Simon Gyula

SZÁMÍTÁSTECHNIKA

középiskolásoknak

Simon Gyula

SZÁMÍTÁSTECHNIKA

középiskolásoknak

Harmadik, átdolgozott kiadás

MKM tankönyvi engedély száma: 61.675/95.XII.

Szerző:

Simon Gyula
szakvezető tanár

Lektor:

Dr. Bölcskei András
egyetemi adjunktus
KLTE Matematikai és Informatikai tanszék

Csontó Béla
középiskolai tanár

Papp Ferenc
nyelvi lektor

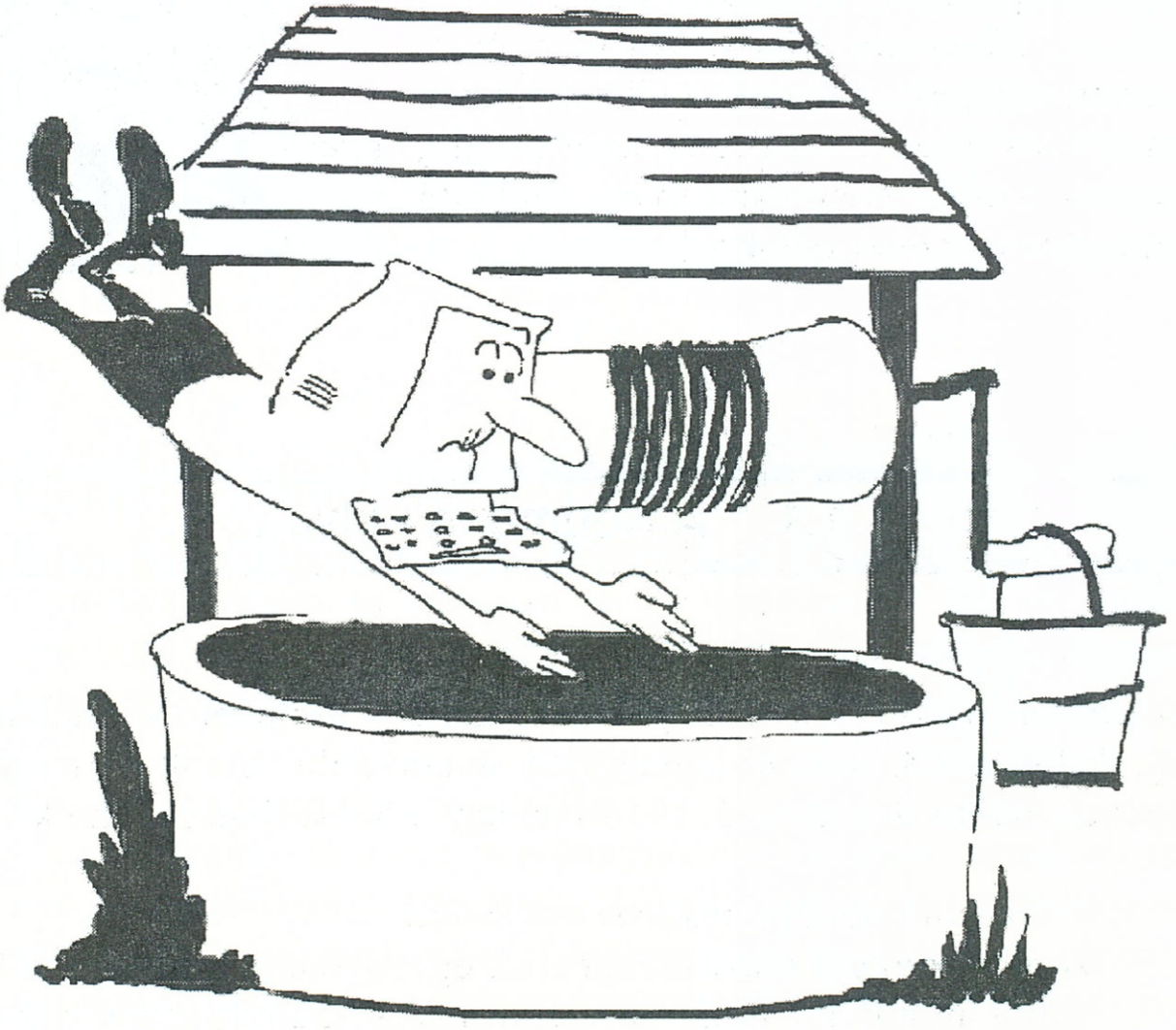
ISBN 963 9216 18 6

Kiadó: PEDELLUS NOVITAS Kiadó
4001 Debrecen, Pf. 430

Terjedelem: 9,6 (A/5) ív
Illusztráció: Hadházi László
Alkotó szerkesztő: Nagy Jánosné
Technikai szerkesztő: Kiss Tamás és Molnárné Balázs Zsuzsanna
1999.
Harmadik, átdolgozott kiadás

Nyomda: **Kapitális Bt.**
Felelős vezető: **Kapusi József**

TARTALOM



1. SZÁMÍTÁSTECHNIKAI ALAPFOGALMAK

Az eddigi tanulmányaitokban már találkoztatok sok fontos fogalommal a számítógéppel kapcsolatban. A továbbiakban felelevenítjük ezeket, illetve néhány új dologgal is megismerkedünk.

A mai modern számítógépek elvét egy magyar származású amerikai matematikus fektette le: *Neumann János*. Ma már léteznek számítógépek, amelyek működésükben eltérnek ezektől az elvektől, de még mindig a hagyományos, Neumann-elven működő számítógépek az elterjedtebbek.

1.1 A Neumann - elvek

Neumann János (1903-1957) több tudomány területén is maradandót alkotott. Talán a XX. század legnagyobb matematikusa volt. A modern számítógépek alábbi öt alapelvét 1946-ban egy előadásában fejtette ki.

1. A *számítógép legyen soros működésű.*
A gép az egyes utasításokat csak egyenként hajtja végre. Neumann abból indult ki, hogy az elektronikus számítógépek párhuzamos működésmód nélkül is elég gyorsak lesznek. A mai számítógépek többsége soros működésű, bár léteznek többprocesszoros számítógépek, amelyek képesek egy időben több részfeladatot is végrehajtani.

2. *A számítógép a kettes számrendszert használja, és legyen teljesen elektronikus.*

A kettes számrendszert nagyon könnyű megvalósítani elektronikus, úgynevezett kétállapotú áramkörökkel. (Például: 1- magasabb feszültség, 0- alacsonyabb feszültség) Kezdetben a számítógépek elektroncsöveket tartalmaztak, ezek élettartama és megbízhatósága gyenge volt. Később tranzisztorokat alkalmaztak, amelyek mérete lényegesen kisebb, megbízhatósága sokkal jobb volt. A 60-as évektől nagybonyolultságú integrált áramköröket hoztak létre, amelyek kis méretben tartalmaznak nagyszámú félvezető alkatrészt, amikkel sok feladatot láthat el az áramkör.

3. *A számítógép belső memóriát tartalmaz.*

A számítógép gyors működése következtében nincs lehetőség arra, hogy minden egyes lépés után ember avatkozzon be a számítás menetébe. Neumann egy belső memória kialakítását javasolta, amelyben a részeredmények tárolhatók, és így a gép egy bizonyos műveletsort automatikusan el tud végezni. A mai számítógépekben félvezető memóriát alkalmaznak, amelyek integrált áramköröket tartalmaznak.

4. *A tárolt program elve.*

A programot megvalósító utasítások kifejezhetők számmal, azaz adatként kezelhetők. Ezek a belső memóriában tárolhatók, ugyanúgy, mint bármilyen más adat. Így a számítógép önállóan képes működni, hiszen az adatokat és az utasításokat is a memóriából veszi elő. Természetesen ezek a memória más-más részében találhatók egy adott program futása közben.

5. *A számítógép legyen univerzális.*

A számítógép különféle feladatainak elvégzéséhez nem kell speciális berendezéseket készíteni. *Turing,* angol matematikus matematikai logikai eszközökkel bizonyította be, hogy az olyan gép, amely el tud végezni néhány alapvető műveletet, elvileg bármilyen számítás elvégzésére is alkalmas.

1.2 Az információ fogalma

Az információ az egyik legellentmondásosabb fogalom: olyan sok értelemben használják, hogy jelentése nem fedhető le egy meghatározással.

Milyen kijelentéseket, üzeneteket tekinthetünk információnak? Azokat, amelyek új ismereteket hordoznak. Annak a közlésnek tehát, amely már ismert eseményről számol be, nincs információs értéke. Minél váratlanabb egy esemény, vagy minél kisebb bekövetkezésének a valószínűsége, annál nagyobb a róla szóló hírben foglalt információmennyiség.

Az információ forrását ADÓ-nak nevezzük. Üzenetét valamilyen jelkészlet jeleiből állítja össze, bizonyos szabályok felhasználásával. A sorozatba rendezett jeleket valamilyen információhordozóra bízza, és közvetítő csatornán küldi az üzenet címzettjének, amit VEVŐ-nek nevezhetünk. A csatorna lehet természetes (hang, fény stb.) és mesterséges (telefon, rádió stb.) is. A természetes csatornákon a jel az eredeti alakjában utazhat, a mesterséges csatornákban való szállításhoz át kell alakítani. Ez a *kódolás.* Pl.: Amikor telefonon beszélünk, kódolást hajtunk végre: a beszédhangot elektromos impulzusokká alakítja a készülék. A VEVŐ-nek vissza kell alakítani a jeleket, ez a *dekódolás.*

Az információs csatornát jellemzi a jel/zaj viszony. Zaj pl.: az utcai lárma, a sajtóhiba, légköri zavarok stb. Ha zajos utcán akarunk beszélgetni, akkor nagyobb hangerővel kell beszélni, hogy a jel/zaj viszony jobb legyen, hogy megértsük egymást.

A kódolt információ sokszor tartalmaz felesleges részleteket is. Pl.: *„Elmész ma este a moziba?”* - szól a kérdés. Erre többféleképpen is válaszolhatunk: *„Igen, ma este elmegyek moziba.”*, vagy *„Igen elmegyek.”*, vagy *„Igen.”*. Mindegyiknek ugyanaz az információtartalma, de különböző a jelek száma. A lényeget az utolsó is tartalmazza, de nincs benne felesleges jel. Az első két választ *redundánsnak* nevezzük. Felületesen nézve a redundanciát, úgy tűnhet, hogy az káros jelenség. Ez sokszor így is van (Sok beszédnek sok az alja - tartja a közmondás is), de bizonyos határon

túl nem ajánlatos a redundanciát csökkenteni: ugyanis a redundancia nélküli, maximális információt tartalmazó üzenet könnyen megsérülhet az információs csatornában (zaj). A redundáns részek segítségével küldhetünk olyan információkat, amelyek segítségével kijavíthatjuk a megsérült üzenetet.

Az információ elemi egysége a **bit**. Ha egy állításról megtudjuk, hogy igaz vagy hamis, az 1 bit információnak felel meg.

Megjegyzés:

1. *Itt tehát két, egymást kizáró, és azonos valószínűségű lehetőségről van szó. Miközben választunk közülük, 1 bit információt közlünk.*
2. *Bit = **bi**nary unit, azaz bináris egység.*

1.2.1 Adat

Az információt kódolni kell, majd ezt (ami egy jelsorozat lesz), lehet tovább küldeni a Vevő felé, vagy tárolni. A jelsorozat formai oldalát (Pl.: egy karaktersorozat) nevezzük adatnak. A tartalmi oldal maga az információ. (A tárolás valamilyen tárolóeszközre vonatkozik: memória, lemezegység, stb.) A feldolgozás közvetlenül az adatokkal történik. Ehhez nem feltétlenül szükséges tudnunk az információk jelentését, elegendő az információkat tartalmazó adatokat ismerni. Természetesen az eredmények értékeléséhez szükségesek az információk eredeti jelentései is. A tárolás során a számítógép a kettes számrendszert használja. Az adat mennyiségének jellemzésére a felhasznált kettes számrendszerbeli számjegyek (0,1) számát használhatjuk. Egy kettes számrendszerbeli számjegy **1 bit**nek felel meg.

Megjegyzés:

1. *Bit= **bi**nary digit, azaz bináris számjegy.*
2. *A kétféle bit nem feltétlenül felel meg egymásnak: 1 bitnyi információt nem biztos, hogy 1 bitnyi adatmennyiséggel adhatjuk meg.*

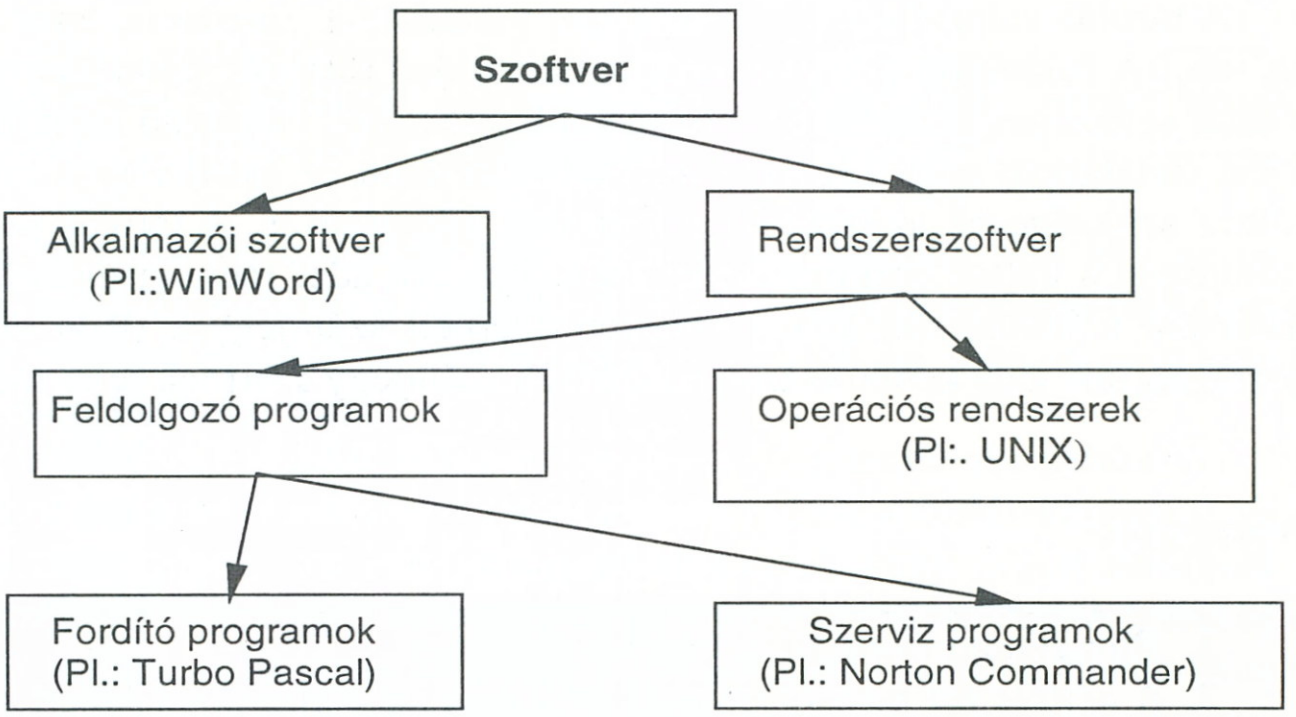
A következő táblázat a bit néhány többszörösét foglalja össze:

1 bájt	8 bit
1 kilobájt (Kb)	1024 bájt ($=2^{10}$ bájt)
1 megabájt (Mb)	1024 kilobájt ($=2^{20}$ bájt)
1 gigabájt (Gb)	1024 megabájt ($=2^{30}$ bájt)

1.3 Hardver, szoftver

A *hardver* a számítógép technikai, műszaki felépítését jelenti: az elektronikus áramköreit, mechanikus egységeit, kábeleit, csatlakozóit, perifériáit. A *szoftver* különböző programok, programrendszerek összefoglaló neve. A szoftverek teszik működőképessé a számítógépet.

A szoftvereket feloszthatjuk a következőképpen:

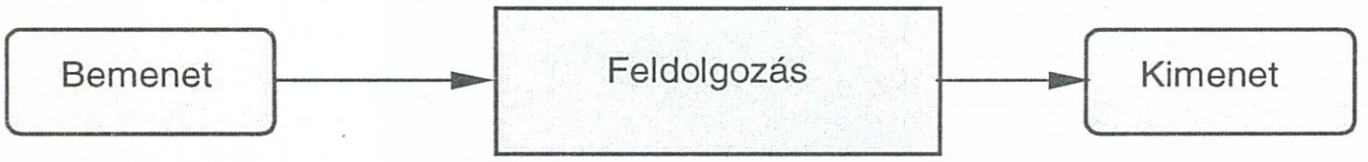


A számítógép *hardver erőforrásán* a számítógép munkájában résztvevő önálló hardver eszközöket értjük (Pl: CPU, memória, háttértárak stb). A *szoftver erőforrásokon* a számítógép működését segítő különböző szoftvereket értjük.

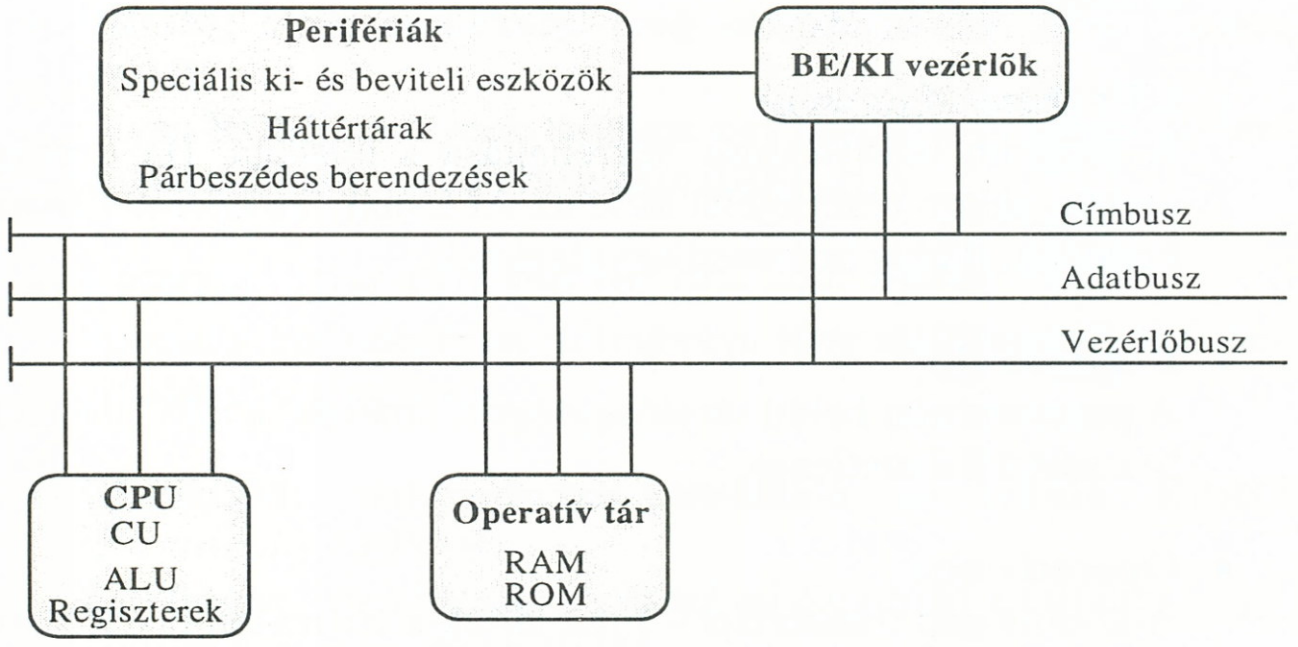
Minden programhoz egy változó, de minden időpillanatban jól meghatározott erőforrásigény tartozik, ezért alapvető feladat az erőforrásokkal való gazdálkodás.

1.4 A számítógép felépítése

A számítógép működése nagyon leegyszerűsítve a következő: beviszünk adatokat a gépbe, ezeket az feldolgozza, majd az eredményt a kimeneten közli a környezetével. (Maga a processzor szó is *feldolgozót* jelent.)



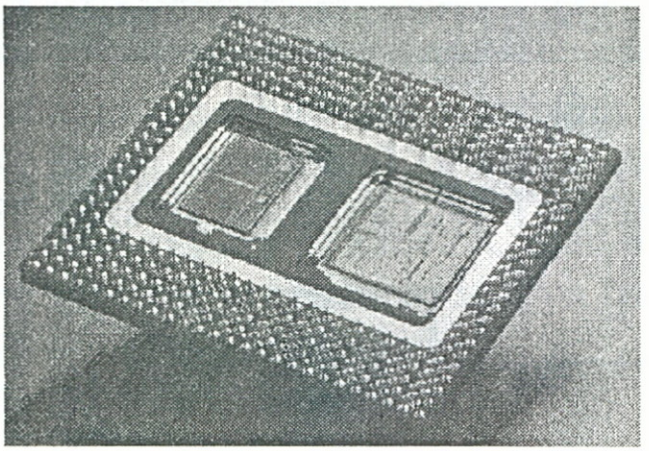
Természetesen a számítógépek elég változatos képet mutatnak felépítésük és működésük alapján. Mi az egyik legelterjedtebb változattal ismerkedünk meg, az ún. sínrendszerű kiépítéssel. Ilyenek az általunk használt PC-k, ilyen felépítésű az egyik legismertebb mini számítógép, a VAX is.



A számítógép főbb részei az ábrán láthatók. Az egyes részek funkciói:

- **CPU:** (*Központi Feldolgozó Egység*)
  A **CU**-t és az **ALU**-t és a regisztereket együttesen nevezzük **CPU**-nak. Ez a **processzor**, amelyet szoktak a számítógép „agyának" is nevezni.

- **Mikroprocesszor:**
  Olyan integrált áramkör, amely egyetlen szilíciumlapkán tartalmaz egy CPU-t. Az első mikroprocesszort 1971-ben készítette az Intel cég.



- **ALU:** *(Aritmetikai-Logikai Egység)*
  Általában minden programutasítás végrehajtásában részt vesz. Ez aritmetikai, logikai műveletek végrehajtását, illetve a relációk kiértékelését jelenti.

- **CU:** *(Vezérlő Egység)*
  A program utasításainak végrehajtása a feladata. Ha a program előírja, akkor vezérlőjelet küld az ALU-nak, az operatív tárnak, a kimeneti, illetve bemeneti egységeknek.

- **Regiszterek:**
  A processzoron belüli tárolóegységek, amelyek a CPU működéséhez nélkülözhetetlenek.

- **Operatív tár:**
  A számítógép memóriája – mint láttuk a Neumann-elvek között – tartalmazza magát a programot, és a program által használt adato-

kat is. Ahhoz, hogy a CPU el tudja helyezni az információt a memóriába, valahogyan el kell különíteni egymástól az egyes adatokat. A mai számítógépek memóriája *bájtszervezésű*, ami azt jelenti, hogy a legkisebb adatmennyiség, amit kezelni tudunk: 1 bájt. A központi tárat úgy képzelhetjük el, mint sok, azonos méretű (1 bájtos) *rekesz* összességét. Minden rekesznek van egy sorszáma, vagy másképpen *címe,* amivel azonosítani tudjuk. Amikor azt mondjuk, hogy a processzor *megcímez* egy memóriarekeszt, az tulajdonképpen azt jelenti, hogy megkeresi azt a rekeszt, amellyel dolgozni akar, az ahhoz rendelt szám segítségével.

- **RAM: véletlen hozzáférésű memória**. (*Random Acces Memory)*
  Minden tárolóhely címének megadásával akárhányszor olvasható, törölhető, vagy írható. A *véletlen hozzáférés* azt jelenti, hogy bármelyik tárolócella eléréséhez közel ugyanannyi idő szükséges. A gép kikapcsolásakor vagy áramkimaradás esetén az adatok elvesznek.

- **ROM: csak olvasható memória, tartalma nem változtatható meg**. *(Read Only Memory)*
  Tartalmát a gyártó a vevő kívánsága szerint az előállítás során rögzíti. A ROM is véletlen hozzáférésű. A gép kikapcsolásakor vagy áramkimaradás esetén az információk nem vesznek el. A ROM-ok általában lassabbak a RAM-oknál.

- **PROM: Programozható, csak olvasható tár.** (*Programmable ROM*)
  Olyan ROM, amelynek tartalma egy speciális programozó berendezéssel írható, ezután megváltoztathatatlan.

- **EPROM: újraírható PROM**. *(Eraseble PROM)*
  Speciális berendezéssel (ultraibolya fény segítségével) törölhető a tartalma, és újraírható.

- **EEPROM: elektromosan törölhető és írható EPROM.** *(Electrically EPROM)*
  Elektromos impulzusok segítségével törölhető a tartalma. Egy ismert típusa az ún. **flash ROM**.

- **Buszrendszer:**
  Az adat-, cím-, ill. vezérlőbusz az adat-, cím-, ill. vezérlőjeleket közvetítő vezetékek összességét jelenti. Az adatbuszon haladnak az egyes részek között az adatok. A címbuszon a memória rekeszek kiválasztásához szükséges adatok, a vezérlőbuszon a CPU által küldött vezérlőjelek, haladnak.

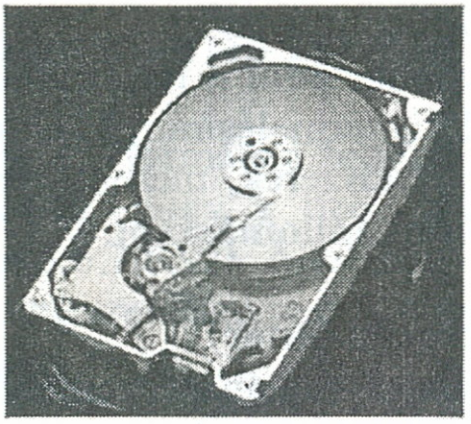
- **Be/Kiviteli vezérlők:**
  A perifériák és a vezérlőegység közötti adatforgalom vezérlése a feladatuk. Megcímezik (kiválasztják) a megfelelő perifériát, majd kiépítik a megfelelő kapcsolatot vele. Ezután pedig vezérlik az átvitelt. A vezérlők intelligens egységek, képesek önállóan dolgozni. Így amíg a viszonylag lassú periféria dolgozik, addig a processzor mással is foglalkozhat.

- **Párbeszédes berendezések:**
  Ezeket más néven terminálóknak is nevezik. Az adatbevitel és az adatkivitel váltakozva követik egymást. A terminálok általában képernyősek, azaz egy billentyűzetből és egy monitorból állnak, de régebben voltak nyomtatós terminálok is.

- **Beviteli berendezések:**
  Például:
  - billentyűzet
  - egér
  - scanner vagy lapolvasó
  - vonalkód olvasó
  - hangdigitalizáló stb.

- **Kiviteli berendezések:**
  Például:
  - nyomtató
  - monitor
  - plotter
  - hangerősítő, hangszóró, stb

- **Háttértárolók:**
  A gyors működésű operatív tár csak viszonylag kevés adat átmeneti tárolására alkalmas. Nagy tömegű adatok tárolására a háttértárolók alkalmasak.
  Például:
  - hajlékonylemezes egység (FDD)
  - winchester (merevlemezes egység, FDD)
  - streamer (szalagos egység)
  - optikai tárolók (Pl.: CD-ROM, DVD), stb.



1.5 A számítógépek jellemzése

A számítógépeket különböző szempontok szerint csoportosíthatjuk:

- **Műveleti sebesség**
  Az az utasításszám, amelyet átlagosan egy időegység alatt dolgoz fel a gép.
  MIPS *(millions of instructions per second)*: A másodpercenként végrehajtott utasítások száma, milliókban megadva.

  MFLOPS: *(millions of floating point operations per second):* Az ún. lebegőpontos műveletek másodpercenkénti számát adja meg. (A lebegőpontos számokkal a következő fejezetben találkoztok.)

- **Teljesítőképesség**
  1. *Nagyszámítógépek* (**szuperszámítógép, mainframe computer**).
     Nagy végrehajtási sebességgel, nagy kapacitású tárolókkal, és nagy teljesítményű perifériákkal rendelkeznek. Legtöbb esetben több processzort tartalmaznak. (Pl.: IBM AS/400 Modell 740 típusú gépben 12 processzor található) Ezeket elsősorban nagyvállalati adatfeldolgozásokhoz (nagyon sok adat) és tudományos számításokhoz (nagy számítási igény) használják elsősorban.

  2. *Középszámítógépek* (**minigép, mini computer**)
     Az átlagos teljesítőképességük kisebb, mint a nagyszámítógépeké. Jellemzően folyamatvezérlési, termelésirányítási feladatok megoldására, térinformatikai rendszerek, mérnöki tervező rendszerek, használják őket ún. **munkaállomás**ként.

  3. *Kisszámítógépek* (**mikrogép, microcomputer**)
     Teljesítményük viszonylag alacsony. Általában személyi számítógépként, vagy hálózatba kötve használják őket. Az alacsony teljesítményt sok esetben a perifériák (tárolók, nyomtatók stb) kis teljesítménye okozza. A nagyobb teljesítményűek munkaállomásként, ill. szerverként is használhatók.

1.6 Számrendszerek

Egy tízes számrendszerbeli szám mindig felírható egy 10 hatványait tartalmazó polinomként.

1. Minden egyes pozíciónak (helyiértéknek) meghatározzuk az értékét. Legyen a szám:

Számjegy	1	9	9	3	,	0	3	0	7
Pozíció	3	2	1	0		-1	-2	-3	-4
Szorzó	10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}	10^{-4}

2. 10 hatványait képezzük az egyes pozícióknak, mint kitevőknek a segítségével. A hatványértékeket annyiszor vesszük, mint amennyi az adott pozícióra írt együttható.

3. Az így kapott értékeket összeadjuk.

$$1993{,}0307 = 1*10^3+9*10^2+9*10^1+3*10^0, 0*10^{-1}+3*10^{-2}+0*10^{-3}+7*10^{-4}$$

Ebben a formában felírhatunk más számrendszerbeli számokat is. Észrevehetjük, hogy a tízes számrendszerben összesen 10 számjegy van: 1,2,3,4,5,6,7,8,9,0. Általában - más számrendszerben is - igaz, hogy egy N alapú számrendszerben N db számjegy áll rendelkezésünkre, a legnagyobb számjegy N-1.

1.6.1 Kettes számrendszer

Kettes számrendszerben az alapszám 2, tehát két számjegy van értelmezve: 0 és 1. Egy kettes számrendszerbeli *(bináris)* szám tízes számrendszerbeli értékét a következőképpen kaphatjuk meg: az egyes helyiértékeket elfoglaló bináris számjegyeket (0 vagy 1) megszorozzuk kettőnek a helyiértékéből adódó hatványával, majd a kapott értékeket összeadjuk.

Legyen egy kettes számrendszerbeli szám **10011**! Tízes számrendszerben ez a következőképpen írható fel:

$$1*2^4+0*2^3+0*2^2+1*2^1+1*2^0 = 19.$$

Törtszámoknál is hasonlóképpen járhatunk el. Pl: **0.1011** esetén:

$$2^{-1}+0*2^{-2}+2^{-3}+2^{-4}=1/2+0+1/8+1/16=0.6875$$

Írjunk át egy decimális számot binárissá! Pl.: Legyen a szám **183,79!**

- **A szám egészrésze:** Az egész rész átírása sorozatos osztásokkal végezhető el, és a maradékok adják a bináris számrendszerbeli számjegyeit a számnak.

183	: 2
91	1
45	1
22	1
11	0
5	1
2	1
1	0
0	1
Hányados	maradék

A **maradékokat** *alulról-felfelé* kell olvasni:

$$183_D = 10110111_B$$

- **A szám törtrésze:** A kapott szorzat törtrészét kell mindig 2-vel szorozni.

0, 79	*2
1	58
1	16
0	32
0	64
1	28
0	56
1	12
0	24
Egészrész	törtrész

A szorzatok **egészrészeit** *felülről-lefelé* kell olvasni:

$$0{,}79_D = 0{,}11001010_B.$$

Így $183{,}79_D = 10110111{,}11001010_B$.

1.6.2 Összeadás és szorzás kettes számrendszerben

A számítógép szinte minden műveletet el tud végezni, de valójában csak összeadásra van szükség, hogy a többi műveletet ennek a segítségével végezhesse el.

Az összeadás műveleti szabályai:

+	0	1
0	0	1
1	1	0

Legyen például a két összeadandó szám **1001** (9) és **1110** (14)!

```
    1  0  0  1
    1  1  1  0
+-------------------
 1  0  1  1  1
```

Az 1+1 =0 és az ún. *átvitel* 1. (Hasonlóképpen a tízes számrendszerben például 5+7=2, és az átvitel szintén 1.)

A szorzás összeadási szabályai:

*	0	1
0	0	0
1	0	1

Legyen a két összeszorzandó szám: **1001** (9) és **1110** (14)!

1	0	0	1	*	1	1	1	0
1	0	0	1					
	1	0	0	1				
		1	0	0	1			
			0	0	0	0		
1	1	1	1	1	1	0		

1.6.3 Hexadecimális számrendszer

A hexadecimális számrendszerben, - vagy másképpen a tizenhatos számrendszerben - 16 számjegy van:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Az A,B,C,D,E,F betűk rendre a 10,11,12,13,14,15 számoknak megfelelő *számjegyeket* jelentik. Legyen például a tizenhatos számrendszerbeli szám: **1A0F4**! Decimális megfelelője:

$$1*16^4+10*16^3+0*16^2+15*16^1+4*16^0 = 106\,740.$$

Törtszámok felírása hasonlóképpen történhet. Legyen például a szám **0,3EC!** Ennek decimális megfelelője:

$$3*16^{-1}+ 14*16^{-2} +12*16^{-3} = 0.245117187.$$

Írjunk át egy decimális számot hexadecimálissá! Legyen a szám most is **183,79**!

- **A szám egészrésze:**

183	: 16
11	7
0	11
hányados	maradék

A **maradékokat** *alulról-felfelé* kell olvasni:

$$183_D = B7_H$$

- **A szám törtrésze:**

0, 79	*16
12	64
10	24
egészrész	törtrész

Az **egészrészeket** *felülről-lefelé* kell összeolvasni:

$$0{,}79_D = 0{,}CA_H.$$

Így $183{,}79_D = B5{,}CA_H$.

Megjegyzés:

Nyilvánvaló, hogy a törtszámok átszámítása nem mindig végezhető el pontosan.

Egy adott számrendszerben megadott szám átszámítását egy másik számrendszerbe *konvertálásnak* nevezzük. Ha egy számrendszer alapszáma egy másik számrendszer alapszámának egész kitevős hatványa, akkor a konvertálás elég egyszerűen elvégezhető.

Binárisból hexadecimálisba:

Mivel $16 = 2^4$, ezért a bináris számot a bináris ponttól balra (és ha a szám nem egész szám, akkor jobbra is) bitnégyesekre osztjuk, majd az így kapott bitnégyeseket mint önálló hexadecimális számjegyeket értelmezzük.

Például:

$$11\ 1011\ 1010\ ,\ 1010\ 1110_B = 3BA{,}AE_H$$
$$3\quad B\quad A\qquad A\quad E$$

Hexadecimálisból binárisba:

A hexadecimális számjegyeket külön-külön átírjuk binárissá.

Például:

$$A81,FC_H = 1010\ 1000\ 0001,\ 1111\ 1100_B$$

1.7 Fixpontos számábrázolás

A fixpontos ábrázolásnál a bináris pont („kettedes vessző") fix helyen van. Ez *általában* az utolsó pozíció utáni helyet jelenti, és így *egész számokat* adhatunk meg vele. (Ha a fixpont előrébb van, akkor törtszámokat is lehet vele ábrázolni, de ez nem jellemző, legfeljebb a lebegőpontos ábrázoláson belül.) Mivel a bináris pont mindig ugyanazon a helyen van, ezért ezt külön tárolni nem szükséges.

Alapvetően kétféle fixpontos ábrázolást használnak a gyakorlatban:

- **Előjeles:** A szám negatív, nulla, pozitív is lehet. Ilyen *Turbo Pascalban* a 2 bájtos *integer* típusú szám.
- **Előjel nélküli:** A szám csak 0 vagy pozitív lehet. Ez egyszerűen a kettes számrendszerbeli alakja a számnak. Ilyen például a *Turbo Pascalban* a 2 bájtos *word* típusú szám.

Az **előjeles fixpontos** ábrázolásoknak is többféle típusa használatos:

1.7.1 Abszolútértékes ábrázolás

Az első bit előjelbit:

- 0: 0 illetve pozitív szám
- 1: negatív szám.

A többi helyen a szám abszolút értékét ábrázoljuk, binárisan. Leginkább a lebegőpontos számábrázoláson belül használják ezt az ábrázolási módot.

1.7.2 Kettes komplemens ábrázolás

A továbbiakban az *integer* típussal szemléltetjük a fixpontos számábrázolást.

Előjeles fixpontos számok esetén az első bitet (balról az elsőt, azaz a legmagasabb helyiértékűt) előjelbitnek tekintjük.

- **Pozitív számok és 0 esetén:**
  A számokat bináris alakban ábrázoljuk, az előjelbit értéke: 0.
  Például: ábrázoljuk **100011**-t 16 bit hosszúságú fixpontos számként.

0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1

Tehát a bináris szám jegyei előtt fennmaradó összes pozícióba 0 kerül. A legnagyobb pozitív szám, amelyet így ábrázolhatunk 15 darab 1-est tartalmaz. Ennek az értéke: $2^{15}-1 = 32767$.

- **Negatív számok esetén:** A számot kettes komplemensével ábrázoljuk. Az előjelbit: 1.
  Kettes komplemens: A szám minden bitjét az ellenkezőjére változtatjuk, majd az így kapott számhoz hozzáadunk egyet.
  Legyen a szám, amelyet 2 bájton ábrázolunk: **-29!**

1. *lépés:*

A szám abszolút értékét ábrázoljuk binárisan:
+29 = 11101

0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

2. *lépés:*

Invertáljuk a számot, azaz minden egyes pozícióban az ott levő bitet az ellenkezőjére változtatjuk:

1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	0

3. *lépés:*

A legalacsonyabb helyiértéken 1-et hozzáadunk:

1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1

Ellenőrzés:

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

A bal szélső bitet *álátvitelnek* nevezzük, és figyelmen kívül hagyjuk.

A legnagyobb abszolútértékű, 16 biten ábrázolható negatív szám: $-2^{15} = -32768$.

1.7.3 Analógia a kettes komplemensre a tízes számrendszerben

9-es komplemens: Legyen a szám 456!

999 - 456 = 543

10-es komplemens: 1000 - 456 = 544

Tehát 544 = 543 +1

A cél: olyan aritmetikai egység, amelyben nincs szükség kivonó egységre, a kivonást visszavezetjük összeadásra.

Például: Számoljuk ki, mennyi **873 - 456 !** Feltesszük, hogy a számok ábrázolására **3** pozíció (helyiérték) áll rendelkezésre.

873-456 = 873+(-456) = 873+(1000-456)-1000 = 1417-1000 = 417.

Az utolsó kivonásra (1417-1000) nincs szükség, ugyanis csak 3 pozíció áll rendelkezésre, ezért az 1417-ben az első 1-es túlcsordul, ami azt jelenti, hogy az eredményt tároló helyen csak 417 jelenik meg, ami a helyes eredmény. Van még egy kivonás (1000-456), amelyet el kell végezni, de észrevehetjük, hogy az a 456 10-es komplemense. Tízes számrendszerben nem tudjuk „megspórolni" a 10-es komplemens kiszámításához a kivonást.

Hasonló a helyzet a kettes számrendszerben, de ott természetesen 2-es komplemens szerepel a 10-es komplemens helyén. Viszont a kettes számrendszerben a 2-es komplemenst kivonás nélkül is végre lehet hajtani: az invertálás és az összeadás segítségével.

Az invertálás eredményét szokták 1-es komplemensnek is nevezni, ez nyilván a 9-es komplemensnek felel meg, és kiszámolhatjuk úgy is, hogy a csupa 1-esből álló, maximálisan lehetséges hosszúságú számból kivonjuk az adott számot. A 2-es komplemens pedig nem más, mint a 100...0 számból kivonva az adott szám, úgy, hogy eggyel több számjegyből áll az 100...0, mint a rendelkezésre álló pozíciók száma.

A kettes komplemensben megadott szám visszaalakítása teljesen hasonló módon történik, mint maga az ábrázolás. Azaz először invertálni kell minden egyes pozíció tartalmát, utána pedig a legkisebb helyiértéken 1-et hozzá kell adni.

Például: Legyen a szám kettes komplemensben megadott alakja:

11111111 11100011!

1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1

1. *lépés:* invertálás.

0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0

2. *lépés:* 1 hozzáadása.

0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Tehát az eredmény: -29, (+29-et kaptunk, de negatív a végeredmény, hiszen kettes komplemensben volt ábrázolva.)

Ha két olyan nagy számot adunk össze, hogy az eredmény már nem fér el az adott hosszon, (pl. 15 biten) úgynevezett *túlcsordulás keletkezik.* Ez az eset csak azonos előjelű számok összevonásakor léphet fel, hiszen az eredmény abszolútértéke csak akkor lesz nagyobb bármelyik összeadandónál. A túlcsordulást az előjelbit indokolatlan megváltozása jelzi.

Például: Legyen a két szám:

00100000 00100101 és 01100000 00010010!

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

Az eredmény első bitje 1-es, amely negatív számot jelez. Ez nyilvánvalóan rossz eredmény, hiszen az operandusok pozitívak. Az eredmény jegyei már nem fértek el a 15 biten, túlcsordultak a 16. pozícióra, és ez okozta az előjel megváltozását. Szerencsére a számítógép műveletvégzés közben állandóan figyeli, hogy nem következett-e be túlcsordulás. Ha igen, akkor jelzi. Erre nyilván a felhasználónak is figyelnie kell, és ha nagyobb számokkal kell dolgoznia, akkor vagy másik fixpontos típust kell választania - ha van ilyen - vagy lebegőpontos számot.

1.7.4 Többletes ábrázolás

Minden számhoz hozzáadunk egy adott értéket, a többletet. Ha például 8 biten ábrázoljuk a számot, akkor a többlet $128(=2^7)$. (Általában n bit esetén 2^{n-1} az eltolás.) A rendelkezésre álló 8 biten a 0 értéket megegyezés szerint az *alapérték,* az $1000\ 0000_B = 128_D$ segítségével ábrázolják. Ez tulajdonképpen úgy keletkezett, hogy a 8 biten kifejezhető 2^8=256-féle különböző számot tartalmazó tartományt megfeleztük. A felezőpontot (128) tekintjük 0-nak. A pozitív kitevő értékét az alapértékhez hozzáadjuk, a negatív kitevő értékét pedig az alapértékből levonjuk. Így az egész számok –128 és 127 közé esnek.

Például: +5 többletes ábrázolása: 133. –5 esetén: 123.

1.8 Lebegőpontos számábrázolás

A lebegőpontos számábrázolással *valós számokat* ábrázolhatunk. Tulajdonképpen egész- és törtszámok ábrázolására egyaránt alkalmas.

Például a *Turbo Pascalban* használható a *real* típus, amely 6 bájtos lebegőpontos ábrázolású. A továbbiakban a *real* típussal szemléltetjük a lebegőpontos ábrázolást.

1.8.1 Normálalak

Minden szám felírható a következő formában:

$$N = m \cdot 2^k$$

ahol

- m mantissza, amelyre teljesül: $1/2 < |m| < 1$. Ez a feltétel azt jelenti, hogy a valódi tört első számjegye nem lehet 0.
- k karakterisztika, a kitevőt jelenti. Nagysága a bináris pont eltolásának a mértékét, az előjele pedig az eltolás irányát adja meg. Az eredeti bináris pont balra léptetésekor a kitevő előjele pozitív, a jobbra léptetéskor negatív.
  Pl: $-0.000111011 = -0.111011 \cdot 2^{-3}$
  $10.00110 = 0.1000110 \cdot 2^{+2}$

Mivel a mantissza első, legmagasabb helyiértékű jegye mindig 1, ezért ez nincs tárolva, helyette a mantissza előjele van itt.

Hasonlóképpen nincsen tárolva a hatványalap (2), hiszen ez is minden esetben ugyanaz.

A mantissza általában abszolútértékes, vagy kettes komplemens ábrázolású. A *real* típusú számok *abszolútértékes* ábrázolásúak.

S	mantissza (m)	k+128
1 bit	39 bit	8 bit

S előjelbit: ha a mantissza pozitív, akkor S=0,
ha negatív, akkor S=1.

A karakterisztika eltolt ábrázolású, azaz a valódi értékét úgy kapjuk, hogy a tárolt értékből 128-at kivonunk, azaz *128 többletes ábrázolású.*

Például: a +5 kitevőt a következőképpen ábrázoljuk:

```
      1000 0000 = 128
+          101 = 5
-------------------
      1000 0101 = 133
```

-5 esetén

```
      1000 0000 = 128
-          101 = 5
-----------------------
      0111 1011 = 123
```

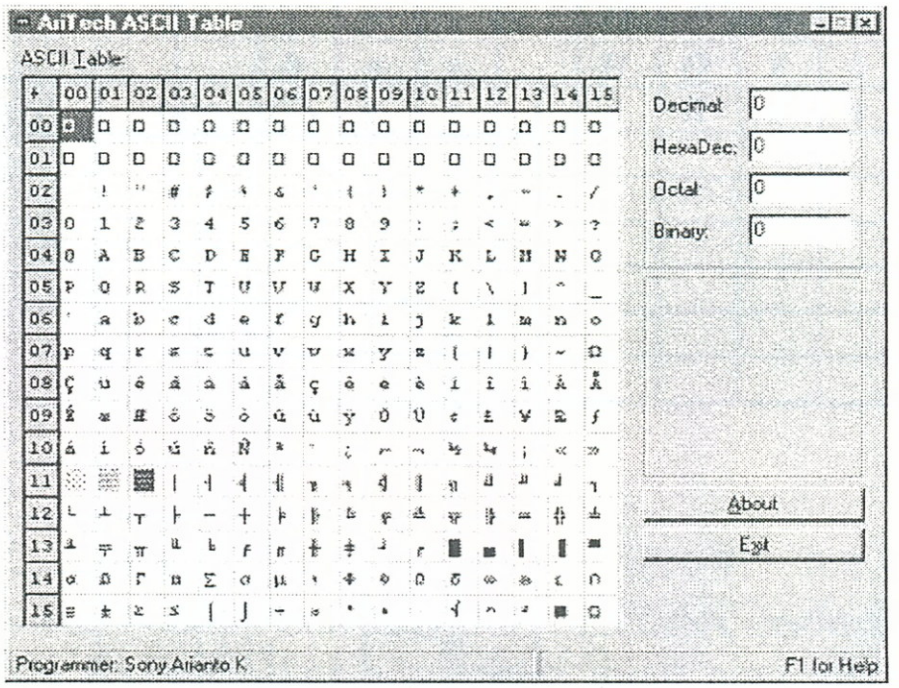
A többletes karakterisztika ábrázolás előnye, hogy nem kell külön feltüntetni valamilyen formában a kitevő előjelét, mégis azonnal megállapítható a 8 bitből: ha az ábrázolt érték 1-gyel kezdődik, akkor pozitív a karakterisztika, ha 0-val, akkor negatív.

1.9 Kódrendszerek

Mint láttuk, az információt a számítógépben kódolt formában kell tárolni. A karakterek kódolására a számítógépek különböző kódrendszereket használnak. A legfontosabb kódrendszerek a következők.

1.9.1 ASCII kódrendszer

A számjegyek, betűk, írásjelek és egyéb jelek (azaz a karakterek) számítógépes ábrázolására úgynevezett *bináris kódokat* használunk. Általában 8 bitet használ a gép a karakterek ábrázolására, így összesen 256 (2^8) karakter ábrázolására van lehetőség. A PC-ken a legelterjedtebben az úgynevezett ASCII kódrendszert alkalmazzák (*American Standard Code for Information Interchange Code*). A nemzetközileg elismert ASCII szabvány 7 bites. Ez magában foglalja az angol ABC kis- és nagybetűit, a decimális számjegyeket, az írásjeleket, és az úgynevezett adatátviteli vezérlőjeleket. Az IBM cég által a PC-ben használt ASCII 8 bites (**437-es kódtáblázat**) és az előbbieken kívül tartalmaz grafikus jeleket, néhány, az angol ABC-n kívüli betűt, és más egyéb karaktereket is. A teljes magyar ABC-t tartalmazza a **852-es kódtáblázat**.



A vezérlőkaraktereknek megfelelő bitkombinációk általában valamilyen tevékenységet váltanak ki abból a berendezésből, amelyikre küldjük őket, pl. **LF** - soremelés, **BEL** - csengő megszólaltatása.

1.9.2 EBCDIC kódrendszer

(EBCDIC: *Extended Binary Coded Decimal Interchange Code)*

8 bit használható, de ez általában nincs teljesen kihasználva, így a kódolandó karakterkészletet csoportokba lehet sorolni, illetve megkülönböztetni. Az egyes csoportokba közel azonos darabszámú karakter tartozik. Az EBCDIC a karakterek ábrázolására a 8 bitet két 4 bites részre osztja. Az első rész a *zónarész*, a második a *számrész*. Az egyes csoportokat a zónarésszel lehet jellemezni.

0-9	1111
A-I	1100
J-R	1101
S-Z	1110

A csoporton belül az egyes karaktereket a számrésszel lehet azonosítani. Például a számok esetén:

	Zónarész	Számrész
0	1111	0000
1	1111	0001
2	1111	0010
3	1111	0011
4	1111	0100
5	1111	0101
6	1111	0110
7	1111	0111
8	1111	1000
9	1111	1001

Például:

1100	1001
zónarész	számrész
C	9

Legyen a kódolandó szó: MOZI

Hexadecimális alakban az EBCDIC kódja: D4 D6 E9 C9

		ZÓNA														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0					SP	&	-									0
	1							/		a	j			A	J		1
										b	k	s		B	K	S	2
	3									c	l	t		C	L	T	3
	4									d	m	u		D	M	U	4
	5									e	n	v		E	N	V	5
	6									f	o	w		F	O	W	6
S	7									g	p	x		G	P	X	7
Z	8									h	q	y		H	Q	Y	8
Á	9									i	r	z		I	R	Z	9
M	A						!	^	:								
	B					.	§	,	#								
	C					<	*	%	@								
	D					(	)	-	'								
	E					+	;	>	=								
	F							?	"								

1.9.3 BCD kód

(BCD: ***B**inary **C**oded **D**ecimal* = *binárisan kódolt decimális*)A decimális számjegyek EBCDIC kódú ábrázolásakor a zónarész miatt nem lehet aritmetikai műveleteket végezni. Ha megállapodunk abban, hogy csak számokat ábrázolunk, akkor az egyéb karakterek csoportjait nem szükséges ábrázolni. Így eljutunk a decimális számjegyeknek egy 4 bites kódjához, ez a BCD kód. A rendelkezésre álló 8 biten 2 binárisan kódolt decimális számjegyet lehet ábrázolni. Az egyes számokat úgy ábrázoljuk, hogy a számjegyeket binárisan írjuk fel, és a decimális számrendszer helyi értékeinek megfelelően helyezkednek el.
Például:

A **24** BCD kódja: **0010 0100**

1.9.4 Zónázott decimális ábrázolás

Észrevehetjük, hogy minden decimális számjegy kódjának zónarésze: 1111 (= F). Minden számjegy ábrázoláshoz 8 bit szükséges. Ezek a számok természetesen pozitívnak tekintendők. Előjeles számok esetén az előjelet a szám legalacsonyabb helyiértékű számjegyének zónarészében jelezzük.

Az előjel kódolása:

+ előjel: **1100 (C)**
– előlel: **1101 (D).**

Például **+269** esetén:

1111	0010	1111	0110	1100	1001
F	2	F	6	C	9

–269

1111	0010	1111	0110	1101	1001
F	2	F	6	D	9

1.9.5 Tömörített decimális ábrázolás

Ha csak számokat akarunk ábrázolni, akkor a zónarész elmaradhat. Így a 8 biten már 2 decimális számjegyet lehet ábrázolni. Ilyen tömörítéssel az ábrázolandó szám által lefoglalt bitek száma *közel* felére csökken. Azért nem *pontosan* a felére, mert az előjel ábrázolására szükség van még 4 bitre. Elképzelhető, hogy félbájtnyi hely kihasználatlan marad.

Például:

1. ***Előjel nélküli számok***

Legyen a szám **126**!

Zónázott alak:

F1	F2	F6

Tömörített alak:

01	26

Legyen a szám **3452!**

Zónázott alak:

F3	F4	F5	F2

Tömörített alak:

34	52

2. *Előjeles számok:*

Legyen a szám **-126!**

Zónázott alak:

F1	F2	D6

Tömörített alak:

12	6D

Legyen a szám **3452!**

Zónázott alak:

F3	F4	F5	C2

Tömörített alak:

03	45	2C

1.10 Logikai áramkörök

A számítógép teljesen elektronikus működésű. A bonyolult integrált áramkörök logikai áramkörökből épülnek fel. A logikai áramkörök logikai műveletek sokaságát végzik el egy másodperc alatt.

Ítélet:

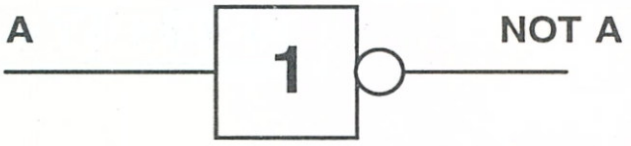
Ítéletnek nevezünk minden olyan állítást, kijelentést, amelyről egyértelműen eldönthető, hogy *igaz* vagy *hamis*. Például:

„A Duna egy folyó." – ítélet.
„Bécs Svájc fővárosa." – ítélet.
„Hogy vagy?" – nem ítélet.
A kijelentésekhez logikai értékeket rendelhetünk, attól függően, hogy igaz vagy hamis.

igaz	1
hamis	0

1.10.1 Logikai műveletek

- **Tagadás:**
  Például: *„A tábla fekete."* Ennek a tagadása a következő: *„A tábla **nem** fekete."* (Nem pedig az, hogy fehér.)
  Jelölés: **NOT A**
  Áramköri jelölés:



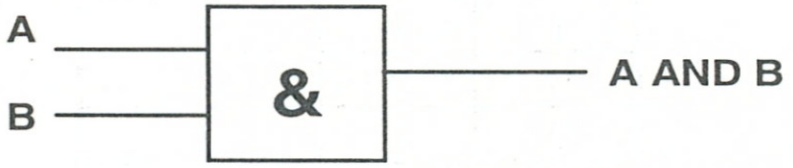
Igazságtáblázat:

A	NOT A
1	0
0	1

- **És:** Az eredmény akkor lesz igaz, ha mind a két operandus igaz. Például:
  A: „Elmegyek a boltba."
  B: „Gumikalapácsot veszek."
  A és **B:** „Elmegyek a boltba, ***és*** gumikalapácsot veszek."

  Ahhoz tehát, hogy gumikalapácshoz jussunk, mind a két állításnak igaznak kell lenni.

Jelölés: **A AND B**
Áramköri jelölés:



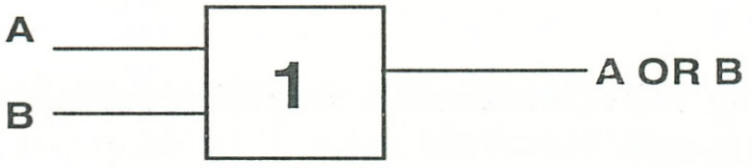
Igazságtáblázat:

A	B	A AND B
1	0	0
1	1	1
0	0	0
0	1	0

- **Vagy:** Az eredmény csak akkor hamis, ha mind a két operandus hamis. Például:
  A: „Ajándékba kapok ma egy kék esernyőt."
  B: „Veszek magamnak egy piros esernyőt."
  A vagy **B:** „Ajándékba kapok ma egy kék esernyőt, ***vagy*** veszek magamnak egy piros esernyőt."

Ahhoz, hogy legyen végre egy szép színes esernyőnk, elég ha A és B közül legalább az egyik igaz.

Jelölés: **A OR B**
Áramköri jelölés:



Igazságtáblázat:

A	B	A OR B
1	0	1
1	1	1
0	0	0
0	1	1

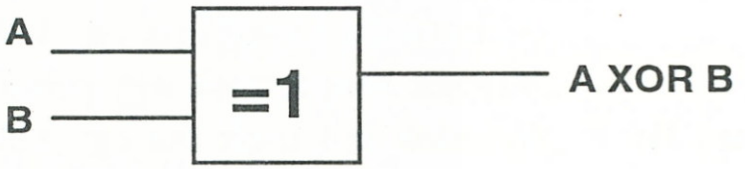
- **Kizáró vagy:** Az eredmény akkor igaz, ha az operandusok különbözőek. Például:

  A: „Ma délelőtt tíz órakor az iskolában leszek."

  B: „Ma délelőtt tíz órakor moziban leszek."

  A vagy **B:** „Ma délelőtt tíz órakor vagy az iskolában vagy a moziban leszek."

  Vagyis a két lehetőség közül csak az egyik lehetséges.
  Jelölés: **A XOR B**
  Áramköri jelölés:

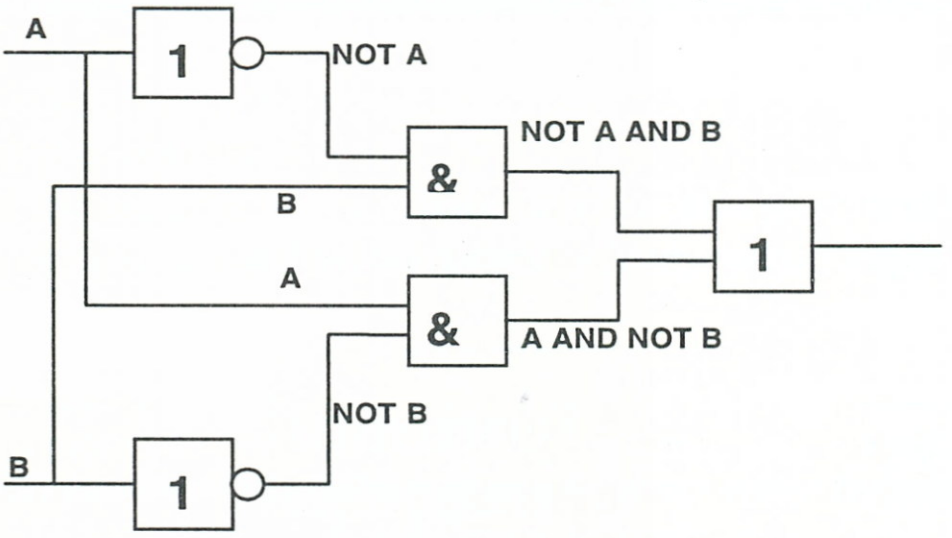


Igazságtáblázat:

A	B	A XOR B
1	0	1
1	1	0
0	0	0
0	1	1

Ezeknek az műveleteknek a segítségével bonyolultabb logikai kifejezések is felépíthetők.

Például: A (NOT A AND B) OR (A AND NOT B) kifejezés **blokkvázlata**:



1.10.2 A Boole-algebra alapjai

George Boole (1815-1864) tételeinek segítségével logikai kifejezéseket átalakíthatunk, egyszerűsíthetünk. Az összetett logikai logikai kifejezések kiértékelését segítik a következő alaptörvények:

1. A OR NOT A = 1
2. A AND NOT A = 0
3. NOT (NOT A) = A (A tagadás tagadása maga az állítás)
4. NOT (A OR B) = NOT A OR NOT B
5. NOT (A AND B) = NOT A + NOT B
6. A OR 0 = A
7. A AND 0 = 0
8. A OR 1= 1
9. A AND 1 = A
10. A OR (A AND B) = A AND (1 OR B) = A

Az egyes logikai értékeknél meghatározott feszültségsávok képviselik az **igaz** és **hamis** állítást. Ha például 5 V-os tápfeszültség áll rendelkezésünkre, akkor a külső terheléstől függően +3 és 5 V közötti feszültségek jelzik az igaz állapotot, a 0 és 0.5 V közöttiek pedig a hamisat.

2. ALGORITMUSOK

Egy probléma megoldása során általában meghatározott lépések sorozatát hajtjuk végre, azaz egy algoritmust követünk.

Az algoritmus tehát tevékenység-sorozatot jelöl, amelyet végrehajtva eljuthatunk a kívánt eredményhez. Nyilvánvaló, hogy egy adott feladat megoldásához különböző algoritmusokat találhatunk. Az algoritmusokkal szemben a következő elvárásokat fogalmazhatjuk meg:

1. **Általános érvényű legyen.**
   Az egymástól csak a bemenő adatokban különböző feladatok megoldása is lehetséges vele.
2. **Véges számú lépés után véget ér.**
3. **Egyértelműen meghatározott lépések sorozatából áll.**
   Minden részlépésnek van egy rákövetkezője, kivéve persze a legutolsót.

Nyilvánvaló, hogy az algoritmust „kitalálása" után megpróbáljuk olyan formában megadni, hogy azt a számítógép is „megértse". Az algoritmusnak ezt a formáját nevezzük *programnak.*

A program részlépéseit *utasításoknak* nevezzük, a számítógép ezeknek az utasításoknak a sorozatát hajtja végre.

A számítógép természetesen nem képes eldönteni azt, hogy egy algoritmus helyes-e, vagyis, hogy azt csinálja-e, amit mi szeretnénk. Így írt erről *Neumann János:*

„A számítógépeknek sajátosságuk, hogy mindazt, amit velük közöltek, abszolút komolyan és szó szerint veszik, azaz valóban azt végzik el, amit előírtak nekik, beleértve a sajtó- és gondolkodási hibákat is."

Nem lényegtelen egy algoritmus működésekor, hogy milyen a hatékonysága.

Ezt általában két szempontból szokták vizsgálni:

1. Mekkora a memóriaigénye?
2. Mennyi a végrehajtási idő?

A legtöbb esetben a két hatékonysági mutató csak egymás rovására javítható.

2.1 A változó fogalma

Az algoritmusban egy adat vagy állandó értékkel bír, vagy megváltoztathatja az értékét. Az állandó értékkel bíró adatokat *konstansoknak* nevezzük. Pl.: 3,14. Azt az adatot, amely az algoritmusban megváltoztatja az értékét, *változónak* nevezzük.

A programban szereplő változókat a következőképpen jellemezhetjük:

1. Név: A változókat a programban névvel azonosítjuk.
2. Típus: A változó természetesen valamilyen adott típussal rendelkezik. (Pl.: szám, karakter, stb.), ami egyúttal meghatározza azt is, hogy milyen műveletek végezhetők el rajtuk, milyen értékhatárok között.
3. Érték: A változónak mindig van a programban valamilyen konkrét értéke.
4. Cím: A változó által tárolt adat tárbeli helyének a címe.

Megjegyzés:

Ezek a tulajdonságok a 4. kivételével az algoritmusokban szereplő változókra is érvényesek. A 4. tulajdonságot a programozónak általában nem kell figyelembe vennie.

2.2 Alapvető lépések

Az algoritmusok általában a következő elemi lépésekből épülnek fel:

- **Beolvasás**
  Az algoritmusnak rendszerint szüksége van „kívülről" bevitt adatokra, amelyekkel műveleteket végez.

- **Kiírás**
  Az algoritmus végcélja többnyire az, hogy a „külvilág" felé adatokat közöljön, mintegy eredményképpen.

- **Értékadás**
  Az értékadás jele általában a „ := ". Pl.: *x:= 5*. A baloldalon álló változónak értéket adunk a jobboldal segítségével, ahol konstans és változó is állhat, illetve ezek összekapcsolása műveletekkel (*kifejezésekké*). A következő szabályoknak kell teljesülniük az értékadásnál:

  1. Az értékadás jobb oldalán csak konstansok, vagy olyan változók szerepelhetnek, amelyek előzőleg már értéket kaptak.
  2. Az értékadás bal oldalán szereplő változó az előző tartalmát elveszti.

  Például: $x := x + 1$ a következőt jelenti: az x változó új értékét megkapjuk, ha a régi értékéhez hozzáadunk *1*-et. Ha például eddig *2* volt, akkor ezután *3* lesz.

- **Elágazás**
  Egy feltétel teljesülésétől vagy nem teljesülésétől függően különbözőképpen folytatjuk az algoritmust. Itt egy döntésről van szó, aminek eredményeként egyik illetve másik irányban folytatódik az algoritmusunk. Például: Ha $x= 0$ akkor legyen y új értéke 2, egyébként pedig maradjon változatlan.

- **Ciklus**

  Sokszor előfordul, hogy egy bizonyos részlépést vagy részlépéseket többször is végre kell hajtani. Ezt nevezzük *ciklusnak*, és amit ismételten végre kell hajtani, azt *ciklusmagnak*. Ahhoz, hogy a ciklusból egyszer ki tudjon lépni az algoritmus, szükség van egy ún. *ciklusfeltételre*, amely teljesülésétől függően fog végrehajtódni a ciklusmag. A ciklusfeltétel és a ciklusmag kétféleképpen helyezkedhetnek el:

  1. A ciklusfeltétel megelőzi a ciklusmagot: **elöltesztelő ciklus**
     Mivel a feltétel kiértékelésével kezdődik a ciklus végrehajtása, ezért lehet, hogy a ciklusmag egyszer sem hajtódik végre.

  2. A ciklusfeltétel a ciklusmag mögött helyezkedik el: **hátultesztelő ciklus**
     A ciklusmag legalább egyszer végrehajtódik, hiszen a feltétel vizsgálata a ciklusmag után következik.

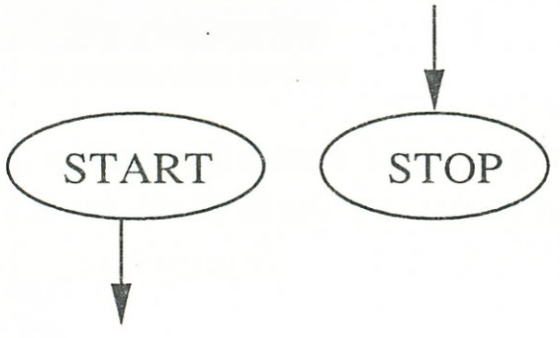
2.3 Algoritmusleíró eszközök

Az algoritmusok leírási módjára nincsenek megkötések, sok esetben elképzelhető akár szóbeli leírás is.

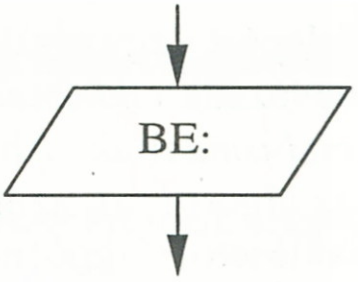
2.3.1 Folyamatábra

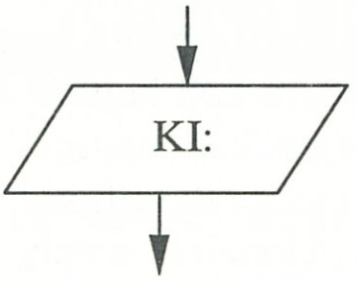
A folyamatábra az algoritmus részlépéseit különböző geometriai szimbólumokkal szemlélteti.

Ezek a szimbólumok a következők:

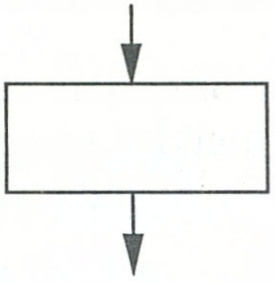


Határszimbólumok. A program elejét és végét jelzik

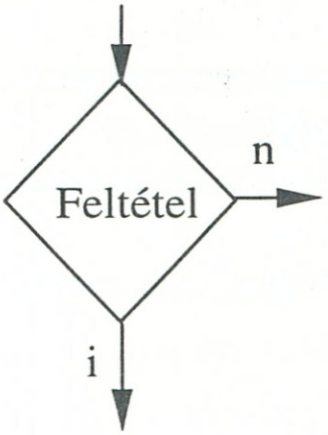




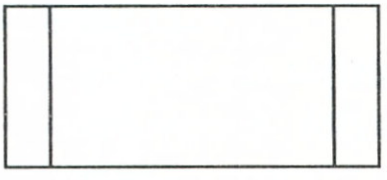
Beolvasó, kiíró utasítások.



Általános műveletvégzés. (Pl.:értékadás.)



Elágazás. Ha a rombuszban levő feltétel igaz, akkor az i irányában, ellenkező esetben az n irányában folytatódik.

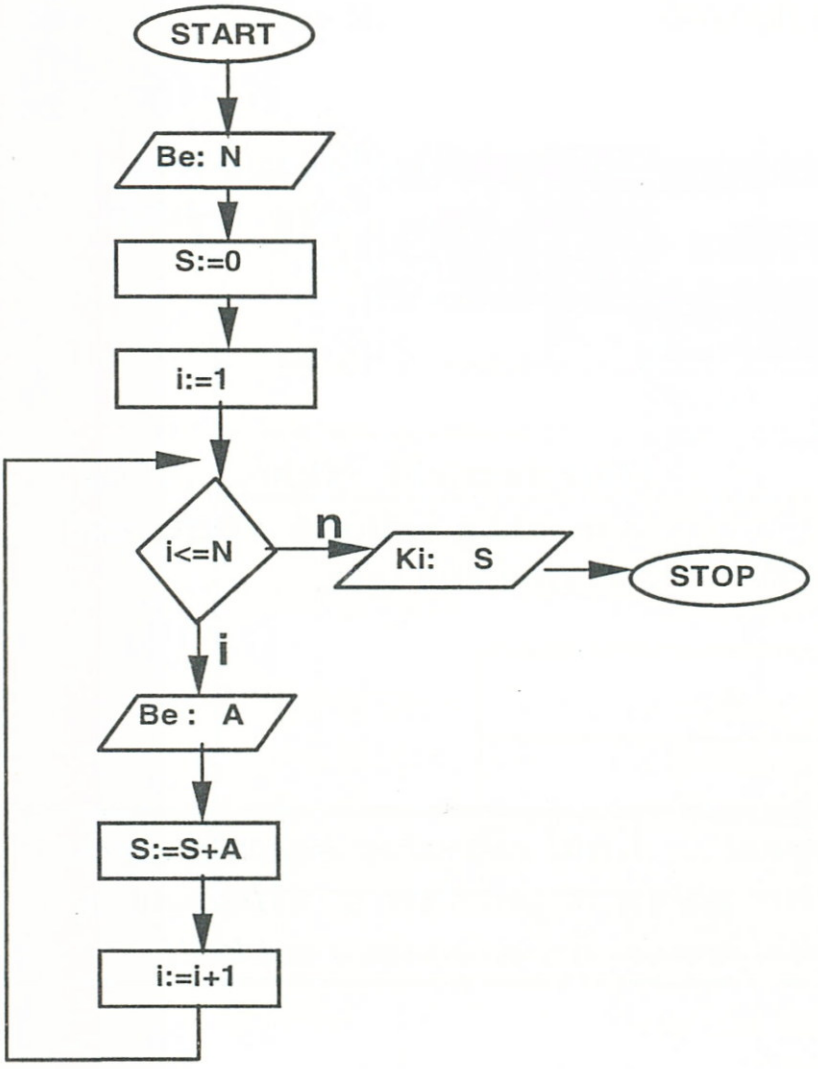


Részalgoritmus. Ezzel jelezzük, hogy a későbbiekben részletezzük az itt következő lépéseket.

Megjegyzés:

Ciklusnak megfelelő szimbólum nincs, ha szükségünk van rá, akkor azt külön kell összerakni. Erre látunk példát a következő feladatban.

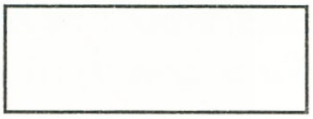
Példa: Határozzuk meg N db beolvasott szám összegét!



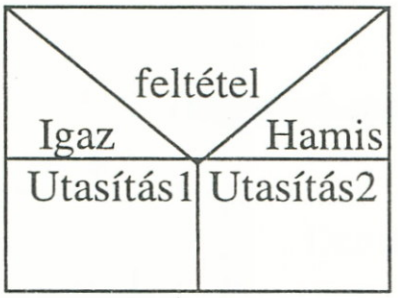
2.3.2 Struktogram

Az egész algoritmust egy téglalapba írjuk be. Ennek a téglalapnak a felosztásával ábrázoljuk az algoritmust. Az algoritmuson belül **felülről - lefelé** kell haladnunk.

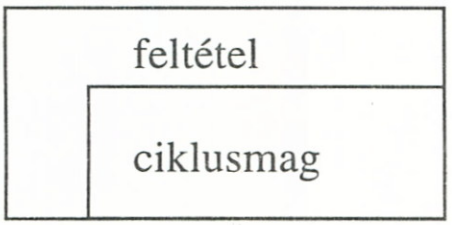
A következő szimbólumokat használhatjuk:



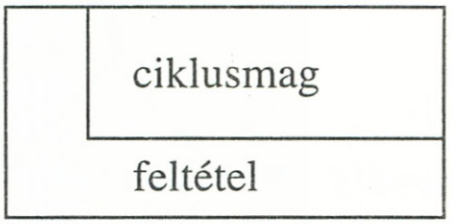
Általános műveletvégzés. (Pl.: értékadás, beolvasás, kiírás.)



Elágazás. Ha a *feltétel* teljesül, akkor az *utasítás1* hajtódik végre, ha nem, akkor az *utasítás2*.



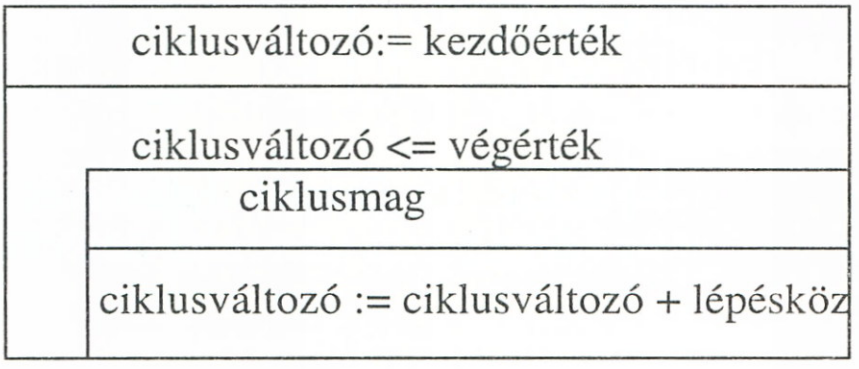
Elöltesztelő ciklus. Amíg a *feltétel* igaz, addig hajtódik végre a *ciklusmag*.



Hátultesztelő ciklus. A *ciklusmag* addig hajtódik végre, amíg a *ciklusfeltétel* igaz.

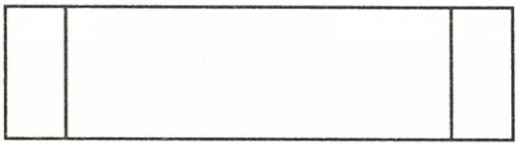
Megjegyzés:

*Ha **Pascalban** szereplő **Repeat ... Until** ciklushoz hasonló szerkezetet akarunk használni, akkor ebben az esetben a ciklusmag addig hajtódik végre, amíg a feltétel **hamis**. Ezt előre tisztázni kell!*



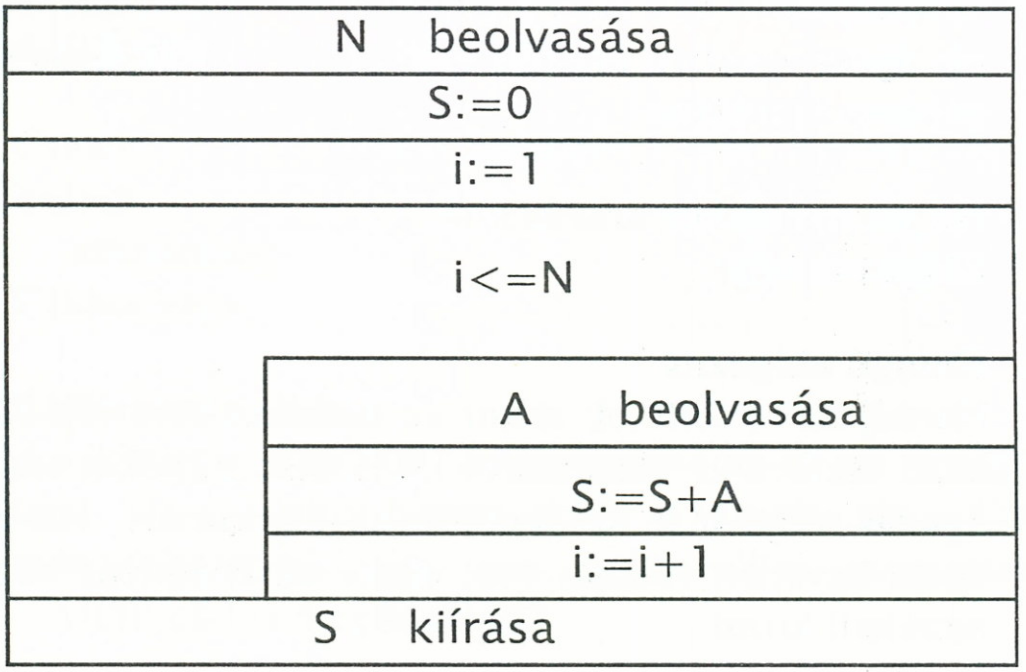
Számlálásos ciklus. Speciális elöltesztelős ciklus.

A *feltétel* itt az, hogy a *ciklusváltozó* <= *végérték*. A *ciklusmag* végrehajtása után minden alkalommal, a *ciklusváltozó* értéke *lépésközzel* változik. Mindaddig tart a ciklus végrehajtása, amíg a *feltétel* teljesül.



Részalgoritmus (Eljárás). Ezzel jelezzük, hogy a későbbiekben részletezzük az itt következő lépéseket. Az eljárást külön téglalapban kell megadni, itt csak hivatkozunk rá egy névvel.

Példa: Határozzuk meg N db beolvasott szám összegét!



2.3.3 Mondatszerű leírás

Az algoritmust egymás után következő „mondatokkal" írjuk le. Egy anyanyelven megfogalmazott, viszonylag pontos leírás. (Tulajdonképpen olyan, mintha egy „pszeudo" programozási nyelvet használnánk.)

A következő elemeket használhatjuk:

- **Beolvasó, kiíró utasítások**
  BE: *változók* [*az adatokkal szemben támasztott követelmények*]
  KI: *kifejezések* [*a kiírás formájára vonatkozó követelmények*]

- **Értékadás**
  A változó felveszi a jobb oldalon álló kifejezés aktuális értékét.

  Változó := kifejezés

- **Elágazások**

  1. **Egyágú elágazás**
     Ha teljesül a **feltétel**, akkor végrehajtódnak az utasítások, majd az **Elágazás vége** után folytatódik az algoritmus. Ha az elágazás ágain egyetlen utasítás van, akkor az **Elágazás vége** szöveget nem kell kiírni.

     Ha *feltétel* **akkor** *utasítás(ok)*
     Elágazás vége

  2. **Kétágú elágazás**
     Ha teljesül a **feltétel**, akkor az **utasítás(ok)1** lép életbe, ellenkező esetben az **utasítás(ok)2**. Ezután mindkét esetben az **Elágazás vége** után folytatódik az algoritmus. Ha az elágazás ágain egyetlen utasítás van, akkor az **Elágazás vége** szöveget nem kell kiírni.

     Ha *feltétel* **akkor** *utasítás(ok)1*
     különben utasítás(ok)2
     Elágazás vége

  3. **Többágú elágazás**
     Ha az **i. feltétel** teljesül, akkor a neki megfelelő **utasítás(ok)i** hajtódik végre, majd ezután az **Elágazás vége** után folytatódik az algoritmus. Ha egyik feltétel sem teljesül, akkor az **egyéb esetben** megadott **utasítás(ok)** hajtódik végre, illetve ha ezt

nem adjuk meg, akkor az **Elágazás vége** után folytatódik az algoritmus.

Elágazás
1. feltétel **esetén** . *utasítás(ok)1*
2. feltétel **esetén** . *utasítás(ok)2*
.
.
.
n. feltétel **esetén** . *utasítás(ok)n*
egyéb esetben *utasítás(ok)*
Elágazás vége

- **Ciklusok**

1. **Számlálásos ciklusutasítás**
A ciklusmag utasításai a **ciklusváltozó kezdő-** és **végértékének**, és a **lépésköznek** megfelelően kerülnek végrehajtásra. Ha a lépésköz 1, akkor elhagyható.

Ciklus *ciklusváltozó* = *kezdőérték* **–től** végérték **-ig** *lépésköz* **-ösével**
ciklusmag
Ciklus vége

2. **Elöltesztelő ciklus**
A ciklusmag utasításai addig hajtódnak végre, **amíg a feltétel igaz**. Ha a feltétel hamis, akkor a végrehajtás a **Ciklus vége** után folytatódik. (Ez nem a Pascalban megismert Repeat …Until ciklus megfelelője!)

Ciklus amíg *feltétel*
ciklusmag
Ciklus vége

3. **Hátultesztelő ciklus**
A ciklusmag utasításait addig hajtja végre a számítógép, **amíg a feltétel igaz**. Ha a feltétel hamis, akkor a végrehajtás a **Ciklus vége** után folytatódik. Mivel a vizsgálat a ciklus végén van, ezért egyszer mindenképpen lefut a ciklus.

Ciklus
ciklusmag
amíg *feltétel*
Ciklus vége

- **Eljárások**
  Az eljárásokat paraméterek nélkül is használhatjuk, ekkor az eljárás neve mögötti zárójeleket sem kell kiírni. Az eljárás hívása a neve leírásával történik.

Eljárásnév (paraméterek)
utasítás(ok)
Eljárás vége

2.3.4 Egyéb jelölések

Ha egy sorba több utasítást írunk, akkor közéjük kettőspontot kell írni.

Az algoritmus leírásába bárhová elhelyezhetünk megjegyzéseket, a kódoláshoz szükséges magyarázatokat szögletes zárójelek közé zárva.

Példa: Határozzuk meg N db beolvasott szám összegét!

```
Be: N
S:=0
Ciklus i:=1-től N-ig
    Be: A
    S:=S+A
Ciklus vége
Ki: S
```

3. LOGIKAI ADATSZERKEZETEK

A programokkal különböző típusú adatokat dolgozunk fel. A feldolgozásnál fontos, hogy milyenek lehetnek az egyes értékek, amellyel dolgozhatunk. Az adattípus segítségével adhatunk erre választ.

Az adattípus a következőktől függ:

1. Minden adattípus mögött van egy *belső ábrázolás*, amely az adott adattípusra jellemző.
   Például: az egész típusú változó esetén fixpontos ábrázolás.
2. Az adattípus értékkészlete: milyen *értékeket* vehet fel az adott típusú változó? Például: Turbo Pascalban a word típus értékkészlete: 0 - 65535 közötti egész számok.
3. Milyen *műveleteket* lehet végezni az adott típusú változón ?

Egy probléma megoldásakor több adattal is együtt kell dolgozni, ezek általában egymástól nem függetlenek, nem egyedi adatokból állnak, hanem adatcsoportokból, amelyekben valamilyen logikai összefüggés érvényesül. Ezek az összefüggések az *adatszerkezetek*.

A következőkben néhány fontosabb adatszerkezetet vizsgálunk meg.

3.1 Lineáris adatszerkezet

Ez egy általános jellegű adatszerkezet, amelyre a továbbiakban látunk példákat. A lineáris adatszerkezet a következőkkel jellemezhető:

1. Van egy első és egy utolsó eleme.
2. Az ezeken kívüli elemeknek létezik megelőzője, és rákövetkezője.
3. Az első elemnek csak rákövetkezője van, az utolsónak csak megelőzője.

3.2 Tömb

A tömb azonos típusú adatok sorozata. A tömböknél a logikai szerkezetet tulajdonképpen az adatelemek egymáshoz viszonyított elhelyezkedése adja. Így nyilván az első elem kitüntetett helyzetű, hiszen a többi elem helyét az elsőhöz viszonyíthatjuk.

3.2.1 Egydimenziós tömb (vektor)

Ez a legegyszerűbb tömbtípus. Az egyes elemekre egy számmal, az *index* segítségével hivatkozhatunk. Az index tulajdonképpen azt adja meg, hogy az adott elem hányadik a többi elem között. (Még akkor is, ha az indexek esetleg nem 0 -val kezdődnek, vagy esetleg nem is számok.)
Például a Turbo Pascal esetén a deklaráció formája:

var tomb : array[1..100] of integer;

Ennek a tömbnek a 6. eleme: *tomb[6]*

A tömbelemek a memóriában egymás után helyezkednek el, az indexeknek megfelelően. Az, hogy mekkora helyet foglal el a tömb a memóriában, az egyrészt az elemek szmától, másrészt az elemek típusától függ.

3.2.2 Kétdimenziós tömb

Itt az elemek logikailag egy táblázatban helyezkednek el. Például:

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}
A_{51}	A_{52}	A_{53}	A_{54}

Tehát a kétdimenziós tömb logikailag sorokból, és oszlopokból áll. Az első index általában a sor sorszámát jelenti, a második szám oszlop sorszáma.

Például a Turbo Pascal -ban a deklaráció formája:

var A: array[1..5, 1..4] of char;

Az A_{32} -re való hivatkozás: *A[3,2]* .

A memóriában az elemek általában *sorfolytonosan* helyezkednek el. Az elemek sorrendje az előző példánkon szemléltetve:

A_{11} A_{12} A_{13} A_{14} A_{21} A_{22} A_{23} A_{24} A_{31} A_{32} A_{33} ... *stb.*

Megjegyzés:

Természetesen három-, négy-, öt- stb. dimenziós tömböket is használhatunk a legtöbb programozási nyelv esetén, de ezek ritkábban fordulnak elő.

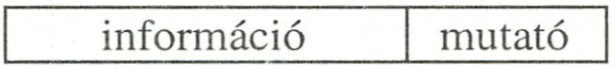
3.3 Lista

Mint láttuk, a tömbelemeket a memóriában sorban, egymás után elhelyezve tárolja a számítógép. Ezt a tárolási módot szokták *szekvenciálisnak* is nevezni. Ennek a tárolási módnak hátránya, hogy néhány műveletet nehéz elvégezni. Ilyen például egy új elem beszúrása, törlése. A hátrányok közé tartozik az is, hogy a tömbök memóriafoglalása fix nagyságú, a program futása közben nem lehet megváltoztatni.

A következőkben egy olyan adatszerkezetet ismerünk meg, amelyben az egyes elemek sorrendje szintén jól meghatározott, de az egyes elemek egyáltalán nem biztos, hogy a memóriában is ugyanilyen sorrendben helyezkednek el. A listával végezhető műveletek rendkívül rugalmasan végezhetők el, ami miatt bizonyos problémák megoldásánál szinte nélkülözhetetlenek.

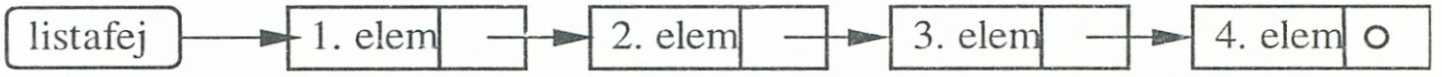
3.3.1 Egyirányú láncolt lista

A lista elemei a következő szerkezetűek:



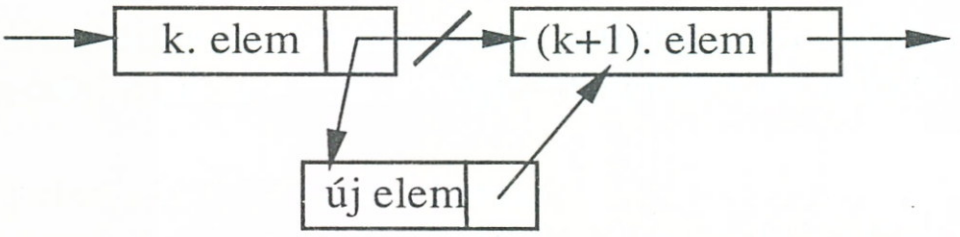
A mutató a következő elem címét tartalmazza. Az elemek összeláncolása ezeknek a mutatóknak a segítségével történik. Tehát a lista következő elemének nem szükséges az előzővel szomszédos memória helyet elfoglalnia. Így könnyebb lesz az új elemek beszúrása, törlése.

A lista tartalmaz két speciális elemet. Az egyik az ún. *listafej*, amely nem tartalmaz információt, hanem csak kijelöli a lista elejét. A másik egy *végelem*, amely tartalmaz információt, de a benne levő mutató speciális, nem mutat következő elemre.

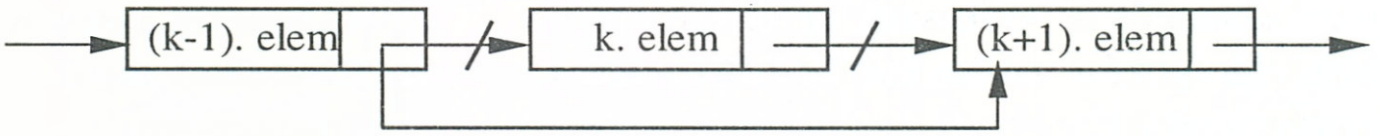


Két fontos művelet a következőképpen végezhető el:

Beszúrás: Egy elem beszúrásához elegendő az előző elem mutatójának a megváltoztatása.



Törlés: Egy elem törléséhez itt is az előző elem mutatóját kell megváltoztatni.

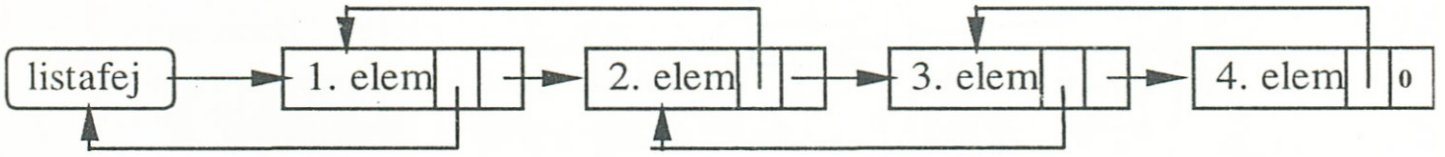


Megjegyzés:

Látható, hogy az elem törlése nem történik meg fizikailag, tehát az adott elem benn marad a memóriában, csak a listában nem fog szerepelni.
Problémát jelent ezeknek a törölt adatelemeknek a jelenléte a memóriában, hiszen csökkentik a szabad területet, így gondoskodni kell ennek a megoldásáról.

3.3.2 Kétirányú láncolt lista

Még egy résszel kiegészítjük az adatelemeket: az első rész az információt tartalmazza, a második az előző elemet mutatja, a harmadik pedig a következő elemre mutat.



Tehát a listák is lineáris adatszerkezetek, de a memóriabeli ábrázolásuk nem szekvenciális.

3.4 Verem (LIFO)

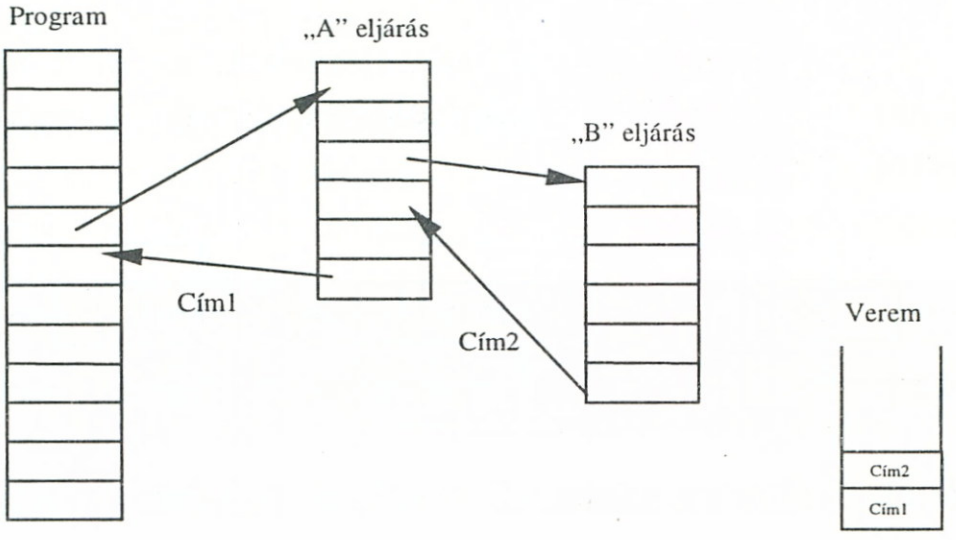
A verem egy olyan lineáris adatszerkezet, amelynek mindig csak az utolsó elemével lehet műveletet végezni. Ezek a műveletek a következők lehetnek:

1. Új adat beírása utolsó elemnek.
2. Az utolsó elem kiolvasása, egyúttal ez az elem törlődik.

Szemléletesen azt szokták mondani, hogy az utolsó elem a verem „tetejére" kerül, és ha kivesszük az utolsó elemet is, akkor „kiürül" a verem. Ezt a feldolgozási módot hívják **LIFO** -nak (***Last-In-First-Out : utoljára be- először ki***).

A verem legfontosabb alkalmazási területe az *eljáráshívás:*

A programban van egy eljáráshívás, amely az „*A*" eljárást hívja, a veremben tároljuk a *cím1*-et, amely megadja, hogy hol kell majd folytatni a programot az eljáráshívás után. Az „*A*" eljárásban van egy eljáráshívás, amely hívja a „*B*" eljárást. Most a *cím2*-t kell tárolni, hogy a „*B*" eljárásból visszatérve lehessen folytatni az „*A*"-t. Látható, hogy először a *cím2*-re lesz szükség, majd a *cím1*-re, márpedig a veremből ezek a címek a megfelelő sorrendben vehetők ki.



3.5 Sor (FIFO)

Olyan lineáris adatszerkezet, amelynek mindig a legelső elemével lehet műveletet végezni. Ezek a műveletek a következők lehetnek:

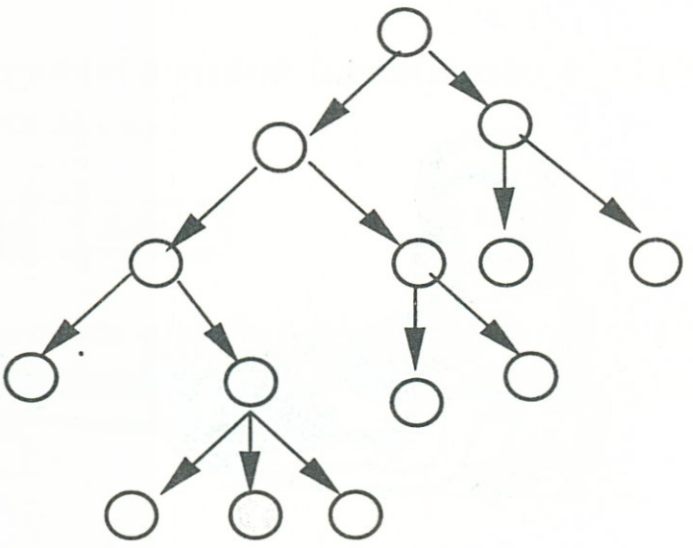
1. Új elem beírása utolsó elemnek.
2. A legelső elem kiolvasása, egyúttal ez az elem törlődik.

Mondhatjuk úgy, hogy az adatoknak a feldolgozáshoz sorba kell állniuk. Ezt a feldolgozási módot **FIFO**-nak nevezik (*First-In-First-Out: először be - először ki*)

A sor tipikus alkalmazási területe a *nyomtatási sor*. A közel egyidőben érkező nyomtatási kérelmeket sorba kell állítani, mert különben a nyomtatási munkák összekeverednének. Így a nyomtatási kérelmek abban a sorrendben lesznek kielégítve, amilyen sorrendben azok beérkeztek.

3.6 Bináris fa

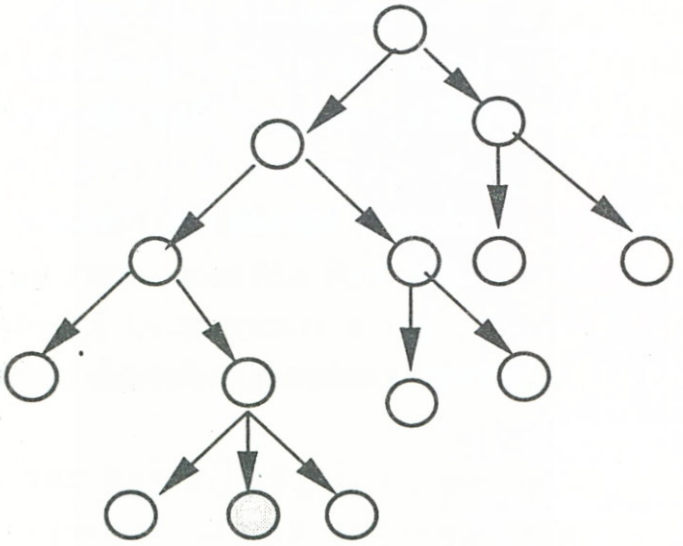
A fagráf a matematikából is ismert fogalom: egy olyan összefüggő gráf, amelyben nincsen kör. Hasonlóképpen értelmezzük mi is a fa adatszerkezetet, de itt az egyes szögpontokat összekötő élek irányítottak lesznek. Fát szerkeszthetünk a következőképpen: kiválasztunk egy szögpontot, ez lesz a fa *gyökéreleme*. Ezután a gyökérelemtől éleket húzunk a szomszédos szögpontokhoz, ezekből újra a velük szomszédos szögpontokhoz, stb.



Az adatszerkezetben az egyes szögpontoknak az adatelemek felelnek meg. Az élek pedig az adatelemek egymás utáni sorrendjét határozzák meg. Azokat az elemeket amelyekből nem indul él újabb elemhez, szokták *levélelemnek* is nevezni. Elég a gyökérelem címét ismerni, ennek a segítségével megkaphatjuk a többi elem címét.

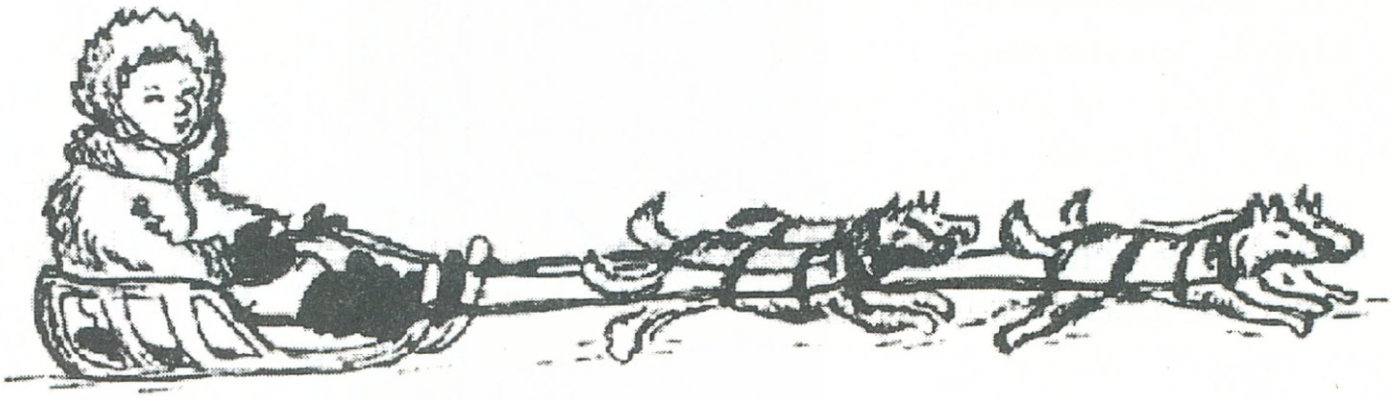
A fa adatszerkezetek között fontos szerepet játszanak azok, amelyeknél egy elágazási elemből *legfeljebb két* elem indul ki. Ezeket *bináris fának* nevezzük.

Az előző példát módosítva egy bináris fát kapunk:

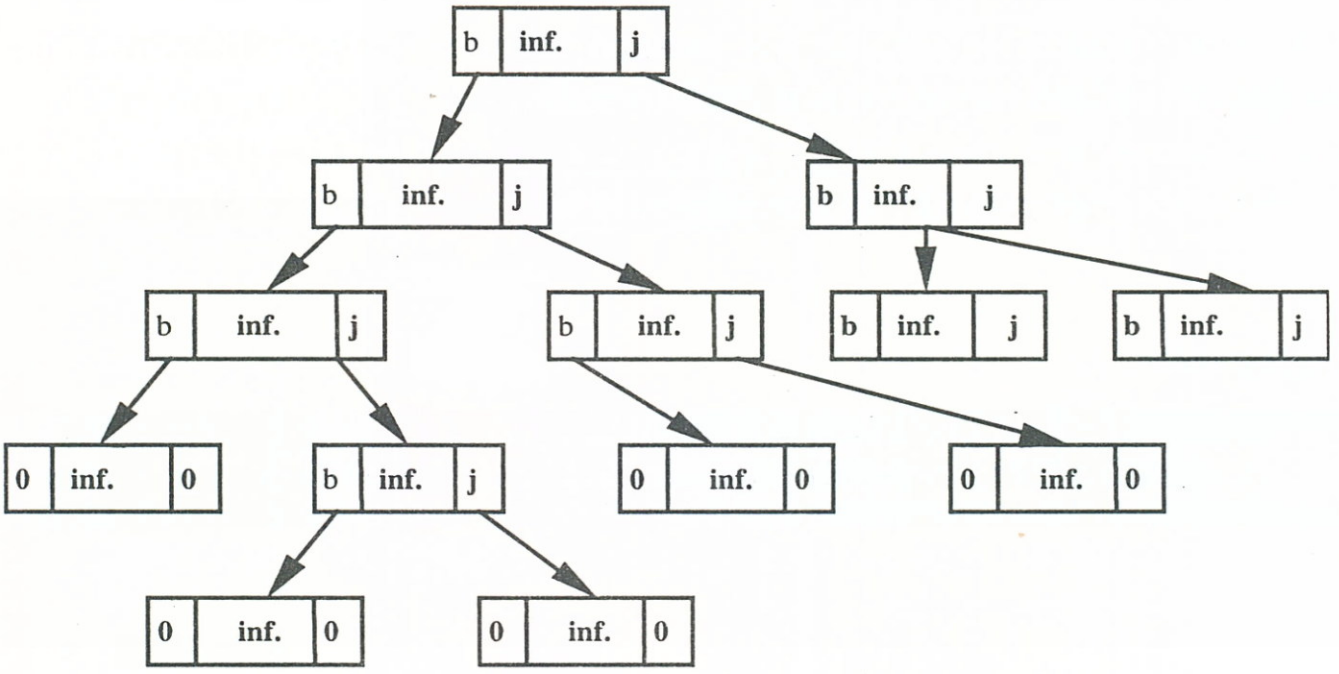


Bináris fák esetén az adatelem három részből áll:

bal mutató	információ	jobb mutató



A mutatók az elemből elágazó, megfelelő (bal, jobb) adatelemekre mutatnak. Az előző példát véve:



A fákat a memóriában listákkal szokták ábrázolni (tárolni), de ehhez a szögpontokat egy egyértelmű sorszámmal kell ellátni, hogy a listába be tudjuk őket illeszteni. Ezt nevezik a fa *bejárásának.*

3.7 Táblázatok

A táblázat tulajdonképpen egy függvénykapcsolat összetartozó értékek között. Az adatelem a következőképpen néz ki:

Argumentum	érték

Az argumentum és az érték is állhat több mezőből. A táblaelemeket általában az argumentumuk alapján szokták keresni, nevezhetjük ezt tartalom szerinti keresésnek is.

Például: A mikrogépek BASIC értelmezői (Commodore 64, ZX Spectrum) a beírt BASIC programok utasításait kódoltan tárolják a memóriában. Ezeket a kódokat nevezik **token**eknek.

Az utasításokat és a hozzájuk tartozó tokeneket egy táblázatban tárolják. A tárolt adatelemek a következő formájúak:

utasítás	token

Megjegyzés:

A táblázatokat általában láncolt listákkal tárolják a memóriában.

4. PROGRAMOZÁSI TÉTELEK

Problémamegoldás során sokszor találkozunk bizonyos elemi algoritmusokkal, amelyeket programozási tételeknek is szoktak nevezni. A tételekben szereplő elemsorozatok elemeit legegyszerűbben egy tömbben tárolhatjuk.

4.1 Elemsorozathoz egy elemet rendelünk

- **Eldöntés**
  Adott egy N elemű A tömb, és egy T tulajdonság.
  Kérdés: Van-e a sorozatnak T tulajdonságú eleme?

```
Eldöntés
    i:=1
    Ciklus amíg i<=N ÉS A(i) NEM T tulajdonságú
            i:= i+1
    Ciklus vége
    Van:=i<=N
Eljárás vége
```

- **Kiválasztás**
  Adott egy N elemű sorozat (A). Tudjuk, hogy a sorozatban legalább egy T tulajdonságú elem van.
  Kérdés: Melyik ez az elem?

```
Kiválasztás
    i:=1
    Ciklus amíg A(i) NEM T tulajdonságú
            i:=i+1
    Ciklus vége
    Sorszám:=i
Eljárás vége
```

- **Lineáris keresés**

Adott egy N elemű sorozat, és egy rajta értelmezett T tulajdonság.
Kérdés: Van –e egy T tulajdonságú elem a sorozatban, és ha van, akkor mi a sorszáma?

```
Lineáris
    i:=1
    Ciklus amíg i<=n ÉS A(i) NEM T tulajdonságú
            i:=i+1
    Ciklus vége
    Van := i<=N
    Ha Van akkor sorszám:=i
Eljárás vége
```

- **Logaritmikus keresés**

Adott egy N elemű *rendezett* sorozat, és egy keresett elem (*elem*).
Kérdés: Szerepel-e a keresett elem a sorozatban, és ha igen, akkor mi a sorszáma?

```
Logaritmikus
    alsó :=1 : felső:= N
    Ciklus
            középső := int((alsó+felső)/2)
            Ha A(középső)<elem akkor alsó:=középső+1
            Ha A(középső)>elem akkor felső:=középső-1
    Amíg alsó<=felső ÉS A(középső) <> elem
    Ciklus vége
    Van := A(középső)=elem
    Ha Van akkor sorszám := középső
Eljárás vége
```

A rendezettségből következik, hogy bármely elemről el lehet dönteni, hogy a keresett elem előtte van, utána van, vagy megtaláltuk. Először a középső elemet vizsgáljuk meg, hogy megegyezik-e a keresett elemmel. Ha igen, akkor kész vagyunk. Ha nem, akkor megnézzük, hogy a keresett elem a tömb alsó vagy felső részébe esik-e. (Rendezett tömb!) A továbbiakban azt a tömbrészt vizsgáljuk tovább, ahová a keresett elem esik, a tömb másik felé-

vel nem foglakozunk. Az adott résztömbben megkeressük a keresett elemet, és a továbbiakban ugyanúgy járunk el, mint előzőleg. Két esetben áll le a ciklus: megvan a keresett elem, vagy a vizsgált résztömbben már csak egyetlen elem maradt (és nem azt kerestük).
Az *alsó* és *felső* változók annak a részintervallumnak az alsó és felső végpontjai, amelyben a keresett elem benne lehet.
Ezt az algoritmust **logaritmikus** vagy **bináris keresés**nek is nevezik. Azért hívják logaritmikusnak, mert az elem megtalálásához szükséges lépések száma kb $log_2 N$.

- **Megszámolás**
  Adott egy N elemű sorozat (A), és egy rajta értelmezett T tulajdonság.
  Kérdés: Hány T tulajdonságú elem van A-ban?

```
Megszámolás
    s:=0
    Ciklus i=1 –től N-ig
            Ha A(i) T tulajdonságú akkor s:=s+1
    Ciklus vége
Eljárás vége
```

4.2 Sorozathoz sorozatot rendelünk

- **Kiválogatás**
  Adott egy N elemű sorozat (A). Keressük az összes T tulajdonságú elemet, és ezeket az elemeket helyezzük el egy B tömbben.

```
Kiválogatás
    j:=0
    Ciklus i=1-től N-ig
            Ha A(i) T tulajdonságú akkor j:=j+1: B(j):=A(i)
    Ciklus vége
Eljárás vége
```

Ennek a típusnak a legismertebb típusa a rendezés.

4.3 Rendezések

Ezen algoritmusokban a sorozat elemeit permutáljuk, azaz a sorrendjüket változtatjuk meg. Az adott sorozat elemeit csökkenő vagy növekvő sorrendbe kell rendezni.

1. **Minimum kiválasztásos rendezés**
   Megkeressük a legkisebb elemet a tömbben (lineáris kereséssel), majd az első helyre tesszük. Ezután azt a résztömböt tekintjük, amely az első elem kivételével áll elő. Itt is megkeressük a legkisebb elemet, és a tömb első helyére tesszük. Ezt az eljárást addig folytatjuk, amíg az (N-1). elemet is a helyére nem tesszük.

```
Minimum (N, A)
   Ciklus i=1-től N-1-ig
      hely:=i
      Ciklus j=i+1 –től N-ig
         Ha A(hely)>A(j) akkor hely:=j
      Ciklus vége
      csere := A(i)
      A(i):= A(hely)
      A(hely):=csere
   Ciklus vége
Eljárás vége
```

2. **Buborékos rendezés**
   Minden szomszédos elemet felcserélünk 1 és N között, ha fordított sorrendben helyezkednek el. Ekkor a legnagyobb elem, mint egy buborék „felszáll" a tömb „tetejére", azaz végére. Ezután azt a résztömböt vesszük, amely nem tartalmazza a már helyére került elemet. Ezt folytatjuk, amíg a 2. helyre is a megfelelő elem kerül.

```
Buborék(N,A)
    Ciklus i=N-től 2-ig –1-esével
        Ciklus j=1-től i-1 -ig
            Ha A(j)>A(j+1) akkor
                csere:=A(j-1)
                A(j-1):=A(j)
                A(j):=csere
            Elágazás vége
        Ciklus vége
    Ciklus vége
Eljárás vége
```

3. Beillesztéses rendezés

Az eddigi módszerekben az összes elemnek rendelkezésre kell állnia már kezdetben. A következő módszernél ezt nem tesszük fel. Abból indulunk ki, hogy egyetlen elem mindig rendezett. Ezután a következő elemet könnyű elhelyeznünk hiszen csak az elsővel kell összehasonlítanunk. Így kapunk egy újabb rendezett résztömböt. A következő elem helyét megkeressük a rendezett résztömbben, beillesztjük úgy, hogy a nála nagyobbakat egy hellyel elcsúsztatjuk.

```
Beillesztés(N,A)
    Ciklus i=2 –től N –ig
        j:=i-1
        Ciklus amíg j>0 ÉS A(j)>A(j+1)
            csere:=A(j)
            A(j):=A(j+1)
            A(j+1):=csere
            j:=j-1
        Ciklus vége
    Ciklus vége
Eljárás vége
```

4. Gyorsrendezés

A sorozatot két részre osztjuk (*középső_elem*). Jobbról és balról addig lépegetünk a tömbben, amíg kisebb illetve nagyobb elemet találunk a középső elemhez viszonyítva. Ahol „megállunk”, ott felcseréljük a két elemet (A(i), A(j)) és folytatjuk a lépegetést,

amíg a középső elemhez nem érünk. Így a tömböt kétfelé „vágtuk", a középső elem előtt csupa kisebb elem van, utána pedig csupa nagyobb. Ezután megismételjük az előbbieket a két résztömbre vonatkozóan. Ezt ún. **rekurzív hívás**sal hajtjuk végre (*Válogat(bal, j, A) ill Válogat(jobb, j, A)*, azaz az eljárást újra meghívjuk, az eljáráson belülről. Addig ismételjük, amíg a részsorozatok hossza „el nem fogy".

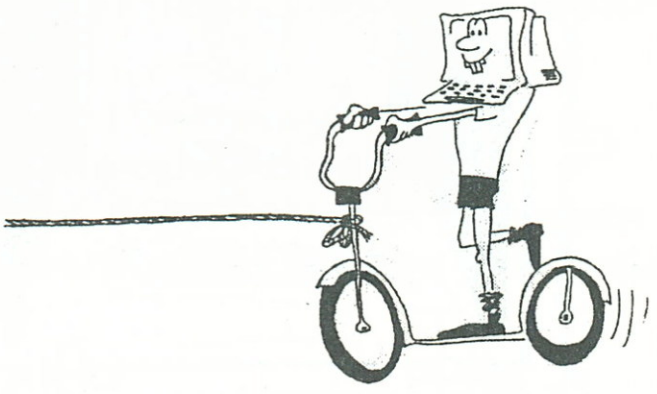
```
Válogat(alsó, felső, A)
   i:=bal
   j:=jobb
   középső_elem:= A(int(bal+jobb) /2))
   Ciklus
      Ciklus amíg A(i) < középső_elem
         i:=i+1
      Ciklus vége
      Ciklus amíg középső_elem < A(j)
         j:=j-1
      Ciklus vége
      Ha i<=j akkor
         csere:=A(i)
         A(i):=A(j)
         A(j):=csere
         i:=i+1
         j:=j+1
      Elágazás vége
   Amíg i<=j
   Ciklus vége
   Ha bal<j akkor Válogat(bal, j, A)
   Ha i<jobb akkor Válogat(i,jobb,A)
Eljárás vége

Program
   Válogat(1,N,A)
Program vége
```

4.4 Több sorozathoz rendelünk sorozatot

- **Metszetképzés**
  Adott egy N és egy M elemű sorozat az A illetve B tömbben tárolva. Képezzük a két sorozat metszetét egy C tömbbe. Csak azok az elemek kerülnek ide, amelyek mindkettőben szerepelnek.

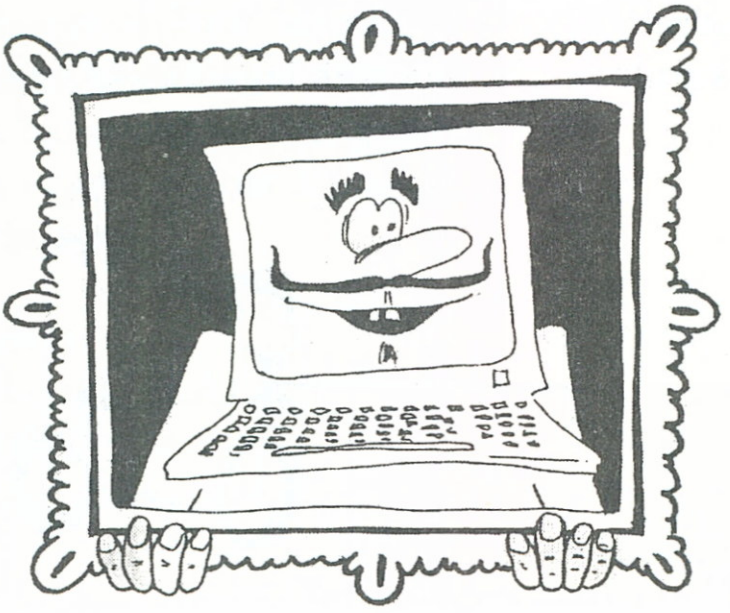
```
Metszet(N,M,A,B,C)
   k:=0
   Ciklus i=1-től N-ig
      j:=1
      Ciklus amíg j<=M ÉS A(i)<>B(j)
         j:=j+1
      Ciklus vége
      Ha j<=M akkor
         k:=k+1
         C(k):=A(i)
      Elágazás vége
   Ciklus vége
Eljárás vége
```

- **Unióképzés**
  Adott egy N és egy M elemű sorozat az A illetve B tömbben tárolva. Képezzük a két sorozat unióját egy C tömbbe. Azok az elemek kerülnek ide, amelyek legalább az egyikben szerepelnek.



```
UNIÓ(N,M,A,B,C)
    Ciklus i=1 –től N-ig
        C(i):=A(i)
    Ciklus vége
    k:=N
    Ciklus j=1-től M-ig
        I:=1
        Ciklus amíg i<=N ÉS  A(i) <>B(j)
            i:=i+1
        Ciklus vége
        Ha i>N akkor
            k:=k+1
            C(k):=B(j)
        Elágazás vége
    Ciklus vége
Eljárás vége
```

- **Összefuttatás**
  Adott egy N és egy M elemű *rendezett* sorozat az A illetve B tömbben tárolva. A C tömbbe azok az elemek kerülnek, amelyek az A-nak vagy B-nek az elemei. A C tömbnek is rendezettnek kell lenni. Tulajdonképpen ez is unióképzés, úgy, hogy a rendezettségnek is meg kell maradni.



```
Összefuttat(N,M,A,B,C)
        i:=1
        j:=1
        k:=0
        Ciklus amíg i<=N ÉS j<=M
            k:=k+1
            Elágazás
                A(i)<B(j) esetén C(k):=A(i) : i:=i+1
                A(i)=B(j) esetén C(k):=A(i) :i:=i+1: j:=j+1
                A(i)>B(j) esetén C(k) :=B(j): j:=j+1
            Elágazás vége
        Ciklus vége
        Ciklus amíg i<=N
            k:=k+1
            C(k):=A(i): i:=i+1
        Ciklus vége
        Ciklus amíg j<=M
            k:=k+1
            C(k):=B(j): j:=j+1
        Ciklus vége
    Eljárás vége
```

4.5 A PROGRAMÍRÁS LÉPÉSEI

A programok írásának általában a következő lépéseit szokták megkülönböztetni:

1. **A feladat pontos megfogalmazása**
   Nagyon lényeges, hogy a program megfogalmazása pontos, szabatos legyen, mert így sok későbbi kellemetlenséget előzhetünk meg.

2. **Az algoritmus megtervezése, elkészítése**
   Nagyon sokféle elv, elmélet létezik az algoritmusok készítésére. Néhányat majd mi is megismerünk. A legelső lépés annak eldöntése, hogy milyen bemenő és kimenő adatokat használjon az algoritmus.

3. **Kódolás**
   Itt történik az algoritmus „átírása" programmá.

4. **Tesztelés**
   A cél az, hogy minél több hibát felderítsünk az elkészült programunkban. A legtöbb esetben gyakorlatilag megoldhatatlan, hogy a programunkat minden bemenő adatra megvizsgáljuk. A legelső teszt a program szövegének teljes átvizsgálását jelenti, vajon tényleg azt írtuk-e le, amit akartunk. A következő lépés, hogy a programunkat *működés közben* vizsgáljuk. Különböző bemenő adatokat véve megnézzük, hogy helyes kimenő adatokat szolgáltat-e. A tesztadatok megválasztásánál törekedni kell arra, hogy minden lényeges esetet vizsgáljunk. Például: Ha egy számról el kell dönteni, hogy prímszám-e, akkor kipróbáljuk a programot prímszámra, nem prímszámra is, és például még 1-re és 2-re is. Lehetőleg olyan bemenő adatot is meg kell vizsgálni, amely a feladat szempontjából határesetnek minősül. Például: Ha a programunk az R=U/I képlet alapján számolja az R-et, akkor ki kell próbálnunk a programot I=0 esetére is. Vizsgálni kell azt is, hogy a program az előforduló hibalehetőségeket hogyan kezeli. Például: Rossz típusú adat esetén mit tesz a program?

5. **Hibakeresés, javítás**
   Az első lépés most is a program szövegének az átvizsgálása. Amíg a hibát meg nem találtuk, addig ne változtassunk a programon. Nagyon vigyázni kell a hiba javításánál, mert a tapasztalat szerint az esetek többségében új hibák keletkeznek. Általában a hibakeresés a teszteléssel kapcsolódik össze: ez a megfelelően megválasztott tesztadatok segítségével történik. A korszerűbb programfejlesztő-rendszerekben (ilyen például a Turbo Pascal is) különböző hibakereső eszközök is rendelkezésre állnak:

- Az egyes változókat *nyomon követhetjük* a futás során, láthatjuk az éppen aktuális értéküket.
- Lépésenkénti végrehajtás. A programunkat a program szövege alapján utasításonként végrehajthatjuk.

Töréspontok elhelyezése. A program futása a programban elhelyezett jelig, a töréspontig tart, majd innen folytathatjuk a végrehajtást, például lépésenként.

6. A hatékonyság vizsgálata

Az elkészült programot vizsgálni kell a futási idő és a tárfelhasználás szempontjából, nyilván ezeknek a minimalizálására kell törekedni.

7. Dokumentáció

A kész programban is érdemes magyarázó megjegyzéseket elhelyezni, hiszen így a későbbiekben könnyebb lesz tovább fejleszteni vagy csak megérteni is a saját programunkat. Nagyobb program esetén érdemes külön dokumentációkat is készíteni.

- Felhasználói dokumentáció:
  - A futtatáshoz szükséges géptípus, konfiguráció.
  - A program telepítése, installációja.
  - A program használatának részletes leírása.
  - A program hibajelzéseinek leírása.
  - Az egyes hibák kezelésének leírása.

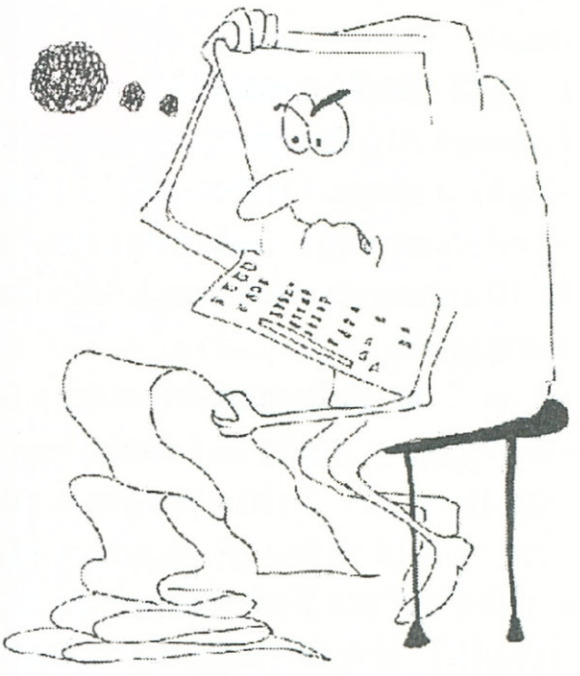
- Fejlesztői támogatás:
  - Az algoritmus részletes leírása.
  - A programban szereplő adatok leírása.
  - A program fejlesztési lehetőségei.A program teljes listája.

5. PROGRAMOZÁSI NYELVEK

Az első időkben közvetlenül programozták a számítógépeket, ún. *gépi kódban*. A gépi kód 0-k és 1-esek sorozata, amelyet a processzor közvetlenül „megért". Ez meglehetősen nehézkes módja volt a programozásnak. Ezért alakultak ki a programozási nyelvek.

Assembly:

Az assembly nyelv: a gépi kódhoz legközelebb álló alacsony szintű programozási nyelv. Ebben az utasítások már nem számsorozatként jelennek meg, hanem az utasításoknak megfelelő, a funkciójukra „emlékeztető", rövid szavakként. Az assembly nyelvű programozás is eléggé kényelmetlen, hiszen nagymértékben alkalmazkodni kell az adott hardver sajátosságaihoz. Assembly-ben gyakorlatilag minden programozási feladat megoldható.



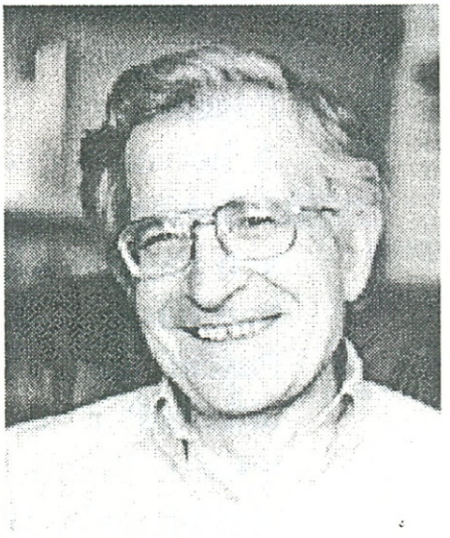
Ma már a programozási munka nagy részét magasszintű programozási nyelveken végzik. (Még az operációs rendszereket is ily módon írják.)

Megjegyzés:

A magasszintű nyelvek fordítása után az eredményül kapott gépi kód általában kevésbé hatékony, és hosszabb, mint az assembly nyelvű program fordítása során nyert gépi kód.

5.1 Magasszintű programozási nyelvek

A magasszintű programozási nyelveket az assembly nyelv kényelmetlenségei, nehézkessége miatt fejlesztették ki. Míg az assembly nyelv erősen kötődik a hardverhez, addig a magasszintű nyelvek viszonylag gépfüggetlenek. A magasszintű nyelvekre jellemző, hogy sok utasítással rendelkeznek, és ezek többsége összetett feladatokat képes megvalósítani. Az utasítások általában angol szavak, vagy azok rövidítéseiként adhatók meg. Például a Pascalban: Write, Read, SetColor, stb.



A mesterséges nyelvekkel foglalkozott **Noam Chomsky,** akit a matematikai nyelvészet atyjának is neveznek. A nyelveket négy nyelvosztályba sorolta 1959-ben. Az ún. *környezet független* nyelvtanú nyelvek közelítik meg legjobban a magasszintű programozási nyelveket.

Néhány példa a magasszintű programozási nyelvekre:

- **Fortran**
  Az ötvenes évek közepén jelent meg. Tudományos - műszaki nyelv, az IBM hozta létre, a saját számítógépeire. Nagyon egyszerű logikájú nyelv, matematikai számításokra kitűnően alkalmas, nagyobb gépeken még ma is használják.

- **Cobol**
  Az ötvenes évek második felében jelent meg. (Az Amerikai Védelmi Hivatal megrendelésére jött létre.) Adatfeldolgozási területeken, nagyszámítógépes rendszerekben még ma is nagyon elterjedt.

- **Basic**
  Kemény János magyar származású amerikai tudós „találta" ki a BASIC-et, a hatvanas évek elején, a FORTRAN nyelv egyszerűsítéseként. A közben létrehozott rengeteg nyelv között sokáig csak egy volt a sok között, egészen addig, amíg a hetvenes években meg nem jelentek

a mikroszámítógépek. A BASIC ugyanis könnyen tanulható, egyszerű nyelv, így a mikroszámítógépek a legszélesebb társadalmi rétegek számára elérhetők lettek. Ma már nagymértékben veszített a jelentőségéből.

- **Logo**
  A grafikus alapú nyelvet *Bolt, Beranek* és *Newman* készíti el. Később bekapcsolódik a fejlesztésbe és népszerűsítésbe Seymour Papert is. Ma is sok változata létezik, legismertebb Magyarországon a *Comenius Logo*.

- **Pascal**
  A hetvenes évek elején *Nicklaus Wirth* „találta ki". Nagyon következetes, szigorú nyelv. A PC-ken nagyon népszerű, de elsősorban az oktatásban használják. Érdekesség, hogy létrehozója a Pascal megalkotása után két évvel már nem foglalkozott a nyelvvel, sőt „ifjúkori tévedésnek" tekinti. Wirth később más nyelveket is fejlesztett: Modula, Oberon.

  A Turbo Pascal elődjét *Anders Heilsberg* dán programozó írta, és Európában Kompass Pascal néven terjesztette. Ennek a programnak a továbbfejlesztésére és terjesztésére hozták létre az USA-ban a *Borland International* nevű céget, amely 1983-ban megjelentette a Turbo Pascal 1.0 -ás verzióját. A program, és maga a nyelv is azóta sok változáson ment keresztül, legújabb verziója a 7.0-ás.

- **C**
  1972-ben fejlesztette ki *Dennis Ritchie*. A C nyelv kialakulása, fejlődése erősen kötődik a UNIX-hoz, mert ezt az operációs rendszert az első assembly-ben írt változat után C-ben írták. Így létrehozták az első olyan operációs rendszert, amelyet nem assembly-ben írtak. A C nyelv a lehetőségek szerint gépfüggetlen, azaz az egyik számítógépen megírt C program egy másik számítógépen is lefordítható és futtatható.

  1980-ban *Bjarne Stroustrup* megalkotja a C++ nyelvet. Ma talán a legnépszerűbb programozási nyelv. (Létezik egyébként C-- nyelv is...)

- **Java**
  A *Sun* cég hozta létre a Java programozási nyelvet. Elsősorban a World Wide Web–re írt alkalmazások fejlesztésére találták ki. Erősen épül a C/C++ nyelvekre.

- **Prolog:**
  A hetvenes évek elején Franciaországban hozzák létre. Az addigi nyelvektől teljesen eltérő logikával épül fel, a mesterséges intelligencia kutatását segítő nyelvként jött létre.

A magasszintű programozási nyelveket a következőképpen is jellemezhetjük:

- **Algoritmikus nyelvek:**
  A programozó határozza meg azt, hogy mit és hogyan kell a gépnek elvégezni. Azaz megadja az algoritmust, a probléma megoldásához szükséges lépéssorozatot. Ilyen nyelv például a *Pascal*, a *C*, a *BASIC*, stb.

- **Deklaratív nyelvek:**
  Ezek a nem algoritmikus nyelvek. A feladat megoldásához azt közlöm a géppel, hogy mit akarok megoldani, és nem azt, hogy hogyan. A programnyelv dolga, hogy a felvetett problémára a választ megkeresse.
  Tipikus képviselője a deklaratív nyelveknek a *Prolog*.

6. FORDÍTÓK ÉS ÉRTELMEZŐK

6.1 Fordítók

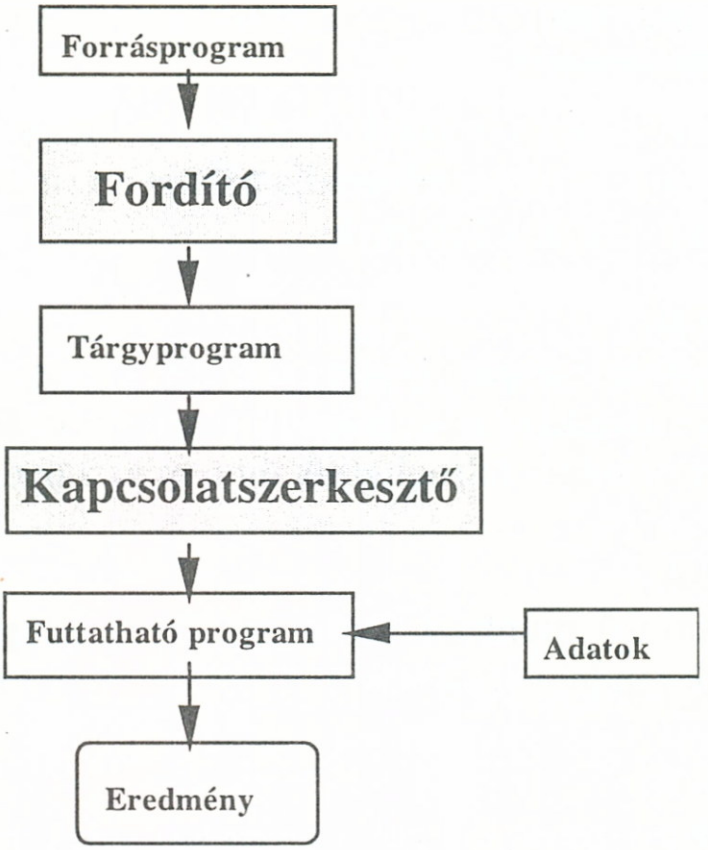
Az operációs rendszer, mint láttuk, kapcsolatot teremt a hardver és a felhasználó között. Ahhoz viszont, hogy a számítógépet a céljainknak megfelelően használjuk, programokat kell írnunk. Ehhez egy *ASCII karaktereket használó szövegszerkesztőt* használhatunk általában. A program írása valamilyen programozási nyelven történik. A megírt programszöveget *forrásprogramnak* nevezzük. Ahhoz, hogy a programot „megértessük", a forrásprogramot le kell fordítani a gép által már értelmezhető formájúra. A fordítóprogram: a forrásprogramot ún. *tárgyprogrammá* vagy *tárgykóddá* alakítja át, amely már *gépi kódú* program, tehát a gép által közvetlenül megérthető formájú, de ez általában még nem futtatható közvetlenül, szükség van egyéb programrészekkel való szerkesztésre, illetve el kell látni a futtatáshoz szükséges információkkal.

Ha a forrásprogram assembly nyelvű, akkor a fordítóprogramot *assembler*nek nevezzük, egyébként *compiler* a neve.

Megjegyzés:

Vannak olyan fordítóprogramok, amelyek assembly nyelvű programformát hoznak létre, és ezt még egy assembler programmal le kell fordítani.

A forrásprogramot először lefordítjuk a fordítóprogrammal, amelynek az eredményeképpen előáll a tárgykód, ennek szerkesztése után a megfelelő adatok felhasználásával megkapjuk a kívánt eredményt.



A legtöbb fordító 2 menetben fordítja le a forrásprogramot. Az első menetben a forrásszöveg lexikai és szintaktikai elemzése történik. Ennek a menetnek az eredményeként előállít egy ún. *közbülső formát*. A második menetben a fordító a közbülső formából előállítja a tárgyprogramot.

6.1.1 Lexikális elemzés

A lexikális elemzés legfontosabb feladata az, hogy felderíti a program ún. *alapjeleit*, és szabványos formában átadja azokat a szintaktikai elemzőnek.

Az alapjelek a következők lehetnek:

1. **Alapszavak**
   Az alapszavak tulajdonképpen a nyelv szavai, amelyeket a programozás során használhatunk. Például: Begin, For, Procedure, stb. Az alapszavak összességét felfoghatjuk úgy, mint a nyelv szókészletét.

2. **Azonosítók**
   A programozó munkája során egy sereg azonosítót használ. Például azonosítót rendel változókhoz, általa definiált eljárásokhoz, stb.

3. **Konstansok**
   A legtöbb programban szükség van konstans értékekre, amelyek értéke a program egészében állandó. Pl: 3.14, stb.

4. **Elhatároló jelek**
   A programozási nyelveknek (a „valódi" nyelvekhez hasonlóan) a jelkészletét nemcsak a szavak alkotják, hanem egyéb jelek is. Például: szóköz, =, (,), /, +, stb.

A program felfogható úgy, mint az alapjelek sorozata. Az alapjelekhez tartozó adatokat *táblázatokban* tárolja.

Az ún. standard formában minden alapjelhez két bejegyzés tartozik. Az *első* azt adja meg, hogy milyen típusú az adott alapjel az 1. -4. lehetőségeknek megfelelően.

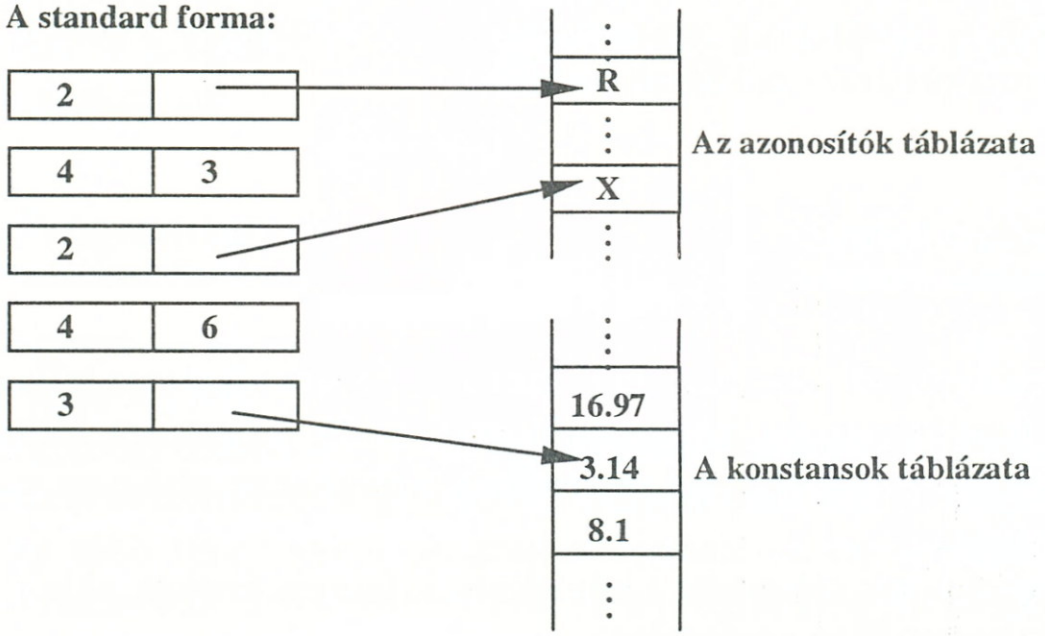
Alapszavak és *elhatároló jelek* esetén (amelyek a nyelv állandó részét képezik, a szabályokkal együtt) a *második* bejegyzés egy sorszám, amely megadja, hogy az alapjel a típusának megfelelő listában hányadik. (Tehát az adott típusú alapjeleket sorba állítják, és az ennek megfelelő sorszámmal lehet majd rá hivatkozni. Természetesen ez a sorszámozott lista tárolva van, táblázatos formában.)

Azonosítók és *konstansok* esetén a *második* bejegyzés egy *mutató*, amely azt jelzi, hogy az adott alapjel értéke a típusának megfelelő táblázatban hol van. (Tehát az azonosítóknak és a konstansoknak is van egy-egy táblázata, amelyben az egyes alapjelekhez tartozó értékek megtalálhatók. Azonosítók esetén ez az érték tulajdonképpen egy karaktersorozat.)

Legyen például egy utasítás a következő:

$$R = X + 3.14$$

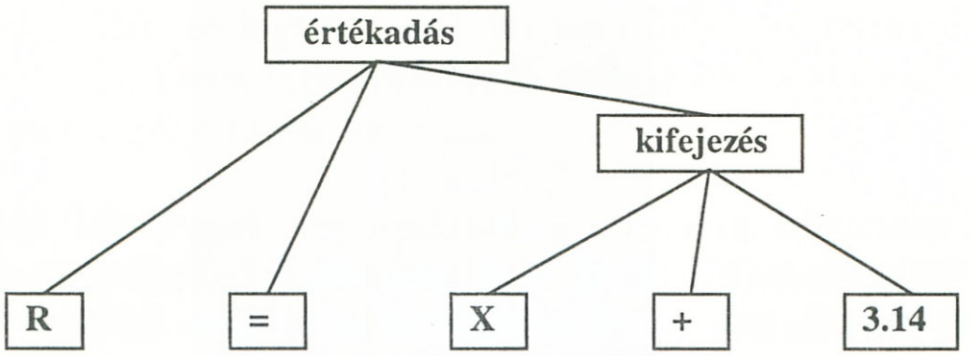
Itt az **R** és **X** azonosító, az = és a + elhatároló jel. Tegyük fel, hogy az = jel sorszáma 3, a + jel sorszáma 6. Az X és R azonosítók, méghozzá változóké. A 3.14 konstans. Az utasítás standard formája a lexikális elemzés után.



6.2 Szintaktikai elemzés

A szintaktikai elemzést végző rész a lexikális elemzőtől már standard formában kapja meg a programot. Itt a legfontosabb a program szintaktikai ellenőrzése, amely ahhoz hasonló, ahogyan egy adott nyelven megírt szöveget megvizsgálhatunk, hogy a nyelvtani szabályok szerint helyes-e. A szintaktikai elemzésre több eljárás is létezik. Az egyik ilyen módszer az ún. *szintaxis* fa.

Az előző példát véve ez a következőképpen néz ki:



A szintaktikai elemző elkészíti minden utasítás szintaxis fáját, majd ezek segítségével felépíti az egész program szintaxis fáját. A folyamat végén a fordítóprogram elkészíti a program közbülső formáját.

6.2.1 A közbülső forma

A lexikális elemzésnél látott standard forma még nem alkalmas teljesen arra, hogy abból a fordítóprogram közvetlenül gépi kódú programot állítson elő, ugyanis a műveleteket nem abban a sorrendben tartalmazza, ahogyan majd azokat végre kell hajtani.

Például, ha ki akarjuk számítani a következő kifejezés értékét, akkor abban a műveletek rossz sorrendben szerepelnek:

2 + 3 / 4

Ugyanis, balról jobbra haladva, először az összeadást kell elvégezni, és csak utána az osztást. Többféle megoldás létezik, az egyik elterjedt módszer a *fordított lengyel forma*, amelyet például a **Clipper** fordítóprogram esetén is alkalmaztak.

Tekintsük a következő kifejezést:

a * b

Ennek a fordított lengyel formája:

a b *

Az előző kifejezés fordított lengyel formája a következő:

2 3 4 / +

Tehát a közbülső forma abban a sorrendben tartalmazza az egyes utasításokat, amilyen sorrendben azokat majd a tárgynyelvű program is tartalmazza.

6.2.2 Kódgenerálás

A tulajdonképpeni fordítást jelenti, ennek a végeredménye a tárgynyelvű program. Ebben a fázisban sok esetben ún. *optimalizálás* is történik, amivel csökkenhet a programidő és tárigény. Kihagyja például a soha nem használt programágakat, stb.

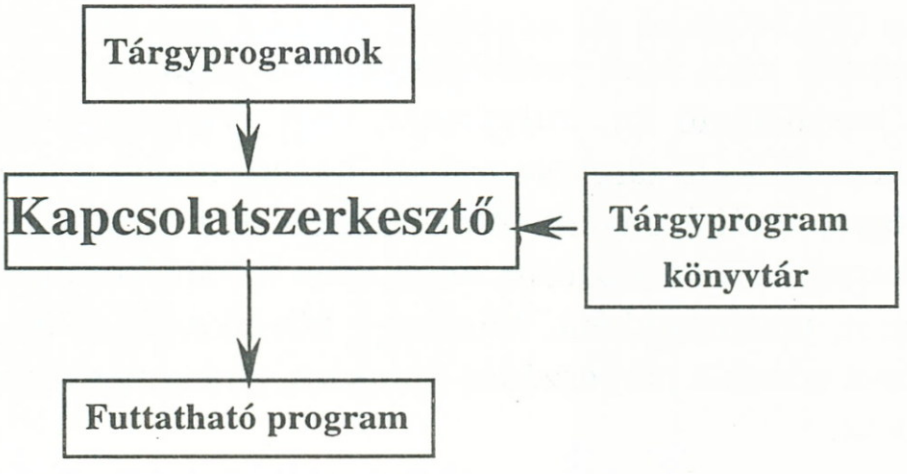
6.2.3 Programszerkesztés

A tárgynyelvű programot még szerkeszteni kell ahhoz, hogy végleges, futtatható formájú legyen.

A tárgyprogramban sok olyan *feloldatlan hivatkozás* van még, amely a megfelelő *programkönyvtár* segítségével oldható csak fel. A szerkesztés folyamán egy vagy több tárgynyelvű programot használ az ún. *kapcsolatszerkesztő*, és ezeket a futtatható forma létrehozásához a programhoz szerkeszti.

Például, ha a Turbo Pascal programunkban szerepel a *ClrScr* eljárás, akkor a szerkesztés során meg kell keresni a megfelelő programkönyvtárban, ez esetünkben a *Crt Unit* lesz, és hozzá kell szerkeszteni a programunkhoz.

A tárgykódú programot szerkeszteni kell akkor is, ha nincsenek benne feloldatlan hivatkozások, mert tárgykódú forma még nem tölthető be közvetlenül a memóriába.



6.2.4 A programfejlesztés lépései

Egy program megírása a következő lépésekből áll általában:

- A forrásprogram írása, illetve szerkesztése egy szövegszerkesztő segítségével.
- A forrásprogram lefordítása a fordítóprogram segítségével.
- A kapcsolatszerkesztő segítségével a tárgynyelvű program létrehozása.
- A program futtatása, hibakeresés.
- Ha szükséges, újra az első lépés következik, stb.

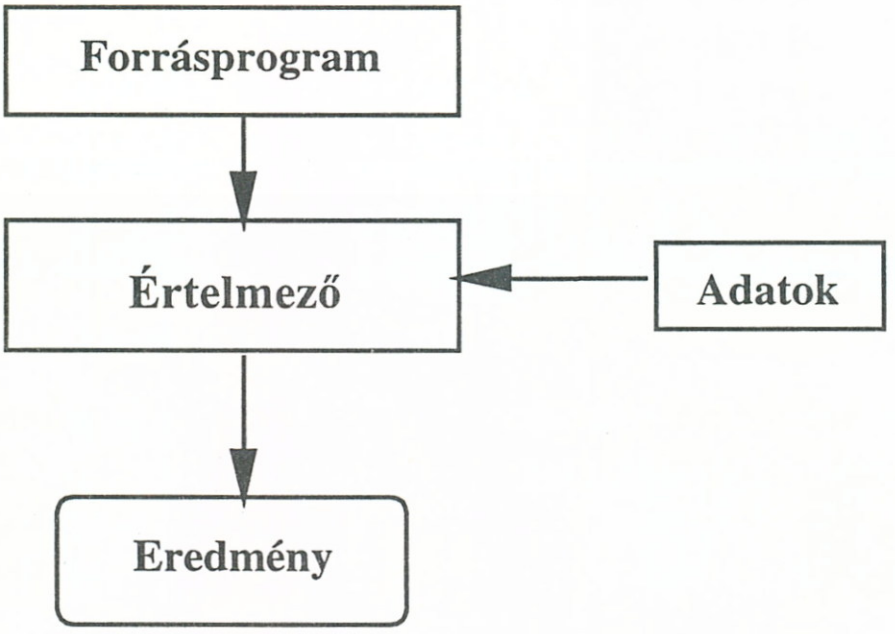
Megjegyzés:

1. *Manapság nagyon elterjedtek az ún. integrált környezetű fordítóprogramok.*
   Ebben az esetben egy szövegszerkesztőben dolgozhatunk végig, a programból való kilépés nélkül minden lépés elvégezhető. Ilyen fordítóprogram például a Turbo Pascal.
2. *Egy assembly nyelvű program esetén a szövegszerkesztést végezhetjük az EDIT-tel, a fordítást a TASM (Turbo Assembler) nevü programmal, a kapcsolatszerkesztést a TLINK (Turbo Linker) segítségével.*

6.3 Értelmezők

Programfejlesztés nem csak fordítóprogramok segítségével lehetséges. Néhány nyelvhez használható ún. *interpreter*, vagy *értelmezőprogram*. Az értelmezőprogram nem állít elő tárgyprogramot, hanem magát a forrásnyelvű programot hajtja végre, utasításonként. Az értelmezéskor nem képződik tárgykód, nincs szükség kapcsolatszerkesztésre, de minden futtatáskor újra kell értelmezni a forrásprogramot, utasításonként. Mindezek következtében az értelmezőprogramok lehetőséget adnak a párbeszédes munkára, amely nagyban megkönnyíti a programfejlesztést.

Egy forrásprogram értelmezésének a lépései:



Az értelmezők használatának több hátránya is van a fordítókkal szemben. Az egyik az, hogy az értelmezőnek mindig benn kell lenni a memóriában, hiszen az hajtja végre a programot, csökkenti a szabad memória területét. A másik hátránya a fordítóprogramokkal szemben abból adódik, hogy a programokat utasításonként értelmezi és hajtja végre, aminek következtében lassú lesz a végrehajtás.

BASIC értelmezővel vannak ellátva az ismert mikrogépek (Commodore 64, ZX Spectrum, stb). Manapság leginkább az Interneten használnak értelmezővel nyelveket: Perl, Java Script, stb. Különleges eset a Java: a forrásszöveget először le kell fordítani ún. bájtkódba. A bájtkód úgy képzelhető el, mint egy virtuális processzor gépi kódja. Ezt a bájtkódot végrehajtani egy értelmezővel lehet, ami a valódi CPU gépi kódjára fordítja le (és hajtja végre) az egyes utasításokat.

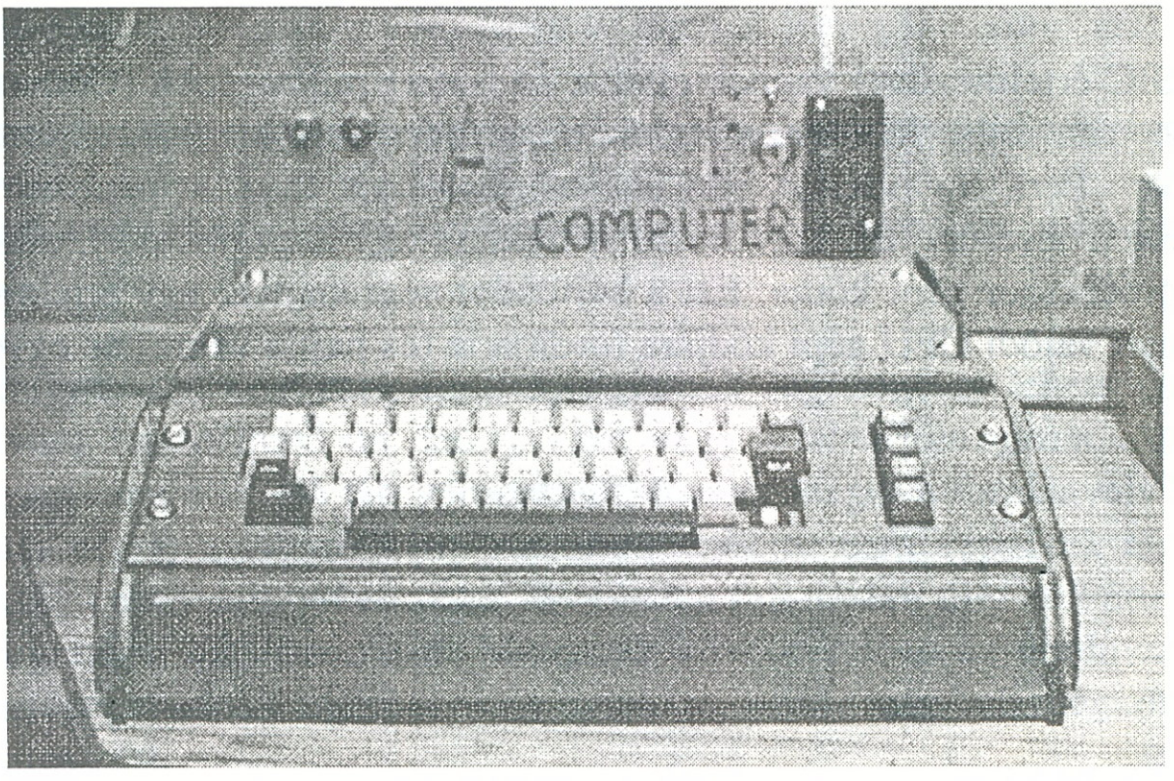
7. HARDVER ALAPISMERETEK

7.1 A Mikroprocesszor

A PC-kbe beépíthető mikroprocesszorok többfélék lehetnek, és tulajdonképpen ezek határozzák meg a számítógép típusát. A CPU vezérlőjelek, tárcímek, adatok küldésével, vételével irányítja a számítógépet.

A következőkben a legfontosabb jellemzőket vizsgáljuk meg.

- **Regiszter**
  A CPU-k ún. belső *regisztereket* tartalmaznak, amelyek a működéséhez szükséges belső tárolók. Ezek mérete határozza azt meg, hogy hány *„bitesnek"* nevezünk egy számítógépet. Az első mikroszámítógépek (házi számítógépek) 8 bitesek voltak. Ilyenek voltak a Apple, ABC80, Commodore 64, ZX81, ZX Spectrum, Enterprise, HT 1080Z, Videoton TVC, Primo, stb számítógépek.



A regiszterek általános célúak, ill. speciális feladatúak lehetnek.
A legtöbb processzornál megtalálhatók a következő regiszterek:

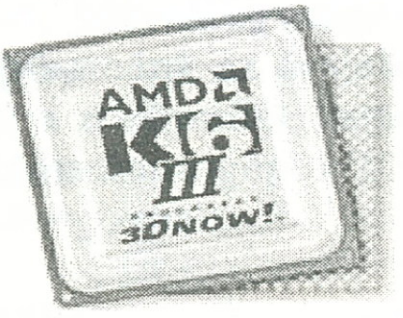
- **Utasításszámláló regiszter**
  A következő utasítás memóriabeli címét adja meg.

- **Báziscím regiszter, Index regiszter**
  A CPU által végrehajtott utasításban szereplő *operandusok* címzését segítik.

- **Állapot regiszter**
  Az egyes utasítások végrehajtása után bekövetkező állapotokat jellemzik. Pl.: az eredmény 0, *átvitel* keletkezett, *túlcsordulás* (túl nagy számérték, amely nem fér el az adott regiszterben) következett be, stb.

- **Veremmutató regiszter**
  A veremtároló tetejét adja meg. A verem általában a központi tárban (RAM) található, és a már tanult verem adatszerkezetről van szó.

- **Adatvezetékek, címvezetékek**
  Jellemző még a processzorra, hogy hány *adatvezeték*et tartalmaz, hiszen ez határozza meg az egyszerre küldhető adatmennyiséget. További jellemző, hogy hány *címvezeték*et tartalmaz, amely a címezhető legnagyobb tárterületet adja meg.



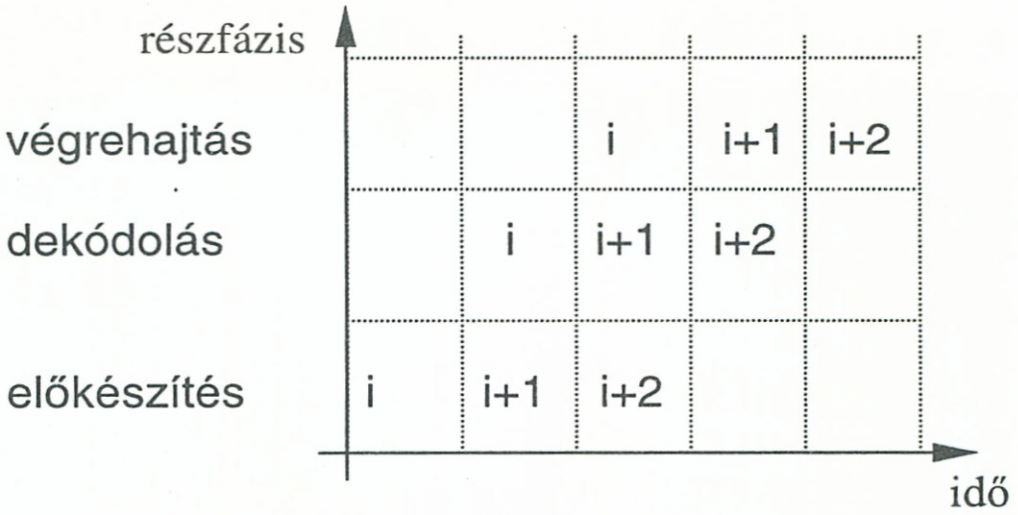
- **Az utasítások végrehajtásának lépései**
  A mikroprocesszor működésére jellemző az *utasításciklus*: utasításelérés, utasításdekódolás, utasításvégrehajtás.

➢ **Utasításelérés**

A mikroprocesszor az *elő-sorbaállítóból* elolvassa a következő utasítást. Az elő-sorbaállítóba az ún. **cache** memóriából (lásd később!) kerülnek az utasítások. Azért alkalmaznak sort (és nem egy utasítást), mert így gyorsabb lehet a végrehajtás, hiszen amíg a CPU végrehajtja az egyik utasítást, addig a következő utasítást már előkészíti, dekódolja.

Akkor hatástalan a sor alkalmazása, ha az utasítások között valamelyik egy ugróutasítás, ilyenkor törölni kell az utasítássort, és újra feltölteni. Arra is találtak megoldásokat, hogy előre megjósolják (nyilván nem százszázalékosan), hogy lesz-e ugrás, vagy sem.

A párhuzamosításra azért van lehetőség, mert az utasítások végrehajtása részfázisokra bontható. Egy szerelőszalaghoz hasonlíthatjuk ezt a megoldást. Ha egyetlen alkatrészt kell előállítanunk, akkor nincs szükség szalagra. Ha már két alkatrészt gyártunk, akkor már megéri szerelőszalagot alkalmazni. Ehhez persze két munkás kell, és egyik alkatrész sem fog gyorsabban elkészülni, de a termelés összességében mégis felgyorsul. Ezt a technikát nevezik **pipelining feldolgozásnak.**

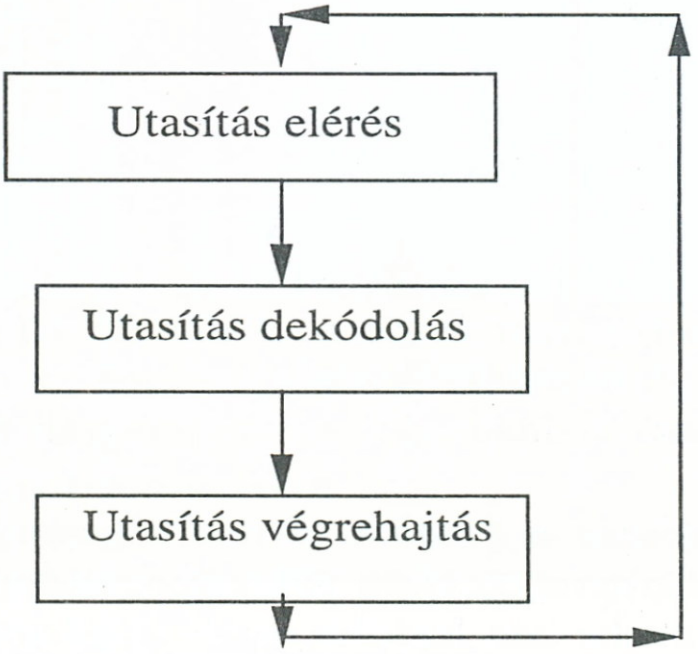


i: az i. utasítás

➢ **Utasításdekódolás**

Az utasítás értelmezése, az egyes operandusok címének meghatározása.

➢ **Utasításvégrehajtás**
Ebben a részben történik az utasítás tényleges végrehajtása. A *címegység* segítségével történik az adatok, illetve utasítások kiolvasása a memóriából és a memóriába történő írás.



- **Órajelfrekvencia**
Fontos a CPU-nál, hogy mekkora az ún. **órajelfrekvencia**, amely összefüggésben van a processzor által elérhető műveletvégzési sebességgel. A számítógép az órajel segítségével szinkronizálja az egyes részek működését. A nagyobb órajelfrekvencia nagyobb sebességű művelet végrehajtást is jelent általában. Mértékegysége a **MHz** (megahertz). (Egy metronómhoz hasonlíthatnánk az órajelet előállító *órajel-generátort*: minden műveletet a metronóm kattanásaihoz kell igazítani.) A mai (1999) személyi számítógépek 100 MHz és 800 MHz közötti órajelfrekvenciával dolgoznak.

- **Cache-memória**
A 386-os gépektől kezdődően ún. **cache-memóriát** is építenek az alaplapra. A cache egy viszonylag kis kapacitású (16 Kbájt – 2 MBájt) gyors RAM típusú memória (átlagos elérési idő 10-20 ns). A cache ún. szegmensekre, tartományokra van osztva. A szegmensek egy memóriaterület adatain, ill. utasításain kívül az első adat ill. utasítás címét is tárolják, és nyilvántartják a hozzáférések gyakoriságát vagy a legutolsó hozzáférés idejét. Ha a processzor egy memóriarekeszt címez, akkor először a cache memóriát vizs-

gálja meg> ha itt van a keresett adat, illetve utasítás, akkor innen olvassa ki. Ha itt nincs, csak akkor fordul a viszonylag lassú RAM-hoz. A legrégebben vagy legkevesebbet használt szegmensbe tölti be a keresett adat, ill. utasítás címétől kezdődő, egy szegmens méretének megfelelő RAM tartomány adatait. Így végül is a program végrehajtása gyorsul.
A 486-os processzortól kezdve a processzorok különböző méretű belső (a CPU chipen belüli, **első szintű –L1**) **cache**-t is tartalmaznak. Az alaplapon is találhatunk általában cache-t (**L2 szintű**). Egyes típusoknál az L2 cache is a CPU tokján belül található, az alaplapon már egy harmadszintű (L3) cache-t is elhelyeznek. Ilyen például a *Pentium II Xeon*, *AMD K6 III* processzor is. A *Pentium II,* ill. *Pentium III* processzoroknál az L2 cache a CPU-val egy dobozban található, de nem az integrált áramkör (IC) tokján belül.
L2 cache-t tartalmaz a *Pentium Pro*, *Celeron vagy az AMD K7* processzor. (A Celeron esetén a 300A típustól kezdve.)
Az első szintű cache-ben legtöbb esetben különválasztják az utasítás és adat cache-t.

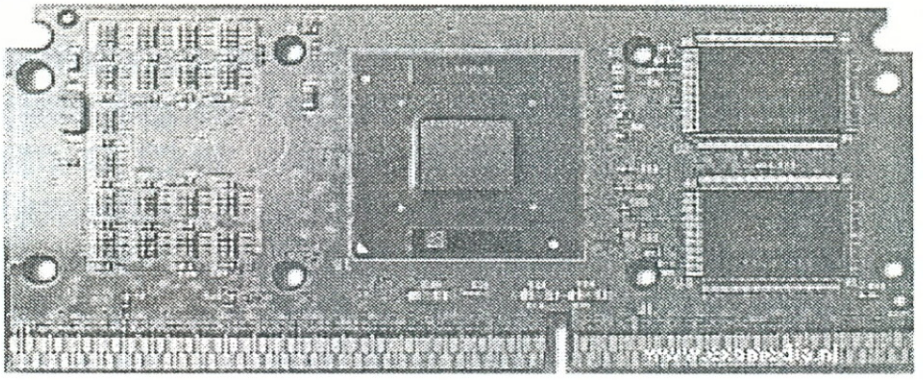
- **CISC, RISC**
A mai mikroprocesszorokkal kapcsolatban két fogalmat sűrűn lehet hallani: az egyik a **CISC**, a másik a **RISC.**

  - CISC: *komplex utasításkészletű mikroprocesszor.* Az egyes utasítások viszonylag összetettek, bonyolultak és különböző hosszúságúak. Az összetett utasításokat csak úgy tudja végrehajtani a CPU, hogy az egyes utasításokat ún. **mikro-programok**ra bontja le, és azt a **mikrovezérlő** végrehajtja. (Olyan, mintha egy számítógép működne a számítógépen belül.)

  - RISC: *csökkentett utasításkészletű mikroprocesszor.* Ezeknél a processzoroknál viszonylag kevés számú utasítás használható, amelyek azonos hosszúságúak, és viszonylag egyszerűek. Nincs mikroprogram.

A PC-s processzorok, (ma még) CISC felépítésűek. Az újabbakban (Pl.: Pentium II-III, K6, K6 -2, K6 III, K7) már egy ún. RISC

mag található. A RISC egyszerűbb felépítésű, de lényegesen gyorsabb mint a CISC, ezért egyre szélesebb körben alkalmazzák.



Intel Pentium III

- **Processzor üzemmódok**
  A továbbiak az Intel 80386-os processzortól kezdve érvényesek, bár a legtöbb jellemző a többi PC-s processzorra is igaz (AMD, Cyrix CPU-k).

  ➢ **Valós mód**
  A CPU ilyenkor úgy működik, mint egy gyors 8086-os. A gyorsabb működés két okra vezethető vissza:
  - A belső felépítés korszerűbb, hatékonyabb volta.
  - A nagyobb órajelfrekvencia.

  ➢ **Védett mód**
  Védett módban a processzornak a 8086-ostól eltérő tulajdonságai is vannak, de minden olyan program, amely a 8086-ost tartalmazó XT-n fut, az fut a Pentiumot tartalmazó AT-n is. Azt mondjuk, hogy a Pentium felülről kompatíbilis a 8086-ossal. (Azért felülről, mert annál többet is tud.) Védett módban olyan képességek kerülnek előtérbe, amelyek lehetővé teszik, hogy egy időben több program is fusson a gépen. Ezt a következő négy jellemző teszi lehetővé:

  1. Multitasking
  A processzor gyors átkapcsolási lehetőséget teremt a programok között a multitasking megvalósítására. Egy adott pillanatban a processzor csak egy program utasítá-

saival foglalkozik, de a gyors átkapcsolások miatt úgy tűnik, hogy minden program egyszerre fut.

2. **Védelem**
   A processzor által támogatott védelmi funkció biztosítja azokat a korlátokat, amelyek megakadályozzák, hogy egy program más programokkal, vagy az operációs rendszerrel összeütközésbe kerüljön. Egy rosszul megírt program a védelem mellett nem tud szabadon „garázdálkodni" a memóriában.

Megjegyzés:

Mivel az MSDOS operációs rendszerrel valós módban működik a CPU, az operációs rendszer kevés védelmi lehetőséggel rendelkezik.

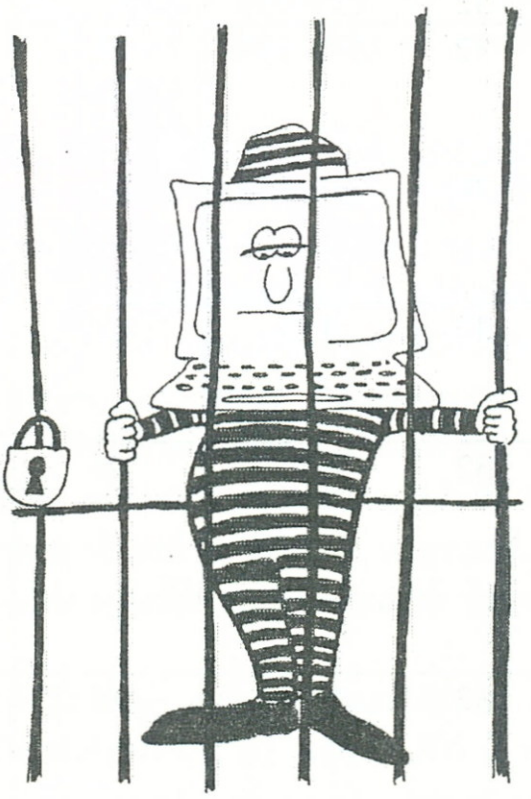
3. **Extended memória**
   A memóriának az 1 Mbájt fölötti részét nevezik extended memóriának.
   Valós módban a processzor a 8086-oshoz hasonlóan 1 Mbájt memóriát képes kezelni. Védett módban a CPU megnöveli az elérhető memóriatartományt.

4. **Virtuális memória**
   A virtuális tárkezelés lényege, hogy az operatív memóriát látszólagosan kiterjesztjük, megnagyobbítjuk a winchester segítségével. A winchester egy megadott területét címekkel látjuk el, mint a rendes memóriát, és így nagyobbnak látjuk az operatív tárat.

➢ **Virtuális 8086-os mód**
Olyan a működése, mint egy 8086-osnak, de a processzor védett módban működik. Bármilyen 8086-os program futtatható a processzor védett módú felügyelete alatt. Ez a 8086-os program lehet akár maga az MSDOS is. Így lehet például UNIX operációs rendszerben MSDOS–t emulálni, „utánozni". Egy látszólagos, virtuális gépet kapunk 640

Kbájtos memóriával, amelyet aztán valós módban gond nélkül használhatunk az MSDOS-ban. Ilyen módon több MSDOS alkalmazás is futhat egymás mellett biztonságosan.

- **Privilegizálási szintek**
  Négy privilegizálási szint van 0-tól 3-ig számozva. A legmagasabb szint a 0. Csak a legmegbízhatóbb programok futhatnak a 0-s szinten (jellemző módon ez az operációs rendszer legfontosabb része, a kernel). A magasabb szinten futó program hozzáférhet az alacsonyabb szinten levő adatokhoz, de fordítva ez nem lehetséges. A magasabb szinten futó program viszont nem hívhat alacsonyabb szinten levő programot (hiszen az valószínűleg kevésbé megbízható). Az operációs rendszerek általában nem használják ki mind a négy szintet.



A következő táblázat tartalmazza néhány ismertebb PC-s processzor jellemzőit.

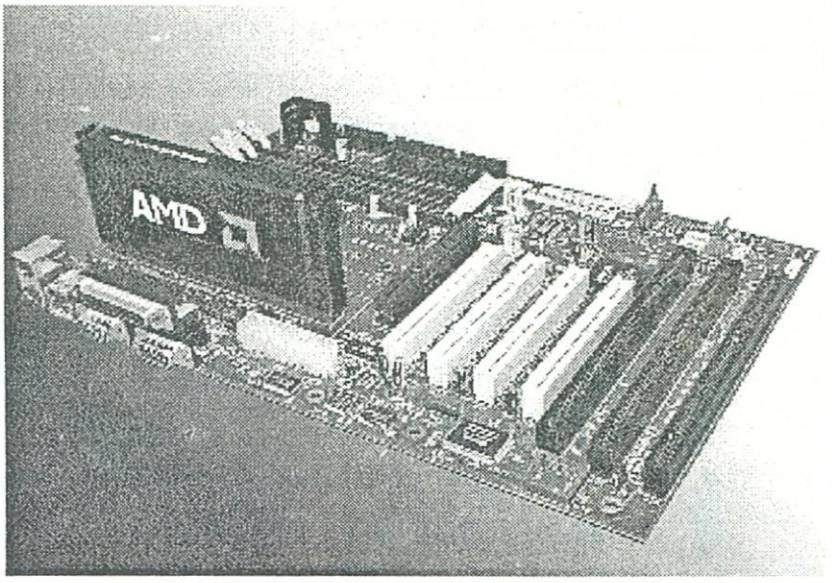
A CPU Neve	Mat. Ko-proc	Re-giszter (bit)	Órajel (MHz)	Adatve-zetékek száma	Belső cache (L1) (KB)	Egyéb
8088	-	16	4,77-16	8	-	-
80286	-	16	6-25	16	-	-
80386SX	-	32	16-40	16	-	-
80386DX	-	32	20-40	32	-	-
80486SX	-	32	25-40	32	8	-
80486DX	+	32	25-50	32	8	-
80486DX/2	+	32	50-80	32	8	-
80486DX/4	+	32	80-120	32	8	-
AMD 5x85-P75	+	32	133	32	16	-
Pentium	+	32	60-166	64	8+8	-
AMD K5	+	32	90-133	64	16+8	-
Cyrix 6x86(M1)	+	32	100-180	64	16	-
Pentium MMX	+	32	166-300	64	16+16	MMX
AMD K6	+	32	166-266	64	32+32	MMX
Cyrix MII	+	32	150-220	64	64	MMX
Pentium II	+	32	233-450	64	16+16	MMX
Celeron	+	32	233-450	64	16+16	MMX 0-128 KB L2 cache
AMD K6-2	+	32	266-450	64	32+32	MMX 3D-Now! Graf. kieg.
Pentium III	+	32	450-	64	16+16	MMX ISSE Graf.kieg.
AMD K6-III	+	32	400-	64	32+32	MMX 3D-Now! Graf. kieg. 256K L2 cache
AMD K7	+	32	600-	64	128	MMX 3D-Now! Grafikai kiegészítés 512 KB cache
Pentium Pro	+	32	166-200	64	8+8	256-1024KB L2 cache
Pentium II Xeon	+	32	400-	64	16+16	512-2048KB L2 cache

Megjegyzés:

1. *A 386/SX processzor utasításkészletét tekintve teljesen kompatibilis a 386-ossal, csak az adatvezetékek száma kevesebb: 16.*
2. *A 486/DX processzor utasításkészletében teljesen kompatibilis a 386-ossal, de az áramköri tok tartalmaz egy másik processzort is, amely a számításigényes műveletekben segít. Ezt a processzort nevezik matematikai társprocesszornak, vagy koprocesszornak. Ezenkívül tartalmaz egy belső gyorsítótárat, egy ún. cache memóriát.*
3. *A 486/DX2 processzor olyan, mint egy „rendes" 486/DX processzor, csak a belső órajele (66 MHz) kétszerese a gép órajelének (33 MHz).*
4. *A 486/DX4 processzor olyan, mint egy „rendes" 486/DX processzor, csak a belső órajele (Pl 100 MHz) háromszorosa a gép órajelének (33 MHz).*
5. *A 486DX/4-et követő processzorokban ez már „természetessé" vált. Pl.: az AMD K7-es processzor esetén a külső órajel 200 MHz, a belső 500 MHz.*
6. *A 486/SX processzor nem tartalmaz koprocesszort, de egyébként mindenben megegyezik a DX változattal. (Pontosabban az SX-et a DX gyártása során keletkező selejtből nyerik: azok a processzorok lesznek az SX-ek, amelyekben csak a koprocesszor hibás. Ennek a működését letiltják, és a processzort SX-ként használják.)*
7. ***MMX**: 1996 végén az Intel bejelentette az új, MMX-es processzorát. (MMX=Multimedia Extension). Az utasításkészlet 57 új utasítással bővült, aminek a segítségével egyszerre több (fixpontos!) adaton képesek műveleteket végezni. A multimédiás adatok (kép, hang, stb) könnyebb és gyorsabb kezelését teszik lehetővé az új utasítások.*
8. ***3DNow!**: Az AMD cég 21 új utasítással bővítette az MMX processzorok utasításkészletét az AMD K6 2 processzorban. Ezeknek az utasításoknak hasonló a célja, mint az MMX utasításoknak, csak ezek lebegőpontos adatokkal dolgoznak.*
9. ***ISSE:** Az Intel cég a Pentium III-ban a 3DNow!-hoz hasonló céllal bővítette az utasításkészletet 74 új utasítással.*

7.2 Alaplap

A PC alaplapján, vagy más néven alapkártyáján helyezkednek el a legfontosabb egységei a számítógépnek.



7.2.1 A buszrendszer

A számítógép belső eszközeit összekötő vezetékkötegeket nevezzük busznak. Amikor egy új bővítőkártyát helyezünk az alaplapon lévő csatlakozók egyikébe, akkor tulajdonképpen közvetlenül a buszra csatlakozunk. Ha egy adat továbbításra kész (a memóriába), akkor a rendeltetési hely címét először el kell küldeni a *címbuszon*, és ezt követi az adat az *adatbuszon*. Mivel több eszköz is ugyanarra a buszra csatlakozik, ezért az adatátvitel alkalmával a következőket kell megoldani:

- Az adatátvitelben résztvevő eszközök kijelölését.
- Az adatátvitel irányát.
- A kapcsolatban résztvevők szinkronizálását.

- **Címbusz**
  A Pentium processzor címbusza 32 vezetékből áll, így a legnagyobb memóriacím, amit elérhetünk: $2^{32}-1$ bájt (4 Gbájt). Valós módban viszont csak 20 vezetéket használ (1 Mbájt).

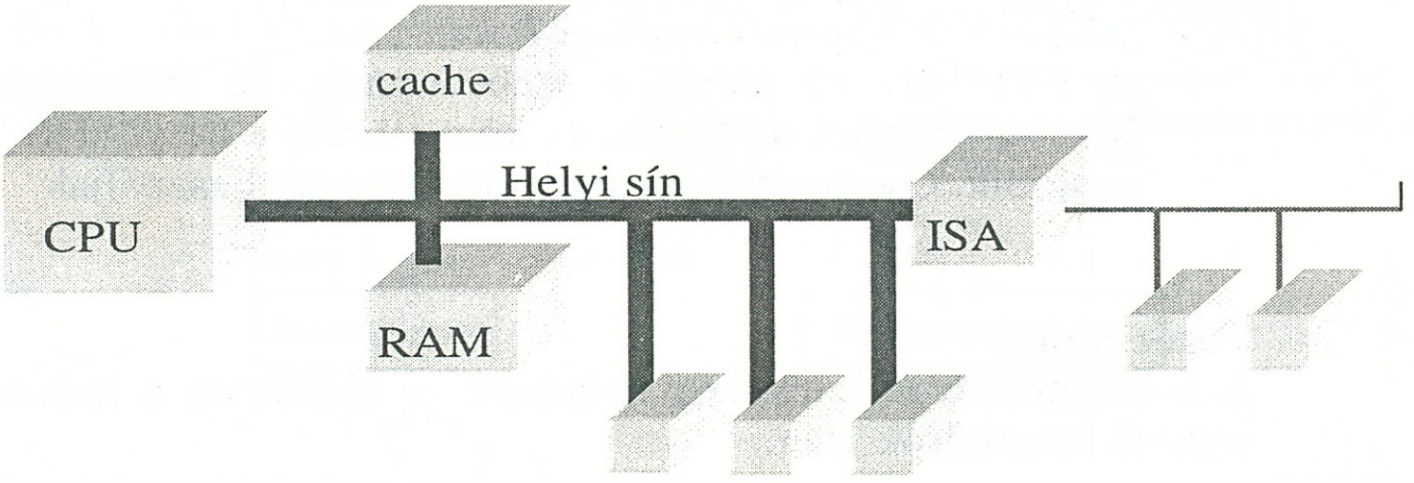
- **Adatbusz**
  A Pentium 64 adatvezetékkel rendelkezik. Különféle szabványok léteznek az adatbusz megvalósítására. Ezek közül a legelterjedtebb az AT géptípusokon az úgynevezett *ISA* busz, de egyre inkább terjednek a korszerűbb, 32 bites szabványú adatbuszok, például az EISA, VESA Local Bus, PCI, SCSI és az AGP.

- **ISA** *(Industry Standard Architecture)*
  16 bites adatátvitel és 8 MHz frekvencia jellemzi . Előnye az olcsó megvalósíthatóság, hátránya az, hogy a 32 bites Pentiumos gépben is csak 16 bites adatátvitel lehetséges.

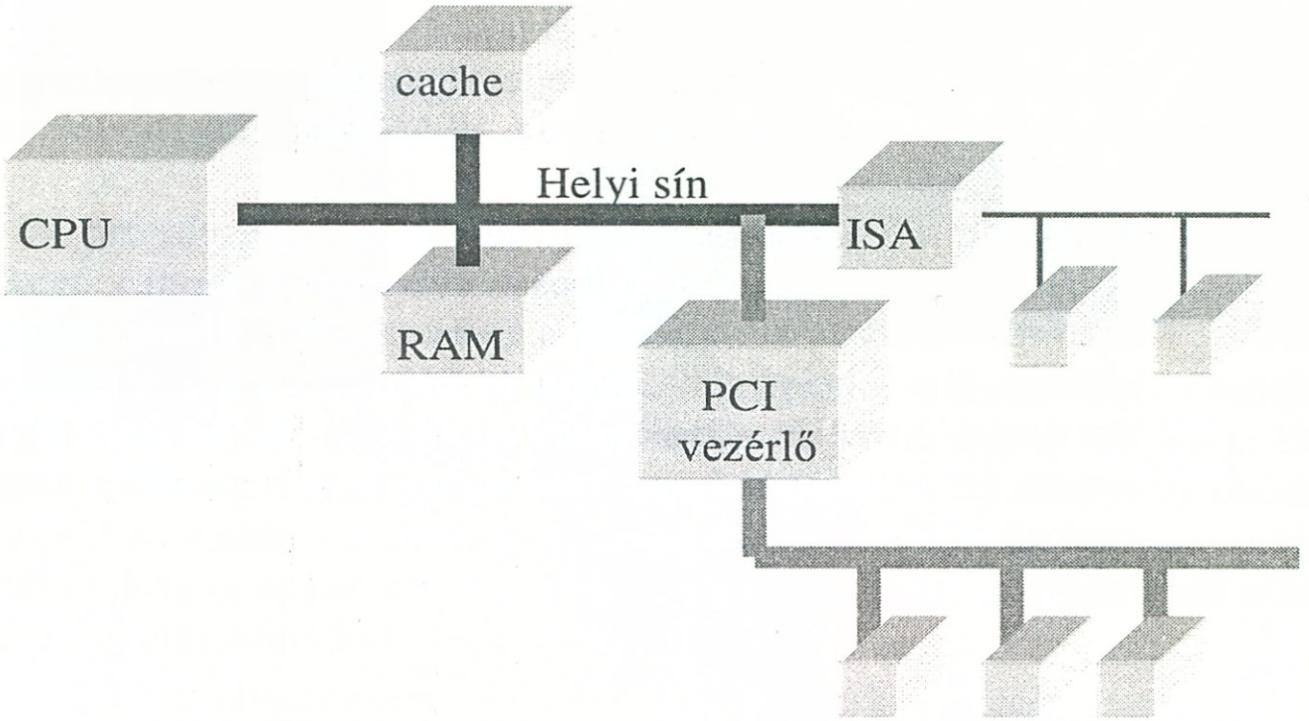
- **EISA** *(Extended Industry Standard Architecture)*
  Az ISA továbbfejlesztése: 32 bites lett az adatátvitel, de a sebesség továbbra is 8 MHz maradt.

- **MCA** *(Micro Channel Architecture)*
  Az IBM cég a PS/2 gépekre fejlesztette ki a mikrocsatornát. Ez is 32 bites adatbusz tartalmaz, 10 MHz mellett.

- **VLB** *(Vesa Local Bus)*
  A helyi sín (local bus) a processzorhoz közvetlenül kapcsolódó részt jelenti, közvetlenül a CPU vezérli 32 bites adatátvitellel. Ezt a helyi sínt „hosszabbították" meg a VLB-vel. Az órajel 33 MHz. Gyakorlatilag csak a 486-os gépekben terjedt el. Hátránya, hogy viszonylag kevés periféria kapcsolódhat (max. 2-3 eszköz).



➢ **PCI** *(Peripherial Component Interconnect)*
A PCI, ellentétben a VLB-vel, nem egy meglévő sínrendszerre épül, hanem teljesen önálló szabvánnyal rendelkezik. Az Intel cég fejlesztette ki a Pentiumos gépekhez. Az adatátvitel 64 bites, az órajel 33 MHz, elvileg 10 eszköz csatlakoztatható hozzá.



➢ **AGP** *(Accelerated Graphics Port)*
A PCI busz az alapja a szabványnak, de magasabb, 66 MHz-s órajellel működik, és közvetlenül a helyi sínre kapcsolódik. Gyakorlatilag csak video kártyák használják.

- **Vezérlőbusz**
A hozzátartozó vezetékeken haladnak az alaplapon a különböző vezérlő áramkörök jelei.

- **Tápvonalak**
Az ún. tápvonalak az egyes egységek áramellátását biztosítják.

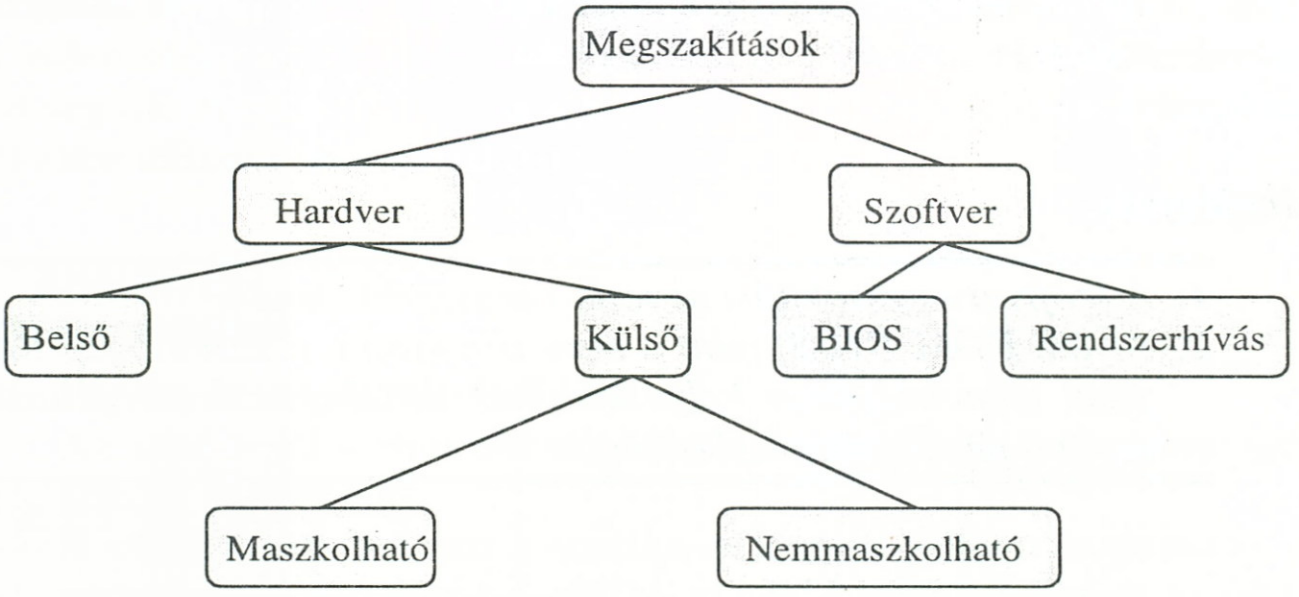
7.3 Kiszolgáló áramkörök

7.3.1 Megszakítás vezérlő

A megszakítás vezérlő az ún. *megszakítások* működését felügyeli. A megszakítási jelet általában a hardver küldi a CPU-nak. Pl.: egy hardvereszköz felhívja magára a figyelmet, hogy valamilyen intézkedésre van szükség a processzor részéről.

A megszakítás vezérlő (röviden: MV) fogadja ezeket a jeleket, és meghatározza fontosságukat a korábban érkezettekhez képest. A döntés után megszakítást küld a CPU-nak. A CPU, miután érzékeli a megszakítást, elindítja az adott megszakításhoz tartozó programot. Miután a megszakítás vezérlő program befejeződik, a CPU ott folytatja a tevékenységét, ahol a megszakítás előtt abbahagyta.

A megszakítások fajtái:



- **Hardvermegszakítások**
  Ezek tulajdonképpen azok a megszakítások amelyeket, a MV kezel. A hardvermegszakításoknak két fajtája van:

- **külső**: A számítógép különböző részei adnak jelet a CPU-nak, hogy felhívják magukra a figyelmet. (pl.: lenyomunk egy billentyűt).
- **belső**: Olyan megszakítások, amelyeket a CPU hoz létre. Ezek a programban fellépő rendkívüli események következményei (pl.: 0-val történő osztás).

- **Nemmaszkolható megszakítás (NMI)**
  Ezek olyan megszakítások, amelyek a processzor azonnali reakcióját váltják ki (pl.: a memória meghibásodik). Az NMI-t az első helyre helyezi a megszakítás vezérlő. Azok a megszakítások, amelyek letilthatók, azaz nem kerülnek feltétlenül végrehajtásra, az ún. *maszkolható megszakítások* (pl.: egy billentyű leütése során keletkező megszakítás).

- **Szoftvermegszakítások**
  Ezt a fajta megszakítást egy gépi kódú utasítás hozza létre egy programon belül (pl.: az 5. számú megszakítás a *Print Screen* funkciót váltja ki, azaz a képernyő tartalmát kinyomtatja a nyomtatón). Így hív meg a program egy RAM-ban vagy a ROM-ban tárolt szubrutint (alprogramot), ami a BIOS, illetve az operációs rendszer szolgáltatásainak a része.

Megjegyzés:

1. *A szoftvermegszakítást nevezik rendszerhívásnak.*
2. *Az ún.* ***ROM BIOS****, mint a neve is mutatja, a számítógép ROM típusú memóriájában helyezkedik el, és alapvető szolgáltatásokat nyújt a perifériák kezeléséhez.*

Bármelyik megszakításról is van szó, a megszakítást kérőnek nem kell ismernie a megszakítás vezérlő program címét, elég a megszakítás sorszámát ismernie. A megszakítás vezérlő programok címeit egy táblázat tartalmazza, az ún. *megszakítási vektortábla*, és a megszakítás sorszáma alapján ebből a táblázatból kikereshető a megfelelő cím.

7.3.2 DMA vezérlő

A számítógép egyes részei a CPU megkerülésével, közvetlenül fordulhatnak a memóriához, így tehermentesítik ez alól a CPU-t. Ezt hívjuk közvetlen memória hozzáférésnek (DMA).

A DMA vezérlő feladata, hogy a lemezes írási, olvasási műveleteket a CPU nélkül is végre lehessen hajtani. Mivel a lemezműveletek viszonylag lassúak, ezért a DMA jelentősen gyorsítja a számítógép működését.

7.3.3 Órajel generátor

Ha a CPU-t a számítógép „agyának" tekintjük, akkor az órajel generátor a számítógép „szíve". Ez a „szív" másodpercenként több milliószor „üt", ő ütemezi a CPU és a többi egység működését. Ez a frekvencia nagy hatással van a számítógép működésének a sebességére.

7.3.4 Koprocesszor

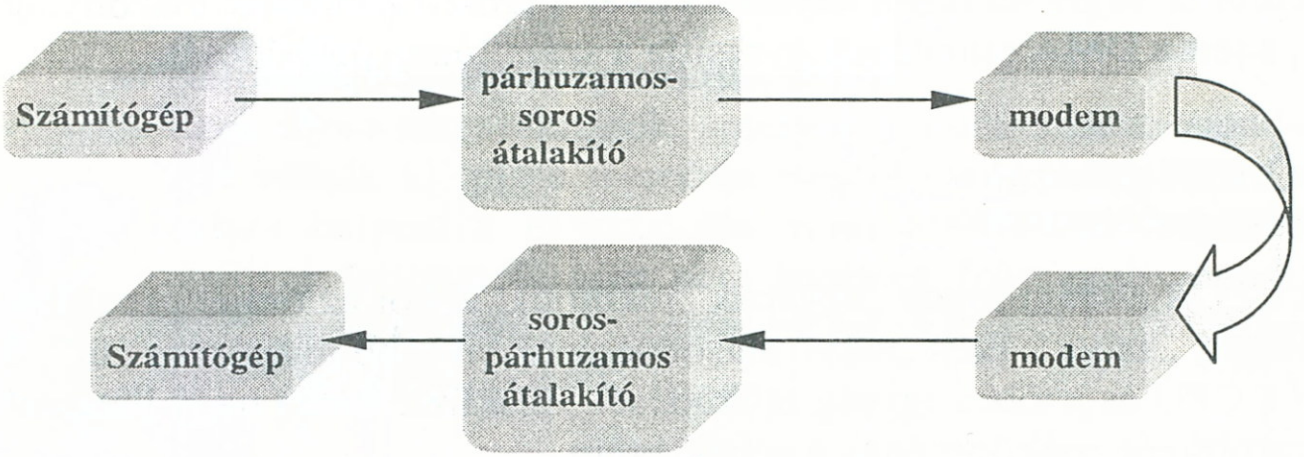
A koprocesszor egy ún. társprocesszor, amely önállóan, CPU nélkül képes műveleteket végezni. Ezek a koprocesszorok lehetnek olyanok, amelyek a video megjelenítést vagy olyanok, amelyek a matematikai műveletvégzést segítik. A matematikai társprocesszorok a lebegőpontos számokkal végzett számításokat támogatják.

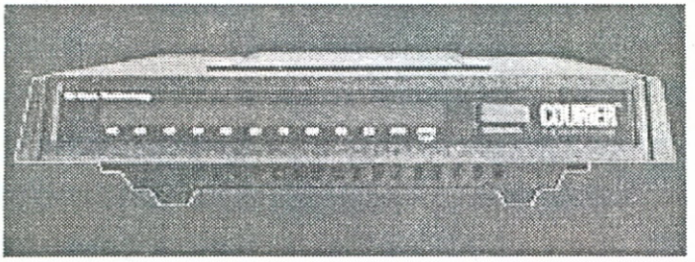
7.3.5 Adatátvitel

- **Párhuzamos adatátvitel**
  Az adat bitjei párhuzamosan haladnak a periféria és a számítógép között, bájtonként.
  A párhuzamos vonalat használja például a printer (nyomtató). A párhuzamos vonalakhoz a BIOS az *LPT1, LPT2, LPT3* és az *LPT4* logikai neveket rendeli.

- **Soros adatátvitel**
  Ennél az adatátviteli módnál az adat egyes bitjei sorban egymás után haladnak, általában 1 bájtos alakban a perifériát és az illesztőt összekötő vezetéken. Ehhez először az illesztőben az adatbuszról érkező jeleket sorossá kell alakítani, és a jeleket így továbbíta-

ni a periféria felé. A perifériában a soros jeleket vissza kell alakítani párhuzamos formájúvá.
A soros vonalat használja például az egerek egy része, vagy az ún. *modem*, amelynek a segítségével telefonvonalra csatlakoztathatjuk a számítógépet, és kapcsolatot teremthetünk egy másik géppel.

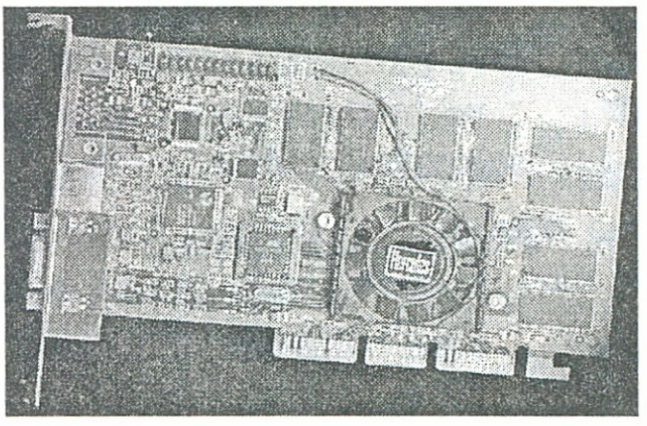




Az egyes soros vonalakra a *COM1, COM2, COM3* és *COM4* neveken hivatkozhatunk.

7.3.6 Periféria csatlakozások

- **Monitorvezérlő**
  A monitor típusától függően különböző vezérlő áramkörökre van szükség a monitoroknak a számítógéphez való illesztésére. A monitorokkal a későbbiekben foglalkozunk. 1997-ben jelentek meg az ún. *3D-s gyorsító kártyák*, amelyek a térbeli grafikai megjelenítést segítik.

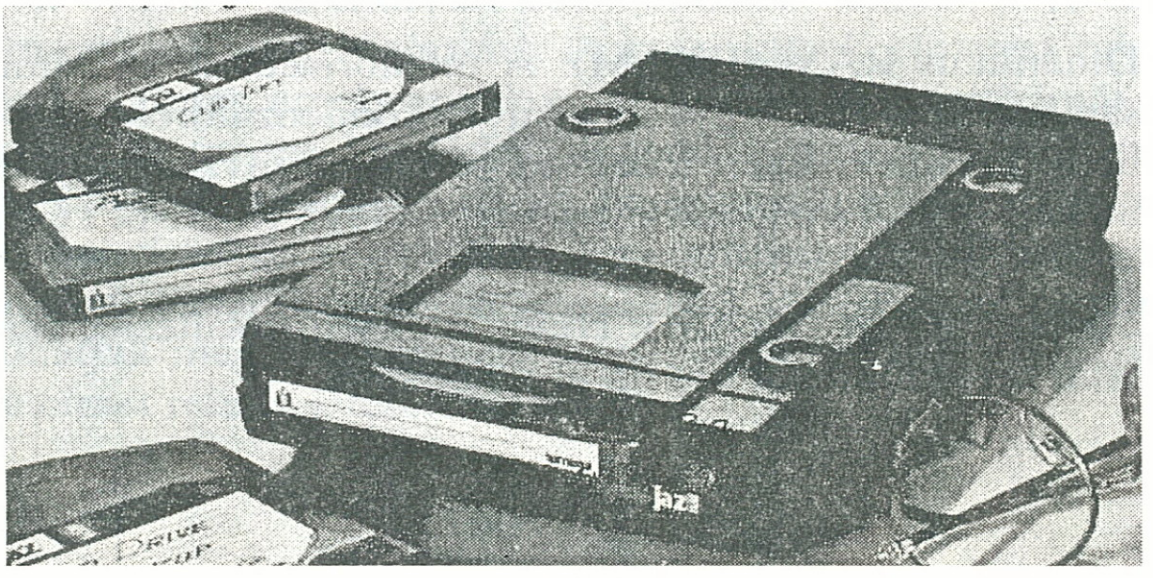


- **Winchester, CD-ROM vezérlő**
  A winchester vezérlő áramkör ma már a készülékkel egybeépítve helyezkedik el. Az adatátvitel az IDE vagy SCSI csatolón keresztül történik.

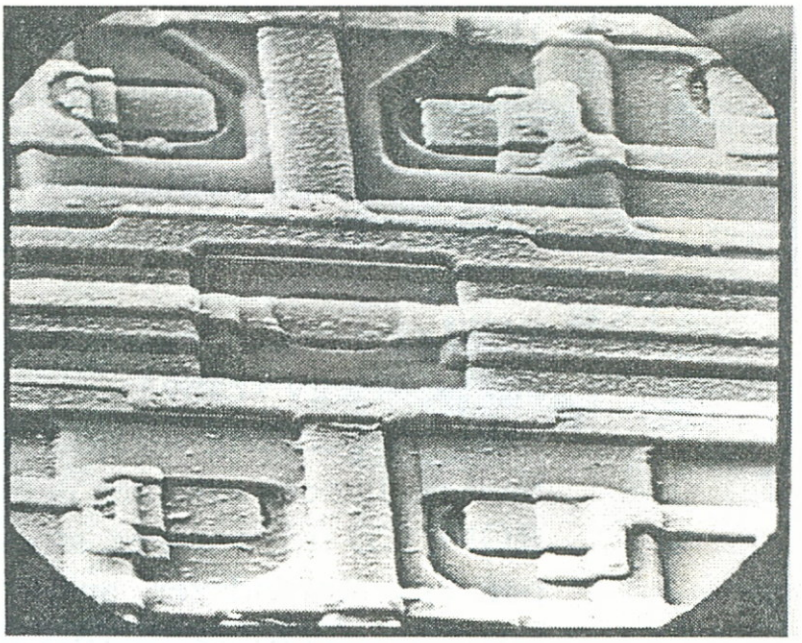
  - **IDE** *(Integrated Device Electronic)*
    AT buszos csatolónak is szokták nevezni. A vezérlő elektronika magán a winchesteren helyezkedik el, és közvetlenül az ISA buszra csatlakozik. Az adatátvitel párhuzamos. Továbbfejlesztett változata az EIDE (Enhanced IDE).

  - **SCSI** *(Small Computer System Interface)*
    Az SCSI egy szabványosított, univerzális, párhuzamos, intelligens periféria illesztő. Nem csak merevlemez kapcsolható rá, hanem sok más is (8 db összesen): hangkártya, CD-ROM, CD író, scanner, streamer, stb. Az SCSI tulajdonképpen egy sínrendszer, amely a különböző perifériák csatlakoztatására szolgál. 8,16 vagy 32 bites adatátvitel érhető el vele, típustól függően.

- **Hajlékonylemezes meghajtó vezérlő**
  A hajlékonylemezes meghajtó adatforgalmának vezérlését látja el. Az AT gépekben lehetőség van 360 Kbájtos, 720 Kbájtos, 1,2 Mbájtos, és 1,44 Mbájtos lemezek használatára is, de egyszerre csak két meghajtót használhatunk. Ma már más szabványú, nagyobb kapacitású eszközök is léteznek: LS, Zip Jaz drive, stb.



7.4 A memória



Amint azt láttuk, a memóriák bájtszervezésűek, azaz minden rekesz 1 bájt információt tárol. A mai félvezetős memóriákban valamennyi memóriarekesz mellett egy 9. bit is van, az ún. *paritásbit*. A paritásbit szerepe az esetleges tárolási hiba felismerésében van (redundancia!). A paritásbit értéke 0 vagy 1 lesz attól függően, hogy a rekeszbe íráskor páros vagy páratlan a tárolt egyesek száma. Ha a rekesz kiolvasásakor nincs összhangban az egyesek száma a paritásbit értékével, a CPU megszakítást kap, amely a memóriahibát jelzi.

A PC-kben levő félvezetős memóriák véletlen hozzáférésűek, bármely memóriarekesz eléréséhez kb. ugyanannyi idő szükséges. Az elérési idő 60-70 ns közötti idő általában, és ezen azt értjük, hogy ennyi idő szükséges egy adott memóriarekesz „megtalálásához" és az onnan kiolvasott adatnak a CPU-ba való megérkezéséhez.

Két alapvető típusa létezik a félvezető memóriáknak:

- Dinamikus RAM (DRAM): A memóriacellák tartalmát folyamatosan frissíteni kell, ugyanis a tároló kondenzátorok nagyon rövid időn belül kisülnek. Ezért időről-időre ki kell olvasni a tartalmat, és újra be kell írni. A központi memóriában levő RAM-ot készítik belőle.

- Statikus RAM (SRAM): Nincs szükség frissítésre, csak az áramellátás megszűntekor veszíti el a tartalmát.

7.4.1 A Pentium processzor címzése valós módban

Azoknak a tárhelyeknek a száma, amelyekhez a processzor hozzá tud férni, egy speciális regiszternek, a *címregiszternek* a nagyságától függ.

Mindegyik tárhelynek saját címe van, amelyen keresztül elérhetjük. Az egyes tárhelyek 0-tól kezdve, folyamatosan vannak számozva, ezek a sorszámok a *címek*. A címregiszterben ábrázolható legnagyobb szám határozza meg azt, hogy hány különböző tárhelyhez tud fordulni a processzor. A 8086-os előtti mikroprocesszorokban a címregiszter 16 bites volt. Ez a regiszter 0 és 65535 (0 és 2^{16}-1) közötti értékeket vehet fel, így összesen 65536 féle memóriarekesz címezhető. Ez összesen 64 Kbájt tárkapacitást jelent. Ilyen volt például a ZX Spectrum processzora, a Z80.

Mint tudjuk Pentium processzor valós módban úgy működik, mint egy gyors 8086-os, így ami az utóbbira igaz, az érvényes a Pentiumra is. Felmerült az a probléma a 8086-os esetén, hogy 1 Mbájt memóriát kellene címeznie. Ehhez viszont 20 bites címregiszter szükséges. A technika akkori szintjén ez nehézséget okozott volna. Ezért találták ki a *szegmentált* tárcímzést.

A 8086-as nem tartalmaz külön címregisztert, hanem két közönséges 16 bites regisztert használ.

7.4.2 Szegmentált címzés

Az első 1 Mbájthoz (a legnagyobb cím: 2^{20} -1 = 1048575) tartozó tárterületet ún. *szegmensekre* osztjuk. A szegmensek maximális mérete 64 Kbájt, és 16-tal osztható tárcímeken kezdődhetnek. Ezeket a 16-tal osztható címeket nevezzük *paragrafusnak*. A szegmensek címéhez hozzárendelünk egy logikai címet, amely annak a paragrafusnak a sorszámát jelenti, amelyen az adott szegmens kezdődik. A valódi, fizikai kezdőcímét úgy kaphatjuk meg, ha az előző logikai címet megszorozzuk 16-tal. (Hiszen 16-tal osztható fizikai címeken kezdődhet.) A szegmensen belüli, maximálisan 64 Kbájt nagyságú tárterületen levő tetszőleges tárcímet, az úgynevezett *ofszetcímet* adhatjuk meg. A szegmens kezdetén az ofszetcím 0. Ezért nevezik az ofszetcímet relatív eltolásnak is.

Megjegyzés:

1. *Mivel a szegmensek 16-tal osztható címeken kezdődhetnek, ezért összesen 65536 különböző szegmenst lehet használni. (1048576 : 16 = 65536) Ez a szám éppen megfelelő, hiszen a 16 bites regiszterben is éppen ennyiféle szám tárolható.*
2. *Mivel a szegmensek maximális mérete legfeljebb 64 Kbájt (65536 bájt), az ofszetcím is legfeljebb ekkora lehet, az pedig még éppen tárolható egy 16 bites regiszterben.*
3. *Láttuk, hogy a szegmensek 16-tal osztható címeken kezdődhetnek, és maximális méretük 64 Kbájt, ezért a szegmensek természetesen átfedhetik egymást.*

A 20 bites cím az úgynevezett *címösszeadó* egységben áll elő, a szegmenscím és az ofszetcím összeadásával. Pontosabban nem egyszerűen a két cím összeadásáról van szó (hiszen így maximálisan 17 bites címet kaphatunk), hanem a szegmens fizikai címéhez (amely 20 bites, hiszen a logikai címet 16-tal meg kell szorozni, ami a bináris számrendszerben 4 számjeggyel való balra tolást jelent) kell hozzáadni az ofszetcímet.

Például:

Legyen az adott memóriarekesz fizikai tárcíme decimálisan **759 314**. (Ez a hexadecimális számrendszerben: B9612). Ez a tárcím beleesik a **47 456**-os sorszámú (**B960**) szegmensbe. Mivel **47 456** x 16 = **759 296**, ezért ehhez még hozzá kell adni az ofszetcímet (a relatív eltolást), hogy megkaphassuk az eredeti fizikai címet, azaz 18-at (12). Tehát a szegmenscím **47 456** (**B960**), az ofszetcím 18 (**12**).

Hexadecimális számrendszerben:

$$\begin{array}{rr} & \mathbf{B9600} \\ + & \mathbf{12} \\ \hline & \mathbf{B9612} \end{array}$$

Megjegyzés:

Hexadecimális számrendszerben a 16-tal való szorzást úgy is elvégezhetjük, hogy a számjegyeket egy pozícióval balra toljuk, és a végére egy 0-át illesztünk.

Kettes számrendszerben:

```
   1011 1001 0110 0000 0000
+  0000 0000 0000 0001 0010
   1011 1001 0110 0001 0010
   B    9    6    1    2
```

Megjegyzés:

Kettes számrendszerben a 16-tal való szorzást úgy is elvégezhetjük, hogy a számjegyeket 4-gyel balra toljuk, és a végére négy 0-t illesztünk.

A 16-tal való szorzásnak megfelelően a szegmenscímeket mindig 5 hexadecimális számjeggyel írhatjuk le, és az utolsó jegy természetesen mindig 0. A teljes (20 bites) fizikai cím is mindig 5 hexadecimális számjeggyel írható le.

Ha a 20 bites címet szegmenscímre és ofszetcímre bontva írjuk le, akkor az előző tárcím a következőképpen adható meg:

B960 : 0012

Megjegyzés:

Ugyanazt a 20 bites fizikai címet természetesen többféle szegmentált címmel is megadhatjuk. Az előző példában szereplő cím (B9612) például a következőképpen is felírható:

B961 : 0002

7.4.3 A memória felépítése MSDOS-ban

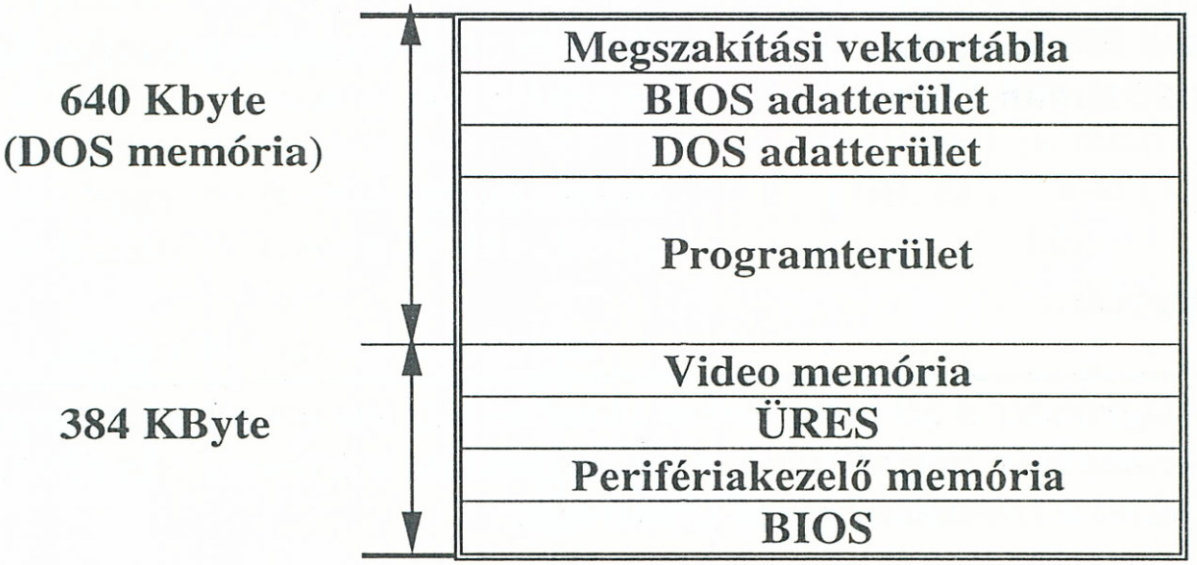
A memória alapvetően két részből áll: RAM és ROM. A ROM típusú memóriában található a BIOS, és vannak bővítőkártyák, amelyek tartalmaznak saját ROM-ot. Ilyen például néhány winchester-vezérlő, a hálózathoz való csatlakozást biztosító ARCNET kártya, stb.

A RAM legfontosabb része az ún. *DOS* vagy hagyományos memória, amelynek a maximális mérete *640 Kbájt*. Ez a méret a mai nagyméretű programok futtatásához általában elég szűkös, de a gyártók ragaszkodnak ehhez a 640 Kbájtos határhoz, ami valószínűleg üzleti okokra vezethető vissza, hiszen így a programok futtathatók a legfeljebb 640 Kbájt RAM-ot tartalmazó XT gépeken is.

Megjegyzés:

Ez a határ csak az MSDOS-hoz kötődik, más operációs rendszerekben (UNIX, Windows 95, Windows 98, Windows NT, OS/2) nincs ilyen probléma, hiszen képesek a teljes memóriát használni.

A következő ábrán a memória vázlatos felépítése látható 1 Mbájtig.



Mint látható, a 640 Kbájtos határ és az 1 Mbájtos határ közötti 384 Kbájtos területen található az ún. *videómemória* is, amely a monitor működéséhez szükséges RAM. A ROM BIOS mérete általában 64 Kbájt.

Az MSDOS memória kis méretéből adódó nehézség áthidalására több megoldás is született. A következőkben ezeket ismertetjük vázlatosan.

- **Overlay technika**
  Ezzel a lehetőséggel majd később ismerkedünk meg.

- **Virtuális memória**
  Ezt a megoldást is később ismerjük meg alaposabban. Az MSDOS nem ismeri ezt a technikát, a Windows viszont igen.

- **Extended memória (XMS)**
  Az 1 Mbájt feletti memóriát a Pentium képes alkalmazni védett módban, és vannak operációs rendszerek, amelyek ezt a lehetőséget igénybe veszik. (Ilyen a Windows 95 is.) Használatához szükséges egy ún. extended memóriamenedzser-program is. (Pl.: HIMEM.SYS)

- **Expanded memória (EMS)**
  Ezt a megoldást eredetileg az XT gépekhez találták ki, amelyet speciális memóriabővítő kártya beépítésével lehetett létrehozni. Nem terjedt el igazán ez a megoldás (Magyarországon legalábbis), ma már inkább az AT gépeken találkozhatunk vele, hiszen ezekbe a gépekbe általában eleve 640 Kbájtnál nagyobb RAM van beépítve.
  A módszer lényege a következő:
  Tegyük fel, hogy a számítógép több memóriát tartalmaz, mint amit a címtartomány megenged. (Valós mód!) Ezt a többletmemóriát 16 Kbájtos részekre, ún. lapokra osztják, és egy speciális program segítségével összesen 4 lapot (= 64 Kbájt) kapcsolhatunk be a processzor által címezhető tartományba. Ez a program az ún. expanded memória menedzser (EMM) (pl.: EMM386.EXE). Az EMM keres egy olyan nem használt területet az 1 Mbájtos címtartományban, ahol egy 64 Kbájtos munkaterületet alakíthat ki. (Angolul ezt *page frame*-nek nevezik.) A page frame az előző ábra üresnek jelölt tartományában helyezhető el. Miután ez megtörtént, a munkaterületet 4 lapra osztja. Erre a 4 lapra kapcsolja be a

többletmemória egy részét. A négy lapra bekapcsolt részek természetesen folyamatosan cserélődnek az igényeknek megfelelően.
A megoldás előnye az, hogy a többletmemóriát az 1 Mbájtos határ alatti címen használhatjuk.

Megjegyzés:

1. *Az EMS memóriát sok program használja, például a Turbo Pascal 6.0.*
2. *A Windows megjelenésével és rohamos elterjedésével az EMS-használat kezd háttérbe szorulni, mert mint láttuk, a Windows az XMS memóriát használja.*

- **Felső-memóriablokk (UMB)**
  Néhány speciális 286-os, és a 386-ostól fejlettebb processzorú gép képes az extended memória egy részét 640 Kbájtos és 1 Mbájtos határ közötti szabad memóriaterületre leképezni. Ennek használata már az MSDOS 5.0-hoz kötődik. Az UMB-re általában ún. *eszközvezérlő* programokat szoktak tölteni, amelyek többnyire valamilyen periféria vezérlését, illesztését oldják meg. (Például az egér használatához is szükséges egy ilyen vezérlőprogram.)

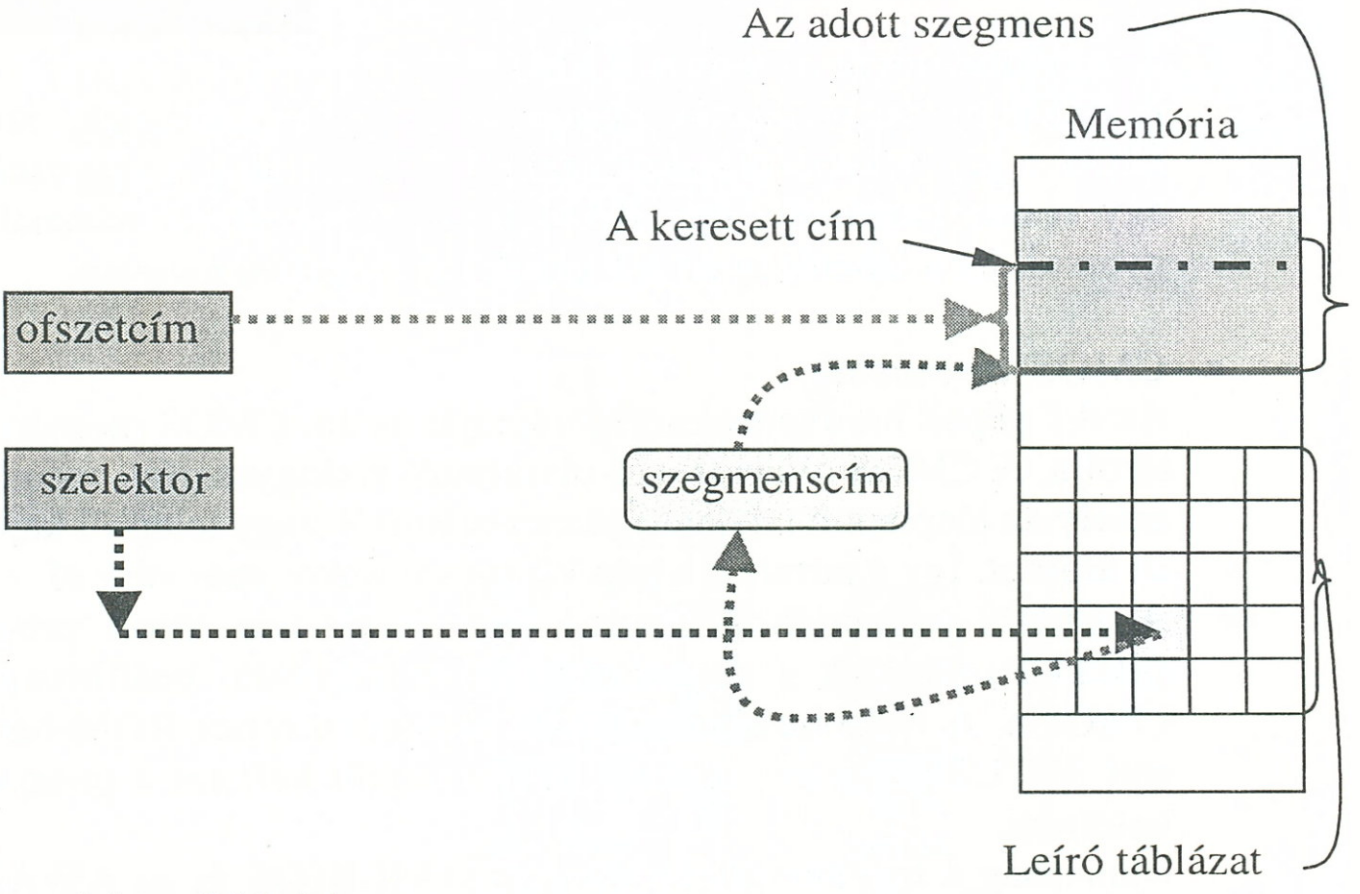
- **Magas memória terület (HMA)**
  Ha utána számolunk, akkor kiderül, hogy az utolsó szegmens az 1048560-as (FFFF0) címen kezdődik (azaz FFFF a szegmenscím). Ennek a maximális mérete így csak 16 bájt lehet, de mivel a ROM BIOS az utolsó 64 Kbájtot foglalja el, így ez a 16 bájt RAM-ként nem jöhet számításba. Ha bekapcsoljuk a 21. címvezetéket is, akkor ez az utolsó szegmens is teljes méretében kihasználható.

Megjegyzés:

Az egyes címvezetékekre a A0, A1, ... , A19 jelöléseket használják. A 21. vezetéket tehát A20-szal jelölik.

7.4.4 A Pentium processzor címzése védett módban

Valós módban a szegmenscímeket a **szegmensregiszterek** tartalmazzák. Védett módban a szegmensregiszterek ún. **szelektor**okat tartalmaznak, amelyek egy táblázatban levő leírót választanak ki. Ez a leíró tartalmazza az aktuális szegmens címét, méretét és egyéb jellemzőit. A leíró 8 bájtos, és a szegmenscím 32 bites (és nem 20 bites), így lényegesen nagyobb memória címezhető meg. Valós módban a szegmens mérete 64 KB, védett módban beállítható. Alapértelmezésként ez 1MB (2^{20}-1), de beállítható, hogy a 2^{20}-1 bájtban, vagy 4 KB-os *lapok*ban értendő-e. Az újabb processzorokban a lap mérete 4 MB is lehet. Az ofszetcím most is szegmens elejéhez viszonyított relatív címet adja meg, maximális értéke nyilván a szegmens méretével egyezik meg. A szegmenscím most egy lineáris, 32 bites cím, tehát nem 16 bájtos egységekre van osztva.



A szelektor felépítése:

Index	TI	RPL

Index: A leíró táblázat egy cellájára mutat.

TI: Megadja, hogy melyik leíró táblázatról van szó (lokális vagy globális).

RPL: A privilégium szintjét adja meg. Ily módon megakadályozható, hogy egy alacsonyabb privilégium szintű program hozzáférjen az adott szegmenshez.

Tehát a cím védett módban két részből áll: **szelektor:ofszetcím**. A tényleges *(lineáris)* cím a szegmenscím és ofszetcím összegeként áll elő.

7.4.5 Speciális memóriák

Az AT gépekben még egyéb memóriaegységek, illetve területek vannak, amelyeket működés közben a gép használ. Ezek a következők:

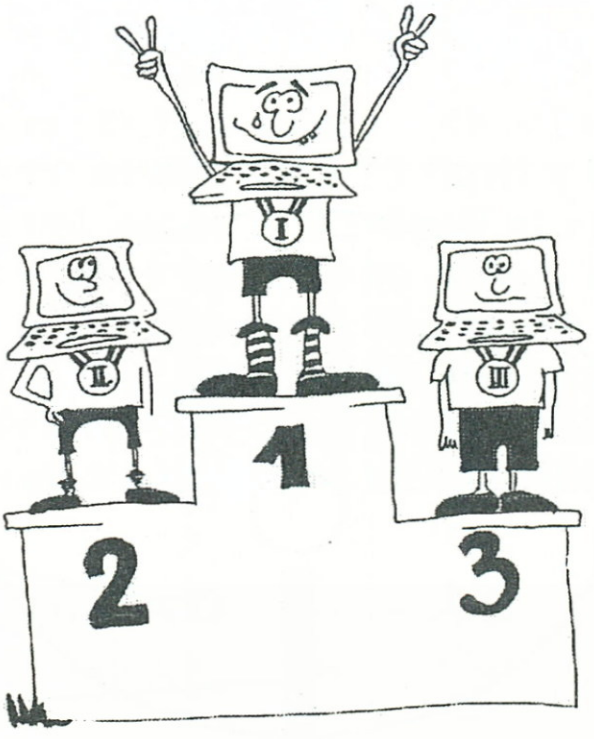
- **Árnyékmemória (Shadow RAM)**
  A ROM típusú memóriák viszonylag lassú működésűek, ezért a BIOS a gép bekapcsolásakor ezeket a memóriákat, vagy ezeknek egy részét bemásolja egy RAM területre, amellyel használatuk gyorsul. Ez a lehetőség az AT gépek többségében megvan.

- **CMOS-memória**
  Az AT gépek hardveres konfigurációját az ún. CMOS memória tárolja. A CMOS olyan írható-olvasható, kisfogyasztású memória, amely az alaplapon található akkumulátorról vagy telepről kapja az áramot. Így a tartalma a gép kikapcsolásakor sem vész el.
  A CMOS-memóriához tartozik egy ROM-ban tárolt program (**SETUP**), aminek a segítségével lekérdezhetjük, beállíthatjuk a CMOS-t. A régebbi típusú 286-os gépekben nincs ROM-ban tárolt SETUP program, ezeknél egy lemezről kell ezt a programot betölteni.
  Manapság a legelterjedtebb típus az **AMI-BIOS** és az **AWARD-BIOS**. A következők elsősorban ezekre vonatkoznak. A gép bekapcsolásakor indíthatjuk a SETUP programot, általában a memóriateszt után a DEL billentyű lenyomásával.
  Az alapvető beállítási lehetőségek általában a következőek a SETUP programban:

Date:	*Az aktuális dátum.*
Time:	*Az aktuális idő.*
Hard disk C: type:	*Az első (C) winchester típusa.*
Hard disk D: type:	*A második (D) winchester típusa.*
Floppy drive A:	*Az első floppy (A) típusa.*
Floppy drive B:	*A második (B) floppy típusa.*
Primary display:	*A monitor típusa.*
Keyboard:	*Van-e a gépre billentyűzet kapcsolva.*
Base memory:	*A DOS-memória mérete.*
Ext. memory:	*Az extended-memória mérete.*

A módosítás a Page Up és Page Down segítségével történhet.

- **Cache memória**
  Ezeknek a szerepével már foglakoztunk a processzorok esetén. Cache memóriát találhatunk a winchesterekben is, funkciója hasonló, mint a processzor cache-nek, csak amíg ott a központi tár (RAM) és a cache memória közötti sebességkülönbség miatt, itt a cache memória és a winchester közötti sebességkülönbség miatt hasznos, és gyorsítja a lemez elérését.



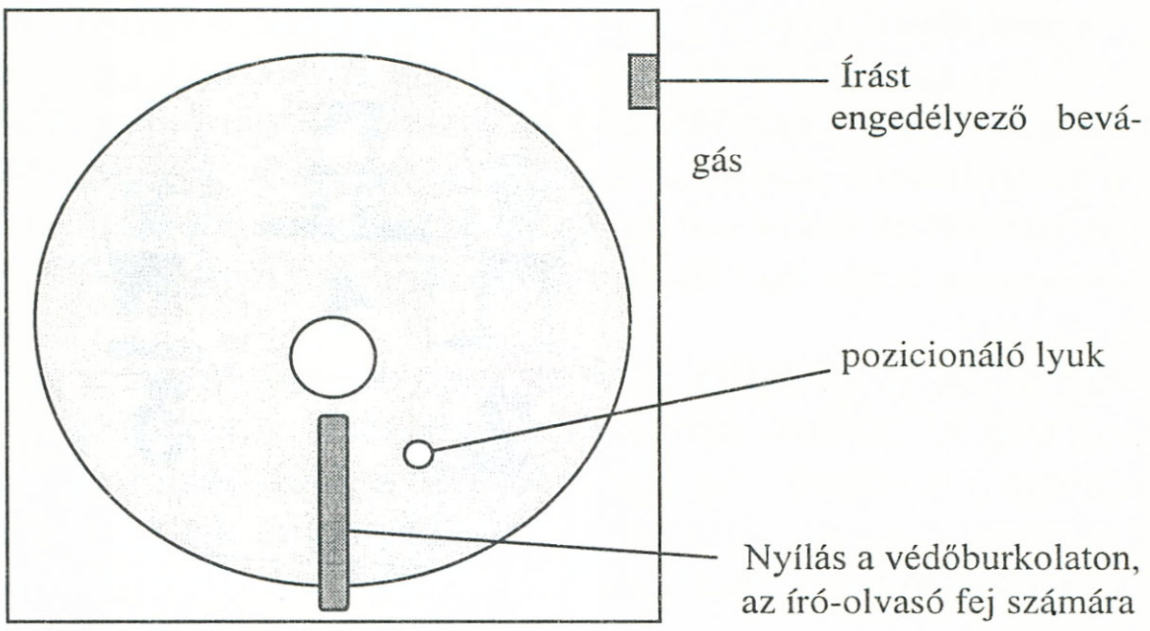
7.5 A lemezegységek

A PC-k általában legalább kétféle lemezegységgel rendelkeznek. Az egyik típus a *hajlékonylemezes egység*, (angolul *floppy disk drive* röviden *FDD*), másik a *winchester*, amelyet *merevlemezes egységnek* is szoktak nevezni (angolul *hard disk drive*, röviden *HDD*).

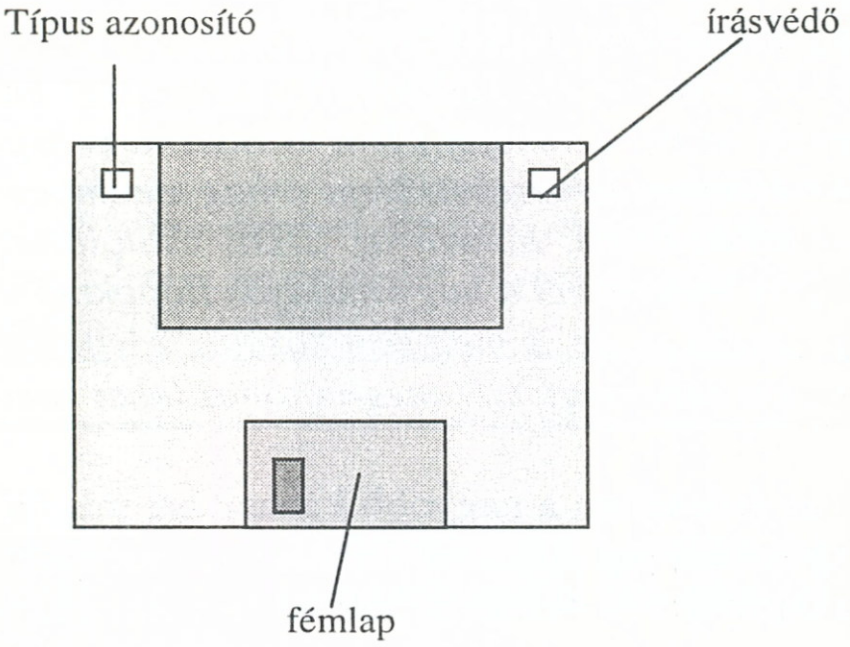
7.5.1 A hajlékonylemezes egységek

A hajlékonylemezes meghajtók felépítése röviden a következő:

Ha a lemezt behelyezzük, és a készülék előlapján levő kart lefordítjuk, akkor a lemez közepén levő lyukba egy tárcsa kerül be. Ez a tárcsa forgatja a lemezt, ha valamilyen írási vagy olvasási műveletet végzünk. A lemez mindkét oldalát mágnesezhető anyaggal vonják be, ez tárolja az információt. A lemez két oldalánál egy-egy ún. *író-olvasó fej* található, a lemez felületéhez közel. Ezek az író-olvasó fejek írják az információt a lemezre, illetve onnan leolvassák. (A jelrögzítés elve hasonló a magnetofonoknál alkalmazott eljáráshoz. Ott egy kombinált fejnek nevezett egység lát el hasonló feladatot, mint itt az író-olvasó fej.)



Az író-olvasó fejek egy karra vannak rögzítve, és sugárirányban tudnak mozogni. A lemez tasakján levő ovális alakú nyíláson keresztül tud a fej a lemezre írni, illetve onnan olvasni.



A lemezek mérete 3,5", illetve 5,25". Az 5,25"-os lemezek kétféle változatban készülnek (ma már): az egyik 360 Kbájtra (DD), a másik 1,2 Mbájtra (HD) formázható. A 3,5"-os lemezek is kétféle változatban készülnek: az egyik 720 Kbájtra (DD), a másik 1,44 Mbájtra (HD) formázható. Mindkét típus írásvédetté tehető. Az 5,25-ösnél a lemez oldalán lévő bevágás leragasztásával érhetjük ezt el. A 3,5-ösnél az egyik sarkában lévő lyukat egy kis fémlap segítségével kell elfedni ahhoz, hogy a lemezre írhassunk. A 3,5-ös lemez sokkal biztonságosabb, hiszen a lemez használaton kívül nem látszik ki (a fémlap csak használat közben, a meghajtóban nyílik ki), és a lemez tokja is erősebb. Az előző rajzon a bal felső sarokban levő lyuk segítségével azonosítja a meghajtó a DD-s illetve HD-s lemezt: a DD-snél nincs lyuk.

Megjegyzés:

1. *A továbbiakban DD-s és HD-s lemez alatt az 3,5"-os lemezeket értjük.*
2. *Szabványosnak tekinthető az a 3,5-ös lemeztípus, amely 2,88 Mbájtra formázható.*
3. *Több nagykapacitású hajlékonylemez típus is elterjedt. Ilyen például az LS drive, amely 120 Mbájtos lemezeket használ. (LS: lézervezérlésű az író-olvasófej.) Írni és olvasni is képes a hagyományos 1,44 Mbájtos lemezeket. Nem a hagyományos floppyvezérlőre kell csatlakoztatni, hanem az EIDE csatolóhoz. A hozzávaló 120 Mbájtos lemezekkel a hagyományos floppknál sokkal nagyobb sebesség érhető el, de ez meg sem közelíti a winchesterét. Nagyjából egyszeres sebességű CD-meghajtóhoz lehet hasonlítani.*

A kompatibilitás érdekében a nagyobb kapacitású meghajtók tudják kezelni a kisebb kapacitású meghajtóval formázott lemezeket is. Például az 1,2 Mbájtos meghajtóval olvashatók a 360 Kbájtos meghajtóval formázott lemezek is.

A lemezeket két nézőpontból vizsgáljuk. Megnézzük a lemezek fizikai és logikai felépítését.

7.5.2 Fizikai felépítés

A fizikai felépítést az oldalak, a sávok és a szektorok elhelyezkedése határozza meg.

- **Oldal**
  A mai hajlékonylemezeknek 2 oldala van (azaz mindkét oldalát lehet használni). Számozásuk: 0 és 1.

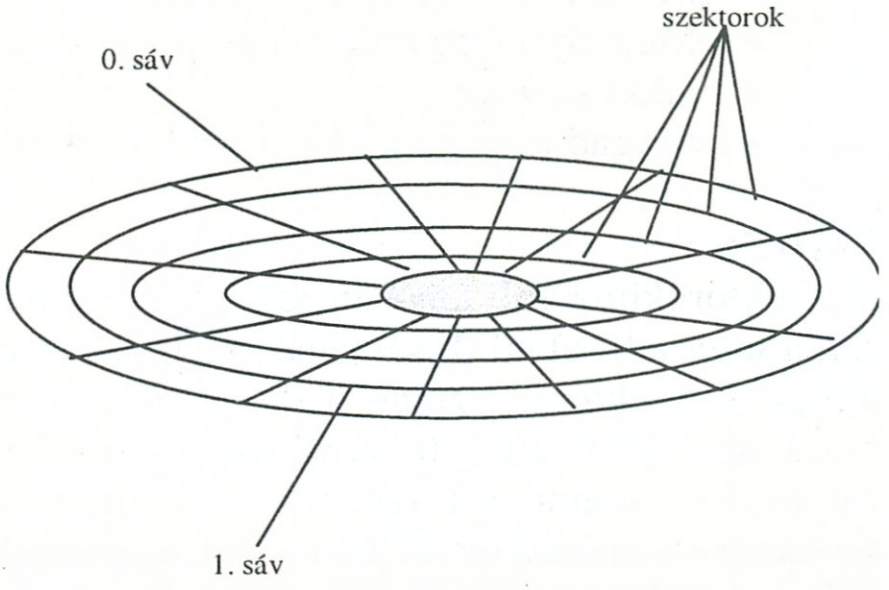
- **Sáv**
  Az adatokat az egyes oldalakon koncentrikus körök mentén, úgynevezett sávokon helyezi el a meghajtó. A sávok számozása 0-tól kezdődik. Például: A HD-s lemezen egy oldalon 80 sáv van (0-79).

Az azonos sorszámú sávokat a két oldalon együttesen *cilindernek* nevezzük. A 0-s sáv a legkülső, és a számozás befelé folytatódik. A sávok kezdetét a lemezeken az a helyzet azonosítja, amikor a floppy-n és a borítón levő lyuk fedésbe kerül. Egyúttal arra is szolgál, hogy a meghajtó megállapíthassa, hogy van-e lemez benne, és arra is, hogy megfelelő fordulatszámmal forog-e a lemez.

Megjegyzés:

A lemez csak akkor kezd el forogni, amikor szükség van rá. Ez nyilván lassítja a lemez használatát, hiszen meg kell várni, amíg a lemez a megfelelő fordulatszámot eléri.

- **Szektor**
  A sávok részekre, szektorokra vannak osztva. Az adatok végül is a szektorokra kerülnek. A szektorok számozása (egy sávon belül) 1-gyel kezdődik.
  Például: A HD-s lemeznél 18.



Megjegyzés:

1. *A szektorok mérete általában 512 bájt.*
2. *Az egyes szektorok elé egyéb segédinformációk is kerülnek, így például egy hibafelismerő kód is, amelyet CRC-nek neveznek. Ennek a kódnak a segítségével felismerhető, ha az adatok megsérülnek a szektoron. (Redundancia!)*

A hajlékonylemezek a formázás során kapják a fizikai felépítésüket.

Megjegyzés:

Lehetséges teljesen „szabálytalan" formázás is, amelyet néha másolás elleni védelemként alkalmaznak, hiszen ezeket a sávokat a DOS nem képes felismerni (de egy megfelelő programmal lehetséges).

A DOS és a BIOS különbözőképpen tartja nyilván az egyes szektorok helyét:

- **BIOS**
  Minden szektorhoz három számot rendel:
  - Megadja, hogy melyik oldalon van az adott szektor.
  - Megadja, hogy az oldalon belül melyik sávon van az adott szektor.
  - Megadja, hogy a sávon belül hányadik az adott szektor.

- **MSDOS**
  A szektorokhoz folytonosan számozva egy sorszámot rendel. A számozást a 0. oldal 0. sávjának az 1. szektorával kezdi, ez lesz a lemez 1. szektora. Ezután a 0. sáv összes szektorát végigveszi, majd áttér az 1. oldal 0. sávjának 1. szektorára, és ezen a sávon folytatja a számozást (az utolsó szektor a 30-as lesz), majd visszatér 0. oldalra az 1. sávra, stb. Láthatjuk, hogy a számozást cilinderenként végzi.

7.5.3 Logikai felépítés MSDOS-ban

Hajlékonylemez formázásakor a fizikai szerkezet létrehozása után a logikai szerkezet kialakítása történik. A logikai szerkezet függ az alkalmazott operációs rendszertől. Mi természetesen a továbbiakban az MSDOS által használt logikai szerkezetet ismerjük meg, ami a következő ábrán látható.

Betöltő szektor (BOOT szektor)
File elhelyezési táblázat (FAT)
Gyökérkönyvtár (ROOT directory)
Adatterület

- **Betöltő szektor (BOOT szektor)**
  Ez az a szektor, amely a betöltő programmal az operációs rendszert az operatív tárba tölti. A gép bekapcsolásakor a BIOS, a gép tesztelése után a rendszerlemezről betölti a 0. oldal 0. sávjának 1. szektorát (ez a betöltő szektor) a memóriába.

1.	Ugró utasítás	3 bájt
2.	A verzió száma	8 bájt
3.	A szektorok mérete bájtban	2 bájt
4.	A klaszterek mérete szektorokban	1 bájt
5.	A foglalt szektorok száma	2 bájt
6.	A FAT-ok száma	1 bájt
7.	A gyökérkönyvtár bejegyzéseinek a száma	2 bájt
8.	A szektorok száma a lemezen	2 bájt
9.	A lemez leíró bájtja	1 bájt
10.	A FAT egy példánya által elfoglalt szektorok száma	2 bájt
11.	A sávok mérete szektorokban	2 bájt
12.	A író-olvasó fejek száma	2 bájt
13.	A rejtett szektorok száma	2 bájt
14.	Nem használt	
15.	A betöltő program	

A betöltő szektor első 3 bájtja egy gépi kódú utasítást tartalmaz. Ez egy ugróutasítás, amely a betöltő programot indítja el. A következő 8 bájton a DOS verziószámát, vagy a formázó program nevét tartalmazza.
A harmadik rész az ún. *BIOS paraméter blokk (BPB)*, amely a lemez legfontosabb fizikai adatait tartalmazza.
Néhány mező jelentése részletesebben:

6. A FAT-nak lehetnek másolatai is, így ez a szám általában egynél nagyobb (többnyire 2).
7. Megadja, hogy a gyökérkönyvtárban maximum hány bejegyzés lehetséges.
8. Megadja, hogy összesen hány szektor lehet a lemezen, beleértve a BOOT szektort, a FAT-ot, a gyökérkönyvtárat, az adatterületet.
9. A DOS a különböző szabványos lemez formátumokhoz egy kódot rendel, az ún. *leíró bájt*ot. Ez például a HD-s lemeznél F9, a DD-s lemeznél FD. Ez a leíró bájt egyébként nem egyértelmű, előfordul, hogy különböző formátumhoz ugyanazt a leíró bájtot rendeli a DOS.
13. Ez a szám a hajlékonylemezeknél 0, a winchestereknél a BOOT szektor előtti szektorok számát adja meg.

- **Fájlelhelyezési táblázat (FAT)**
A FAT nyilvántartja, hogy az adatterület (ahol az állományok találhatók) mely szektorai szabadok illetve foglaltak, és hol találhatók az egyes fájlokat tartalmazó szektorok.
Sok esetben a fájl nem foglalhat el folytonos adatterületet.
Például: Ha egy létező állományt bővítünk, akkor lehetséges, hogy a szomszédos szektorokat már egy másik fájl foglalta el, és így a bővített fájl többi részét máshová kell elhelyezni.
Ha egy fájlt letörlünk, akkor a helye felszabadul, de ha egy új állományt viszünk a lemezre, akkor nem biztos, hogy a megürült helyen elfér.

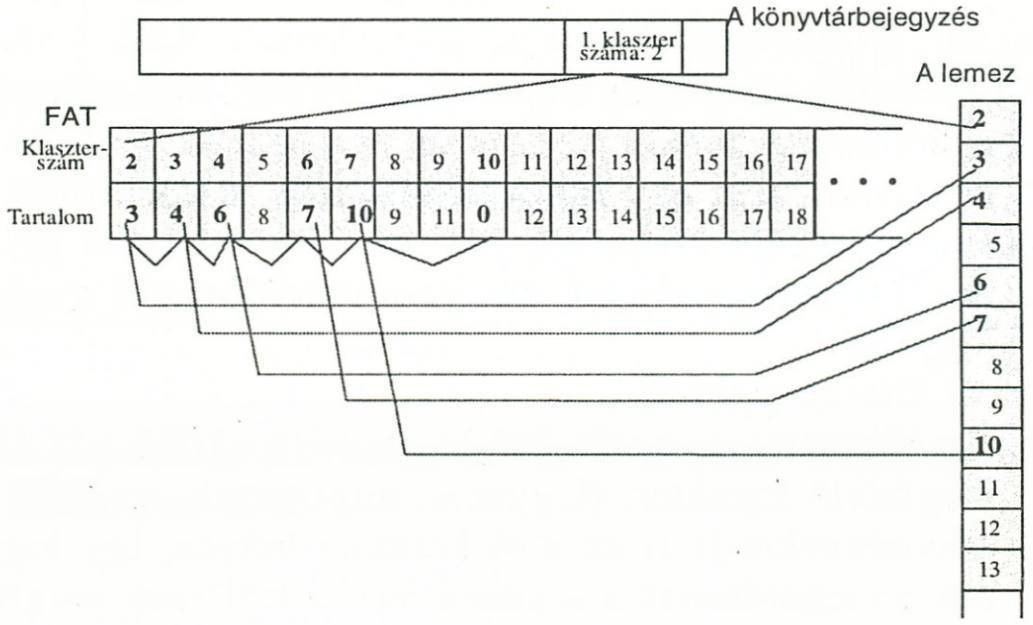
Ezeket a problémákat úgy lehet megoldani, hogy az állományokat részekre bontva tartjuk nyilván, egy táblázatban. Ez a táblázat a FAT. A részeket állomány elhelyezési egységnek, vagy *klaszternek (cluster)* nevezzük. A klasztert alkotó szektorok természetesen folytonos sorszámozásúak.

A klaszterbe foglalt szektorok száma a lemez kapacitásától függ elsősorban. Itt két különböző szempontot kell figyelembe venni; ha túlságosan sok szektor alkot egy klasztert, akkor ugyan gyorsabban lehet egy állomány részeit összeilleszteni, de ez gazdaságtalan helykihasználást eredményezhet, hiszen például egy 5 bájtos állomány is lefoglal egy teljes klasztert (ami lehet, hogy 2-4 Kbájt). viszont, csökkentjük a klaszter méretét, akkor a fájl több darabból áll, ami az összeillesztés idejét növeli. Meg kell tehát találni az optimális klaszter méretet, a lemez kapacitásától függően.

Megjegyzés:

1. *HD-s lemez esetén 1 klaszter 1 szektorból áll.*
2. *A lemezek a FAT-ot általában két példányban tárolják, biztonsági okokból.*
3. *A FAT által elfoglalt terület szintén az adott lemez formátumától függ. (A HD-s lemez esetén 2x7 szektor.)*

A FAT minden bejegyzése általában egy klaszterhez van rendelve. Ahhoz, hogy a DOS az állományt össze tudja illeszteni, a FAT alapján a fájl klasztereit össze kell láncolnia. Egy bejegyzés mindig az adott állomány következő klaszterének a számát adja meg, és egyúttal a következő FAT-bejegyzés helyét a táblázatban. Az állomány legelső klaszterét az állomány könyvtárbejegyzése tartalmazza. (Lásd később!)



Vannak speciális FAT-bejegyzések is:

- A klaszter szabadon felhasználható.
- A fájl végét jelző bejegyzés.
- Hibás klasztereket jelző bejegyzés.

Megjegyzés:

1. *A FAT tényleges bejegyzései kódoltan tartalmazzák a megfelelő klaszter sorszámokat.*
2. *Az adatterületet tehát klaszterekre osztjuk. A klaszterek számozása 2-vel kezdődik. Ennek az az oka, hogy a FAT első két bejegyzése speciális, más jellegű információkat tartalmaz.*

- **Gyökérkönyvtár (ROOT directory)**
  A gyökérkönyvtár közvetlenül a FAT után következik. A gyökérkönyvtár 32 bájtos bejegyzéseket, az állományokkal, alkönyvtárakkal, a kötetnévvel kapcsolatos információkat tartalmazza.

Név	Kiterjesztés	Attributum	Foglalt	A létrehozás, ill. az utolsó módosítás időpontja	A létrehozás, ill. az utolsó módosítás dátuma	Az állmány 1. klasztere	Az állmány hossza
8 byte	3 byte	1 byte	10 byte	2 byte	2 byte	2 byte	4 byte

- **Név**
  Ha az állomány neve 8 karakternél kisebb, akkor az üres helyekre szóköz kerül.
  Ha a bejegyzés nem egy fájlra vonatkozik, akkor a név első karaktere egy speciális kód:
  0H Az illető bejegyzéshely még nem tartalmazott tényleges bejegyzést.
  E5H Az állomány törölt.
  05H Az állomány nevének első karaktere az E5H ASCII-kódnak megfelelő karakter. (Ugyanis, mint láttuk, az **E5H** a törölt állományokat jelzi az első karakter helyén, így foglalt. Ennek a megoldásnak a segítségével az E5H-nak megfelelő karakter is használható az állomány nevében.)

2EH Az aktuális könyvtárat jelzi. Ha a második karakter is 2EH, akkor az aktuális könyvtárat tartalmazó könyvtárra vonatkozik. (Lásd később!)

- **Kiterjesztés**
  Az állomány kiterjesztése, szóközökkel kiegészítve, ha szükséges.

- **Attribútum**
  Az attribútum bájtértéke az állomány tulajdonságaival kapcsolatos. Az egyes bitek jelentése a következő táblázatban látható.

7	6	5	4	3	2	1	0	Jelentése
							1	Csak olvasható
						1		Rejtett
					1			Rendszer
				1				Kötetcímke
			1					Alkönyvtár
		1						Archivált
	1							Nem használt
1								Nem használt

Megjegyzés:

1. *A 0., 1., 2., 5. bitek használhatók egyszerre is. A 3., 4., 5. bitek nem használhatók egyszerre.*
2. *Az 5. bit értékéről: A fájl az utolsó megváltoztatása óta le lett zárva (ha igen, akkor az értéke 1). A bitet BACKUP és RESTORE parancsok használják, ez alapján lehet eldönteni, hogy az utolsó mentés óta megváltozott-e a fájl.*
3. *Ha kötetcímkéről van szó, akkor az első két mező összevonható, így a kötetcímke összesen 8+3=11 karakteres lehet.*

A létrehozás illetve az utolsó módosítás időpontja és dátuma kódoltan van tárolva.

- **Az állomány 1. klasztere**
  Itt annak a klaszternek a száma áll, amelyen az adott állomány kezdődik. Ehhez a klaszterhez tartozik természetesen egy FAT-bejegyzés, amely a következő klaszternek a számát tartalmazza, ehhez a klaszterhez is tartozik egy FAT-bejegyzés stb.

- **Alkönyvtárak**
  A gyökérkönyvtárban levő bejegyzések háromfélék lehet:

  - fájl
  - kötetcímke
  - alkönyvtár.

  Az alkönyvtárak természetesen szintén tartalmazhatnak fájlokat illetve további alkönyvtárakat. Azt a könyvtárat, amelyben egy másik alkönyvtárra vonatkozó bejegyzés található, „szülőkönyvtárnak", a benne levő alkönyvtára „gyermekkönyvtárnak" is nevezik.
  Ezeknek az alkönyvtárakra vonatkozó bejegyzéseknek a formája, felépítése megegyezik az állományokra vonatkozó bejegyzésekével, de az attribútum bájt 4. bitje jelzi, hogy alkönyvtárról van szó: a hosszát megadó bájtok értéke 0, az első klaszter számát megadó bájtokban annak a klaszternek a száma áll, ahol a gyermekkönyvtár található. (Természetesen az adatterületen.)

  Amikor létrehozunk egy alkönyvtára, a DOS két különleges bejegyzést helyez el benne:

  1. „.": A kezdő klaszterszám ennek a klaszternek a száma. (Tulajdonképpen ez a szülőkönyvtárnak erre az alkönyvtárra vonatkozó bejegyzésmásolata, kivéve a nevet.)

2. „..": A kezdő klaszter száma annak a klaszternek a számát adja meg, ahol a szülőkönyvtár van. Ha ez a klaszterszám 0, akkor a szülőkönyvtár a gyökérkönyvtár.

7.5.4 A logikai felépítés Windows 95-ben

Az egyes operációs rendszerek általában más-más *fájlrendszert* használnak, azaz különböző a logikai felépítés. A következőkben a *Windows 95* fájlrendszerének – MSDOS-tól különböző – jellemzőit nézzük meg. (A *Linux fájlrendszerének* jellegzetességeit a **Linux** című fejezetben ismerheted meg.)

- **VFAT** *(Virtual FAT)*
  A leglényegesebb eltérés a FAT felépítésében mutatkozik. A cél az volt, hogy az MSDOS által használt fájlnevekkel kompatibilis legyen. Ezért két fájlnevet rendel minden állományhoz:
  - Hosszú, maximum 255 karakteres név.
  - Rövid, 8+3 karakteres, *MSDOS* fájlnév, ami a hosszú név csonkításával jön létre. Ha DOS alkalmazást futtatunk Windows alatt, akkor az tönkreteheti a hosszú fájlnevet, és helyette csak a rövid marad meg.

  A FAT bejegyzések továbbra is 16 bitesek, ami korlátozza a használható partíciók méretét 2 GB-ban : 1 MB fölött a klaszter méretének már 32 KB -nak kell lennie, a 16 bites FAT bejegyzés maximális értéke kb. 2^{16} bájt. A kettőt összeszorozva 2 Gbájtot kapunk (2^{16} x 2^{5} x 2^{10} bájt = 2^{31} bájt= 2 x 2^{30} bájt = 2 GB), mint felső határt.

- **FAT32**
  A *Windows 95* javított változatában (*OSR2*) és a *Windows 98*-ban már a FAT32 fájlrendszert találjuk. Itt is használhatunk hosszú fájl neveket. A FAT bejegyzések 32 bitesek, ami lényegesen kibővítik a használható partíció méretet. A legnagyobb klaszterméret itt is 32 KB, de ezt csak 32 GB fölött használja, 1 és 2 GB között csak 4KB.

7.5.5 Winchesterek

A winchesterek felépítése sok mindenben különbözik a hajlékonylemezes egységektől. Általában több lemezt is tartalmaznak. A lemezek mérete általában 5,25", illetve 3,5". (Ma már vannak ennél kisebbek is.) A lemezek nem cserélhetők a hagyományos típusok esetén. A lemezek alumíniumból készülnek, amelyek mágnesezhető anyaggal vannak bevonva. A lemezek legalább 3600 fordulat/perc fordulatszámmal forognak, mely a gép bekapcsolásától a kikapcsolásig, folyamatosan.

Az író-olvasó fej itt sem érintkezik közvetlenül a lemez felületével, de a hajlékonylemezes egységekkel összehasonlítva lényegesen közelebb helyezkedik el. A fej 0,3..0,5 μm magasságban mozog a lemez fölött. A felhajtóerőt a lemez felületén, a gyors forgás következtében létrejövő légpárna adja. Egy porszem mérete 1 μm-nél nagyobb, így ha egy porszem szorulna be a fej és a lemez közé, az megkarcolná a lemez felületét, és így az a terület használhatatlanná válna. Ezért zártak a winchesterek, ezzel akadályozzák meg, hogy szennyeződés jusson be a készülékbe. A légpárna csak a lemezek forgása után alakul ki, mert a lemez forgásának indulásakor illetve leállásakor a fej hozzáérne a lemezhez, és az információk megsérülnének. Ennek elkerülésére a fejeket olyan területre viszik, ahol nincsenek adatok. A mai winchestereknél ez a gép kikapcsolásakor automatikusan megtörténik.

A winchesterek tárolási kapacitása lényegesen nagyobb mint a hajlékonylemezeké, és ráadásul sokkal gyorsabban is érik el a kívánt adatokat, mint azok. A mai winchesterek átlagos elérési ideje kb. 10 ms.

- **Fizikai felépítés**
  A winchesterek fizikai felépítése megegyezik a hajlékonylemezekével, a különbség csak ott van – mint láttuk –, hogy több lemezből állnak. Így az oldalak száma is több.

  Például: Egy két lemezt tartalmazó winchester esetén az oldalak számozása: 0, 1, 2, 3.

Megjegyzés:

A hajlékonylemez formázásakor a fizikai és logikai szerkezet is kialakításra kerül. A winchesterek esetén csak a logikai szerkezet épül ki,

a fizikai szerkezet kialakítása egy alacsonyszintű formázásnak nevezett folyamattal történik, a gyártó cégnél. (Igaz, speciális programmal a felhasználó is elvégezheti ezt, de ezzel óvatosnak kell lenni egyes winchestertípusok esetén, mert ezek gyártói a „házilagosan" végzett alacsonyszintű formázás után nem vállalnak garanciát rá.)

- **Logikai felépítés**
  A logikai felépítés is hasonlatos a hajlékonylemezekéhez, de van egy alapvető különbség: a winchestereket részekre, ún. partíciókra kell osztani.

Partícionálás:

A partícionálás a teljes lemezterület részekre osztását jelenti. Előfordulhat, hogy ugyanazon a winchesteren több operációs rendszer is helyet kap. (Például: MSDOS és UNIX.) Az egyes operációs rendszerek különböző logikai felépítést használnak. A partícionálással lehetőséget adunk arra a különböző operációs rendszereknek, hogy az egyes partíciókban létrehozzák saját logikai szerkezeteiket. A partícionálás után az egyes partíciókat formázni kell az adott operációs rendszer saját formázási parancsával. Természetesen lehetőség van arra is, hogy az egész lemezt egy partíciónak adjunk meg, vagy hogy több partíciót is ugyanaz az operációs rendszer használjon.

A partícionálást az MSDOS-ban, Windows 95-ben az FDISK paranccsal végezhetjük el. Ennek a segítségével létrehozhatunk, törölhetünk partíciót. A partícionálás során (FDISK) meg kell adnunk az ún. *aktív partíciót.* Az aktív partícióval azt adjuk meg, hogy a partíciók közül melyik „viselkedik" rendszerlemezként a rendszer indulásakor. (Természetesen ahhoz, hogy a partíció rendszerlemezként működhessen, még más feltételeknek is teljesülni kell.) Ha egyetlen partíció az egész lemez, az aktív partíciót akkor is ki kell jelölni. A lemezt maximum 4 ún. elsődleges *(primary)* partícióra oszthatjuk. Minden elsődleges partíción egy további kiterjesztett *(extended)* partíciót hozhatunk létre, amelyeket pedig további logikai *(logical)* partíciókra oszthatjuk. (MSDOS-ban csak egy elsődleges partíciót hozhatunk létre.)

Megjegyzés:

Ha több operációs rendszer is van a merevlemezen, akkor az aktív partíció állításával lehet közöttük „választani", azaz, hogy a gép bekapcsolásakor melyik operációs rendszer induljon el.

A merevlemez 0. oldal, 0. sáv, 1. szektorának az elején egy, az operációs rendszerektől független program található, amely az aktív partíció indítását végzi. Ezután (még mindig ezen a szektoron) a partíciókra vonatkozó adatok helyezkednek el, az ún. *partíciós táblában.* A partíciós tábla tartalmazza az egyes partíciók elhelyezkedését, és az aktív partíciót is jelzi.

Az MSDOS által használt partíció logikai felépítése teljes mértékben megfelel a hajlékonylemezeknél látottakkal. (BOOT szektor, FAT, gyökérkönyvtár, adatterület.)

7.5.6 CD-ROM (*Compact Disc-ROM*)

Manapság egyre elterjedtebbé váló adattároló. Az adatok itt is egy kör alakú, 5,25" átmérőjő műanyag lapon helyezkednek el. Az adatok kiolvasása lézerfény segítségével történik.

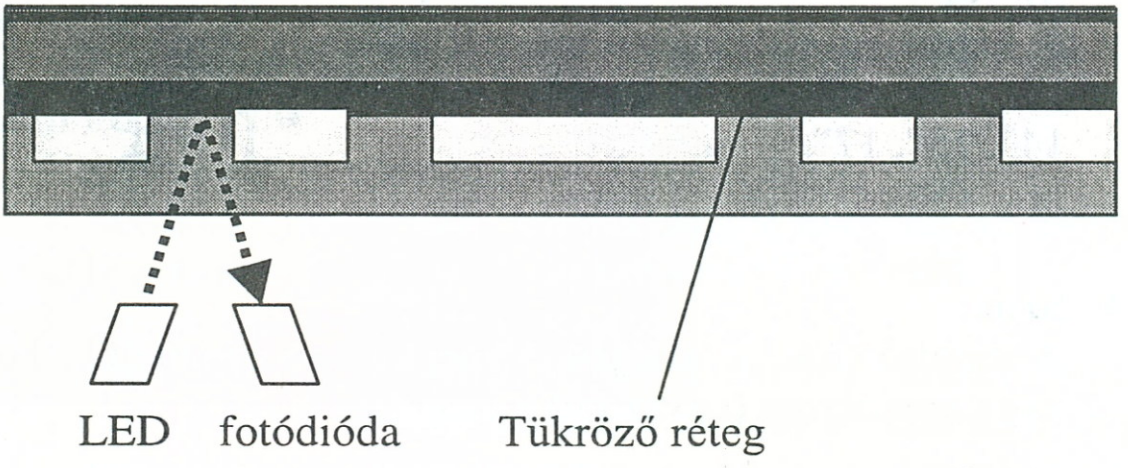
Kétféleképpen lehet a lemezre írni:

- Préseléssel, hasonlóképpen, ahogyan régebben a bakelit lemezeket gyártották. Elsőként egy mesterlemezt készítenek, és ez alapján préselik a többit.
- A lézerfény segítségével mikroszkopikus lyukakat (piteket) égetnek a lemezbe.

Olvasáskor egy kis intenzitású lézersugár letapogatja a CD felszínét, és a visszavert fényt vizsgálja. Az éppen hagyott felületről több fény verődik vissza, mint a lyukról. Az első esetben 0, a másodikban 1 lesz a kiolvasott adat értéke. A lyukak egy spirális pálya mentén helyezkednek el. Két szomszédos sávjának (csíkjának) a távolsága 1,6 μm. A spirál 2 Kbájtos szektorokra van felosztva. A fej és a lemez egymáshoz viszonyított sebessége végig állandó, így a spirál külső részén kisebb a fordulatszám, mint a belső részen. Az átviteli sebesség kezdetben 150 KB/s volt (tehát a lemezről a memóriába beolvasásnak a sebessége), ezt nevezik egyszeres sebességűnek.

A kétszeres sebességű CD-ROM 300 KB/s átviteli sebességű, a négyszeres esetén 600 KB/s, stb. A CD-ROM lemezek kapacitása kb. 650 MB.

CD-R-nek a CD-íróval (lézerrel) megírt lemezeket, CD-RW-nek pedig a többször is írható CD lemezeket hívják.



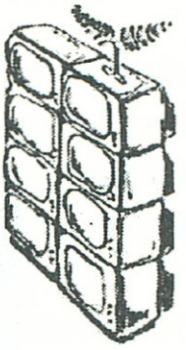
Megjegyzés:

LED:	*Világító dióda. Félvezető anyagból készült, áram hatására fényt bocsát ki, az áram erősségével arányosan.*
Fotódióda:	*Fény hatására áram keletkezik benne, a fény erősségével arányosan. Ez is félvezető anyagból készült.*

Újabban az ún. **DVD** ***(Digital Versatile Disc)*** kezdi felváltani a CD-t. Nagyobb adatsűrűség, kisebb pit méret, sűrűbben elhelyezkedő sávok jellemzik, így a kapacitása 4,7 GB. A DVD-nek létezik egyoldalas kétrétegű (8,5 GB), ill. kétoldalas kétrétegű (17,4 GB) változata is.

7.6 Monitorok

A monitorokat több szempont szerint is osztályozhatjuk:

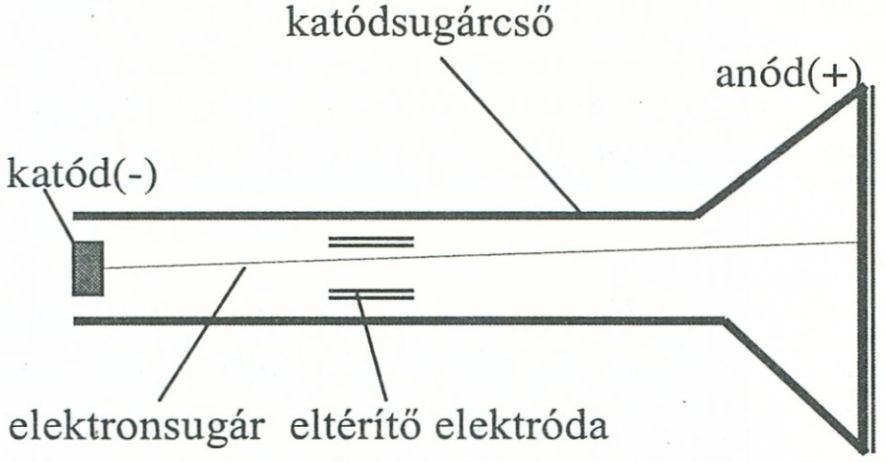


- **Méret**
  A képernyő átlójának a méretét adják meg collban. Jellemző értékek: 14", 15", 17".

- **Szín**
  A monitorokat az alapján is megkülönböztethetjük, hogy színes, vagy *monokróm* (egyszínű) a megjelenítés.

- **Működési elv**

  - **Katódsugárcsöves**
    Működési elve hasonlít a közönséges TV készülékekéhez.



  - **Folyadékkristályos**
    Bizonyos folyadékok elektromos impulzusok hatására megváltoztatják a fényáteresztő képességüket. Elsősorban hordozható számítógépekben alkalmazzák.

➢ **Gázplazmás**
Minden képpont mögött pici gázzal teli kamrák helyezkednek el. A gáz elektromos feszültség hatására fényt bocsát ki. Elsősorban nagyméretű kivetítőkben használják.

- **Képpont méret**
Általában minél kisebb ez az érték, annál „élvezhetőbb" a monitor képe. Átlagos értéke 0,26 mm – 0,28 mm.

- **Átlapoltság**
1. Nem átlapolt üzemmód *(Non-Interlaced)*
A sorokat egymás után jeleníti meg a monitor, az utolsó sor után a katódsugár a bal felső sarokban újra kezdi.
2. Átlapolt üzemmód *(Interlaced)*
A teljes kép két lépésben áll elő. Először a páratlan sorszámú sorok jelennek meg, utána pedig a páros sorszámúak. A gyors megjelenítés miatt természetesen egy képet látunk, az előzőhöz képest rosszabb minőségben (kissé vibrál a kép), és hosszabb távon fárasztja a szemet. Az üzemmódok között automatikusan vált a monitor.

- **Felbontás**
A képernyőn a pontok sorok és oszlopok mentén helyezkednek el. A sorok és oszlopok száma függ az adott típustól. Felbontás alatt a sorok és oszlopok számának szorzatát értjük.
(A karakterek meghatározott számú pontból állnak.)

Néhány monitor esetén a felbontás:

Típus	Felbontás	Színek
CGA	320x200	4
Hercules	720 x 348	Monochrom
EGA	640 x 350	16
VGA	640 x 480	16

Természetesen vannak monitorok, amelyek az előző üzemmódok mindegyikét ismerik, sőt ennél még többet is. Például: ún. *grafikus üzemmódban* 640 x 480-as felbontás 256 szín mellett, vagy

1024 x 768-as felbontás 256 szín mellett (ún. SVGA *(Super VGA)* monitorok), stb.
Az ún. *szöveges üzemmód* esetén a képernyőn karakterek jeleníthetők meg. Jellemző felbontás: 80 x 25. EGA és VGA monitorok esetén lehetőség van a 80 x 43 illetve 80 x 50-es felbontásra is.
Egy adott monitor általában többfajta üzemmódban képes működni, amelyet a monitor vezérlőkártyája (videokártya) határoz meg. A legtöbb VGA videokártya például a CGA üzemmódtól kezdve akár a 1024x768-as SVGA felbontásig képes vezérelni egy SVGA monitort. A felbontás és a használt színek száma függ a vezérlőkártyán levő memória nagyságától is.

- **Színmélység**
  Grafikus üzemmódban ha minden képponthoz n bitet rendelünk, akkor az összesen 2^n különböző színt vehet fel. Az n számot nevezzük színmélységnek. A kép minőségét a felbontás és a színmélység növelésével is javíthatjuk.
  A teljes képernyő tartalmának a tárolásához szükséges helyet a következőképpen kell kiszámítani:

$$Hely = felbontás \cdot színmélység$$

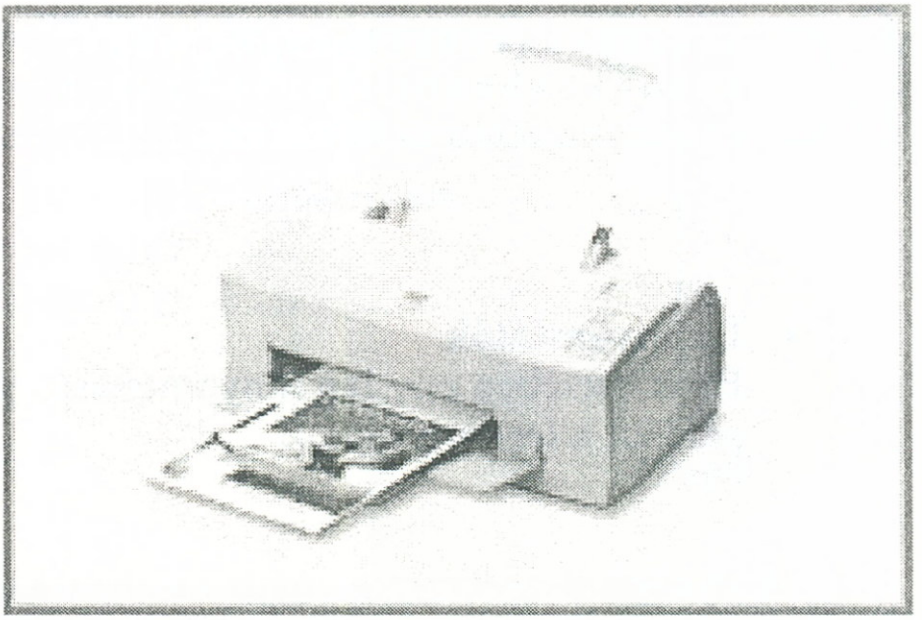
Például:

Hely =800 · 600 · 24 bit = 11 520 000 bit = 1 440 000 bájt = 1,373 MB

Néhány jellemző SVGA felbontás:

Felbontás	Színmélység	Színek	Memória (MB)
640 x 480	8	256	0,293
640 x 480	15	32768	0,549
640 x 480	24	16,7 millió	0,879
800 x 600	8	256	0,458
800 x 600	15	32768	0,858
800 x 600	24	16,7 millió	1,373
1024 x 768	24	16,7 millió	2,25
1280 x 1024	24	16,7 millió	5,493

7.7 Nyomtatók



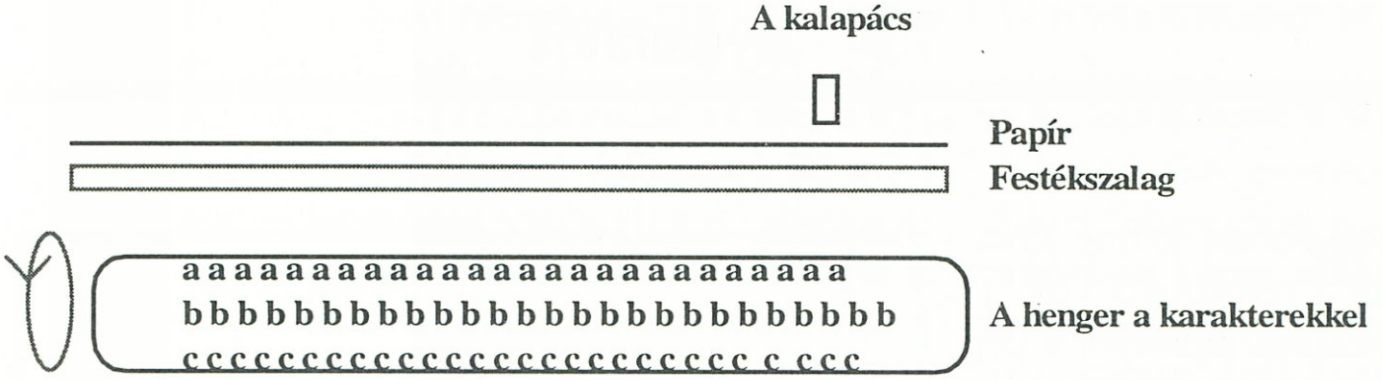
Régebben szinte kizárólag a nyomtató volt az az eszköz, amellyel a gép az ember számára közvetlenül érthető üzenetet tudott küldeni. Amíg a program nem volt kész, a teszteléseknél kapott rengeteg papír a szemétbe került. De ezt ma már teljes mértékben pótolja a képernyős megjelenítő. Természetesen ma is rengeteg olyan terület van, amelyeken nem lehet a nyomtatókat helyettesíteni: újságok, hivatalos értesítések, jelentések stb.

A nyomtatókat a következők szerint osztályozhatjuk:

1. *Egyszerre mennyit nyomtat?*

   - **Szeriális nyomtató**
     Egyszerre egy karaktert nyomtat, majd továbbítja a nyomtatómechanizmust a következő pozícióra. Ilyenek a *mátrix* illetve *tintasugaras nyomtatók* is.

   - **Sornyomtató**
     Először a memóriájában összegyűjt egy egész sorra való adatot, és azt egyszerre nyomtatja ki. Ezek a nyomtatók általában viszonylag nagy sebességgel nyomtatnak, de nem túlságosan jó minőségűek. Ilyenek az ún. *hengeres* és *láncos nyomtatók*. Ezeknél egy hengerre illetve láncra vannak rögzítve a karakterek, és egy kis kalapács a megfelelő pillanatban megüti a papírt, amelyre a festékszalag segítségével rákerül a karakter. A hengeres nyomtató esetén az ábrának megfelelően helyezkedik el a henger. A hengeren egyébként egy adott helyen, a kerületen a teljes jelkészlet megtalálható.

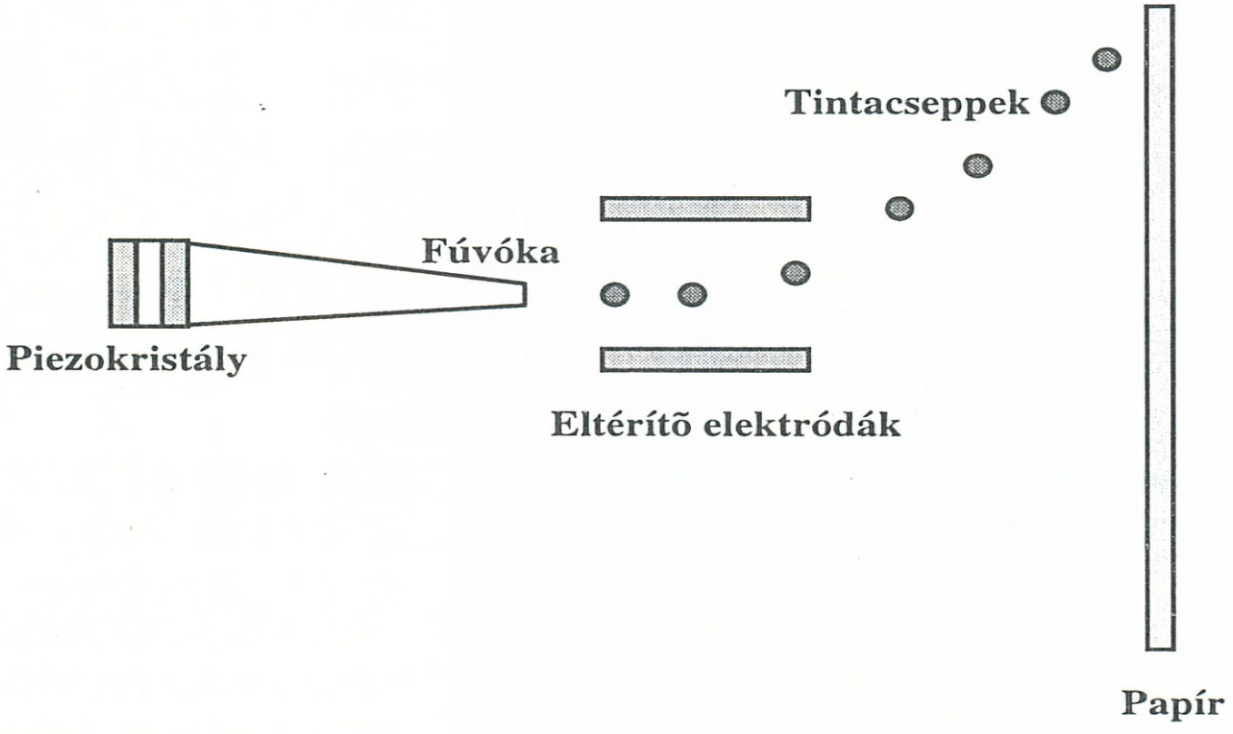


- **Lapnyomtatók**
  Ezek esetében egy lap kinyomtatása egy menetben történik. Ilyen például a *lézernyomtató*.

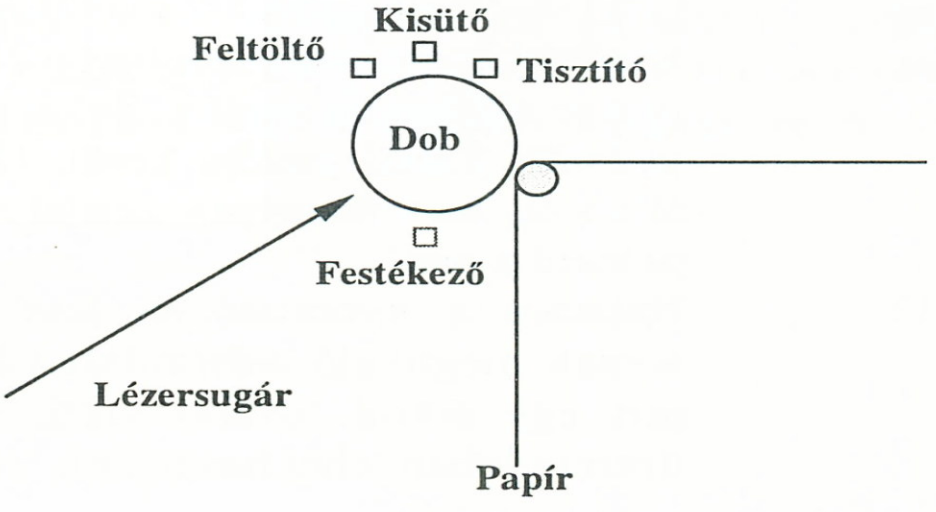
2. *A nyomtatószerkezet a tintát mechanikai érintkezéssel viszi-e fel a papírra?*

- **Érintéses nyomtató**
  A festékszalagon keresztül adott erejű fizikai érintkezést hoz létre a nyomtatandó karaktert tartalmazó szerkezet és a papír között. Az érintéses nyomtatókat már messziről hallani lehet működés közben, amely egy kisebb irodában nagyon zavaró lehet. Ilyenek a *mátrixnyomtatók*, vagy az előzőekben megismert *hengeres* és *láncos nyomtatók*.

- **Érintés nélküli nyomtató**
  Mechanikai érintkezés nélküli például a **tintasugaras** *nyomtató*. Ezeknél a berendezés tintacseppeket „lő" a papírra, amelyek pontszerű nyomot hagynak rajta. Ezekből a pontokból a leütéses mátrixnyomtatókhoz hasonlóan lehet az egyes karaktereket előállítani.
  A tintacseppeket egy piezo kristályból készült fúvóka segítségével lövik ki a papírra. A piezo kristály elektromos áram hatására megváltoztatja az alakját, így megváltozik a nyomás a fúvóka belsejében.
  A kilőtt tintacseppek elektromosan feltöltődnek a porlasztás során, és így az elektródák között eltéríthetők.



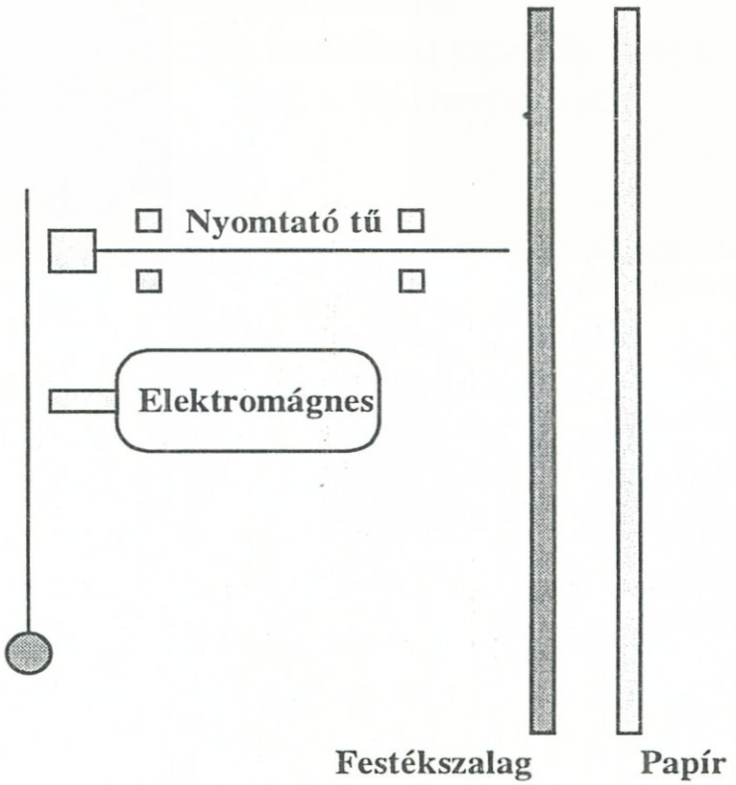
Az érintés nélküli nyomtatók másik elterjedt típusa a **lézernyomtató.** Egy fényérzékeny, töltéstároló képességű félvezető anyagú bevonattal ellátott hengert 1000 V-ra feltöltenek. A forgó henger felületét az alkotói mentén végigpásztázza egy lézersugár, a nyomtatandó képnek megfelelően. Ahol a fénysugár éri a hengert, ott az erősségének megfelelően a henger felülete elveszti a töltését. A forgó henger felülete elhalad a festékport tartalmazó tartály előtt. A festékpor azonos töltésű a hengerrel, ezért csak azokra a pontokra tapad, ahol nincs töltés (nincs taszítás). Így létrejön a henger felületén a kívánt kép. Ezután a papírra viszik a képet, majd forró hengerek között vezetik át a papírt, és a pontokon levő festék „ráég" a papírra.



3. *Az egyes karaktereket egyben, van pontokból állítja-e elő?*

- **Karakternyomtató**
  A karakterek tükörképét teljes egészében megtalálhatjuk egy hengeren vagy gömbön. A papírral mindig az egész karakter kerül érintkezésbe, ennek következtében a kapott kép elég jó minőségű. Hátránya, hogy a karakterkészlet rögzített a mechanikus beépítettség miatt, és viszonylag kevés a rendelkezésre álló karakter.

- **Mátrixnyomtató**
  Kis pontokból rakja össze a karaktereket. A pontokat általában 9 vagy 24, egymás alatt és mellett elhelyezett tű hozza létre, amelyek együtt mozognak a papír előtt, vízszintes irányban. A megfelelő helyen a megfelelő tűk kiugranak, és a festékszalagon keresztül festéknyomot hagynak a papíron. Nagy előnye a mátrixnyomtatóknak, hogy nincsen kötött jelkészlete, így gyakorlatilag bármilyen jel nyomtatására alkalmas, akár rajzok készítésére is.



A legtöbb mátrixnyomtatón megtalálhatók a következő „gombok”:

ON LINE: Ehhez általában tartozik egy világítódióda, egy LED is. Segítségével a számítógép és a nyomtató közötti kapcsolatot szakíthatjuk meg (OFF LINE), illetve állíthatjuk vissza. Bekapcsoláskor a nyomtató ON LINE módba kerül, OFF LINE lesz az állapota, ha valamilyen akadály merül fel, kifogy például a papír.

LF: Hatására a nyomtatóban levő gumihenger egy sornak megfelelő elfordulást végez, vagyis a papírt egy sorral tovább viszi. Csak OFF LINE üzemmódban lehet használni.

FF: Csak OFF LINE üzemmódban használhatjuk. A nyomtató egy lapnyit továbbítja a papírt.

DRAFT: Durva felbontású üzemmód. Minden sort csak egyszer nyomtat ki.

NLQ: Közel levél minőségű nyomtatást hozhatunk létre a segítségével. Minden sort kétszer nyomtat végig ilyenkor a nyomtató, csak másodjára kismértékben eltolva halad az írófej. Így a pontok kissé összemosódnak, jobb lesz a szöveg minősége.

CONDENSED: Sűrített írásmódot jelent, az eredetinél keskenyebb lesz a szöveg, és a karakterek mérete is kisebb lesz.

A papírtovábbító mechanizmus típusonként eltérő. Egy jellemző megoldás a perforált (a két szélen található lyukakba kétoldalt fogas-kerekek kapaszkodnak, és ezeknek a kerekeknek a forgása továbbítja a papírt, megakadályozva annak csúszását) leporelló papír. A nyomtatók másik részénél a papírtovábbítás egy gumihenger segítségével történik.

A nyomtatók által előállított kép minőségét az egységnyi távolságra jutó képpontok számával szokták jellemezni. (mértékegység: *dpi* = *dot/inch*, azaz pont/inch) A lézernyomtatók esetén például kb. 600-1200 dpi.

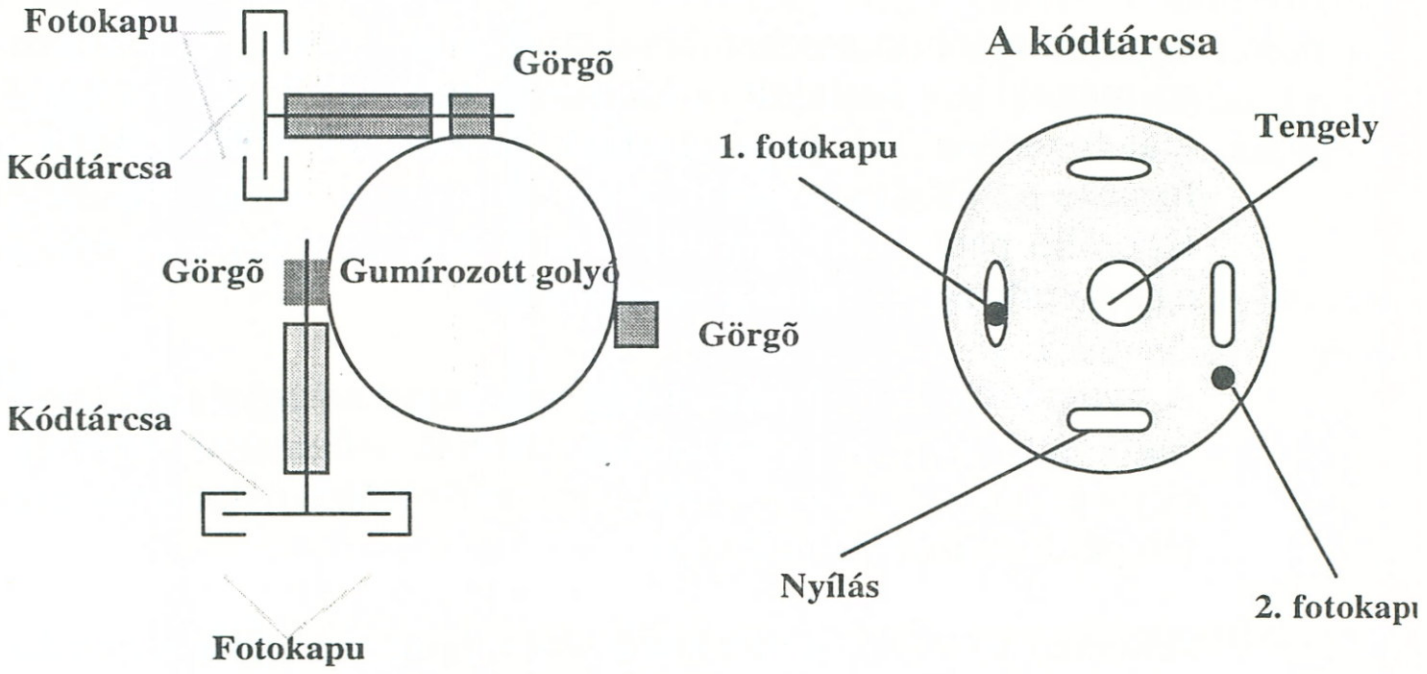
Jellemző még a nyomtatókra a sebességük. Az egyik a *cps*: *karakter/sec*, ezzel jellemezhetjük például a szeriális nyomtatókat. A másik *lpm*: *sor/perc*, például a sornyomtatók esetén kb. 1000 lpm. Lapnyomtatók esetén: lap/perc: ppm.

7.8 Az egér

Az egér egy speciális adatbeviteli eszköz. Az egeret az asztalon kell mozgatnunk, és ennek hatására a képernyőn egy ún. *egérkurzor* (pl.: egy nyíl) mozog. Három alapvető típusa létezik:

- **Mechanikus**
  Két, egymásra merőleges görgőrendszer mozgását érzékeli.

- **Opto-mechanikus**



A gumírozott golyó két görgő segítségével hajtja a kódtárcsákat. A közvetítő görgők elhelyezése olyan, hogy általuk a függőleges és vízszintes irányú mozgás szétválasztható. A kódtárcsa egy réseket tartalmazó műanyag korong. A rések száma általában 36, de lehet ennél több is. (A rajzon csak négy rés látható, az egyszerűség kedvéért.)
A fotókapu érzékeli, ha a kódtárcsa elfordul előtte, hiszen amikor éppen a rés van előtte, akkor a fény átjut a tárcsán, egyébként

nem. (A fotókapu egyik oldalán egy *világító dióda* van, egy ún. LED, amellyel szemben a tárcsa túloldalán egy fényérzékeny alkatrész található, például egy fotótranzisztor.) Azért van két fotókapu egy tárcsához, mert így lehet érzékelni a tárcsa forgási irányát. A két fotókapu elhelyezkedését az ábrán láthatjuk.

- **Optikai**
  Egy finom rajzolatú négyzetrácsos egér alátéten mozog az egér. Egy LED fényének a visszaverődését érzékeli a beépített fotódióda, ami alapján érzékeli a mozgást.

Megjegyzés:

A hordozható gépekben több „szabálytalan" egeret is használnak. Az egyik típusnál a golyót az ujjainkkal közvetlenül mozgathatjuk („hanyattegér"). A másik típusnál már nincs is golyó, az ujjainkkal egy téglalap alakú tartományt „ingerelhetünk", aminek megfelelően fog az egérkurzor mozogni („tapipad").

8. ADATÁLLOMÁNYOK

Az *adatállomány* vagy *fájl* a számítástechnika egyik központi fogalma, szinte minden programozási nyelvben fontos szerepet játszik. Az állományokban adatokat tárolunk valamilyen háttértáron.

8.1 Logikai fájl

A programban egy ún. *logikai állományt* kezelünk, amelynek a szerkezetét, tulajdonságait mi definiáljuk. Általában nem kell foglalkoznunk azzal, hogy az állomány ténylegesen hogyan kerül tárolásra a háttértáron, számunkra csak a logikai állomány létezik.

Mezőnek nevezzük azt a legkisebb adatrészt a fájlban, amelynek saját neve van, és valamilyen tulajdonságot ír le.
Például: név, kor, hosszúság, ár, stb.

A **rekord** pedig azoknak a mezőknek az együttese, amelyek egy adott *egyedhez* tartoznak.
Például:

1. Egy személyi nyilvántartásban egy dolgozót mint egyedet jellemezhetünk a következőkkel: neve, lakcíme, telefonszáma és beosztása.
   A NÉV, LAKCÍM, TELEFON, BEOSZTÁS a logikai rekord egy-egy mezője.

2. Egy könyvtári nyilvántartásban egy könyv, mint egyed jellemzői lehetnek a következők: cím, szerző, kiadó, raktári szám.
   Itt a logikai rekord mezőinek a nevei lehetnek a következők például: CÍM, SZERZŐ, KIADÓ, RAKSZÁM.

Azt a mezőt, amelynek az értéke minden rekordban más és más, tehát *egyedi*, az adott rekord *azonosítójának* nevezzük. Szokták ezt az azonosítót *elsődleges kulcsnak* is nevezni. Egy személyi nyilvántartásban például nem tekinthető egyedi azonosítónak a dolgozó neve, hiszen könnyen előfordulhat, hogy több azonos nevű dolgozó is található egy munkahelyen. Elsődleges kulcs lehet például a dolgozó személyi száma (bár ma már ez elvileg nem használható), vagy valami hasonló numerikus azonosító. A többi azonosítót *másodlagos kulcs*nak szokták nevezni.

A rekordokat hosszúságuk szerint a következőképpen vizsgálhatjuk:

- **Fix hosszúságú**
  Az egyes mezők hossza rögzített, sorrendjük is kötött a rekordon belül.

- **Változó hosszúságú**
  Tartalmazhat változó hosszúságú mezőket is, vagy lehet olyan mezője, amely többször is előfordulhat. Például: a *GYEREK NEVE* egy személyi nyilvántartásban többször is szerepelhet egy személy (egyed) adatai között. A mezők sorrendje kötött.

- **Határozatlan hosszúságú**
  A mezők egy része akár ki is maradhat egy rekordból, és ráadásul a sorrendjük sem meghatározott.

8.2 Fizikai fájl

A fizikai fájl az adott periférián ténylegesen megjelenő adatokat jelenti. A logikai fájlt a háttértáron a fizikai fájlban tároljuk.

A fizikai fájlt többféleképpen is jellemezhetjük:

- Milyen háttértárat használunk (szalagos egység, lemezegység)?
- Az adott adathordozón hol helyezkedik el a fizikai állomány (pl. a lemezegységnél oldal, sáv, szektor)?

A fizikai állomány alapegysége a fizikai rekord. A fizikai rekord nem feltétlenül fedi le a logikai rekordot. A kettő közötti kapcsolatot a blokk fogalmának a segítségével érthetjük meg.

Blokk

A logikai fájlnak helyet foglalunk az adathordozón. Az elhelyezett fájl lesz a fizikai fájl. Az elhelyezés *alapegysége* a *blokk*. Egy blokkban kerül tárolásra a fizikai állomány egy fizikai rekordja. A fizikai rekord nem feltétlenül felel meg egy logikai rekordnak, előfordulhat, hogy egy blokkban több logikai rekord kerül tárolásra.

A blokknak tehát kettős szerepe van:

1. Az információ *tárolása* a periférián blokkokban történik. Az információk blokkokban vannak összefogva.
2. A blokk úgy is értelmezhető, mint az *egyidejűleg, egyszerre átvitt adatnak a mennyisége* a háttértár felé.

A blokk és a logikai rekord között a következő kapcsolatok lehetségesek:

- Egy rekordhoz egy blokk tartozik.
- Egy rekordhoz több blokk tartozik.
- Egy blokkban több rekord helyezkedik el.

Megjegyzés:

A legegyszerűbb az 1 rekord - 1 blokk megfeleltetés. Általában elegendő azonban csak a logikai fájlra gondolnunk, nem fontos ezt a megfeleltetést tudnunk.

8.2.1 Pufferelés

A puffer a tárnak egy olyan speciális területe, ahol az információ az *adatátvitel* során ideiglenesen tartózkodik. A pufferelés arra szolgál, hogy kiegyenlítse a CPU és a periféria közötti sebesség különbségét, ugyanis a CPU lényegesen gyorsabb, mint a háttértár, és amíg egy rekord kikerül a háttértárra, addig a CPU vár. Amikor feldolgozunk egy rekordot, nem biztos, hogy utána az adott rekord azonnal kikerül a háttértárra, ez csak akkor fog megtörténni, ha megtelik a puffer, azaz összegyűlik annyi logikai rekord, mint amekkora a puffer mérete. Így a CPU-nak csak a puffer ürítésekor kell tétlennek lennie.

A pufferelésnek van egy hátránya: előfordulhat, hogy nem telik meg, és valamilyen műszaki hiba lép fel (pl. áramszünet), ekkor a puffer tartalma elvész, ami nyilván zavart okozhat az adatfeldolgozásban.

8.3 Műveletek fájlokkal

A következőkben az alapvető, állományokkal kapcsolatos műveleteket soroljuk fel. Természetesen az egyes állománytípusoknál ezeknek a műveleteknek a megvalósítása elég változatosak lehetnek.

1. **Létrehozás**

2. **Megnyitás**
   Bármilyen fájlt, amellyel dolgozni akarunk, meg kell nyitni. A megnyitás tulajdonképpen azt jelenti, hogy megteremtjük a kapcsolatot a fizikai és a logikai fájl között, és a fizikai fájlt előkészítjük a további műveletekre.

3. **Bővítés**
   Az állományt újabb rekorddal bővítjük.

4. **Keresés**
   A keresés mindig valamilyen kulcs szerint történik: a megadott kulcsot összehasonlítjuk a vizsgált rekordok megfelelő mezőivel.

5. **Törlés**
   Az állományból egy rekordot törlünk. A törlés lehet csak logikai, azaz csak kijelöljük a rekordot törlésre, és a továbbiakban a logikai állományban már nem hozzáférhető, annak ellenére, hogy a fizikai állományban még létezik. A tényleges, fizikai törlés után a rekord már a fizikai állományban sem található.
   Ez a megoldás (logikai-fizikai törlés) megtalálható a dBase-ben is.

6. **Módosítás**
   A rekordokat sokszor módosítani kell, ez a leggyakrabban előforduló fájlművelet.

7. **Lezárás**
   Ez a megnyitás ellenkezője. Ilyenkor általában a következő dolgok történnek:

   - Ha nem telt meg még a puffer, akkor az ilyenkor kiürül, azaz kikerül a tartalma a lemezre.
   - Az adatállomány adatai aktualizálódnak a háttértáron. Pl.: hossz, idő, dátum stb.

8.4 Egyszerű állomány szerkezetek

Az egyszerű állományokban a keresés a rekordok mezőinek a segítségével megoldható, kiegészítő adatokra nincsen szükség.

8.4.1 Szeriális állomány

Ez tulajdonképpen *szerkezet nélküli* állományt jelent. Az egyes rekordok olyan sorrendben helyezkednek el egymás után, ahogyan a rögzítésük történt. A szeriális állományok **bővítése** rendkívül könnyű, hiszen az új rekord egyszerűen az állomány végére kerül. A **keresés** művelete csak nagyon gazdaságtalanul végezhető el: az állományt a végéig kell átnéznünk, ha nem találunk egy rekordot.

8.4.2 Szekvenciális állomány

A szekvenciális állomány a szeriálisból keletkezik, úgy, hogy valamilyen azonosító alapján sorba *rendezzük*. Ez természetesen fizikai rendezettséget is jelent. Így bármelyik rekord azonosítójának az ismeretében egyértelműen megadható egy adott kulcsú rekord helye. A rekordokhoz való hozzáférés csak a rendezettségből adódó sorrendben lehetséges.

A rekordok lehetnek *fix vagy változó hosszúságúak* is.

Megjegyzés:

Szekvenciális állományt létrehozhatunk lemezegységen és szalagos egységen is. Mivel a PC-ken nemigen találkozhatunk szalagos egységen tárolt adatállományokkal, ezért csak a lemezegységen létrehozott szekvenciális állományokkal foglalkozunk a továbbiakban.

Nagy előnye a szekvenciális állománynak a szeriálishoz képest, hogy egy rekord meg**kereséséhez** nem kell az egész állományt végignézni. Elindulunk az állomány elejétől, és addig lépegetünk előre az állományban, amíg meg nem találtuk a keresett rekordot, vagy túl nem haladtunk azon a helyen, ahol az adott rekordnak lennie kellene a kulcsa alapján. Ezt hívjuk *soros keresésnek*. De alkalmazhatunk a soros keresésnél hatékonyabb módszert is, az ún. *bináris keresést.*

A **módosítás** *fix hosszúságú rekordok esetén* rendkívül egyszerű, hiszen visszaírhatunk a lemezre, közvetlenül a módosítandó rekordba.

A **törlés** egy kicsit már körülményesebb: legtöbbször csak logikai törlést alkalmazunk, és ha már nagyon zavaró a fizikailag meglevő, de logikailag már „meghalt" rekordok jelenléte, csak akkor kell az egész állományt újraszervezni.

Ez az *újraszervezés* azt jelenti, hogy létrehozunk egy új fizikai állományt, és ebbe átmásoljuk a „régi" állományból azokat a rekordokat, amelyek logikailag is léteznek.

A **bővítés** a „legkellemetlenebb" művelet, hiszen mindig az előzőekben megismert *újraszervezéssel* jár együtt.

8.4.3 Direkt állomány

A benne szereplő rekordok azonosítóinak sorszám jellegűeknek kell lenniük. Kölcsönösen egyértelmű leképezést kell létesíteni a rekordok azonosítói és a háttértáron levő, lehetséges rekordhelyek címei között, így bármely rekord csak azt a helyet foglalhatja el, amelyet a hozzárendelés alapján kap.

A rekordhelyek száma tehát rögzített. Az előzőek alapján nyilvánvaló, hogy direkt állományt csak olyan háttértáron hozhatunk létre, amelyet közvetlenül címezhetünk. Ilyen például a lemezegység.

A direkt állomány természetesen csak *fix hosszúságú* rekordokat tartalmazhat.

A direkt állomány **létrehozása** úgy történik, hogy meg kell határozni a szóba jöhető összes azonosítót, és a kölcsönösen egyértelmű leképezés segítségével minden lehetséges rekord helyét ki kell jelölni a lemezen, ún. *álrekordokkal* töltjük fel az adatállományt.

Ha meg akarunk **keresni** egy rekordot, akkor az egyértelműen megtehető a rekord azonosítója alapján.

A **módosítás** a lemezre történő közvetlen visszaírással érhető el.

A **bővítés** is rendkívül egyszerű, hiszen minden rekordnak eleve lefoglaltuk a helyet a lemezen már a létrehozáskor.

A **törlés** nyilván csak logikai törlést jelent.

A *Turbo Pascal* típusos fájljai és a *dBase* állományai alapvetően szeriálisak, de minden rekordhoz tartozik egy ún. *rekordmutató*, ami egy sorszámot rendel hozzájuk. Ez a sorszám nem az adott rekord azonosítójából keletkezett, hanem abból a sorrendből, ahogyan felvittük őket egymás után. Így úgy kezelhetjük az állományt, mintha direkt lenne, hiszen a rekordmutató alapján tetszőleges rekordra állhatunk rá. Szokták ezt **kvázi direkt állománynak** is nevezni.

8.4.4 Random állomány

Hasonlatos a direkt állományokhoz, de a rekordok azonosítói és a lemezcímek között nincsen kölcsönösen egyértelmű hozzárendelés. Előfordulhat, hogy több rekord is ugyanarra a helyre kerülne. Ilyenkor az első rekord ide is kerül, és ha érkezik még egy rekord, amelynek az azonosítója alapján szintén ez a helye, akkor egy megfelelő algoritmus segítségével helyet kell neki keresni. Ezek a rekordok az ún. *túlcsorduló rekordok.*

8.5 Adatállományok feldolgozása

A feldolgozási műveletnél a már létező fájlban levő rekordokkal valamilyen műveletet végzünk. A feldolgozásnál alapvetően három ún. ***elérési mód*** van, amely az egyes rekordokhoz való hozzáférést jellemzi:

- **Soros elérés**
  A rekordokat a fizikai sorrendjükben tudjuk elérni.
- **Szekvenciális elérés**
  A rekordokat valamilyen logikai sorrend alapján érjük el. Sok esetben a logikai sorrend nem egyezik meg a fizikai sorrenddel (lásd később: láncolás, indexelés).
- **Direkt elérés**
  A rekord azonosítóját (esetleg a rekordmutatót) megadva a rekord egyértelműen elérhető.

Megjegyzés:

Vannak olyan állományok, amelyeknek a rekordjai többféleképpen is elérhetők.
*Például: egy kvázi direkt állomány (**Turbo Pascal, dBase**) feldolgozható sorosan és direkt módon is, illetve a **dBase**-ben lehetséges a szekvenciális elérés is indexelés segítségével.*

8.6 Összetett állományok

Az összetett állományszerkezet rekordjaiban a tényleges adatokon kívül más segédinformációk is vannak, az ún. *szerkezet hordozó adatok*. Ezek a fizikai állománynak a háttértáron való elhelyezkedésére vonatkozó adatait tartalmazzák.

Összetett állományokat csak címezhető háttértáron hozhatunk létre, például lemezegységen.

Megjegyzés:

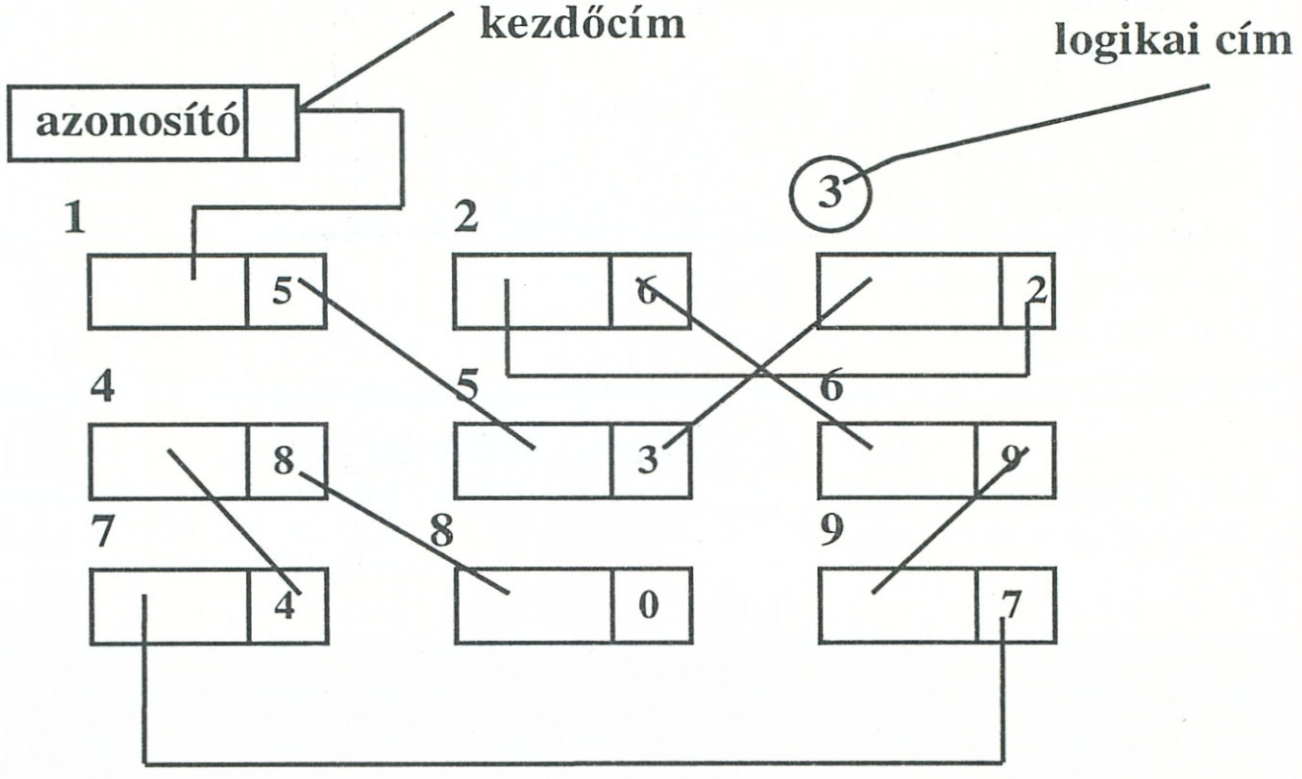
Az összetett állomány is lehet szeriális, szekvenciális, direkt vagy random.

8.6.1 Láncolás

A láncolás segítségével egy adatállomány *szekvenciális feldolgozását* tesszük lehetővé egy adott kulcs szerint. Minden rekordot kiegészítünk egy *mutató mezővel*, amely mindig a következő rekord helyét adja meg a rendezési kulcsnak megfelelő sorrendben. Az így kialakított lánc a fizikaitól lényegesen eltérő logikai sorrendet mutathat. Felfoghatjuk az állományt úgy is, mint egymás után felfűzött rekordok összességét.

A láncolás történhet az elsődleges vagy valamelyik másodlagos kulcs szerint.

Az állományon végzett műveletek (keresés, törlés, bővítés, módosítás) hasonlóképpen végezhetők el, mint ahogyan azt az egyirányú láncolt listánál láttuk.



8.6.2 Indexelés

Az adatállományhoz létrehozunk egy állományt, amely az ún. *indextáblát* tartalmazza. A legegyszerűbb esetben ennek annyi sora van, ahány rekordból áll az állomány, és két oszlopra osztható.

Az *első oszlop* az adatállomány rekordjainak az *azonosítóit* tartalmazza növekvő vagy csökkenő sorrendben. A második oszlopban egy mutató van, ami az adott rekord helyét jelöli a lemezen. Ez lehet egy konkrét lemezcím, vagy olyan információ, amiből ki lehet számítani a tényleges helyet.

Az *indextábla* az eredeti állomány azonosítójára vonatkozó *szekvenciális állomány*. Érdemes használni az indextáblát, hiszen az kisebb méretű, mint az eredeti állomány, ráadásul valószínűleg teljes egészében befér a memóriába, és így gyorsabb a feldolgozás.

Indextábla

kulcsmező	mutató
televízió	
rádió	
homokóra	
számítógép	
horgászbot	
légpuska	
mókuskerék	

Az adatállomány

horgászbot	1500	van	1267-56
mókuskerék	350	nincs	567-567
televízió	30000	van	2156-98
homokóra	145	van	324
számítógép	78000	van	234-768
rádió	1200	nincs	213-45
légpuska	2300	nincs	213-65

Az azonosítót elsődleges kulcsnak is nevezhetjük. Lehetséges az indexelés egy *másodlagos kulcs* alapján is, ami tulajdonképpen valamely mező az azonosító mezőn kívül. Ebben az esetben a táblázat annyi sorból áll, ahány különböző másodlagos kulcsérték van az adatállományban. A második oszlopban több bejegyzés is van, annyi, ahány rekord tartalmazza az adott kulcsértéket.

Ezt az indexelési formát nevezzük *egyszintű indexelésnek*. Sok előnnyel rendelkezik ez a megoldás, hiszen az indextábla kezelése lényegesen egyszerűbb, mint az eredeti állományé. Nagyobb állományok esetén előfordulhat, hogy az indextábla is nagyméretű lesz, így lassul a feldolgozás.

Mivel az indextábla is egy állomány, ezért ezt is lehet indexelni. Ezt hívják *két-* vagy *többszintű indexelésnek.*

Az indexelés következtében az állományt *szekvenciálisan* kezelhetjük. Másik nagy előnye, hogy gyakorlatilag bármely kulcsra készíthetünk indextáblát.

Megjegyzés:

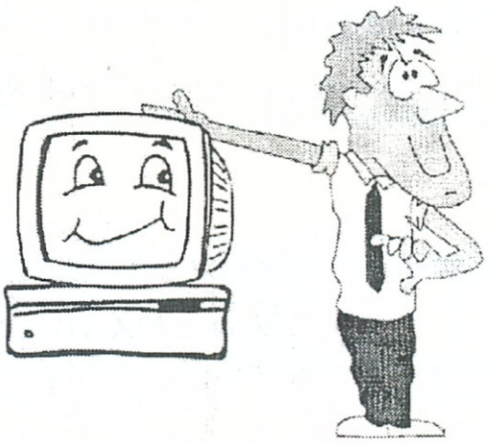
A dBase is egyszintű indexelést tartalmaz. Lehetséges elsődleges és másodlagos kulcsra is indexelni.

9. ADATBÁZISOK

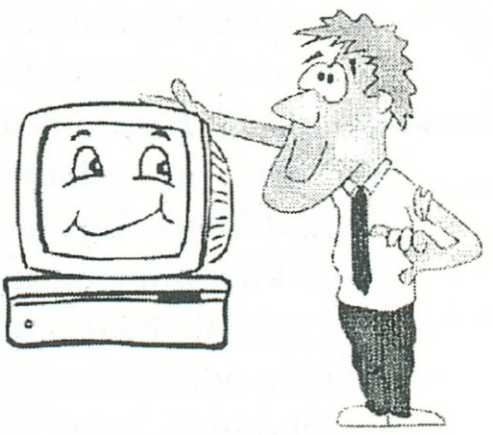
Az adatbázis szemléletű adatfeldolgozást az ún. *hagyományos adatkezelés* felváltására hozták létre.

9.1 A hagyományos adatkezelés jellemzői

- Minden alkalmazás *saját adatállományokkal* dolgozik.
- Az előző tulajdonság miatt sokszor előfordul az adatok *többszörös tárolása*. Például egy vállalatnál több üzemben is tárolják ugyanazokat az adatokat a dolgozókról, hiszen különböző programokat használnak az egyes üzemekben, egymástól függetlenül.



- Az előzőek miatt jellemző az *inkonzisztencia*. Ez azt jelenti, hogy ha egy bizonyos adatot megváltoztatnak az egyik helyen, előfordulhat, hogy a másik helyen nem végzik el a szükséges változtatást, hiszen az adat többszörösen van tárolva, egymástól függetlenül.



- *Nehézkes módosíthatóság*: A felhasználói programban definiálták a hozzátartozó adatállományok szerkezetét. Ezért ha igény merül fel az adatállomány szerkezetének a megváltoztatására, az együtt jár a program átírásával.

9.2 Az adatfüggetlenség elve

A cél az, hogy ugyanazokat az adatokat több felhasználó, több programmal is használhassa, ezért az adatokat, amennyire csak lehetséges, függetleníteni kell a programoktól.

Például, ha valamelyik felhasználó egy újabb mezőt igényel, akkor elég az adatállományt egy új mezővel kiegészíteni, és csak azt a programot kell kis mértékben módosítani, amellyel ez a felhasználó dolgozik. Ugyanakkor az összes többi program változatlan maradhat. Ráadásul az egyes programoknak nem szükséges az összes mezőt „látniuk" (akkor is, ha nincs is rá szüksége).

Ezt a függetlenséget úgy lehet megvalósítani, hogy *az adatállomány logikai szerkezetét nem a programban definiáljuk, hanem magában az adatállományban.*

Ezt hívjuk *adatfüggetlenségnek.*

9.3 Az adatbázis-szemlélet jellemzői

- A legfontosabb eltérés a hagyományos adatkezeléssel szemben az, hogy *megszüntetjük* az adatok többszörös tárolását.
- Az előzőleg említett többszörös tárolás megszűnése miatt az adatok *inkonzisztenciája is megszűnik.*
- A adatok *központilag tárolhatók.* A központi tárolás következményeként az adatok *felhasználásának ellenőrzése* könnyebben megoldható. Egyszerűbben kiszűrhetők azok, akik az adatokhoz illetéktelenül akarnak hozzáférni.
- Az adatok központi tárolásának a következményeként lehetőség van az adatbázisok *közös, megosztott használatára*, ami rendkívül hatékonnyá teszi az adatfeldolgozást egy nagyobb szervezeten belül.
- Az *adatfüggetlenség elve* teljesül.

Az adatok feldolgozásához szükség van egy *adatkezelő nyelvre*, amelynek a segítségével elvégezhetők az adatbázison a szükséges műveletek. (Az adatbázis előállítása, módosítása, lekérdezése, rekordok keresése, törlése, stb.)

Ez a nyelv lehet önálló (pl.: *dBase, Clipper, FoxPro,* stb.), vagy egy magasszintű nyelvbe beágyazott. Ez azt jelenti, hogy egy általános célú programozási nyelvet kiegészítünk az adatbázis kezeléséhez szükséges utasításokkal.

Az önálló adatbázis-kezelő rendszerek általában egyszerű, könnyen elsajátítható nyelvet tartalmaznak. Az adatokat csak logikai szinten kezelhetjük. Ezeknek a rendszereknek nagy előnye, hogy nem csak előre megírt programokat használhatunk, hanem *interaktív lekérdező rendszert* is tartalmaznak.

9.4 A relációs adatmodell

A relációs adatmodellben az adatok között relációk (kapcsolatok) állnak fenn. Minden egyes reláció egy *táblázat* segítségével adható meg. A táblázat sorait *bejegyzésnek* nevezzük, az oszlopokat pedig *tartománynak.*

Fájl szervezési szempontból a bejegyzéseknek az egyes *rekordok*, a tartományoknak a *mezők* felelnek meg. Minden bejegyzés összetartozó adatokat tartalmaz.

Relációs adatmodellt használ például a *dBase* és a *FoxPro*, ezeket szokták *relációs adatbázis-kezelőknek* is nevezni.

Például:

Cikkszám	Cikknév	Ár	Készlet
123	álruha	1230	12
32432	működm	321	1242
23	fakalapács	156	17000
3465	hosszú kés	1756	1

A táblázatban nem fordulhat elő két teljesen egyforma bejegyzés, viszont a bejegyzések sorrendje nem lényeges.

9.5 Műveletek relációs adatbázisokkal

Az adatbázisokkal többfajta művelet végezhető. Itt mi kettőt nézünk meg, amelyek a dBase-ben is elvégezhetők.

1. **Összekapcsolás (JOIN):**
   Két adatbázisból készít egy harmadikat. Az összekapcsolás egy összekapcsoló feltétel felhasználásával történik. Venni kell az első reláció (adatbázis) első bejegyzését, és hozzárendelni a másik reláció azon bejegyzéseit, amelyek megfelelnek az összekapcsoló feltételnek. Ezután az első reláció összes bejegyzésével meg kell ismételni az előzőeket.

   Például: Legyen adva a következő két reláció:

   KÖLCSÖN reláció:

Raktári szám	Igazolvány szám	Kölcsönzési idő
234	AB-1001	1994-01-12
234	AB-2002	1993-12-24
111	AB-1	1994-02-03
222	AB-2	1993-12-23
333	AB-3	1994-01-18
123	AB-3003	1994-03-04

   KÖNYV reláció:

Raktári szám	Kiadó	Cím	Szerző
234	Magvető	A cinkos	Konrád Gy.
123	SZÁMALK	C	Pethő Á.
111	Európa	A per	Franz K.
222	Magvető	Az áruló	Sánta F.
333	Európa	Macskabölcső	K. Vonnegut
345	LSI	80386 I.	Kovács M.

Kapcsoljuk össze a KÖLCSÖN relációt a KÖNYV relációval! Legyen az összekapcsoló feltétel az, hogy a „Raktári szám" mezők tartalma megegyezik.

Az eredmény reláció:

Raktári szám	Kiadó	Cím	Igazolvány szám	Kölcsönzési idő	Szerző
234	Magvető	A cinkos	AB-1001	1994-01-12	Konrád Gy.
234	Magvető	A cinkos	AB-2002	1993-12-24	Konrád Gy.
111	Európa	A per	AB-1	1994-20-03	F. Kafka
222	Magvető	Az áruló	AB-2	1993-12-23	Sánta F.
333	Európa	Macska-bölcső	AB-3	1994-01-18	K.Vonnegut
123	SZAMALK	C	AB-3003	1994-03-04	Pethő Ádám

2. Szelekció (FILTER):

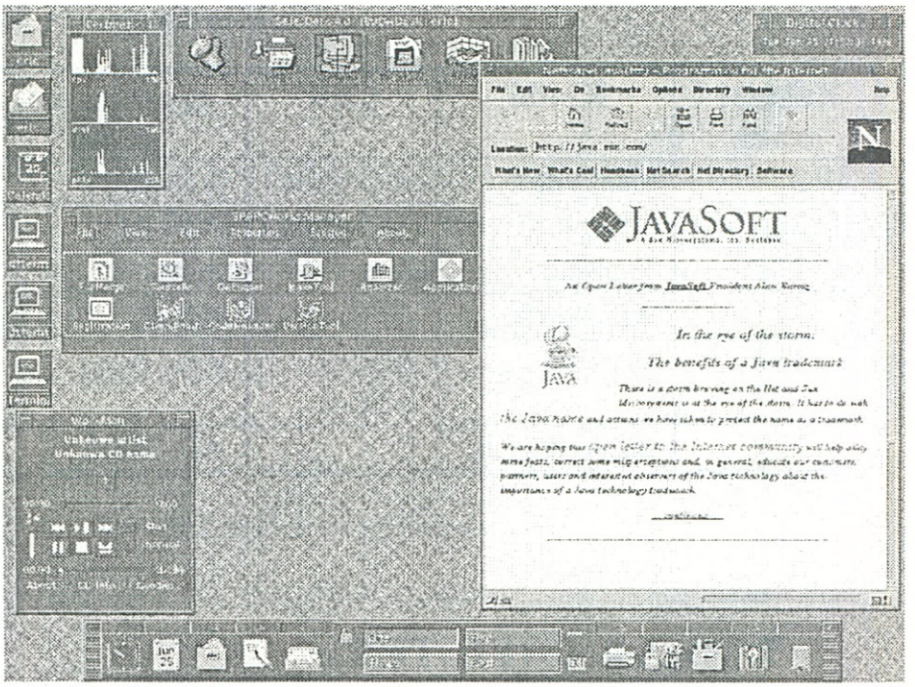
A relációt leszűkítjük meghatározott tulajdonsággal rendelkező bejegyzésekre.

Például: A KÖNYV relációt véve, válasszuk ki azokat a rekordokat, amelyekben a „Kiadó" mező tartalma: Európa.

Az eredmény reláció:

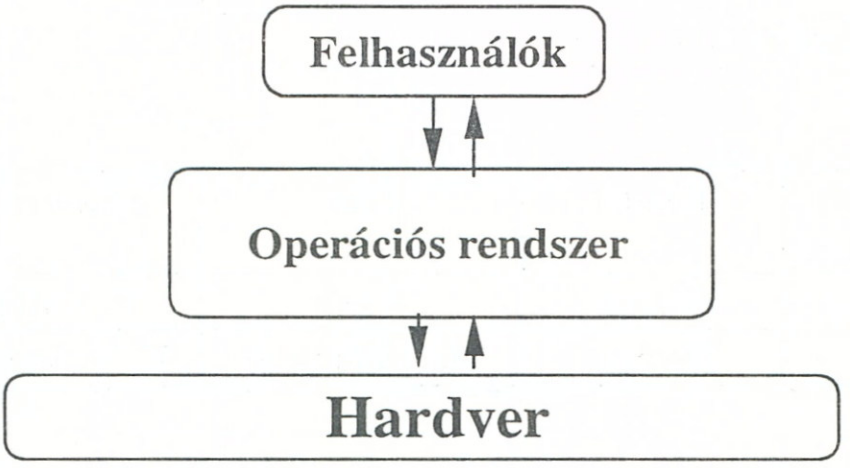
Raktári szám	Kiadó	Cím	Szerző
111	Európa	A per	F. Kafka
333	Európa	Macskabölcső	K. Vonnegut

10. AZ OPERÁCIÓS RENDSZER FOGALMA



Az operációs rendszer kapcsolatot teremt a gép és az ember között, lehetővé teszi programok írását, futtatását.

Az operációs rendszerek a számítógépek megjelenése után körülbelül 20 év múlva terjedtek el. Az első gépeken vagy nem volt operációs rendszer, vagy nagyrészt csak hardver elemekből állt (kapcsolók, indító-, megállító, lépésenkénti végrehajtást kiváltó gombok stb). Csak később alakultak ki a főként szoftverrel megvalósított operációs rendszerek.



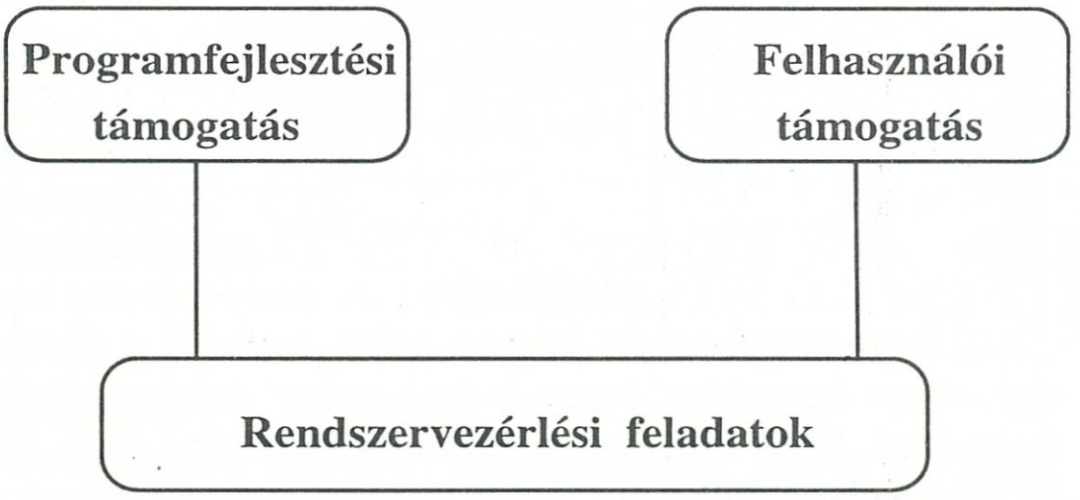
Az operációs rendszer egy olyan programrendszer, amely vezérli a programok végrehajtását, ütemezi a programok futását, ha több program is fut a gépen, elosztja az erőforrásokat, a felhasználó és a számítógép között kapcsolatot teremt.

Megjegyzés:

A PC-ken a legelterjedtebb operációs rendszer az MSDOS, mi is ezzel ismerkedtünk meg, de léteznek más rendszerek is. A legismertebbek: UNIX (ennek ráadásul több változata is forgalomban van), OS/2, Windows 95, Windows 98, Windows NT, Novell IntranetWare, stb.
Természetesen ezeken kívül vannak más operációs rendszerek is, hiszen a PC-k csak egy részét jelentik a számítógépeknek.

10.1 Az operációs rendszer funkciói

Az operációs rendszerek funkcióit többféleképpen is csoportosítják. Mi a következő felosztást használjuk:



A továbbiakban olyan fogalmakat ismerünk meg, amelyek általában az operációs rendszerekkel kapcsolatosak, és egyáltalán nem biztos, hogy ezek mindegyike az MSDOS-ra is jellemző. Azért foglalkozunk mégis általánosan ezekkel, mert már vannak más PC-s operációs rendszerek (UNIX, OS/2, Novell DOS stb.), amelyek az MSDOS-nál lényegesen nagyobb teljesítményűek, és amelyekre a továbbiak már érvényesek.

10.1.1 Rendszervezérlési feladatok

Az operációs rendszernek a hardver kezelésével, működtetésével kapcsolatos, feladatairól van itt szó. Vázlatosan láthatjuk a következőkben, hogy általában az egyes operációs rendszereknek milyen rendszervezérlési feladataik vannak. Ezek a funkciók nem biztos, hogy minden operációs rendszernél megtalálhatók.

- **Processzorütemezés**
  A CPU idejét kell szétosztani az egyes folyamatok között. Mivel a számítógép egy időben csak egy utasítást tud végrehajtani, ezért, ha azt szeretnénk, hogy a gép párhuzamosan több feladatot is végrehajtson, akkor a CPU idejét meg kell osztani az egyes feladatok között. A legegyszerűbb esetekben is meg kell osztani a processzor idejét: Például az operációs rendszernek és egy felhasználói programnak a végrehajtása között.

- **Megszakítás vezérlés**
  Az egyes megszakításkérések elemzéséről és végrehajtásáról van szó. A megszakításokra azért van szükség, mert a számítógépek olyan részeket is tartalmaznak, amelyek párhuzamosan is működtethetők. Ilyenek például az egyes perifériák. Ezek a berendezések általában időigényes feladatokat látnak el, és többnyire csak indítást igényelnek a központi egységtől, végrehajtásukról az egyes intelligens vezérlők gondoskodnak. A probléma az, hogy az elindított folyamatok végét hogyan lehet észlelni. Lehetséges lenne egy olyan megoldás, hogy a processzor bizonyos időközönként megnézné, hogy az egyes perifériák foglaltak-e még. Ha sok perifériával kell foglalkozni, akkor a processzornak másra nem is jutna ideje, mint a perifériákat figyelni. Jobb megoldás az, ha a periféria jelez vissza a processzornak. Ilyenkor a processzornak az éppen folyó munkáját meg kell szakítania ahhoz, hogy a folyamat végén szükséges teendőket el tudja végezni. Megszakítást hoznak létre az egyes hardvereszközök meghibásodásai is.
  A processzor ilyenkor egy úgynevezett *megszakításkezelő* rutint hajt végre. A megszakításkezelő rutin indítása előtt rögzíteni kell a processzor aktuális állapotát, hogy később a processzor képes legyen folytatni az eredeti munkáját.

Bonyolítja a helyzetet, hogy egy megszakítás feldolgozása közben újabb megszakítási kérelmek is érkezhetnek. Ezt a problémát különböző módszerekkel lehet megoldani. A megszakításokat sorba lehet állítani valamilyen előre meghatározott fontossági sorrend alapján, és ez alapján dőlhet el, hogy az éppen futó megszakításkezelő rutint megszakítjuk-e. Lehetnek olyan megszakítások, amelyeket fontosságuk miatt nem lehet megszakítani. (Például processzorhiba.)

- **Tárkezelés**
  Az operatív tár az egyik legfontosabb erőforrás, amelynek az elosztása, kezelése fontos feladata az operációs rendszernek. Amikor például elindítunk egy programot, akkor gondoskodni kell a megfelelő elhelyezéséről az egyes programok között. Ha több program fut párhuzamosan a gépen, akkor a memóriát meg kell közöttük osztani, és biztosítani kell, hogy ezek a memóriaterületek védettek legyenek.

- **Perifériakezelés**
  Az egyes perifériákkal való kapcsolattartás biztosítása.
  A hardverhez általában több periféria kapcsolódik (például monitor, billentyűzet, winchester, stb.), amelyeknek megfelelő kezelése szintén az operációs rendszer feladata.

- **Adatkezelés**
  Az állományok tárolása, nyilvántartása, illetve a rajtuk végzett műveletek támogatása minden operációs rendszer alapvető feladata. Ilyen műveletek például: állományok létrehozása, írása, olvasása, stb. Az operációs rendszerek csak néhány alapvető állománytípust ismernek. A UNIX például csak egyféle állománytípust ismer: minden fájl bájt-ok sorozata. (Lásd: *Függelék!*)
  Az adatkezelés része a fájlok tárolásának rendszere is. Például a DOS esetén a lemezek megismert logikai szerkezete.

10.1.2 Programfejlesztési támogatás

Az operációs rendszer csak a kapcsolatot teremti meg a felhasználó és a hardver között, de hogy ténylegesen használhassuk a gépet, programokat kell hozzá írni. Az operációs rendszerek általában támogatják ezt a tevékenységet.

- **Szövegszerkesztés**
  A programokat *szövegszerkesztő* segítségével írjuk, ezt a legtöbb operációs rendszer tartalmazza is (az MSDOS-ban az **EDIT**, a Linuxban például a **joe**, **vi**). A megírt programszöveget *forrásszövegnek* is nevezzük.

- **Fordítóprogram**
  A forrásszöveget *fordítóprogrammal* le kell fordítani. A fordítóprogram megérti az adott programozási nyelven írt programot, és ha hibátlan, akkor lefordítja olyan kódra (gépi kód), amelyet a gép értelmezni tud. A legtöbb operációs rendszerrel a gyártók fordítóprogramokat is szállítanak. Például: a UNIX-al C nyelvűt, az MSDOS operációs rendszerhez viszont közvetlenül nem tartozik egyetlen fordítóprogram sem.

- **Programbetöltés**
  Az operációs rendszer feladata, hogy a már kész, lefordított programot indításkor betöltse a memóriába.

- **Programkönyvtárak**
  Bonyolultabb feladatokat általában nem lehet egyetlen nagy programmal megoldani, a feladat részekre bontásának a segítségével hatékonyabban dolgozhatunk. Ennek a módszernek van még egy előnye: a részfeladatok többször felhasználható programjaiból programkönyvtárak hozhatók létre.

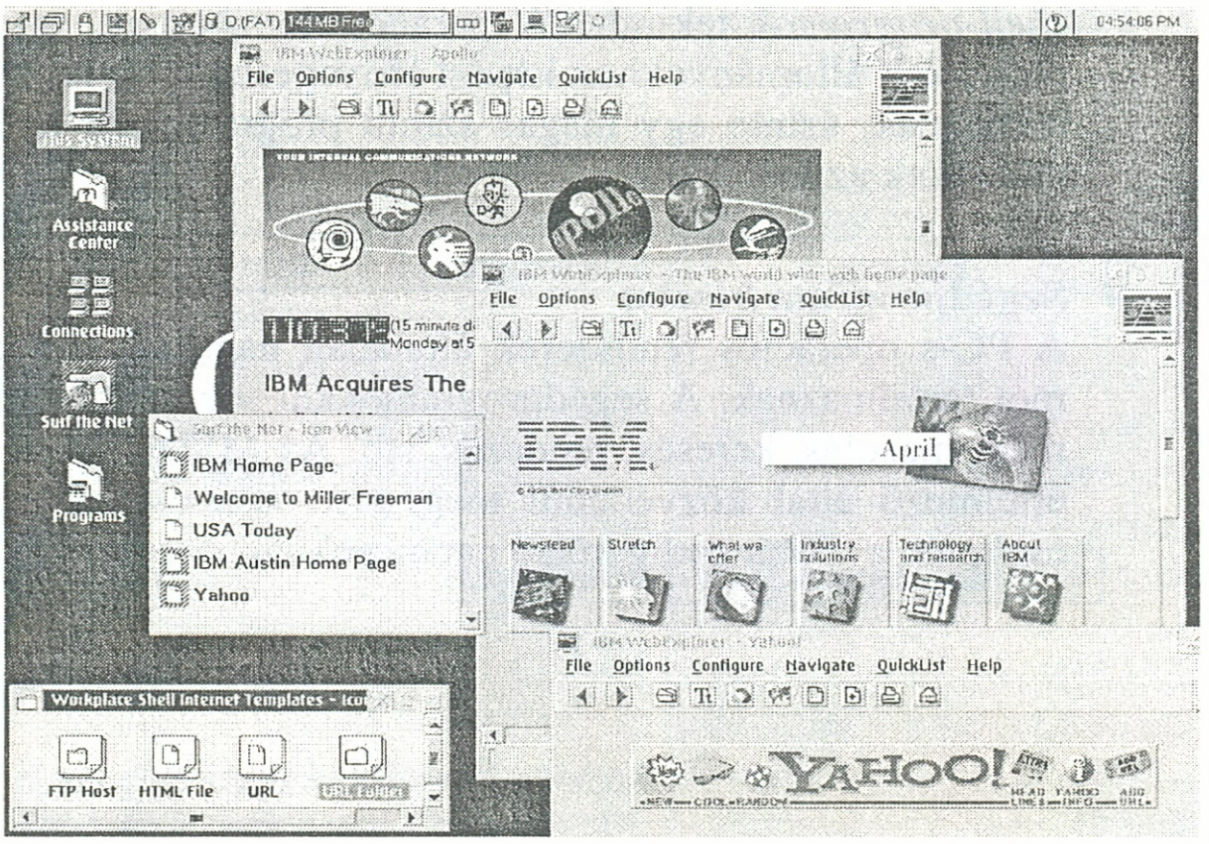
- **Programszerkesztés**
  Az egyes részprogramokat össze kell szerkeszteni egyetlen, futtatható programmá.

- **Rendszerhívások**
  A programkönyvtárak felhasználása nyilván segítséget jelent a programozónak, hiszen ezeket neki már nem kell megírnia, elég csak hivatkozni a benne levő eljárásokra, szubrutinokra.
  Ezt a hivatkozást szokták *rendszerhívásnak* is nevezni. A rendszerhívás sok esetben a *szoftvermegszakítás* segítségével történik, amely egy speciális processzorutasításnak végrehajtását jelenti. (Az MSDOS-ban az INT utasítás.)

- **Programtesztelés**
  Az operációs rendszer segítséget nyújt az elkészült program ellenőrzéséhez, vizsgálatához. Nagyon fontos lehetőség az úgynevezett *nyomkövetés*, amellyel a programot lépésekre bontva nyomon követhetjük. Az MSDOS is nyújt ilyen szolgáltatást, a **DEBUG** paranccsal, a UNIX-ban például az **sdb**.

10.1.3 Felhasználói támogatás

- **Felhasználói interfész**



Kapcsolattartás a felhasználóval: ha kapcsolatot akarunk teremteni a számítógéppel, akkor valamilyen formában utasítást, parancsot kell adnunk neki. Ez történhet parancsszavak beírásával, a gép válaszát pedig általában a monitoron láthatjuk, a beírt parancsot a parancsértelmező hajtja végre.
Az MSDOS-ban a **COMMAND.COM** látja el a parancsértelmező feladatát, a UNIX-ban több lehetőség is van, de az sh, csh, ksh minden UNIX rendszerben megtalálható. Manapság megjelentek, és egyre elterjedtebbek az úgynevezett grafikus felhasználói felületek (pl. a Windows), ahol a kapcsolattartás menüvezérelt, az egyes választási lehetőségek nemcsak szövegesek lehetnek, hanem történhet a kiválasztás az egérkurzorral, az *ikonokra* való rámutatással.

- **Parancsállományok**
  Ezek olyan szöveges állományok, amelyekben ún. *parancsnyelven* írt szövegek vannak, és ezekkel lehet helyettesíteni a terminálról kiadható parancsokat. Az MSDOS-ban a *batch fájl*ok játsszák ezt a szerepet, amelyekben DOS-parancsok kötege van. Ezeket a batch fájlokat ugyanúgy a parancsértelmező hajtja végre, mint a billentyűzetről bevitt egyedi parancsokat. A UNIX-ban az ún. *shell-programok* tekinthetők parancsállománynak, igaz, a DOS-beli batch állományokhoz képest lényegesen több lehetőséget tartalmaznak, szinte egy magas szintű programozási nyelvet használhatunk az írásukkor.

- **Segédprogram készlet**
  A PC-s operációs rendszerek általában nagyszámú segédprogramot tartalmaznak. A segédprogramokkal sok rutinfeladat oldható meg: másolás, keresés, rendezés, stb. Az MSDOS-ban a parancsértelmező által közvetlenül megértett parancsokat belső parancsoknak, a többit pedig külső parancsoknak nevezzük.

10.2 Tárkezelés

Az operációs rendszer egyik legfontosabb feladata a megfelelő tárkezelés. A hatékony programozás feltétele a gyors és megfelelő méretű tár.

10.2.1 Overlay programok

A programozók egy idő után szembekerültek azzal a problémával, hogy az operatív tár túl kicsi. Eleinte megpróbálták mindenféle programozási trükk segítségével csökkenteni a programok méretét, de ez sem hozott igazi megoldást.

Ennek a problémának a megoldására hozták létre az overlay technikát. A programokat olyan kis darabokra, *szegmentumokra* bontották, amelyek egyenként elfértek a memóriában, de nem ott, hanem valamelyik háttértárolón tárolták őket. A programokat úgy írták meg, hogy amikor egy rész befejezte a munkáját, akkor beolvasódik a következő, esetleg felülírva eközben az előző részt.

Például:

Kezdetben:

rezidens rész	1. szegmentum	szabad

A 2. szegmentum betöltése:

rezidens rész	1. szegmentum	2. szegmentum

A 3. szegmentum betöltése

rezidens rész	3. szegmentum	2.szegmentum

A program egy *rezidens* (állandó tartalmú) és egy *tranziens* (változó tartalmú) területen helyezkedik el. A vezérlőrész a rezidens területen működik, és fő feladata, hogy a tranziens részre a program logikája szerinti sorrendben behívja az egyes szegmentumokat.

Ha nincs már hely a következő szegmentumnak, akkor az felülír egy már bent levő, éppen nem használt szegmentumot.

Az overlay technika általában jól működik, de nem mindig könnyű használni. Néha nagyon nehéz a programokat szegmentálni, vagy ha már sikerült is, nem mindig tudjuk, hogy a program futása közben mikor melyik szegmentumra lesz szükség.

Egy másik probléma, hogy a központi tárban egy sor, szegmentumokra vonatkozó információt kell tárolni. Ahogyan nő a program mérete, úgy nő az overlay struktúra elkészítésének és fenntartásának munka- és időigénye. Azt mondhatjuk tehát, hogy csak akkor érdemes az overlay szerkezetet használni, ha a programunk már semmiképpen nem fér el konvencionális memóriában.

Megjegyzés:

Az overlay technikát sok ismert program használja. Például: ***Norton Commander, dBase III Plus,*** *stb., de mi magunk is használhatjuk például a* ***Turbo Pascalban.***

10.2.2 Virtuális tárkezelés

Ha az overlay technikát az operációs rendszer szintjén valósítjuk meg, *virtuális tárkezelésről* beszélünk. A programozó elfelejtheti, hogy a tár korlátozott méretű, számára a tár gyakorlatilag korlátlan.

A virtuális tárkezelés alapművelete a *lapozás*. A virtuális tárkezelést végző operációs rendszer a programot több lapra osztja. A *lap* mérete nem lehet nagyobb, mint a szabad tárterület. A program futásának kezdete előtt az operációs rendszer betölti a lapokat a memóriába. A program megkezdi működését, és fut egészen addig, amíg nem hivatkozik egy olyan utasításra vagy adatra, amely egyik lapon sincs rajta. Ekkor az operációs rendszer kimásolja a tár egy részét a lemezre, ha nincs elég hely a memóriában, majd a lemezről a tárba másolja azt a lapot, amelyen a szükséges adat vagy utasítás található.

Az overlay technika és a virtuális tárkezelés első pillantásra nagyon hasonlít egymásra, de lényeges különbségek vannak közöttük. Az overlay technikánál a programozónak kell megszerveznie a tárfelhasználást, míg a virtuális tárkezelésnél nem kell vele törődnie.

Így tulajdonképpen rendelkezünk egy logikai tárral, amelynek egy része a fizikai tárban (a „valódi" tárban), a többi a lemezen helyezkedik el. A logikai és fizikai tárat is lapokra bontjuk. Az úgynevezett *laptábla* megadja, hogy a logikai tárban elhelyezkedő egyes lapok hol találhatók: a fizikai memóriában, vagy a háttértáron.

A lapok mérete általában lényegesen kisebb, mint a teljes fizikai tárméret, általában néhány Kbájt. Egy adott pillanatban a lapok egyik része a memóriában, a másik a lemezen van.

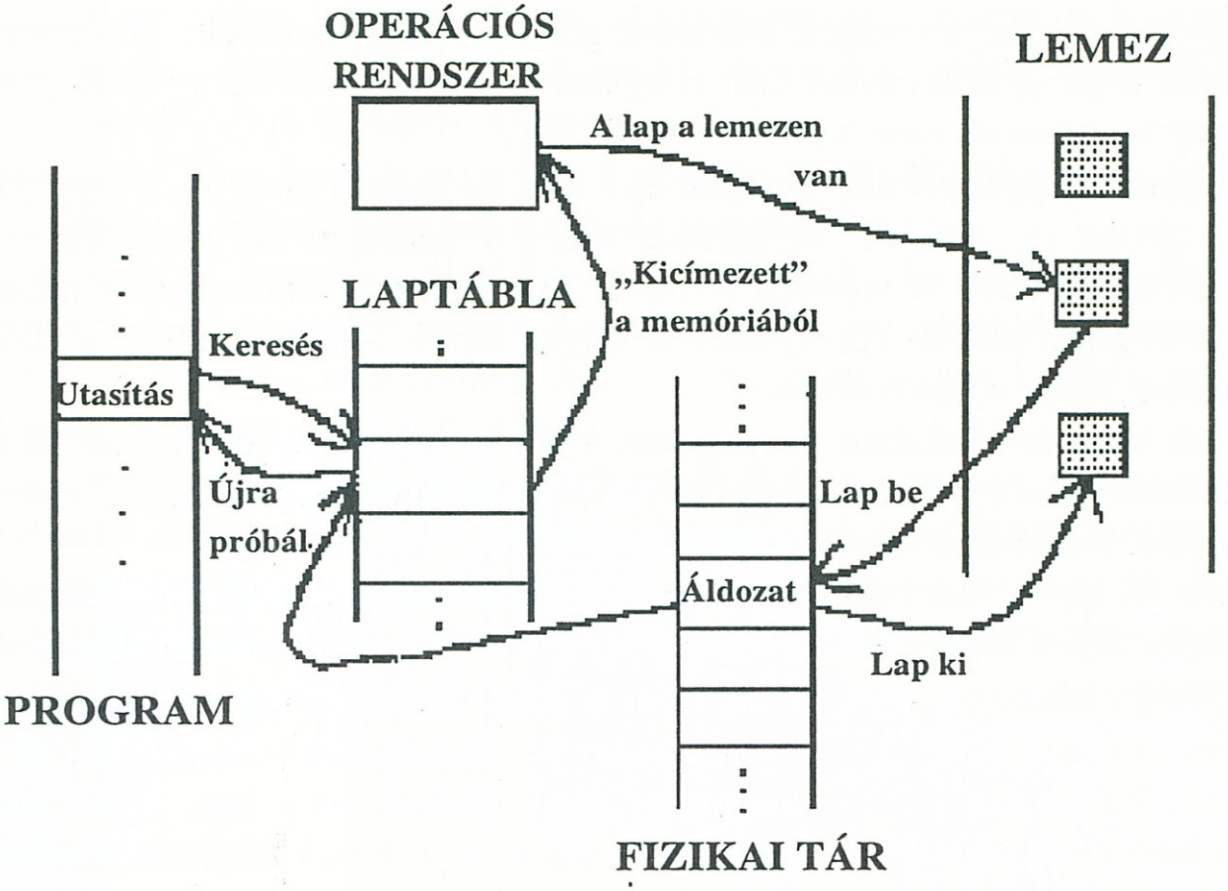
Amikor új lapra van szükség, az operációs rendszernek el kell döntenie, melyik lapot lehet lecserélni és visszaírni a lemezre. A tervezési algoritmus feladata, hogy eldöntse, melyik lapra nem lesz valószínűleg szükség a közeljövőben. Az egyik lehetséges megoldás a *„legrégebben használt lap algoritmus"*. Ez az algoritmus megnézi, hogy melyik az a lap, amelyiket a program hosszabb ideje nem használt *(„áldozat")*. Erről feltételezi, hogy a rajta levő adatokra illetve utasításokra nem lesz szükség egy ideig. Ha a feltételezés igaz, akkor a többi, gyakrabban használt lap marad a memóriában, ezt a lapot pedig kicseréli.

Összefoglalva

A program egyik utasítása hivatkozik egy logikai címre, az operációs rendszer *megkeresi* a laptáblában, hogy melyik lap tartozik hozzá, és megnézi, hogy hol van ez a lap. Ha *„kicímezett"* a memóriából, azaz ez a lap nincs a fizikai memóriában, és ha van még hely a fizikai tárban, akkor az *operációs rendszer a lemezhez fordul*, és *beolvassa* a szabad helyre a megfelelő lapot. Ha nincs hely a fizikai tárban, akkor valamelyik lapot egy megfelelő algoritmus segítségével kijelöli („áldozat"), és *kiviszi* a lemezre, majd a helyére *behozza* a szükséges lapot. Ezután a laptáblába beírja az új helyzetet, és az utasítást *újra megkísérli végrehajtani*.

Természetesen az is előfordulhat, hogy a keresett cím egyáltalán nincs a fizikai tárban, ilyenkor a program programhibával leáll.

A virtuális tárkezelés segítségével megnövekszik a tárkapacitás, de ennek a módszernek is vannak hátrányai: a bonyolultabb címzés és a lemez használata miatt lassul a feldolgozás.



Megjegyzés:

Az MSDOS operációs rendszer „természetesen" nem képes a virtuális tárkezelésre sem, viszont alkalmazza például a UNIX, és a Windows is, az Enhanced módban, vagy a Novell DOS is.

10.3 Az operációs rendszerek osztályozása

Az operációs rendszereket több szempont szerint is csoportosíthatjuk. Mi ebből három szempontot választunk ki:

1. **A felhasználók száma szerint:**
   - egyfelhasználós
   - többfelhasználós
2. **A feldolgozás módja szerint:**
   - kötegelt
   - interaktív
   - valós idejű
3. **Az egyidőben futtatható programok száma szerint:**
   - monoprogramozott
   - multiprogramozott

1.

10.3.1 Egyfelhasználós rendszerek

Tipikusan egyfelhasználós operációs rendszer az MSDOS, de ilyen operációs rendszer található a mikrogépeknél is (Commodore 64, ZX Spectrum, stb.). Ezekre jellemző, hogy a gépet egyidőben csak egy felhasználó tudja használni. A felhasználói interfész általában interaktív jellegű, csak a régi egyfelhasználós rendszerekre volt jellemző a kötegelt feldolgozás (lásd: később).

10.3.2 Többfelhasználós rendszerek

A többfelhasználós operációs rendszerek értelemszerűen azt jelentik, hogy a gépet egyidőben több felhasználó használja, terminálokon keresztül.

A számítógépen tehát több program is fut egyidőben, ez a multiprogramozás.

10.3.3 Kötegelt feldolgozás

A felhasználók a teljes feladatot adják át az operációs rendszernek, majd később a kész eredményeket kapják vissza. A kötegelést a régi rendszerekben az *operátor* folytatta valamilyen szisztéma alapján.

Például összegyűjtötte külön a FORTRAN programokat, külön a COBOL programokat. Esetleg valamilyen fontossági szempontot is figyelembe vett a kötegek összeállításánál.

Volt egy nagyon kellemetlen vonása ennek a feldolgozási módnak: a programozók teljesen kiszorultak a gépteremből.

Kötegelt feldolgozást meg lehet valósítani egyfelhasználós rendszerekben is, például az MSDOS-ban is, erre valók a *batch fájl*ok, és természetesen többfelhasználós rendszerekben is, így például a UNIX-ban is.

10.3.4 Interaktív feldolgozás

Az interaktív feldolgozás igénye régen felmerült a felhasználók körében. Az egyes felhasználók a parancsaikra azonnal választ kapnak a rendszertől. A mikrogépek megjelenésével ez természetessé is vált.

Többfelhasználós rendszerekben az *időosztásos* üzemmóddal együtt jelent meg ez a lehetőség. A mai időosztásos rendszerek képesek interaktív és kötegelt feldolgozásra is.

10.3.5 Valósidejű feldolgozás

Speciális feladatok ellátására fejlesztették ki a valós idejű *(real-time)* operációs rendszereket. Az adatok a számítógépbe érzékelőkről érkeznek. Az operációs rendszer ezeknek az adatoknak a felhasználásával dönti el, hogy beavatkozik-e a rendszerbe, és ha igen, akkor milyen módon. Valós idejű rendszereket használnak például különböző ipari folyamatok, űrhajók irányítására, stb. A *valós* jelző arra utal, hogy a számítógép reagálásának, beavatkozásának egy meghatározott időn belül kell megtörténnie. Gondoljunk arra, hogy milyen következménnyel járna egy atomerőműben, ha a számítógép egy kritikus helyzetben késlekedne.

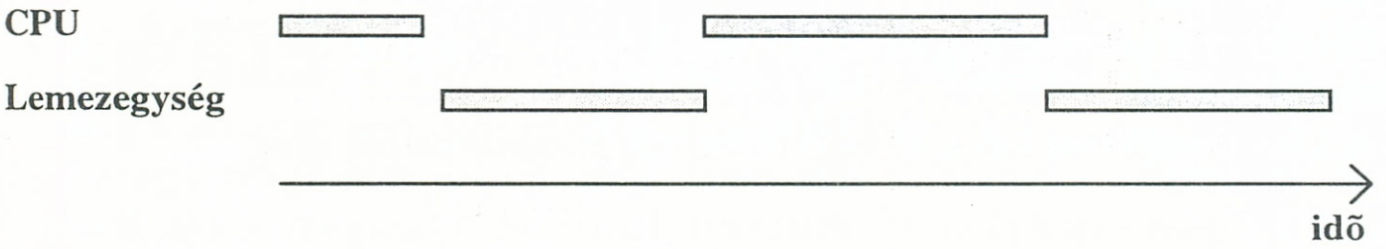
10.3.6 Monoprogramozás

Ezekben a rendszerekben egyidőben csak egy program futhat a számítógépen. Ilyen van a mikrogépeken is.

10.3.7 Multiprogramozás

Egy program végrehajtása során gyakran szükség van valamelyik periféria közreműködésére. A legtöbb periféria intelligens vezérlőegységgel rendelkezik. Például a lemezvezérlőnek olyan saját processzora van, amely önállóan végzi a feladatát: elhelyezi az író-olvasó fejet a megfelelő sávra, vezérli az adatátvitelt a lemezegység tárolójába stb. Eközben a CPU tétlen.

Az ilyen üzemmód időbeli ábrázolása:

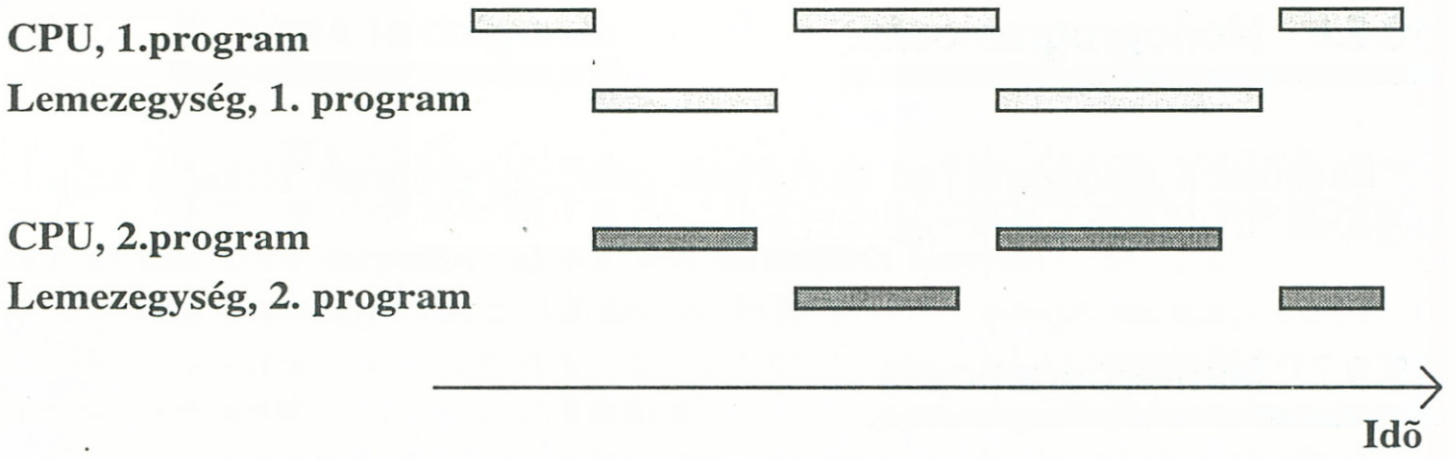


A processzor lényegesen gyorsabb, mint bármelyik periféria. Ilyen működés mellett a CPU az idő nagy részében tétlenül várakozik. Ennek a problémának a feloldására találták ki a *multiprogramozást.*

Multiprogramozás esetén általában több program is benn van a memóriában, közülük az aktívat (tehát amelyik éppen fut) *folyamatnak*, vagy *processnek* nevezzük.

A multiprogramozás lényege: amikor valamelyik periféria dolgozik, akkor a CPU nem várakozik tétlenül, hanem átkapcsol egy másik folyamat végrehajtására.

Például:



Az operációs rendszer egyszerre tartja a memóriában mind a két programot. A példa szerinti **1. program** elindult, és egészen addig folyamatosan működik, amíg nincs szüksége a lemezegységre. Ekkor az operációs rendszer várakozási állapotba helyezi az **1. programot**, és közben megindítja a lemezműveletet. Mivel a lemezművelethez nem szükséges a processzor működése, ezért a CPU foglalkozhat a **2. programmal**. Látható, hogy így a CPU sokkal jobban kihasználható.

Gondoskodni kell arról is, hogy a programok ne „garázdálkodhassanak" a memóriában, illetve a lemezegységen, tönkretéve a többi programot illetve azok adatait.

Felmerül a kérdés, hogy a CPU idejét hogyan osszuk meg az egyes folyamatok között.

A következőkben két alapvető lehetőséget ismerhettek meg.

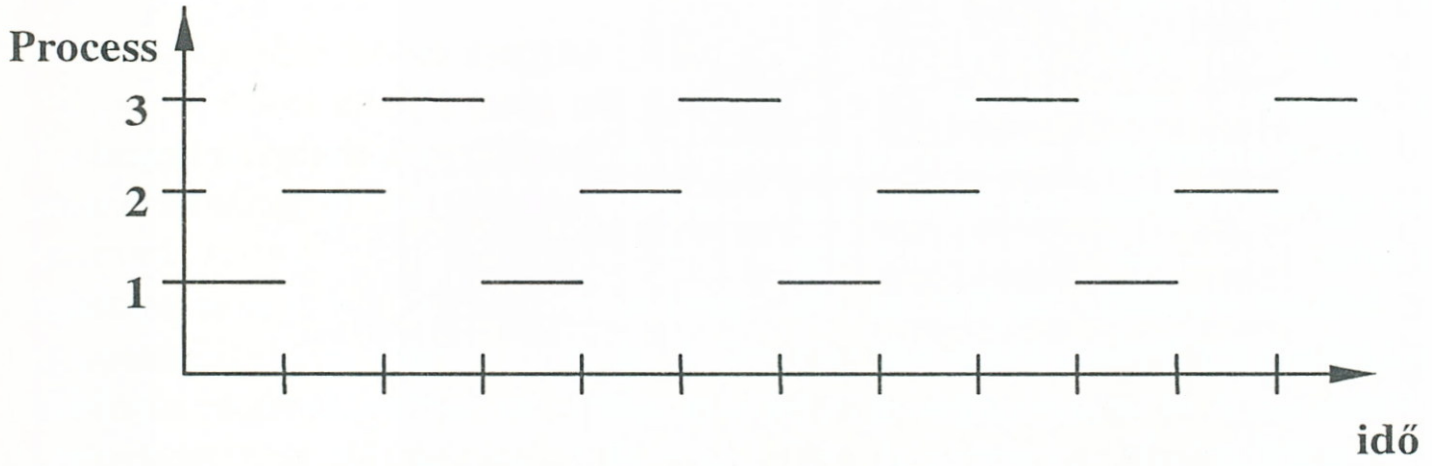
10.3.8 Időosztásos rendszer

Sok esetben igény van arra, hogy a felhasználó párbeszédes kapcsolatban legyen a számítógéppel. Ennek megvalósítására találták ki az időosztásos rendszert *(time-sharing)*.

Az operációs rendszer nagyon rövid, néhány századmásodperces időszeleteket oszt ki az egyes folyamatoknak.

A számítógép ennyi idő alatt nagyon sok műveletet képes végezni, így minden felhasználó úgy érzi, hogy a gép csak vele foglalkozik. Beír egy parancsot, úgy érzi, hogy a válasz azonnal megjelenik a képernyőn. Amíg a felhasználó beírja a parancsot, a CPU az időszeletek százezreit osztja ki.

Ha egyidőben túl sok felhasználó dolgozik, előfordulhat, hogy a gép túlterheltté válik, és a gép válaszideje erősen megnőhet. Ilyenkor csökkenteni kell a gép terhelését.



10.3.9 Prioritásos rendszer

A feladatokat valamilyen szempont szerint *fontossági sorrendbe* kell állítani. Ezután ezen sorrend szerint osztja ki a rendszer a CPU idejét az egyes folyamatok között. Ez a prioritás vagy fontosság sok mindentől függhet: például a folyamat periféria- vagy számítási igényeitől, vagy esetleg a programozó személyétől (ő a főnök) stb.

Ha egy magasabb prioritású – fontosabb – folyamat érkezik az éppen futónál, akkor megszakítják ezt a folyamatot, és az új kapja meg a CPU-t.

Előfordulhat, hogy egy folyamatot „kiéheztetnek" a többiek: csak vár-vár, miközben egyre-másra érkeznek a „fontosabb" folyamatok. Ennek elkerülésére szokták használni az *öregedés* fogalmát: bizonyos idő eltelte után a folyamat eggyel előbbre lép a fontossági sorrendben, majd újabb várakozás után megint lép, stb. Így előbb-utóbb sorra kerül, és végre megkapja a CPU-t, de közben visszakapja eredeti helyét a sorban, és vár, amíg megint rákerül a sor.

A multiprogramozás speciális esete a *multitasking*, amely már PC-ken is elérhető.

10.3.10 Multitasking

Egyes operációs rendszerek lehetővé teszik, hogy a felhasználó „egyszerre" futtasson több programot a számítógépen, amelyek mind a memóriában tartózkodnak.

Az egyidejűség azt jelenti, hogy a CPU nagyon rövid időszeleteket oszt ki az egyes programoknak.

A multitasking tulajdonképpen egy *időosztásos rendszert* feltételez, de az egyes programokat ugyanaz a felhasználó indítja el, ugyanazon a terminálon.

Megkülönböztethetjük az ún. *preemptív*, és az ún. *non-preemptív* multitasking operációs rendszereket. A non-preemptív rendszerekben (pl. Windows 3.1) az egyes programok – taskok – addig foglalják le a számítógép erőforrásait, ameddig akarják, és csak ezután adják át a vezérlést az operációs rendszernek.

A preemptív multitaskingra jellemző, hogy minden task meghatározott nagyságú (néhány ezredmásodperc nagyságrendű) időszelettel rendelkezik, és nem fordulhat az elő, hogy a task „önző módon" lefoglalja a számítógép erőforrásait.

Preemptív multitasking rendszerrel rendelkezik például a Novell DOS, OS/2, vagy a UNIX.

Megjegyzés:

Az MSDOS operációs rendszer a multitasking megvalósítására nem alkalmas.

11. HÁLÓZATOK

Úgy 30 évvel ezelőtt még azt gondolták a számítástechnikai szakemberek, hogy a leghatékonyabb alkalmazási módja a számítógépeknek az, ha ún. *számítóközpontokat* hoznak létre. A számítóközpontokban nagyteljesítményű számítógép dolgozza fel a legkülönbözőbb helyekről érkező adatokat, majd a feldolgozott adatok visszakerülnek (mágnesszalagon, floppyn stb.) az eredeti helyre. Ez a feldolgozás a számítógépek technikai fejlődése után elavulttá vált. Ma már ésszerűtlen, hogy minden adatot egyetlen helyen dolgozzanak fel, és a felhasználónak kelljen ezeket a számítógéphez elszállítani. Felmerült az a megoldás, hogy ezt a munkát sok önállóan működő, de egymással összekötött számítógép végezze el. Ezeket a rendszereket nevezzük *számítógépes hálózatnak.*

Természetesen az egyes gépek önállóan is működhetnének, de a számítógépeket alkalmazó szervezetek (vállalatok, bankok, stb.) számára előnyösebb, ha a különállóan működő gépek az adataikat egymással megosztják. Általánosabban itt a *szoftver-erőforrások* megosztásáról van szó. A másik megfontolás az, hogy sokkal olcsóbb úgy üzemeltetni gépeket, ha a *hardver-erőforrásokat* (háttértárolók, nyomtatók, stb.) megosztjuk közöttük.

Ezeket problémákat a hálózat segítségével lehet megoldani. Ezzel az egyes számítógépek erőforrásait mintegy megtöbbszörözhetjük.

Manapság a nagyobb hálózatok nagyon gyakran használt lehetősége az ún. *elektronikus levelezés*, amely segítségével az egyes, általában térben távol lévő felhasználók egymás számára információkat küldhetnek.

11.1 Kiterjedtség

A térbeli kiterjedtség alapján a hálózatok következő felosztását szokták alkalmazni:

- **LAN** (*Local Area Network*)
  Ez az ún. *helyi hálózat*, mi a későbbiekben erre fogunk nagyobb figyelmet fordítani. Egy olyan számítógépes hálózatról van szó, amely általában egy intézményen belül működik, néhány épületre terjed ki legfeljebb. Helyi hálózati operációs rendszer a *Novell IntraNetWare és Windows NT is.*

- **MAN** (*Metropolitan Area Network*)
  Szokták ezt városi hálózatnak is nevezni. A nevének megfelelően egy városon belüli hálózatról van szó, amely több helyi hálózat összekapcsolásával jön létre. Az egyes LAN-ok összekapcsolása történhet a hagyományos telefonhálózat segítségével vagy speciális, ún. üvegszálas vezetékekkel.

- **WAN** (*Wide Area Network*)
  Ez kiterjedt területű országos, vagy földrészek közötti hálózatot jelent. Itt az egyes MAN-ok összekapcsolásáról van szó. A MAN-ok összekötése lehet vezetékes, de történhet távközlési műholdak bekapcsolásával is. Ilyen például az országos méretű *Sulinet*, vagy a világméretű *Internet.*

11.2 Összeköttetések

Az egymással összekötött számítógépeket **gazdagépnek** *(host)* nevezik. Ezeket kommunikációs részhálózatok kötik össze, amelyek két részből állnak:

- Csatorna: az adatok rajta keresztül jutnak el a gépekhez.

- Kapcsolóelemek: a kommunikációt biztosítják az egyes hostok között.

A részhálózatokat a következőképpen csoportosíthatjuk az összeköttetés módja szerint:

- **Pont-pont összeköttetéses**
  A gépek közvetlenül össze vannak kötve. Amikor egy gép megkapja a másik által küldött üzenetet és az nem neki szól, akkor azt továbbadja egy másik pont-pont kapcsolaton keresztül. Az egyes gépek csak a pont-pont kapcsolaton keresztül tartanak kapcsolatot, ezért nincs szükség egymás azonosítására (címre).

- **Üzenetszórásos**
  Egy közös kommunikációs csatornára több host kapcsolódik, és közösen osztoznak rajta. Minden gép saját azonosítóval, **cím**mel rendelkezik. A csatornán mozgó üzenetek minden hosthoz eljutnak, de csak a címzett fogadja. Lehetőség van csoportcímzésre is, azaz üzenetet lehet küldeni egyszerre a gépek egy csoportjának.

A kommunikációs vonalak gazdaságos kihasználására több módszer is létezik:

- **Multiplexelés**
  Az átvivő közeget több csatornára osztjuk, így egymással párhuzamosan folyhat kommunikáció az egyes csatornákon.

- **Vonalkapcsolás**
  A vonalat csak akkor kapja meg az adó és a vevő, ha szükségük van rá, és ha az szabad is. Tipikusan ilyen a modemes adatátvitel.

- **Csomagkapcsolás**
  Az adatok kisebb részekre, csomagokra bontják, ezeket különkülön továbbítják. Az adó és a vevő számára úgy tűnik, hogy folyamatos az összeköttetés.

11.3 Átviteli közegek

Különböző módszerek léteznek az adatátviteli csatorna megvalósítására (átviteli közegek), amelyek többek között az átvitel sebességében is különböznek. Sebességként a másodpercenként továbbított bitek számát szokták megadni: **bit/s.**

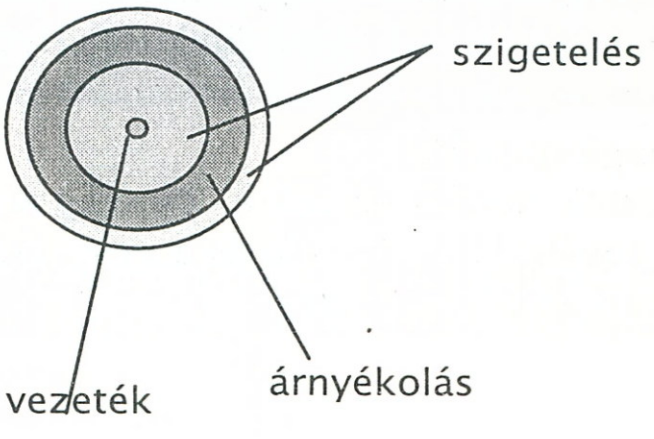
Alapvetően két típust különböztethetünk meg:

- **Vezeték nélküli**
  Ilyen például a rádióhullám, infravörös, ill. lézeres átvitel.

- **Vezetékes**
  Valamilyen kábellel (elektromos, optikai) hozzák létre az összeköttetést. Három elterjedt típussal találkozhatunk:

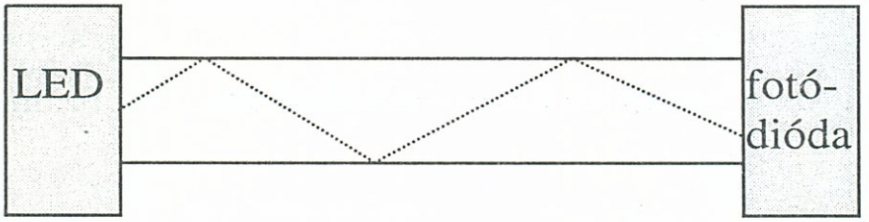
  - **Csavart érpár (UTP, STP)**
    Két szigetelt, egymásra spirálisan felcsavart rézvezeték alkotja. A külső elektromos mező elleni védelem miatt fémből készült sodrattal vehetik körül az érpárt (árnyékolás, ami a Faraday-kalitka elvén alapszik). Árnyékolást az STP típusnál alkalmaznak. A csavarásra azért van szükség, hogy a külvilág felé zavaró sugárzást ne bocsásson ki. Jellemző sebessége 10-100 Mbit/s. Általában a telefonoknál alkalmazott csatlakozóval látják el az UTP kábeleket.

  - **Koaxiális kábel**
    Hasonló felépítésű, mint a TV antennához használt kábel. Jellemző impedenciája 50Ω (Ethernet) , 75Ω (TV), 93Ω (Arcnet). Arcnet és Ethernet esetén ún. BNC csatlakozókat használnak. Sebessége: 10-100 Mbit/s.



➢ **Üvegszálas kábel**

Egy üvegszál belsejében az adatokat fény segítségével továbbítják. A fényforrás egy lézer LED, a fényérzékelő pedig egy fotódióda. Az üvegszál nyilván nem tökéletesen egyenes (sőt lehet, hogy erősen eltér ettől), a fény mégsem jut ki belőle. Hogyan lehetséges ez? A fizikában tanult *teljes visszaverődés* segítségével: Ha az üveg-levegő határhoz érkezik a fény és a beesési szög nagyobb egy meghatározott határszögnél, akkor a fénysugár mint egy tükörről visszaverődik az üvegbe.

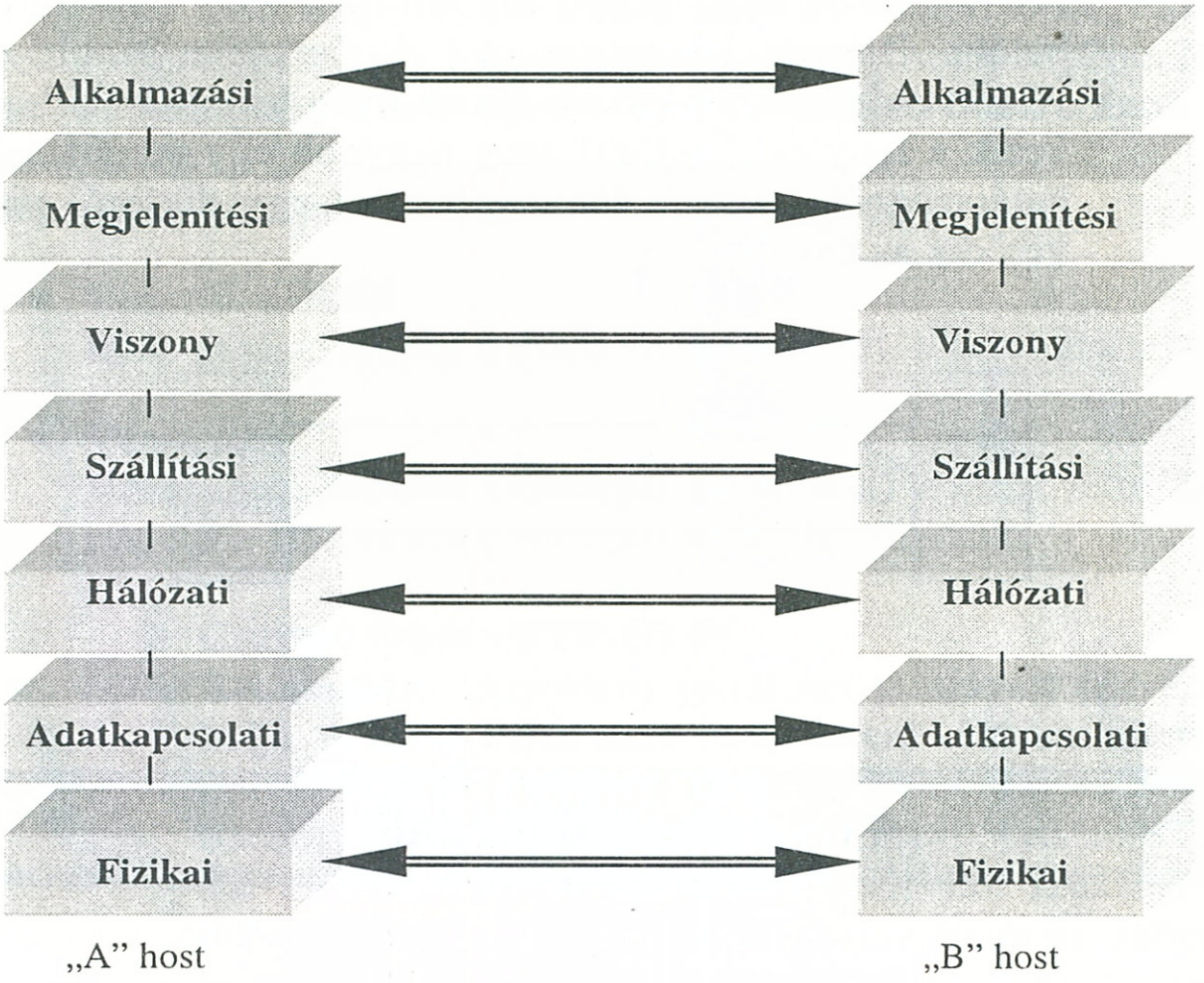


11.4 Az OSI modell

A Nemzetközi Szabványügyi Szervezet (ISO) ajánlást dolgozott ki a számítógépes hálózatok felépítésére vonatkozóan, amit OSI (*Open System Interconnect)* modellnek hívnak. A hálózat egyes részeit **réteg**ekbe szervezik, amelyek egymásra épülnek.

Szint	Réteg
7	Alkalmazási
6	Megjelenítési
5	Viszony
4	Szállítási
3	Hálózati
2	Adatkapcsolati
1	Fizikai

Minden rétegnek sajátos feladata van a hálózati kommunikációban.



Amikor két gép kapcsolatba kerül, akkor az egyik gép n. rétege a másik gép n. rétegével kommunikál. A kommunikációnál használt szabályokat **protokoll**oknak nevezik. Az egyes rétegek elfedik az alattuk levő réteget a fölöttük lévőtől, ami azt jelenti, hogy a fölötte lévő réteg tervezésekor nem kell az alsó réteg működését ismerni. A szomszédos rétegek közötti kapcsolatot ún. **réteginterfész**ek írják le.

Az egyes rétegek feladata a következő:

1. **Fizikai réteg**
   A továbbítandó biteket fizikai jelekké alakítja és elküldi a csatornán. Az üzenetnek ezen a szinten nincs szerkezete, csak biteket „lát". Az üzenet vételekor a fizikai jeleket bitekké alakítja.

2. **Adatkapcsolati réteg**
   A legfontosabb feladata az adó és a vevő közötti pont-pont kapcsolat segítségével az üzenet továbbítása. Ehhez az adatokat **keretekbe** foglalja. A kereteket kiegészítő információkkal látja el, majd továbbítja őket. A vevő a sikeres vételt nyugtázza.

3. **Hálózati réteg**
   A különböző típusú hálózatok összekapcsolásával olyan hálózatot kapunk, amelyben az egyes eszközök azonítása másképpen történhet, azaz más-más szabályok szerint működik. Ezért egy 3. rétegbeli címre is szükség van, amely egységes, és független a konkrét fizikai hálózattól. Nagy hálózatoknál az adó és vevő között több út is lehetséges, ezért meg kell határozni az útvonalat. Ezt hívják útvonalválasztásnak *(routing)*.

4. **Szállítási réteg**
   A beérkező csomagokat össze kell fűznie, és továbbítania a *Viszony réteg* felé. A küldendő adatokat **csomag**okba foglalja és továbbítja *Hálózati réteg* felé.

5. **Viszony réteg**
   Az egyes gépek közötti felhasználói viszony létrehozása a feladata. Kiépíti, illetve szinkronizálja közöttük a kapcsolatot.

6. **Megjelenítési réteg**
   Az egyes számítógépeken különböző lehet az adatoknak ábrázolása, és ennek a rétegnek a feladata az ebből származó problémák kiküszöbölése, továbbá az adattömörítés, az adatok titkosítása.

7. **Alkalmazási réteg**
   Ezen a szinten a hálózati szolgáltatásokat igénybe vevő programok működnek.

11.5 Hálózatok összekapcsolása

A különböző hálózatokat többféleképpen is összekapcsolhatjuk:

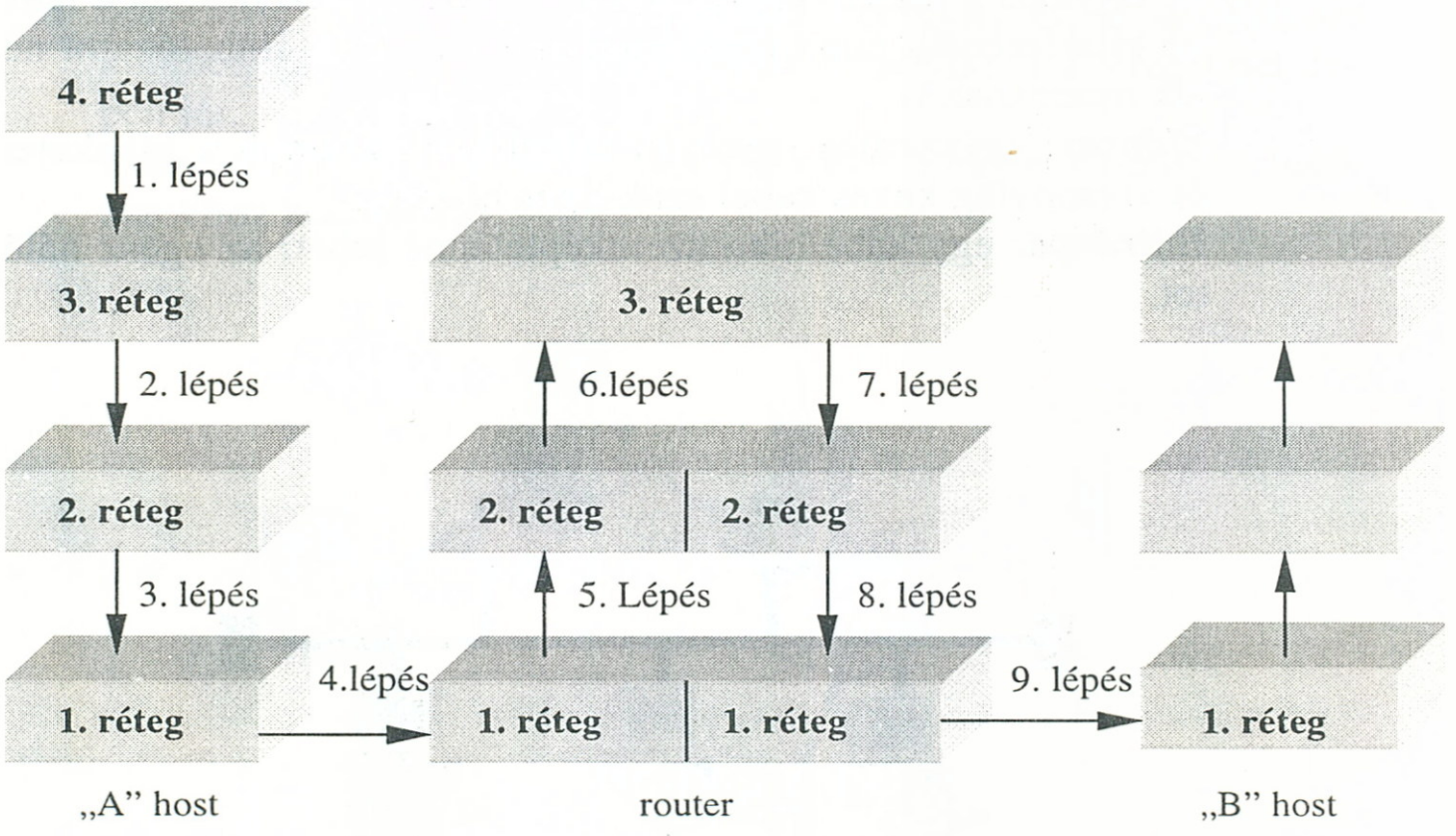
- **A fizikai rétegen keresztül (Repeater)**
  Azonos típusú hálózatokat kötnek így össze, a fizikai réteg tulajdonságainak megfelelően itt csak a bitek másolása történik, jelismétlőnek is nevezik.

- **Adatkapcsolati rétegen keresztül (Bridge)**
  Azonos típusú hálózatok összeköttetésére szolgál. A célja az, hogy csak akkor tesz át egy csomagot egy másik részhálózatba, ha a címzett ott található. Így az egyes részhálózatok forgalma nem zavarja egymást. A bridge egy táblázat segítségével végzi a munkáját, amelyet „menet közben" épít fel, a tapasztalatok alapján: egy új csomag érkezésekor végigkérdezi az összes gépet, és amelyik „magára ismer" a címzettben, annak az adatait bejegyzi a táblázatba. Közös az adatkapcsolati réteg a bridge-ben.

- **A hálózati rétegen keresztül (Router)**
  A router két vagy több, esetleg különböző típusú fizikai hálózatot köt össze. Az a feladata, hogy a beérkező csomagokat a 3. rétegbeli címnek megfelelően továbbítsa. (A 3. rétegbeli címet **virtuális cím**nek is nevezik.) Mivel a csomagok továbbítása ténylegesen a fizikai réteg szintjén történik, ezért szükség van a virtuális címnek a **fizikai cím**re (1. rétegbeli) való megfeleltetésre.
  A routerben a hálózati réteg közös.
  Példa: Legyen a két host neve „A" ill. „B". A csomagküldés lépései a következők lesznek:

  1. Az „A" 3. rétege a felette levő szintről megkapja a továbbítandó csomagokat.
  2. A csomagot járulékos információkkal (fejrésszel) látja el, amiben elhelyezi a virtuális címet, és átadja a 2. rétegnek.

3. A 2. réteg saját fizikai hálózatának megfelelő keretet hoz létre, amiben elhelyezi a router fizikai címét, és a küldendő adatokat, majd átadja az 1. rétegnek.
4. A keret a csatornán a routerre érkezik.
5. A routeren kialakított adóoldali 1. réteg a 2. rétegnek továbbítja a csomagot.
6. A routeren kialakított adóoldali 2. réteg eltávolítja az adóoldali részhálózat fejrészét, és az adatokat továbbítja a 3. rétegnek.
7. A router 3. rétege a cím alapján megállapítja, hogy a címzett a másik részhálózaton található, és a routeren kialakított vevőoldali 2. rétegének továbbítja a csomagot.
8. A 2. réteg létrehozza vevőoldali részhálózatnak megfelelő keretformátumot, és elhelyezi benne a címzett számítógép fizikai címét, majd átadja az 1. rétegnek.
9. A fizikai cím alapján elküldi a csomagot a címzettnek.



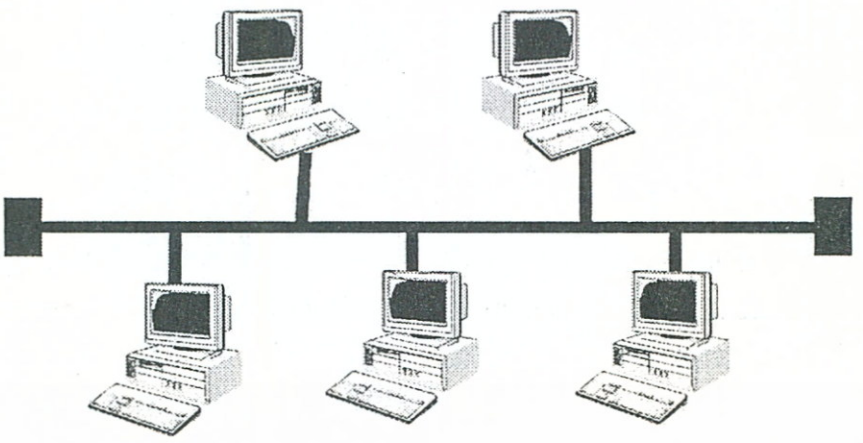
Nagyobb hálózatok esetén természetesen bonyolultabb a helyzet, több részhálózaton keresztül jut el a csomag a címzetthez.

11.6 Hálózati topológiák

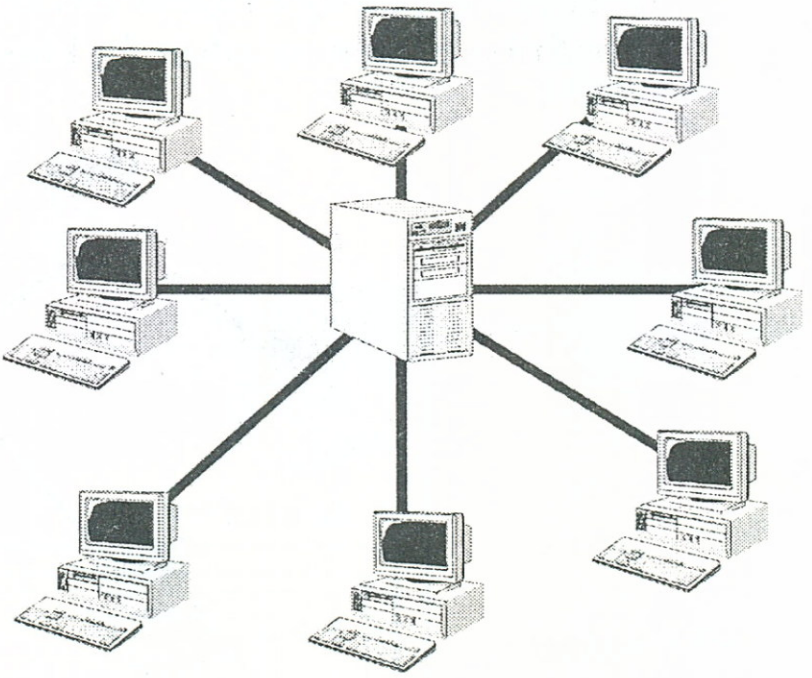
A hálózatra jellemző, hogy az egyes gépek milyen módon kapcsolódnak össze. Ez a hálózat ún. topológiája. Az egyes állomások általában egyenrangúak, és mindegyiknek saját, egyedi azonosítója, címe van, amellyel azonosítani lehet őket.

A következő hálózati topológiákat különböztetjük meg:

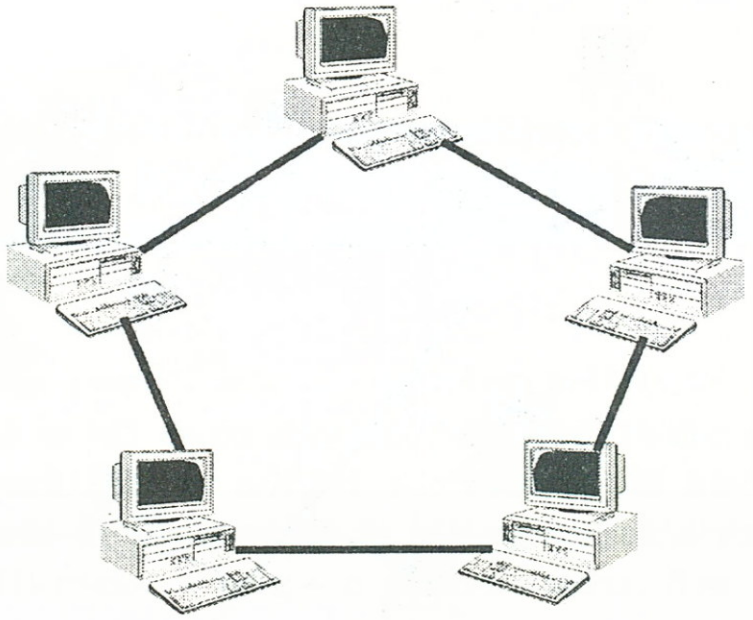
- **Sín-hálózat**
  A munkaállomások egy vezetékre csatlakoznak, az adatátvitel is ezen a vezetéken történik. A vezetékre adott információ a teljes vezetéken végigfut, és az operációs rendszer biztosítja, hogy a megfelelő állomás is megkapja.
  A vezeték végét meghatározott nagyságú ellenállással le kell zárni, ugyanis a kábel végéről a jelek visszaverődnének, és a létrejövő interferencia miatt problémát okoznának. A sínhálózatot nevezik buszosnak is.
  Előnyei: egyszerűen, gyorsan telepíthető, könnyű a hibakeresés, és viszonylag kevés kábel szükséges hozzá.
  Hátránya: egy kábelhiba üzemképtelenné teheti az egész hálózatot.



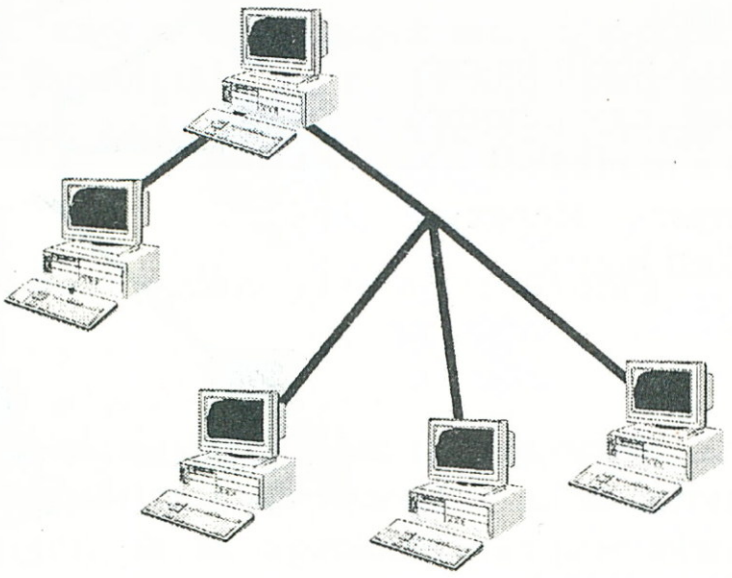
- **Csillag-hálózat**
  Az egyes munkaállomások külön vezetékkel csatlakoznak a szerverhez.
  Előnye: Kábelhiba esetén csak egy állomás esik ki a munkából.
  Hátránya: Rengeteg kábel kell hozzá.



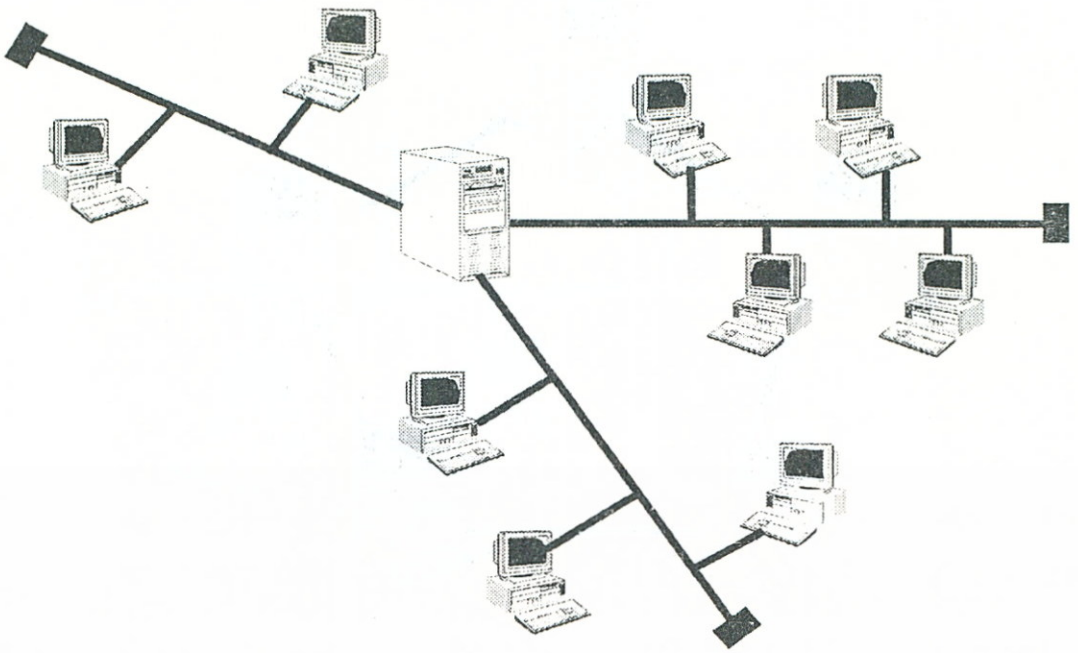
- **Gyűrűs hálózat**
  Az állomások egy zárt görbe mentén helyezkednek el. A zárt kábelben egy ún. *vezérlőjel* állandóan körbejár, és az az állomás küldhet, illetve vehet át információt, amelyiknél a vezérlőjel éppen jár.
  Hátránya, hogy nagyon érzékeny a kábel hibájára (a buszos hálózatokhoz hasonlóan).



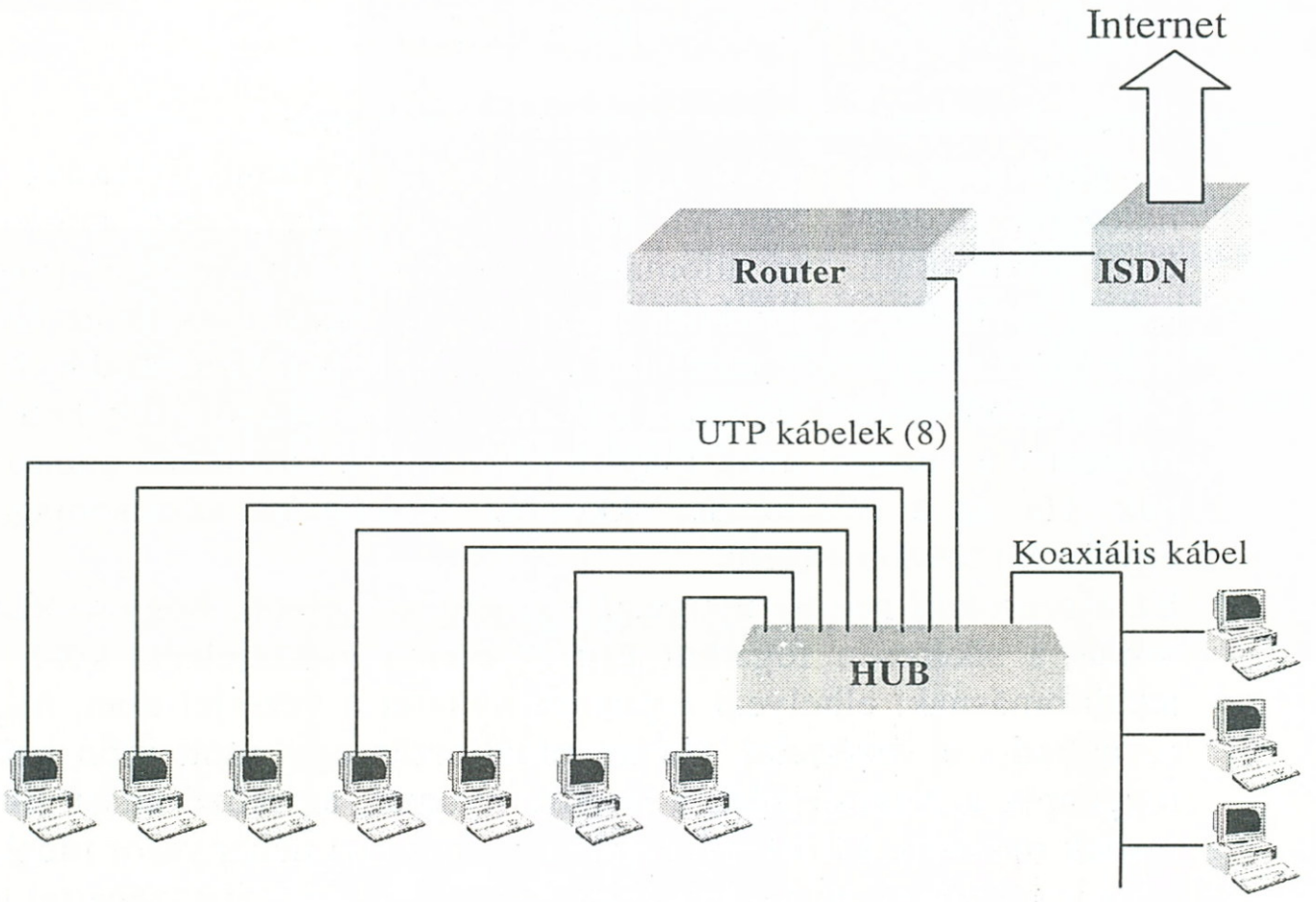
- **Fa topológia**
  Az egyes gépek egy fagráf csúcsaiban helyezkednek el. Kábelhiba esetén az adott ág kiesik.



- **Vegyes hálózat**
  Az előző topológiák kevert alkalmazásáról van szó. Például: sín- és csillagtopológiák együttes alkalmazása.



A Sulinet programban szállított *Internet laboratórium* gépei is Ethernet hálózati kártyákat tartalmaznak. Az elvi felépítése a következő:



HUB: Jelerősítő és elosztó. Csillag kapcsolást lehet a segítségével kialakítani.

11.7 Közeg-hozzáférési módszerek

Üzenetszórásos csatornájú hálózatokra jellemző, hogy *egy* adott csatornát *több állomás* is közösen használ. A vétel esete egyszerű: minden állomás veszi mindenkinek az adását, a cím alapján mindegyik el tudja dönteni, hogy neki szól-e az üzenet. A *közeg-hozzáférés* alatt azt értjük tehát, hogy az egyes állomások hogyan tudnak *adni*. Ezt különböző *protokollok* írják le.

A protokoll biztosítja, hogy a közös kábelt minden állomás a saját adatforgalmazásának az idejére kisajátíthassa. Biztosítani kell azt, hogy az

egyes állomások üzenetei ne ütközhessenek egymással, mert akkor ezek összekeveredhetnek, megsérülhetnek. Ha mégis előfordul ilyen ütközés, akkor meg kell ismételni az üzeneteket.

Két elterjedt hálózati protokollt ismerünk meg.

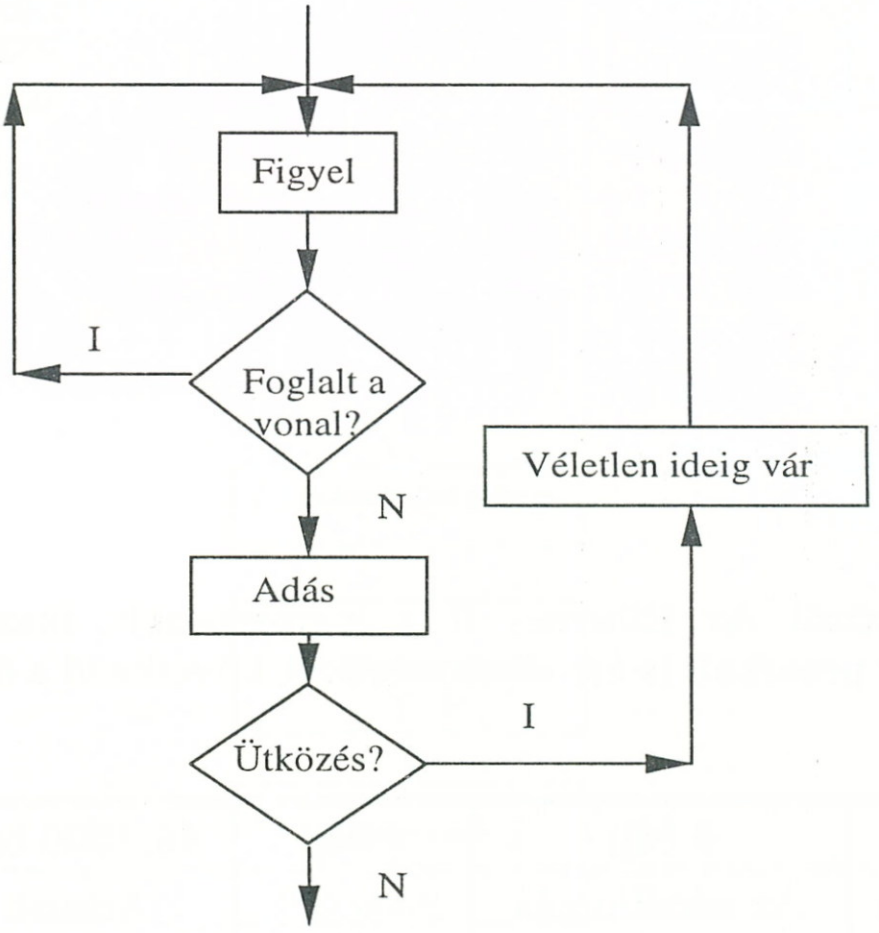
- **Token Passing:** *adási jog továbbításos protokoll*
  Az állomások egy logikailag gyűrűs (nem feltétlenül fizikailag is) hálózatot alkotnak. Minden állomásnak van egy *címe*, amellyel lehet azonosítani. Az állomások között egy ún. *vezérjel* (vagy *token*) jár körbe, a címeknek megfelelően. Adást csak az az állomás kezdeményezhet, amelynél a vezérjel éppen tartózkodik. Ekkor az adott állomás az általa küldendő adatcsomagot „felteheti" a hálózatra a vezérjel után, majd tovább adja a következő állomásnak. Ha az a célállomás, akkor az fogadja az adatcsomagot, egyébként tovább engedi.
  Ez a *protokoll pozitív* nyugtázásos, ami azt jelenti, hogy a célállomás a sikeresen fogadott üzenet esetén visszajelzést küld az adóállomásnak: elhelyezi a pozitív nyugtát a vezérjel után, majd továbbadja a vezérjelet. Ha az adóállomás egy adott időn belül megkapja a visszajelzést, a pozitív nyugtát, az előző adást sikeresnek minősíti. Ellenkező esetben az üzenetet néhányszor megkísérli továbbítani, ha továbbra sem sikerül, akkor hibaüzenettel leáll.
  Ilyen protokollt használ például az *ARC-Net* hálózat.

- **CSMA/CD:** *ütközésfigyeléses protokoll*
  Az állomás, mielőtt elküldené a kábelen az üzenetét, „belehallgat" a csatornába, és figyeli, hogy nincs-e már egy másik állomás által elküldött üzenet rajta. Ha úgy érzékeli, hogy szabad a kábel, akkor elküldi az üzenetet. Természetesen előfordulhat, hogy két állomás így is egyszerre küld üzenetet, így az üzenetek „összeütköznek" és megsérülnek. Az adó ezt érzékeli, hiszen ő is veszi saját adását, és összehasonlítja az elküldött üzenettel. Ha nem egyeznek meg, akkor az azt jelenti, hogy egy másik állomás is ad. Ezt hívják **ütközés**nek. Ilyenkor az üzenet küldését meg kell ismételni. Mindkét állomás egy ideig vár, majd újra megpróbálják az üzenet küldését. (A kivárási idők véletlenszerűek.)

Megjegyzés:

Ha a hálózat terhelése nagy (sok állomás van, és nagy az adatforgalom), akkor a hálózat erősen lelassulhat, hiszen viszonylag sok ütközés fordulhat elő.

A CSMA/CD legnagyobb hibája az, hogy tulajdonképpen olyan esetre „találták ki", amikor a munkaállomások elsősorban egymás között bonyolítanak forgalmat, nem pedig egy központi géppel, egy szerverrel. A legtöbb helyi hálózatban pedig éppen ez az utóbbi eset a jellemző. Így a szerveren a lemezműveletekre vonatkozó kérelmek sokszor ütköznek.



Ilyen protokollt használ például az *Ethernet* hálózat.

11.8 Az Ethernet hálózat

Az Ethernet hálózat logikailag sín topológiájú, üzenetszórásos, CSMA/CD protokollal. Fizikailag lehet sín (50Ω-os koaxiális kábellel), csillag (UTP kábellel), és vegyes a kettő összekapcsolásával. Az adatátviteli sebesség 10 Mbit/s – 100 Mbit/s.

Az Ethernet kártyának 6 bájtos fizikai címe van, amit gyárilag „égetnek" bele. Az egyes gyártók egyeztetik az általuk alkalmazott címtartományokat.

A gyártó azonosítója	A kártya azonosítója
3 bájt	3 bájt

Az Ethernet hálózati kártya az 1. és a 2. réteget valósítja meg. Az adatkapcsolati rétege több keretformátumot is ismer:

- Ethernet 802.3
- Ethernet 802.2
- Ethernet II

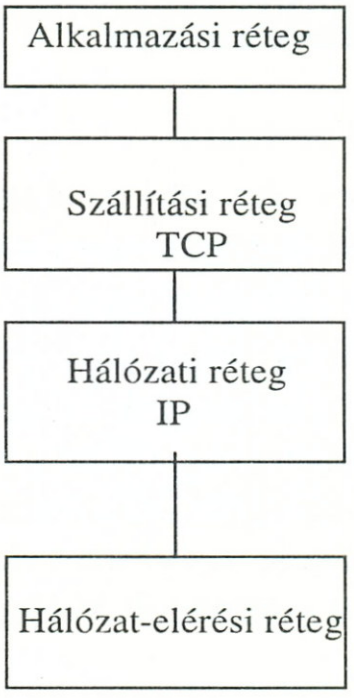
Ezek közül Az Ethernet II a legfontosabb, hiszen az internetes TCP/IP protokoll is ezt alkalmazza. A következő a felépítése:

6 bájt	6 bájt	2 bájt	46-1500 bájt	4 bájt
A célállomás fizikai címe	Az adóállomás fizikai címe	Protokoll azonosító	Adatok	Ellenőrző összeg

11.9 TCP/IP hálózatok

Az Internet egy világméretű számítógépes hálózat, amely kisebb hálózatok összekapcsolódásából áll. A hetvenes évek elején az amerikai védelmi minisztérium megbízásából fejlesztették ki az ARPANET nevű hálózatot. A kutatók azt vizsgálták, hogyan érhetők el az egyes gépeken levő információk még akkor is, ha a hálózat gépei közül néhány üzemképtelenné válik. Korábban az egymással kommunikáló gépeket telefonvonal segítségével, közvetlenül kötötték össze. Az új hálózatban csomagkapcsolt adatátvitelt használtak. Az ARPANET hálózat később átalakult: először egyetemek, kutatóintézetek kapcsolódtak be, majd a hálózat katonai jellege szűnt meg. Az új hálózat lett az Internet, amelyben ma már több tízmillió számítógép van összekötve a világ minden részén.

Az Internet az ún. TCP/IP protokollokra épül. Ha összehasonlítjuk az OSI modellel (amit csak később alakítottak ki), akkor azt látjuk, hogy nem illeszkedik arra tökéletesen, négy réteget tartalmaz. A TCP/IP a UNIX operációs rendszerhez erősen kötődik, hiszen először ebben valósították meg.



11.9.1 Hálózat – elérési réteg

Az OSI modell szerinti fizikai és adatkapcsolati rétegnek felel meg. Az adó fizikai réteg csak a cél fizikai címének ismeretében tudja a csomagot elküldeni. A cél címét viszont a 3. rétegből kapja (virtuális cím), amelyhez meg kell keresni a megfelelő fizikai címet. Ethernet hálózat esetén –mint láttuk- ez egy 6 bájtos azonosító, míg a virtuális címet **IP címnek** nevezzük. (Az IP címről a későbbiekben tudhatsz meg többet.) Üzenetszórásos hálózatokon az **ARP** *(Address Resolution Protocol)* protokollal lehet megoldani a problémát.

1. lépés: Az adó kiküldi a kérdését az üzenetszórásos hálózatra: az adott IP címhez melyik fizikai cím tartozik.
2. lépés: Az üzenetet minden gép megkapja, és a kapott IP címet összehasonlítja a sajátjával. Ha megegyezik, akkor a saját fizikai címét küldi vissza.
3. lépés: Az adó a fizikai cím birtokában elküldi a csomagokat a címzetthez.

Az adó nem dobja el a megszerzett címet, hanem eltárolja.

11.9.2 Hálózati réteg (IP protokoll)

Az **IP** *(Internet Protocol)* protokoll:

- *Összeköttetés mentes*: Az adó az üzenetet csomagokra bontja. A csomagok egymástól függetlenül mozognak a hálózatban.
- *Nem megbízható*: A folyamat során csomagok veszhetnek el, illetve duplázódhatnak meg, rossz sorrendben érkezhetnek meg. Az IP nem ellenőrzi, hogy a csomagok megérkeztek-e.
  Az összeköttetés mentesség miatt gyors a csomagok átvitele. A hálózat rugalmas, könnyen alkalmazkodik a hálózatban fellépő akadályokhoz (torlódás, meghibásodás), ugyanis nincs előre lefoglalt útvonal, így a csomagok különböző utakon juthatnak a célhoz.

11.9.3 Szállítási réteg (TCP protokoll)

Az IP megbízhatatlansága miatt szükség van egy megbízható protokollra. A **TCP** *(Transmission Control Protocol)* protokoll:

- *Összeköttetéses*: A két végponton a TCP rétegek kommunikálnak egymással.

- *Megbízható*: Az IP szintű csomagtovábbítás során fellépő hibákat kell kiküszöbölni. Ehhez a csomagokat számozza, így a célállomáson akkor is össze lehet rakni a csomagokat, hogyha hiányzik csomag, vagy a sorrend „elromlik". Az adás után elindít egy időzítőt („stoppert"), és ha megadott időn belül nem érkezik **nyugta** a célállomástól, akkor elveszettnek veszi a csomagot, és újra elküldi.

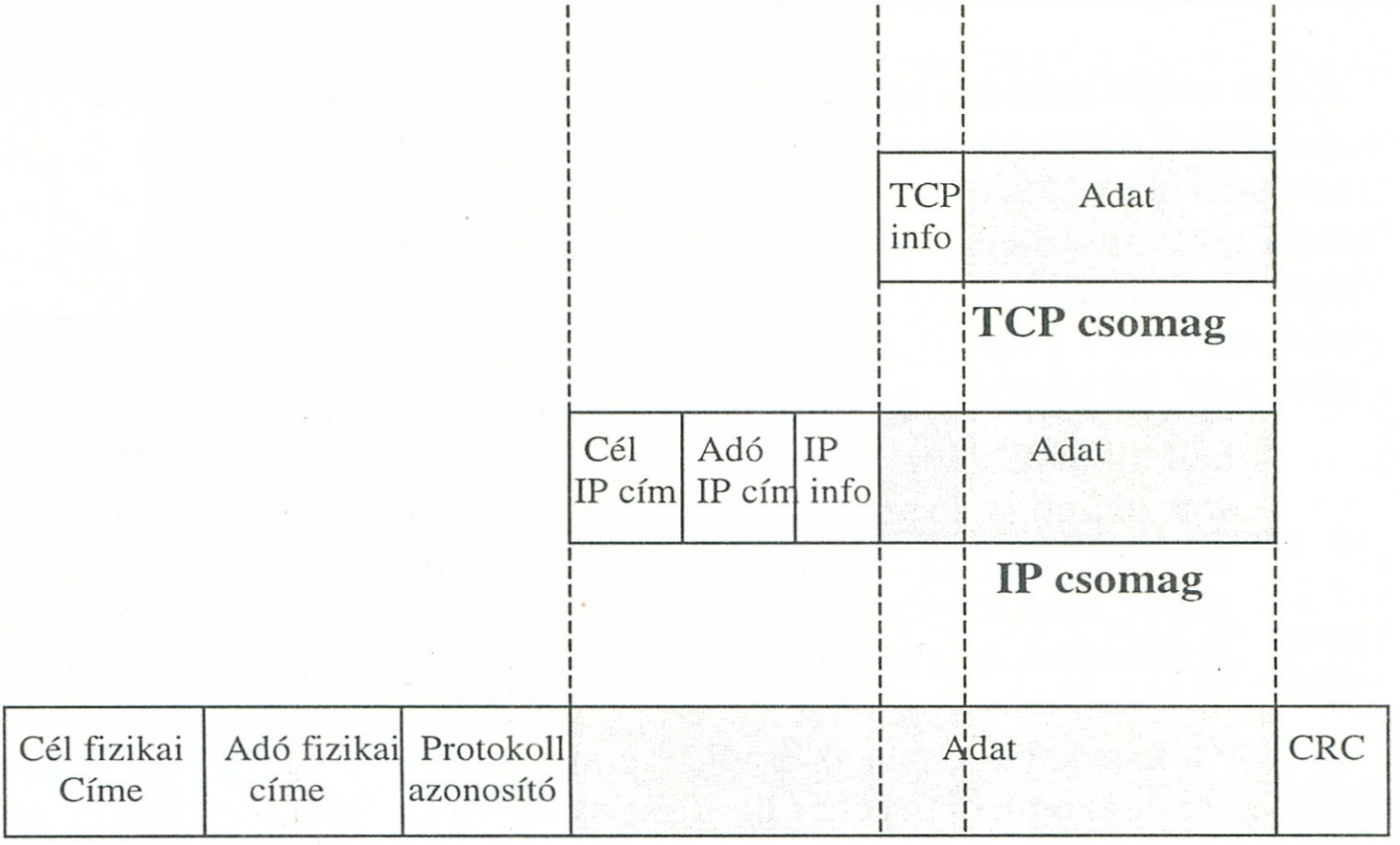
A TCP a kapcsolatot az adó, a cél IP címe, és az egyes gépek **TCP port** száma alapján azonosítja. Tehát ha a szerveren (kiszolgálón) egy program valamilyen szolgáltatást akar nyújtani, akkor ehhez hozzárendel egy portszámot, ami alapján a kliens azonosítani tudja a szolgáltatást. Pl. az FTP szokásos portszáma 21, a HTTP portszáma pedig 80. Ha a kliens élni akar ezzel a lehetőséggel, akkor a saját gépén szintén lefoglal egy portszámot. Miután létrejött a kapcsolat, a kiszolgáló más kliensekkel is tud „foglalkozni", hiszen igaz, hogy a kiszolgáló portszáma ugyanaz, de az egyes kliensek portszáma más.

Megjegyzés:

*A TCP protokoll helyett használhatjuk az **UDP** (**U**ser **D**atagram **P**rotocol) protokollt, ha az üzenet elfér egyetlen csomagban. Ilyenkor nincs szükség a bonyolultabb TCP-re. Ha megadott időn belül nincs válasz, akkor megismétli az adást. Az UDP is használ portszámokat, de más-mást jelentenek ugyanazok a portszámok.*

A TCP protokoll tehát a két végponton működik. Az IP-t viszont minden közbülső routeren meg kell valósítani.

A csomagok képzése tehát a következő fázisokon megy keresztül:



Ethernet II keret

- TCP info: Az elküldendő adatot a TCP saját információkkal látja el (pl.: a csomag sorszáma, ellenőrző összeg, a célgépen a szolgáltatás portszáma, stb), de nem tartalmaz semmilyen címet. Az ellenőrző összeghez, az adat bájtjaihoz számokat rendel és összeadja őket. A célnál ugyanezt elvégezve ugyanazt a számot kell kapni hibátlan átvitel esetén.
  Az így kapott csomagot adja át az IP-nek, amely az egészet adatnak tekinti.

- IP info: Az elküldendő adatot az IP saját információkkal látja el. Pl.: a csomag teljes hossza, az üzenet azonosítója (a teljes üzenet minden csomagja ugyanazt az azonosítót kapja).

- CRC: Ellenőrző összeg.

- Protokoll ID: Protokoll azonosító (Itt: Ethernet II).

11.9.4 Alkalmazási réteg

Ez tartalmazza a felhasználói, és a hálózat-alkalmazási programokat. A legfontosabb alkalmazási protokollok a következők:

- **FTP** *(File Transfer Protocol)*
  Állományok átvitelére szolgál, alapvető fájl és könyvtár műveleteket tesz lehetővé a helyi és távoli gép között.
  Az átvitel két üzemmódban működhet: ascii és binary. Az ascii 7 bites ASCII kódot használ (az angol nyelvhez elegendő!) szöveges állományok átvitelére. Így gyorsabban lehet átvinni az állományokat. Binary mód esetén 8 bites az átvitel.
  A távoli gépre, illetve onnan csak akkor tudunk állományokat átvinni, ha az adott gépre van felhasználói jogosultságunk (bejelentkezési név, jelszó, stb). Sok helyen nyújtanak ún. **anonymous ftp** szolgáltatást. Itt nem kell külön felhasználói jogosultsággal rendelkezni, hanem bizonyos, előre behatárolt lehetőségekkel élhetünk.

- **GOPHER**
  Szöveges információ keresése hierarchikus felépítésű adathalmazban. Ma már egyre kisebb a jelentősége. Egy menürendszeren keresztül lehet a keresett információhoz eljutni.

- **HTTP** *(HyperText Transport Protocol)*
  Hypertext dokumentumok keresésére szolgál a WWW-n *(World Wide Web)*. A dokumentumok szerkezetét egy speciális nyelvvel, a **HTML** *(Hyper Text Markup Language)* segítségével lehet megadni. A dokumentumok (oldalak) különböző hivatkozásokat is tartalmazhatnak: képek, hátterek, hang, film, link (más lapra vonatkozó hivatkozás).
  A HTML lapcímét az **URL** *(Uniform Resource Locator)* segítségével adhatjuk meg:

  protokoll://szervercím/elérési út/állománynév

  A protokoll ftp, gopher, http lehet.

A WWW **kliens/szerver** (ügyfél/kiszolgáló) alapú. A kliensek kérelmekkel fordulnak a szerver felé, az pedig–ha lehetséges- eleget tesz annak.

A HTTP protokoll **állapotmentes,** a szerver és a kliens között a kapcsolat csak az átvitel idejére jön létre, kiszolgálás után a kapcsolatot bontja a kiszolgáló. Ha kliens több kérést is küld a szervernek, az egyes kéréseket teljesen függetlenül kezeli egymástól. A HTTP kapcsolat lépései:

- A kliens kapcsolatot kezdeményez
- A kérés elküldése
- A szerver válaszol
- A szerver bontja a kapcsolatot

A kapcsolat során csak egy dokumentumot ad át a szerver, azaz ha egy HTML oldal több hivatkozást tartalmaz (kép, hang, film, stb.), akkor mindegyiknek külön fel kell építeni a kapcsolatot, majd minden alkalommal a szerver bont. Az állapotmentesség következtében a szerver nem is érzékeli, hogy ugyanannak az oldalnak a részeiről van szó.

- **SMTP** *(Simple Mail Transfer Protocol)*
  Elektronikus levelek küldését és fogadását teszi lehetővé. A leveleket elektronikus postahivatalok (levelező szerverek) kezelik. Ezekben a postahivatalokban postafiókok hozhatók létre.
  Az elektronikus levélcímek felépítése a következő:

postafiók@gépcím

Az elektronikus levélen keresztül közvetlenül csak 7 bites átvitel lehetséges, azaz csak 0-127 ASCII kódú karakterek használhatók. Így bináris állományok közvetlen átvitele nem lehetséges. Ezért különböző kódolási módszereket alkalmaznak.

➢ **UUENCODE/UUDECODE**

A UUENCODE a bináris állományt 7 bites szöveggé alakítja: vesz a fájl elejéről kezdve sorban 3x8 bitet, és azt 4 db 6 bites csoportra osztja. Mivel a vezérlő karakterek ASCII kódja 0-31-ig terjed, ezért, hogy csak normál karakterek szerepeljenek benne, a 6 bites csoportokhoz hozzáad 32-t. Így 4 db 7 bites csoportot kapunk, amelyek képernyőn is megjeleníthető karaktereket tartalmaznak.

A UUDECODE a kódolt állományt dekódolja, visszaalakítja az eredeti formába.

➢ **MIME**

Ezzel akár magyar ékezetes betűket tartalmazó levelet is küldhetünk, illetve olvashatunk.

Több kódolási módszert is alkalmazhatunk:

- 8 bites ASCII szöveg. Nem felel meg az eredeti szabványnak.
- Bináris állományokra, **base64** kódolás. A UUENCODE-hoz hasonlóan itt is a 3x8 bitet 4x6 bitté alakítja. (6 biten 0 és 63 ($2^6 - 1$) közötti számokat lehet felírni.) Ezt a következőképpen kódolja tovább: 0-nak „A" felel meg, 1-nek „B", 2-nek „C", ... 25-nak „Z", „a"-nak 27 ...51-nek „z", 52-nek „0", ... 61-nek „9", 62-nek „+", 63-nak „/". Így az üzenetet 0 és 63 közötti ASCII kódú karakterekből álló üzenetként továbbítódik.
- Bináris állományokra, **quoted–printable** kódolás. Olyan üzeneteknél, amelyek „majdnem" tiszta 7 bites ASCII szövegek, nem elég hatékony a base64 kódolás. 0 és 127 közötti karaktereknél nincs kódolás, a 127 felettieknél a következőképpen járnak el:
  egyenlőségjel (=) után az ASCII kód két hexadecimális karakterével helyettesítik (tehát összesen három karakter).

- **TELNET**
  Távoli gépre való bejelentkezést tesz lehetővé. Az esetleges távolságból adódó késleltetéstől eltekintve úgy érezhetjük, hogy a távoli gép terminálja előtt ülünk. A legtöbb esetben a távoli gépen UNIX operációs rendszer fut.
  Általában szükség van arra, hogy a távoli gépre felhasználói jogokkal rendelkezzünk a bejentkezéshez, de itt is van **anonymous** szolgáltatásra lehetőség. Ezt nyújtaja például az Országos Széchenyi Könyvtár, KLTE Könyvtára, stb).

11.10 Címzés az Interneten

Az IP protokoll virtuális címként az ún. IP címet használja, amely egy 32 bites szám. Az IP cím nem feltétlenül számítógépet jelöl, hanem hálózati csatlakozási pontot (más néven **végpont**ot). Tehát, ha egy gépben két hálózati kártya van, akkor mind a két kártyának külön IP címe lesz.

Az Internet routerekkel összekötött részhálózatokból áll, tehát ha egy csatlakozási pontot azonosítani akarunk, akkor meg kell adnunk, hogy mi annak a hálózatnak a címét, ahol az található, és azon belül meg kell adni a csatlakozási pont címét.

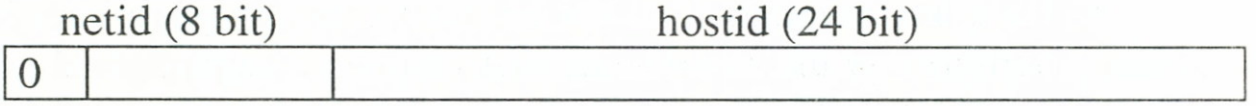
Tehát az IP cím két részből áll:

netid	**hostid**
A tartalmazó hálózat azonosítója	Az alhálózaton belüli végpont azonosítója

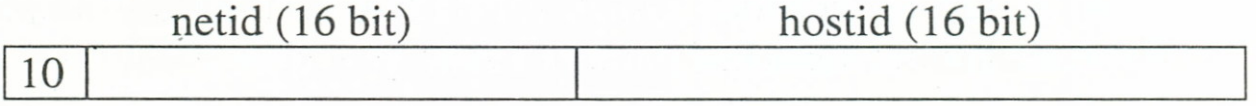
A címeket címosztályokba szokták sorolni:

A, B, C, D, E. Ezek közül az első hárommal találkozhatunk, a D speciális célú, az E fenntartott.

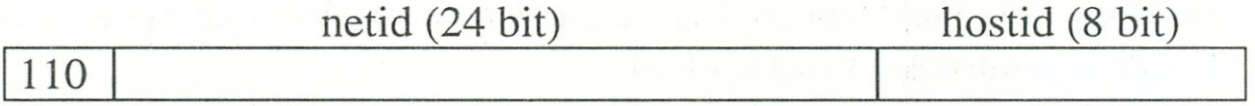
- **A osztályú címek**
  A netid 8 bites, és az első bitje 0. Így a hostid nyilván 24 bites. Mivel a hostid azonosítja a végpontot, ezért összesen 2^{24} db végpontot tartalmazó hálózatok számára alkalmas, de ilyen hálózat legfeljebb 128 (2^7) db lehet.



- **B osztályú címek**
  A netid 16 bites, az első két bit 10. Így összesen 2^{14} db hálózat lehet, hálózatonként 2^{16} géppel.



- **C osztályú címek**
  Az első 3 bit 110, összesen 2^{21} hálózat lehetséges, és mindegyikben 256 (2^8) db gép.



11.10.1 Pontozott decimális jelölés

A 32 bites cím nagyon nehezen „emészthető", ezért a 32 bitet 4 db decimális számjegyként szokták ábrázolni. Ez azt jelenti, hogy a 4 bájtos számot bájtonként egy-egy decimális számra bontjuk, és pontokkal választjuk el egymástól. Például: 195.199.40.29

Címosztály	Az első bájt értéktartománya	Lehetséges IP címek
A	1-127	1.h.h.h-127.h.h.h
B	128-191	128.n.h.h.-191.n.h.h
C	192-223	192.n.n.h-223.n.n.h

Ahol n: hálózat azonosító, h: végpont, host azonosító.
Speciális címek:

- Ha a hostid csupa 0, akkor az nem egy egyedi gépet, hanem egy hálózat címét jelenti. Pl.: 195.199.40.0

- Ha a hostid csupa 1, akkor az most sem egy gép címét jelenti, hanem az egy ún. **broadcast cím**, aminek a segítségével a netid által megadott hálózat minden gépének egyszerre üzenhetünk.

- Vannak címek, amelyeket belső hálózatok (Intranetek) céljára tartanak fenn. Ilyen például a 192.168.0.0-192.168.255.255 címtartomány.

- A 127-tel kezdődő címek (pl.: 127.0.0.1), az ún. *loopback* nevű eszközhöz tartoznak, amely a saját gépünket jelenti.

11.10.2 Domén neves jelölés

Az IP címek nehezen jegyezhetők meg, és nem derül ki semmi plusz információ belőlük. Ezért alkalmazzák a domén neves címzést. A gépek azonosítása domének láncának a megadásával történik, az egyes domén neveket itt is pontokkal választjuk el.

Például a server.fazekas-debr.sulinet.hu cím a következőképpen értelmezhető: az adott gép Magyarországon van (hu), ezen belül része a Sulinet országos hálózatnak (sulinet), a gépet tartalmazó helyi hálózat neve fazekas-debr, a gép neve pedig server. A legfelső domén nevek szabványosak (hu, com, de, fi stb).

Nyilván szükség van a kétfajta címzés közötti megfeleltetésre. Az előző címhez például a 195.199.40.29 IP cím tartozik. A kapcsolat nem biztos, hogy kölcsönösen egyértelmű: az előbbi IP címhez más domén neves cím tartozik (www.fazekas-debr.sulinet.hu).

A kétféle címre vonatkozó információkat adatbázisokban tárolják, de mivel nagyon sok gép van, ezért ezeket az információkat elosztják az egyes aldomének között. Minden aldoménhez tartozhat egy ún. **domén név szerver (DNS)**, amelynek, a feladata az aldoménre vonatkozó domén nevek és IP címek tárolása, karbantartása. A DNS-ek hierachikus kapcsolatban állnak egymással.

Pl.: Tegyük fel, hogy a felhasználó elindít egy böngésző programot, és címként a *w3.fazekas-debr.sulinet.hu* címet adja meg. A folyamat – leegyszerűsítve – a következő:

1. A program megkeresi az aldomén DNS-ével, és kéri a megfelelő IP címet.
2. Ha a DNS a saját adatbázisában megtalálja, akkor azt megadja, és vége a folyamatnak.
3. Ha nem találja a saját adatbázisában, akkor a legmagasabb doménhez fordul (*.hu*), és annak DNS-étől kéri az információt. Ez a DNS felismeri, hogy az egyik alatta levő aldomén DNS-éhez tartozik a megadott cím: *.sulinet.hu*. A helyi DNS-nek ad egy listát azokról a DNS-ekről, amelyek ebben az aldoménben dolgoznak (pl.: debrecen.sulinet.hu).
4. A helyi aldomén sorban megkérdezi őket, és amelyik ismeri a címet az megadja a kért IP címet.
5. A helyi DNS tárolja az előbb megszerzett információt a gyorsítótárában.

11.11 Hálózati maszk

Ha egy üzenetet küldünk egy másik gépre, akkor hogyan tudjuk eldönteni, hogy a cél a saját hálózatunkban, vagy pedig egy másik hálózatban van? Ebben segít a **hálózati maszk (subnet mask)**, amelyet minden hálózat esetén meg kell adni. Az IP címből kell tehát a hálózat címét „kinyerni". A hálózat címét pedig úgy kaphatjuk meg, hogy a címet szét kell választani netid és hostid részekre, a netid részt meghagyjuk, a hostid-t pedig 0-val helyettesítjük.

Például 192.168.2.10 egy C osztályú cím, ezért a netid az első három szám, a hostid pedig az utolsó. Így a hálózat címe 192.168.2.0 lesz.

Mit is csináltunk? Az IP cím és a hálózati maszk között bitenkénti ÉS műveletet alkalmaztunk. A hálózati maszk pedig a következő volt: 255.255.255.0. Általánosan azt mondhatjuk: a hálózati maszk függ a címosztálytól, és a **netid helyén** csupa **1-es** áll, a **hostid helyén** pedig csupa **0**.

A bitenkénti ÉS műveletet pedig úgy végezzük el, hogy mind a két esetben a decimális számokat átírjuk binárissá, majd egymás alá írva, például a végükről kiindulva az egyes bináris számjegyekre páronként az ÉS műveletet alkalmazzuk, úgy, hogy az igaznak az 1-es, a hamisnak pedig a 0 felel meg.

Az előző példát véve:

192.168.2.10 ⟶ 1100 0000.1010 1000.0000 0010.0000 1010
255.255.255.0 ⟶ 1111 1111.1111 1111.1111 1111.0000 0000

Az ÉS művelet eredménye: 1100 0000.1010 1000.0000 0010.0000 0000
Azaz 192.168.2.0.

A következő alapértelmezett hálózati maszkokat használhatjuk a címosztálytól függően:

Címosztály	Hálózati maszk
A	255.0.0.0
B	255.255.0.0
C	255.255.255.0

11.12 Alhálózatok

A teljes hálózatot néha részekre szokták osztani, alhálózatokra. Például a Sulinet programban nem teljes C osztályú címtartományt kaptak az iskolák. A lehetséges 256 címből álló tartományt 16 részre, azaz alhálózatra osztották, és azt kapták meg az iskolák.

Például a debreceni Fazekas Mihály Gimnázium a 195.199.40.0 – 195.199.40.255 tartományból a 195.199.40.0 - 195.199.40.15 tartományt kapta meg (tehát 256 cím helyet 16-ot).

Hogyan lehet egy alhálózat címét meghatározni? Mi lesz a hálózati maszk?

Az alhálózatokra bontás alapgondolata az, hogy a rendelkezésre álló hostid néhány bitjét a netid-hez kapcsoljuk. Az így kapott alhálózatokhoz

viszont nem használhatjuk az alapértelmezett hálózati maszkot. Új maszkra van szükség, amit úgy képezünk, hogy az alapértelmezett maszkban azokat a biteket, amelyeket a netid-hez kapcsoltuk, 0 helyett 1-re változtatjuk.

Például C osztályú címtartomány esetén a következő lehetőségek vannak:

A netid-hez kapcsolt bitek száma (n)	Az alhálózatban használható címek száma ($256/2^n$)	Hálózati maszk
0	256	0000 0000
1	128	1000 0000 (128)
2	64	1100 0000 (192)
3	32	1110 0000 (224)
4	**16**	**1111 0000 (240)**
5	8	1111 1000 (248)
6	4	1111 1100 (252)
7	2	1111 1110 (254)
8	1	1111 1111 (255)

A C osztályú alhálózati címek tehát a következőképpen néznek ki, ha 4 bitet használunk az alhálózat azonosítására:

nnnn nnnn.nnnn nnnn.nnnn nnnn.aaaahhhh

ahol:
n: netid bit,
h: hostid bit,
a: alhálózati azonosító bit.
A hálózati maszk tehát a következő lesz: 255.255.255.240

Megjegyzés:

A debreceni Fazekas sikeresen igényelt még 16 IP számot, így a címtartomány most a következő: 195.199.40.0 – 195.199.40.31. A hálózati maszk pedig ennek megfelelően 255.255.255.224.

Példa: A berettyóújfalui Brózik Dezső Szakközépiskola alhálózatának hostid-je: 1000 0000 (128), itt is az első négy bit az alhálózatot azonosítja, az utolsó 4 bit pedig a hostid-t adja meg.

1. Hálózati maszk: 255.255.255.240
2. A hálózat címe: 195.199.40.128
3. A broadcast cím: 195.199.40.143
4. A router címe: 195.199.40.142

Az előző példában elvileg 16 címet használhatunk, de látszik, hogy nem mindegyike szolgál gép azonosítására.

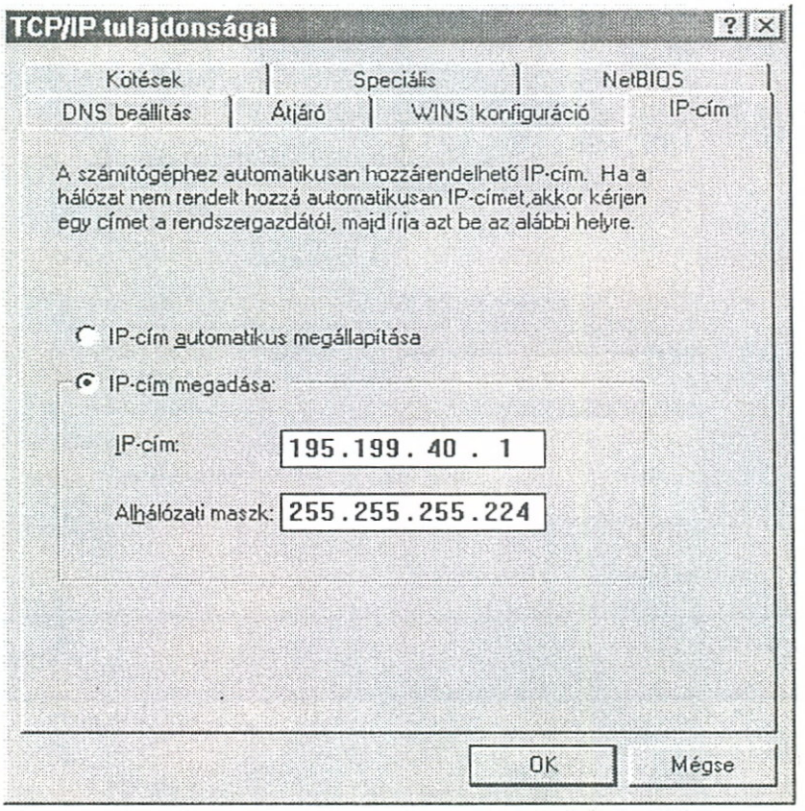
A 195.199.40.0 hálózatot 16 részre „vágták". Az egyes alhálózatok a következők:

Az alhálózatot azonosító bitek (aaaa)	A teljes hostid mező binárisan	A teljes hostid mező decimálisan	Az alhálózat címe
0000	0000 0000	0	195.199.40.0
0001	0001 0000	16	195.199.40.16
0010	0010 0000	32	195.199.40.32
0011	0011 0000	48	195.199.40.48
0100	0100 0000	64	195.199.40.64
0101	0101 0000	80	195.199.40.80
0110	0110 0000	96	195.199.40.96
0111	0111 0000	112	195.199.40.112
1000	**1000 0000**	**128**	**195.199.40.128**
1001	1001 0000	144	195.199.40.144
1010	1010 0000	160	195.199.40.160
1011	1011 0000	176	195.199.40.176
1100	1100 0000	192	195.199.40.192
1101	1101 0000	208	195.199.40.208
1110	1110 0000	224	195.199.40.224
1111	1111 0000	240	195.199.40.240

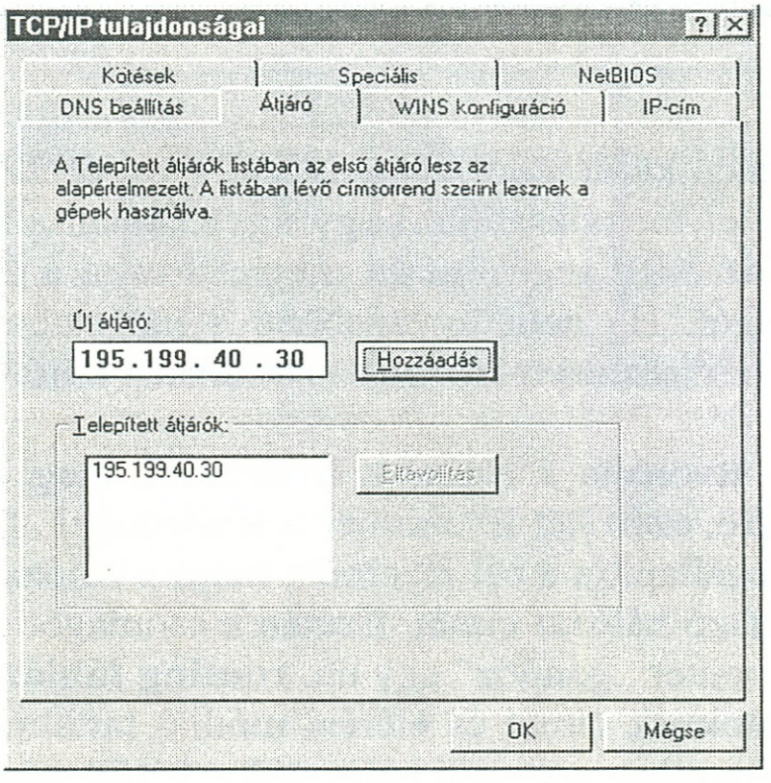
A TCP/IP hálózati beállításokat a következőképpen végezhetjük el Windows 95-ben, illetve Windows 98-ban:

Start/Beállítások/Vezérlőpult/Hálózat/TCPIP Tulajdonságok

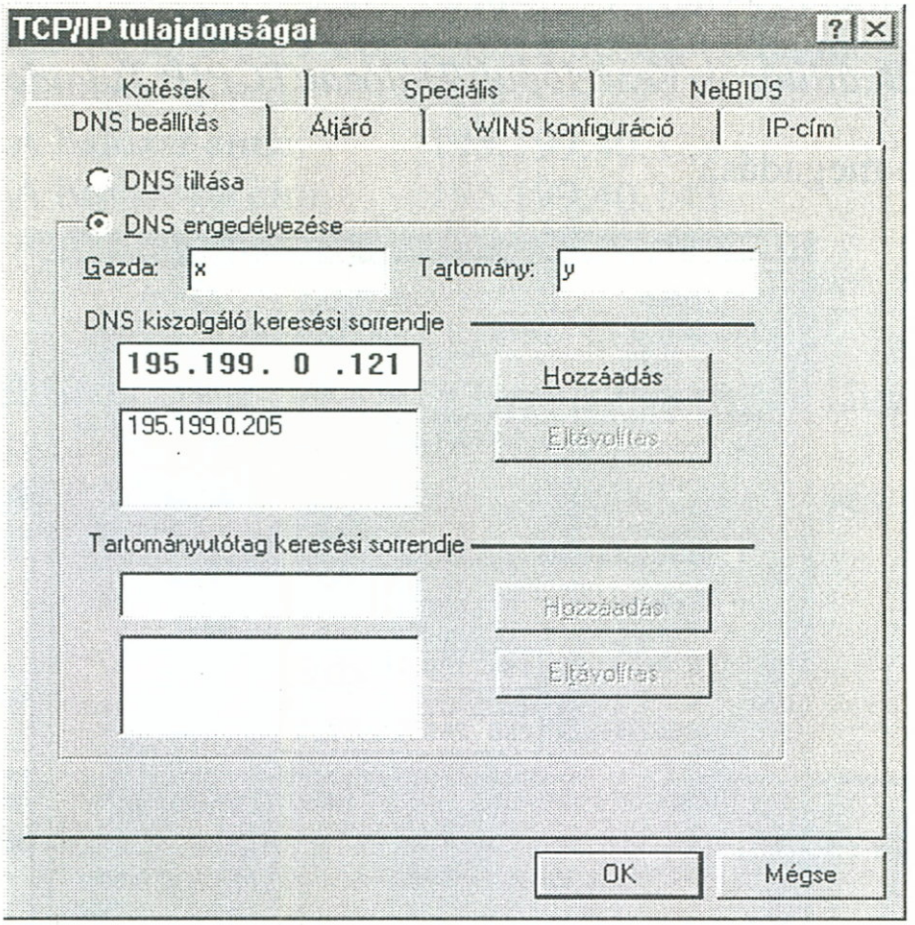
- Az IP cím megadása:



A router címének megadása:



A DNS szerverek megadása:



11.13 Útvonalválasztás

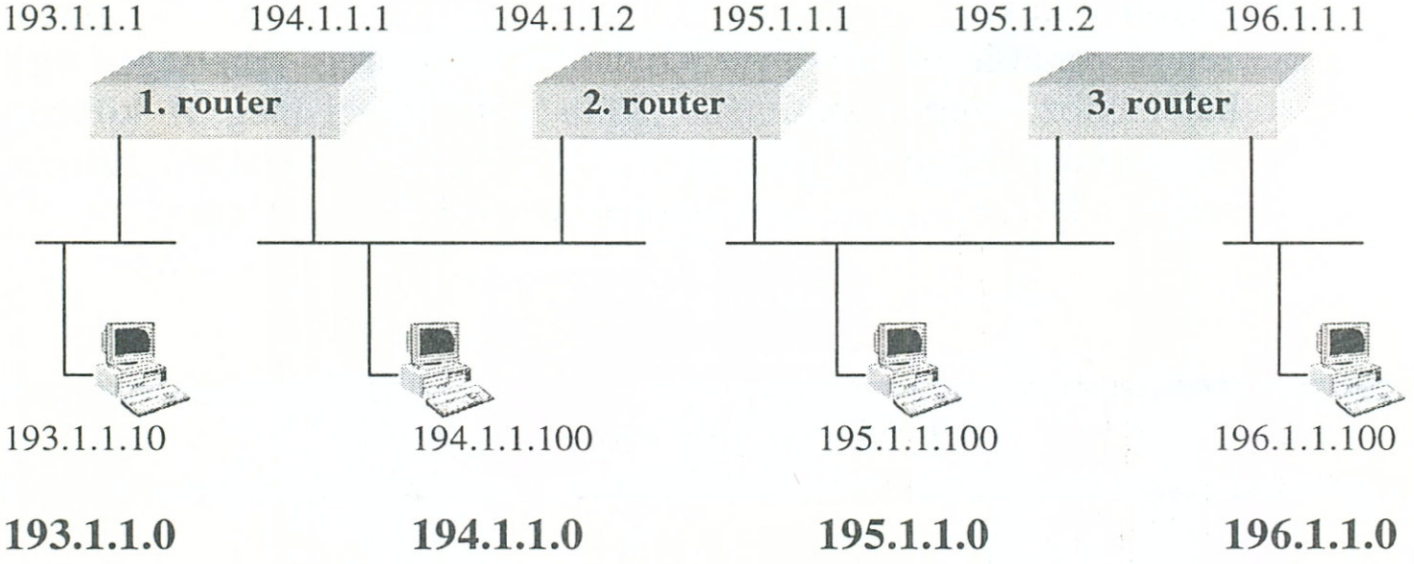
Az adó csomag küldésekor megvizsgálja a cél IP címét, a hálózati maszk segítségével megvizsgálja, hogy ugyanabban a hálózatban vannak. Ha igen, akkor az ARP segítségével meghatározza a cél fizikai címét, és elküldi a csomagot. Ha nem ugyanabban a hálózatban vannak, akkor a csomagot az **alapértelmezett (default) router**nek küldi.

A csomagok útvonalát a routerek határozzák meg. Mivel a router két hálózatnak is része, ezért két IP címmel is rendelkezik. Egy csomag érkezésekor először megállapítja a cél IP címét, majd a hálózati maszk segítségével a célt tartalmazó hálózat címét. Ezután a csomagot a megfelelő hálózat felé irányítja. A router „gondoz" egy ún. **routing táblázat**ot, amely mindazoknak a hálózatoknak címét és elérési módját tartalmazza, amelyekről a router tud. A táblázatban szereplő hálózatokat kétféleképpen érhetik el:

1. Közvetlenül: A router erre hálózatra is csatlakozik.
2. Közvetve: Egy vagy több lépésben, más routereken keresztül. Ilyenkor a táblázat a soron következő router (**next hop router**) címét tartalmazza. A next hop router mindenképpen valamelyik közvetlenül kapcsolódó hálózatban van.

A routing táblázatban tehát kétféle bejegyzés lehet:

1. Közvetlenül kapcsolódik a hálózathoz.
2. A next hop router címe



1. router		2. router		3. router	
Hálózat	**Next hop**	**Hálózat**	**Next hop**	**Hálózat**	**Next hop**
193.1.1.0	KK	193.1.1.0	193.1.1.1	193.1.1.0	195.1.1.1
194.1.1.0	KK	194.1.1.0	KK	194.1.1.0	195.1.1.1
195.1.1.0	194.1.1.1	195.1.1.0	KK	195.1.1.0	KK
196.1.10	194.1.1.1	196.1.10	195.1.1.1	196.1.10	KK

KK: közvetlenül kapcsolódik.

A host címe	A host számára kijelölt default router címe
193.1.1.100	193.1.1.1
194.1.1.100	194.1.1.1 (lehetne 194.1.1.2)
195.1.1.100	195.1.1.1 (lehetne 195.1.1.2)
196.1.1.100	196.1.1.1

A routing tábla kitöltése kétféleképpen történhet:

- Kézzel: A rendszergazda tölti ki a tálázatot.

- Automatikusan: A routerek egymás között kicserélik a megszerzett útvonal-információkat. Általában több út is lehetséges egy hálózathoz, ezért a minimális „költségűt" jegyzi meg. A **költség** kiszámítása nagyon egyszerű: a közvetlenül kapcsolódó hálózat esetén a költség 1, egy routeren keresztül elérhetőé 2, stb.

11.14 Proxy szerverek

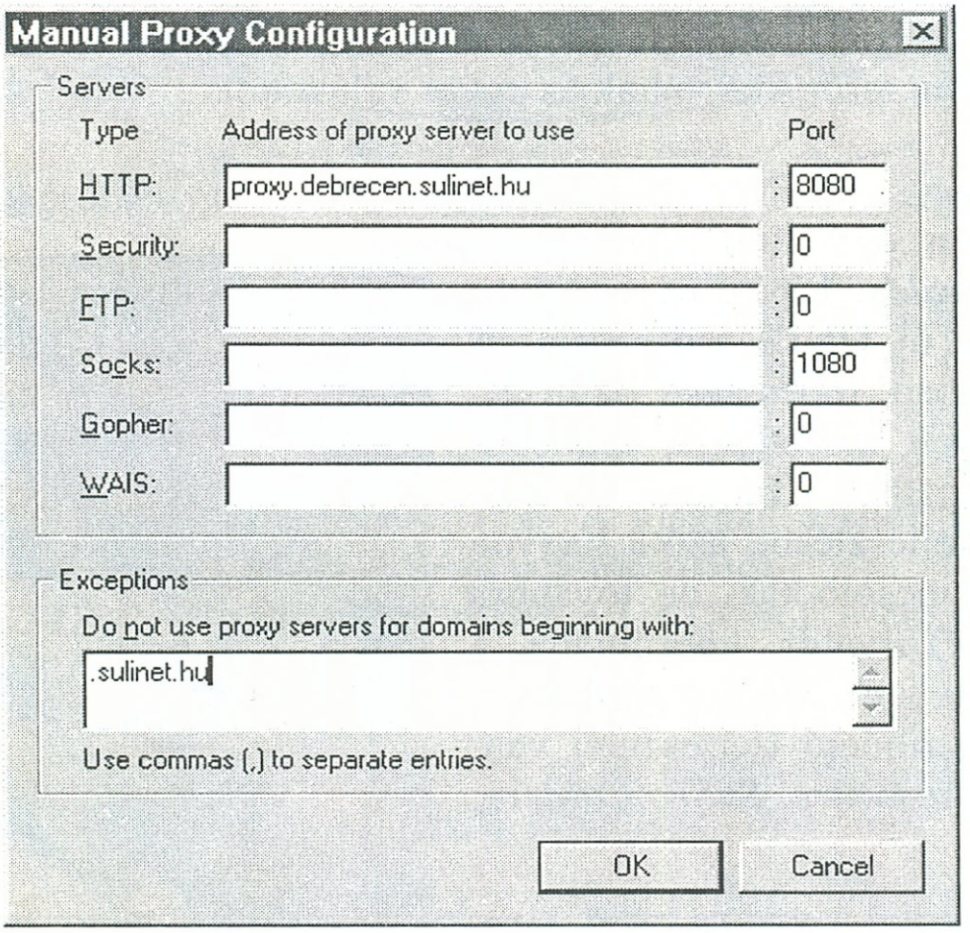
A proxy szervereknek legalább két hálózati csatlakozója van: az egyikkel az Internetre csatlakozik, a másikkal pedig egy hálózatra. Tehát a csatlakozó hálózat gépei csak a proxy szerveren keresztül érhetik el az Internetet. Alapvetően két funkciója van:

1. **Tűzfal**
   A hálózati forgalmat szűri. Ez azt jelenti, hogy a kimenő és bemenő forgalmat lehet vele ellenőrizni. Ez egy védelmi lehetőség, hiszen a kívülről jövő illetéktelen beavatkozásokat meg lehet vele akadályozni. Természetesen arra is lehet használni, hogy a rendszergazda meghatározza, milyen címeket ne lehessen az alhálózatból elérni.

2. **Cache**
   A proxy szervernek nagykapacitású merevlemeze(i) és nagy memóriája van. Valamilyen szisztéma alapján menti a már kért HTML lapokat, illetve fájlokat. Így ha újra kéri egy másik gép az alhálózatból, akkor nem kell újra letölteni az Internetről, hanem a saját merevlemezéről küldi el a kliensnek.

A böngészők és ftp programok esetén megadhatjuk a proxy szerver címét. Netscape Navigator-ben a következőképpen tehetjük meg:

Edit/Preferences/Advanced/Proxies/Manual proxy configuration/View



A Sulinet hálózatban kötelező böngésző program esetén megadni egy proxy szervert, elsősorban a cache funkció miatt.

12. LINUX

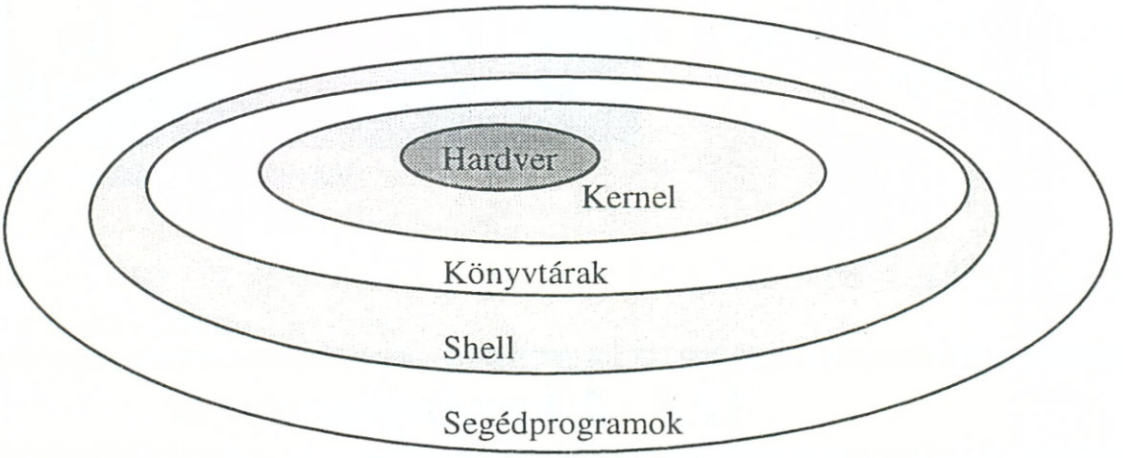
1969-ben az AT&T Bell Laboratóriumában *Ken Thomson* és *Dennis Ritchie* assembly nyelven kifejlesztette - egy PDP-7 típusú gépre - a **UNIX** operációs rendszer első változatát. 1973-ban nagy részét C nyelven újraírták. 1974-től az AT&T az egyetemeken ingyen terjesztette, aminek következtében 1979-re már 500 különböző UNIX rendszer működött az USA-ban. A szabad terjesztésnek nemcsak a gyors elterjedés lett a következménye, hanem sok, egymástól többé-kevésbé eltérő változat jött létre, hiszen a forráskód is szabad volt. A két fő változat, a System V és BSD. Később kereskedelmi termékké váltak az egyes változatok.

1991-ben egy finn egyetemista *Linus Torvalds*, Linux néven létrehozott egy új **UNIX változat**ot. A Linux szabad terjesztésű szoftver, bárki ingyen másolhatja, módosíthatja, terjesztheti - ha nem kér érte pénzt. A fejlesztésbe azóta nagyon sokan bekapcsolódtak, de továbbra is Linus Torvalds koordinálja a munkát (ingyen). Ma már több nagy Linux disztribúció (terjesztés) van: *RedHat, Slackware, SuSE, Caldera, Debian.*

Nem a Linux az egyetlen ingyenes UNIX változat. *FreeBSD* néven is fejlesztenek változatot, amely (legalábbis egyelőre) nem olyan elterjedt, mint a Linux.

12.1 Felépítés

A UNIX egy **többfelhasználós, többfeladatos** operációs rendszer. Vázlatos felépítése a következő:



A **kernel** az operációs rendszer magja: ez végzi az operációs rendszer erőforrásainak a szétosztását, a folyamatok ütemezését. A felhasználói programok a **megosztott könyvtárak**ban levő szubrutinokat (alprogramokat, könyvtári függvényeket) futásuk közben felhasználhatják, tehát dinamikusan gazdálkodnak a memóriával. A maggal, a **shell** (burok, parancsértelmező) segítségével teremthetünk kapcsolatot.

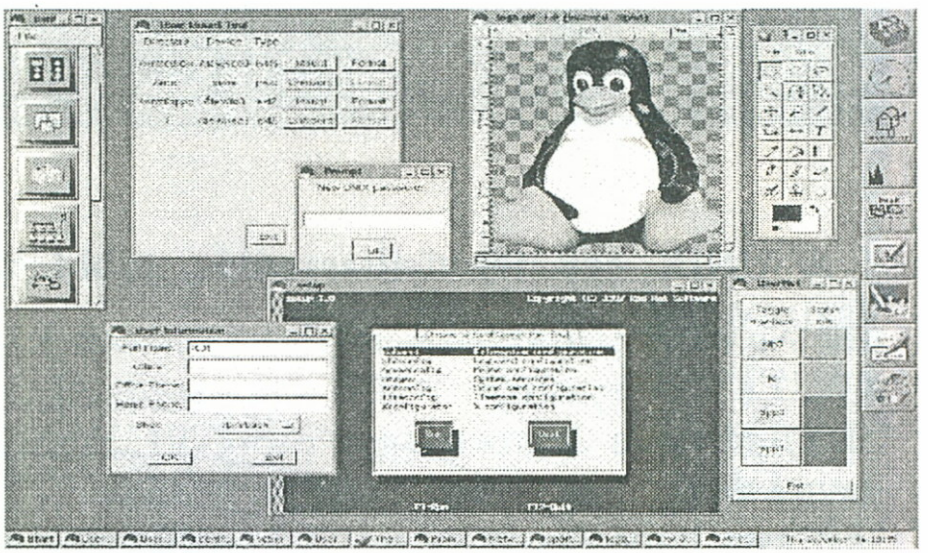
Mivel az MSDOS nagyon sokat örökölt a UNIX-ból, ezért párhuzamot vonhatunk a két rendszer felépítése között. Így a kernelnek az IO.SYS és az MSDOS.SYS felel meg, a shellnek a COMMAND.COM, és a segédprogramoknak a külső parancsok. Persze ez az összehasonlítás is „sántít", hiszen az MSDOS funkciói, az egész felépítése, működése lényegesen egyszerűbb.

A többfelhasználós jellegből következik, hogy az egyes felhasználóknak **felhasználói név**vel kell rendelkezni, az azonosításuk miatt, és **jelszó**val a védelmük érdekében. A rendszerbe való belépés a **login** paranccsal történik:

login *felhasználói_név*

A felhasználói név beírása után, a jelszót is meg kell adnunk. Kijelentkezni a **logout,** vagy a **ctrl-d** paranccsal tudunk.

A rendszert karakteres módban és grafikus felületen (X Window) keresztül is használhatjuk – a legtöbb esetben.



12.2 Állományrendszer

Azt szokták mondani, hogy a UNIX-ban *minden fájl.* Az állomány adatok tárolására szolgál, nincs szerkezete, egyszerűen csak bájtok sorozata, és nem tartalmaz fájl-vége jelet. Egy fájl tartalmazhat ténylegesen adatokat, lehet könyvtár, és lehet valamilyen periféria is. A képernyő például a */dev/tty1* nevű fájl (*tty1* a fájl neve, ami a *dev* nevű könyvtárban van).

12.2.1 Fájltípusok

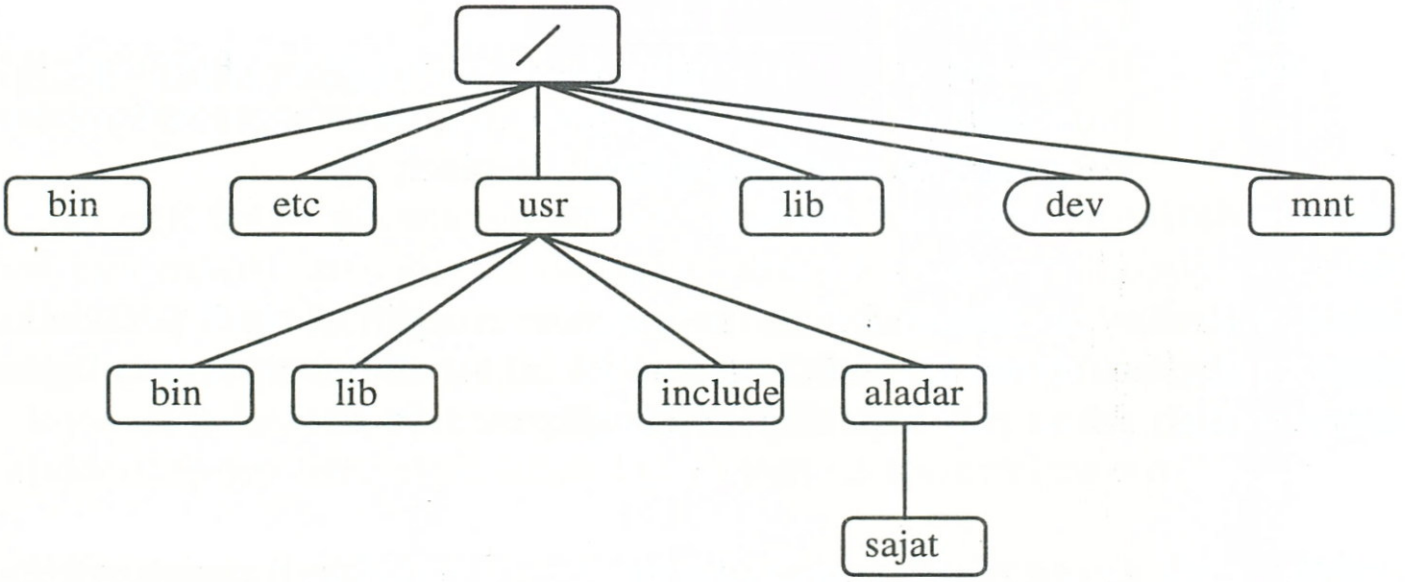
A következő típusú állományokat használhatjuk:

Jele	Típusa
-	Közönséges fájl
d	Könyvtár
p	Cső (pipe)
l	Szimbólikus lánc
c	Karakteres készülék
b	Blokkos készülék

-: „Rendes", már ismert állományok: programok, dokumentumok, stb.
d: Könyvtár, a már az MSDOS-ban is megismert fogalom.
c: Valamilyen bemeneti/kimeneti készülék, amely az adatokat karakterfolyamként kezeli. Ilyen például a képernyő, soros kimeneti eszköz, stb.
b: Valamilyen bemeneti/kimeneti készülék, amely az adatokat blokkokban kezeli. Ilyen például a floppy, winchester, stb.

12.2.2 Könyvtárstruktúra

A könyvtárak egy fa gráfnak megfelelő könyvtárstruktúrában helyezkednek el. A gyökérkönyvtár jele: /.



Az aktuális könyvtár jele: „.", a szülő könyvtáré „..". A felhasználó saját könyvtárát a „~" jelzi.

Az állományok helyét az MSDOS-ban megismert **abszolút** ill. **relatív út** segítségével adhatjuk meg.

Az állományok, könyvtárak neve gyakorlatilag tetszőleges hosszúságú lehet, és szinte minden karaktert használhatunk. (Kivételek: „/", „*", „+", „?", „<", „>", „ , ") Az MSDOS-ban megismert kiterjesztéseket a UNIX nem ismeri. Pontosabban, használhatunk kiterjesztést, de itt a pont is része a névnek, és a UNIX nem kezeli külön. Akár több kiterjesztést is adhatunk, így pl. a következő is egy szabályos név: *valami.gz.tar. Megkülönbözteti a kis- és nagybetűket*, tehát pl.: az ls, Ls, LS, lS különböző nevek.

Megjegyzés:

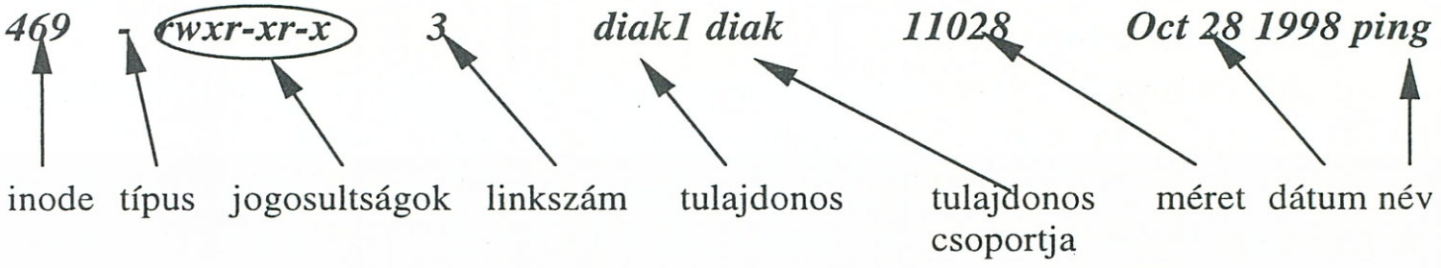
A UNIX a parancsok esetén is különbséget tesz a kis- és nagybetűk között.

Néhány fontosabb könyvtár funkciója:

bin: A rendszer működtetéséhez legszükségesebb segédprogramokat tartalmazza.
dev: Az egyes eszközökhöz tartozó állományokat tartalmazza.
etc: A rendszer konfigurálásához szükséges állományokat tartalmazza.
home: A felhasználók saját könyvtárai.
mnt: Állományrendszer becsatolására szolgál.
lib: Programkönyvtárakat tartalmaz.
usr: Itt van az ún. felhasználói állományrendszer. Publikus felhasználói programok, a beépített súgó-rendszer, az ún. kézikönyvek (manual) állományai, leírások stb.

Az állományok listázására az **ls** parancs szolgál. A **ls-l** paranccsal hosszú (részletes) listát kérhetünk. Ha az ún. rejtett állományokat is listázni szeretnénk, azt az *„a"* kapcsolóval tehetjük meg: **ls–a**.

Például: *ls –la /usr/bin/ping* (Az /usr/bin könyvtárbeli ping nevű fájl listázása.)



12.2.3 Háttértárak

A UNIX-ban *nincsenek meghajtók.* Az egyes háttértárakat a nekik megfelelő állományokkal azonosítjuk. Néhány példa:

/dev/hda1: Az első EIDE winchester („a") 1. partíciója.
/dev/hda2: Az első EIDE winchester („a") 2. partíciója.

A „b" jelű eszköz jelentheti a második EIDE winchestert, de CD-ROM-ot is.

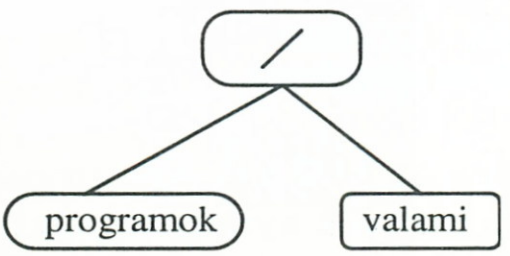
/dev/sda1: Az első SCSI winchester („a") 1. partíciója.
/dev/fd0: Az első floppy.

A partícionálást az **fdisk** paranccsal lehet elvégezni.

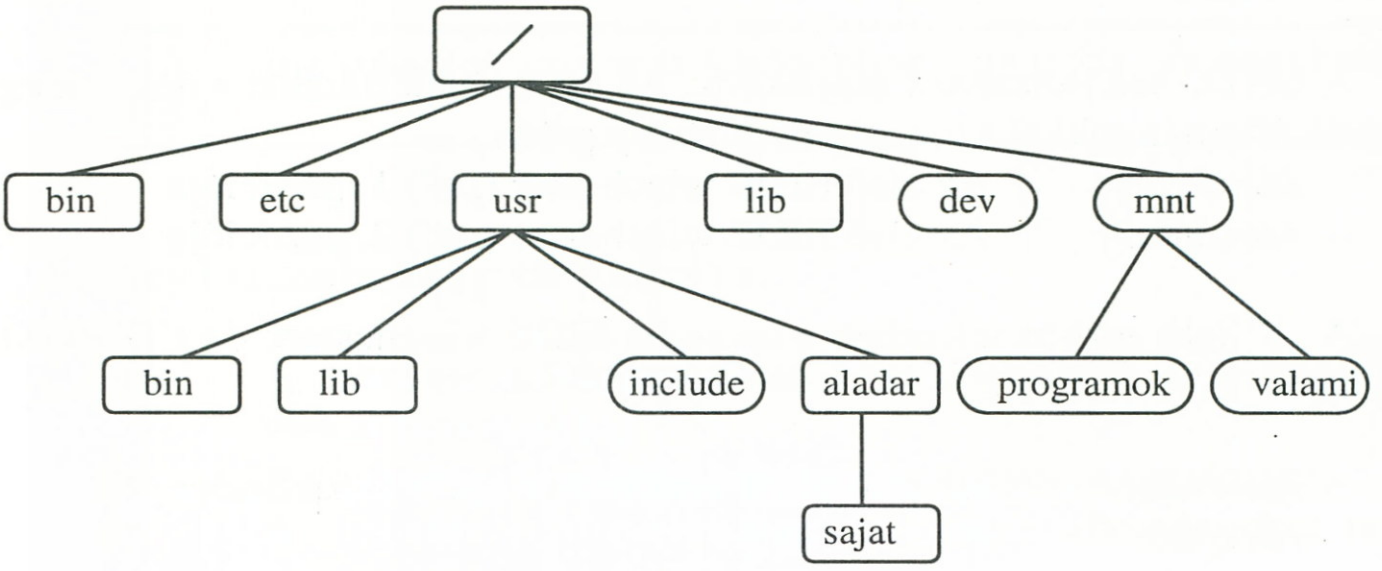
12.2.4 Mountolás

Ha egy újabb állományrendszert akarunk használni (a winchesterünknek egy másik partícióját is használni akarjuk, vagy egy floppy, vagy egy CD-ROM állományait szeretnénk elérni, stb.), akkor azt be kell **csatolni**, vagy más szóhasználattal fel kell **mountolni**. Ez azt jelenti, hogy az új könyvtárszerkezetet beillesztjük valamelyik könyvtár „alá". Ez a könyvtár tulajdonképpen bármelyik lehet, bár a **/mnt** az alapértelmezett.

Példaképpen csatoljuk be a floppy-t az /mnt alá. Legyen a floppy-n levő állományrendszer a következő:



A becsatolás után a fájlrendszer:



Ezek után úgy hivatkozhatunk a floppy-ra, mint az állományrendszerünk /mnt-ből nyíló alkönyvtárrendszerére.

Ha már nincs szükség a becsatolt állományrendszerre, akkor **lecsatoljuk**.

Megjegyzés:

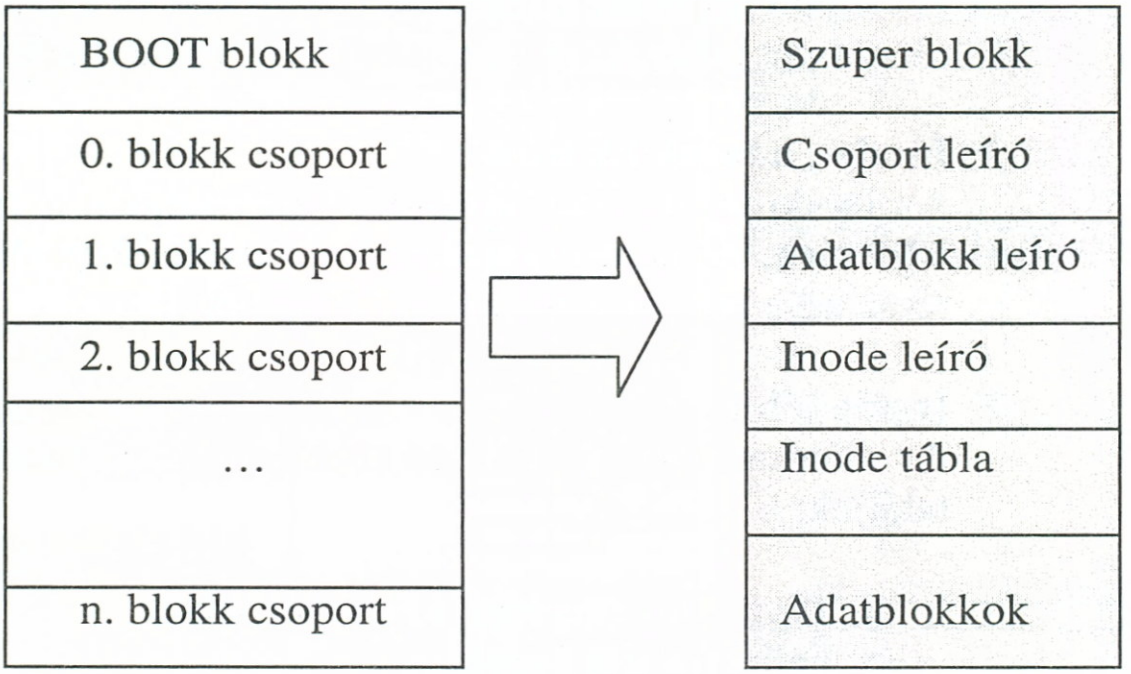
1. *A mountolást általában csak a rendszergazda, a* ***root*** *nevű felhasználó végezheti el.*
2. *MSDOS-ban is létezik hasonló lehetőség: a JOIN parancs.*

12.2.5 Az állományok tárolása

Az állományok tárolása az MSDOS-ban megismerttől teljesen különbözik. Itt nincs FAT, az állományok nyilvántartása teljesen más rendszerű.

Manapság az ún. EXT2 fájlrendszert használják a Linux-ban. A nyilvántartási egység neve **blokk,** amely éppen két szektor méretű (1 KB).

Az EXT2 partíció logikai felépítése: **A blokk csoportok felépítése**



- **BOOT blokk**
  Az operációs rendszer indításához szükséges a BOOT blokk. A partíció, méretétől függően „n" db csoportra van felosztva, amelyeknek a felépítése azonos.

- **Blokk csoportok**
  - A **Szuper blokk** a partíció adatait tartalmazza, és a biztonság kedvéért minden csoportban megtalálható.

  - **Könyvtárbejegyzések**
    A könyvtárakra vonatkozó könyvtárbejegyzések is adatterületen (Adatblokk) vannak. Az állományokra, alkönyvtárakra vonatkozóan a következő adatokat találhatjuk a bejegyzésekben:

    - **Inode szám**
      Az inode szám egy *egyedi azonosító* (egész szám), amely minden állomány, könyvtár esetén más. Minden csoport blokkban van inode tábla, de a számozás folyamatos.

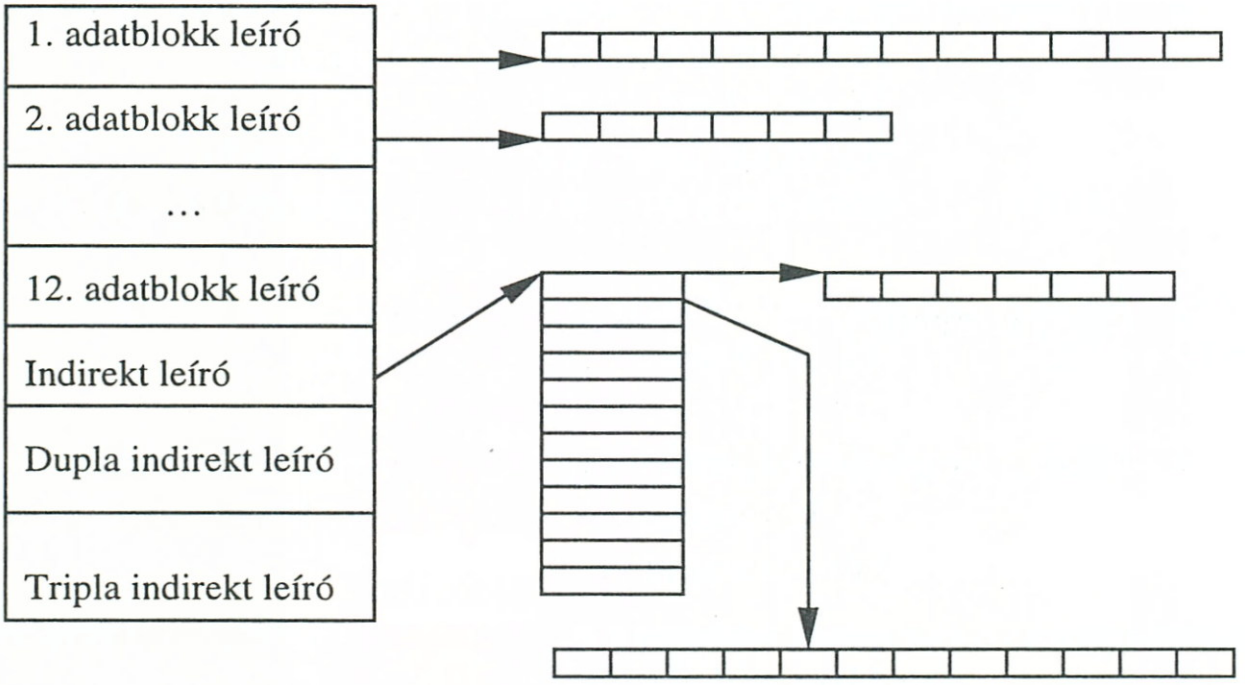
    - **Név**

➢ **Csoport leíró**
A csoport leíróban a szabad adatblokkok és a szabad inode-ok számát találjuk, az adott csoportra vonatkozóan.

➢ **Inode-, adatblokk leíró**
Az inode leíró és az adatblokk leíró szintén a foglaltságot adja meg, de nem összességében, hanem egyenként, konkrétan adja meg *egy-egy bit* segítségével.

➢ **Inode tábla**
Az állományokra vonatkozó többi adatot az inode táblában találjuk:

- **Típus (közönséges, könyvtár stb.).**
- **Felhasználói jogok.**
- **Tulajdonos.**
- **Méret.**
- **A létrehozás dátuma.**
- **Az utolsó módosítás dátuma.**
- **A fájl elhelyezkedésének leírói.** Az állományok összefüggő blokkokból álló adatterületen tárolódnak, amelyeket sorszámmal lehet azonosítani. Ilyen összesen 12 db lehet.
  Ha az állomány nem fér el ezen a helyen, akkor **ún. indirekt hivatkozások**kal lehet a többi darabot tárolni. (Ez egyébként ritkán fordul elő.) Az indirekt leíró 12 újabb adatblokk leíróra mutat.



12.3 Állomány és könyvtárkezelő parancsok

A UNIX parancsok a következő szerkezetűek:

parancsnév -kapcsolók paraméterek

A parancsokról a következőképpen nyerhetünk információt:

man parancsnév

Megjegyzés:

1. *A „[" és „]" közöttiek opcionálisak, azaz nem kötelező a használatuk.*
2. *Az állományokat illetve alkönyvtárakat, ha szükséges, akkor az elérési úttal együtt kell megadni (relatív, abszolút).*

- **ls** *[kapcsolók] [fájl]* : Állományok, könyvtárak listázása.
  - -a: A rejtett állományokat is listázza.
  - -l: Hosszú lista.
  - -i: Az inode számokat is listázza.
  - -R: Kilistázza rekurzívan az összes alkönyvtárt.

- **Cd** *[könyvtár]:* Könyvtárváltás.

- **Mkdir** *[kapcsolók][könyvtár]*: Könyvtár létrehozása.
  - -p: A létrehozandó könyvtárig létrehozza a hiányzó könyvtárakat is.

- **rmdir** *[kapcsolók][könyvtár]:* Üres könyvtár törlése.
  - -p: A szülőkönyvtárakat is törli.

- **mv** *[kapcsolók][forrás][cél]:* Állományok mozgatása.
  mv *[kapcsolók][források][célkönyvtár]:*
  Ha a forrás és célkönyvtár azonos, akkor átnevezés történik.

- **cp** *[kapcsolók][forrás][cél]:* Állományok másolása.

- **rm** *[kapcsolók][fájlok]:* Állományok törlése.
  - -r: Az szülőkönyvtárakkal, és a bennük levő állományokkal együtt mindent töröl.

Megjegyzés:

A UNIX-ban a törölt fájlok semmilyen módon nem állíthatók vissza!

- **cat** *[kapcsolók][fájl]:* Fájl tartalmának megtekintése.

- **more** *[kapcsolók][fájl]:* Szöveges állományt listáz, képernyő oldalanként.

A fájlnevekben használhatunk **joker** (helyettesítő) **karaktereket** is:

- ? Egy karaktert helyettesít.
- * Tetszőleges számú karaktert helyettesít.
- [ ] A zárójelen belüli halmazból egy karaktert helyettesít.

Pl.: a[xyz] Olyan két karakterből álló nevek, melyek első karaktere „a", a második az „x", „y", „z" karakterek közül valamelyik.

[A-Z]* Az összes nagybetűvel kezdődő név.

12.4 Állományok láncolása

A UNIX-ban lehetőségünk van arra, hogy egy állományra több néven is hivatkozzunk. Erre két lényegesen különböző lehetőségünk van:

12.4.1 Merev láncolás (hard link)

Láthattuk, hogy az állományokra vonatkozó adatok többsége az inode táblában található, míg magában a könyvtárbejegyzésben tulajdonképpen csak a neve és az inode száma található.

Ha egy adott fájlra létrehozunk egy újabb bejegyzést (az inode szám nem változik, csak a név), akkor most már egy másik néven is hivatkozhatunk ugyanarra az állományra. Ezt hívják merev láncolásnak.

A láncolás létrehozására az **ln** parancs szolgál:

ln *eredeti_fájl* *újnév*

Példa: Legyen az állomány neve *proba1*. Hozzunk létre egy „hard linket" erre az állományra, *proba2* néven!

```
ln proba1 proba2
```

Listázzuk ki a könyvtárt:

```
ls –lai
```

```
474 drwxr-xr-x   2    diak1    diak   8192  Aug  3 13:52   .
 99 drwxr-xr-x  14    diak1    diak   8192  Aug  3 13:49   ..
509 -rw-r- - r--   1    diak1    diak   425   Feb  1  1995  .bash_history
507 -rw-r – r- -   1    diak1    diak   34    Nov 25 1993   .less
508 -rw-r- - r- -  1    diak1    diak   114   Nov 25  1993  .lessrc
510 –rw-r- - r- -  2    diak1    diak   6     Aug  3 13:55  proba1
510 –rw-r- - r- -  2    diak1    diak   6     Aug  3 13:55  proba2
```

Mi történik akkor, ha letöröljük az egyik linket? Csak annyi, hogy a linkszám eggyel csökken. Ha az utolsó töröljük, akkor viszont maga a fájl is törlődik!

A következőképpen ábrázolhatjuk a kapcsolatot:

láncolt név ⟶ inode ⟶ fájl

12.4.2 Szimbolikus láncolás (soft link)

Ezzel egy olyan könyvtárbejegyzést hozunk létre, amelyben az inode helyett a láncolt állomány neve lesz.

Példa: Legyen az állomány neve *proba1*. Hozzunk létre egy hard linket erre az állományra, *proba1* néven!

```
ln –s proba1 proba3
```

Listázzuk ki a könyvtárt:

```
ls –lai
```

```
474 drwx r–xr-x   2    diak1    diak   8192  Aug  3 13:52   .
 99 drwx r–x r-x  14    diak1    diak   8192  Aug  3 13:49   ..
509 –rw – r- - r- -  1    diak1    diak   425   Feb  1  1995  .bash_history
507 –rw – r – r- -   1    diak1    diak   34    Nov 25 1993   .less
508 –rw – r- -r - -  1    diak1    diak   114   Nov 25  1993  .lessrc
510 –rw –r- - r- -   2    diak1    diak   6     Aug  3 13:55  proba1
510 –rw –r- - r- -   2    diak1    diak   6     Aug  3 13:55  proba2
511 lrwxrwxrwx   1    diak1    diak   6     Aug 3 14:10   proba3-> proba1
```

Mi történik akkor, ha letöröljük a linket? Semmi, az eredeti állomány ezt nem „veszi észre" (nem változik a link szám sem).

Mi történik, akkor, ha az eredeti állományt töröljük, de a linket nem? Mindaddig semmi gond, amíg nem hivatkozunk az árván maradt szimbolikus linkre. Ha mégis hivatkozunk rá, akkor hibaüzenetet kapunk. Ha ugyanott, ugyanolyan néven újra létrehozunk egy állományt, akkor az erre a névre mutató szimbolikus link „feltámad", azaz újra használhatjuk. Éppen ez a tulajdonsága teszi hasznossá olyan esetekben, amikor az eredeti könyvtár, vagy állomány nem mindig érhető el, mert néha lecsatoljuk az őket tartalmazó fájlrendszert.

Tehát a következőképpen ábrázolhatjuk a kapcsolatot:

szimbolikus név ⟶ eredeti név ⟶ inode ⟶ fájl

12.5 Felhasználói jogosultságok

A UNIX többfelhasználós operációs rendszer, tehát a rendszert több felhasználó, egymástól függetlenül használhatja. Természetes igény a felhasználók részéről, hogy a saját könyvtárukat, állományaikat a többiek ne érhesse el. Ennek a problémának a megoldására találták ki a felhasználói jogosultságok rendszerét.

A felhasználókat a következőképpen csoportosíthatjuk:

- **root**: „Szuperfelhasználó", rendszergazda. Ő hozhatja létre a felhasználókat, és ő törölheti őket.

- **Felhasználói csoportok.**
  A root, egy felhasználó létrehozásakor egy csoportba is besorolja a felhasználót, az alapján, hogy hogyan kapcsolódik a többi felhasználóhoz (ugyanabba az osztályba járnak, stb).

- **„Egyszerű" felhasználók.**

Egy állomány tulajdonlása szempontjából a következőképpen csoportosíthatjuk a felhasználókat:

- **Tulajdonos** (user)
  Ha létrehozunk egy állományt, akkor annak mi leszünk a tulajdonosai.
- **A tulajdonos csoportja** (group)
- **A többiek** (other)
- **Az összes felhasználó** (all)

1. A **fájlokhoz** rendelt jogosultságok háromfélék lehetnek:

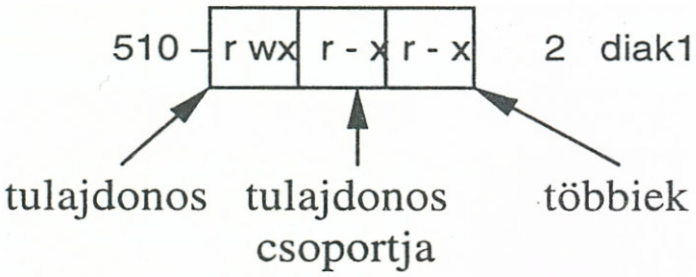
   - **Olvasás** (read)
     A fájl olvasásra való megnyitását jelenti.
   - **Írás** (write)
     A fájl módosítását, törlését jelenti.
   - **Végrehajtás** (execution)

2. **Könyvtárak** esetén a következő jogosultságok vannak értelmezve:

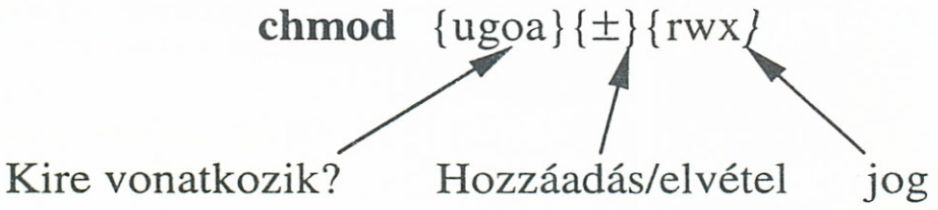
   - **Olvasás** (read)
     A tartalma listázható.
   - **Írás** (write)
     A benne levő fájlokat módosíthatjuk: létrehozás, törlés, átnevezés, stb.
   - **Végrehajtás** (execution)
     A könyvtárban levő állományokat kereshetjük.

Megjegyzés:

1. *Ha egy könyvtárra nincs végrehajtási jogunk (pl.: r - -), de olvasási igen, akkor a könyvtár neve listázható, de a benne levő állományok nem lehet, és nem is lehet végrehajtani.*
2. *Ha egy könyvtárra nincs olvasási jogunk (pl.: - -w), de végrehajtási igen, akkor az ls paranccsal nem tudjuk listázni az állományt, de el lehet indítani.*



Ha valamelyik jogosultság hiányzik, akkor azt a „-" jel mutatja.
A jogosultságok megváltoztatása a **chmod** paranccsal történhet:



Példa: Tegyük írhatóvá a *proba* nevű állományt a tulajdonos csoportja számára!

```
chmod g+w proba
```

Listázzuk ki az állományt!

```
ls –lai proba
```

```
510 – r wx r w x r - x  2 diak1 diak 6  Aug 3 13:55 proba
```

A tulajdonost a **chown** paranccsal lehet megváltoztatni:

chown kinek mit

Példa: Adjuk át a *diak2* felhasználónak a *proba* tulajdonát!

```
chown diak2 proba
```

Listázzuk ki az állományt!

```
ls –lai proba
```

```
510 – r wx r w x r - x  2 diak2 diak 6  Aug 3 13:55 proba
```

12.6 Folyamatok

Mivel a UNIX többfeladatos (multitasking) operációs rendszer, lehetőségünk van több folyamatot párhuzamosan futtatni. A parancs után az „&" jellel jelezhetjük, hogy a parancsot a háttérben akarjuk elindítani.

parancs &

Ezután a promptot azonnal visszakapjuk, és kiírja a folyamat azonosítóját *(process ID, PID).*

12.6.1 Folyamatok kilistázása

Az éppen futó folyamatokat a **ps** paranccsal kérdezhetjük le.

```
PID   TTY   STAT  TIME  COMMAND
95    S0    S     0:00  /usr/sbin/gpm -t ms -m /dev/mouse
122   1     S     0:00  -bash
123   2     S     0:00  -bash
124   3     S     0:00  /sbin/mingetty tty3
125   4     S     0:00  /sbin/mingetty tty4
126   5     S     0:00  /sbin/mingetty tty5
127   6     S     0:00  /sbin/mingetty tty6
142   1     S     0:00  mc
144   p0    S     0:00  bash -rcfile .bashrc
152   2     R     0:00  ps
```

A „-ef" kapcsolóval a rendszerben futó összes folyamatot listázza.

12.6.2 A folyamatok leállítása

A hibásan vagy túl sokáig futó programunktól akarunk megszabadulni, akkor azt a **kill** paranccsal tehetjük meg.

kill PID

12.6.3 Futtatás kilépés után

Sok esetben hasznos ha a programunk akkor is fut, amikor mi már kijelentkeztünk a rendszerből (pl.: sok állományt akarunk letölteni az *ftp* vagy a *wget* programmal az Internetről). Erre a **nohup** parancs ad lehetőséget.

nohup *parancs* **&**

12.6.4 Programok időzített futtatása

Az **at** parancs segítségével meghatározott időpontban indíthatunk el programokat.

at *idő*

Az *idő* többféle formátumú lehet, a legegyszerűbb, ha „óraperc" formátumban adjuk meg. Pl.: 1330 :délután fél kettő.

12.7 A standard input/output átirányítása

Az MSDOS-hoz hasonlóan a parancsok esetén standard inputról és outputról. Pl.: az ls parancs standard bemenete egy könyvtár-fájl, a kimenete a képernyő (ami szintén fájl).

A standard bemenetet és kimenetet átirányíthatjuk egy fájlra:

1. **< fájl** A bemenetet egy fájlból veszi a parancs.
2. **>fájl** A kimenet egy fájlba lesz átirányítva, felülírással.
3. **>> fájl** A kimenet egy fájlba lesz átirányítva, hozzáfűzéssel.

Speciális lehetőség a csővezeték:

parancs1 | parancs2 Az első parancs kimenete a második bemenetére kerül, mintha egy csövön keresztül haladna, ezért hívják ezt a megoldást **csővezeték**nek (**pipe**).

Példa: 1. *ls > lista* : a könyvtár tartalma nem a képernyőre kerül, hanem a *lista* nevű állományba.

2. *ls | sort*: Az *ls* parancs nem írja ki a képernyőre a könyvtár tartalmát, hanem átküldi a csővezetéken a *sort* parancsnak. A *sort* a standard bemenete a billentyűzet, de most a csővezetéktől kapja a bemenetet. Végül a *sort* sorba rendezve írja ki a listát a képernyőre.

12.8 Néhány hasznos parancs

- **grep** *[kapcsolók][minta][fájlok]:* A megadott fájlokban megkeresi azokat a sorokat, amelyekben az adott minta előfordul.
  -c: csak a mintát tartalmazó sor számát írja ki
  -l: csak az állományneveket írja ki
  Pl: *grep süsü proba* : a proba nevű állományban azokat a sorokat írja ki, amelyekben szerepel a „süsü" karaktersorozat.

- **sort** *[kapcsolók][fájlok]:* A bemenetként kapott állományok sorait rendezi, majd a képernyőn megjeleníti.
  Ha nem adunk meg állományt, akkor a standard bemenet a billenytűzet.
  -r: fordított sorrendben rendez.

- **wc** *[kapcsolók][fájlok]:* A megadott állományokban megszámolja a szavakat, sorokat, karaktereket.
  -l: csak a sorokat számolja
  -w: csak a szavakat számolja
  -c: csak a karaktereket számolja

- **echo** *[kapcsolók][szöveg]:* A megadott szöveget kiírja a képernyőre.

- **pwd**: Megadja az aktuális könyvtárt.

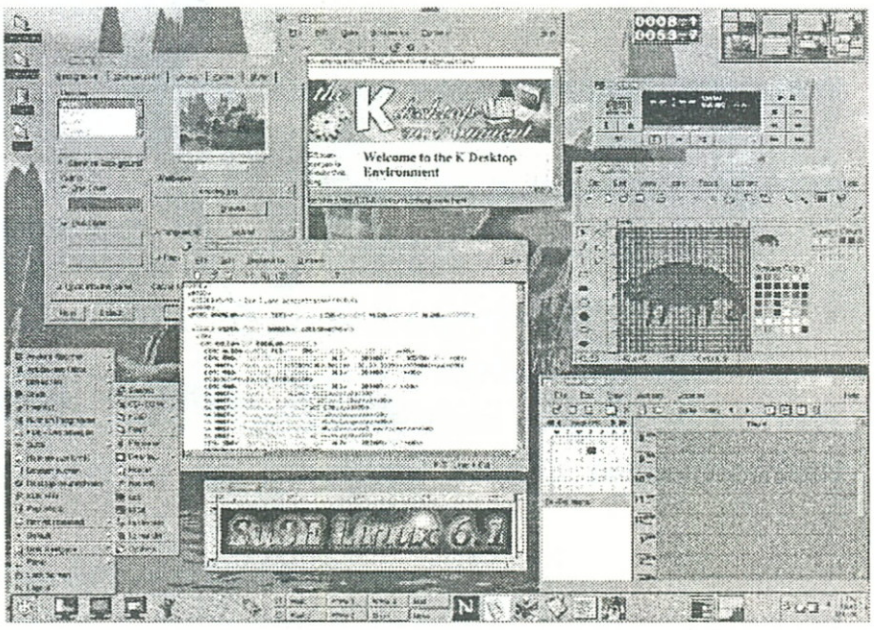
- **who**: Az éppen bejelentkezett felhasználókat kilistázza.
- **date** *[kapcsolók]:* Kiírja az aktuális dátumoz és időt.
- **passwd**: A jelszavunkat változtathatjuk meg vele.

12.8.1 Internetes parancsok

- **mail:** Levelező program.
- **pine:** Levelező program.
- **ftp:** Ftp program.
- **telnet:** Bejelentkezés távoli gépre.
- **ping** *internetes cím:* Csomagokat küld a megadott címre, majd várja a címzettől a visszajelzést, közben pedig méri az időt, és számolja az elveszett csomagokat.

12.9 X Window System

A UNIX felhasználókban is felmerült az igény a grafikus felhasználói felület (**GUI:** ***Graphical User Interface***) iránt, amely lehetőleg minél több fajta számítógépen fut. Az ez irányú fejlesztések eredménye az X Window System, amely ma már nagyon elterjedt.



Nincs semmi köze a rendszernek az MS Windows-hoz. Ott a grafikus felület egybeépült az operációs rendszerrel. Hasonlóság csak a kezelésben van, hiszen itt is ablakok, ikonok, nyomógombok, stb. vannak.

A rendszer alapja a **szerver-kliens** (kiszolgáló-ügyfél) elv. Nem az általában megszokott helyzet érvényesül, hogy a felhasználó gépén fut a kliens program, míg a szerver program pedig a nagy teljesítményű kiszolgálón. Itt a szerver program a felhasználó számítógépén (munkaállomásán) fut, a kliens program pedig a nagy teljesítményű UNIX-os számítógépen. (Persze a szerver és kliens ugyanazon a számítógépen is futhat.)

A felhasználói program a UNIX-gépen fut, és ha meg kell például jeleníteni egy ablakot, akkor a kliens a szerverhez fordul, és majd az fogja megjeleníteni a kért objektumot.

A szerver és kliens TCP/IP hálózaton kommunikál egymással, ezért akár (megfelelő sebesség esetén) az Interneten keresztül is tarthatnak kapcsolatot egymással.

Van egy különleges kliens, az **ablakkezelő,** amelynek alapvető feladata az ablakok megjelenítése, méretezése, áthelyezése. Az ablakkezelő felelős az ablakok külső megjelenéséért (milyen a kerete, milyen gombokkal lehet kezelni, stb.) is.

13. FELADATOK

13.1 Számítástechnikai alapfogalmak

1. *Mi a különbség a RAM és a ROM között?*
2. *Mi a CPU, és mi a feladata?*
3. *Írd fel a következő decimális szám bináris megfelelőjét: 196,17!*
4. *Írd fel a következő bináris szám decimális megfelelőjét: 100111001,1001!*
5. *Határozd meg a következő decimális szám hexadecimális alakját: 3960,78!*
6. *Határozd meg a következő hexadecimális szám decimális alakját: B30,1C!*
7. *Írd fel a következő decimális szám 16 bites, fixpontos ábrázolását : -139!*
8. *Határozd meg a következő decimális szám 6 byte-os lebegőpontos alakját!*
9. *Mondj példát olyan esetre, amikor a redundancia hasznos !*
10. *Mi a bit?*
11. *Mi a MIPS és MFLOPS?*

13.2 Algoritmusok

1. *Milyen feltételeknek kell megfelelni az algoritmusoknak?*
2. *A következő deklaráció egy Turbo Pascal programban szerepel:* ***Var i: Integer*** *; Az* ***i*** *változó mely jellemzőit adtuk meg ezzel közvetlenül, illetve közvetve, és melyik az, amelyről még a továbbiakban kell gondoskodnunk?*
3. *Hogyan történhet egy program tesztelése?*
4. *Készíts folyamatábrát, amely a másodfokú egyenlet megoldására ad egy algoritmust (a megoldó képlet felhasználásával)!*

5. *Készíts algoritmust, amely egy természetes szám valódi osztóit keresi meg!*
6. *A logaritmikus keresés segítségével készíts algoritmust, amely megkeresi egy osztály tanulói közül a Kis Virág nevűt!*
7. *Készíts algoritmust, amely egy osztály tanulóit sorrendbe állítja!*

13.3 Hardver alapismeretek

1. *Mi a különbség a Pentium valós és védett módja között?*
2. *Milyen szerepet játszanak a megszakítások a számítógép működésében?*
3. *Mit nevezünk rendszerhívásnak?*
4. *Mi a DMA szerepe?*
5. *Mik a FAT32 és VFAT jellemzői?*
6. *A B800:0002 szegmentált címnek melyik fizikai cím felel meg?*
7. *Mire használhatjuk a HMA, illetve UMB memóriaterületeket?*
8. *Miért használják a cache memóriákat?*
9. *Mi a virtuális 8086-os mód?*
10. *Az 5,25" -os , HD-s lemez kapacitása 1,2 Mb. Mutasd meg, hogy ez igaz, a lemez fizikai felépítésére vonatkozó adatok segítségével!*
11. *A hajlékonylemezen egyetlen állomány van, mérete 778 byte. A lemez teljes kapacitása 362496 byte, a szabad lemezterület pedig 361472 byte. Hogyan lehetséges ez?*
12. *Mi az oka annak, hogy az állományokat szétdarabolva tárolják a lemezen?*
13. *Hogyan történik a fájlt alkotó klaszterek összeláncolása?*
14. *Hasonlítsd össze a hajlékonylemez és a winchester logikai felépítését?*
15. *Mi a színmélység?*
16. *Mi a feladata oz ON LINE kapcsolónak a nyomtatókon?*

13.4 Az operációs rendszer fogalma

1. *Mi a szerepe az operációs rendszernek a számítógépben?*
2. *Az MSDOS az operációs rendszerek fontosabb funkciói közül melyeket valósítja meg?*
3. *Milyen programokat ismersz, amelyek használják az overlay technikát?*
4. *Mi a virtuális tárkezelés?*
5. *Miért nem alkalmas az MSDOS a multitaskingra?*
6. *Mi a különbség a folyamat és program között?*

13.5 Adatállományok

1. *Mit nevezünk egy rekord azonosítójának?*
2. *Mi a blokk fogalma?*
3. *Mi a szerepe a pufferelésnek?*
4. *Hogyan lehet megvalósítani a legfontosabb adatállományokra vonatkozó műveleteket a Turbo Pascalban, illetve a dBase-ben?*
5. *Mi a szerepe a rekordmutatónak a dBase-ben?*
6. *Miért érdemes a indexelést használni?*

13.6 Adatbázisok

1. *Mik a hagyományos adatkezelés hátrányai?*
2. *Mi az adatfüggetlenség elve?*
3. *Mi a relációs adatbázis-kezelő?*
4. *Milyen műveleteket lehet végezni az adatbázisokon?*

13.7 Fordítók és értelmezők

1. *Mi a különbség az assembler és az assembly fogalma között?*
2. *Mi a lexikális elemzés célja?*
3. *Mi a szintaktikai elemzés célja?*
4. *Mi van szükség a kapcsolatszerkesztőre?*
5. *Mi a különbség a fordítók és értelmezők között?*

13.8 Hálózatok

1. *Mi a célja hálózatok létrehozásának?*
2. *Milyen hálózati topológiákat ismersz?*
3. *Jellemez a csomagkapcsolt és vonalkapcsolt hálózatokat!*
4. *Mi a router?*
5. *Jellemezd a CSMA/CD protokollt!*
6. *Mi a TCP és az IP protokolloknak?*
7. *Mi a base64 kódolás?*
8. *Mi a hálózati maszk?*
9. *Hogyan történik az útvonalválasztás az Interneten?*
10. *Mi a proxy szerver?*

13.9 Linux

1. *Mi a megosztott könyvtárak szerepe a programok futtatásában?*
2. *Milyen fájltípusokat ismersz?*
3. *Mi a mountolás?*
4. *Mi az inode szám szerepe?*
5. *Mi a különbség a merev- és a szimbolikus láncolás között?*
6. *Mi a felhasználói jogosultságok szerepe fájlok ill. könyvtárak esetén?*
7. *Hogyan tudod a folyamatokat kilistázni?*

14. IRODALOMJEGYZÉK

Számítástechnika középfokon *(OMIKK, 1987.)*

Szlávi Péter - Zsakó László:
Módszeres programozás: Programozási tételek *(µlógia 19.)*

Hack Frigyes:
Informatikai ismertek *(µlógia 31.)*

Szlávi Péter - Zsakó László:
Módszeres programozás *(Műszaki Könyvkiadó, 1986.)*

Racskó Péter:
Bevezetés a számítástechnikába *(SZÁMALK, 1989.)*

Csépai János:
A számítástechnika alapjai *(Műszaki Könyvkiadó , 1985.)*

Tuba Péter:
Számítógép-alapismeretek *(KSH, 1981.)*

Fülöp Géza:
Ember és információ *(Múzsák Közművelődési Kiadó, 1983.)*

Peter Norton:
Az IBM PC programozása *(SZÁMALK , 1989.)*

Ila László-Sághi Balázs:
PC-MŰHELY 1,2,*3 (Panem-McGraw-Hill,1996.)*

Bordás Gábor-Páskuj Attila:
Multimédia középiskolásoknak *(Pedellus Novitas Kiadó, 1999.)*

Cserny László:
Mikroszámítógépek *(LSI Oktatóközpont 1996.)*

Dr. Kovács Magda:
32 bites mikroprocesszorok 80386 I.-II. *(LSI, 1991.)*

Inotai lászló - Lázár László:
IBM XT/ AT rendszerprogramozás I, II, III *(Novotrade , 1991.)*

Bakos Tamás- Zsadányi Pál:
Operációs rendszerek I, II *(SZÁMALK, 1992.)*

Knapp Gábor:
Operációs rendszerek *(LSI ktatóközpont, 1998.)*

Varga László:
Rendszerprogramozás I, II, III *(Tankönyvkiadó, 1991.)*

Kónya László:
Számítógépes hálózatok *(LSI Oktatóközpont)*

Nagy Sándor:
Internet és Intranet IntranetWare hálózaton *(COMPUTERBOOKS, 1997.)*

Salátné Bocsi Éva:
Adatállományok tervezése kezelése *(SZÁMALK , 1989.)*

Quittner Péter - Kotsis Domokos:
Számítástechnika rendszerszervezőknek *(Akadémiai kiadó, 1989.)*

Bognár Júlia:
dBase III PLUS *(SZÁMALK)*

Bartók Nagy János- Lauper Judit:
UNIX felhasználói ismeretek *(Openinfo, 1994.)*

Jedlovszky Pál:
UNIX lépésről lépésre *(LSI Oktatóközpont.)*

Csórián Sándor:
Az OS/2 és a Linux állományrendszere *(Számítástechnika, 1998. XIII/45.)*

Brian W. Kernighan-Rob Pike:
A UNIX operációs rendszer *(Műszaki könyvkiadó , 1987.)*

Stefan Strobel-Thomas Uhl:
Linux *(Kossuth Könyvkiadó, 1996.)*

A Pedellus Novitas Kft.

informatikai és számítástechnikai kiadványai:

- *Számítástechnika 5–6. osztálynak – tankönyv*
- *Számítástechnika 5–6. osztálynak – munkafüzet*
- *Számítástechnika 7. osztálynak – tankönyv*
- *Számítástechnika 7. osztálynak – munkafüzet*
- *Számítástechnika 8. osztálynak – tankönyv*
- *Számítástechnika 8. osztálynak – munkafüzet*
- *Számítástechnika 9. osztálynak – tankönyv*
- *Számítástechnika 9. osztálynak – munkafüzet*
- *Számítástechnika 10. osztálynak – tankönyv*
- *Számítástechnika 10. osztálynak – munkafüzet*
- *Számítástechnika középiskolásoknak*
- *Számítástechnika és DOS 6.22 alapismeretek – tankönyv*
- *Számítástechnika és DOS 6.22 alapismeretek – munkafüzet*
- *Számítástechnika – Norton Commander*
- *Az editortól a szövegszerkesztőig*
- *Szövegszerkesztő – Word for Windows 6.0*
- *Táblázatkezelő – Excel*
- *Turbo Pascal iskolásoknak*
- *Számítástechnikai feladatgyűjtemény*
- *Út a forráshoz 7. osztálynak – könyvtárhasználat*
- *Multimédia középiskolásoknak – elméleti és gyakorlati ismeretek*

Megrendelhetők:
Pedellus Novitas Kft.
4001 Debrecen, Pf. 430

9 638397 160000

Raktári szám: PL – 0012