok megfeleltetéséről, a Unicode esetén pedig szimbólumok és kódok megfeleltetéséről. A betű elnevezés helyett itt helyesebb az egyes nyelvek által használt szimbólumokról beszélni, mert a nyelvek, ill. a Unicode Consortium által összegyűjtött több mint 10000 írásban használt jel tartalmazza pl. a kínai vagy a koreai képírás szimbólumait is.

A Unicode a köztudatba a 16-bites, 65536 karakter ábrázolására alkalmas kódrendszerként ment át, mint láttuk ma már ezen a fejlődési szakaszon túllépett.

3.3. Állóképek tárolása

Az állóképek számítógépes ábrázolásának megértéséhez először röviden meg kell ismerkednünk a képek számítógépen történő megjelenítésének logikájával.

A számítógép a képeket képpontokból (pixel – picture elemet) állítja össze. A képek négyszög alakúak, egymás alatt sorokba rendezett képpontokból állnak. A kép felbontását a benne szereplő képpontok száma határozzuk meg, ezt a függőleges és a vízszintes irányban külön szoktuk megadni. Például a 640×480 pontos felbontás azt jelenti, hogy az adott képen 480 sor van és minden sorban 640 képpont, vagyis összesen 307200 képpont.

A számítógép a képpontok színét – bizonyos esetekben más tulajdonságait is – tárolja. A képpontok színét úgy definiálja a számítógép, hogy a megadja, hogy a három alapszínből (vörös, zöld, kék) az egyes színkomponensek mennyire intenzívek az adott képponton. Modern számítógépeken általános, hogy az intenzitást színkomponensenként egy-egy byte adattal jellemzzük.

A képek tárolási szabványainak kidolgozásakor az egyik legfontosabb szempont a tömör ábrázolás volt. Ha kiszámoljuk, hogy egy viszonylag kicsi, 640X480 pontos felbontású kép 7 372 800 bit, vagyis 921 600 bájt, közel 1 Mbyte adat tárolását jelenti azonnal megértjük, hogy miért olyan fontos a tömör ábrázolás.

A képek számítógépen történő tárolására két egymástól eltérő módszer létezik:

- *Bittérképes tárolás*, amely közvetlenül tükrözi a fent leírt, megjelenítés során használt adatszerkezet felépítését. Több szabvány is van, amelyek abban különböznek, hogy milyen tömörítő módszert használnak a képek kisebb tárhelyen történő ábrázolására.
- *Vektorgrafikus ábrázolás* vagy képobjektumok tárolása. Ebben az esetben a képet grafikai objektumok csoportjának tekintjük, és azokból állítjuk össze. A grafikai objektumok különböző geometriai alakzatok, vonalak és görbék stb., melyek formáját és megjelenését valamilyen matematikai függvénnyel, ill. annak paramétereivel pontosan meg tudunk határozni. A kép leírása nem lesz más, mint a képen levő alakza-

tok (grafikai objektumok) típusának, helyének, ill. az objektumok paramétereinek definíciója.

A bittérképes ábrázolást gyakrabban használják, mert a hagyományos formában meglevő képeket digitalizálni – vagyis a számítógép által kezelhető adatokká konvertálni – képes bemeneti eszközökről (pl. szkenner (lapolvasó), fényképezőgép stb.) a számítógépbe ilyen formátumban érkeznek az adatok. A bittérképes ábrázolás előnye, hogy közvetlenül megjeleníthető, a megjelenítése viszonylag kevés adatfeldolgozást igényel. Hátránya a viszonylag nagy tárigény.

A bittérképes képek esetén a kép minőségét meghatározó két paraméter:

- *a kép felbontása*, vagyis az, hogy a képet a két dimenzióban hány képpontra (pixelre) bontották, valamint
- *a kép színmélysége*, vagyis az, hogy egy képpont színének leírásához hány bitet használunk.

A színmélység vagy a felbontás növelésével jobb minőségű (valósághűbb, ill. részletgazdagabb) képet kapok, viszont a képek tárolásához szükséges tárhely négyzetesen növekszik.

Bittérképes képek tárolására számos szabvány létezik:

- **GIF (Graphics Interchange Format)**
  - 1987-től kezdődően fejlesztette ki a CompuServe, azóta szabványosították.
  - A GIF formátum a képen levő képpontok színének meghatározására egy nyolc bites kódokat tartalmazó táblázatot, ún. palettát használ. Így legfeljebb 256 szín szerepelhet egy képen. A nyolc bites kódokhoz rendelt színek meghatározása 24 biten történik, nyolc bit alapszínenként. Az adatokat ún. LZW (Lempel-Zif-Welch) algoritmussal tömöríti, ami veszteségmentes tömörítés. A használható színek limitált száma (256) a legszigorúbb korlát a képek tárolásakor.
  - A GIF89a szabvány lehetővé teszi több, egymást követő kép tárolását, ill. megjelenítését rögzített időközönként, így lehet animációkat, ill. egyszerű mozgóképeket tárolni.
  - Főként vonalas ábrákhoz, rajzokhoz, művészi és nagyméretű azonos színű területeket tartalmazó képekhez ajánlott.
  - A GIF képeket tároló file-ok kiterjesztése általában „.gif”.

- **JPEG (Joint Photographers Expert Group)**
  - Sok színt tartalmazó, részletgazdag fényképek és festményekhez tárolásához ajánlott formátum.
  - Bár létezik a szabványnak veszteségmentes tömörítést lehetővé tevő változata, a JPEG formátum veszteséges tömörítési algoritmust alkalmaz, amely a megjelenítéskor kevéssé látható adatokat eldobva csökkenti a tároláshoz szükséges adatok méretét. A tömörítés mértékét a felhasználó állíthatja. Az adattömörítés mértékétől függően csökkenhet a kép minősége, ami elsősorban az éles vonalak és kontúrok elmosódásában látható. Ha a képet egymás után többször szerkesztjük, akkor az egyes mentések, ill. konvertálások ronthatják a kép minőségét.
  - A JPEG szabványt kidolgozó JPEG Group kidolgozta a szabvány számos hiányosságát (pl. konvertálás alkalmával történő minőségromlás) részben kiküszöbölő JPEG 2000 szabványt, mely a JPEG-nél hatékonyabb veszteségmentes és veszteséges tömörítési algoritmusok alkalmazását teszi lehetővé.
  - A JPEG képeket tároló file-ok kiterjesztése „.jpeg”, „.jpg”, ritkábban a „.jpe”.
- **További bittérképes formátumok:**
  - TIFF (Tag Image File Format). Képek jó minőségű tárolását lehetővé tevő formátum. Megengedi a tömörítést, a különböző tömörítési algoritmusok használatát lehetővé tevő változatai egymás után jelentek meg 1986 és 1992 között. A szabvány megengedi kb. ötven előre definiált ún. tag (címke) használatát a kép leírásában, ezért viszonylag körülményes a megjelenítő szoftver készítése. Az ilyen formátumú képeket tároló file-ok kiterjesztése általában „.tif”.
  - BMP (BitMaPped, bittérképes). A formátumot a Microsoft definiálta megjelenítő eszközök adatábrázolásától független módon lehessen képeket tárolni. Tömörítést nem használ ezért a BMP képek mérete viszonylag nagy. Az ilyen formátumú képeket tároló file-ok kiterjesztése általában „.bmp”.
  - PNG: (Portable Network Graphics). A PNG szabvány különböző változatai 1996 és 32004 között jelentek meg. A kidolgozását motiváló egyik legfontosabb tényező az volt, hogy a képeket kezelő alkalmazások fejlesztőinek a GIF és a JPEG szabványok használatáért jogdíjat kellett fizetni. A PNG szabvány ezzel szemben egy sza-

badon felhasználható szabvány. Ennek megfelelően az internetes alkalmazások előszeretettel alkalmazzák. Viszonylag fejlett veszteségmentes tömörítést alkalmazó szabvány. A GIFF formátummal összehasonlítva általában nagyobb tömörítést tesz lehetővé, szélesebb lehetőségeket kínál az áttetsző képi elemek definiálására, azonban nem támogatja az animált képek megjelenítését. Az ilyen formátumú képeket tároló file-ok kiterjesztése általában „.png”.

Mint korábban említettük, a bittérképes leírás mellett a képek tárolásának másik módszere a vektorgrafikus ábrázolás. A vektorgrafikus képek esetén az adatok tárolásának formája szorosan kötődik egy-egy képszerkesztő környezethez, melyben definiálni tudjuk a képek elemeit, ill. a képelemekből össze tudjuk állítani a képet azok egymáshoz viszonyított helyének és helyzetének meghatározásával. A képek készítésekor használható képi elemek választéka nagymértékben függ a használt képszerkesztő környezettől.

Általában igaz, hogy a vektorgrafikus formátumok jóval tömörebb leírását teszik lehetővé a képi információknak, mint a bittérképes ábrázolások. A vektorgrafikus képek használatának másik előnye, hogy azokat könnyű méretezni, konvertálni (pl. elforgatni) a képminőség romlása nélkül, ami a bittérképesek képekkel esetén gyakran nem igaz. Hátránya, hogy mesterségesen, képszerkesztő eszközökkel előállított képeket tudunk csak ilyen formában tárolni. Maga a vektorgrafikus leírási mód is a háromdimenziós modellezéssel kapcsolatos kutatások eredményeként jelent meg.

Ma már számos olyan megoldás létezik, melyek segítségével valósághű mesterséges képeket tudunk előállítani. Ezeket a módszereket a modellezésen kívül az animációs filmek, az ún. virtuális valóság környezetek készítésekor, valamint a játékprogramok írásakor intenzíven használják.

Gyakran használt vektorgrafikus formátumok:

- **.EPS**: Encapsulated PostScript (EPS) formátum. Ezt a formátumot az Adobe cég – mely számos kép és dokumentum szerkesztő, ill. megjelenítő programot készít –, fejlesztette ki a 80-as években. Ezt az állományformátumot a grafikus programok legtöbbje ismeri.
- **.CDR, .CDT**: CorelDRAW, ill. CorelDRAW Template formátumok. A CorelDRAW program vektorgrafikus leíró fájlformátuma. Tartalmazhat beillesztett bittérképes objektumokat is.
- **.AI**: az Adobe Illustrator program által használt fájlformátum.

- **.WMF**: Windows Metafile. A Windows alkalmazások, elsősorban az Office programjai közös használatára szolgáló grafikus fájlformátum. Microsoft-fejlesztés. Vektoros és bittérképes leírást is tartalmazhat.
- **.EMF**: Enhanced (továbbfejlesztett) Windows Metafile formátum.
- **.DXG, .DWG, .DXF**: Az AutoCAD modellező program által használt fájlformátumok.

Számos további vektorgrafikus formátum létezik, csak felsorolásszerűen néhány: SVG, XAML, CGM, VML, Xgl, STEP, IPA, PRC, STL.

A vektorgrafikus képekből a képek megjelenítésekor minden esetben elő kell állítani egy bittérképet, mert a ma használt megjelenítő eszközök csak bittérképes képeket tudnak közvetlenül megjeleníteni.

A képi információk tárolásáról szólva feltétlenül kell szólni a dokumentumok számítógépes tárolásáról. A dokumentumok ma már általában számítógép segítségével készülnek, valamilyen dokumentum szerkesztő alkalmazás felhasználásával. Az alkalmazások képesek kinyomtatni és megjeleníteni a dokumentumot, magát a tartalmát viszont nem bittérképként tárolják, hanem valamilyen dokumentum leíró nyelv (page description language, PDL) szerint. Ezek a leíró formátumok általában alkalmazás függőek.

Ma két széles körben használt dokumentum leíró nyelv létezik, melyek szabványa szabadon használható, így széles körben elterjedt:

- PostScript (.ps) és a
- PDF, Portable Document Format (.pdf), melyet az Adobe cég által fejlesztett Adobe Acrobat programcsalád használ.

Mindkét szabvány lényegében egy teljes programozási nyelvet definiál a dokumentum tartalmának leírására. Mindkét nyelvhez létezik megjelenítő szoftver, ún. interpreter, mely a nyelvek utasításaiban definiált képmegjelenítő műveleteket végre tudja hajtani. A dokumentumok, ill. a dokumentum leírások tartalmazhatnak tömörített, ill. tömörítetlen bittérképes elemeket is.

3.4. Számítógépekben használt adattípusok

A számítógépes programokban változókat használunk a számítógépprogram által kezelt adatok tárolására, ill. az adatokon végzett műveletek definiálására. A programban használt változókat különböző adattípusokba soroljuk. Az egyes programozási nyelvek különböző típusú adatok haszná-

latát teszik lehetővé. A programozási nyelvek az egyes adattípusokhoz megadják, hogy milyen műveleteket lehet rajtuk végrehajtani.

A programnyelvek által leggyakrabban megengedett egyszerű adattípusok:

- Boolean: kétértékű változó, amelynek értéke lehet igaz vagy hamis.
- Karakter (char): változó, amelynek értéke valamilyen ábécében definiált betű, ill. számjegy.
- Egész szám (integer): pozitív vagy negatív egész értékek tárolására alkalmas változó.
- Valós szám (real): pozitív vagy negatív, tört vagy egész értékek tárolására alkalmas változó.
- Felsorolás típus (enumerated): Felhasználó, ill. a programozó által meghatározott adatértékek tárolására szolgáló változó.

4. Egész értékek tárolása a számítógépen

4.1. Előjeles bináris és bináris és binárisan kódolt decimális ábrázolás

Az egész értékek tárolására több lehetséges ábrázolási mód létezik:

- Bináris ábrázolás: Előjel nélküli egész szám esetén az ábrázolt érték kettes számrendszerben felírt alakját tároljuk. Előjeles bináris szám tárolása esetén az első (legmagasabb) bitet fenntartjuk az előjel tárolására, a szám abszolút értékét a fennmaradó biteken tároljuk, az abszolút érték kettes számrendszerben felírt alakjában.
- BCD (Binary Coded Decimal, binárisan kódolt decimális) ábrázolás a számok decimális egész alakjának közvetlen bináris tárolása. A decimális alakban felírt szám számjegyeit egyenként, egymás után tároljuk. Minden számjegy tárolására a számjegy kettes számrendszerben felírt alakját tároljuk. Így mindegyik számjegy tárolására 4 bitet kell felhasználnunk.

Az egész értékek tárolására mutat példát a 4.1 ábra.

Decimális érték	Bináris ábrázolás	BCD ábrázolás
68	= 0100 0100	= 0110 1000
	= $2^6 + 2^2 = 6^4 + 4 =$ ***68***	= $2^2 + 2^1 =$ ***6*** $2^3 =$ ***8***
99 (legnagyobb 8 biten ábrázolható szám BCD kódolással)	= 0110 0011	= 1001 1001
	= $2^6 + 2^5 + 2^1 + 2^0 =$ = 64 + 32 + 2 + 1 = ***99***	= $2^3 + 2^0$ $2^3 + 2^0$ = ***9*** ***9***
255 (legnagyobb 8 biten ábrázolható szám bináris kódolással)	= 1111 1111	= 0010 0101 0101
	= $2^8 - 1 =$ ***255***	= 2^1 $2^2 + 2^0$ 2^2+2^0 = ***2*** ***5*** ***5***

4.1. ábra Egész számok tárolása bináris és BCD formátumban

A bináris ábrázolás előnye, hogy nagyobb értéket képes tárolni adott számú biten (helyiértéken) és a számolási műveletek elvégzése egyszerűbb, mint BCD ábrázolás esetén. A BCD ábrázolás gyakran használatos olyan vállalati és pénzügyi rendszereknél, ahol a tízes számrendszerbeli tört számok pontos tárolása elengedhetetlen.

Előjeles bináris, ill. BCD tárolás esetén az aritmetikai műveleteket megvalósító, processzor által végrehajtandó lépések összetettek és bonyolultak ahhoz, hogy őket egyszerű hardver hajtsa végre. Problémát okoz, hogy a műveletek operandusainak előjelétől függően más és más módon kell végrehajtani egy adott aritmetikai műveletet. Egy példát a 4.2. ábra mutat. Az első esetben egyszerűen a két szám abszolút értékét kell összeadnom, hogy megkapjam az eredményt. Ha negatív szám is van az összeadandók között, akkor először az eredmény előjelét kell meghatároznom, az összeadandók előjele és abszolút értékük nagysága alapján. Két eset lehetséges:

- Nagyobb abszolút értékű számhoz adok egy kisebb abszolút értékű negatív számot. Ekkor az eredmény pozitív lesz és a nagyobb abszolút értékűből kell a kisebbet kivonni.
- Kisebb abszolút értékű számhoz adok egy nagyobb abszolút értékű negatív számot. Ekkor az eredmény negatív lesz és megint a nagyobb abszolút értékűből kell a kisebbet kivonni.

További problémát jelent bináris számábrázolás esetén az, hogy két kód is megfelel a 0 értéknek (+0 és −0), valamint kezelni kell az egyes helyiértékeken elvégzett műveletek elvégzésekor keletkező átvitelt.

Összeadás: két pozitív szám	**Összeadás: negatív és pozitív szám**		
4 +2 6	4 -2 2	2 -4 -2	12 -4 8

4.2. ábra Egész számok összeadása előjeles ábrázolás esetén

A számítógépek a gyakorlatban az előjeles bináris ábrázolást adatok tárolására igen ritkán alkalmazzák az aritmetikai műveletek végrehajtásának bo-

nyolultsága miatt. Hasonló okok miatt a BCD ábrázolást is csak azokban a rendszerekben használják, ahol a tízes számrendszerbeli tört számok pontos tárolása elengedhetetlen.

4.2. Egyes bináris komplemens ábrázolás

Az előjeles bináris ábrázolás helyett az ún. bináris komplemens ábrázolásokat alkalmazzák a számítógépek. Két változatát használják, az egyes és kettes bináris komplemens ábrázolást. Mindkét esetben a bináris kódokat szisztematikusan rendeljük az egész értékekhez, úgy hogy az aritmetikai alapműveleteket – elsősorban az összeadást – az értékekhez rendelt kódok egyszerű feldolgozásával lehessen elvégezni.

Komplemens számításnak azt a műveletet nevezzük, ha egy adott értéket kivonunk egy ún. alapszámból. Az alapszámot a komplemens számítás alapjának nevezünk. Komplemens ábrázolás esetén minden értéknek a komplemens számítás eredényeként kapott értéket feleltetjük meg.

Az egyes komplemens számábrázolás esetén a pozitív értékek ábrázolásakor az értékek kettes számrendszerbeli alakját használjuk. A negatív értékek kódjának megadásakor a komplemens számítást használjuk. A komplemens számítás alapja az a szám, amelynek minden helyiértékén az adott számrendszerben használt legnagyobb számjegy áll. Ez egyel kisebb, mint az adott helyiértéken ábrázolható különböző értékek száma. Emiatt nevezik az egyes komplemens ábrázolást ún. csökkentett alapú komplemens ábrázolásnak.

Nyolcbites bináris számábrázolás esetén ez a 11111111. Komplemens ábrázolást lehet tízes számrendszerben is használni. Ha nyolc digites ábrázolást használunk, akkor a komplemens számítás alapja a 99999999 lesz.

A komplemens számítás szabályai szerint a negatív értékek ábrázolására használt kódokat úgy kell kiszámítani, hogy az adott számrendszer szabályait figyelembe véve a komplemens számítás alapjához hozzáadjuk az ábrázolandó értéket, vagyis a komplemens számítás alapjából kivonjuk az ábrázolandó negatív érték abszolút értékét. Ezt a művelet bináris ábrázolás esetén igen egyszerűen végre tudjuk hajtani, ha megnézzük az egybites összeadó, ill. kivonó táblát, és észrevesszük, hogy a számolás során sehol sem keletkezik maradék. Ha az ábrázolt érték abszolút értékének a kettes számrendszerben felírt alakjában az összes bitet invertáljuk, megkapjuk a negatív érték egyes komplemens alakját.

Az ábrázolás sziszematikus, a számegyenesen egymás után következő értékeknek megfelelő kódok a „kódtérben” is egymás mellett vannak. A

használt kódok alapján egyes értékek előjelét az első bit alapján meg tudjuk határozni:

- a 0-val kezdődő számok pozitívak,
- az 1-el kezdődő számok negatívak.

A nulla értéknek két kódot feleltet meg az ábrázolás (−0, +0), ezt a számítások elvégzésekor figyelembe kell venni.

A nyolc bites egyes bináris komplemens ábrázolás szabályait mutatja 4.3. ábra.

	Negatív értékek		Pozitív értékek	
Ábrázolási mód	Egyes komplemens alak		A szám kettes számrendszerbeli alakja	
Ábrázolható decimális értékek	**-127_{10}**	**-0_{10}**	**$+0_{10}$**	**127_{10}**
Ábrázolt alak számítása	Abszolút érték bitenkénti invertálása		-	
Példa	**10000000**	**11111111**	**00000000**	**01111111**

4.3. ábra Nyolc bites egyes bináris komplemens ábrázolás

4.3. Tízes decimális és kettes bináris komplemens ábrázolás

A kettes komplemens számábrázolás esetén – hasonlóan az egyes komplemens számábrázoláshoz – a pozitív értékek ábrázolásakor az értékek kettes számrendszerbeli alakját használjuk. A negatív értékek kódjának megadásakor a komplemens számítást használjuk. A komplemens számítás alapja eggyel nagyobb, mint az egyes komplemens számábrázolás esetén, tehát megegyezik az adott helyiértéken ábrázolható különböző értékek számával.

Nyolcbites bináris számábrázolás esetén ez a 1000000000. Komplemens ábrázolást lehet tízes számrendszerben is használni. Ha nyolc digites ábrázolást használunk, akkor a komplemens számítás alapja ugyancsak az 1000000000 lesz, de természetesen 10-es számrendszerben értelmezve.

A komplemens számítás szabályai szerint a negatív értékek ábrázolására használt kódokat megint csak úgy kell kiszámítani, hogy az adott számrendszer szabályait figyelembe véve a komplemens számítás alapjához

hozzáadjuk az ábrázolandó értéket, vagyis a komplemens számítás alapjából kivonjuk az ábrázolandó negatív érték abszolút értékét.

Ha tudjuk, hogy az egyes komplemens ábrázolás esetén milyen egyszerűen lehetett a komplemens számítást elvégezni, és a komplemens számítás alapja a kettes komplemens számítás esetén pontosan eggyel nagyobb, mint az egyes komplemens ábrázolás esetén, akkor egy könnyen végrehajtható módszert kapunk a negatív értékeknek megfeleltethető kódok kiszámítására: kiszámítjuk az egyes komplemens alakot egy egyszerű invertálással és a kapott értékhez hozzáadunk egyet.

A számábrázolás ebben az esetben is sziszematikus, a számegyenesen egymás után következő értékeknek megfelelő kódok a „kódtérben" is egymás mellett vannak. A használt kódok alapján egyes értékek előjelét az első bit alapján meg tudom határozni:

- a 0-val kezdődő számok pozitívak,
- az 1-el kezdődő számok negatívak.

A nulla értéknek a kettes komplemens ábrázolás csak egy kódot feleltet meg, tehát egyszerűbb lesz az aritmetikai műveletek végrehajtása, ennek az ára viszont a valamivel bonyolultabb komplemens számítás.

A nyolc bites kettes bináris komplemens ábrázolás szabályait mutatja a 4.4. ábra.

	Negatív értékek		Pozitív értékek	
Ábrázolási mód	Kettes komplemens alak		A szám kettes számrendszerbeli alakja	
Ábrázolható decimális értékek	**-128_{10}**	**-1_{10}**	**$+0_{10}$**	**127_{10}**
Ábrázolt alak számítása	Abszolút érték bitenkénti invertálása, majd 1 hozzáadása		-	
Példa	**10000000**	**11111111**	**00000000**	**01111111**

4.4. ábra Nyolc bites kettes bináris komplemens ábrázolás

5. Lebegőpontos értékek tárolása a számítógépen

5.1. Számok normál alakja

A tört értékek tárolásához meg kell ismernünk a számok normál alakját.

Egy pozitív valós szám normál alakjának meghatározásakor a számot kéttényezős szorzat alakban írjuk fel. A szorzandó egy nulla és egy közé eső szám, a szorzó pedig a számrendszer alapjának valamilyen egész kitevős hatványa. Pl. az 12345_{10} szám normál alakja: 0.12345×10^5. A normál alakot felírhatjuk úgy is, hogy a szorzandó nem nulla és egy közé, hanem egy és a számrendszer alapja közé eső szám. Ebben az esetben a szorzandó az előző forma szorzandója lesz megszorozva a számrendszer alapjával, a szorzó kitevője pedig eggyel kisebb lesz, mint az előző forma szerinti kitevő.

A negatív valós számok normál alakjának számításakor a szám abszolút értékének normál alakját számítjuk ki, és a szorzandót (a szorzat első tényezőjét) megszorozzuk mínusz eggyel.

Megjegyezzük, hogy a 0 értéknek nincs normál alakja.

A számok normál alakjának tárolása esetén négy adatelemet kell egy valós szám (valós érték) tárolásához kezelni:

- előjel (a példánkban „+”, hiszen pontosan írva a normál alak első tényezője: +0.12345);
- érték, más néven mantissza (példánkban 12345);
- kitevő, más néven karakterisztika előjele (példánkban „+”, hiszen pontosan írva a normál alak második tényezője: 10^{+5});
- kitevő, más néven karakterisztika abszolút értéke (példánkban 5).

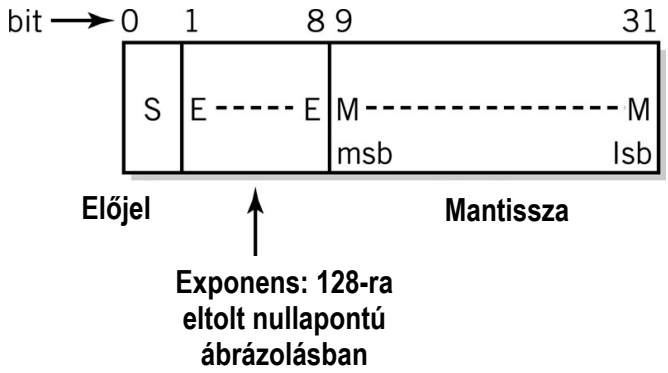
A normál alak értelmezéséhez, vagyis ahhoz, hogy megtudjuk, hogy milyen értéket kódol egy lebegőpontos szám, a fenti adatokon kívül még a következő – számábrázolástól függő – információra van szükség:

- kitevő alapja,
- tizedesvessző (ill. más számrendszer esetén a törtrész) helye a mantisszsa által jelölt érték értelmezéséhez.

5.2. Lebegőpontos számábrázolás

Lebegőpontos számábrázolás esetén a számok normál alakját tároljuk.

A tároláshoz a korábban említett négy adatelemet kell eltárolnunk. Az adatelemek tárolásának logikáját bináris tárolóeszköz esetén az 5.1. ábra mutatja.



5.1. ábra Lebegőpontos számábrázolás esetén az adatelemek tárolásának logikája

A tárolt szám előjelét az első biten tároljuk.

Ezután a kitevő (karakterisztika) értékét és előjelét tároljuk. A kitevő tárolására szolgáló adatábrázolást ún. eltolt nullapontú ábrázolásnak nevezzük. Általában azonos számú negatív és pozitív kitevő értéket szeretnénk tárolni, ezért nem a kitevőt magát, hanem annak adott alapszámmal megnövelt értékét tároljuk. Az alapszám az adott lebegőpontos ábrázolásban a legkisebb (negatív) ábrázolható kitevő abszolút értéke lesz. Ez nyolc bites kitevő ábrázolás esetén 127 vagy 128 lesz. Vegyük észre, hogy az ábrázolás onnan kapta a nevét, hogy a legkisebb kitevő értéknek a csupa nulla kódot felelteti meg.

A megmaradt biteken a mantissza abszolút értékét tároljuk bináris formában. Értékét a törtrész és az egészrész határoló „kettedespont” alapján tudjuk megmondani.

5.3. Műveletek végrehajtása lebegőpontos számokkal

Lebegőpontos számokkal végzett aritmetikai műveleteket a processzor a tárolt adatelemeken (mantisszán, kitevőn, ill. azok előjelein) végzett aritmetikai műveletek sorozataként hajtja végre.

A matematika szabályai szerint a hatványszorzatok összeadásakor az összeadandókat azonos kitevőre kell hozni. Két lebegőpontos szám összeadásakor a processzornak elő kell állítani a számok azonos kitevőjű alakját, ahol az egyes szorzók (kitevők, karakterisztikák) azonosak. Az összeadás ezután viszonylag egyszerű, hiszen a két szám mantisszáját kell összeadni.

Szorzás esetén a mantisszákat össze kell szorozni – természetesen a helyiértékek figyelembe vételével –, a kitevőket pedig össze kell adni.

Láthatjuk, hogy a lebegőpontos számokon végzett aritmetikai műveletek így végrehajthatók, több egyszerű aritmetikai műveletet sorozataként. A lebegőpontos számokon végzett aritmetikai utasítások gyors végrehajtása érdekében processzorok általában külön erre a célra szolgáló részegységet tartalmaznak, melyek áramkörei lehetővé teszik a lebegőpontos számokon végzett műveletek optimális végrehajtását.

5.4. Gyakorlatban használt lebegőpontos számábrázolási szabványok

A számítógépekben leggyakrabban alkalmazott számábrázolási szabvány az IEEE 754/1985 szabvány.

Az IEEE 754/1985 szabvány szerint négyféle lebegőpontos számformátum létezik, melyek csak a számábrázolás bitszélességében (a számok tárolására használt bitek számában) térnek el egymástól:

- short real (32 bit),
- long real (64 bit),
- temporary real (80 bit),
- quad real (128 bit).

Az IEEE 754/1985 szabvány szerint a lebegőpontos szám mantisszájának ún. normalizált explicit bites, alakját tároljuk. Ekkor a mantissza értékét (m) úgy kapjuk meg, ha a mantissza helyén tárolt értéket (F) egy egyes és a törtpont után írnánk, tehát m = 1.F. Az IEEE 754/1985 szabvány szerint

a mantisszában tárolt értéket akkor kell így értelmezni, ha a kitevő értéke nem nulla. Ha a kitevő értéke nulla, akkor a mantissza értéke így számítható: m = 0.F.

A szabvány külön értéket feleltet meg a 0 (ill. a +0 és −0) értéknek, valamint lehetőséget ad a végtelen érték, ill. a hiba események kódolására. Ezen értékekhez tartozó kódokat az 5.2. ábra mutatja.

Kódolt érték	Kitevő	Mantissza
Nulla érték	0	0
De-normalizált érték (0.F)	0	nem nulla
Normalizált érték (1.F)	1 to $2^e - 2$	bármi
Végtelen	$2^e - 1$	0
Nem számérték (pl. hiba)	$2^e - 1$	nem nulla

5.2. ábra IEEE 754/1985 szabvány szerint a lebegőpontos számábrázolás
(*e*: a kitevő ábrázolásának bitszélessége)

Az IEEE 754/1985 szabvány szerint négyféle lebegőpontos számformátum esetén az egyes adatelemek ábrázolására szolgáló adatmezők szélessége, ill. az eltolt nulla pontú ábrázolás alapértéke:

- short real (32 bit)
  Előjel: 1 bit, kitevő: 8 bit, mantissza: 23 bit, eltolás alapértéke: 127.
  Ábrázolható számtartomány: $8{,}43 \cdot 10^{-37} < |N| < 3{,}37 \cdot 10^{38}$.

- long real (64 bit)
  Előjel: 1 bit, kitevő: 11 bit, mantissza: 52 bit, eltolás alapértéke: 1023.
  Ábrázolható számtartomány: $4{,}19 \cdot 10^{-307} < |N| < 1{,}67 \cdot 10^{308}$.

- temporary real (80 bit)
  Előjel: 1 bit, kitevő:15 bit, mantissza: 64 bit, eltolás alapértéke: 16383
  Ábrázolható számtartomány: $3{,}4 \cdot 10^{-4932} < |N| < 1{,}2 \cdot 10^{4932}$.

- quad real (128 bit)
  A mantissza explicit bites egyes normalizált: m = 1.F
  Előjel: 1 bit, kitevő:15 bit, mantissza: 112 bit, eltolás alapértéke: 16383
  Ábrázolható számtartomány: $3{,}4 \cdot 10^{-4932} < |N| < 1{,}2 \cdot 10^{4932}$.

III. RÉSZ

SZÁMÍTÓGÉPEK MŰKÖDÉSE ÉS FELÉPÍTÉSE

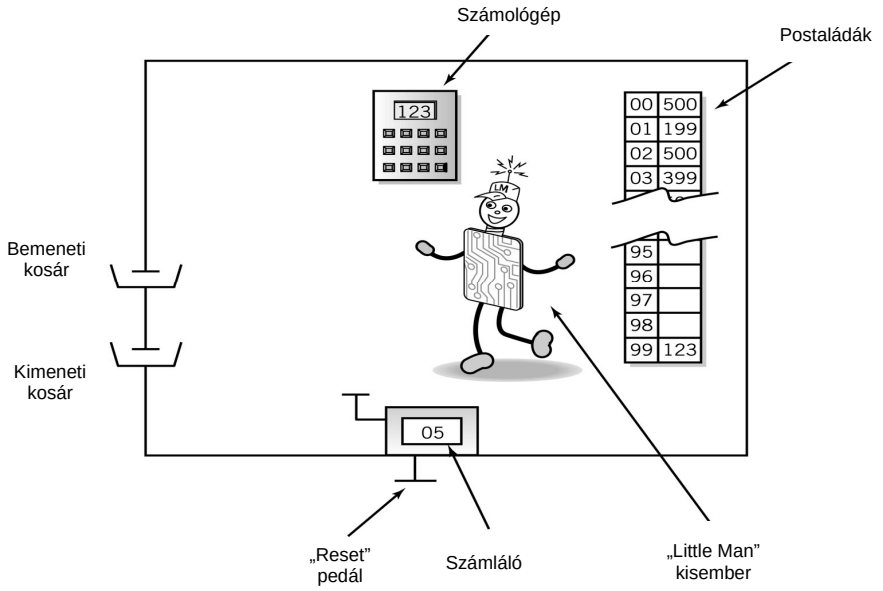
6. A „Little Man Computer” modell

A számítógépek működésének bemutatására az ún. „Little Man Computer” (LMC) modellt fogjuk használni. Az LMC modell nagyon leegyszerűsíti a számítógép működését, így segítségével nagyon könnyen megérthető a számítógép működésének alapjai.

A modellt Dr. Stuart Madnick dolgozta ki a Massachusetts Institute of Technology-n 1965-ben. Azóta világszerte széles körben használják a számítógépek működésének leírásához és oktatásához. Az eredeti modellt igen sokan továbbfejlesztették – például létezik a mikroprogramozott számítógépek működését leíró ún. LMMS (Little Man Microcode Simulator) változata [Thad Crews, William Yurcik: LMMS: An 8-Bit Microcode Simulation of the Little Man Computer]. Maga az alapmodell ettől függetlenül változatlan formában használható a számítógépek működésének leírására, ami önmagában is elismerésre méltó teljesítmény, ha végiggondoljuk, milyen sokat fejlődtek a számítógépek 1965 óta.

6.1. A „Little Man Computer” modell

A „Little Man Computer” (LMC) modell részeit a 6.1. ábra mutatja.



6.1. ábra A „Little Man Computer” (LMC) modell elemei

A modell lelke a „Little Man” (magyarul emberke, vagy kisember), aki működtetni fogja a modellünket. A modellt működtető emberkénk használhat egy hagyományos számológépet, melybe három számjegyű decimális számokat tud begépelni. A számológép képes kivonni és összeadni, valamint a begépelt vagy a számítás eredményeként kapott értéket eltárolni.

A kisembernek a feladatai elvégzésére a rendelkezésére áll egy postafiók, amelyben 100 darab rekesz (levelesláda) van. A levelesládák 0 és 99 közötti számokkal vannak felcímkézve. Minden levelesládába egy cetlit lehet rakni, melyen rajta lehet egy három számjegyű decimális szám.

Van két kosár, egyiket bemeneti, másikat kimeneti kosárnak nevezzük. Ezekbe az emberke – a kosártól függően – beletehet, ill. kivehet egy-egy cetlit, melyen ugyancsak egy három számjegyű decimális szám lehet.

A modell tartalmaz még egy számlálót, ami 0 és 99 között tud számolni. A számlálón van egy pedál, ennek megnyomására a számlálóban tárolt szám értéke eggyel megnövekszik. A számlálóban tárolt értéket nullára állíthatjuk egy külső ún. „reset” (beállító) gomb megnyomásával.

6.2. Az LMC modell alap utasításai

A kisember képes utasításokat végrehajtani. Az LMC által végrehajtandó utasítások sorozatát a kisember működése előtt készítjük el. A kisember által végrehajtható legegyszerűbb utasítások listáját a 6.2. ábra mutatja.

Utasítás típusa	**Utasítás gépi kódja**	Utasítás jelentése
Aritmetikai	**1XX**	***Összeadás***
	2XX	***Kivonás***
Adat mozgató	**3XX**	***Tárolás***
	5XX	***Betöltés***
Input/Output	**901**	***Beolvasás (INPUT)***
(Be-/Kimeneti)	**902**	***Kiírás (OUTPUT)***
Vezérlő	000	***Leállás (COB)***

6.2. ábra A „Little Man Computer” legegyszerűbb alap utasításai

Az utasításokat különböző csoportokba oszthatjuk, így vannak:

- aritmetikai,
- adat mozgató,
- input/output (be-/kimeneti) és
- vezérlő

típusú utasítások.

Az aritmetikai utasításokkal a számítógépen tárolt adatokon számolási műveleteket tudunk végrehajtani, az adatmozgató műveletekkel adatokat tudunk a számítógép különböző adattárolói között mozgatni. Az LMC modellben adattárolásra a levelesláda, ill. a kézi számológép szolgál. Az input/output utasításokkal a számítógép ki és bemenetein – az LMC modellben a bemeneti és a kimeneti kosáron keresztül – tudunk adatokat kiírni, ill. beolvasni. A vezérlő utasítások az utasítások végrehajtásának menetét, ill. sorrendjét határozzák meg.

Minden utasításhoz hozzá van rendelve egy kód – az LMC esetén egy háromjegyű decimális szám –, ezeket a táblázat középső oszlopában láthatjuk. Ezt, az utasításokhoz tartozó kódot *gépi kódnak* nevezzük, mert ezeket az utasításkódokat valós számítógépekben a processzor közvetlenül tudja értelmezni és végrehajtani.

A gépi kódokat jellemzően két részre oszthatjuk, két darabra vághatjuk. Az első rész meghatározza a végrehajtandó műveletet, ezért „műveleti kódnak” nevezzük, míg a második része meghatároza azt, hogy a műveletet milyen – egész pontosan hol tárolt – adaton kell elvégezni. Ezt a részt operandusnak nevezzük. Az összeadás, kivonás, tárolás és betöltés esetén a 6.2. ábra középső oszlopában azért szerepel a gépi kód utasítás végén két X, mert ez azt jelenti, hogy ott bármilyen két számjegyű érték szerepelhet. Ez azért van így, mert az utolsó két számjegy ezekben az utasításkódokban az operandus helye, és itt határozzuk meg, hogy az adott műveletet melyik levelesládában tárolt adaton kell elvégeznünk. A 6.3. ábra az egyes LMC utasítások működésének pontos definícióját mutatja a műveleti kód és az operandus figyelembe vételével.

Utasítás gépi kódja	Végrehajtott művelet
1XX	*Összeadás:* ***A számológépben tárolt értékhez az XX-edik postaláda fiókban tárolt érték hozzáadása. Az eredmény a számológépben lesz.***
2XX	*Kivonás:* ***A számológépben tárolt értékből az XX-edik levelesláda fiókban tárolt érték levonása. Az eredmény a számológépben lesz.***
3XX	*Tárolás:* ***A számológépben tárolt érték elmentése a XX-edik levelesláda fiókba.***
5XX	*Betöltés:* ***A XX-edik levelesládában tárolt érték beírása a számológépe.***
901	*Beolvasás (INPUT):* ***A bemeneti kosárban levő érték betöltése a számológépbe.***
902	*Kiírás (OUTPUT):* ***A számológépben tárolt érték betétele a kimeneti kosárba.***
000	*Leállás:* ***A munka befejezése.***

6.3. ábra Az LMC legegyszerűbb alap utasításainak értelmezése

6.3. Az Assembly nyelv

Az LMC (ill. a számítógép) által végrehajtott programokat az előző fejezetben megismert, ill. azokhoz hasonló utasításokból állítjuk elő. Egy program – ebben a jelentésében – nem más, mint a számítógép által vég-

rehajtható utasítások sorozata. A programok definiálják, hogy milyen utasítássorozatot kell a számítógépnek végrehajtania egy-egy feladat megoldása érdekében. A programokat a számítógép használata, vagyis a program futtatása előtt állítjuk össze, és töltjük be a számítógép memóriájába. Az LMC esetén ez a programnak a levelesládákba történő betöltését jelenti. Az LMC esetén nem foglalkozunk azzal, hogy a programok hogyan kerülnek a levelesládába, elfogadjuk, hogy azok valamilyen módon az induláskor ott vannak. Hasonlóan nem foglalkozunk azzal, hogy a különböző adatok hogyan kerülnek a bemeneti kosárba, és hogyan távoznak a kimeneti kosárból.

A programokat készíthetjük úgy, hogy a gépi kódú utasításokat használjuk. Programfejlesztéskor, egy hosszabb program megírása esetén a gépi kód használata igen nehézkes lehet. Ezért a gépi kódú programok egyszerűbb fejlesztése érdekében minden gépi kódú utasításhoz hozzárendeltek egy könnyen értelmezhető szöveges rövidítést, mely sokkal könnyebben megérthető programszöveget eredményez. Ezeket a könnyen értelmezhető rövidítéseket ún. mnemonic-okat tartalmazó programozási nyelvet nevezzük *assembly nyelvnek*. Mivel minden ma használatos processzor más és más utasításkészlettel rendelkezik, emiatt az egyes processzorokhoz tartozó assembly nyelvek is különböznek. A legfontosabb közös tulajdonságuk, hogy az assembly nyelven megírt programok közvetlenül lefordíthatók gépi kódra, hiszen az assembly utasítások és a gépi kódú utasítások között mindig egy az egyben megfeleltetés létezik. Az assembly utasításokat tartalmazó programszöveget egy fordító program fogja a programfejlesztés befejezésekor gépi kódú utasításokká konvertálni.

Az LMC modellnek is van assembly nyelve. Az egyszerű utasítások assembly kódjait a 6.4. ábra tartalmazza.

Utasítás gépi kódja	Utasítás assembly kódja	Utasítás jelentése
1XX	**ADD XX**	***Összeadás***
2XX	**SUB XX**	***Kivonás***
3XX	**STO XX**	***Tárolás***
5XX	**LDA XX**	***Betöltés***
901	**IN**	***Beolvasás (INPUT)***
902	**OUT**	***Kiírás (OUTPUT)***
000	COB	***Leállás (Coffe break)***

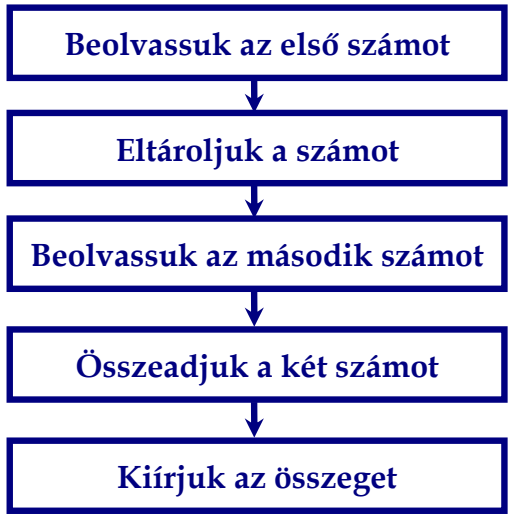
6.4. ábra Az LMC legegyszerűbb alap utasításainak assembly kódja

Az LMC modellben tehát az utasításokat, egészen pontosan az utasítások gépi kódját a kisember a levelesládákban tárolja. A postafiókokat a kisember számértékek – három számjegyű decimális számok – tárolására használhatja, amelyek jelenthetnek adatokat, vagy jelenthetnek utasításokat. Az adatokat tároló levelesláda rekeszeket a programok futása során fogja használni az LMC a számolás eredményeinek átmeneti tárolására. Ezeket a rekeszeket is a programfejlesztés során kell kijelölni, a program kódjában a **DAT** kód fogja az adattároló rekeszek (levelesládák) helyét jelölni.

6.4. Programok futtatása LMC modellben

A kisember működése, vagyis egy program futatása igen egyszerű. A számlálót a „reset” gomb segítségével beállítjuk alapállapotba, vagyis nullára. Feltételezzük, hogy a levelesládákban a 0. rekesztől kezdődően sorban benne vannak az utasítások. A kisember kiveszi az utasítás kódját a számláló által mutatott postaládából, értelmezi azt és végrehajtja az ott definiált utasítást, majd az utasítás végrehajtása végén megnyomja a számláló pedálját, aminek hatására az egyet számol felfelé.

Készítsünk egy egyszerű programot, adjunk össze két számot. A program megírásához először tervezzük meg, hogyan fogjuk elvégezni az LMC segítségével a feladatot. A program által elvégzendő lépéseket egy folyamatábrán terveztük meg, melyet a 6.5. ábra mutat.



6.5. ábra Két szám összeadásának folyamatábrája LMC segítségével

A folyamatábra elég részletes, így a folyamatábra alapján közvetlenül megírhatjuk a lépéseket megvalósító assembly utasításokat (lásd 6.6. ábra). Minden sor végén egy-egy megjegyzést írtunk pontosvesszővel elválasztva, mely segíti a program értelmezését. Ezeket a fordító program a fordítás során figyelmen kívül hagyja.

Levelesláda címkéje	*Assembly kód*	*Utasítás leírása*
00	**IN**	;Első szám beolvasása
01	**STO 99**	;Szám tárolása
02	**IN**	;Második szám beolvasása
03	**ADD 99**	;Két szám összeadása
04	**OUT**	;Eredmény kiírása
05	**COB**	;Leállás
99	**DAT 00**	;Adat rekesz a szám tárolására

6.6. ábra Két szám összeadását végző LMC számítógépen végrehajtható assembly program

A fordító az assembly programból közvetlenül generálja azt a gépi kódú utasítás sorozatot, mely megvalósítja a kívánt funkciót. A gépi kódú utasí-

tás sorozatot a 6.7. ábra mutatja. Ezt a programot a levelesládákba kell betölteni ha szeretnénk azt, hogy az LMC végrehajtsa a két szám összeadását. A gépi kódú utasítás tehát nem lesz más, mint a levelesládák tartalma a program futás megkezdésekor. Az utasítások után szereplő megjegyzések természetesen nem lesznek a levelesládákban.

Levelesláda címkéje	*Gépi kód*	*Utasítás leírása*
00	**901**	;Első szám beolvasása
01	**399**	;Szám tárolása
02	**901**	;Második szám beolvasása
03	**199**	;Két szám összeadása
04	**902**	;Eredmény kiírása
05	**000**	;Leállás
99	**000**	;Adat rekesz a szám tárolására

6.7. ábra Két szám összeadását végző program gépi kódja (LMC)

6.5. Egyszerű programvezérlő utasítások

A programok felhasználóinak gyakori elvárása, hogy a programok a bemeneti, ill. a feldolgozott adatok értékétől függően különbözőképen viselkedjenek. Általános igény, hogy a felhasználó ki szeretné választani, milyen feldolgozási lépéseket akar végrehajtani egy adott bemeneten, vagy szeretné, ha a program automatikusan felismerné, ha hiba történik az adatok feldolgozása során. A programok végrehajtása ezért gyakran megköveteli, hogy az elvégzett számolási művelet eredményétől függően más és más utasítássorozatot hajtsunk végre. Erre a programvezérlő utasítások adnak lehetőséget. A programvezérlő utasítások képesek a programszámlálóban tárolt értéket felülírni, így lehetővé teszik, hogy a számítógép – az LMC modellben a kisember – ne minden esetben a memóriában (az LMC modellben levelesládákban) soron következő utasítást hajtsa végre.

A legegyszerűbb programvezérlő utasítások az *ugró vagy elágazás utasítások*. Ezek az utasítások közvetlenül felülírják a programszámlálót az operandusukban tárolt értékkel. Léteznek feltételes és feltétel nélküli ugró utasítások. A feltétel nélküli minden esetben felülírja programszámlálóban tárolt értéket, tehát mindig eltéríti a program vezérlését, a feltételes ugró utasítás csak az adott feltétel teljesülésekor. A legalapvetőbb program vezérlő utasítások definícióját a 6.8. ábra tartalmazza.

Léteznek bonyolultabb működésű vezérlő utasítások, ezek működését a későbbi fejezetekben mutatjuk majd be.

Utasítás gépi kódja	Utasítás assembly kódja	Utasítás jelentése
6XX	**BR**	***Ugrás az XX számú rekeszre.***
7XX	**BRZ**	***Ha a számológép tartalma nulla, ugrás az XX számú rekeszre.***
8XX	**BRP**	***Ha a számológép tartalma nagyobb nullánál, ugrás az XX számú rekeszre.***
000	COB	***Leállás (Coffee break)***

6.8. ábra Az LMC egyszerű vezérlő utasításainak assembly kódja és definíciója

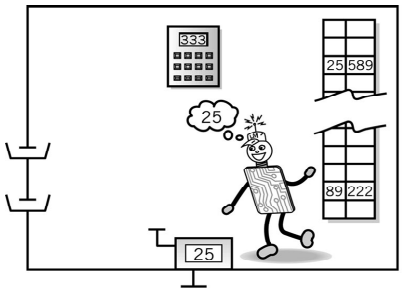
6.6. Utasítás-végrehajtási ciklus

A kisember egy program végrehajtása során egymás után hajtja végre az utasításokat. Ez a tevékenység ciklikus abban az értelemben, hogy a kisembernek hasonló lépéseket kell periodikusan végrehajtania.

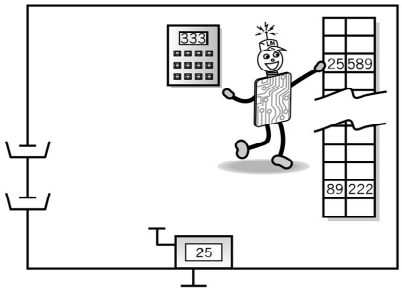
Ha közelebbről megvizsgáljuk a kisember tevékenységét az utasítások végrehajtása során, minden utasítás végrehajtása feloszthatő két fázisra:

- *Fetch (kikeresés):* A kisember kideríti, hogy milyen utasítást hajtson végre.
- *Execute (végrehajtás):* A kisember elvégzi az utasítás által definiált műveletet.

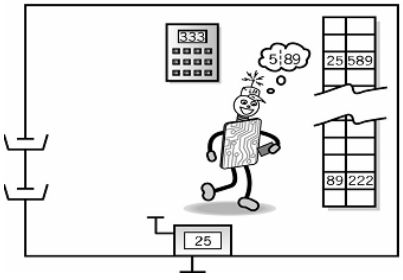
Ezt a két fázist azért fontos megkülönböztetni, mert a kettő közel azonos bonyolultságú és a valós számítógépekben végrehajtásuk ideje közel azonos lesz. Az első fázis minden utasítás végrehajtása esetén hasonló lépésekből áll, a második függ az aktuálisan végrehajtott utasítás típusától. A két utasítás-végrehajtási fázist a 6.9. ábra és a 6.10. ábra mutatja.



- 1. A kisember elolvassa a levelesláda címét a végrehajtandó utasítás helyét tároló számlálóból (utasítás-számláló).

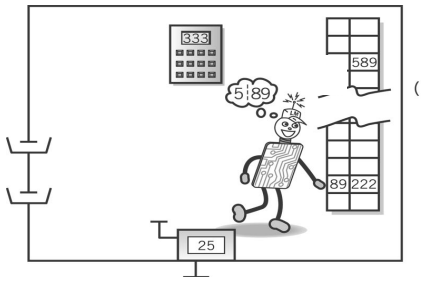


- 2. Elmegy a levelesládához, ami az utasítás-számlálóban tárolt értékhez tartozik.

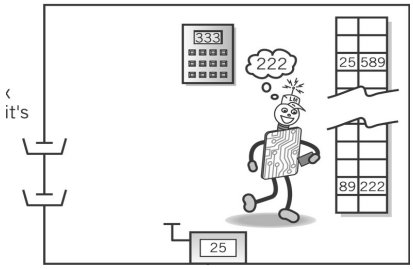


- 3. A cetliről leolvassa levelesládában tárolt a számot (visszarakja a cetlit, hátha szüksége lesz még rá később).

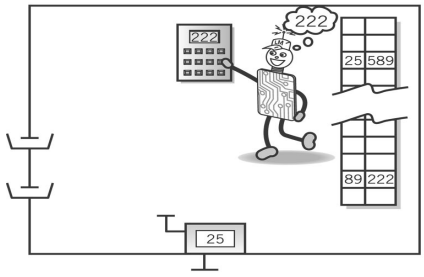
6.9. ábra Az utasítás-végrehajtási ciklus „fetch” fázisa



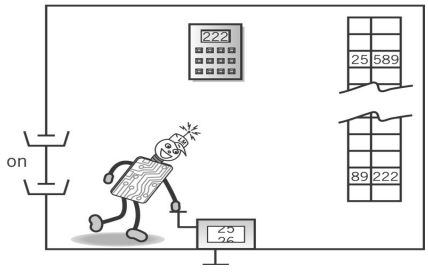
- 1. A kisember odamegy ahhoz a levelesládához, ami az imént fetch-elt utasítás kódjában volt definiálva.



- 2. Kiolvassa a számot a levelesládából (megjegyzi, hátha szüksége lesz rá később).



- 3. Odamegy a számológéphez és begépeli a számot.



- 4. Odamegy a az utasítás helyét tároló számlálóhoz (utasításszámlálóhoz) és megnyomja a pedált (megemeli eggyel az értékét), ami után készen áll fetch-elni a következő utasítást.

6.10. ábra Az utasítás-végrehajtási ciklus „végrehajtás” fázisa

A valós számítógépek esetén is az LMC modellben megismert logikához hasonlóan történik az utasítások végrehajtása. A számítógép által egy-egy utasítás végrehajtásakor elvégzett tevékenységek sorozatát *utasítás végrehajtási ciklusnak* nevezzük, mely név utal arra, hogy az utasítás végrehajtás során a végrehajtott lépések ciklikusan ismétlődnek.

7. A CPU és a memória működésének alapjai

Ebben a fejezetben a számítógép működése szempontjából két legfontosabb elemének működésével ismerkedünk meg. Ez a két elem az adatokon végzendő műveletek végrehajtására képes ún. központi feldolgozó egység (Central Processing Unit), és az adatok tárolására alkalmas memória. A központi feldolgozó egységet az angol Central Processing Unit elnevezés rövidítése alapján CPU-nak nevezzük.

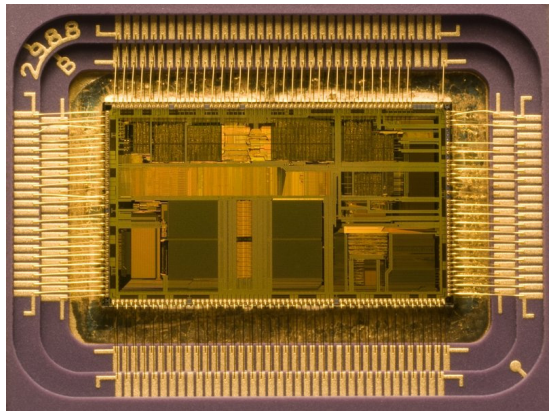
7.1. A CPU

A CPU-t három, különböző funkciót ellátó fő része oszthatjuk:

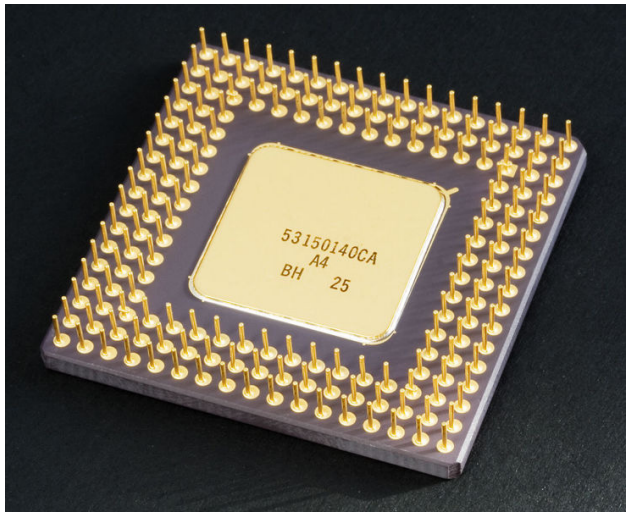
- Az ALU (Arithmetic/Logic Unit, Aritmetikai/Logikai Egység): Számítási (aritmetikai és logikai), valamint összehasonlító műveleteket hajt végre a tárolt adatokon. Működése során megváltoztatja a regiszterek tartalmát a végrehajtott műveletnek megfelelően.
- CU (Control Unit, Vezérlő Egység). A CU vezérli az utasítás-végrehajtást (fetch/execute ciklusokat) a CPU-n belül. Az általa végrehajtott műveletek két típusba sorolhatók:
  - Mozgatja a CPU regisztereiben tárolt adatokat. Jellemzően a tárolt adatértékek nem változnak.
  - Beolvassa a program utasításait a memóriából és parancsokat ad az ALU-nak.

  A CU részei:
  - Memory Management Unit (Memória Vezérlő Egység): felügyeli az utasítások „fetch”-elését és adatok elérését, vagyis az adatok és utasítások mozgatását a memória és a processzor között.
  - I/O Interfész: az adatok mozgatását felügyeli az I/O egységek és a processzor között. Gyakran egybeépítik a Memória Vezérlő Egységgel. Ekkor a két elemet együtt Bus Interface Unit-nak (Bus Vezérlő Egységnek) nevezzük.
- Regiszterek: Kis (néhány bájt) kapacitású tárolóhelyek a CPU-n belül, amiket a CPU a számolások részeredményeinek tárolására, ill. a működésének irányítására használ.

A CPU felsorolt funkcionális részei gyakran fizikailag is elválaszthatók egymástól.



7.1. ábra Az Intel cég 80486DX2-es CPU-jának belseje. A valós mérete 12×6,75 mm tokozással együtt. A külvilággal kapcsolatot teremtő kivezetéseket láthatjuk a kép szélein, míg a különböző funkcionális részeket megvalósító áramköröket a kép közepén.

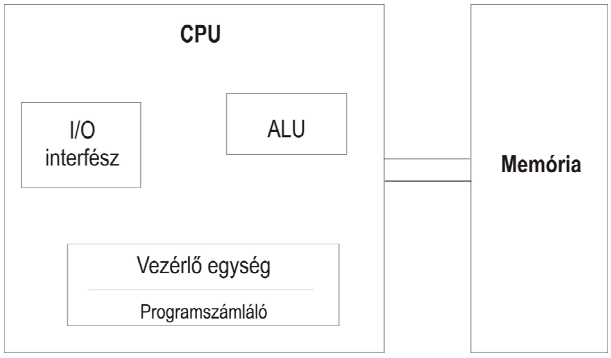


7.2. ábra Az Intel cég 80486DX2-es CPU-jának tokozása alulról, melyen jól láthatjuk a kivezetéseket.

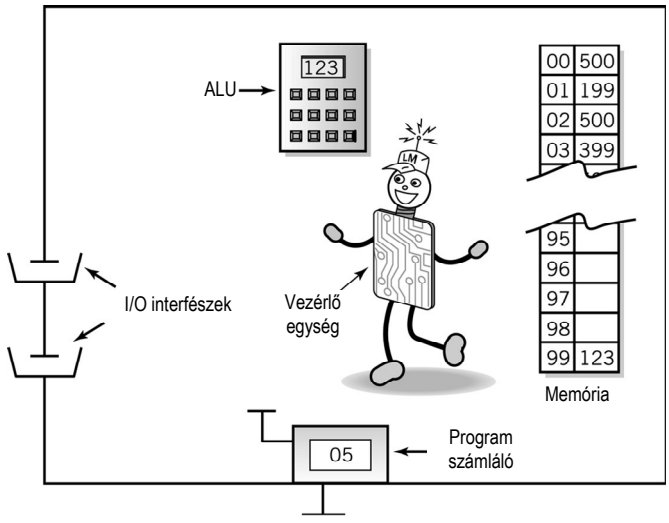
7.2. A számítógép sematikus felépítése

A számítógép felépítését a 7.3. ábra mutatja. A CPU és a memória szoros összeköttetésben van egymással. A memóriában tárolhatunk adatokat és a CPU által végrehajtható utasítássorozatokat, vagyis programokat. A CPU a program utasításait fogja a memóriából a CU által meghatározott sorrendben végrehajtani. A számítógép működése során a memóriában tárolt adatokon végrehajtja a program utasításai által definiált műveleteket.

A számítógépünk, ill. a CPU korábban ismertetett komponenseit közvetlenül megfeleltethetjük az LMC modell egyes elemeinek (lásd 7.4. ábra).



7.3. ábra A számítógép sematikus felépítése



7.4. ábra Az LMC modell elemeinek megfeleltetése a számítógép, ill. a CPU egyes funkcionális elemeinek

7.3. Regiszterek

A regiszterek kis kapacitású tárolóhelyek a CPU-n belül. Működésük egyszerű: regiszterek a bennük tárolt információt változatlan formában megőrzik, míg azt egy CPU utasítás meg nem változtatja. A regisztereket a CPU a számolások részeredményeinek tárolására, ill. a saját működésének irányítására használja.

A regiszterek működését, a köztük történő adatmozgást a CU irányítja. Általában minden regiszternek előre definiált feladata van a CPU-n belül, a feladatnak megfelelően vannak kialakítva, ill. egymással összekapcsolva. Tároló méretük néhány byte (ritkán néhány bit) nagyságú. Funkciójuktól függően tárolhatnak adatértéket, memóriacímeket vagy utasításokat.

Az LMC modellben is találunk regiszter funkciót ellátó elemeket. A számológép képes tárolni egy három számjegyű értéket. Hasonlóan az utasítások helyét tároló számláló is regiszter funkciót lát el.

Az aktuálisan futó program számára a regiszterek a leggyorsabban – legkisebb időveszteséggel – elérhető tárolók. Ennek megfelelően a programok itt tárolják a gyakran használt, ill. gyorsan elérendő adatokat.

A CPU a futó programra és a saját állapotára vonatkozó információkat is a regiszterekben tárolja, regiszter mondja meg a következő program utasítás címét, regiszter fogadja a külső eszközökből érkező jeleket stb.

A regiszterek felhasználása alapján a regisztereket két csoportra osztjuk:

- általános célú regiszterek, ill.
- speciális célú regiszterek.

Az általános célú regiszterek a programok számára láthatóak, a CPU utasításokkal értékeket tölthetünk beléjük, módosíthatjuk azokat. Jellemzően adat értékeket tárolunk bennük. A modern processzorokban akár több tucat ilyen regiszter lehet. Az általános célú regisztereket hasonlóan használjuk, mint az LMC modellben megismert számológép tároló funkcióját.

A speciális célú regisztereknek pontosan definiált feladatuk van a CPU-ban. Ritkán lehet a bennük tárolt értékeket CPU utasításokkal közvetlenül módosítani. A processzorokban leggyakrabban használt speciális célú regiszterek a következők:

- Program Count Register (PC, Program számláló regiszter): A következő utasítás helyét mutatja meg a memóriában. Emiatt instruction pointer-nek (utasítás mutatónak) is hívjuk.
- Instruction Register (IR, Utasítás regiszter): A memóriából utoljára beolvasott utasítás kódját tartalmazza.

- Memory Address Register (MAR, Memória Cím Regiszter): A memória olvasás/írása esetén az írni/olvasni kívánt memóriarekesz címét tartalmazza.
- Memory Data Register (MDR, Memória Adat Regiszter): A memória olvasás/írása esetén az írni/olvasni kívánt memóriarekesz tartalmát tartalmazza.
- Status Register (SR, Állapot regiszter): A processzor és a futó program állapotát leíró információt tárolja. A processzorban ún. flag-eket (jelzőbiteket), vagyis egybites logikai változókat használunk az egyes események bekövetkeztének jelzésére. Az SR ezeket a jelzőbiteket tartalmazza. Segítségével nyomon tudjuk követni a számítógép működését, láthatjuk például az ún. túlcsordulás eseményeket aritmetikai műveleteknél, tudomást szerezhetünk az elektromos kimaradás, vagy egyéb belső processzor hibákról.

7.3.1. Regiszterekben tárolt értékeken végezhető műveletek

A regiszterekben tárolt adatokon különböző műveleteket végezhetünk. A műveletek egy része ún. adatmozgató művelet. A processzor számos utasítást tartalmaz, melyek segítségével más regiszterekből, ill. a memóriából származó adatokat tölthetünk a regiszterekbe, ill. regiszterekben tárolt adatokat mozgathatjuk a memóriába.

A regiszterekben tárolt adatokon végezhetünk különböző számolási műveleteket pl. összeadás vagy kivonás. Végezhetünk bitenkénti műveleteket pl. Shift – Eltolás vagy Rotate – Forgatás. A regiszterek tartalmát ellenőrizhetjük, pl. megnézhetjük, hogy az érték nulla vagy pozitív. Ezen utóbbi műveletek eredménye az SR-ben lesz látható.

7.4. Memória

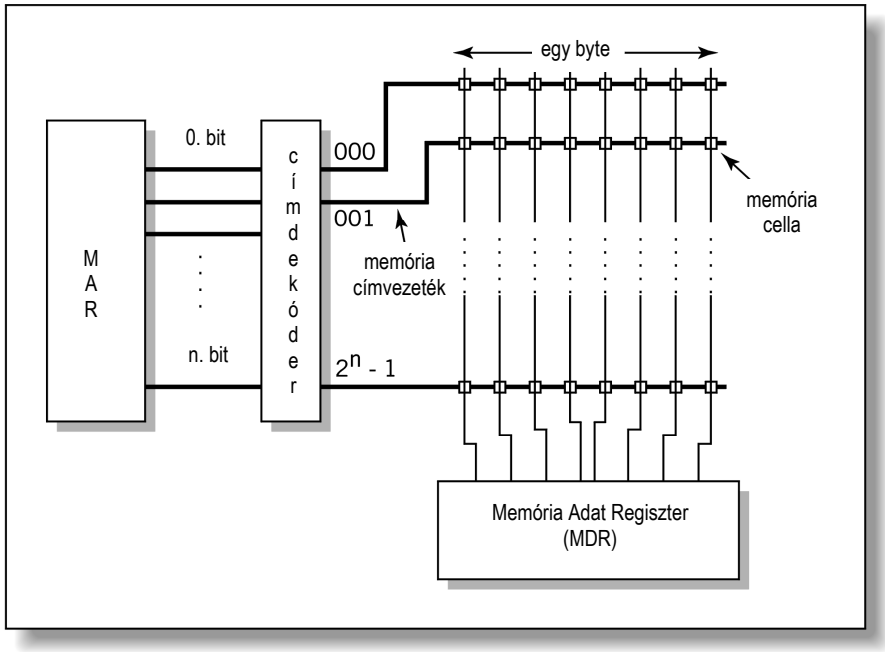
A számítógépnek a programok futtatásához, adatfeldolgozó műveleteinek végrehajtásához adatokat és utasításokat kell tárolni. A memória a számítógépben az adatok és utasítások tárolására szolgáló eleme. A memóriát elsődleges tárolónak (primary storage) is szokták nevezni, mert a CPU az adattároló funkciót betöltő nagy tároló kapacitású komponensek közül a memóriában levő adatokat éri el a leggyorsabban, ez a komponens van a processzorhoz a „legközelebb”.

A memória működése igen egyszerű. A memória információ tárolására szolgáló rekeszeket tartalmaz. A memóriarekeszek a memória típusától – szervezésétől – függően különböző számú bit tárolását teszik lehetővé. Ez

lehet 8 bit, de akár 128 bit is. A memóriarekeszekben az egyes biteket ún. memóriacellák tárolják.

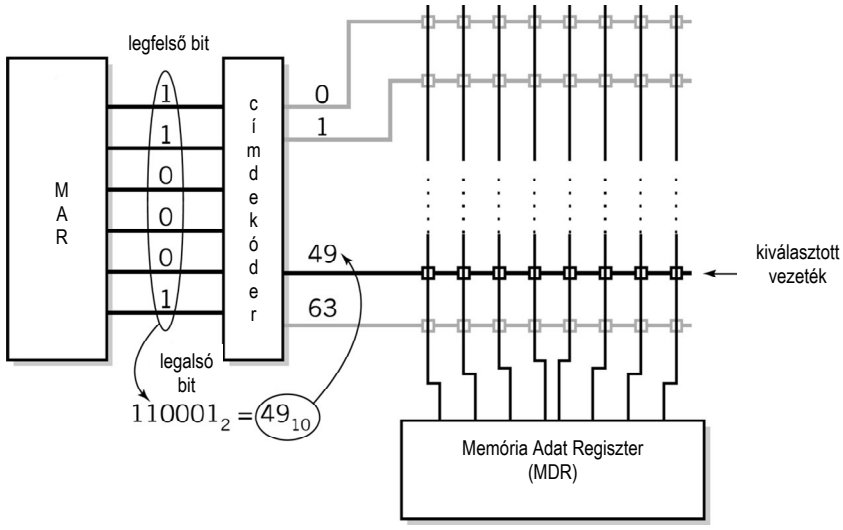
Minden rekesz egyedi címmel rendelkezik. Az Egyes rekeszekre ezzel a memóriacímmel hivatkozunk. Egy program a CPU utasításai segítségével egy adott memóriacímen tárolt adatot olvashat, ill. írhat. A memória biztosítja, hogy egy adott memóriarekeszben – egy adott memóriacímen – tárolt érték változatlan marad, amíg végre nem hajtunk rajta egy memória író utasítást.

A memória tartalmát a CPU utasításokkal tudjuk elérni. Amikor a processzor egy utasítás végrehajtása során olvasni vagy írni akarja a memória egy adott memóriacímmel azonosított rekeszét, akkor azt a címet a Memória Cím Regiszterbe (MAR-be) tölti. A CPU ezután meghatározza, hogy írni vagy olvasni akarja az adott memóriarekeszt. Az utasítás végrehajtása eredményeként a MAR által kijelölt memória rekesz és a Memória Adat Regiszter (MDR) között adatcsere történik. A memória írásakor az MDR tartalma felülírja a memóriarekeszben tárolt értéket, a memória olvasásakor a memóriarekeszben tárolt értékkel felülíródik az MDR tartalma. Az MDR és a MAR kapcsolatát a memóriával a 7.5. ábra mutatja.



7.5. ábra Az Memória Cím Regiszter (MAR) és az Memória Adat Regiszter (MDR) kapcsolata a memória rekeszekhez

A memória egy-egy bit tárolására szolgáló memóriacellákból, azok összeköttetéseiből, valamint egy ún. címdekóderből áll. A címdekóder egy egyszerű logikai áramkör, melynek – bináris (kétállapotú) jelek használata esetén – n darab bemenete és 2^n kimenete van. Működése egyszerű. Mindig legfeljebb egyetlen kimenete van aktív (engedélyezett) állapotban, a bemenetén pedig kiválaszthatom, hogy hányadik (hányas sorszámú) kimenete legyen aktív. A dekóder és a MAR összekapcsolását a 7.6. ábra mutatja. Az ábrán láthatjuk, hogyan tudjuk a címdekóder segítségével a MAR által mutatott memóriarekeszt kiválasztani.



7.6. ábra Az Memória Cím Regiszter (MAR) és az Memória Adat Regiszter (MDR) kapcsolata a memória rekeszekhez

7.5. A számítógépben használható memória mérete

A számítógépben rendelkezésre álló memória mérete meghatározó jelentőségű a számítógép segítségével megoldható feladatok, vagyis a számítógépen futatható programok szempontjából. A számítógépekben a használható memória méretét az határozza meg, hogy a CPU hány különböző memóriacímet tud elérni. A megcímezhető memóriarekeszek számát két tényező korlátozza:

- a MAR bit-jeinek száma, valamint
- a CPU utasítások operandus részéhez rendelt bitek száma.

Ha a MAR K bit szélességű, akkor – bináris jeleket feltételezve – 2^K memóriarekeszt tudunk elvileg megcímezni. A használható memória címek: $0..2^K-1$. Az LMC modellben a MAR két digit szélességű volt, viszont egy cím vezetéken 10 különböző jelet használhattunk, ennek megfelelően ott 10^2 vagyis 100 rekeszt tudtunk megkülönböztetni. Ezek címei a 00-99 tartományba estek.

Az utasítások operandus részének hossza általában szigorúbb korlát, mint a MAR szélessége. A szabály itt is azonos, K szélességű operandus címmel – bináris jeleket feltételezve – 2^K memóriarekeszt tudunk elvileg megcímezni.

Ez azt jelenti, hogy 4 biten 16 rekeszt tudunk megkülönböztetni, 8 biten 256 rekeszt, 4 Giga (4,294,967,296) rekesz megcímzésére pedig 32 bit szükséges. Mivel a CPU utasítás kódjának az operanduson kívül a műveleti kódot is tartalmaznia kell, ez még a modern, széles utasításokat használó processzorokban is csak ritkán valósítható meg. Emiatt a processzorok tervezői különböző áthidaló megoldásokat találtak annak érdekében, hogy a memória elérése esetén az operandus résznek ne kelljen az egész memóriacímet, csak annak egy részét tartalmaznia. Ezekkel a megoldásokkal a memóriacímzési módokkal foglalkozó részben fogunk részletesebben megismerkedni.

A számítógép memóriájában tárolható információ nagysága nem csak a rekeszek számától, hanem az egyes rekeszek méretétől is függ. A rekeszek tartalmát a processzor a MDR-en keresztül olvassa, ill. írja, tehát az MDR bitszélessége ugyancsak meghatározó a memória méretének kiszámításakor. Míg 8 bites MDR – ill. a hozzá kapcsolódó, adatok szállítását biztosító ún. adatsín (adat-bus) – esetén 32 címbitet feltételezve 4 Giga byte adatot tudunk tárolni, addig 32 bites MDR esetén ez négyszer nagyobb, vagyis 16 Giga byte.

A számítógépben használt memória mérete azért fontos, mert a programok végrehajtási sebessége erősen függ attól, hogy mennyi memória áll rendelkezésre. Sok adatot feldolgozó programok futtatáskor kevés memória esetén a processzornak tétlenül kell várakoznia arra, hogy az adatok a memóriába töltődjenek. Tapasztalatok szerint kritikus esetekben a várakozási idő akár a futási idő 50%-át is kiteheti.

7.6. Különböző típusú memóriák

A számítógépekben a használhat memóriákat használatuk módja alapján két csoportra oszthatjuk:

- Random Access Memory (RAM, véletlen elérésű/hozzáférésű memória),
- Read Only Memory (ROM, csak olvasható memória).

A két memória közötti különbség az, hogy a RAM memóriában tárolt adatokat nem csak olvashatjuk, hanem írhatjuk is, míg a ROM tartalmát a számítógép futása során egyáltalán nem, vagy csak – az olvasás sebességéhez képest – igen lassan írhatjuk.

A RAM eredetileg mágnesezhető tároló cellákból épült fel. Ezt a technológiát már nem használják, ma DRAM és SRAM memória chipeket használunk.

A DRAM (Dynamic RAM, Dinamikus RAM memória) a legáltalánosabban használt memóriatípus, mert a többi memóriához képest olcsó. Úgynevezett volatilis tároló, a feszültség kikapcsolásával elvész a benne tárolt tartalom. A DRAM nevét onnan kapta, hogy a benne levő memóriacellák tartalmukat normál működés esetén is hamar elveszítik: másodpercenként kb. 1000-szer újra kell őket írni az aktuálisan tárolt tartalmukkal. Ez a felhasználó elöl rejtett mechanizmus, amit a memória-chip automatikusan elvégez.

Az SRAM (Statikus RAM) a DRAM-hoz viszonyítva gyorsabb adatelérést tesz lehetővé, de az eltérő gyártási technológia miatt lényegesen drágább. Az SRAM is volatilis tároló, vagyis a feszültség kikapcsolásával elvész a tartalma. Viszonylag magas ára miatt keveset használunk belőle, a számítógépekben jellemzően az ún. gyorsító-tárakban (cache) alkalmazzák.

A ROM memória stabil, nem volatilis tároló, mely a benne tárolt tartalmat a számítógép kikapcsolása után is megőrzi. A ROM-okat olyan programok tárolásához használják, amelyek változatlanok a számítógép élete során. Legismertebb alkalmazási területük a számítógépek indulásakor futó ún. betöltő program tárolása. ROM alkalmazásával a számítógép bekapcsoláskor a betöltő program azonnal a memóriában lesz, lehetővé téve az operációs rendszer felhasználó közreműködése nélkül történő elindulását.

A ROM-ok legkorábban megjelent és ma is használt olcsó változatainak tartalma a tokozáskor véglegesedik, azt később nem lehet megváltoztatni.

Ennél egyszerűbb a PROM (Programmable Read-Only Memory, Programozható ROM) használata, amelyet használata során egyszer lehet írni, a programozása valamilyen visszafordíthatatlan folyamatot jelent (pl. ellenállások elégetését).

Az EPROM (Erasable Programmable Read-Only Memory, Törölhető és Programozható ROM) azt a lehetőséget is biztosítja, hogy az egyszer betöltött tartalmat törölni lehessen. A törlést leggyakrabban a memória chip ultraibolya tartományba eső fénnyel történő megvilágításával tehetjük meg.

Az EEPROM (Electrically Erasable Programmable ROM, elektromosan törölhető és programozható ROM) esetén a tervezők azt a lehetőséget is megadják a felhasználónak, hogy a memóriában tárolt információ elektromosan írható legyen. Az EEPROM írása azonban sokkal (nagyságrendekkel) lassabb, mint a RAM memóriák írása. További hátránya az EEPROM memóriáknak a RAM memóriákhoz képest, hogy gyakran vagy csak egy-egy bitet lehet törölni egyszerre, vagy az egész EEPROM chip tartalmát.

Meg kell említeni még az ún. Flash memóriákat, amelyeket ugyancsak ROM típusú memóriáknak tekinthetünk. Nevüket a törlésük módjáról kapták („flashed back to zero”), amit magyarul leginkább kisütésnek lehet fordítani. A Flash memóriák a technológia szempontjából sokban hasonlítanak az EEPROM-okra, azonban szervezésük már gyakran nem a memória chip-ek szervezésére hasonlít, ahol egy-egy rekeszt (néhány byte adatot) lehet egy címmel elérni, hanem a háttértárak szervezésére, ahol egyszerre több tíz, száz, vagy ezer byte adatot érünk el, kezelünk, olvasunk, ill. írunk fölül egyszerre.

7.7. CPU utasítások végrehajtása

Megismerve a processzor felépítését egy fontos megállapítást tehetünk. A processzor a CPU utasításokat olyan lépések sorozataként hajtja végre, mely lépések vagy regiszterek közötti, vagy memóriacellákból regiszterbe, ill. regiszterekből memóriacellákba történő adatmozgató műveletek, vagy az ALU által végrehajtott adatmanipuláló műveletek.

Ha a korábban megismert Fetch és Execute utasítás-végrehajtási ciklusokat nézzük, akkor láthatjuk, hogy mindkét utasítás-végrehajtási fázis egy-egy memória-hozzáférést tartalmaz, hiszen általában mind az utasítás kódja, mind az az adat, amin az utasítást végrehajtjuk a memóriában van eltárolva.

Az LDA (betöltés) utasítás végrehajtása	
PC → MAR	A PC-ben tárolt érték (a következő utasítás címe) a MAR-ba töltődik.
MDR → IR	A memóriából az MDR-be töltődő adatokat az IR-be mozgatjuk.
IR(cím) → MAR	Az utasítás operandus részében szereplő cím betöltődik a MAR-ba.
MDR → A	A MDR-ben levő adatot az A regiszterbe mozgatjuk.
PC + 1 → PC	A programszámlálót megnöveljük eggyel az utasítás végrehajtás végén.
A STR (tárolás) utasítás végrehajtása	
PC → MAR	A PC-ben tárolt érték (a következő utasítás címe) a MAR-ba töltődik.
MDR → IR	A memóriából az MDR-be töltődő adatokat az IR-be mozgatjuk.
IR(cím) → MAR	Az utasítás operandus részében szereplő cím betöltődik a MAR-ba.
A → MDR	Az A regiszterben levő adatot az MDR-be mozgatjuk.
PC + 1 → PC	A programszámlálót megnöveljük eggyel az utasítás végrehajtás végén.
Az ADD (összeadás) utasítás végrehajtása	
PC → MAR	A PC-ben tárolt érték (a következő utasítás címe) a MAR-ba töltődik.
MDR → IR	A memóriából az MDR-be töltődő adatokat az IR-be mozgatjuk.
IR(cím) → MAR	Az utasítás operandus részében szereplő cím betöltődik a MAR-ba.
A + MDR → A	Az A-ban és az MDR-ben levő értékeket összeadjuk, és az eredmény az A regiszterben tároljuk.
PC + 1 → PC	A programszámlálót megnöveljük eggyel az utasítás végrehajtás végén.

7.7. ábra Az LDA, STR és ADD utasítások regiszter-transzfer lépésekre bontott végrehajtása az LMC számítógépben

A két utasítás-végrehajtás ciklust a következő lépésekre bonthatjuk le:

- *Fetch:* Megkeressük és a memóriából a regiszterekbe töltjük, majd dekódoljuk az utasítást. Ezután a megfelelő jelzést küldünk az ALU-nak, hogy milyen műveletet kell végrehajtani.
- *Execute:* Végrehajtjuk azt a műveletet, amit az utasítás kódol. Ehhez ha szükséges adatokat mozgatunk a memóriából a regiszterekbe, ill. szükség esetén az eredményt megint csak eltároljuk a memóriában.

A 7.7. ábra három utasítás (LDA, STR és ADD) végrehajtását mutatja az LMC számítógépben. Az egyes regiszterek nevei megegyeznek a korábban megismert regiszter elnevezésekkel, az LMC modellbeli számológép adattároló kapacitásának megfelelő regisztert „A" regiszternek neveztük el. A processzor műveletek két végrehajtási ciklusát szaggatott vonallal választottuk el. Az egyes regiszterek közötti műveletek között értelemszerűen a processzor használja a memóriát, vagyis végrehajtódik memóriaolvasás, ill. írás. Az LDA és STR utasítások között a különbség tehát nem csak az, hogy a negyedik lépésben az adatmozgatás iránya az A és az MDR regiszterek között más, hanem az is, hogy az egyiknél memória olvasás, míg a másiknál memória írás történik.

Az LMC modellben megismert minden utasítást lebonthatunk hasonló lépéssorozatokra. Az utasítások végrehajtásának lépéseit a 7.8. ábra mutatja.

SUB
PC → MAR
MDR → IR
IR[addr] → MAR
A – MDR → A
PC + 1 → PC

IN
PC → MAR
MDR → IR
IOR → A
PC + 1 → PC

OUT
PC → MAR
MDR → IR
A → IOR
PC + 1 → PC

COB
PC → MAR
MDR → IR

BR
PC → MAR
MDR → IR
IR[addr] → PC

BRZ, BRP
PC → MAR
MDR → IR
Hamis a feltétel: PC+1 → PC
Igaz a feltétel: IR[addr] → PC

7.8. ábra Az SUB, IN, OUT, COB, BR, BRP és BRZ utasítások regiszter-transzfer lépésekre bontott végrehajtása az LMC számítógépben

7.8. Bus-ok

Mint láttuk a számítógép működéséhez, a CPU utasításoknak a végrehajtásához szükség van a CPU egyes részei, komponensei között adatokat

mozgatni. A számítógép elemeinek páronként történő összekötése az elemek nagy száma miatt általában nem hatékony megoldás. Ezért a számítógép komponenseinek összekapcsolására, az adatátvitel lehetőségének biztosítására több elem között ún. síneket (angolul bus-okat) használunk. A sín fizikai kapcsolatot teremt a sínre kapcsolódó számítógép komponensek között, ami lehetővé teszi az adatátvitelt a kapcsolódó elemek között.

A sín (bus) fizikai megvalósítását tekintve elektromos jeleket továbbító vezetékek csoportja. A számítógép ezeknek az elektromos jeleknek az átvitelével valósítja meg az adatcserét. A vezetékeken történő adatforgalmat, a csomópontok közötti jelátvitelt a sín (bus) típusától függően általában valamilyen intelligens vezérlő elem szabályozza.

A sínen az egyes vezetékeket vonalnak (line) vagy jelvezetéknek nevezzük. Az egyes vezetékekre jellemző, hogy azokat hogyan használja a számítógép, rajtuk milyen információt reprezentáló jeleket továbbít. A sín (bus) vezetékeken továbbított jeleknek négy típusát különböztetjük meg:

- *Adat jelek:* A számítógép által kezelt adatokat továbbító jelek. Az adatok típusa különböző lehet: alfa-numerikus, numerikus, utasítások kódja stb.
- *Cím jelek:* A memória rekesz címét, vagy a sínre csatlakozó elemek azonosítóját továbbító jelek.
- *Vezérlő jelek:* A sínen történő adatforgalmat szabályozó jelek. Ezeket a jeleket mind a sínre csatlakozó elemek, mind a sínvezérlő áramkör használhatja. Ezek segítségével tudják pl. a sínre csatlakozó elemek megkülönböztetni, hogy mikor tudják a sínt használni, mikor küldhetnek adatot.
- *Tápellátást biztosító jelek:* A különböző sínre csatlakozó elemek is elektronikus áramkörök. Az elemek gyakran a tápfeszültséget is a sín vezetékeken kapják.

7.9. A számítógépben használt sínek (bus-ok) jellemzői

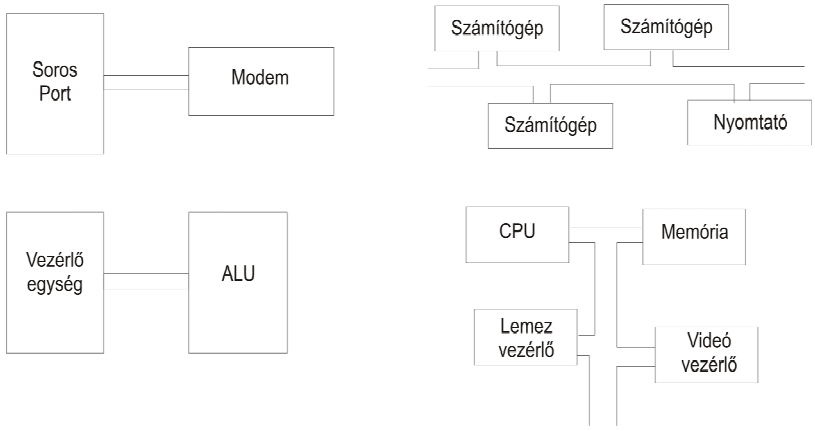
Egy számítógépben általában több bus is van. Az egyes bus-ok működése különböző, a működésüket a bus-hoz kapcsolódó és rajta keresztül kommunikáló elemek igénye szerinti optimalizálták.

A számítógép működése szempontjából legfontosabb bus a processzor és memória közötti bus. Az I/O perifériák lehetnek azonos bus-on a

CPU-val és a memóriával vagy külön bus-on. Ha a CPU, a memória és az I/O elemek közös bus-on vannak, ezt a bus-t *rendszer bus*-nak, vagy *külső bus*-nak nevezzük. Angol elvevezései: system bus, external bus, backplain. Rendszer bus-használata esetén az elemeket egy nyomtatott áramkörre szokásos ültetni, ezt nevezzük alaplapnak (motherboard). Ekkor a bus az alaplapra nyomtatott áramkör egy része lesz, a bust-t használó egyes elemek közvetlenül az alaplapra lesznek beültetve, vagy oldható csatlakozókon keresztül lesznek a bus-ra csatlakoztatva. A PC-kben általában ezt a megoldást alkalmazzák.

A számítógépben használt bus-ok egyik legfontosabb jellemzője a azok fizikai felépítése. Felépítés szempontjából egy bus lehet

- pont-pont kapcsolatot megvalósító, vagy
- többpontos kapcsolatot megvalósító (lásd 7.9. ábra).



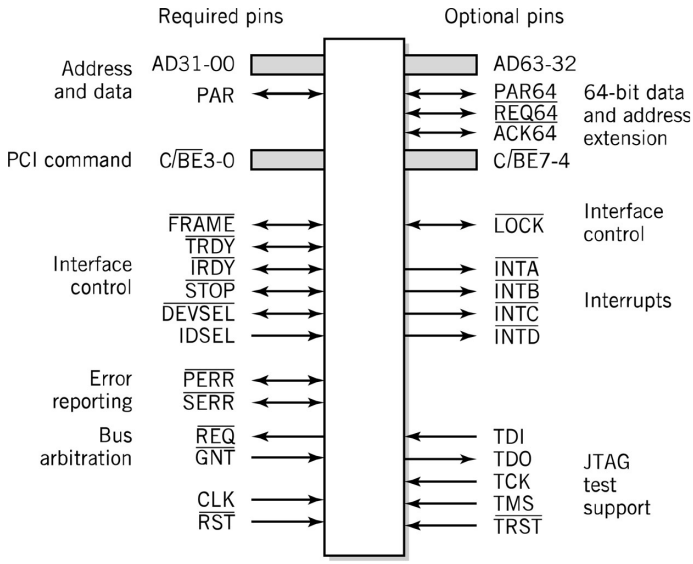
7.9. ábra Pont-pont kapcsolatot és többpontos kapcsolatot megvalósító bus-ok

A többpontos kapcsolatot megvalósító bus-okat szokták még „broadcast" bus-nak is nevezni. Ez az elnevezés arra utal, hogy ilyen bus-okon lehetséges az ún. „broadcast" (információ szórásos) kommunikáció. „Broadcast" kommunikáció esetén az elemek úgy kommunikálnak, hogy az általuk elküldött jeleket, vagyis információt minden bus-ra kapcsolódó elem megkapja.

A számítógépben használt bus-okat további egyéb paraméterrel is jellemezhetjük. Ezek közül a legfontosabbak a következők:

- *Adatátviteli kapacitás*: milyen mennyiségű adatot tudunk a bus-on egységnyi idő alatt átvinni. Mértékegysége bit/másodperc.
- *Adat-bus szélessége*: az egyszerre (egy kommunikációs ciklusban párhuzamosan) átvihető adatok szélessége bitekben mérve.
- *Bus vezeték maximális hossza*: a használható leghosszabb bus vezeték.

A bus-ok működése, használatuk módja, a rajtuk továbbítható jelek típusa, azok jellemzői lényegesen különbözhetnek egymástól. A számítógépben használatos bus-ok egy része ún. dedikált bus, ezek működését a gyártójuk nem feltétlenül hozza nyilvánosságra. A nem dedikált használatú bus-ok esetén a számítógép tervezőinek a bus jellemzőit pontosan definiálniuk kell. Ezt a leírást vagy szabályrendszert *protokollnak* nevezzük. A bus protokoll leírás pontosan megadja a bus-on használt összes vonal és a vonalakon használható jelek jelentését. Egy gyakorlatban is használt bus protokoll leírásának részletét a 7.10. ábra mutatja.



Source: Copyright © PCI Pin List/PCI Special Interest Group, 1999.

7.10. ábra A PCI bus-on használt jelek

7.10. CPU utasítások

Az LMC modell bemutatásakor megismerkedtünk a processzor által végrehajtható legfontosabb utasításokkal. A valós számítógépek a CPU utasí-

tásokat a számítógépben, ill. a processzorban levő digitális áramkörök működtetésével hajtják végre. Az utasítások végerehajtásakor elektromos jelek hatására a számítógép különböző áramkörei működésbe lépnek és a parancsban definiált műveleteket végrehajtják.

A különböző processzorok különböző utasításokat tudnak végrehajtani. Ez azt jelenti, hogy ugyanannak a feladatnak a megoldására készített program két különböző számítógépen (két különböző processzoron) futó változata más és más gépi kódú utasításokat fog tartalmazni.

Egy *processzor utasítás készletének* nevezzük az utasítások azon halmazát, melyet a processzor végre tud hajtani. A processzor utasítás készletét a processzor tervezői definiálják.

A processzorok között az utasítás készletük szempontjából a következő szempontok alapján tehetünk különbséget:

- Végrehajtható utasítások száma, utasításkészlet nagysága.
- Az egy utasítással végrehajtható műveletek bonyolultsága.
- Processzor által támogatott, az utasításokkal feldolgozható adattípusok.
- Utasítások formátuma: kötött vagy változó hosszúságú utasítások.
- Regiszterek használatának lehetséges módja.
- Az operandus címzési módja, a cím mérete.

A számítógép minden utasításához tartozik egy számítógép által értelmezhető kód, a gépi kód, amit az LMC modell ismertetésekor már láttuk. A számítógép utasítások gépi kódját az LMC utasításaihoz hasonlóan két részre oszthatjuk:

- a műveleti kód, a feladatnak a processzor által értelmezhető kódja, amit a processzornak végre kell hajtania, és
- az operandus definíciója.

Az operandus definíciója határozza meg, hogy milyen adatokon kell az adott műveleti kód által meghatározott műveletet elvégezni. Egy adott művelet esetén az operandus definíciónak két dolgot kell meghatároznia: azt, hogy a műveletnek milyen adatokat kell felhasználnia a művelet végrehajtása során, ill. a művelet eredményét hova kell elhelyezni. Ennek megfelelően egy utasítás operandus definíciója tartalmazhatja a

- forrás operandus (source operand) és az
- eredmény operandus (destination operand)

meghatározását.

A processzorutasítás általános felépítését a 7.11. ábra mutatja.

7.11. ábra Egy processzorutasítás általános felépítése

Vannak utasítások, melyekben mindkét operandus meghatározása szerepel, de vannak olyanok is, melyekben egy, vagy egy sem.

Az operandusok utasítástól függően elhelyezkedhetnek a regiszterekben vagy a memóriában. Ha egy utasítás kódja közvetlenül tartalmazza az operandus helyét *explicit* operandus definícióról, vagy más néven operandus címzésről beszélünk, míg ha nem tartalmazza az operandus helyét, azt az utasítás kódja alapján meghatározható alapértelmezés szerinti helyen találjuk *implicit* címzésről beszélünk.

A processzorok utasításainak, utasításkészletének leírását egy számítógép-specifikus leírás tartalmazza, mely része a processzor fejlesztők által készített dokumentációjának. Ez a leírás pontosan meghatározza:

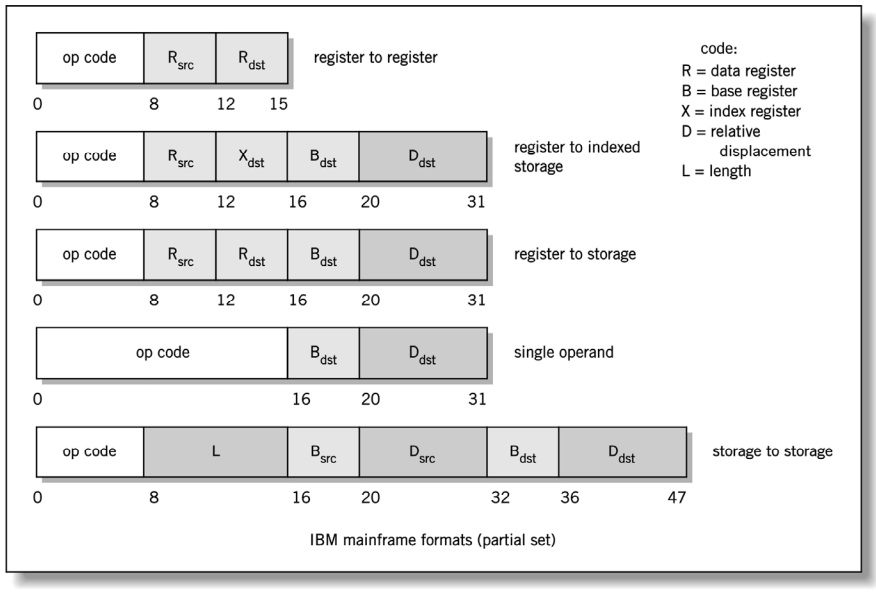
- a műveleti kód hosszát,
- az operandusok számát, és
- az operandusok hosszát.

A processzorok utasításkészletének megtervezése igen bonyolult és nagy körültekintést igénylő feladat. Az utasítások gépi kódjának hosszát számos fizikai paraméter korlátozza, elsősorban a CPU adatbuszának szélessége. Az utasítás kódjának bitjeit fel kell osztani a műveleti kód és az operandus definíció között. A legfontosabb dilema a következő. Ha csökkentjük az operandus definíció hosszát, akkor kisebb memóriaterületet tudunk az utasításban levő címmel elérni, tehát emiatt az operandust minnél szélesebbre kellene választani és csökkenteni kellene a műveleti kód hosszát. Azonban, ha csökkentjük a műveleti kódot tartalmazó bitek számát kevesebb különböző utasítást tudunk megkülönböztetni, ill. használni a programokban. Számos érv szól a mind a két irányba.

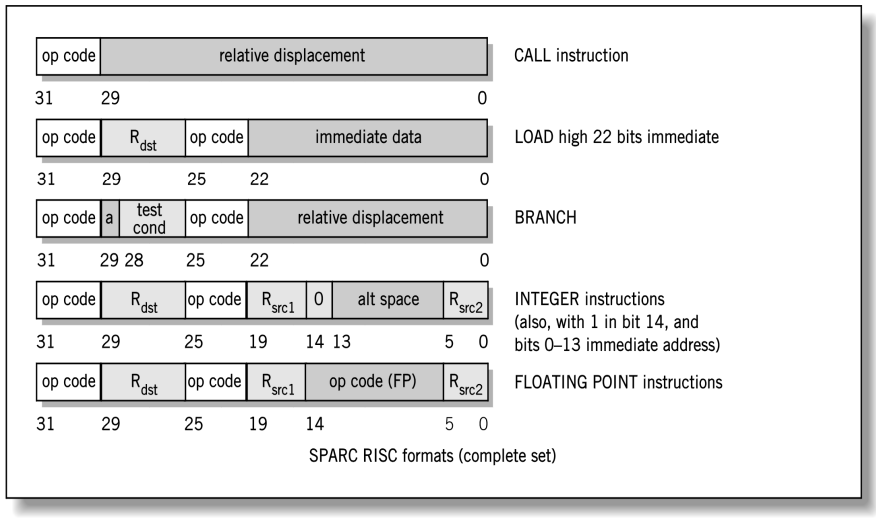
A probléma egyik megoldása lenne a különböző felépítésű és különböző hosszúságú utasítások használata. Ilyen utasításkészletű processzorokat ún. Complex Instruction Set Computer (CISC, Összetett Utasítás Készletű Számítógép) nevezzük. Ennek a megoldásnak több

hátránya is van, melyek elsősorban az utasítások párhuzamos végrehajtásakor jelentkeznek.

A CISC típusú utasításkészlettel rendelkező processzorok ezen hátrányait az ún. Reduced Instruction Set Computer (RISC, Csökkentett Utasítás Készletű Számítógép) típusú számítógépek kiküszöbölik úgy, hogy az utasítások felépítése és hossza is azonos lesz. A CISC és RISC utasításkészletre a 7.12. ábra és a 7.13. ábra ad példát. A CISC és RISC utasításkészlettel rendelkező processzorokkal még fogunk a későbbiekben részletesen is foglalkozni .



7.12. ábra Egy CISC típusú processzor utasításainak felépítése



7.13. ábra Egy RISC típusú processzor utasításainak felépítése

7.11. Utasítások típusai

A különböző processzorok utasításkészlete egymástól igen különböző lehet. Az egyes utasításokat azonban különböző típusba sorolhatjuk, melyeket ebben a fejezetben veszünk sorra. A besorolás azért fontos, mert ugyan az egyes processzorok utasításkészlete különböző, de a különböző processzorok azonos típusba tartozó utasításaiban definiált műveletek tartalma hasonló.

A legegyszerűbb utasítás típusokat az LMC modell kapcsán már megismertük.

7.11.1. Adatmozgató műveletek

Az adatmozgató műveletek adatátvitelt valósítanak meg a számítógép elemei között. Ez lehet két regiszter, vagy egy regiszter és egy memóriarekesz, esetleg két memóriarekesz közötti adatmozgatás. Az LMC modellben ilyen művelet volt a LDA (betöltés) vagy az STR (tárolás).

Az adatmozgató műveleteket minden processzor esetén megtaláljuk. A megvalósítás formája, az operandus definíciója, az egy utasítással mozgatott bitek száma azonban igen változatos lehet.

7.11.2. Aritmetikai utasítások

A processzorok aritmetikai utasításai általában a négy alapműveletet: összadás szorzás, kivonás osztás, valamint a negálás műveleteit tartalmaz-

zák. Néhány speciális célú processzor kivételével általában a legtöbb processzor tartalmaz aritmetikai utasításokat.

A műveletek által kezelt adattípusok szempontjából már kicsit nagyobb a különbség a processzorok között. Általában minden processzor esetén az utasításokat egész és lebegőpontos (tört) típusú adatokon is el lehet végezni, azonban egyes fejlett processzorok megengedik mind a két adattípusnál különböző bitszélességű adatok kezelését.

Általában az aritmetikai utasítások között megtalálhatóak a relációs operátorok: kisebb, nagyobb, egyenlő.

Külön szolgáltatása lehet a processzornak, hogy képes végrehajtani az aritmetikai műveleteket BCD formátumú adatokon is.

7.11.3. Boolean logikai műveletek

A legtöbb processzor lehetőséget ad a Boolean logikában definiált logikai operátorok használatára: **AND, OR, XOR, NOR** és **NOT.**

7.11.4. Egy operandusú utasítások

Ezt az utasítástípust azért kezdték el használni, mert a programok végrehajtásakor gyakran van szükség ebbe a kategóriába tartozó műveletekre. Három jellemző egy operandusú művelet van:

- Dekrementálás: egy adott érték eggyel történő csökkentése.
- Inkrementálás: egy adott érték eggyel történő növelése.
- Negáció: egy adott érték invertálása. A művelet eredménye függ az adattípustól, gyakran azonos a NOT művelettel.

7.11.5. Bitmanipuláló utasítások

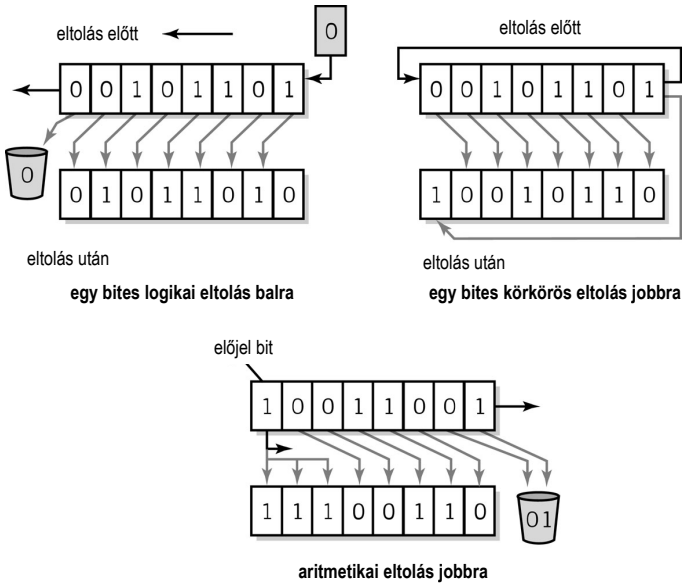
Egy-egy bitet módosító utasítások, ill. egy adott bit értéke alapján történő ugró (elágazás) utasítások. Ezeket az utasításokat elsősorban a jelzőbitek (flag-ek) használata, ill. a rájuk vonatkozó feltételek tesztelése esetén használhatjuk.

7.11.6. Bitenkénti eltolás, ill. forgatás

A bitenkénti eltolás művelet az adott regiszterekben tárolt értéket tolja el egy bittel. Az eltolás művelet elsősorban azért jelent meg a CPU utasításai között, mert a kettővel való osztás, ill. szorzás műveletek eltolással megvalósíthatók. Ennek megfelelően van ún. aritmetikai eltolás, ahol az eltolás esetén a processzor figyelembe veszi, hogy a legszélső bit az előjel bit (ill. egész pontosan a kettes komplemens ábrázolás esetén az előjelnek is meg-

feleltethető), így azt nem kell eltolni, valamint azt, hogy jobbra tolás esetén az üres biteket egyesekkel, balra tolás esetén nullákkal kell feltölteni.

Természetesen létezik logikai eltolás művelet is, ahol egyszerűen eltoljuk a tárolt biteket nullákkal feltöltve a szabaddá vált bithelyeket. A forgatás ugyan ezt csinálja azzal a különbséggel, hogy a processzor a regiszter végén „kiléptetett" biteket a szabaddá váló helyekre „belépteti". A logikai eltolást és forgatást nem csak aritmetikai műveletek elvégzésére használják. Példákat az eltolás és forgatás műveletekre a 7.14. ábra mutat.



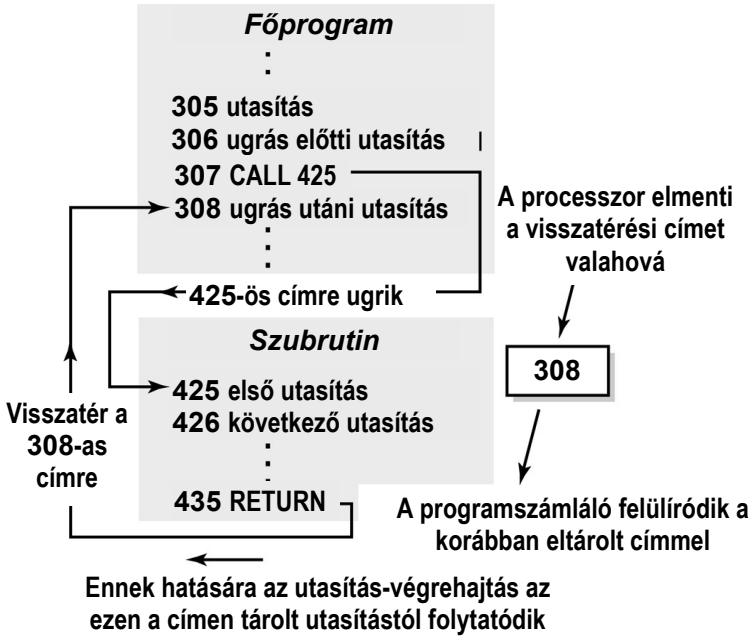
7.14. ábra Az aritmetikai és logikai eltolás valamint a forgatás műveletek végrehajtása

7.11.7. Összetett működésű program vezérlő utasítások

Az összetett működésű programvezérlő utasítások olyan speciális utasítások, melyek lehetővé teszik olyan programok futtatását, melyek részekre, ún. alprogramokra, szubrutinokra vannak feldarabolva. A szubrutin egy olyan programrészlet, mely a program végrehajtása során többször aktivizálódik, többször végrehajtódik, a főprogram szempontjából többször is meghívódik. A programvezérlő utasítások lehetővé teszik ezen programrészletek egyszerű meghívását.

Két utasítás tartozik ide, bár adott processzor esetén más lehet az elnevezésük. Mi CALL és RETURN utasításnak fogjuk nevezni őket. A

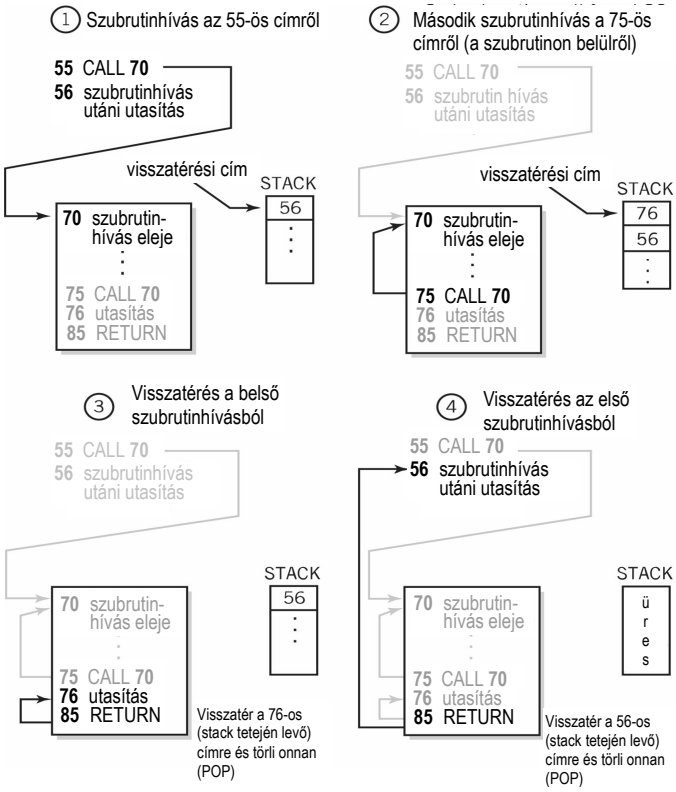
CALL utasításnak egy operandusa van. A processzor a CALL végrehajtásakor elmenti a CALL utasítás címét, majd elugrik az oprerandus által definiált helyre, ahol a szubrutin kezdődik. A RETURN utasítás hatására, amely a szubrutin végén helyezkedik el a processzor a CALL utasítás utáni utasításra ugrik. Ezért kell elmentenie a CALL utasítás pozícióját. A szubrutin hívó utasítások működését a 7.15. ábra mutatja.



7.15. ábra A CALL és a RETURN utasítások végrehajtása

A CALL és a RETURN utasítások végrehajtásához a processzornak szüksége van egy speciális működésű tárra, melyet veremtárnak (stack) neveztek el. Ez egy nem címezhető, hanem ún. FILO (First In Last Out) logikával működő tár, amit ha olvasunk mindig a legutoljára elmentett értéket fogja visszaadni. Erre a tárra azért van szükség, mert egymás után több CALL utasítás is szerepelhet egy programban anélkül, hogy lenne RETURN utasítás, és a processzor rákényszerül, hogy több CALL utasításhoz tartozó visszatérési címet is párhuzamosan tároljon, sőt azt is tudja mindig, hogy melyik CALL utasításhoz melyik visszatérési cím tartozik.

A CALL utasítás és a veremtár működését a 7.16 ábra mutatja be.



7.16. ábra A CALL és a RETURN utasítások végrehajtása veremtár segítségével

7.11.8. Verem tár kezelő utasítások

A veremtár kezelésére a processzor közvetlenül is meghívható utasításokat is felkínál. Két ilyen utasítás van POP és PUSH. Ezekkel az utasításokkal a program regiszterek tartalmát tudja a veremtárba mozgatni (PUSH), ill. tudja a veremtárból kiolvasni (POP), vagyis a veremtárból törölni és a regiszterbe mozgatni.

A veremtár kezelését a processzor egy speciális regiszter segítségével valósítja meg amit veremtár mutatónak (Stack Pointer) nevezünk.

7.11.9. Összetett adatelemeken végzett utasítások

A modern multimédiás alkalmazások jellemzően nagy számításigényű alkalmazások. Ezeknél az alkalmazásoknál általában sok hasonló adaton – ami pl. egy képernyő tartalom leírását tartalmazza – kell elvégezni azonos típusú feldolgozó műveleteket.

Az összetett adatelemeken végzett utasításokat angolul Single Instruction, Multiple Data (SIMD, (magyarul: egy utasítás, több adat) utasításoknak nevezik. Az összetett adatelemeken végzett utasításokkal a processzor párhuzamosan (egy utasításban) képes végrehajtani egyszerű műveleteket regiszterek vagy memóriacellák csoportján.

A közelmúltban ezeket az utasításokat a PC-kben használt processzorok is tartalmazzák. Az Intel cég a Pentium III processzoroktól kezdve kiegészítette processzorainak utasításkészletét ún. MMX™ utasításokkal, ami nem más, mint 57 multimédiás funkciót támogató utasítás. A későbbi modellek már tartalmaznak SIMD lehetőséget lebegőpontos és egész típusú adatok feldolgozására is.

A SMID utasításokat a PC-kben való megjelenésük előtt is alkalmaztak elsősorban nagyteljesítményű szerverekben, ahol a fő processzor mellett egy segéd, ún. coprocesszort is lehetett használni. Ez a processzor alkalmas volt ún. vektor feldolgozásra, vagyis SMID utasítások végrehajtására. Ezekben a nagyteljesítményű szerverekben a multimédiás alkalmazások mellett elsősorban kódolásra, szöveg- és képfeldolgozásra, szöveggenerálásra használták ezeket a processzorokat, ahol az adatokat vektorokban, vagyis adattömbökben lehet tárolni és feldolgozni.

8. Modern megoldások a CPU és a memória hatékony működésének megvalósítására

A processzorok műveletvégző sebességének meggyorsítása a számítógép tervezők egyik legfontosabb célja. Ez nem csak az elektronikai gyártási technológia fejlesztését jelenti, hanem olyan működési módok, ill. processzor-architektúrák kidolgozását, amelyek lehetővé teszik például a processzorutasítások párhuzamos, vagy időben átlapolt végrehajtását, vagy a memóriában tárolt adatok gyors elérését. Ezeknek a megoldásoknak az alkalmazása a felhasználó számára az alkalmazások látványos gyorsulását eredményezheti. Ebben a fejezetben ilyen, a processzor, ill. az egész számítógép utasítás végrehajtási gyorsaságát növelő megoldásokkal foglalkozunk.

8.1. CISC és RISC architektúrák

A korábban, nagyjából a nyolcvanas évek vége előtt tervezett klasszikus felépítésű, nagyteljesítményű processzorokat az ún. CISC (Complex Instruction Set Computer) utasításkészletű processzorok közé soroljuk.

Az elnevezés tükrözi a processzor tervezők akkori törekvését mind nagyobb számítási kapacitású CPU-k készítésére. Abban az időben a processzorok kapacitásának legnagyobb korlátját abban látták, hogy mivel a CPU utasításkészletét a processzor megvalósítására rendelkezésre álló korlátozott számú tranzisztor limitálja, ezért a programok számára nincs elég „bonyolult” működést biztosító utasítás. Abból indultak ki, hogy ha egy-egy új processzor utasításkészletébe egyre több összetett utasítást tesznek bele, mely összetett utasításban elvégzett műveletet a régebbi processzor csak több utasítással tudta végrehajtani, akkor az új utasításokat használó alkalmazások gyorsabban fognak majd lefutni.

Ennek a viszonylag hosszú fejlődésnek az eredményeként kifejlesztett CISC processzorok utasításkészletére a következők a jellemzők:

- Kisszámú általános célú regiszter.
- Több különböző címzési mód használata az operandus definiálására.
- Nagyszámú speciális, összetett funkciókat is megvalósító utasítások.
- Változó méretű utasítás kódok.

CISC processzor például az Intel x86 családja vagy az IBM Z sorozatába tartozó processzorok.

A processzortervezők várakozásai nem teljesen igazolódtak be. A CISC típusú processzorok gyakorlati használata során a következő problémák jelentkeztek:

- A nyolcvanas évek végén tanulmányok kimutatták, hogy az alkalmazások programjainak túlnyomó részét (általában több mint 70%-át) az egyszerű memóriakezelő (LOAD, STORE) és elágazás utasítások (BRANCH) teszik ki. Ezeknek a gyakran használt utasítások végrehajtási idejének csökkentése tehetné lényegesen gyorsabbá a programvégrehajtást, nem pedig az új, bonyolult utasítások bevezetése.
- Az egy-egy processzorban előforduló bonyolult funkciókat megvalósító CISC utasításokat sem az assembly, sem a magas szintű programnyelven íródott programok nem használták, vagy csak igen ritkán. Tehát a processzorok összetett utasításai használatából adódó gyorsítási lehetőséget a programok a gyakorlatban nem használták ki. Ennek oka az assembly programok esetén elsősorban az volt, hogy egy-egy új processzor megjelenésekor újonnan bevezetett utasításokat a programozók nem ismerték vagy idegenkedtek a használatuktól. A magas szinten megírt programok esetén a fordító programok okozták a gondot: a fordító programokat nem, vagy csak részben tudták az egyes processzorokra optimalizálni.
- A „jól strukturált” programokban szereplő gyakori szubrutinhívások lényegesen lelassították a programfuttatást. Ennek az az oka, hogy a szubrutinhívások esetén a processzor regisztereit el kell menteni annak érdekében, hogy a szubrutinból történő visszatérés után változatlan állapotból lehessen továbbfolytatni a program végrehajtását. További lassító tényező szubrutin hívások használatakor a hívási paraméterek átadása, amely megint csak további memória kezelő utasításokat jelent.

Ezen problémák megoldására fejlesztették ki a RISC – Reduced Instruction Set Computer utasításkészletű processzorokat. Ezek a következő jellemzőkkel rendelkeznek:

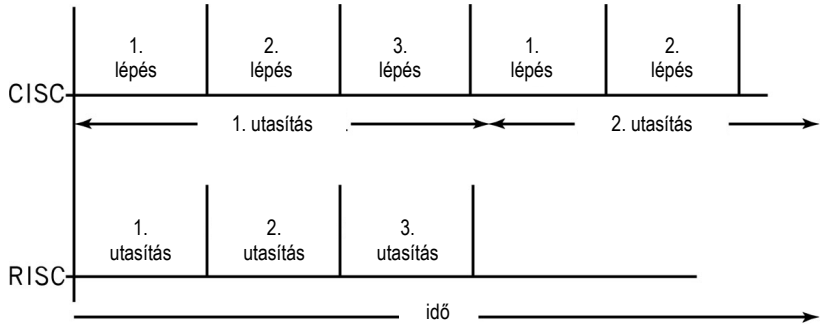
- Korlátozott utasítást tartalmazó és a CISC utasításokhoz képest egyszerű utasításkészlet.
- Fix hosszúságú, meghatározott formátumú utasítás szavak.

- Ún. pipeline feldolgozás lehetősége, mely alkalmazása esetén egy utasítás fetch fázisa párhuzamosan tud végrehajtódni az előtte levő utasítás execution fázisával. (A pipeline feldolgozásról ebben a fejezetben részletesen fogunk beszélni.)
- Korlátozott számú címzési mód.
- Jellemzően regiszter-orientált utasításkészlet, vagyis olyan utasítások használata, melyek operandusai a regiszterekben vannak.
- Nagyszámú regiszter a CPU-ban.
- Hatékony megoldások a szubrutin hívások végrehajtására.

A processzor utasításokat végrehajtó részét az utasításkészlet csökkentése egyszerűbbé tette. Az így felszabaduló tranzisztorok felhasználásával további regisztereket lehetett a processzorokba integrálni. A nagyszámú regiszter használata csökkentette a programok futtatása során szükséges memória-műveletek számát.

RISC típusú processzor van a Power PC-ben, ilyen a Sun gépekben használt Sparc processzorok, és a Motorola 68000-es processzorai.

A RISC típusú processzorok tehát kevés számú, és a CISC típusú processzoroknál rövidebb végrehajtási idejű utasítást tartalmaztak. Ha összehasonlítunk egy hasonló technológiával készült CISC és egy RISC típusú processzort, akkor azt látjuk, hogy azonos feladat megoldásához általában a CISC processzornak kevesebb utasítást kell végrehajtania, azonban a RISC processzor utasításai rövidebbek. Ezt a sematikus összehasonlítást a 8.1. ábra mutatja.



8.1. ábra Utasítás végrehajtás CISC és RISC processzorok esetén

Az első RISC típusú processzorok megjelenésekor még viszonylag éles határ volt a RISC és a CISC processzorok között. Mára ez a határ kezd

egy kicsit elmosódni. A ma használt processzorok általában RISC típusúak – elsősorban a pipeline utasítás-feldolgozás lehetősége miatt –, viszont egyes tulajdonságaik (pl. címzési módok, vagy speciális utasítások megléte) a CISC processzorok jellemzőit mutatják.

8.1.1. Körkörös regiszter tömb

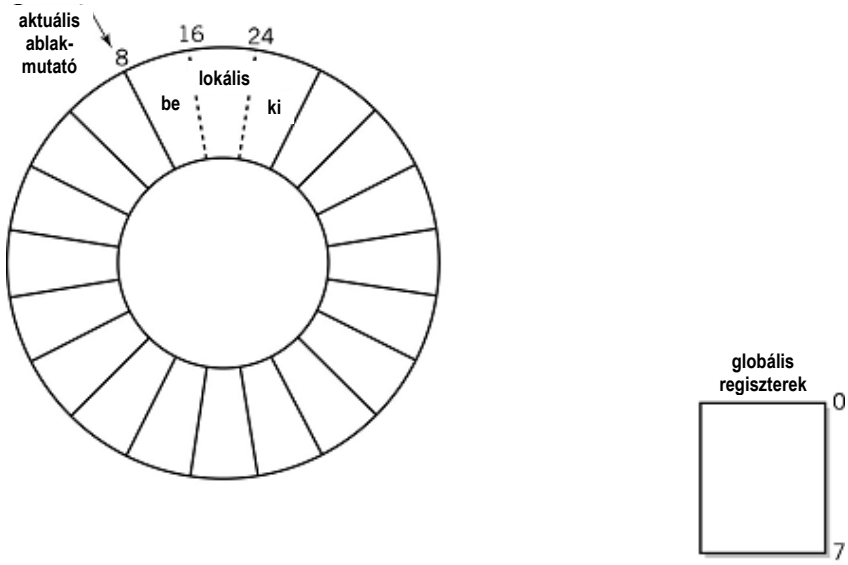
A RISC processzorok egyik fontos lehetősége a nagyszámú regiszter használata. A CISC processzorok esetén nyolc-tizenhat általános célú regiszter volt az általános. RISC-ek esetén lényegesen többet tudtak a processzorok tervezői a processzorba integrálni. A kérdés az volt, hogyan lehet ezt a sok regisztert hatékonyan felhasználni.

Az egyik legnagyobb probléma, hogy ha pl. 128 regisztert akarunk a processzor utasításaival kezelni, akkor az utasítások operandust definiáló részében egy adott regiszter kiválasztására $\log_2(128)=7$ darab bitet kell használnunk. Ez, főleg a kétoperandusú utasítások esetén túl sok.

A nagyszámú regiszter használata okozta problémák megoldására ad lehetőséget az ún. körkörös regiszter tömb (circular register buffer) használata, mely egyben hatékonyan felgyorsítja a szubrutinhívások esetén szükséges paraméterátadás folyamatát is.

A körkörös regiszter tömb nem más, mint a regiszterek okos szervezése. A regiszterek közül elkülönítünk nyolc darab ún. globális regisztert. Ezeket a globális regisztereket a processzor állandóan el fogja tudni érni. A processzor egy adott állapotában – a program futtatás egy adott pillanatában – a globális regisztereken kívül eső többi regiszter közül csak néhányat fog tudni elérni.

A processzorban levő globális regisztereken kívül eső többi regisztert nyolc elemű regisztertömbökbe szervezzük. Egy adott pillanatban a processzor ezekből mindig három tömböt, vagyis 24 regisztert lát. Azt, hogy a processzor melyik három nyolc elemű regisztertömböt látja egy speciális célú regiszter az ún. window pointer-ben ([regiszter] ablak-mutatóban) tárolt érték fogja meghatározni. Ennek az ablak-mutatónak az értékének a növelésével a nyolcelemű regisztertömböknek más és más hármasát tehetjük elérhetővé (lásd 8.2. ábra).

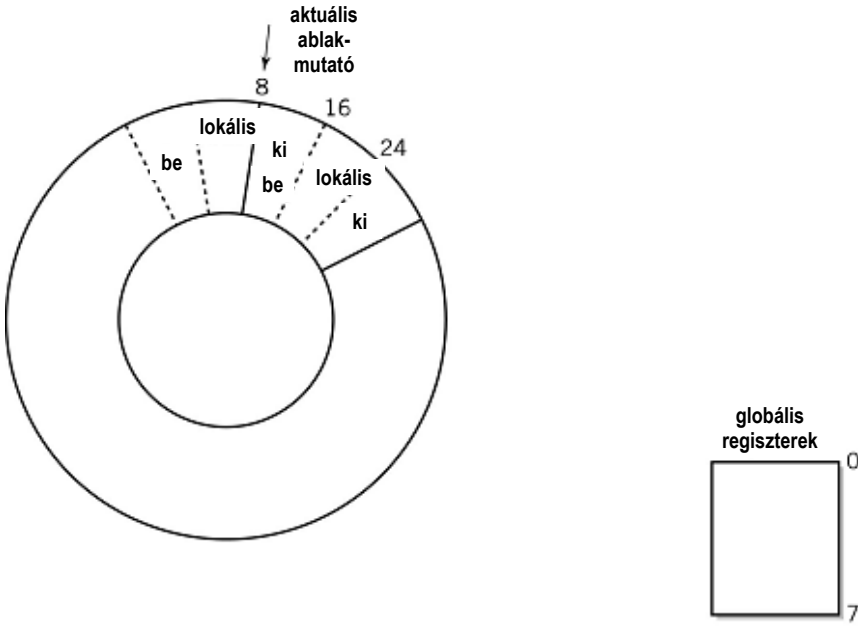


8.2. ábra Körkörös regiszter tömb felépítése

A processzor a processzor által egy adott pillanatban használható három nyolcelemű regisztertömböt a különbözőképpen használja. A középső nyolc regiszterben az aktuálisan végrehajtott programrészlet (szubrutin) végrehajtásához szükséges adatokat tárolja. Ezeket lokális regisztereknek nevezzük. Ha egy szubrutint kell meghívni, akkor a meghívandó szubrutin számára átadandó adatokat a program a harmadik nyolc elemű regisztertömbbe rakja.

Az első nyolcelemű regisztertömb használatának megértéséhez nézzük meg, mit tesz a processzor egy szubrutin meghívásakor. Ekkor a processzor úgy állítja be az ablak-mutató értékét, hogy két nyolc elemű regisztertömbbel elcsúsztatja azt (lásd 8.3. ábra). Ekkor a processzor az első nyolcelemű regisztertömb helyén a szubrutinhívás előtti állapotban utolsó helyen levő nyolcelemű regisztertömböt fogja látni. A mutató átállításával két korábban nem használt nyolcelemű regisztertömb válik elérhetővé, ezeket a processzor a második és harmadik helyen fogja látni.

Az ablak-mutató átállításával a meghívott szubrutin az első nyolcelemű regisztertömbben ott fogja találni a szubrutint meghívó programrész szubrutinnak átadni szándékozott adatait. Láthatjuk, hogy az első nyolcelemű regisztertömb tehát mindig az adott szubrutinnak átadni szándékozott (bemeneti) adatokat tartalmazza.



8.3. ábra Körkörös regiszter tömb kezelése szubrutin hívás alkalmával

8.2. Párhuzamos utasítás végrehajtásra alkalmas megoldások

A processzorok utasítás végrehajtásának felgyorsítására leginkább a párhuzamos utasítás végrehajtás alkalmas. A párhuzamos utasítás végrehajtás elve alapján több processzor architektúra is született. Ezeket közös néven instruction-level parallelism (ILP, utasítás szintű párhuzamosítás) megoldásoknak nevezzük.

Ebben a fejezetben két párhuzamos utasítás végrehajtást propagáló megoldást mutatunk be:

- VLIW – Very Long Instruction Word,
- EPIC – Explicitly Parallel Instruction Computer.

A gyorsítás módja mindkét esetben a normális esetben szekvenciálisan végrehajtandó utasítások párhuzamos végrehajtása. Természetesen ez nem mindig lehetséges. Vannak esetek, amikor egy adatfeldolgozó utasítás használja az előtte levő utasítás eredményét. Ekkor az utasítások esetén adatfüggésről beszélünk.

Az utasítások párhuzamos végrehajtásának másik korlátozója az ún. vezérlésfüggés. Egy feltételes elágazás utasítás esetén az időben következő végrehajtandó utasítás függ az előző utasítás eredményétől.

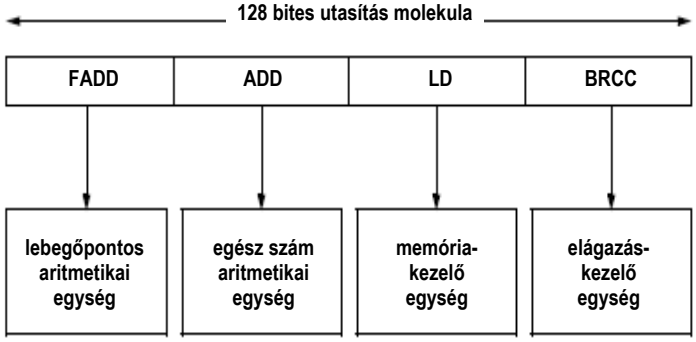
A párhuzamos utasítás végrehajtást lehetővé tevő architektúráknak ezeket a problémákat kezelni kell.

8.2.1. Very Long Instruction Word (VLIW)

A Very Long Instruction Word (VLIW – Nagyon Hosszú Utasítás Szó) processzor architektúrát először Josh Fisher javasolta a nyolcvanas évek elején. A megoldás lényege az, hogy a hagyományos, egymás után végrehajtandó program utasításokat szervezzük át utasítás csoportokba úgy, hogy az egyes csoportokban levő utasításokat egymással párhuzamosan lehessen végrehajtani.

A nyolcvanas évek közepétől több cég is foglalkozott VLIW technológiára alapuló processzor készítésével. A Transmeta cég Crusoe CPU családja egy kereskedelemben elérhető VLIW technológiára alapuló processzor.

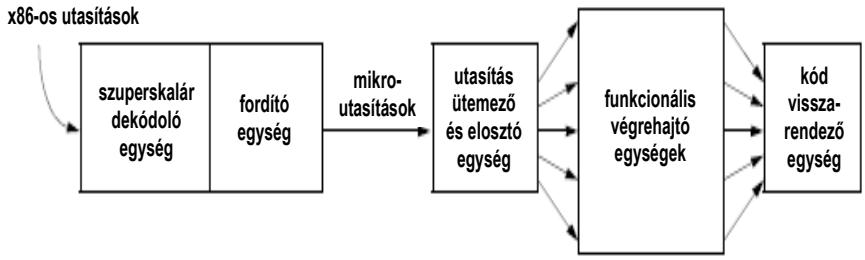
A processzor 128-bites utasítás csomagokat, ún. utasítás molekulákat tud végrehajtani. Egy molekula 4 db 32-bites atomból (elemi utasításból) áll. A molekulában levő 4 elemi utasítást párhuzamosan hajtja végre a processzor. A processzorban a párhuzamos utasítások számára 64 általános használatú regiszter van.



8.4. ábra VLIW utasítás felépítése

A VLIW processzorhoz készítettek egy ún. kód átalakító réteget (code morphing layer/code morphing software), mely a más CPU-k assembly utasításait tartalmazó kódot utasítás molekulákra tudja fordítani. Így a Transmeta Crusoe CPU-ját tartalmazó számítógép képes az Intel x86-os

assemblyutasítás sorozatokat futási időben átfordítani VLIW utasítás molekulákká. A kód átalakító réteg a VLIW utasítás molekulákat úgy fogja összeállítani, hogy az egy molekulába kerülő utasítások párhuzamosan végrehajthatóak legyenek. A párhuzamosítás, a kód optimalizálása párhuzamos végrehajtásra, utasítások függőségének vizsgálata tehát a processzor környezet (a kód átalakító réteg) feladata. A Transmeta cég Crusoe CPU-ját tartalmazó számítógépre tehát nem kell (a CPU által közvetlenül végrehajtható) utasítás molekulákból álló programot írni, hanem az Intel x86-os assemblyutasításokból álló programot a gép automatikusan átfordítja és végrehajtja (8.5. ábra).



8.5. ábra VLIW utasítás végrehajtása Transmeta Crusoe CPU-ban

8.2.2. Explicitly Parallel Instruction Computer (EPIC)

Az EPIC (Explicitly Parallel Instruction Computer – Közvetlenül Párhuzamosított Utasításkészletű Számítógép) a modern processzorok gyorsítását célzó általános processzor-szerszervezési elv, az utasítás szintű párhuzamosítás területén folyó kutatás-fejlesztés eredményeként született, gyakorlatban alkalmazott megoldás.

EPIC architektúrájú processzorok fejlesztésével elsősorban az Intel és a Hewlett-Packard (HP) cégek foglalkoztak, ennek eredményeként született meg az IA-64 architektúra és az Itanium, ill. az Itanium-2 processzorok.

Az IA-64 architektúrájú processzorok utasításai 128-bites utasítás csomagok. Ebben három darab 41-bites utasítás (összesen 123 bit) van. A maradék 5 bit határozza meg a csomagban levő utasítások típusát.

Az Itanium processzoroknak 128 db 64-bites általános használatú regisztere van és 128 db 82-bites lebegőpontos regisztere.

Az Itanium processzorokat alkalmassá tették a hagyományos Intel x86-os utasítások végrehajtására is, így képesek a hagyományos Intel processzorokra készült programokat is lefuttatni.

A processzor kifejlesztői a programozók és a fordítóprogramok írói számára a párhuzamos utasítás-végrehajtást lehetővé tevő utasítás csomagok használatára ajánlásokat dolgoztak ki a közeli utasítások függőségének kiküszöbölésére. Ezek betartása lehetővé teszi, hogy a programok hagyományos utasítások sorozata helyett az IA-64 architektúra utasításcsomagjait használják, biztosítva ezzel az EPIC architektúra párhuzamos utasítás-végrehajthatási lehetőségének kihasználását.

A VLIW és az EPIC architektúrát összehasonlítva megállapíthatjuk, hogy mindkét megoldás hasonló módon, az utasítások párhuzamos végrehajtásával gyorsítja fel a processzor működését. Mindkét processzor új típusú utasításkészlettel rendelkezik, melyben egy utasítás több hagyományos utasításnak felel meg, és ezeket párhuzamosan hajtják végre a processzorok. A különbség az, hogy a VLIW architektúra „bármilyen" hagyományos utasítássorozatot végre tud hajtani, az „optimalizálás" (függőségek vizsgálata) a processzor (ill. a processzor környezet) feladata. Az EPIC megoldás esetén a processzor már „optimalizált" utasítás sorozatot hajt végre, az „optimalizálás" (függőségek vizsgálata) a programozó (ill. a program fejlesztő környezet, a fordító program) feladata.

8.3. Memória használata modern számítógépekben

8.3.1. Lapszervezés

A modern számítógépek memória használata kapcsán meg kell ismerkednünk a memória címtranszformáció fogalmával és a hozzá kapcsolódó memóriakezelési módszerekkel. Ezek jelentőségét igazán az ún. multiprogramozott – több programot párhuzamosan futtatni képes – operációs rendszerek működésének megismerésekor fogjuk megérteni, de mivel a processzor támogatást ad a memória címtranszformációhoz itt kell róluk beszélnünk. A címtranszformációhoz közvetlenül kapcsolódó fogalmak a fizikai és logikai memóriacím, lapkezelésű, ill. szegmens kezelésű memória. A virtuális memóriakezelés – már itt kiemeljük – ugyan távolról kapcsolódik a témához, de nem keverendő össze a lapkezelésű, ill. szegmens kezelésű memória használattal.

A processzor, ill. a számítógép a memória címtranszformáció alkalmazása esetén két különböző típusú memóriacímet fog megkülönböztetni, azokat egymástól függetlenül kezelni. Megkülönbözteti az ún. logikai memória címeket, melyek egy programon belül határozzák meg egy adat vagy utasítás a program más részeitől számított pozícióját az ún. fizikai memóriacímektől, melyek viszont az adatok és utasítások konkrét helyét határozzák meg a fizikai memóriában.

A programok futtatható kódjában a logikai memóriacímek szerepelnek. A memória kezelő (író/olvasó) utasítások végrehajtása során a processzor, ill. a processzort a címtranszformációban támogató komponens, az ún. Memory Management Unit (MMU, Memória Kezelő Egység) konvertálja a logikai memóriacímeket fizikai memóriacímekké. Ezt a memóriacím konvertálást nevezzük címtranszformációnak.

A címtranszformáció lényegében megadja két címtartomány címeinek egymáshoz rendelését. Angolul ezt a megfeleltetést mapping-nek nevezzük.

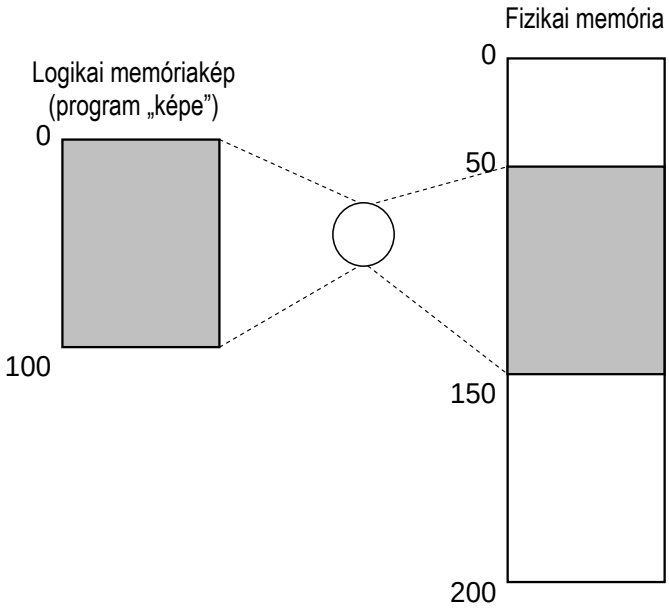
Beszélhetünk a logikai és fizikai memóriacímek „címtereiről", amely címtereken belül érvényesek a memóriacímekről tanultak, tehát a címek egyediek, és egymáshoz képesti elhelyezkedését mutatják az egyes memóriatartalmaknak. A két címtér címeit azonban nem szabad összekeverni, a köztük levő megfeleltetést a címtranszformáció adja.

A 8.6. ábra egy egyszerű címtranszformációt mutat. Egy program 0..100 logikai címeit megfeleltetjük a fizikai memória 50..150 fizikai címeinek. A megfeleltetés teljesen kötetlen – ez az egyik legfontosabb előnye a címtranszformáció alkalmazásának –, tehát nem feltétlenül kell folytonos logikai címtartománynak folytonos fizikai címtartományt megfeleltetni. Erre a 8.7. ábra ad példát.

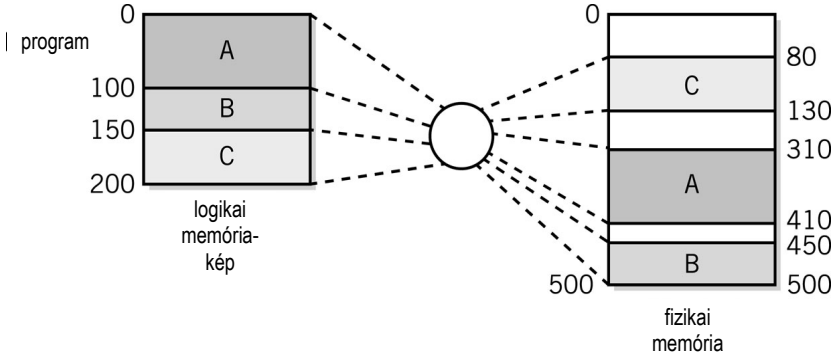
A címtranszformáció előnye több program párhuzamos futtatásakor látható. Ekkor minden program saját logikai címteret használ (ezt a számítógépet irányító operációs rendszer biztosítja), így minden program használhat azonos logikai címeket, a címtranszformáció eredményeként kapott fizikai címek lesznek különbözőek.

Lapszervezésű memóriakezelésnek, röviden lapozásnak nevezzük a memória címtranszformáció használatát azzal a megszorítással, hogy mind a fizikai, mind a logikai címtartományokat azonos méretű darabokra osztjuk. Ezeket a darabokat memória lapoknak nevezzük (angolul memory page), a módszer innen kapta az elnevezését. Lapkezelés esetén a logikai címtér és a fizikai címtér lapjait feleltetjük meg egymásnak. Ennek több előnye van, most ezek közül csak egyet említünk. Lapszervezés esetén a

memória címtranszformációhoz szükséges logikai-fizikai cím-összerendelő táblázata viszonylag egyszerű, csak az összetartozó logikai-fizikai címtartományok (lapok) kezdőcímeinek összerendelését kell eltárolni.

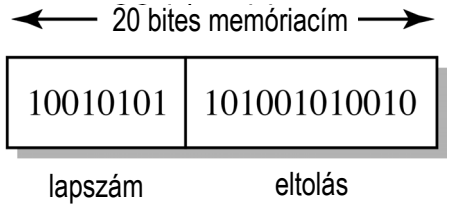


8.6. ábra Logikai és fizikai memóriacímek
és azok megfeleltetése egymásnak



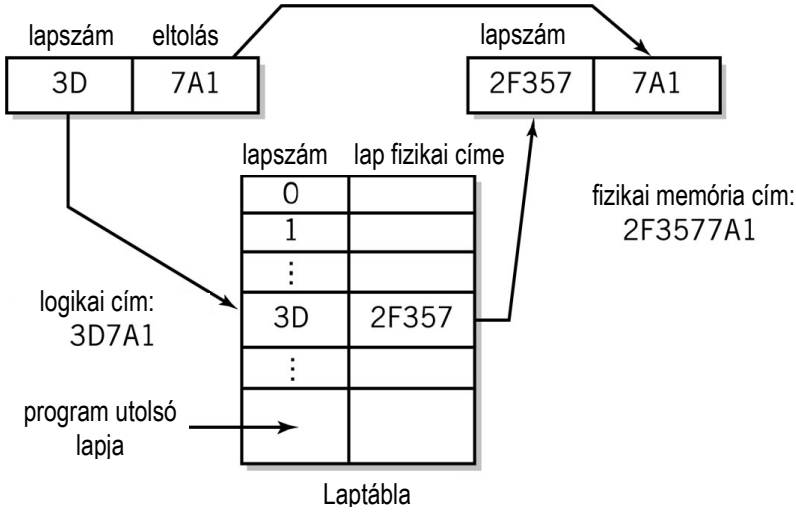
8.7. ábra Logikai és fizikai címtranszformáció nem folytonos
memóriatartományokkal

Ha a memória lapok méretét ügyesen választjuk, bináris memóriacímek esetén a lapméret kettő hatvány darab memóriarekeszt tartalmaz, akkor a címtranszformáció művelete is egyszerű lesz. A memóriacímeket két részre, ún. lapcímre és azon belüli eltolásra oszthatjuk. A lapcím mindig az egyes lapok kezdőcímét adja meg (lásd 8.8. ábra).



8.8. ábra Memóriacím felépítése lapszervezés esetén

A logikai és fizikai lapcímek egymásnak történő megfeleltetését egy ún. címfordító táblázatban (laptábla) tároljuk. A logikai-fizikai címtranszformáció ezek után egyszerűen a logikai cím lapcímének fizikai lapcímre történő lecserélését jelenti (lásd 8.9. ábra).



8.9. ábra Logikai és fizikai címtranszformáció menete

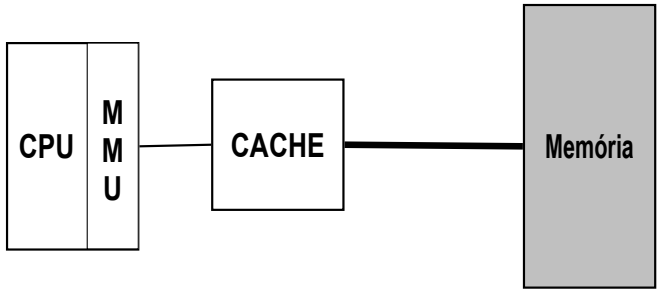
8.3.2. Cache

A memóriában tárolt adatok elérésének meggyorsítására a számítógépekben általánosan alkalmazott megoldás a cache (gyorsítótár). A cache maga

is egy memória, azonban a normál memóriánál sokkal (legalább egy nagyságrenddel) gyorsabb elérésű, vagyis a benne tárolt adatokat a CPU jóval gyorsabban tudja írni és olvasni, mint a memóriában tárolt adatokat.

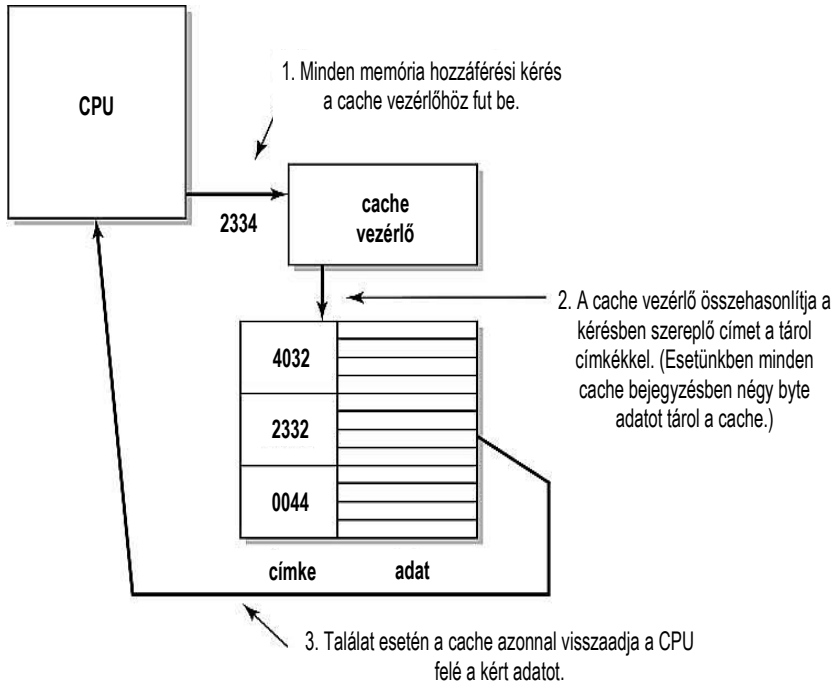
A cache memória szervezése különbözik a hagyományos memória szervezésétől. A cache egy asszociatív tömb. Minden memóriarekeszhez tartozik egy címke (tag). A cache memóriarekeszeinek tartalmát a címkéken keresztül érhetjük el. Írhatunk és olvashatunk egy-egy címkéhez tartozó memóriarekeszt.

Működését tekintve a cache a CPU és a memória közé épülő tároló (8.10. ábra). A számítógép működése során a cache a memória egy részének tartalmát tükrözi, duplikálja. A címkék a tükrözött memória rekeszek címét tartalmazzák, a cache memória rekeszei pedig az adott címkében tárolt memóriacímen tárolt adattartalmat.



8.10. ábra A cache kapcsolata a CPU-val és a memóriával

A cache működését a cache vezérlő szabályozza. A cache vezérlő minden esetben, amikor a CPU a memóriához fordul, megkapja a keresett memóriacímet. Abban az esetben, ha a CPU által keresett memóriarekesz a cache-ben is megtalálható, a rendszer nem hajtja végre a memória hozzáférést, hanem a cache-ben tárolt „másolatot” fogja használni (8.11. ábra). Az ilyen szituációkat cache találatnak (angolul hit) nevezzük. Mivel a cache elérése lényegesen gyorsabb a hagyományos memóriánál, ez a processzor művelet végrehajtási sebességének növekedését fogja eredményezni.



8.11. ábra A cache működése cache találat esetén

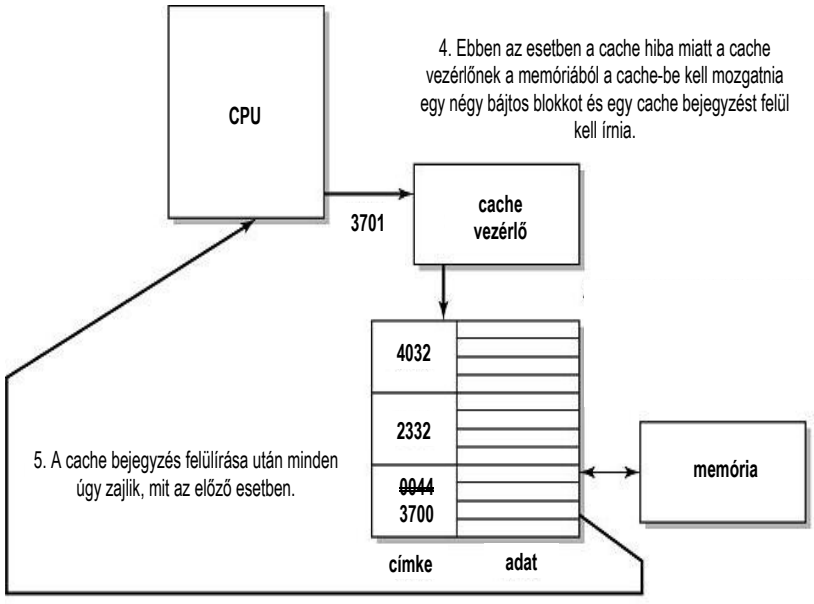
Ha a cache vezérlő nem találja a memória írás, ill. olvasás utasítás által érintett memóriarekesz tartalmát a cache-ben – ezeket az eseteket cache hibának (angolul miss) nevezzük –, akkor végre fog hajtódni a memória írás, ill. olvasás, úgy, ahogy ezt korábban az utasítás-végrehajtás kapcsán leírtuk. A cache vezérlő ekkor figyeli a memória és CPU közötti adatforgalmat a bus-on, és a cache memóriának valamelyik rekeszébe elmenti az írt vagy olvasott memóriacellát a memória címével együtt (8.12. ábra).

A cache vezérlő feladata igen bonyolult és nagymértékben befolyásolja az egész számítógép teljesítményét. A következő feladatokat kell megoldania:

- Ha cache hiba esetén nincs hely a cache memóriában az új memóriatartalom tárolására, ki kell jelölnie egy használt cache rekeszt a memóriából olvasott új memóriatartalom tárolására. (Ha a kijelölt cache rekesz még a memóriába el nem mentett változtatásokat tartalmaz, azt vissza kell írnia a memóriába.) A cache az új memóriatartalom tárolási helyének kiválasztásakor általában a legrégebben nem használt (Least Recently Used) algoritmust igyekszik követni, vagyis azt a cache re-

keszt próbálja kiüríteni, amelyet a legrégebben nem írtak vagy olvastak. Ennek a működésnek a megvalósítására a cache különböző, az adatok használatára vonatkozó információt is tárolhat.

- A cache vezérlőnek döntenie kell, hogy ha a CPU egy cache-ben tárolt rekeszt ír, akkor a változtatást azonnal az íráskor, vagy valamikor később, az adott cache tartalom cache-ben új tartalommal történő felülírásakor fogja a memóriában levő változaton is érvényesíteni. Az előbbi stratégiát write through (keresztül írás) cache működésnek, míg ez utóbbit write back (vissza írás) cache működésnek nevezzük. Mindkét stratégiát használó cache-re találunk példát a számítógépekben.



8.12. ábra A cache működése cache memóriából történő olvasás esetén

A cache memória használata lényegesen – gyakran több mint 50%-kal – megnövelheti egy-egy program végrehajtási sebességét. Ennek oka az, hogy a cache találati arány (hit ratio) egy működő számítógépen 90% körül mozog, vagyis a memória hozzáférések kb. 90%-a esetén a keresett tartalom megtalálható lesz a cache-ben. Ez akkor igazán meglepő szám, ha tudjuk, hogy a központi memória és a CPU között gyakorlatban alkalmazott cache mérete között kb. ezres szorzó van, tehát a cache csak a memóriatartalom durván egy ezrelékét tárolja (tükrözi).

A cache ilyen hatékony működésének az oka a programok ún. lokális természete. A lokális működés két dolgot takar:

- Ha egy adott memóriacellában tárolt utasítást vagy adatot a számítógép egyszer használ, akkor nagy esély van rá, hogy a közeljövőben megint fogja használni (időben lokális működés).
- Ha a számítógép egy adott memóriacellában tárolt utasítást vagy adatot egyszer használ, akkor nagy esély van rá, hogy a közeljövőben használni fogja a körülötte levő memóriacellákat (memória-térben lokális működés).

A programoknak ez utóbbi természetét a cache memória úgy használja ki, hogy egy alkalommal nem egy memória rekesz tartalmát mozgatja be a cache-be, hanem az elért rekesz mellett levő rekeszek tartalmát is.

8.3.3. Széles sávú adatelérés

A széles sávú adatelérés (wide path memory access) a modern számítógépekben gyakran alkalmazott megoldás, azonban az egész számítógép szervezését, architektúráját érinti.

A széles sávú adatelérés azt jelenti, hogy egy memória rekeszben nem egy vagy két bájt adatot, hanem négy, nyolc, vagy tizenhat bájtot tárolunk, vagyis egy-egy memória-hozzáférés alkalmával négy, nyolc, vagy tizenhat bájt adatok kezelünk. Alkalmazásához a memóriát a CPU-val összekapcsoló adat-bus-t, ill. a CPU-ban az adatokat kezelő MDR-t is megfelelő szélességűre kell készíteni.

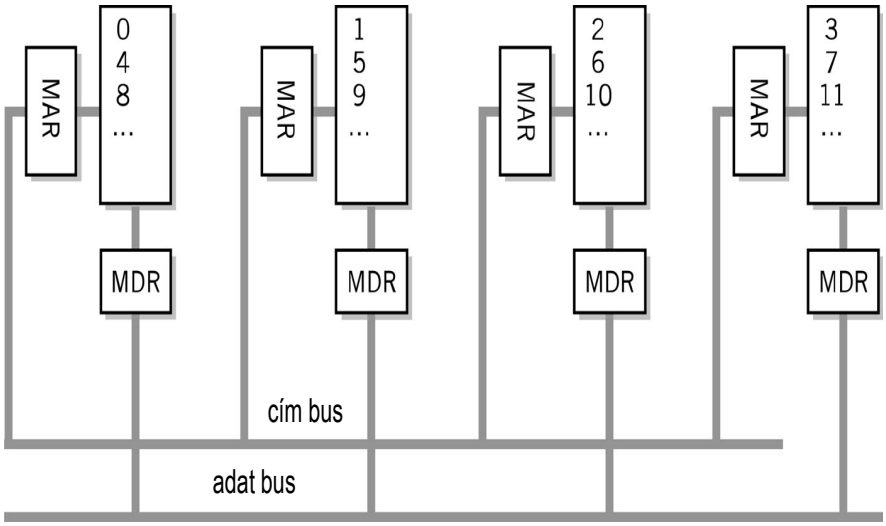
Előnye, hogy egy memória-hozzáférési ciklus alatt a CPU-ba jóval több adat kerül, tehát felgyorsulhat a feldolgozás. A megoldás hátránya lehet, hogy a négy, nyolc, vagy tizenhat bájt adatot nem minden művelet esetén lehet értelmes adatokkal kitölteni, vagyis a memória egy része kihasználatlan marad. A széles sávú adatelérést a mai számítógépek széles körben alkalmazzák, a 64 bites rendszerbusz ma már nem ritka.

8.3.4. Memory Interleaving

A processzor működését lassítja, ha egymás után sok memória elérést igénylő utasítás van. A CPU utasítás-feldolgozási ideje lényegesen kisebb, mint amennyi idő alatt a memóriából az adatok a processzor MDR regiszterébe jutnak, ezen úgy lehet segíteni, ha a processzorhoz több MAR és MDR regisztert kapcsolunk, amiket a CPU felváltva használ, így ezek párhuzamosan tudnak dolgozni, a memóriatartalmat a CPU-ba juttatni.

Az fenti elven működő memóriaszervezést memory interleaving (összefésült memória) szervezésnek nevezzük.

A memory interleaving használatához a memóriát – praktikus szempontok alapján azonos méretű – részekre kell osztanunk, melyeket különbö MAR és MDR regisztereken keresztül érünk majd el. Ezután meg kell határozni, hogy az egyes címeken elért memória rekeszeket melyik memória részekhez rendeljük, vagyis az adott címen elért tartalmat hol fogjuk tárolni. A gyakorlatban a leghatékonyabbnak – a legnagyobb utasítás végrehajtás gyorsulást eredményezőnek – az egymás utáni memóriacímek külön memóriarészekhez történő rendelése bizonyult. A 8.13. ábra a memory interleaving ilyen elven történő megvalósítását mutatja.



8.13. ábra Memory Interleaving

8.4. Pipeline utasítás-feldolgozás

A modern processzorokban az egyik legfontosabb módszer az utasítás végrehajtás meggyorsítására az ún. pipeline (futószalag elvű) utasítás végrehajtás. A pipeline feldolgozás az egyes utasítások időben árlapolt, párhuzamos végrehajtását jelenti.

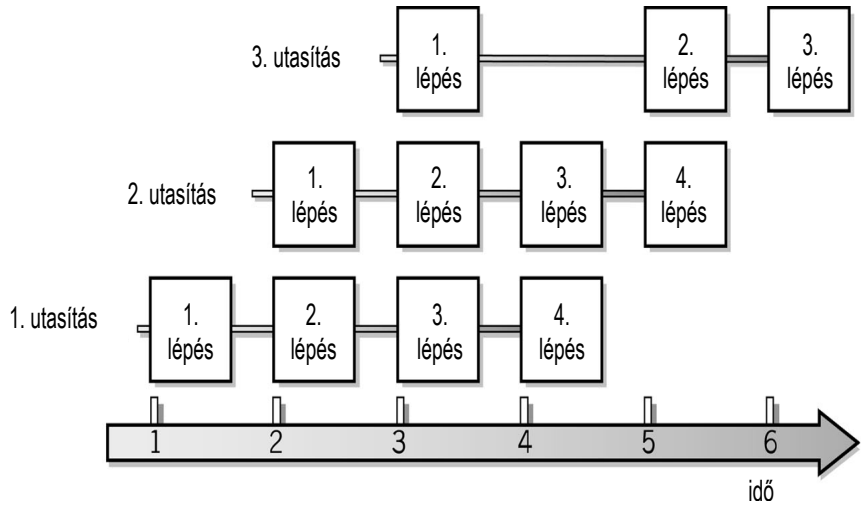
A pipeline utasítás végrehajtás esetén a processzor az utasítások végrehajtását lépésekre bontja. Az utasítások végrehajtásáról szólva már korábban is megkülönböztettünk két utasítás végrehajtási fázist, a fetch és az

execute fázisokat. A pipeline esetén a fetch és az execute fázisok mellett az utasításokat további lépésekre bontjuk.

A pipeline működés hatékony megvalósításához a következő feltételek teljesítésére van szükség:

- Az utasításokat olyan egymástól független lépésekre kell bontani, amely lépéseket a processzor közel azonos idő alatt képes végrehajtani.
- Az utasítás lépéseket a processzor egymástól függetlenül működő részei tudják végrehajtani. A lépések eredményét tárolni képes tároló elemek kössék össze egymással az egyes lépéseket végrehajtani képes processzor részeket.
- Az egyes utasítások végrehajtása lehetőleg azonos számú, vagy egymástól nem nagyon eltérő számú lépést tartalmazzon.

Ha ezek a feltételek teljesülnek, akkor – ideális esetben – megszervezhető a programok utasítássorozatainak pipeline végrehajtása. A processzor első utasításlépést végrehajtani képes része végrehajtja az első utasítás első lépését, majd az eredményét továbbadja a következő processzor résznek, amely késlekedés nélkül elkezdi a második utasítás első lépésének végrehajtását (lásd 8.14. ábra).



8.14. ábra Pipeline utasítás-feldolgozás

A pipeline utasítás-feldolgozás hasonló a futószalag mellett dolgozó munkások munkájának szervezéséhez: a munkások a processzor egyes részei-

nek feleltethetők meg. A munkadarabon a munkások által végzett munkafázisok az utasítás-végrehajtás lépései lesznek, az utasítások végrehajtása pedig a munkadarabok legyártása.

Kívülről nézve pipeline utasítás-feldolgozás esetén a processzor párhuzamosan fog annyi műveletet végrehajtani, ahány lépésre osztottuk az utasítások végrehajtását. Átlagosan – ideális körülményeket feltételezve – a processzor egy utasítást annyi idő alatt fog végrehajtani, ameddig egy utasítás lépés végrehajtása tart.

A pipeline utasítás-feldolgozás esetén, a párhuzamos feldolgozásból adódóan felmerül néhány probléma, hasonlóan, mint a VLIW és EPIC architektúrák esetén. A programok az egyes utasítások szekvenciális végrehajtását feltételezve íródnak, vagyis az egymás után következő utasítások között adat- és vezérlés-függés lehetséges.

A processzorok az adat-függést az utasítások végrehajtási sorrendjének átrendezésével próbálják megoldani.

A vezérlés-függést a pipeline processzorok az ugró utasítások eredményének megbecslésével (branch estimation) vagy több pipeline sor használatával próbálják megoldani. Ez utóbbi esetén a processzor lényegében több processzor megvalósítására képes elemet (áramkört) tartalmaz és mindkét lehetséges végrehajtási szálat egy-egy külön „belső” processzoron végrehajt. Ha kiderül, hogy ténylegesen melyik utasítássorozatot kell végrehajtani, a felesleges szálat (utasítás-sorozatot) végrehajtó „belső” processzort leállítja.

8.4.1. Skalár és szuperskalár processzorok

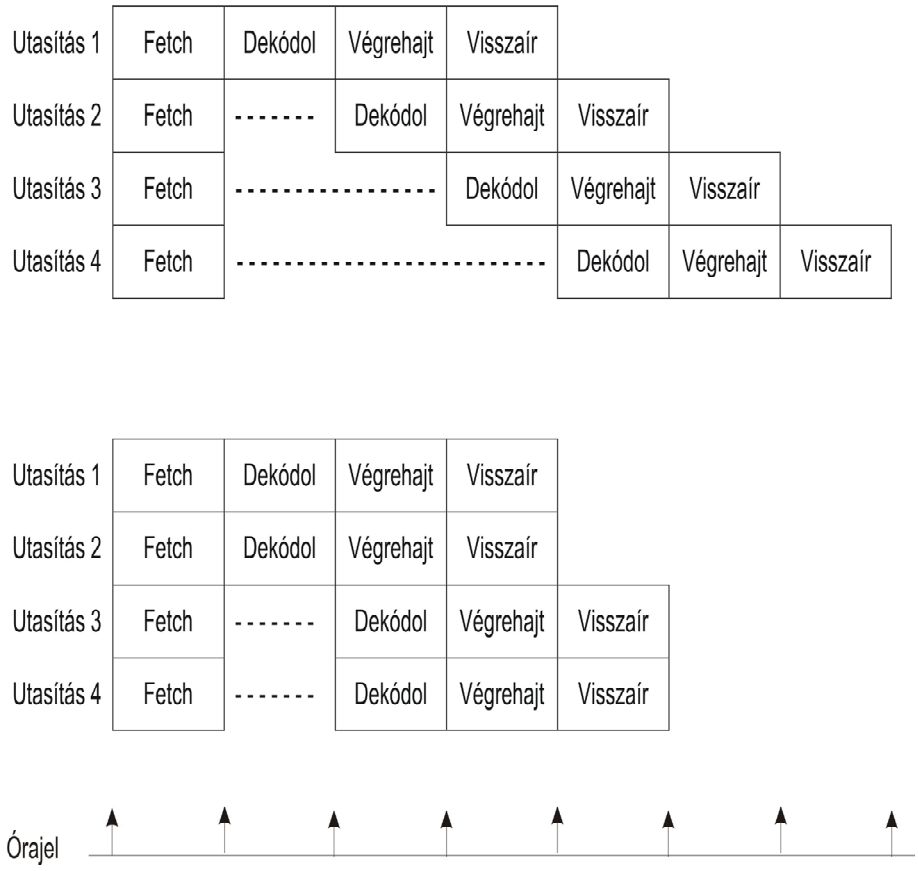
A processzorok az utasítások pipeline feldolgozás kapcsán megismert lépéseit általában egy CPU működését ütemező órajel-ciklus alatt képesek elvégezni.

Skalár processzornak nevezzük azokat a pipeline utasítás végrehajtásra alkalmas processzorokat, melyek teljesítik a pipeline utasítás-feldolgozás fent leírt követelményeit és átlagosan egy utasítást egy órajel-ciklus alatt képesek végrehajtani. Ez természetesen nem azt jelenti, hogy egy-egy utasítás egy órajel ciklus hosszú – az utasítások hossza processzortípustól erősen függ, nagyságrendileg általában 10 órajel ciklus körül mozog –, hanem azt, hogy a pipeline feldolgozás miatt így látszik.

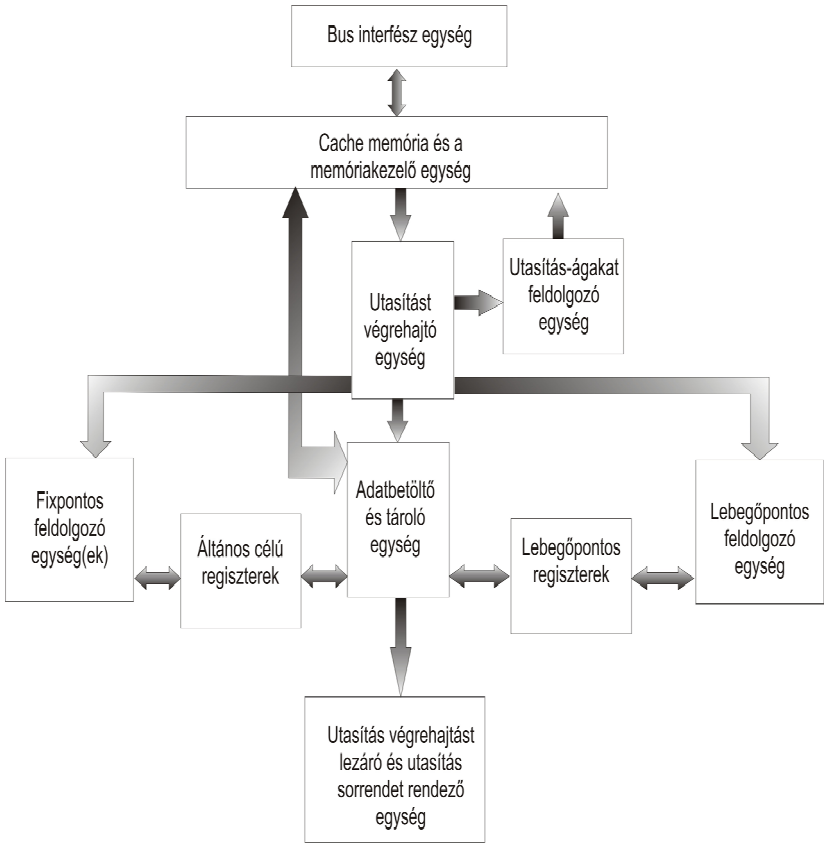
Szuperskalár processzoroknak nevezzük azokat a processzorokat, melyek egy órajel-ciklus alatt több mint egy utasítást képesek végrehajtani. Ez úgy lehetséges, hogy a szuperskalár processzorokban több processzornyi vég-

rehajtó elem van. Ezeknek az elemeknek a segítségével a processzor több pipeline sor kezel, ill. hajtja végre párhuzamosan a pipeline sorokba szervezett utasításokat. A 8.15. ábra a skalár és szuperskalár processzorok működését hasonlítja össze.

A modern pipeline feldolgozásra képes processzorok, melyek az utasítások adat és vezérlés függéséből adódó ütközéseket képesek kiküszöbölni igen bonyolult felépítésűek. A 8.16. ábra egy ilyen processzorra ad példát. Az adat függés kiküszöbölése érdekében a processzor képes az utasítások átrendezésére.



8.15. ábra A skalár és szuperskalár processzorok működése



8.16. ábra Pipeline működés megvalósítására optimalizált modern processzor felépítésének blokk diagramja

9. Input/output (bemeneti/kimeneti) eszközök kezelése és működése

9.1. I/O eszközök csatolása a számítógéphez

A számítógépekhez számos periféria csatlakozik, mely perifériákon keresztül adatokat lehet a számítógépbe bevinni, ill. melyeken keresztül a számítógép adatokat tud megjeleníteni.

Az egyes perifériák egymástól igen különböző sebességű és adatfeldolgozó kapacitású elemek, egy-egy periférián keresztül lebonyolított adatcsere gyakran bonyolult és hosszadalmas folyamat. A számítógép tervezők ezért a perifériákat nem közvetlenül a CPU-hoz kapcsolják, hanem egy ún. I/O modulon, I/O vezérlőn keresztül csatlakoztatják (lásd 9.1. ábra). A CPU az I/O modul számára fog parancsokat küldeni, azokat irányítja közvetlenül CPU utasításokkal, és az I/O modul lesz felelős az I/O művelet végrehajtásáért az I/O eszközön.

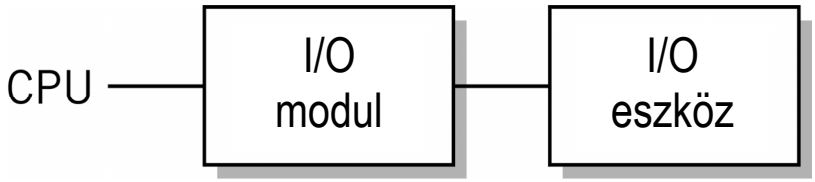
Az I/O modul funkciói a következők:

- Feldolgozza az általa vezérlet egység(ek)től érkező hozzá címzett üzeneteket, és elfogadja a CPU- tól az I/O kezelő parancsokat.
- A memóriából származó adatokat a pufferében átmenetileg tárolja, amíg azt az I/O eszköz fel nem dolgozza. A pufferből adatokat küld az eszköznek, ill. a CPU-tól adatokat fogad és azokat eltárolja a pufferében.
- Biztosítja a közvetlen memória elérés (DMA) megvalósításához szükséges regisztereket és vezérlő funkciókat. (A közvetlen memória elérésről a fejezetnek a második részében lesz szó.)
- Fizikailag vezérli az I/O eszközök működését.
- Megszakítások formájában jelzést küld a CPU számára az I/O folyamatok állapotáról.

Az I/O kezelés kapcsán a következő témákkal fogunk megismerkedni:

- Programozott I/O kezelés, amikor a CPU irányítja az I/O folyamatot.
- Megszakítás vezérelt I/O, amikor a számítógép a kívülről érkező adatokra reagál, a kívülről érkező események vezérlik az I/O eszközön folyó adatcsere folyamatát.

- DMA – Direct Memory Access (közvetlen memória-elérés) módszere, amikor az I/O eszköz és a memória között a CPU közreműködése nélkül történhet adatátvitel.



9.1. ábra I/O eszközök csatolása a CPU-hoz

9.2. Programozott I/O kezelés

Programozott I/O kezelésről akkor beszélünk, ha az I/O eszköz működését a CPU irányítja.

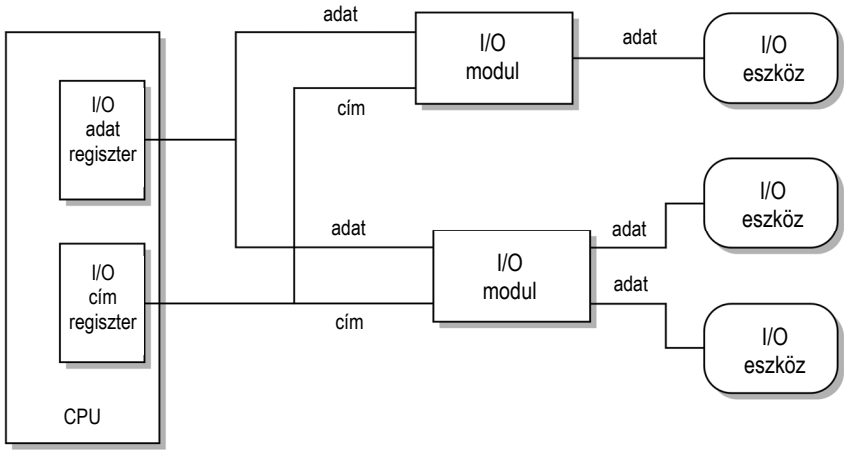
A processzorban vannak I/O adat és cím regiszterek, hasonlóan, mint a memória elérésére szolgáló MAR és MDR regiszterek. Az összes I/O eszköznek van egyedi címe, melyeket a CPU használ. A memória olvasáshoz és íráshoz hasonlóan az I/O eszközöket is tudja a CPU, egy-egy arra szolgáló utasítással írni és olvasni (lásd 9.2. ábra).

A programozott I/O kezelés lépései igen egyszerűek és hasonlóak a memóriaolvasás megvalósításához. A 9.3. ábra és a 9.4. ábra ezeket a lépéseket mutatja.

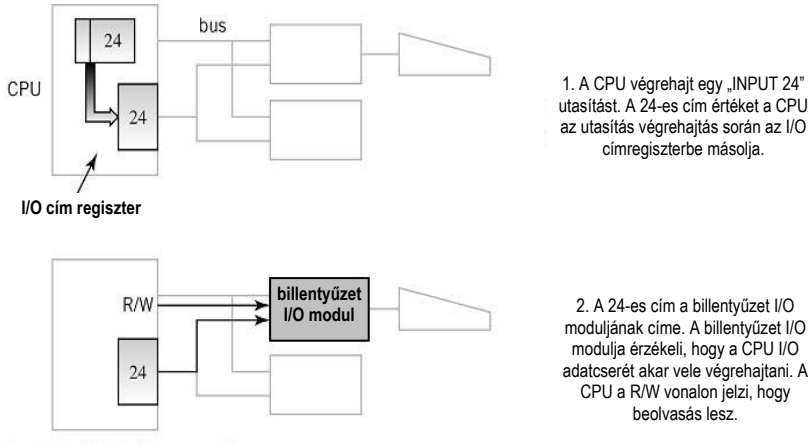
Az LMC modellben is volt egy-egy I/O olvasó és író utasítás (IN, OUT). Ha megnézzük az utasítás formátumát láthatjuk, hogy az LMC 100 különböző című I/O eszközt tud kezelni, melyek címei 0 és 99 közötti értékek.

A programozott I/O kezelést a modern számítógépekben ritkán használjuk, mivel az I/O kezelő utasítás végrehajtása olyan hosszú ideig tart, ameddig a periféria elvégzi az I/O műveletet. Ez egy lassú I/O eszköz esetben nagyon sok CPU utasítás végrehajtásának ideje is lehet. Ilyen eszközök programozott I/O kezelése esetén a processzor kihasználtsága igen lecsökkenne.

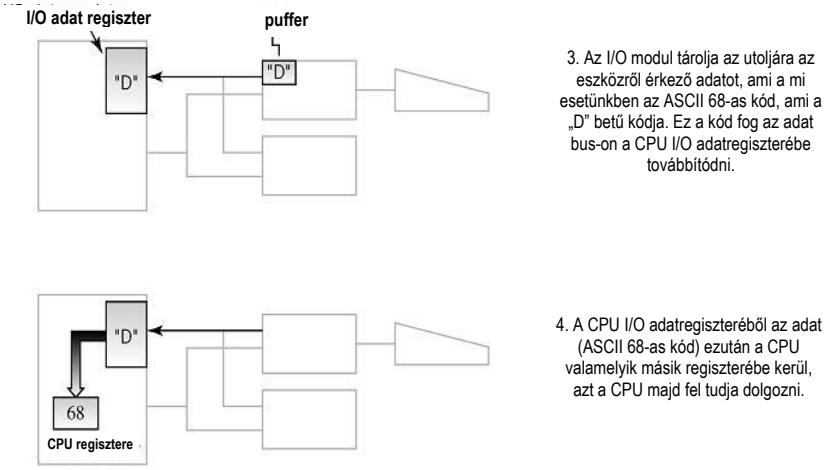
A perifériák közül programozott I/O kezelést gyakorlatban általában a billentyűzet kezelésére használják, ill. más típusú I/O eszköz kezelés részeként, mint pl. a DMA vagy a lekérdezéses I/O kezelés.



9.2. ábra I/O eszközök csatolása a CPU-hoz programozott I/O kezelés esetén



9.3. ábra I/O eszközök használata programozott I/O kezelés esetén – kérés

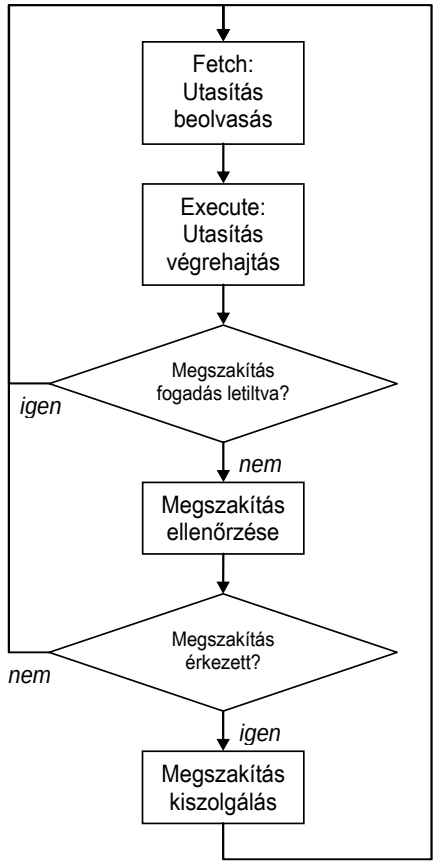


9.4. ábra I/O eszközök használata
programozott I/O kezelés esetén – válasz

9.3. I/O kezelés megszakítás (interrupt) segítségével

A megszakítás (interrupt) kezelés egy olyan mechanizmus a számítógépekben, melyek segítségével jelzéseket tudunk küldeni a CPU számára, mely jelzésekre az reagálhat.

Megszakításnak azt a jelzést nevezzük, mely hatására a CPU – bizonyos feltételek teljesülése esetén – megszakítja az aktuálisan végrehajtás alatt álló utasítássorozatot. A megszakítások kezelésére a processzort erre a célra szolgáló elemekkel egészítik ki. Az aktuálisan végrehajtott utasítássorozat megszakítása úgy valósul meg, hogy a processzor minden utasítás végrehajtása után megvizsgálja, hogy érkezett-e megszakításjelzés. Ha érkezett, nem a soron következő – utasításszámláló által mutatott – utasítást fogja végrehajtani, hanem egy előre definiált helyen elhelyezkedőt. A megszakítások figyelésével kiegészített processzor utasítás-végrehajtási ciklust a 9.5. ábra mutatja.



9.5. ábra A processzor utasítás-végrehajtási ciklusa a megszakítások figyelésével kiegészítve

I/O kezelés esetén a megszakítások használatának az az előnye, hogy nem kell a CPU-nak az egyes I/O eszközök állapotváltozására, I/O folyamat befejezésére várnia, ill. azokat figyelnie. Tipikus ilyen esemény egy korábban megkezdett adatátvitel befejeződése, vagy egy feldolgozandó adat beérkezése a perifériáról. A perifériák (egész pontosan az I/O modulok) ezeket az eseményeket a CPU várakoztatása nélkül tudják jelezni CPU felé megszakítások segítségével.

A megszakításokat a modern számítógépekben az I/O kezelésen kívül számos más feladat megoldására is használják, mert segítségükkel a számítógép reagálni tud a környezetből érkező eseményekre, megvalósítható az

ún. eseményvezérelt működés, mely a modern multiprogramozott számítógépekben különösen fontos.

A megszakításokat a következő feladatok megoldásánál használják:

- I/O eszközök kezelése.
- Váratlan input események kezelése, pl. Ctrl-C lenyomása.
- Abnormális (hiba) szituációkra reagálás a számítógépben, pl. feszültség kimaradás.
- Illegális (nem végrehajtható) utasítások kezelése, pl. nulla értékkel történő osztás.
- Multiprogramozott működésű rendszerekben a processzor idő elosztása a programok között (CPU ütemezés).
- Többprocesszoros gépekben a futó programok lecserélése egy másik processzoron (CPU ütemezés).

Mint láthatjuk nagyon sokféle esemény válthat ki megszakítást, ráadásul egy-egy esemény bekövetkezését több eszköz is okozhatja. A processzornak valamilyen módon azonosítania kell a megszakítást kiváltó eseményt, ill. a megszakítást kiváltó egységet. Erre két lehetséges módszert alkalmazhat:

- Polling – lekérdezéses megszakításkezelés: A CPU a megszakítás jel elfogadása után egy olyan speciális utasítást, vagy utasítás sorozatot hajt végre, melyet minden olyan egység érzékel, mely megszakítást okozhatta, és lekérdezi az I/O egységeket, hogy melyik okozta a megszakítást. A megszakítás okának, ill. a megszakítást okozó egységnek az azonosítása után a processzor arra a megszakítás kezelő programrészletre (melyek hasonlóak a szubrutinokra) ugrik, mely az adott eseményt lekezeli.
- Megszakítás vektor használata: Ebben az esetben a CPU képes különböző megszakításjelzéseket fogadni. Megszakítás küldésekor a megszakítást okozó egység adott típusú megszakításjelzéssel fogja jelezni a CPU felé az esemény bekövetkeztét. A megszakításjelzésen kívül általában elküldi a saját azonosítóját (címét) is. Ekkor a CPU közvetlenül arra a megszakítás kezelő programrészletre ugrik, mely az adott eseményt kezeli le. Ez a működés általában gyorsabb, hiszen nem kell külön lekérdezni az egyes I/O egységeket, azonban a processzor megszakítás kezelő része bonyolultabb.

A megszakításokat kezelő programrészleteknek, melyeket röviden megszakítás-kezelőnek (interrupt handler) nevezünk, tartalmaznia kell egy olyan utasítássorozatot, mely a kezelő futása elején elmenti a processzor állapotát, a futása végén pedig visszaállítja azt. Erre azért van szükség, mert a megszakítás kezelő befejeződése után az a program fog tovább futni, amelyet megszakított az interrupt, tehát a processzor regisztereiben azoknak az értékeknek kell szerepelni, amit a megszakítás előtt végrehajtott utasítás beállított. Ebből a szempontból a megszakítás-kezelők működése hasonlóak a szubrutinok működésére. A szubrutinokhoz hasonlóan a megszakítás-kezelők esetén is jellemzően a veremtárat használják a regiszterek tartalmának, vagyis a processzor állapotának elmentésére.

Egy működő számítógépben a megszakításjelzések igen sűrűn követik egymást. Gyakran előfordul, hogy egy-egy megszakítás kezelése során egy újabb megszakításjelzés érkezik a processzorhoz. Annak érdekében, hogy ekkor ez ne okozzon problémát a számítógép működésében, a megszakításjelzéseket egymáshoz képest rangsorolják, ún. prioritási sorba rendezik. A rangsorolás azt jelenti, hogy ha a processzor elfogadott (elkezdett kezelni) egy adott prioritású megszakítást, akkor a prioritási sorban lejjebb sorolt megszakításra a processzor nem fog reagálni mindaddig, amíg a nagyobb prioritású megszakítás kezelő rutint be nem fejezte. A megszakítások prioritásának kezelését a legtöbb processzor támogatja.

A prioritások kezelésén kívül a processzorok általában adnak lehetőséget, hogy egyes, vagy minden megszakításjelzés elfogadását tilthassa az éppen futó program. A megszakítások tiltását a megszakítások maszkolásának is szokták nevezni.

9.4. Közvetlen memória hozzáférés

Vannak olyan I/O egységek, melyek működése jellegzetes: egy I/O utasítás eredményeként egyszerre nagymennyiségű adatot képesek az őket vezérlő I/O modulon keresztül a processzor felé továbbítani. Ilyen lehet például a mágneslemezes háttértár, vagy a hálózatkezelő. A processzor ezen egységek programozott kezelésekor tipikusan az egységből érkező adatokat a processzor regisztereibe, majd a regiszterekből a memóriába mozgatná.

A közvetlen memória hozzáférés (Direct Memory Access - DMA) egy I/O kezelési mód, melyet a legtöbb modern számítógép támogat. A DMA lehetővé teszi a nagymennyiségű adatot továbbító I/O elemek gyors lekezelését. A DMA során az I/O egységből közvetlenül történik adatátvitel a

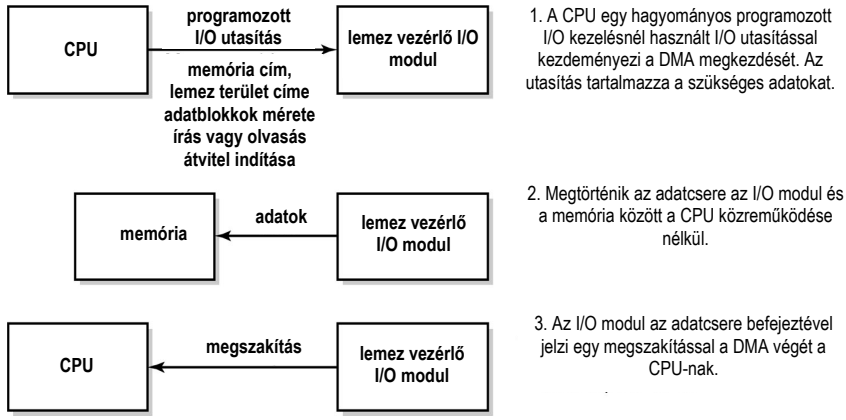
memóriába, ill. a memóriából. A CPU nem vesz részt aktívan az adatátvitelben.

A DMA megvalósításához a következő számítógépre vonatkozó feltételeknek kell teljesülnie:

- Közvetlen kapcsolat kell az I/O egység és a memória között. Ez a gyakorlatban azt jelenti, hogy azonos bus-on kell lennie az I/O egységeknek és a memóriának.
- Az I/O modulnak (vezérlőnek) képesnek kell lennie a memóriából olvasni és abba írni. Ez azt jelenti, hogy kell, hogy legyen MDR és MAR regisztereinek, ill. a memóriakezelést megvalósító egyéb logikai áramköreinek.
- A CPU és a I/O modulok közös memóriahasználatából adódó ütközéseket el kell kerülni, vagyis valamilyen kommunikációs protokollt kell meghatározni, mely kizárja, hogy két elem (I/O egység és a processzor) egyszerre használja a memóriát.

A DMA a következő lépesekben valósul meg (lásd 9.6. ábra):

- A program egy utasítása I/O kezelést eredményező szolgáltatást kér az operációs rendszertől.
- Az operációs rendszer a DMA elindításához egy programozott I/O kezelésnél használt processzorutasítást hajt végre, mely segítségével az I/O egységnek a következő – a DMA végrehajtásához szükséges – adatokat adja meg:
  - Az átviendő adatok elhelyezése (címe) az I/O egységen
  - Az átviendő adatok elhelyezkedése (címe) a memóriában
  - Az átviendő adat-blokk nagysága
  - A művelet típusa: olvasás/írás
- Az I/O eszköz az átvitel végén megszakítást küld a CPU-nak. Az előző programozott I/O utasítás után a megszakítás megérkezéséig a CPU nem használja a memóriát, nehogy ütközés legyen.

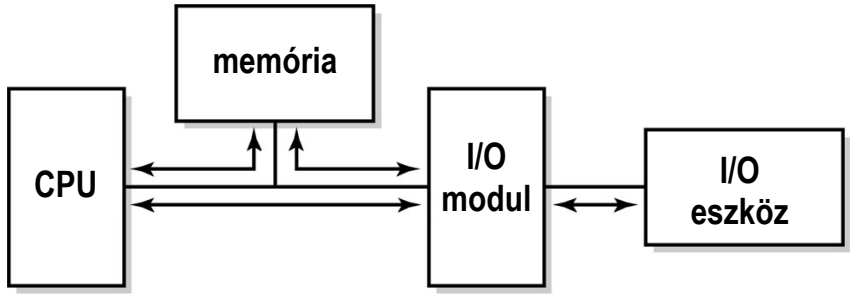


9.6. ábra A DMA (közvetlen memória hozzáférés) megvalósításának lépései

9.5. Különböző típusú I/O bus-ok

A modern számítógépekben – mint azt korábban már említettük – a számítógép különböző komponensei bus-okon keresztül kapcsolódnak egymáshoz.

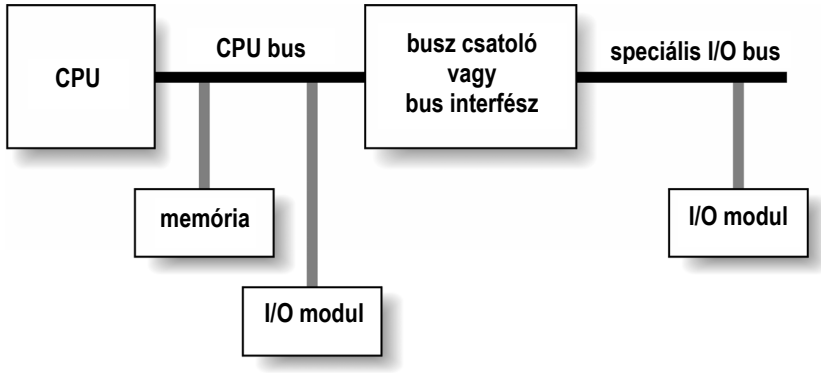
Az I/O elemek egy része – köszönhetően az I/O eszközöket közvetlenül kezelő I/O modul intelligenciájának – közvetlenül a CPU-t és a memóriát összekapcsoló rendszerbuszra köthető (9.7. ábra). Ekkor az I/O modul képes arra, hogy a rendszerbuszon szabványos protokoll szerint fogadja a processzor parancsait.



9.7. ábra A számítógépekben általánosan alkalmazott bus rendszerek felépítése

Az I/O elemek más része nem köthető közvetlenül a rendszer bus-ra. Ilyenkor a rendszer bus-ra valamilyen köztes eszközt csatlakoztatunk. Ezt

az eszközt bus interfésznek, vagy bus bridge-nek (hídnak) nevezzük (lásd 9.8. ábra). Ez a komponens lehetővé teszi a különböző protokollt használó bus-ok közötti adatátvitelt. A bridge-ek közvetítésével a CPU képes parancsokat adni, ill. adatokat küldeni és fogadni bármelyik bus-ra csatlakoztatott I/O eszközöknek, ill. eszköztől.



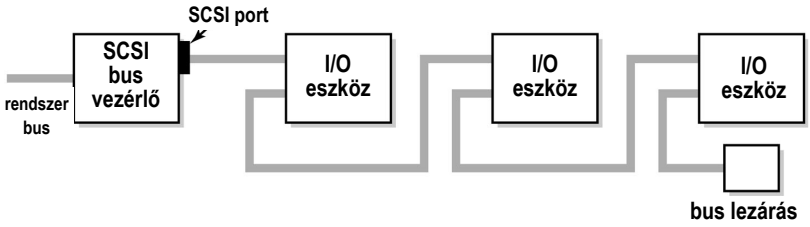
9.8. ábra Bus-ok összekapcsolása

Az egyes eszközöket rendszerhez kötő bus által használt protokoll alapvetően befolyásolja az eszközzel folytatott adatcsere sebességét. Számos szabvány létezik, melyek külső perifériák (I/O eszközök) számítógéphez csatolását teszi lehetővé.

Az egyik széles körben használt szabvány a SCSI (Small Computer System Interface). A szabványt 1979-ben kezdték kidolgozni, a 90-es évek közepére igen elterjedtté vált. Elsősorban a nagyobb számítógépek esetén használták a PC-kben elterjedt ún. ATA (IDE, ATAPI, ill. UDMA) interfészhez viszonyított viszonylagosan magas ára miatt. A SCSI bus-on keresztül elsősorban merevlemezeket, szalagos tárolókat szoktak a számítógéphez csatolni, de lehetséges minden más eszköz (pl. szkenner, optikai tárolók, nyomtatók) csatolása is. Legfontosabb előnye a többi külső buszrendszerhez képest az, hogy több eszköz csatolását teszi lehetővé egyetlen csatoló elemhez, és viszonylag nagy átviteli sebességet biztosít. A SCSI bus felépítését a 9.9. ábra mutatja.

A SCSI szabványnak számos változata létezik, párhuzamos (8 vagy 16 bit szélességű) adatátvitelt tesz lehetővé, szabványtól függően 7 vagy 15 eszköz csatolható egy bus vezérlőhöz (bridge-hez). Az adatátvitel sebessége 5 MB/s-tól 640 MB/s-ig változik, a bus maximális fizikai hossza pedig 1.5 és 12 méter között lehet ugyancsak a szabvány adott változatától függően.

A közelmúltban a SCSI szabványnak is megjelentek a viszonylag nagysebességű soros adatátvitelt megengedő szabványváltozatai (Serial Storage Architecture[SSA], Serial Attached SCSI [SAS]).

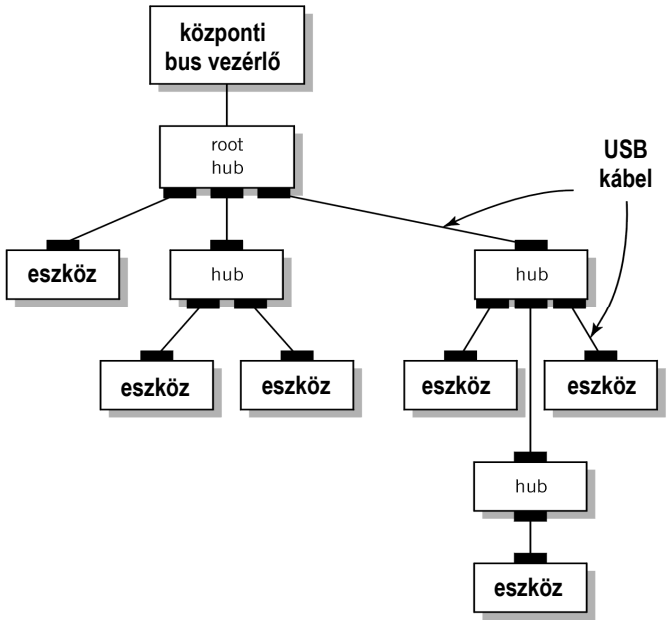


9.9. ábra Az SCSI bus felépítése

A PC kategóriájú, viszonylag olcsónak számító számítógépek körében egyre elterjedtebben használják a nagysebességű soros adatátvitelt lehetővé tevő soros bus-okat. A soros bus-okat eredetileg kis sávszélességű adatátvitelkor használták, azonban az Universal Serial Bus (USB) ill. a FireWire (IEEE 1394) nagy sávszélességű átvitelt biztosító változatainak megjelenésével ez megváltozott.

Az USB szabványt eredetileg a PC-kben korábban alkalmazott soros és párhuzamos porokat használó eszközök szabványos protokollon keresztül történő csatolását lehetővé tevő szabványnak szánták.

Az USB bus felépítése speciális abból a szempontból, hogy lehetővé teszi a fa topológiájú bus felépítését ún. hub (hálózati középpont) egységeken keresztül (lásd 9.10. ábra).



9.10. ábra Az USB bus felépítése

Az USB protokollt használó eszközök köre ma már igen széles köszönhetően az USB 2.0 szabvány viszonylag nagy (480 Mbit/sec) adatátviteli sebességének, a külső merevlemezes tárolók, videó felvevők, nyomtatók, FlashDrive-ok, külső optikai tárolók stb. szinte kivétel nélkül rendelkeznek USB csatoló lehetőséggel.

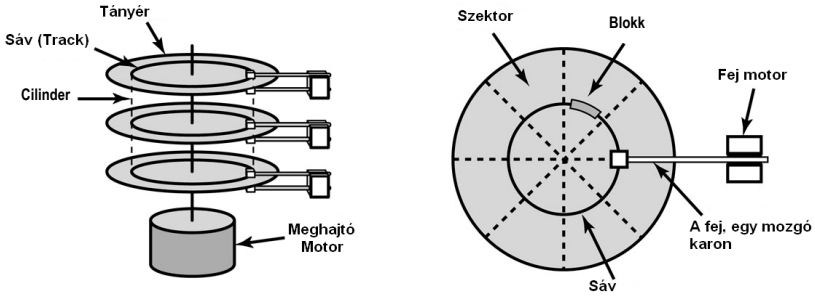
10. Számítógép perifériák működése

10.1. Mágneslemezes háttértár működése

A mágneslemezes tárolók az információ tárolására mágnesezhető anyagot használnak. A mágnesezhető anyagot különböző képen mágnesezik, úgy, hogy egy-egy bitnyi információ tárolásához a mágneslemez felületének egy adott kicsi darabját használják. A mágneslemezes tárolóknak igen sok típusa létezik, van ahol egy, van ahol több lemezt használnak, valamikor a lemezek mindkét oldalát, valamikor csak az egyiket használják adattárolásra. A lemezek lehetnek merevek vagy rugalmasak.

A mágneslemezes tárolók esetén az olvasás a mágnesezett felület egy tekercs közelében történő elmozgatásával történik. A tekercset tartalmazó részt olvasó/író fejnek nevezzük. A fej közelében nagy sebességgel elhaladó mágneses felület az olvasó fejben elektromos áramot indukál, mely indukált áram, ill. feszültség nagysága függ a felület mágnesezettségétől.

A mágneslemezes háttértárak részeit a 10.1. ábra mutatja.



10.1. ábra A mágneslemezes háttértár

A mágneslemezes tárolók esetén általában nem biteket vagy bájtokat, hanem jellemzően bájtok csoportjait ún. adat blokkokat tárolunk, írunk és olvasunk. Fontos kérdés, hogy az egymás után tárolt adat blokkokat milyen sorrendben, vagyis hol tároljuk a lemezen, ill. lemezeken.

A tároló egyszerű kezelése érdekében a lemezek adattároló felületét különböző egységekre osztjuk. A sávnak nevezzük az egy lemezfelületen egy adott fejállással olvasható mágnesezett felület. Az egymás alatt egy tengelyen elhelyezett lemezeket levő sávokat ún. cilindernek hívjuk. A

sávokat pedig szektorokra bontjuk. Egy szektor azonos mennyiségű adatot – általában egy adatblokknyi információt – tárol.

A mágneslemezes tárolók jellemzője, hogy használatuk során azonos szögsebességgel forgatjuk őket. Egy-egy szektor különböző sávba eső adatblokkjaiban azonos mennyiségű adatot tárolunk függetlenül attól, hogy az adott szektort tartalmazó sáv a lemez külső részén, vagy a közepéhez közel helyezkedik-e el, hiszen a szektor minden sávjához azonos lemezelfordulás, így olvasási idő tartozik.

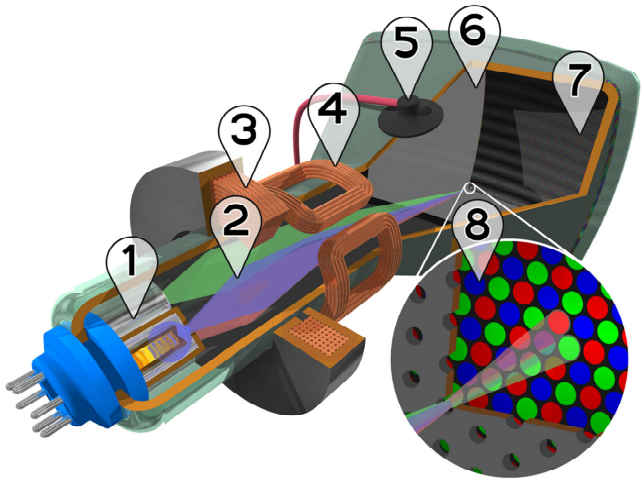
10.2. Katódsugárcsöves monitor

A katódsugárcsöves (CRT – Cathode Ray Tube) monitorok a legrégebben használt számítógépes megjelenítők. A katódsugárcsöves monitorok eleinte egy színű („fekete-fehér", a valóságban inkább „fekete-zöld") képek megjelenítésére voltak alkalmasak, ma már természetesen nagyfelbontású színes képeket tudnak megjeleníteni. A más elven működő, elsősorban a folyadékkristályos megjelenítő eszközök (monitorok) egyre inkább kiszorítják őket a számítógépes megjelenítők piacáról.

Működésük hasonló a hagyományos televízió készülékek működéséhez. A kép megjelenítés alapját jelentő fizikai jelenség igen egyszerű. Ha egy elektronsugárral megvilágítunk valamilyen fluoreszkáló (fluoreszcens) anyagot, akkor az fényt bocsát ki, felvillan.

A monitor egy katódsugárcsőből és a katódsugárcsövet vezérlő elektronikából áll. A katódsugárcsőben van egy elektronforrás, amelyből elektronnyalábot állítunk elő. Az elektronnyalábot elektromágneses tér segítségével pontosan tudjuk irányítani, ill. igen gyorsan mozgatni, valamint az erősségét szabályozni (lásd 10.2. ábra).

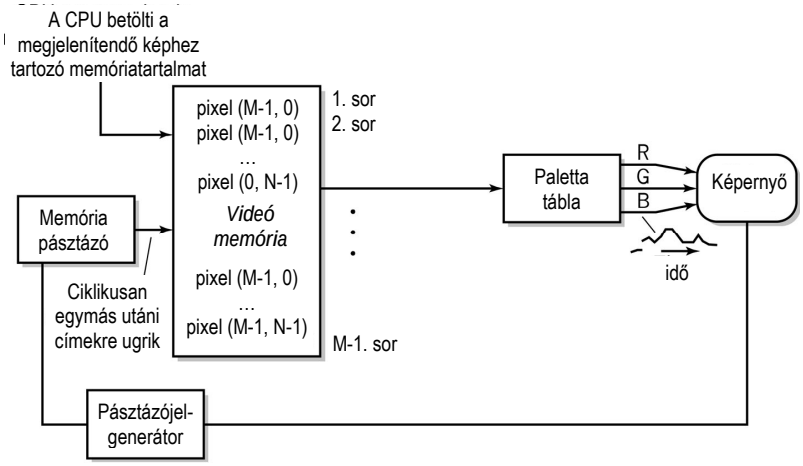
A kép a katódsugárcső elülső felületére vékonyan felvitt fluoreszcens anyag segítségével áll elő. Az elektronnyaláb folyamatosan pásztázza a fluoreszcens anyaggal bevont felületet, újabb és újabb képpont felvillanását okozva. Színes megjelenítők esetén nem egy, hanem három elektronnyaláb van, melyek közvetlenül egymás mellett elhelyezkedő három különböző (vörös, zöld és kék) képpontot villantanak fel. A képpontok fényessége, ill. a színes megjelenítő esetén a színe a becsapódó elektronsugár erősségétől függ.



10.2. ábra A színes katódsugárcső felépítése
1. elektronágyú; 2. elektronnyalábok (színenként egy); 3. fókuszáló tekercsek; 4. eltérítő tekercsek; 5. anódcsatlakozó; 6. maszk a megjelenítendő kép vörös, zöld és kék részének szétválasztásához; 7. foszforréteg vörös, zöld és kék zónákkal; 8. képernyő foszforborítású belső rétegének közelképe

A monitor képét az egyes képpontokban levő fluoreszcens anyag felvillantásával „rajzolja” ki az elektronsugár. Mivel az elektronsugár igen gyorsan képes végigpásztázni a teljes képernyőt (ami a katódsugárcső elülső felülete), így mi az egymás után gyorsan kirajzolt képpontokat a szemünk egy képnek fogjuk érzékelni, mert egy képpont két felvillantása között igen kicsi idő telik el.

Ha megnézzük, hogy a számítógép hogyan tárolja a képi tartalmakat, akkor láthatjuk, hogy az eltárolt adatok alapján viszonylag könnyen meg lehet határozni, hogy melyik pillanatban milyen erősségű elektronnyalábot kell egy-egy képpont megjelenítéséhez előállítani. Ennek logikáját mutatja a 10.3. ábra.



10.3. ábra A kép megjelenítés logikája katódsugárcsöves monitor esetén

Láthattuk, hogy katódsugárcsöves monitorok esetén a képpontok által kirajzolt képet a képpontokat megjelenítő fluoreszcens anyag felvillantásával állítjuk elő, vagyis a képpontok nem világítanak folyamatosan. A katódsugárcsöves monitorokat ezért passzív megjelenítőnek nevezzük, mert egy-egy képpont nem világít „aktívan" állandóan, az csak a szemünk látja így. A kép vibrálását ugyan a szemünk közvetlenül nem érzékeli, de – elsősorban az alacsonyabb, kb. 60 Hz alatti képfrissítési frekvenciájú monitorok esetén – a vibrálás fárasztja a szemet. Ezért fontos szempont egy képernyő kiválasztásakor a monitorok képfrissítési frekvenciájának figyelembevétele.

10.3. LCD monitor

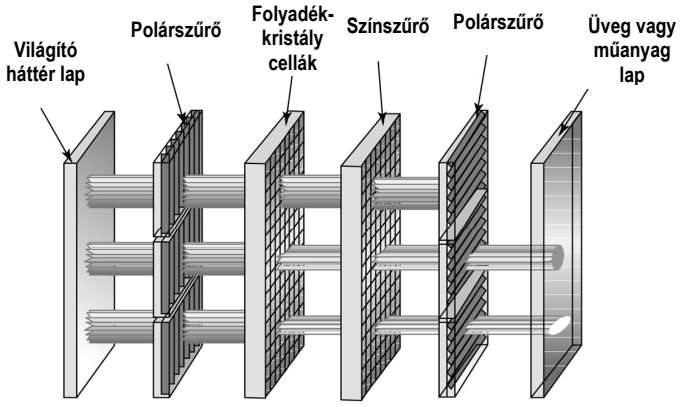
A folyadékkristályos (LCD) megjelenítő eszközök (monitorok) egészen más fizikai jelenséget használnak, mint a katódsugárcsöves monitorok. Míg a katódsugárcsöves monitorokat passzív megjelenítőnek, addig az LCD monitorokat aktív megjelenítőknek nevezzük, mert minden képpont folyamatosan „aktívan" világít a képernyőn egy kép megjelenítésekor.

Az LCD monitorok azt a fizikai jelenséget használják a kép megjelenítésére, hogy a folyadékkristály – ami egy részben fényáteresztő anyag – térben elforgatja a rajta áthaladó fény polaritását. Az elforgatás mértéke elektromos feszültség hatására megváltozik. Ha tehát egy folyadékkristályra feszültséget kapcsolunk, és azon a tér egy adott irányába polarizált fényt

bocsátunk át, akkor a feszültség változtatásával a folyadékkristályból távozó fény más és más irányba lesz polarizálva.

A megjelenítés logikájának megértéséhez még azt kell tudnunk, hogy egy adott irányban polarizált fényből a polárszűrő által elnyelt fénymennyiség arányos a polarizált fény és a polárszűrő polarizálása által bezárt szöggel. Ha a polárszűrő és a polarizált fény polaritása azonos, akkor minden fény átmegy a szűrőn, ha azok egymásra merőlegesek, a szűrő minden fényt elnyel.

Az LCD monitorok fizikai felépítését a 10.4. ábra mutatja.



10.4. ábra A folyadékkristályos (LCD) monitorok fizikai felépítése

A monitor fényét egy világító lap állítja elő. A lapból kijövő fény egy polárszűrőn halad át, így utána már egy irányba polarizált fényt kapunk. Utána ez a fény áthalad a folyadékkristályt tartalmazó panelen, melynek minden egyes képponthoz tartozó részét külön tudjuk vezérelni, vagyis az adott képponthoz tartozó folyadékkristály „darabon” (cellán) mérhető feszültséget a többi résztől függetlenül tudjuk beállítani. (A korábban készített ún. passzív mátrixos kijelzőknél ez még nem feltétlenül volt így, ott egy kis kondenzátor „tartotta” a folyadékkristály cella feszültségét, amit bizonyos időnként lehetett „újratölteni”. Ma már többnyire nem ezt a megoldást, hanem az ún. aktív mátrixos kijelzőket használjuk, ahol a cellák feszültségét folyamatosan lehet állítani.)

Színes LCD monitor esetén ezután következik egy a fehér fényből színes fényt előállító színes szűrő, mely a katódsugárcsöves monitorokhoz hasonló módon állítja elő a közvetlenül egymás mellett levő három különböző (vörös, zöld és kék) képpontból a színesen megjelenő képpontokat.

A megjelenítendő kép igazándiból ezután, a következő polárszűrőn rajzolódik ki, mely pontosan merőlegesen polarizált az előző polárszűrőhöz képest. Ez azt jelenti, hogy ha a folyadékkristály semennyit nem fordít egy adott ponton áthaladó fénynyalábon, akkor ez a szűrő az adott ponton minden átjövő fényt elnyel. Ha a képpontot vezérlő folyadékkristály elfordította a fénynyalábot, akkor az elfordítás mértékével lesz arányos a pont fényessége, színes monitor esetén pedig a színe és a fényessége.

A monitor utolsó rétege egy rugalmas (műanyag), vagy merev (üveg) fényáteresztő borítás, mely védi az alatta levő rétegeket. A felhasználó ezen a fényáteresztő borításon keresztül látja a monitor képét.

Irodalom

Irv Englander: *The Architecture of Computer Hardware and Systems Software: An Information Technology Approach* 3rd Edition, John Wiley and Sons, 2003, ISBN 0-471-07325-3, 729 o.

Andrew S. Tanenbaum: *Számítógép-architektúrák,* 2. átdolgozott kiadás, Panem 2006, ISBN 9635454570, 816 o.

Andrew S. Tannenbaum: *Structured Computer Organisation,* 5. kiadás, Prentice Hall, 1999., ISBN 0-13-148521-0, 800 o.

Sima Dezső – Terence Fountain – Kacsuk Péter: *Korszerű számítógép-architektúrák tervezésitér-megközelítésben,* Szak Kiadó, Bicske, 1998.

Kai Hwang: *Advanced computer architecture; parallelism, scalability, programmibility,* McGraw-Hill International, 1993

Németh G. – Horváth L.: *Számítógép architektúrák,* 2. kiadás, Akadémiai Kiadó, 1993.

Iványi Antal (szerk.): *Informatikai algoritmusok I.,* ELTE Informatikai Kar, Budapest, 2005,
http://compalg.inf.elte.hu/~tony/Elektronikus/Informatikai/Infalg1H.xml

John L. Hennessy – David A. Patterson: *Computer architecture, a quantitative approach,* Morgan Kaufmann Publishers, 1990