Számítógépes architektúrák I.

Antal Péter

MÉDIAINFORMATIKAI KIADVÁNYOK

Számítógépes architektúrák I.

Antal Péter

Eger, 2013

Korszerű információtechnológiai szakok magyarországi adaptációja

TÁMOP-4.1.2-A/1-11/1-2011-0021

Lektorálta:
Nyugat-magyarországi Egyetem Regionális Pedagógiai Szolgáltató és Kutató Központ

Felelős kiadó: dr. Kis-Tóth Lajos
Készült: az Eszterházy Károly Főiskola nyomdájában, Egerben
Vezető: Kérészy László
Műszaki szerkesztő: Nagy Sándorné

Tartalom

1. BEVEZETÉS

CÉLKITŰZÉSEK, KOMPETENCIÁK A TANTÁRGY TELJESÍTÉSÉNEK FELTÉTELEI

Célkitűzés

A hallgató ismerje meg a számítógépek működésének matematikai alapjait, a működés logikai rendszerét, a Boole-algebra alapjait. Ismerje meg a számítógéprendszerek alapvető működését felépítését, processzorokkal, az operációs rendszerek koncepcióival. Sajátítsa el az számítógépes architektúrák rendszerszintű felépítését, az operációs rendszerek gépi szintű alapjait. Ismerje az alapvető adattárolási architektúrák működését, az operációs rendszerek és a hardver gépi szint alatti kapcsolatát. Legyen tisztában az operációs rendszerek felépítésével, típusaival, alkalmazási területeivel.

Kompetenciák

Alapvető matematikai ismeretek megszerzése a számítógépes architektúrák működésének szempontjából.

Rendelkezik az architektúrák felépítésének elsajátításához, szükséges logikai, rendszertervezői ismeretekkel, valamint folyamatszervező és irányító képességekkel.

Ismeretekkel rendelkezik a számítógépek felépítésének rendszerkövetelményeiről, a belső adattárolás komplex működéséről és felépítéséről.

Ismeretekkel rendelkezik az operációs rendszerek logikai felépítéséről, és rendszerszintű kapcsolatával, a hardverrel.

Releváns ismeretekkel rendelkezik az operációs rendszerek használatáról, szolgáltatásairól.

Attitűdök / nézetek

- Alakuljanak ki azok a nézetek, kompetenciák, amelyek a számítógépes architektúrák, működtetéséhez és továbbfejlesztéséhez szükségesek.
- A tanultak ismeretében, alakuljon ki a konstruktív, új ötletek és megoldások iránti igénye, a problémaelemzés, feltárás, megoldás iránti attitűdje.

- A tananyag elsajátítása révén fejlődjön logikai készsége, érzékenysége a problémamegoldó gondolkodás iránt.

Képességek

- Rendelkezik a számítógépek működésének, elméleti felépítésének matematikai alapjaival.
- Elsajátítja a számítógépes architektúrák rendszerszintű elemzését, logikai felépítésük absztrakciós képességet.
- Képes a rendszeralapú problémamegoldásra, tisztában a rendszerszintek felépítésével, a hardver és szoftver strukturális kapcsolatával.
- Az ismeretek alapján alakuljon ki a rendszerező, logikai, kombinatív képessége, legyen képes konstruktív, innovatív személyes és szervezeti stratégiákat alkalmazni, cselekvési programokat kialakítani és megvalósítani.

2. MATEMATIKAI ALAPOK – SZÁMRENDSZEREK

2.1. A SZÁMOLÁS TÖRTÉNETE[1]

Számolás pálcikákkal, kavicsokkal, avagy hogyan számoltak őseink?

A számolás képessége gyakorlatilag egyidős az emberrel. Igaz, a kezdetben, az ősember még nem úgy számolt, mint ahogyan ma számolunk, de összeadnia és kivonnia – bizonyos határok között – már neki is tudnia kellett.

Tudnia kellett, hogy mennyi állatot kell a vadászaton elejtenie ahhoz, hogy a családja elég élelemhez jusson. Így aztán nyilvánvalóan képes is volt összeszámolni a családja tagjait éppúgy, mint az elejtett állatokat.

Vajon mit használhatott ehhez akkoriban az ősember? Először is ott volt már akkor is a kezén az a néhány ujj. Aztán ha az ujjak már nem voltak elegendőek, lehetett rovátkákat rajzolni a földre, a fába vagy a falra. Egyes elképzelések szerint a sziklarajzoknak is lehetett az a szerepe, hogy leltárba vegyék az elejtett állatokat, a megtermelt növényeket, illetve a törzs létszámát.

Régészeti leletek bizonyítják, hogy egyes ókori népek kis kövecskéket helyeztek agyagedényekbe, hogy így jegyezzék fel valaminek a mennyiségét.

A kőkorszaki kultúrákban, a számokat fából vagy kövekből faragott „pálcikák" reprezentálták. Az ősi amerikai indián csoportok a pálcikákat lovak, szolgák, személyes szolgáltatások adás-vételénél, illetve szerencsejátékoknál használták.

A legelső írott emlékeket a pálcikák használatáról a sumérek hagyatékai között találták, agyagtáblákba karcolták, amelyeket később néha kiégettek. A sumérek a kissé különleges, a 10-es, 12-és és 60-as alapú számrendszerek kombinációját használták az asztronómiai és egyéb számításaiknál. Ezt a rendszer átvették és az asztronómiában használták az ősi mediterrán nemzetek (akkádok, görögök, rómaiak és egyiptomiak). A rendszer maradványait könnyen felismerhetjük a mai idő- (órák, percek) és a szögmérésben (szögpercek).

Kínában a katonák és a gazdálkodók már a maradékokat is használták a számításaikban (prímszámok). A csapatok számának, illetve a rizs mennyiségé-

[1] http://www.arcania.hu/Informatika/intro/counters.html

nek méréséhez a pálcikák egyedi kombinációi szolgáltak. A számításokat kényelmesebbé tette a moduláris aritmetika, amely megkönnyítette a szorzást. A moduláris aritmetika használata egyszerűvé tette a számításokat. A moduláris aritmetikát ma a digitális jelfeldolgozás használja.

A Római Birodalomban a pálcikákat viaszba vagy kőbe karcolták, vagy papiruszra írták, és a számok ábrázolására a görögöktől átvett rendszert használták, de egyes számokra saját jeleket vezettek be. A római számrendszer használata a helyiérték-rendszer bevezetése előtt (1500-as évek) általános volt.

A közép-amerikai maja kultúra egy 20 vagy 18 alapú számrendszert használt, ismerték már a helyiértékeket és a nulla fogalmát. Nagyon pontos asztronómiai számításokat végeztek, különösen az év hosszával és a Vénusz pályájával kapcsolatban.

Az Inka Birodalom kiterjedt gazdaságirányítási rendszert működtetett kipu, ahol pálcikák helyett színes fonalakra kötött csomókat használtak. A csomók és színek használata a spanyol hódítók a 16. században történt megjelenésével feledésbe merült, ennek ellenére egy kipuhoz hasonló egyszerű jelzésrendszer még ma is használatos az Andok területén.

Néhány szerző azt feltételezi, hogy a helyiérték-rendszert széles körben az abakusz használatával a kínaiak terjesztették el. Az első írásos emlékek a pálcikákról, illetve az abakusz használatáról 400 körüliek. A kínai matematikusok a nullát csak 932 körül írták le.

Indiából, ahol már ismerték a modern helyiértékes rendszert, valószínűleg egy Indiába küldött követ által egy 773 körül vásárolt asztronómiai táblázat közvetítésével jutott el a rendszer az arabokhoz. A rendszerek részleteit lásd arab számok és indiai számok.

A iszlám fejedelmek és Afrika, valamint az India közötti élénk kereskedelem juttatta el az indiaiak által használt rendszert Kairóba. Az arab matematikusok kibővítették az általuk addig használt rendszert a decimális hatványokkal, amit al-Hvárizmi a 9. században már írásban rögzített. A rendszerrel Európát Fibonacci a *Liber Abaci* 1201-ben, Spanyolországban megjelent munkájában ismertette meg, lefordítva az arab forrást. Így Európába a 12. században jutott el arab közvetítéssel a nullával kiegészített teljes indiai rendszer.

A 2-es alapú bináris rendszert már a 17. században Gottfried Leibniz ismertette, aki Kínában hallott róla, de általános használata a 20. században, a számítógépek megjelenésével terjedt el.

2.2. HELYIÉRTÉKES SZÁMOLÁS, SZÁMOLÓTÁBLÁK

Felmerül a kérdés: mi van, ha olyan nagy számokkal akarunk dolgozni, hogy a kavicsok és pálcikák már nem elegendőek a biztos eredményhez. Ekkor is lehet persze még használni a kavicsokat, mint ahogyan használták is őket több helyen is.

Kis agyagtáblácskákba vonalakat vésték, és ebbe pakolták a kavicsokat. Ezután már a kavics pontos értéke attól is függött, hogy melyik barázdába helyezték. Ezzel kialakult a helyiértékes számolás, amit nagyon sok helyen használtak különféle változatokban.

1. ábra: Lee Kai-Chen féle abakusz

A számolás latin eredetű neve, a **calculare** – amely a számológépek angol eredetű kalkulátor elnevezésének is az eredete – a kavics, azaz **calculus** szóból származik.

A különféle számolótáblák az egyes népeknél más-más néven szerepeltek, de működésük hasonló volt. Kínában például **szuan-pan** volt a neve (szó szerint számolótáblát jelent), Japánban **soroban** a neve, a görög változatot **abakusz** néven ismerjük. Ezek persze már egy kicsit továbbfejlesztett változatai az eredetinek, hiszen itt a kavicsok helyett drótokra felfűzött golyók mozgatásával lehet számolni, de az elv megegyezik a fentebb említettel, amelyet a következőkben az inkák számolótábláján fogunk bemutatni.

Az inkák a számok ábrázolására, sőt, sokáig való tárolására, például az adók beszedésénél, és a termés mennyiségének elkönyvelésénél egy érdekes, szintén helyiértékes rendszert alkalmaztak:

Minden madzagra egy szám került, ami több csomóból állt. Minden csomó egy helyiérték volt, egy csomón pedig annyi hurok volt, ahány az adott helyiértékből volt a számban. A madzagokat pedig egy másik kötélre erősítve máris nyilván lehetett tartani akár egy egész város adóját is.

Összeadásnál a hozzáadandó helyiértéket rácsomózták az addigi csomóra, majd ahol az alapszámnál több lett a csomó, ott lecsomóztak annyit, amennyi az alapszám volt, és a következő helyiértékre csomóztak egyet.

Az inka számolótábla mélyedései négy oszlopba és valahány sorba voltak rendezve. A sorok jelentették a helyiértékeket, de az oszlopoknak is volt jelentésük: balról kezdve 1,2,3 és 5 volt a bele helyezett kő értéke.

Egy rekeszben egy kő lehetett. Az ábrázolt szám értéke úgy állt össze, hogy soronként a rekeszekben levő kövek értékét összeadva az adott sor helyiértékét kaptuk. (A helyiértékek 1, 10, 100, 1000 stb. voltak).

Két szám összeadásánál belerakjuk mindkét számnak megfelelően a kavicsokat a mélyedésekbe, majd optimalizálni próbáljuk a kövek számát az alábbi szabályok szerint:

- két 5-ös kő (10 az értéke) helyett a következő sorba egy 1-es kő kerül
- két 1-es kő (2) helyett egy 2-es kő
- két 2-es kő helyett (4 az értéke) egy 1-es és egy 3-as kő
- két 3-as kő helyett (6 az értéke) egy 1-es és egy 5-ös kő
- egy 1-es és egy 2-es kő (3 az értéke) helyett egy 3-as kő
- egy 2-es és egy 3-as kő (5 az értéke) helyett egy 5-ös kő

Ha kipróbáljuk, kiderül, hogy az abakusz sokkal egyszerűbb, hiszen ott kevesebb a szabály.

Az abakusz római változata a római számokkal való számoláshoz való, amely nem helyiértékes számrendszer. Itt egy sort két részre, az egyesekre és az ötösökre kell osztani.

Ezután a szabályok:

Ha az egyesek száma eléri az ötöt, vegyél el öt követ, és adj egyet az ötösökhöz.

Ha az ötösök száma egynél több, vegyél el két követ, és a következő sorba tedd az egyiket az egyesekhez.

Persze később egyre bővült az emberiség számtantudása. Elsősorban az araboknak volt ez köszönhető a középkorban. Nekik köszönhetjük például a jól ismert, tízes számrendszerbeli számjegyeinket: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Persze ezek nem voltak mindig ilyenek, ma már némelyiknek az eredeti alakjára aligha ismernénk rá.

Szintén az araboknak köszönhetjük az **algoritmus** szót is, amelyet később még sokat fogunk emlegetni. A XII. századig Európában még az összeadás és a kivonás is egyetemi tananyagnak számított. Persze ezen talán nem csodálkozunk annyira, ha figyelembe vesszük, hogy ekkor még egész Európában a római számokat használták.

Keleten ekkor már megjelentek az indiai eredetű számjegyek, amelyek az általunk arab számokként ismert fenti számjegyek ősei voltak. Ezen kívül a XI. században megjelent egy könyv egy bizonyos **Muhammed ibn Muza al-Chvarizmi** tollából, amelyben a legfontosabb számolási módszereket, eljárásokat szinte receptkönyv alakjában adta meg.

Sokakban él az a téves elképzelés, hogy az ő nevéből származik az algoritmus szó. A valóság inkább az, hogy a könyv latin nyelvű címéből: **Algoritmi dicit**. Ebből származtatva a feladatok megoldásának az egymás utáni lépések sorozatával megadott eljárását **algoritmus**nak nevezzük.

2.3. SZÁMRENDSZEREK

A fentiekben már szó esett a számrendszerekről. Nézzük most meg röviden, hogy mi is ez.

> **Amikor egy mennyiséget (számértéket) úgy írunk föl, hogy ugyanazokat a számjegyeket használjuk egymás után többször, és ezeket úgy tekintjük, hogy az egyes számjegyek helye más-más értéket jelent, akkor beszélhetünk helyiértékes számábrázolásról.**

A használt számjegyek száma az ábrázoláshoz használt számrendszer **alapszáma**. Ez egyben az a szám is, amellyel a számjegyet meg kell szorozni valahányszor, hogy az adott helyiértéken megkapjuk a számjegy által jelzett értéket. Hogy hányszor kell megszorozni az alapszámmal, az attól függ, hogy jobbról hányadik jegyről van szó: a legjobboldali mindig az egyes helyiérték, így azt nem szorozzuk. A következőt egyszer kell szorozni, és minden továbbit egyel többször.

Az így kapott értékeket összeadva kapjuk a szám értékét.

Példaként ez az általunk használt tízes számrendszerben így néz ki:

Helyiérték:	**1000**	**100**	**10**	**1**
Számjegy:	**2**	**3**	**9**	**1**

Vagyis: 2*1000 + 3*100 + 9*10 +1= 2000+300+90+1=2391.

A tízes számrendszeren kívül a köznapi életben még más számrendszereket is használunk. Például a babiloni eredetű 60-as számrendszert az idő- illetve szögmérésben. Vagy a 12-est szintén az időmérésben, de a tucat elnevezés is a 12-es számrendszerre utal.

2.4. A BINÁRIS SZÁMRENDSZER TÖRTÉNETE

Ismereteink szerint Pingala ókori indiai matematikus mutatta be először a bináris számrendszert i.e. 800-ban. Az ókori Kínában íródott klasszikus I Ching-ban leírt hexagrammok sorrendbeli bináris elrendezését a 11-ik századi Shao Yong kínai tudós és filozófus végezte el, amely megfelelt a 0-tól 63-ig tartó decimális számsornak – ennek ellenére nincs bizonyíték arra, hogy Shao értett a bináris számrendszerben történő műveletek elvégzéséhez. 1605-ben Sir Francis Bacon egy olyan rejtjeles írásmódot mutatott be, amelyben az ábécé betűit bináris betűsorokkal helyettesíti, s így egy rejtett üzenetet tud kódolni egy tetszőleges szövegtestben két eltérő betűkép használatával. A bináris kódolás fontos mozzanataként hozzátette, hogy ez a módszer bármilyen más formában is használható, feltéve, ha ebben a formában kétfajta eltérés tisztán megjeleníthető.

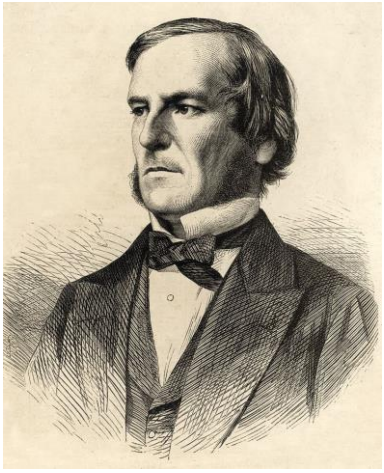
2. ábra: bináris kód

A modern bináris számrendszert teljes egészében Gottfried Leibniz dokumentálta az „Explication de l'Arithmétique Binaire” című munkájában a 17.

században. A Leibniz által leírt rendszer nullát (0) és egyest (1) használt, mint a mai bináris számrendszer.

3. ábra: Leibniz

1854-ben George Boole brit matematikus egyik kiadott művében egy olyan logikai rendszert részletezett, amely később a „Boolean algebra” néven vált ismertté. Ez a logikai rendszer nagyon hasznosnak bizonyult a bináris számrendszer megalkotásában, különösen elektromos áramköri alkalmazásban.



4. ábra: Boole

1937-ben George Stibitz, aki akkor a Bell Labs-nél dolgozott, megalkotott egy reléalapú számítógépet, amelyet „K Modell”-nek nevezett el (K, mint

„Kitchen” – Konyha – ahol összeszerelte a gépet) és amely bináris számolásra volt képes. Ezek után 1938 második felében a Bell Labs egy teljes kutatóprogramot engedélyezett Stibitz vezetésével. Az 1940. január 8-án elkészült számítógépük bonyolultabb műveletek elvégzésére is képes volt. 1940. szeptember 11-én a Dartmouth Főiskolán megrendezett Amerikai Matematikai Társaság konferenciáján Stibitz képes volt a számítógépnek parancsokat küldeni telefonvonalon keresztül telex segítségével. A konferencia részvevői közül néhány: Neumann János, John Mauchly és Norbert Wiener, akik tanúi voltak a bemutatónak és írtak is róla az emlékirataikban. (WEB)

Neumann János az elektronikus számítógépek logikai tervezésében kiemelkedő érdemeket szerzett. Ennek alapvető gondolatait – a kettes számrendszer alkalmazása, memória, programtárolás, utasításrendszer – Neumann-elvekként emlegetjük. Ő irányította az EDVAC – az első olyan számítógép, amely a memóriában tárolja a programot is – megépítését 1944-ben, amelyet 1952-ben helyeztek üzembe. A számítógépnek köszönhetően világszerte óriási tekintélye lett. Ennek a számítógépnek a terve és az ő továbbfejlesztett elmélete (Neumann-elvek) alapján készülnek a mai számítógépek is.

5. ábra: Neumann

1973-ban Claude Shannon az MIT-nél beadott diplomamunkájában a Boolean algebrát és a bináris számtant alkalmazta elektromos relék és kapcsolók használatával a történelemben először. A *„Symbolic Analysis of Relay and*

Switching Circuits" címmel megalkotott diplomamunka megalapozta a gyakorlati digitális áramkörtervezést.

2.5. A BINÁRIS SZÁMRENDSZER

A számítógép, mint már említettük, egy két jelből álló jelkészlettel dolgozik. Ez a két jel a számítógép számára két különböző feszültségszint (az egyik általában a feszültség hiánya szokott lenni), azonban ezeket 0 és 1 jelekkel szoktuk szemléltetni, mivel a kettes számrendszerben is ezt a két számjegyet használjuk. Mint a következőkből kiderül, a kettes számrendszerben számolni is sokkal könnyebb, mint a hétköznapi életben használt tízes számrendszerben.

2.5.1. Átváltás kettes és tízes számrendszer között

Ahhoz, hogy tudjunk kettes számrendszerben számolni, előbb fel kell tudnunk írni a hétköznapi életben használt tízes számrendszerből a számokat kettes számrendszerbe. Ehhez először nézzük meg, hogy hogyan is néznek ki a kettes számrendszerben a számok.

A kettes számrendszerbeli számok felépítése

Minden helyiértékes számábrázolási rendszerben, így a kettes vagy a tízes számrendszerben is a számot valahány különböző számjegyből gazdálkodva írjuk fel. A rendelkezésre álló számjegyek száma valójában a számrendszer alapszáma.

Amikor a számjegyeket egymás mellé írjuk, akkor minden számjegy azt mondja meg, hogy az adott helyiértékből mennyi van a számban. Hogy mi a helyiérték? Nos, az szintén az alapszámtól függ. A helyiértékek mindig jobbról balra nőnek. A jobb oldali számjegy mindig az egyesek számát adja meg, majd tőle balra haladva minden számjegyet eggyel többször kell az alapszámmal megszorozni, mint a tőle jobbra levőt. Tehát a helyiértékek nem mások, mint az alapszám hatványai a nulladik hatványtól (ami 1-gyel egyenlő minden szám esetén) kezdve.

Ennek megfelelően tehát kettes számrendszerben a 100101 valójában a következő:

- 2^0=1-ből van egy darab
- 2^1=2-ből van nulla darab
- 2^2=4-ből van egy darab
- 2^3=8-ból és 2^4=16-ból van nulla darab
- és 2^5=32-ből van egy darab

Ha összeadjuk a nem nulla értékeket, akkor megkapjuk a szám értékét, ami éppen 37. Ezzel tulajdonképpen kaptunk egy átváltási módot kettes számrendszerből tízes számrendszerbe. Másrészt az is látható, hogy ha valaki tud kettővel szorozni, és így fel tudja írni a kettő hatványait, akkor az átváltás nem lehet gond, mivel a kettes számrendszerben egy adott helyiértéket vagy figyelembe kell venni (mert 1 az értéke), vagy nem kell figyelembe venni (mert 0 az értéke), így csak bizonyos kettő-hatványok összeadására van szükség. Egyetlen más számrendszerre sem igaz ez.

Átváltás kettes számrendszerre

Most már tudjuk, hogy mit kell kapnunk: olyan felbontását a számnak, amelyben kettő hatványainak összegére bontottuk a számot. Hogy lehet ezt a legkönnyebben meghatározni?

Ez majdnem olyan, mint amikor valamilyen pénzösszeget kell meghatározott címletekben minél kevesebb érmével kifizetni. Most az egyes címletek megkaphatók kettővel osztás illetve szorzás révén.

A legkisebb címlet nyilván akkor kell, ha az eggyel nagyobb címlettel nem osztható a szám, a következő címlet akkor, ha az eggyel nagyobbal nem osztható a megmaradt összeg és így tovább.

Az átváltás módszere ennek megfelelően a következő: osszuk el a számot kettővel, és írjuk fel a maradékot (ez lesz az egyes helyiérték). Az osztás eredményével ismételjük meg a műveletet mindaddig, amíg az nulla nem lesz. Így jobbról balra haladva mindig egy-egy helyiértéket kapunk meg a szám kettes számrendszerbeli alakjából.

A módszer természetesen nemcsak a kettes számrendszerbe történő átváltásra igaz, hanem bármely más számrendszer esetén. Viszont mindig annyiféle maradék van, amennyi a számrendszer alapszáma, és kettővel könnyebb osztani, mint a nagyobb számokkal.

A gyakorlatban az átváltást egy egyszerű, nagyon mechanikus módon lehet elvégezni:

- Írjuk fel a számot egy függőleges vonal bal oldalára
- Ha a szám páratan, írjunk mellé a vonal túloldalára egyet, alá a számnál egyel kisebb páros szám felét
- Ha a szám páros, írjunk mellé a vonal túloldalára nullát, alá pedig a szám felét

- Ismételjük meg az előző két lépést a legalsó baloldali számmal mindaddig, amíg az nullává nem válik (tovább nincs értelme, mert mindig nulla marad)
- Ezután alulról felfelé haladva a jobboldali számjegyeket írjuk egymás mellé balról jobbra haladva. A kapott szám a kettes számrendszerbeli alakja a legelőször felírt számnak.

Az alábbiakban néhány példa látható a számolás elvégzésére:

1. *Átváltás kettes számrendszerből tízes számrendszerbe*

37	1	23	1	52	0
18	0	11	1	26	0
9	1	5	1	13	1
4	0	2	0	6	0
2	0	1	1	3	1
1	1	0		1	1
0				0	

100101 10111 110100

A kettes számrendszerbeli számok tízesbe történő átszámolására már láttunk egy módszert: írjuk fel a számjegyek helyiértékét, és ahol egyes van, azokat a helyiértékeket adjuk össze. Ez egy működőképes módszer, de van egy másik módszer is, amely tulajdonképpen ugyanazon az elven működik, de néha jobban, néha kevésbé jól használható, ezért mindkét módszert érdemes ismerni.

Ennél a módszernél nincs szükség a helyiértékek ismeretére, csupán összeadni és kettővel szorozni kell tudni. A módszer a szám legmagasabb helyiértékű számjegyétől halad a legkisebb helyiérték felé a következőképpen:

- Ez a módszer is működik más számrendszerre is, de kettő helyett mindig a számrendszer alapszámával kell a szorzásokat elvégezni, és a lehetséges számjegy is többféle lehet.
- Szorozzuk meg a legmagasabb helyiértéken levő számjegyet kettővel;
- Adjuk hozzá a következő (eggyel kisebb helyiértéken levő) számjegy értékét;
- Amennyiben az éppen hozzáadott számjegy nem a legkisebb (egyes) helyiértékű számjegy volt, akkor az eredményt szorozzuk meg kettővel, és folytassuk a műveletet az előző ponttal, különben megkaptuk az eredményt.

Hogy ez a módszer ugyanúgy helyes, mint a helyiértékek összeadása, az könnyen belátható: a folyamatos szorzások során az egyes helyiértékek minden lépésben eggyel többször kerülnek szorzásra, mint az előző lépésben. Az `abcde` szám átszámolása során a következő történik:

$$((((2*a+b)*2+c)*2+d)*2+e$$

Ha felbontjuk a zárójeleket, akkor a következőt kapjuk:

$$((((2*a+b)*2+c)*2+d)*2+e = e + 2*d + 4*c + 8*b + 16*a = e*2^0 + d*2^1 + c*2^2 + b*2^3 + a*2^4$$

Vagyis a másik módszer szerinti átváltása a számnak...

2.5.2. A számok ábrázolása

A számítógép a számokat mindig meghatározott számú számjeggyel ábrázolja. Mint később látni fogjuk, ez a számolás során is előnyös lesz. Igen ám, de a számok, amelyeket ábrázolni fogunk nem mindig ugyanannyi jegyűek. A megoldás erre nagyon egyszerű: egészítsük ki balról a számot nullákkal, hogy a kívánt mennyiségű számjegyből álljon.

Bit, byte, szó

A számok (és a számok formájában tárolt bármely adat) ilyen adott számjegyen történő ábrázolása vezetett az **információ mértékegysége**inek is tekintett mértékegységek kialakulásához.

A **bit**ről már említettük, hogy az információ legkisebb egysége, amely két különböző információt tud egymástól megkülönböztetni, mivel két értéke lehet. Ez a kettes számrendszer egy számjegyének felel meg. Egy biten tehát egy egyjegyű bináris számot tudunk megadni.

Két biten már $2*2 = 2^2 = 4$ számot, három biten $2^3=8$ számot, négy biten $2^4=16$ számot stb. lehet megadni. **n** biten tehát 2^n féle számot lehet tárolni. A lehetséges értékek pedig a csupa nullával megadott 0 értéktől a csupa eggyel megadott 2^n-1 értékig terjedhetnek.

Eredetileg 6, majd később 7 bitet használtak együtt. Később a 8 bit vált elterjedté, és ez önálló nevet is kapott: **byte**. Az elnevezés onnan ered, hogy az angolban azt, hogy a biteket nyolcasával csoportosítjuk, a **by eight** kifejezéssel lehet megadni. Ez az amerikai angol szokásos beszédstílusában, amikor a gyors beszédben a szó felét elharapják, éppen úgy hangzik, mint ahogyan a **byte** szót kell ejteni.

Bár ma már 8 bitnél nagyobb méretű számokkal dolgoznak a számítógépek, az**információ alapegysége** továbbra is a **byte**, amely tehát nyolc bitet jelent. A rendelkezésre álló memória, vagy egyéb tárolóegységek kapacitását byte-ban szokás megadni.

Azonban a tényleges számméret megadása során szokás használni a **szó** fogalmát is, amely gépenként változhat, és **az együtt kezelt, egyszerre feldolgozható bitek számát adja meg**. Ha tehát például egy processzorra azt mondjuk, hogy 32 bites, az azt jelenti, hogy 32 bites szavakkal képes műveleteket végezni, vagyis legfeljebb 32 bit hosszúságú számok kezelésére képes.

2.5.3. Mértékegységek

Az informatikában használatos legkisebb egység a bit (sok esetben b-vel rövidítik, de a legfrissebb szabvány[2] a rövidítés nélküli formát ajánlja). Értéke 0 vagy 1 lehet. Használhatjuk tárolókapacitás vagy információ jelölésére.

A bájt (byte) az informatika másik legfontosabb egysége, jele: B. Mi az általánosan elfogadott, a gyakorlatban majdnem kizárólagosan használt 1 B = 8 bit átváltást használjuk, bár egyes (egzotikus) architektúrák esetében ennél több vagy kevesebb bit is alkothat egy bájtot.

Az SI-mértékegységrendszerben használatos k (kilo), M (mega), G (giga), T (tera), P (peta) stb. prefixek mellett a bit és a bájt esetében használatosak a Ki (kibi), M i (mebi), Gi (gibi), Ti (tebi), Pi (pebi) stb. *bináris prefixek is* (lásd az 1.1. ábrán). Fontos kiemelni, hogy az egyre nagyobb prefixek esetében egyre nagyobb a különbség az SI és az informatikai/bináris prefixek között. Például a G (1000^3) és Gi (1024^3) között a különbség kb. 7%, a T (1000^4) és Ti (1024^4) között már kb. 10%.[3]

A kapcsolat a prefixek és a számrendszerek között ott fedezhető fel, hogy a használt prefixek mindig a számrendszer alapja valamely hatványának hatványai. Az SI esetben ez a tíz harmadik hatványa (illetve ennek további hatványai), de ugyanez igaz a bináris prefixekre is, amikor is ez a kettő tizedik hatványa (illetve ennek további hatványai).

[2] ISO/IEC 80000, Part 13 - Informat.ion science and technology

[3] Különösen fontos ez a háttértárak esetében, ahol a gyártók inkább az SI-prefixeket használják, mert így egy 1000000000000 B méretű lemezegység esetében 1 TB-ot tüntethetnek fel, míg ugyanez a bináris prefixekkel csupán 0.9 TiB

1. *SI- és bináris prefixek*

prefix	szorzó	prefix	szorzó
k (kilo)	1000	Ki (kibi)	1024
M (mega)	1000^2	Mi (mebi)	1024^2
G (giga)	1000^3	Gi (gibi)	1024^3
T (terc)	1000^4	Ti (tebi)	1024^4
P (peta)	1000^5	Pi (pebi)	1024^5

A 16-os és a nyolcas számrendszer jelentősége

A számítógép belül tehát a számokat kettes számrendszerben tárolja. Azonban elég nehéz a kettes számrendszerbeli számokat olvasni, vagy leírni. Ezért a számítógéppel mélyebb szinten foglalkozók, akiknek a közvetlen számértékekkel is kapcsolatba kell kerülniük, jobban szerették volna, ha nem kettes számrendszerbeli számokkal kell dolgozniuk, de az átváltás sem jelent problémát.

A tízes számrendszer nem a legmegfelelőbb, mivel a tíz nem kettőhatvány. Viszont van két olyan számrendszer, amelynek alapszáma kettő hatványa, és így a kettes számrendszerbeli számok könnyebb ábrázolását teszi lehetővé. Az egyik a régebben, a 6 bites szavak idején használt 8-as számrendszer, a másik a 16-os számrendszer. Az alábbi táblázat egyszerű átváltási útmutatást ad a kettes, a tízes, a nyolcas és a tizenhatos számrendszer között. A számrendszer megnevezésénél szerepel a számrendszer idegen eredetű, gyakran használt neve is.

Tízes (decimális) számrendszer	**Kettes (bináris) számrendszer**	**nyolcas (oktális) számrendszer**	**Tizenhatos (hexadecimáls) számrendszer**
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8

9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Egy byte, azaz nyolc bit esetén még talán nem tűnik túl nagy gondnak a nyolc számjegy leírása, de több byte hosszúságú értékek megadásánál már fárasztó a sok egyes és nulla leírása és olvasása egyaránt. Például a színek kódolására szokás három byte-ot alkalmazni, amelyeknek byte-onként is van külön jelentésük. Itt már valóban kényelmetlen lenne, ha huszonnégy bitet kellene leírni. Helyette rövidebb és olvashatóbb a hat darab tizenhatos számrendszerbeli, azaz hexadecimális számjegy.

Ebből már látható, hogy mire használható a nyolcas illetve a tizenhatos számrendszer: mivel alapszámuk a kettő valamely hatványa, így a kettes számrendszerbeli számot könnyen, és rövidebben fel lehet írni ezekben a számrendszerekben. Három kettes számrendszerbeli számjegy helyett lehet egy nyolcas, vagyis oktális számrendszerbeli számjegyet írni. Négy darab kettes számrendszerbeli számjegy helyett pedig egy darab tizenhatos számrendszerbeli számjegyet írhatunk. Így egy byte értékét pontosan két, tizenhatos számrendszerbeli számjeggyel lehet felírni, ami időnként sokkal kényelmesebb lehet a nyolc darab bit felírása helyett.

A fenti táblázatból az is kiderül, hogy hogyan szokás megoldani azt a problémát, hogy összesen csak tíz számjegyünk van, mivel a hétköznapok során csak a tízes számrendszerben használt számjegyekre van szükség. A tízes értéktől a tizenötös értékig az angol ábécé első hat betűjét használjuk számjegyként. Az, hogy kis vagy nagy betűváltozatot használunk, az mindegy, de lehetőleg következetesen kell vagy az egyiket, vagy a másikat használni.

Példaként az átváltásra:

kettesben:	00110101	11010001	11111111
tizenhatosban	35	D1	FF
nyolcasban	065	321	377

Természetesen a példáknál a nyolcas számrendszerben a legmagasabb helyiértékűjegy 3-nál nem lehet nagyobb, mivel csak két bit jut erre a jegyre...

2.6. SZÁMOLÁS KETTES SZÁMRENDSZERBEN

Most már fel tudjuk írni a számokat kettes számrendszerben, és értelmezni is tudjuk a kettes számrendszerbeli számokat. A következőkben megtanuljuk, hogyan kell ezekkel a számokkal összeadni, kivonni valamint szorozni. Az osztás még kettes számrendszerben is bonyolult műveletnek számít, amelyet a számítógép is több lépésben végez el, így azzal nem foglalkozunk. Azonban mint látni fogjuk, a szorzás kettes számrendszerben sokkal egyszerűbb, mint bármely másik számrendszerben.

A számolás során mindig olyan számokkal fogunk foglalkozni, amelyek 1 byte-on tárolhatók, és ezeket pontosan egy byte-on, vagyis 8 biten fogjuk ábrázolni. A szükséges számú bit elérése mindig a szám balról nullákkal történő kiegészítésével történik.

Elöljáróban még annyit, hogy a kettes számrendszerbeli számolás könnyűségét az alkalmazandó szabályok alacsony száma okozza, ami abból adódik, hogy kevés számjeggyel kell dolgozni, így a számolás során kevés lehetőséget kell figyelembe venni. Először mindig a szabályokat adjuk meg egy művelettábla segítségével, és azután megnézzük, hogyan is kell a gyakorlatban a szabályokat alkalmazni.

2.6.1. Összeadás

Az összeadás művelettáblája a következő:

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

A csillag jelentése itt az, hogy a művelet eredménye kihat a következő helyiértékre is. Ezt a hatást nevezzük **átvitel**nek. Az átvitel azt mondja meg, hogy az összeadás eredménye nem fér el az adott helyiértéken, így a következő helyiértéken az összeadást úgy kell végrehajtani, hogy az eredményhez még egy egyest hozzá kell adni.

Ebből viszont az is következik, hogy nem ilyen egyszerűek a szabályok, hiszen azt is meg kell vizsgálni, amikor három számjegyet adunk össze. A lehetséges esetek tehát a következők:

- Minden jegy 0: az eredmény 0, és nincs átvitel (0+0=0).
- Egy egyes jegy van: az eredmény 1, és nincs átvitel (0+1=1+0=1).
- Két egyes van: az eredmény 0, és van átvitel (1+1=10).
- Három egyes van: az eredmény 1, és van átvitel (1+1+1=11).

Ezt persze lehetne a fenti művelettáblával is szemléltetni, csak kell egy külön művelettábla arra az esetre is, amikor egy átvitelt is figyelembe kell venni. Jelöljük ezt a műveletet $+^*$-gal, és írjuk fel a táblába az esetlegesen keletkező átvitelt is:

+	**0**	**1**
0	00	01
1	01	10

$+^*$	**0**	**1**
0	01	10
1	10	11

Ezek után nézzük, hogy a gyakorlatban hogyan kell két kettes számrendszerbeli szám összeadását elvégezni! Tulajdonképpen ugyanúgy, ahogyan tízes számrendszerben is csináljuk, ha csak papír és ceruza áll rendelkezésünkre (géppel nem is lehetne tízes számrendszerbeli számokat összeadni, mert a számológép is kettes számrendszerben számol):

Írjuk egymás alá a két számot, majd a legkisebb helyiértéktől haladva a fenti szabályok szerint minden helyiértéken számoljuk ki az eredményt, és írjuk az összeadandó számok alá. Amikor az utolsó, legnagyobb helyiértéken is elvégeztük az összeadást, az esetleges átvitelt is írjuk még a szám elé. Az így kapott szám lesz az összeg.

Íme néhány elvégzett számítás:

1. *A pöttyök minden esetben az átvitelre utalnak.*

```
  00101110
+ 00001111
----------
  00111101
```

```
  00100001
+ 00001111
----------
  00110000
```

```
  00110111
+ 00101111
----------
  01100110
```

A túlcsordulás fogalma

A számítógép mindig meghatározott számú biten dolgozik. Példáinknál ez nyolc bit, vagyis egy byte. Azonban az összeadás eredménye a legmagasabb helyiértéken esetleg keletkező átvitel miatt lehet eggyel több, példánkban 9 bit hosszúságú is. Ez a tárolásnál problémát jelenthet.

Ezt az utolsó átvitelként létrejövő többlet bitet nevezzük **túlcsordulás**nak. Látszólag ez problémát okoz, mivel nem tudunk vele mit kezdeni, azonban lehet hasznos is.

Az egyik eset, amikor ezt felhasználják, a több szót elfoglaló, nagyobb számok összeadása. Mivel az összeadás után ugyan ez nem tárolódik el az eredménnyel, azonban rendelkezésre áll az információ, hogy túlcsordulás történt, így fel lehet használni átvitelként a következő szóban található számrészek összeadásához. Így tulajdonképpen egy olyan összeadást lehet megvalósítani, ahol a számjegyek egy-egy szóban vannak tárolva, a számrendszer alapszáma pedig az egy szóban tárolható értékek száma.

Ennél sokkal nagyobb jelentősége lesz azonban a mi számunkra a túlcsordulásnak a negatív számok ábrázolásánál és a negatív számokkal való számolásnál...

Gyakorló feladatok:

Adja össze a következő kettes számrendszerbeli számokat:

1011+1001=

10011+10110=

1111+1010=

2.6.2. Szorzás

A szorzás legalább olyan egyszerű kettes számrendszerben, vagy talán még egyszerűbb is, mint az összeadás. Összesen csak két dolog kell hozzá: a szorzótábla, és annak ismerete, hogy minden számrendszerben, ha egy számot a számrendszer alapszámával szorzunk, az azt jelenti, hogy egy nullát kell a szám végére írni, vagyis minden helyiérték egyel nő.

A szorzótábla kettes számrendszerben így néz ki:

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

Vagyis, ha két egyest szorzunk össze, akkor az eredmény egy lesz, különben pedig 0. Szerencsére most nincs semmiféle átvitel.

Ezzel a szorzótáblával egy kettes számrendszerbeli számot egy egy bites kettes számrendszerbeli számmal már össze tudunk szorozni: ha ez utóbbi egy, akkor az eredmény a másik szám, ha nulla, akkor az eredmény nulla. Azonban mi két több bites számot akarunk összeszorozni.

Ehhez kell az az információ, hogy ha kettővel szorzunk egy számot, az nem más, mint a szám egy helyiértékkel feljebb tolása, és az egyes helyiértékre nulla írása. Ennek ismeretében nem kell mást tenni, mint a két összeszorzandó szám egyikét felírni kettő hatványok összegeként. Ez bonyolultan hangzik, pedig nem más, mint amit az átváltásnál is alkalmaztunk: minden helyiérték kétszerese az előzőnek.

Magyarul arról van szó, hogy szorozzuk meg az egyik számot mindig a másik szám helyiértékével, majd adjuk össze a kapott eredményeket, de úgy, hogy a helyiértékeire bontott számban a helyiértékkel még meg kell majd az eredményt az összeadás előtt szorozni.

Még érthetőbben ez annyit jelent, hogy ha a szorzó számjegyein a legkisebb helyiértéktől haladunk, akkor a részszorzatot minden esetben eggyel toljuk balra az előzőhöz képest (kettővel szorzás), és így adjuk össze a részszorzatokat.

Ez tulajdonképpen ismét a már tízes számrendszerből is ismert szorzás, csak azzal a különbséggel, hogy most sokkal egyszerűbb szorzótáblát kell használnunk: ha a szorzó helyiértéke egy, akkor a részszorzatnak magát a szorzandót kell leírni, ha pedig nulla, akkor a részszorzat nulla.

A részszorzatok összeadását kettesével végezve pedig az előzőekben ismertetett módszert alkalmazhatjuk.

Egy dologra azonban a részszorzatok összeadása kapcsán még figyelni kell: mivel ha egy *n* bites számot megszorzunk egy *m* bites számmal, akkor összesen *m* darab *n* bites szorzat fog keletkezni, amelyek mind egy-egy helyiértékkel eltolva adandók össze. Ez azt jelenti, hogy az eredmény *m*+*n* hosszú lesz. Vagyis két 1 byte-os szám szorzataként egy 2 byte-os számot kapunk. Általában is igaz, hogy két egyforma hosszúságú bináris szám szorzata kétszer annyi bitet igényel,

aminek megfelelően a számítógép a szorzás eredményét mindig a szorzandóknál kétszer hosszabb helyen tárolja.

Szorozza össze a fent leírtakat használva a következő számokat:

10011 * 10100

10011101*10001

10101011*10111

101010*10110

2.6.4. Negatív számok ábrázolása

A negatív számok ábrázolásakor olyan megoldásra van szükség, amelyben egyértelműen megállapítható a számról, hogy negatív vagy pozitív, és amely ábrázolás mellett a matematikai műveletek is könnyen elvégezhetők maradnak.

Ez utóbbi feltétel azt jelenti, hogy olyan ábrázolási módot kell alkalmaznunk, amely esetén az összeadásnál nem kell további szabályokat megadni, ha valamelyik, vagy mindkét szám negatív. Vagy ha mégis kénytelenek vagyunk új szabályt is bevezetni ehhez, minél egyszerűbben alkalmazható legyen.

Amennyiben találnánk egy olyan ábrázolási módot, amelyben az összeadás szabályait egy negatív számra és annak abszolút értékére alkalmazva az összeadás eredménye nulla lenne, akkor ez egy olyan ábrázolási mód lenne, ahol nincs szükség semmilyen új szabály bevezetésére.

A jó hír, hogy van ilyen számábrázolási mód, mégpedig a **kettes komplemens** alak használata. Lássuk először, hogy ez hogyan is néz ki, majd ezután megvizsgáljuk, hogy ez tényleg megfelel-e a céljainknak!

A kettes komplemens alak

A pozitív számok ábrázolása ugyanaz, mint eddig volt, így azzal nem kell foglalkozni. A negatív számoknál pedig a következő módszert kell alkalmazni:

- Írjuk fel kettes számrendszerben, az ábrázolásban használt biteknek megfelelő számú számjeggyel a szám abszolút értékét.
- Fordítsuk meg a biteket: a 0 helyett írjunk 1-et, az 1 helyett pedig 0-át.
- Adjunk hozzá a kapott számhoz 1-et.

- Az így kapott számot nevezzük a szám **kettes komplemens ábrázolásának**.

Megjegyzendő, hogy azért hívják kettes komplemens alaknak, mert az átalakítás során első lépésként (a 2. pont) végzett bit-fordítás (idegen szóval **bitenkénti negálás**) tulajdonképpen a szám **egyes komplemense**.

A kettes komplemens tulajdonságai

Most már bármilyen negatív számot tudunk kettes komplemensben ábrázolni. A kérdés csupán az, hogy ez az ábrázolási mód megfelel-e a kívánalmainknak.

Ehhez először nézzük meg, mit kapunk, ha a -1 és abszolút értéke, azaz az 1 összegét a tanult módon kiszámoljuk. Először a -1 kettes komplemens alakját kell megkapni. Eddigi példáinkhoz hasonlóan most is nyolcbites számokat használunk. Ekkor az 1 alakja: 00000001. Ennek egyes komplemense, vagyis a bitek megfordítása: 11111110. Ehhez egyet hozzáadva 11111111-t kapunk, vagyis ez a -1 kettes komplemensbeli alakja.

Adjuk most össze ezt és a 00000001-t vagyis az 1-et! Rövid számolás után látható, hogy az eredmény 100000000. Azonban a kilencedik bitet nem ábrázoljuk, vagyis az összeadás eredménye 00000000, ami a nulla alakja. Ugyanezt alkalmazva más számokra is, látható, hogy mindig ezt az eredményt kapjuk, vagyis ha figyelembe vesszük, hogy a túlcsorult egyest nem fogjuk tárolni, akkor valóban nullát kapunk minden szám és kettes komplemense összeadásakor.

Figyeljük most meg azt is, hogy hogyan néznek ki a kettes komplemens ábrázolásban a negatív és a pozitív számok! A negatív számoknál a legmagasabb helyiértéknek megfelelő bit mindig 1 lesz, míg a pozitív számok esetén és a nulla esetén ez a bit mindig 0. Éppen ezért ezt a bitet **előjelbit**nek nevezzük. Ez egyértelműen megadja, hogy a számunk milyen előjelű. Természetesen a nullát pozitív számnak mutatja ez a bit.

Ebből már az is következik, hogy a számok előjeles ábrázolásánál majdnem pontosan 50-50% eloszlással ábrázolhatók negatív és pozitív számok, hiszen egy bit értéke éppen a felét adja az ábrázolható számoknak. Igen ám, de a nulla a pozitív számok közül egynek a helyét veszi el. Akkor ezek szerint eggyel több negatív számot tudunk ábrázolni, mint pozitívat. Vajon melyik ez a szám?

Mielőtt ezt megkeresnénk, még egy tulajdonságát nézzük meg a kettes komplemens alaknak: a -1 kettes komplemens alakjából hogyan tudnánk visszakapni az abszolút értékét? A megoldás nagyon egyszerű: képezzük ennek is a kettes komplemens alakját: 11111111 egyes komplemens alakja: 00000000, ehhez egyet hozzáadva pedig 00000001-et kapunk.

Ugyanez igaz minden negatív számra is, vagyis egy negatív számból az abszolút értékét ugyanazzal a módszerrel kapjuk, amivel az abszolút értékből a kettes komplemens alakját kaptuk.

Most pedig térjünk vissza az előző kérdésre: melyik az a szám, amit negatív számként ábrázolni tudunk, de pozitívként már nem? Válaszként egy feladat: számoljuk ki az 10000000 negatív szám abszolút értékét!

Ha ennek a számnak vesszük a kettes komplemens alakját, akkor 10000000-t kapunk. Ez azonban most már pozitív szám, amit átszámolva 128-at kapunk. Mivel a legnagyobb nyolc biten előjelesen ábrázolható pozitív szám (vagyis aminek az előjelbite 0) a 01111111, ami 127, ebből látható, hogy az ábrázolható számok 127 és -128 közé esnek. Ez *n* bit esetén is igaz. Ekkor az előjelesen ábrázolható számok $-(2^{n-1})$ és $2^{n-1}-1$ közé esnek ellentétben az előjelnélküli esettel, amikor ez a tartomány 0 és 2^n-1 között volt.

Számolás előjeles számokkal

Az előjeles számokkal az összeadás és a kivonás nagyon egyszerű: összeadásnál egyszerűen összeadjuk őket, a túlcsordulást figyelmen kívül hagyva. Kivonásnál előbb a kivonandó ellentetjét kell venni, majd ezután összeadást végzünk.

Egy dologra azért érdemes figyelni: ha az eredmény az ábrázolási tartományon kívül esik, akkor hibás eredményt fogunk kapni. Például előfordulhat, hogy két pozitív szám összeadásának eredménye negatív szám lesz, vagy két negatív szám összege pozitív lesz. Ilyenkor az előjelbit nyilván a két szám összeadásakor keletkezett túlcsorulás miatt lesz hamis.

Szorzásnál egyszerűbb a helyzet, mivel nem kell negatív számokkal foglalkozni. Egyszerűen összeszorozzuk a két szám abszolút értékét, majd a matematika szabályai alapján megállapítjuk az előjelet, és ha negatív az eredmény, akkor képezzük a kettes komplemensét.

Összefoglaló feladatok:

Végezze el kettes számrendszerben a következő műveleteket:

25+31

35-12

18-51

21*15

-4*30

50-110

Mennyi a legnagyobb és a legkisebb ábrázolható szám 16 biten, előjeles számábrázolásnál, illetve előjel nélküli számábrázolás esetén?

Két pozitív szám összeadása után -15 lett az eredmény. Mi ennek az oka?

3. MATEMATIKAI ALAPOK – BOOLE-ALGEBRA

3.1. A TANANYAG KIFEJTÉSE

A Boole-algebrát George Boole matematikus (1854) találta fel, és ő szögezte le az alapvető szabályait, mit sem sejtve, hogy ez később a programvezérelt digitális számítógép kidolgozásának matematikai alapjait fogja képezni.

A Boole-algebra informatikai értelemben olyan mennyiségek közötti összefüggések törvényszerűségeit vizsgálja, amelyek csak két értéket vehetnek fel.

A Boole-algebra egyik szegmense a *kijelentés-logika*, amely a logika algebrájának egy interpretációjaként fogható fel, olyan kijelentésekkel dolgozik, amelyek vagy „igazak", vagy „hamisak", és keressük az olyan kijelentések valóságtartalmát, amelyek igaz vagy hamis elemi kijelentésekből tevődnek össze.

A Boole-algebra másik interpretációja a *kapcsolási algebra*. Alapjául olyan kapcsolási elemek szolgálnak, amelyek csupán két, egymástól különböző állapotot vehetnek fel, például egy áramkörben vagy folyik áram, vagy nem; mágneses állapot fennáll vagy sem stb. A kapcsolási algebra azt vizsgálja, hogy az ilyen kapcsolási elemekből összeállított *háló* kimenetén a lehetséges két állapot melyike valósul meg, ha az elemek az egyik vagy másik lehetséges állapotban vannak. Ezért a Boole-algebra az elektronikus digitális számítógép konstruálásának nélkülözhetetlen elméleti alapja.

A bináris, logikai vagy Boole-féle *változóknak* nevezett mennyiségek kétértékűségét két jel bevezetésével fejezik ki. Ezek: „0" és „1" vagy „*O*" és „*L*". A logikai változók közötti összefüggéseket matematikailag a *függvény* fogalmával lehet leírni. Nevezhetjük ezeket logikai függvényeknek, valóságfüggvényeknek vagy kapcsolási függvényeknek.

A matematika tárgyalásához hozzátartoznak az alapfogalmak, tételek és azok matematikai bizonyítása. Mivel a matematikai logika tárgyalása ezen az alapon történik, a fogalmak megismerése segít abban a törekvésben, hogy a tananyag komolyabb matematikai erőfeszítés nélkül is feldolgozható legyen. A tárgyaláshoz nem nélkülözhetjük a matematikai logika alapfogalmait sem, valamint a további leckék kiindulópontját, a logikai műveleteket.

3.1.1. A matematikai művelet

A logikai műveletek tárgyalása előtt a matematikai művelet fogalma következik, melynek segítségével áttekinthető a tárgyalt logikai műveletek néhány általános érvényű jellemzője.

- **A matematikai művelet a halmaz egy, két, vagy több eleméhez a halmaz egy elemét hozzárendelő szabály. Attól függően, hogy a halmaz hány eleméhez rendelünk szabályt beszélünk egy-, két- vagy többváltozós műveletről. (A halmaz valamely algebrai struktúrán értelmezett.)**

Az algebrában leggyakrabban egy- és kétváltozós műveletek fordulnak elő, de tetszőleges számú változóra (argumentumra) definiálható matematikai művelet pl. az igazságfüggvények közül a többtagú konjunkció.

Példa:

Az algebrai műveletek (illetve ezen belül az alapműveletek) közül az összeadás két számhoz a két szám összegét rendeli, így ez kétváltozós műveletnek tekintendő.

A 3×2 szorzásnál a valós számok halmazának két eleméhez rendeljük azok szorzatának értékét, tehát a szorzás szintén kétváltozós művelet.

Egyváltozós művelet pl. a későbbiekben tárgyalt negáció művelete.

A műveletek leggyakrabban operandusokból (pl.: a, b, x, y, P, Tx…), operátorokból (pl. +, -, ×…) és zárójelekből állnak. A matematika számos területén nagy jelentősége van az egyes matematikai művelet-típusoknak, így beszélhetünk többek között halmazműveletekről, logikai műveletekről stb.

A matematikai műveletek aszerint is vizsgálhatók, hogy teljesülnek-e rájuk bizonyos tulajdonságok: asszociativitás, disztributivitás, kommutativitás, idempotencia.

- **Asszociativitás (csoportosíthatóság). Egy több operátort tartalmazó művelet asszociatív, ha a végrehajtás során tetszőleges operátor lehet a kezdőművelet az eredmény megváltozása nélkül, azaz az operandusokat tetszőlegesen csoportosíthatjuk.**

Példa:

a) Az összeadás asszociatív, mivel tetszőlegesen választhatjuk meg azt az operátort (+), ahol az összeadást kezdjük. Jelekkel, ha a, b, c tetszőleges operandusok:

(a+b)+c = a + (b+c), azaz

ha pl. a=3, b=7, c=23, akkor 10+23 = 3 + 30 (=33)

b) Az osztás nem asszociatív, mivel nem választhatjuk meg azt az operátort (+), ahol az osztást kezdjük. Jelekkel, ha a, b, c tetszőleges operandusok:

(a/b) / c ≠ a / (b/c), azaz

ha pl. a=60, b=10, c=2, akkor 6/2 ≠ 60/5

Kommutativitás (felcserélhetőség). Egy művelet kommutatív, ha bármely operátorhoz tartozó két operandust felcserélhetjük az eredmény megváltozása nélkül, azaz a műveletek végrehajtási sorrendje tetszőleges.

Példa:

a) Az összeadás kommutatív, mivel az operátorhoz (+) tartozó operandusokat felcserélhetem. Jelekkel, ha a, b tetszőleges operandusok:

a+b = b + a, azaz

ha pl. a=3, b=7, akkor 3 + 7 = 7 + 3 (=10)

b) A kivonás nem kommutatív. Jelekkel, ha a, b tetszőleges operandusok:

a-b ≠ b-a, azaz

ha pl. a=3, b=7, akkor 3 – 7 ≠ 7 – 3

- **Idempotencia (azonos hatványúság). Egy művelet idempotens, ha ugyanazzal az operandussal a műveletet bármennyiszer végrehajtva mindig önmagát az operandust kapjuk eredményként.**

Példa:

a) Az egy kitevőjű hatványozás idempotens. Jelekkel, ha a tetszőleges operandus:

$(a^1)^1 = a$, *azaz*

ha pl. a=10, akkor $(10^1)^1 = 10$

b) Az összeadás nem idempotens. Jelekkel, ha a tetszőleges operandus:

$a+a \neq a$, *azaz*

ha pl. a=3, akkor $3 + 3 \neq 3$

- **Disztributivitás (széttagolhatóság). Két művelet esetén a zárójelben lévő művelet a zárójel előtt vagy mögött álló művelettel széttagolható úgy, hogy az értéke nem változik. A tagolás lépései: a zárójelben lévő operandusokat rendre a zárójelen kívüli operandussal és a zárójelen kívüli művelettel kötjük össze. Az így keletkezett műveleteket a zárójelben lévő művelettel kötjük össze.**

Példa:

a) A szorzás az összeadásra nézve disztributív. Jelekkel, ha a, b, c tetszőleges operandusok: $a \times (b+c) = a \times b + a \times c$, *azaz*

ha pl. a=5, b=2, c=8, akkor $5 \times 10 = 10 + 40$ $(=50)$

a) Az összeadás a szorzásra nézve nem disztributív. Jelekkel, ha a, b, c tetszőleges operandusok: $a + (b \times c) \neq a+b \times a+c$, *azaz*

ha pl. a=5, b=2, c=8, akkor $5+16 \neq 5 + 10 + 8$

3.1.2. Logikai műveletek

Tekintsük a következő kijelentéseket!

1) 2 páros szám és 2 osztója 16-nak.

2) 5 prímszám (csak az 1 és önmaga az osztója) és 5 osztója 13-nak.

3) 3 osztója 18-nak, mert 3-nak 6-szorosa 18.

4) 3 osztója 18-nak, mert az 5 pozitív szám.

Az első kijelentés a 2 páros szám, valamint a 2 osztója 16-nak kijelentéseknek (mint komponenseknek) az „és” kötőszóval történő összekapcsolása útján keletkezett. A komponensek logikai értéke igaz, és igaznak tartjuk az 1) összetett kijelentés logikai értékét is.

Ha két kijelentést az *és* kötőszóval kapcsolunk össze, akkor az így keletkezett összetett kijelentés logikai értékét – a megszokással összhangban – pontosan akkor tekintjük igaznak, ha mind a két komponens logikai értéke igaz.

Ennek megfelelően a 2) kijelentés logikai értéke hamis, mert – bár azonos szerkezetű az első kijelentéssel – az egyik komponens logikai értéke igaz, a másiké hamis.

Az 1) és 2) kijelentés logikai értékét tehát a komponensek logikai értéke egyértelműen meghatározza.

A 3) és 4) kijelentés két egyszerűbb kijelentésnek a „mert” kötőszóval történő összekapcsolásával keletkezett. Komponenseik logikai értéke igaz és igaznak tartjuk a 3) kijelentést is. Azt azonban mégsem mondhatjuk józanésszel, hogy a 3 azért osztója 18-nak, mert 5 pozitív szám, ahogy azt a 4) kijelentésnél olvastuk.

Elmondhatjuk, hogy a 3) és 4) kijelentés logikai értékét a bennük szereplő egyszerűbb kijelentések logikai értékei nem határozzák meg egyértelműen.

A műveletekre ezt a korlátozást – amelyet értékelési alapelvnek nevezünk – a logika matematikai módszerekkel történő vizsgálata során (a matematikai logikában) vezették be. Az elmondottakból kitűnik, hogy a logikai műveleteket a kijelentések között egy vagy több kijelentésre értelmezhetjük.

- **Logikai műveletről akkor beszélünk, ha adott kijelentésekből úgy épül fel egy összetett kijelentés, hogy annak logikai értékét az adott kijelentések (komponensek) logikai értékei egyértelműen meghatározzák.**

A logikai műveletek röviden a kijelentések között értelmezett matematikai műveletek. A logikai műveletek fogalma alapján az 1) és 2) kijelentés logikai művelettel keletkezett összetett kijelentés, a 3) és 4) nem logikai művelettel keletkezett összetett kijelentés. A továbbiakban a leginkább használt logikai műveleteket értelmezzük, kitérünk azok legegyszerűbb tulajdonságaira is.

3.1.3. A logikai kifejezés. A kijelentések formulája

Az alapfogalmaknál a kijelentést, míg az előzőekben logikai művelet definícióját ismertük meg, de nyilvánvaló, hogy a kijelentéseket a műveletekkel szeretnénk összekapcsolni. Kijelentésekből tehát azok logikai műveleteivel összetett kijelentéseket képezhetünk. Egy összetett kijelentés kijelentéslogikai szerkezetén (pontosítva: „durva" szerkezetén) annak az ábrázolását értjük, hogy az összetett kijelentés milyen más kijelentésekből és milyen műveletek segítségével írható fel.

- **A logikai kifejezés alatt a kijelentések logikai változóinak és a logikai műveletek együttesét értjük.**

A logikai kifejezéseket a matematikai logika jelölésrendszerével leírva ún. formulákat kapunk.

- **A kijelentéslogikai formulák a kijelentésváltozókból (p, q, r,…), műveleti jelekből és zárójelpárokból épülnek fel.**

A logikai műveleti jeleknek is van egy speciális elnevezése.

- **Logikai konstansoknak nevezzük a logikai műveleti jeleket (¬, ∨, ∧,...).**

A kifejezés és a formula között a különbség tehát annyi, hogy a kifejezés az élő nyelv korlátozott, a logika által engedett eszközeit használva, míg a formulák ugyanazt röviden, szimbólumokkal írják le. A két fogalmat gyakran használják egymás szinonimájaként is.

Példa:

Kedvelem a barátaimat és utálom a hideget.

A két kijelentéshez rendeljük a következő változókat:

b:= kedvelem a barátaimat

h:= utálom a hideget

A két kijelentést a hamarosan részletezett „és” művelet kapcsolja össze, a jele: ∧.

A táblázat első sorában látjuk a logikai kifejezést, míg a második sora annak a kijelentéslogikai formuláját tartalmazza:

Kedvelem a barátaimat	és	utálom a hideget.
b	∧	h

3.1.4. A negáció

A köznapi beszédben gyakran fordul elő egy kijelentés tagadása, amit legtöbbször a nem tagadószóval teszünk, pl.: „Nem igaz, hogy nehéz a nyelvvizsga.” A „Nem igaz, hogy” kezdetű mondatok az eredeti kijelentés logikai értékét ellentétesre változtatják. A beszédben tetten érhető jelenséget a logikában negációnak nevezik. Bármely kijelentésből képezhetünk egy újabb kijelentést a következő definíció szerint.

Tetszőleges p kijelentés negációján (tagadásán) a „Nem igaz, hogy p” kijelentést értjük és ⌉p-vel jelöljük. A „nem igaz, hogy p” mellett szokásos használni a „nem p” rövidebb megfogalmazást is.

Példák:

1) A 17 pozitív szám.

A kijelentés negációja:

Nem igaz, hogy a 17 pozitív szám.

A 17 nem pozitív szám.

2) Zsuzsa barátja Tamás.

A kijelentés tagadása:

Nem igaz, hogy Zsuzsa barátja Tamás.

Zsuzsának nem barátja Tamás.

Nem Zsuzsa barátja Tamás.

Tamás nem barátja Zsuzsának.

3) Mindenki szereti a logikát.

A kijelentés negációja:

Nem igaz, hogy mindenki szereti a logikát.

Nem mindenki szereti a logikát.

Van aki nem szereti a logikát.

A példákból kitűnik, hogy egy kijelentés tagadása többféleképpen is megfogalmazható.

- **A p kijelentés tagadásának a logikai értéke:**

$$|\neg p| := \begin{cases} i, ha\ |p| = h \\ h, ha\ |p| = i \end{cases}$$

- **(A leírtakat a következőképpen olvassuk: nem p logikai értéke (legyen) igaz, ha p logikai értéke hamis, illetve (legyen) hamis, ha p logikai értéke igaz.)**
- **A negáció logikai művelet, mert a p kijelentés logikai értéke egyértelműen meghatározza ¬p kijelentés logikai értékét.**
- **A negáció logikai értéktáblázattal ábrázolva:**

|p |	|¬p|
i	h
h	i

Megjegyezzük, hogy a műveletet és annak eredményét is negációnak nevezzük, valamint azt, hogy a negáció nem a kijelentést változtatja ellenkezőjére, hanem annak logikai értékét. Hangsúlyozzuk, hogy a magyar nyelv a mondat tagadását általában az állítmány tagadásával végzi.

Például:

A szobám fala fehér. Ennek a kijelentésnek a tagadása: A szobám fala nem fehér. Lehet, hogy zöld vagy sárga, de egyáltalán nem biztos, hogy pl. fekete.

A p kijelentés tagadásának (⌉p) is képezhetjük a negációját. Ezt a p kijelentést kétszeres tagadásnak nevezzük és ⌉⌉p-vel jelöljük. A p és a ⌉⌉ p kijelentések nyelvileg nem azonosak, de a kijelentések negációja logikai értékének az értelmezése szerint írhatjuk, hogy |p| = |⌉⌉p|. Mind a mindennapi, mind a tudományos életben találkozunk kétszeresen tagadott kijelentésekkel.

Példa:

1) Nem igaz, hogy az órám nem jár pontosan.

2) Nem igaz, hogy a 6 nem osztható 2-vel.

A természetes nyelvben a tagadás használata igen sokféle lehet, van olyan eset is, amikor nem fejezi ki a „nem” tagadószó a negációt. A példában tagadjuk az állítmányt, így valóban kifejezzük a negációt:

Példa:

Péter magasabb, mint Pál.

Péter nem magasabb, mint Pál.

A nyelvben vannak olyan esetek, amelyekről majd csak a prédikátumlogikában tanulunk, de a példákat már itt is érdemes megvizsgálni. A második példában a „nem” szó nem jelent negációt, mivel mindkét állítás igaz:

Példa:

Némelyik emlős tud repülni. (igaz)

Némelyik emlős nem tud repülni. (igaz)

Tagadjuk kétféleképpen is az alábbi mondatot, de a harmadik példa is azt mutatja, hogy egyik sem negáció:

Példa:

Minden ember hazudik. (hamis)

 *Egyetlen ember sem hazudik. (hamis)*

 *Minden ember igazat mond. (hamis, bár átfogalmaztuk)*

Az előző példa negációját a következő kijelentések fejezik ki:

 *Példa:*

 *Nem minden ember hazudik. (igaz)*

 *Nem igaz, hogy minden ember hazudik. (igaz)*

További gondot jelenthet az összetett mondatra alkalmazott negáció. Nem lehetünk biztosak abban, hogy melyik tagmondatra vonatkozik a negáció. Az élő nyelvben ezt jelezhetjük hangsúllyal, de a formuláknál egyértelművé teszi mindezt a zárójel.

 *Példa:*

 *Nem igaz, hogy Kati csinosabb, mint Klára, és Klára csinosabb, mint Luca.*

 *a)] (Kati csinosabb, mint Klára), és Klára csinosabb, mint Luca.*

 *b)] (Kati csinosabb, mint Klára, és Klára csinosabb, mint Luca).*

3.1.5. A konjunkció

A hétköznapi életben gyakran használt kifejezés az „és” szó. Ha például tulajdonságokat sorolunk fel, akkor azok nem tesszük ki minden tulajdonság után az „és” szót: Ági művelt, olvasott, barna hajú és magas. Kijelenthető, hogy mi voltaképpen arra gondoltunk, hogy Ági művelt és Ági olvasott és Ági barna hajú és Ági magas. Azonban a nyelv lerövidíti ezt a logikai okoskodást, viszont ebben az esetben írhatjuk fel egyszerűen a kifejezések ún. konjunkcióját.

- **Konjunkció fogalma. Tetszőleges p és q kijelentések konjunkcióján értjük a „p és q” összetett kijelentést, illetve ennek valamilyen átfogalmazását. Jelölése: p∧q, ahol p, q a konjunkció tagjai.**
- **Két kijelentés konjunkciójának logikai értékét a következőképpen értelmezzük:**

- $|\mathbf{p} \wedge \mathbf{q}| := \begin{cases} \boldsymbol{i}, \boldsymbol{ha}\ |\boldsymbol{p}| = |\boldsymbol{q}| = \boldsymbol{i}, \\ \boldsymbol{h\ egyébként} \end{cases}$
- **A definíció értéktáblázattal:**

|p |	|q|	|p ∧ q|
i	i	i
i	h	h
h	i	h
h	h	h

- **A definíció négyzetes értéktáblázatba foglalva:**

		q	
	∧	i	h
p	i	i	h
	h	h	h

Az értelmezésből látható, hogy két kijelentés konjunkciójának logikai értékét a komponensek logikai értékei egyértelműen meghatározzák, azaz a konjunkció logikai művelet. Konjunkciónak szoktuk nevezni a művelet eredményét is. Tekintsük a következő példákat!

Példa:

1) Az a oldalú négyzet területe a·a és kerülete 4·a.

2) 5 pozitív és páratlan szám.

3) A 6. osztály első lett a tanulmányi- és második a sportversenyen.

Mind a három kijelentés logikai értéke – összhangban a köznapi nyelvvel – csak akkor igaz, ha mind a két komponens logikai értéke igaz. Természetesen a lényeg nem az „és” kötőszón van, hanem az együttes állításon, ezért az „és” helyett sokszor használjuk pl. a bár; ámbár; de; viszont; meg; noha; szintén; … is … is stb. kötőszavakat anélkül, hogy a logikai tartalom megváltozna.

Példa:

Péter is elment, Juli is elment haza. (rövidebben: Péter is, Juli is elment haza.)

Bár Péter elment, Juli itt maradt.

Esik az eső, noha süt a nap.

Füles átázott, mindazonáltal nem lett jókedvű.

Habár az ibolya elhervadt, a többi virágzik.

Felkészültem, viszont izgulok.

Gulyáslevest ettünk meg bort ittunk.

Megjegyezzük, ha egy kijelentésben az „és” kötőszó szerepel, az nem jelenti minden esetben azt, hogy a kijelentés konjunkcióval keletkezett.

Példa:

1) 3-nak és 7-nek a legnagyobb közös osztója 1.

2) 3-nak és 7-nek a legkisebb közös többszöröse 21.

3) A nappal és az éjszaka egymást követik.

4) Kriszta és Imre barátok. (másképpen: Imre barátja Krisztának)

A 4)-es példa a konjunkciót kifejező alakra átfogalmazva már mást jelentene: Kriszta barát, és Imre barát. A barát ilyen esetben szerzetest jelent és ez már valóban konjunkció.

A következő részekben szereplő tételeket „T” betűvel jelöljük, és következetesen számozni fogjuk, hogy később lehessen rájuk hivatkozni.

Az eddigi ismereteink alapján kijelenthető, hogy a modern matematika a tételeket csak akkor fogadja el helyesnek, ha a tételben szereplő állítások igazvolta belátható, amit bizonyításnak neveztünk. A konjunkcióhoz kapcsolódó tételek logikai értéktáblázat segítségével könnyen igazolhatók.

Kijelentés (tétel) bizonyítási lépései. Az összetett kijelentés (tétel, azonosság) igazolása a következőképpen történik:
1) meghatározzuk az egyenlőség jel bal oldalán lévő kifejezés logikai értékét minden logikai változó (p, q, r…) összes lehetséges logikai értékére.
2) meghatározzuk az egyenlőség jel másik, jobb oldalán lévő kifejezés logikai értékét minden logikai változó (p, q, r…) összes lehetsé-

ges logikai értékére.

3) Ha a bal és a jobb oldali kifejezés logikai értékei minden esetben (az igazságtáblázat minden sorában) megegyeznek, akkor a kijelentést (tételt, azonosságot) beláttuk, azaz bebizonyítottuk.

4) Ha csak egyetlen esetben is a bal oldali kifejezés logikai értéke eltérő a jobb oldali kifejezés logikai értékétől (az igazságtáblázat egy vagy több sorában), akkor az összetett kijelentés nem bizonyítható, tehát nem lehet sem tétel sem azonosság.

Összefoglalva a bizonyítások lényegét: mivel egy változónak mindössze két értéke (igaz vagy hamis) létezik, ezért a logikai értéktáblázat felhasználásával szemléletesen, az összes lehetőség leírásával láthatók be a tételek.

Adott p, q és r kijelentésekből képezhetjük a p∧q; q∧p; (p∧q)∧r; p∧(q∧r); p∧p; p∧⌉p kijelentéseket. Ezek logikai értékére teljesülnek a következő tulajdonságok:

- **T1: A konjunkció kommutatív: | p∧q | = | q∧p |**

A kommutativitás igazolása:

|p |	| q|	|p ∧ q|	|q ∧ p|
i	i	i	i
i	h	h	h
h	i	h	h
h	h	h	h

Mivel a harmadik és a negyedik oszlop minden sorában megegyeznek a logikai értékek, kijelenthető, hogy p, q változók minden lehetséges logikai értéke esetén az egyenlőség jel két oldala megegyezik, tehát a tételt beláttuk.

- **T2: A konjunkció asszociatív: | (p∧q)∧r | = | p∧(q∧r) |**

Az asszociativitás igazolása:

|p |	| q|	| r|	|(p ∧ q)	∧ r|	|p ∧	(q ∧ r)|
i	i	i	i	i	i	i
i	i	h	i	h	h	h
i	h	i	h	h	h	h
i	h	h	h	h	h	h
h	i	i	h	h	h	i
h	i	h	h	h	h	h
h	h	i	h	h	h	h
h	h	h	h	h	h	h
			1	2	2	1

Megjegyzés: a logikai műveletek sorrendjét a zárójellel befolyásolhatjuk, amit a táblázat utolsó sorában a számozással jelöltünk: 1 szerepel az elsőként végrehajtandó művelet alatt, 2 pedig a másodikként végrehajtandó művelet alatt. Ezt a segítséget a következőkben is így érvényesítjük.

A vastag kerettel jelölt, a két oldal formuláját tartalmazó logikai értékek minden sorban azonosak, tehát a tételt bebizonyítottuk.

- **T3: A konjunkció idempotens: |p ∧ p| = |p|**

Az idempotencia igazolása:

|p |	|p ∧ p|
i	i
h	h

A tételt bebizonyítottuk.

- **T4: A konjunkcióra érvényes az ellentmondástalanság elvének tükrözése: |p ∧ ⌉p| = h.**
- **Olvasva: p konjunkciója a önmaga negáltjával mindig hamis.**

Az ellentmondástalanság elvének igazolása:

|p |	|p ∧	⌉ p|	h
i	h	h	h
h	h	i	h
	2	1	

A tételt bebizonyítottuk.

3.1.6. A sem-sem művelet

A negáció és a konjunkció felhasználásával értelmezhetünk további logikai műveleteket is. Ezek közül egyik az úgynevezett sem-sem művelet, amelyet az alábbiak szerint értelmezünk és a || jellel jelölünk.

- **A sem-sem logikai művelet értelmezése:**
  p || q := ⌉p∧⌉q
  A sem-sem művelet eredményének logikai értéke csak akkor igaz, ha |p| = |q| = h.
  A sem-sem művelet logikai értéktáblázata:

|p |	|q|	|]p |	|]q|	|p || q|
i	i	h	h	h
i	h	h	i	h
h	i	i	h	h
h	h	i	i	i

A definíció négyzetes értéktáblázatba foglalva:

		q	
	∧	i	h
p	i	h	h
	h	h	i

Idézzük fel a művelethez tartozó talán leghíresebb magyar mondat sorait Ady Endrétől:

Példa:

„Sem utódja, sem boldog őse,

Sem rokona, sem ismerőse

Nem vagyok senkinek,”

3.1.7. A diszjunkció

Az emberek életében igen gyakori az útelágazás. Dönteni kell, és az esetek többségében ez nem egyszerű. A döntések leírása a logikában is összetett, hiszen a beszédben sem lehet egyértelműségről beszélni.

Példa:

Vagy a fagylaltom nélkülözöm, vagy a tortát!

A mondatot többféleképpen lehet érteni, attól függően miként hangsúlyozzuk. Ha komolyan mondjuk egy diéta közepén, akkor egyszerre nem ehetjük meg a fagylaltot és a tortát, csak az egyiket, de az is lehet, hogy egyiket sem. A diszjunkció művelete is ezt jelenti. Azonban mondhatjuk szigorúan (Vagy a fagylaltom nélkülözöm, vagy tortát! Nincs tovább vita!), ami jelentheti azt is, hogy kizárólag az egyik eset történhet meg, de az mindenképpen. Az első jelentéshez kapcsolódva definiáljuk az új logikai műveletet.

Diszjunkció fogalma. Jelöljön p, q két tetszőleges kijelentést. A p és q kijelentésekkel végzett sem-sem művelet negációjaként értel-

mezünk egy újabb műveletet, amelyet a p, q kijelentések diszjunkciójának nevezünk. Jelölése: p∨q, ahol p, q a diszjunkció tagjai.
p∨q :=]([p∧]q)
Olvasása: p diszjunkciója q-val legyen egyenlő nem igaz, hogy sem p, sem q.
A sem-sem művelet logikai értékeinek figyelembe vételével a diszjunkció logikai értéktáblázatát az alábbiak szerint felírhatjuk:

A definíció értéktáblázattal:

|p |	| q|	|p ∨ q|	|]p ∧]q|	|]([p ∧]q)|
i	i	i	h	i
i	h	i	h	i
h	i	i	h	i
h	h	h	i	h

A definíció négyzetes értéktáblázatba foglalva:

		q	
	∨	i	h
p	i	i	i
	h	i	h

A diszjunkció esetében is igaz, hogy a műveletet és annak eredményét is diszjunkciónak nevezzük. Mivel a diszjunkciót negáció és konjunkció segítségével értelmeztük, ezért logikai művelet. Ez a logikaiérték-táblázatból is látható.

A táblázatból az is kiolvasható, hogy a diszjunkció logikai értéke csak abban az egy esetben hamis, ha mind a két komponens logikai értéke hamis.

Többféle „vagy" létezik

A vagy kötőszót szokás megengedő, kizáró, illetve összeférhetetlenséget kifejező értelemben használni. Két kijelentés megengedő értelmű „vagy" kötőszóval történő összekapcsolása által keletkezett kijelentés akkor igaz, ha a komponensek közül legalább az egyik logikai értéke igaz, vagyis ez a diszjunkció művelete.

A „vagy" kötőszót megengedő (alternáció) értelemben (OR):

Példa:

a) Ákos matematikából vagy fizikából jelesre felelt. Elképzelhető, hogy csak matematikából, csak fizikából kapott jelest, de lehet hogy mind a két tárgyból jelesre felelt.

b) Paradicsom salátát kérek a csirkéhez vagy csalamádét. Lehet, hogy az illető paradicsomsalátát eszik, elképzelhető, hogy csak csalamádét, de akár mindkettőt is ehet felváltva.

Két kijelentés kizáró értelmű „vagy” kötőszóval történő összekapcsolása által keletkezett kijelentés akkor igaz, ha a komponensek közül csak az egyik, de csakis az egyik logikai értéke igaz, vagyis ez a művelete. A „vagy” kötőszót használhatjuk kizáró (antivalencia) értelemben (XOR) is.

Példa:

a) Figyel az órán, vagy kimegy. Itt a vagy kötőszó kizáró értelmű, mivel figyelni illetve elmenni egyidejűleg nem lehet, harmadik lehetőség nincs.

b) Vagy hajóval megyünk, vagy repülővel. Nyilvánvaló, csak az egyikkel utazhatunk.

A „vagy” kötőszó összeférhetetlenséget is kifejezhet.

Az összeférhetetlenséget kifejező „vagy” kötőszót alkalmazó műveletet NAND- vagy Sheffer-műveletnek nevezzük:

Példa:

a) Újságot olvasok, vagy alszok. Újságot olvasni és aludni is egyidejűleg nem lehet, de egy harmadik lehetőség nem kizárt.

b) ön dönt: iszik, vagy vezet. Nem szükséges örökké csak a vezetés meg csak az ivás állapotában lenni, lehetséges, hogy se nem iszunk, se nem vezetünk.

c) Teca, vagy csíkos blúzt veszel fel, vagy pöttyös szoknyát. Ebben a helyzetben egyáltalán nem szükséges, hogy a kettő közül valamelyik eset igaz legyen, mást is felvehet, hiszen a beszélő nyilván nem azt szándékozott kifejezni, hogy Tecának örökké csak a csíkos blúzában, vagy örökké csak a pöttyös szoknyájában kellene járnia. Csupán azt szeretné, hogy a kettőben egyszerre ne jelenjen meg.

Megjegyezzük, hogy a vagy kötőszó nem megengedő értelmű használatáról a „vagy” előtti vessző, más szerkezetben pedig a „vagy ..., vagy ...” informál bennünket.

Diszjunkció és a „vagy” kötőszó

Tekintsük az alábbi kijelentéseket:

a) Nem igaz, hogy (17 se nem pozitív, se nem páratlan szám).

b) 3 osztója 6-nak vagy Zsuzsa jeles tanuló.

c) Nem igaz, hogy (3 nem osztója 6-nak és Zsuzsa nem jeles tanuló).

Az a) kijelentés diszjunkció, amely azt állítja, hogy 17 pozitív szám vagy a 17 páratlan szám. A vagy kötőszót megengedő értelemben használjuk. Úgy tűnik, hogy két kijelentés diszjunkciója felírható a komponenseknek vagy kötőszóval történő összekapcsolásával is. Ehhez azonban egy megjegyzést kell tenni.

A b) kijelentés a „3 osztója 6-nak”, illetve a „Zsuzsa jeles tanuló” komponenseknek „vagy” kötőszóval történő összekapcsolásával keletkezett. Ezt az összetett kijelentést nyelvileg furcsának tartjuk és nem is érezzük igaznak. A c) kijelentés a b) kijelentésben szereplő komponensek diszjunkciója, s ezt a kijelentést nyelvileg is helyénvalónak, s emellett igaznak is érezzük.

Elmondhatjuk, hogy a megengedő „vagy” kötőszóval összekapcsolt két kijelentés mindig felírható diszjunkcióként, de fordítva nem, vagyis két kijelentés diszjunkciója nem mindig fejezhető ki értelmes módon „vagy” kötőszó segítségével. Az előbbi felismerésre nézzünk egy példát!

Példa:

A c) diszjunkciót nem írhatjuk fel a b) formában.

A diszjunkció tulajdonságai

Adott p, q, r kijelentésekből képezhetjük például a pVq, qVp, (pVq)Vr, pV(qVr), pVp kijelentéseket. Ezek logikai értékeit összehasonlítva megállapítható, hogy a diszjunkció, kommutatív, asszociatív, idempotens tulajdonságú logikai művelet. A diszjunkcióhoz kapcsolódó tételek logikai értéktáblázat segítségével könnyen igazolhatók.

T5. A diszjunkció kommutatív: |pVq | = | qVp |

A kommutativitás igazolása:

|p |	| q|	|pVq|	|qVp|
i	i	i	i
i	h	i	i
h	i	i	i
h	h	h	h

Mivel a harmadik és a negyedik oszlop minden sorában megegyeznek a logikai értékek, ezért kijelenthető, hogy p, q változók minden lehetséges logikai értéke esetén az egyenlőség jel két oldala megegyezik, tehát a tételt beláttuk.

T6. A diszjunkció asszociatív: | (pVq)Vr | = | pV(qVr) |

Az asszociativitás igazolása:

|p |	| q|	| r|	|(p V q)	V r|	|p V	(q V r)|
i	i	i	i	i	i	i
i	i	h	i	i	i	i
i	h	i	i	i	i	i
i	h	h	i	i	i	h
h	i	i	i	i	i	i
h	i	h	i	i	i	i
h	h	i	h	i	i	i
h	h	h	h	h	h	h
			1	2	2	1

Megjegyzés: a logikai műveletek sorrendjét a zárójellel befolyásolhatjuk, amit a táblázat utolsó sorában a számozással jelöltünk: 1 szerepel az elsőként végrehajtandó művelet alatt, 2 pedig a másodikként végrehajtandó művelet alatt. Ezt a segítséget a következőkben is így érvényesítjük.

A vastag kerettel jelölt, a két oldal formuláját tartalmazó logikai értékek minden sorban azonosak, tehát a tételt bebizonyítottuk.

T7. A diszjunkció idempotens: | pVp | = | p |

Az idempotencia igazolása:

|p |	|p V p|
i	i
h	h

A tételt bebizonyítottuk.

- **T8. A diszjunkcióra érvényes a harmadik kizárása elvének tükrözése: | p∨]p | = i**
  Olvasva: p diszjunkciója önmaga negáltjával azonosan (mindig) igaz.

A harmadik kizárása elvének igazolása:

|p |	|p ∨] p|		i
i	i	h	i
h	i	i	i
	2	1	

A tételt bebizonyítottuk.

3.1.8. A de Morgan-azonosságok

Konjunkcióval és diszjunkcióval mint logikai műveletekkel és azok tulajdonságaival külön-külön megismerkedtünk. Most olyan összefüggéseket tekintünk át, amelyekben egy logikai kifejezésben vagy egy összetett kijelentésben a konjunkció és a diszjunkció művelete egyszerre szerepel. Elsőként vizsgáljuk meg azt az esetet, amikor a két művelet tagadását próbáljuk meg a másik művelet segítségével leírni. Az összefüggések egy-egy azonosságot takarnak, azaz a kijelentések logikai értékétől függetlenül állandóan igazak.

- **De Morgan-azonosságok:**
  Legyen p, q két tetszőleges kijelentés.
  T9. |](p∧q) | = |]p ∨]q |
  Ezt így olvassuk: p konjunkció q negációja egyenlő a p, q negációjának diszjunkciójával.

Igazoljuk az első de Morgan-azonosságot (T9)!

|p |	| q|	|] (p∧q)|		|]p ∨]q|
i	i	h	i	h
i	h	i	h	i
h	i	i	h	i
h	h	i	h	i
		2	1	

Az első azonosságot beláttuk.

- **T10. |]($p \vee q$) | = |]$p \wedge$]q |**
- **Ezt így olvassuk: p diszjunkció q negációja egyenlő a p, q negációjának konjunkciójával.**

Igazoljuk a második de Morgan-azonosságot (T10)!

|p |	| q|	|] (p$\vee$q)|		|]p $\wedge$]q|
i	i	h	i	h
i	h	h	i	h
h	i	h	i	h
h	h	i	h	i
		2	1	

A 2. azonosságot beláttuk.

3.1.9. Az abszorpció és a disztributivitás tétele

- **Abszorció. Legyen p, q két tetszőleges kijelentés.**
  T11. |($p \wedge (p \vee q)$) | = | p |
  T12. |($p \vee (p \wedge q)$) | = | p |
  A konjunkció abszorbtív (elnyelő) tulajdonságú a diszjunkcióra nézve, és fordítva: a diszjunkció abszortív tulajdonságú a konjunkcióra nézve.

A tétel bizonyítását önállóan próbálja megoldani. Ha nem sikerül, akkor megtalálja az összefoglaló feladatok között.

- **Disztributivitás. Legyen p, q, r két tetszőleges kijelentés.**
  T13. |$p \wedge (q \vee r)$ | = | ($p \wedge q$) $\vee$ ($p \wedge r$) |
  T14. |$p \vee (q \wedge r)$ | = | ($p \vee q$) $\wedge$ ($p \vee r$) |
  A konjunkció disztributív tulajdonságú a diszjunkcióra nézve, és fordítva: a diszjunkció disztributív tulajdonságú a konjunkcióra nézve.

A tétel bizonyítását önállóan próbálja megoldani. Ha nem sikerül, akkor megtalálja az összefoglaló feladatok között.

3.1.10. Az implikáció

A hétköznapi életben gyakran használjuk a „ha p, akkor q” alakú összetett kijelentéseket, és a logikai értékét is csak akkor tekintjük hamisnak, ha |p|= i és |q|=h, bár nagyon ritkán találkozunk olyan „ha p, akkor q” kijelentésekkel, ahol |p|= h. Ezek alapján egy újabb logikai műveletet értelmezünk.

 *Példa:*

 *Ha lesz pénzem, akkor a tengernél nyaralok.*

Tekintsük az alábbi értéktáblázatot!

|p|	| q|	|¬(p ∧ ¬q)|
i	i	i
i	h	h
h	i	i
h	h	i

Látható, hogy ¬(p ∧ ¬q) logikai értéke csak abban az egy esetben lesz hamis, ha |p|= i és |q|=h, más esetben igaz.

Kérdés, milyen kötőszavakat használjunk az olyan összetett kijelentések nyelvi formájában, amelyben az összetett kijelentés logikai értéke a táblázat szerint adódik. Egy ilyen lehetséges összetétel (a ¬és ∧ nyelvi megfelelője alapján) a „nem igaz, hogy (p és nem q)”. Mind a negáció, mind a konjunkció értelmezhető tetszőleges kijelentések között, így a fenti típusú kijelentés is előállítható bármely két kijelentésből, és a komponensek logikai értékei egyértelműen határozzák meg az összetett kijelentés logikai értékét.

A hétköznapi életben hasonló értelemben használjuk a „ha p, akkor q” alakú összetett kijelentéseket, mivel ezek logikai értékét is csak akkor tekintjük hamisnak, ha |p|= i és |q|=h, bár nagyon ritkán találkozunk olyan „ha p, akkor q” kijelentésekkel, ahol |p|= h.

Vizsgáljuk meg, hogy tetszőleges p, q kijelentésekből összetett, „ha p, akkor q” mondat tekinthető-e logikai művelettel összetett kijelentésnek!

 *Példa:*

 *a) Ha tanulok, akkor jó jegyet kapok az iskolában.*

 *b) Ha otthon vagyok, akkor dolgozom.*

c) Ha találkozom a barátommal, akkor visszaadom a könyvét.

d) Ha a 3 prímszám, akkor Kiss Péter az iskola legjobb tanulója.

e) Ha süt a Nap, akkor 2·2= 4

f) Ha az r sugarú kör kerülete 2rπ, akkor a kutyám vadon élő állat.

A d), e), f), példákból láthatjuk, hogy tetszőleges p,q kijelentésekből nem biztos, hogy értelmes – ha, p akkor q – szerkezettel kifejezhető mondatot kapunk.

Példa:

Fogalmazzuk meg a kijelentéseket – Nem igaz, hogy (p és nem q) formában!

a) Ha tanulok, akkor jó jegyet kapok.

Nem igaz, hogy (tanulok és mégsem kapok jó jegyet).

b) Ha otthon vagyok, akkor dolgozom.

Nem igaz, hogy (otthon vagyok és nem akkor dolgozom).

c) Ha találkozom a barátommal, akkor visszaadom a könyvét.

Nem igaz, hogy (találkozom a barátommal és nem adom vissza a könyvét).

d) Ha a 3 prímszám, akkor Kiss Péter a legjobb tanuló.

Nem igaz, hogy (3 prímszám és Kiss Péter nem a legjobb tanuló).

e) Ha süt a Nap, akkor 2×2= 4

Nem igaz, hogy (süt a Nap és 2×2= 4).

f) Ha az r sugarú kör kerülete 2rπ, akkor a kutyám vadon élő állat.

Nem igaz, hogy (egy r sugarú kör kerülete 2rπ és a kutyám nem vadon élő állat.

A d), e), f), példákból láthatjuk, hogy tetszőleges p,q kijelentésekből nem biztos, hogy értelmes – ha, p akkor q – szerkezettel kifejezhető mondatot kapunk.

Az a), b), c), mondatok most is értelmesek maradtak, míg a d), e), f), ebben a megfogalmazásban már nem bántja nyelvérzékünket, s logikai értéküket is meghatározhatjuk.

Általában elmondható, hogy bármely „ha p, akkor q" alakú összetett kijelentés átfogalmazható nem igaz, hogy (p és nem q) alakra, de nincs minden nem igaz, hogy (p és nem q) összetételének ha p, akkor q alakú – értelmes – megfogalmazása.

Megjegyezzük, hogy a ha p, akkor q alakú mondat rendszerint csak akkor értelmes, ha a komponensek között tartalmi és oksági kapcsolat is van.

Az eddigiek alapján egy új logikai műveletet értelmezhetünk:

- **Legyen p, q tetszőleges kijelentés. A nem igaz, hogy (p és nem q) összetett kijelentést a p és q kijelentések implikációjának nevezzük, és p→q-val jelöljük, azaz:**
  p→q := ⌉(p ∧ ⌉q).
  A p-t előtagnak, a q-t utótagnak nevezzük.
  Az implikáció logikai értékét a definíció alapján értelmezzük:

- $|p \to q| := \begin{cases} h, ha\ |p| = i\ és\ |q| = h \\ i\ egyébként \end{cases}$

- **A definíció értéktáblázattal:**

|p |	|q|	|p→q|
i	i	i
i	h	h
h	i	i
h	h	i

- **A definíció négyzetes értéktáblázatba foglalva:**

		q	
	→	i	h
p	i	i	h
	h	i	i

Két kijelentés implikációjának logikai értékét tehát csak akkor tekintjük hamisnak, ha az előtag igaz, az utótag hamis logikai értékű.

Az implikáció tetszőleges p, q kijelentésre teljesülő, egyszerű tulajdonságait foglaljuk össze a következő részben.

Tétel.

T15. Az implikáció nem kommutatív, azaz $|p \rightarrow q| \neq |q \rightarrow p|$.

T16. Az implikáció nem asszociatív, azaz $|(p \rightarrow q) \rightarrow r| \neq |p \rightarrow (q \rightarrow r)|$.

T17. Az implikáció nem idempotens, azaz $|p \rightarrow p| \neq |p|$.

Ezek a tulajdonságok könnyen igazolhatók logikai értéktáblázattal. Megjegyezzük, hogy a p → q kijelentés megfordításának nevezzük a q → p kijelentést. Az alábbiakban először két implikációt és azok megfordítását fogalmaztuk meg.

Példa:

Ha Ákosnak jó a bizonyítványa, akkor megkapja a kerékpárt.

Ha Ákos megkapja a kerékpárt, akkor jó a bizonyítványa.

Ha egy négyszög húrnégyszög, akkor a szemben fekvő szögeinek az összege 1800.

Ha egy négyszög szemben fekvő szögeinek az összege 1800, akkor a négyszög húrnégyszög.

Tétel.

T18. Az implikációra igaz a kontrapozíciós tulajdonság, azaz:

$|p \rightarrow q| = |\neg q \rightarrow \neg p|$.

Bizonyítás:

A következő értéktáblázat bizonyítja a tételt:

$\|p\|$	$\|q\|$	$\|(p \rightarrow q)\|$	$\|\neg q$	$\rightarrow$	$\neg p\|$
i	i	i	h	i	h
i	h	h	i	h	h
h	i	i	h	i	i
h	h	i	i	i	i
			1	2	1

A kontrapozíciós tulajdonság nyelvi vetületeit szemlélteti a következő példa:

Példa:

a) Ha játék közben elestem, akkor eltört a kezem.

b) Ha nem tört el a kezem, akkor nem estem el játék közben.

Az a) és b) mondatok nyelvileg különbözőek, de logikai értékük egyenlő. A matematikában a „ha p, akkor q" alakú kijelentéseket – általában tételek kimondásakor – szokás még az alábbi módon is megfogalmazni:

- q akkor, ha p;
- p elégséges feltétele q-nak;
- q teljesüléshez elegendő p teljesülése;
- q szükséges feltétele p-nek;
- p teljesüléséhez szükséges q teljesülése.

Például:

a) Ha a egész szám osztható 4-gyel, akkor 2-vel is osztható.

- a osztható 2-vel akkor, ha a osztható néggyel.

- 2-vel való oszthatóság elégséges feltétele a 4-gyel való oszthatóság.

- 2-vel való oszthatóság szükséges feltétele a 4-gyel való oszthatóságnak.

b) Ha egy háromszög egyenlő oldalú, akkor a háromszög hegyesszögű.

- Egy háromszög hegyesszögű, ha egyenlő oldalú.

- Egy háromszög oldalainak egyenlősége elégséges feltétel ahhoz, hogy a háromszög hegyesszögű legyen.

- Ahhoz, hogy egy háromszög egyenlő oldalú legyen, szükséges, hogy mindhárom szöge hegyesszög legyen.

Egy „ha p, akkor q" alakban megfogalmazott tétel bizonyításánál a p állításból indulunk ki, s ebből igazoljuk a q állítás helyességét. A kontrapozíciós

tulajdonság alapján a bizonyítás elvégezhető úgy is, hogy a ⌉q állításból indulunk ki és ⌉p állítást igazoljuk.

Példa:

Például ha az előző tételeket akarjuk bizonyítani, akkor egyenértékű, ha a következőket bizonyítjuk:

- Ha az a szám nem osztható 2-vel, akkor nem osztható 4-gyel sem.

- Ha egy háromszög nem hegyesszögű, akkor nem is egyenlő oldalú.

3.1.11. Az ekvivalencia

Legyen p és q két tetszőleges kijelentés. Képezzük a p → q ; q → p kijelentéseket. Mivel az implikáció nem kommutatív tulajdonságú művelet, ez a két implikáció általában különböző logikai értékű.

A p → q, q → p implikációk konjunkcióval egy új logikai műveletet értelmezünk.

Legyen p, q tetszőleges kijelentés. A p, q kijelentések ekvivalenciáján értjük és p□q-val jelöljük a következőt:
p□q := (p→q) ∧ (q→p).
A p□q kifejezés olvasása: p ekvivalens q-val.
Az ekvivalencia logikai értéktáblázata a definíció alapján így adható meg:

|p |	| q|	|(p→q)|	|(q→p)|	|(p□q)|
i	i	i	i	i
i	h	h	i	h
h	i	i	h	h
h	h	i	i	i

A definíció négyzetes értéktáblázatba foglalva:

		q	
	□	i	h
p	i	i	h
	h	h	i

A feltüntetett értéktáblázat szerint p ↔ q logikai értéke akkor igaz, ha p-nek és q-nak azonos a logikai értéke.

Az ekvivalencia logikai műveletre tetszőleges p, q kijelentés esetén teljesülnek a következő tulajdonságok:

Tétel.

T19. Az ekvivalencia kommutatív, azaz $|p \square q| = |q \square p|$

T20. Az ekvivalencia asszociatív, azaz

$|(p \square q) \square r| = |p \square (q \square r)|$.

A tételek logikai értéktáblázattal igazolhatók, a feladatok között ezt a tételt is megtalálja a megoldásával együtt.

Az ekvivalenciának megfelelő „ha p, akkor q” és „ha q, akkor p” kijelentéseket a matematikában „q akkor és csak akkor, ha p”, illetve „p szükséges és elégséges feltétele q” formában szokás megfogalmazni.

Példa:

a) Egy háromszög akkor és csak akkor egyenlő szárú, ha két szöge egyenlő.

b) Egy a egész szám 10-zel való oszthatóságának szükséges és elégséges feltétele, hogy a szám tízes számrendszerben felírt alakja 0-ra végződjön.

Az ekvivalencia segítségével megfogalmazott tételeket a matematikában kritériumoknak is nevezzük.

3.1.12- Feladatok

A következőkben néhány feladatot jelölünk ki a Logika műveletei alfejezeteihez.

Negáció

1) Képezzük a következő kijelentés (a) egy- és (b) kétszeres tagadását!

Esik az eső.

2) Képezzük a következő kijelentés (a) egy- és (b) kétszeres tagadását!

1 kisebb, mint 2.

3) Képezzük a következő kijelentés (a) egy- és (b) kétszeres tagadását!

Ez a paralelogramma négyzet.

4) Képezzük a következő kijelentés (a) egy- és (b) kétszeres tagadását!

3+2=5.

5) Képezzük a következő kijelentés (a) egy- és (b) kétszeres tagadását!

Ákos jó tanuló.

6) Képezzük az alábbi kijelentés (a) egy- és (b) kétszeres tagadását! Ezeket fogalmazzuk meg többféleképpen is!

Az iskola minden tanulója szereti a matematikát.

7) Képezzük az alábbi kijelentés (a) egy- és (b) kétszeres tagadását! Ezeket fogalmazzuk meg többféleképpen is!

11 kisebb, mint 7.

8) Képezzük az alábbi kijelentés (a) egy- és (b) kétszeres tagadását! Ezeket fogalmazzuk meg többféleképpen is!

A 4 pozitív szám.

Konjunkció

11) Képezzük a kijelentések konjunkcióját és határozzuk meg a logikai értéküket! A megoldásnál a kijelentések logikai értékét matematikai ismereteink alapján állapítsuk meg!

a) p := 7 osztója 14-nek

q := 100 nagyobb, mint 3

b) p := A négyzet síkidom.

q := A négyzet nem paralelogramma.

Diszjunkció

12) Igazoljuk, hogy a diszjunkció kommutatív, azaz

Bármely p,q,r kijelentésre: $|p \vee q| = |q \vee p|$!

13) Igazoljuk, hogy a diszjunkció asszociatív, azaz

Bármely p,q,r kijelentésre: $|(p \vee q) \vee r| = |p \vee (q \vee r)|$.

De Morgan-azonosság

14) Igazolja a de Morgan-azonosságokat a diszjunkcióra és a konjunkcióra nézve!

Implikáció

15) Igazolja, hogy bármely p kijelentésre : $|p \rightarrow p| \neq |p|$ (Az implikáció nem idempotens.)

Ekvivalencia

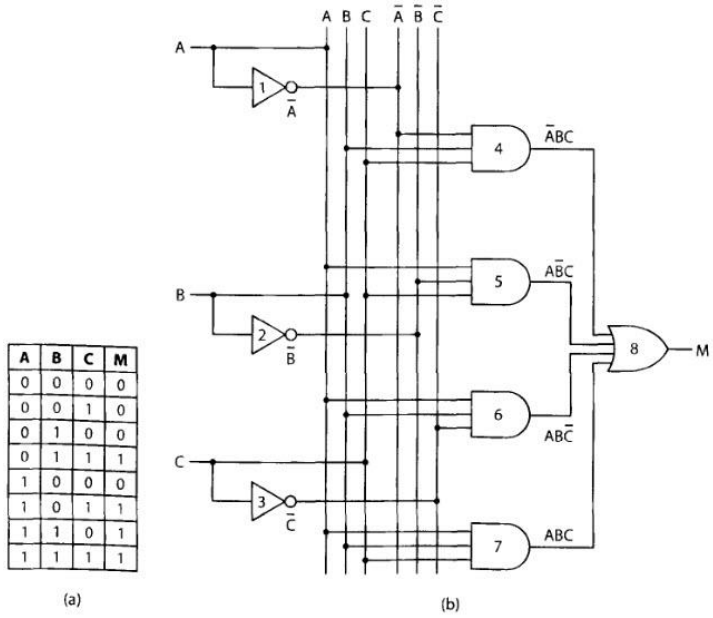
16) (a) Fogalmazzuk meg a következő kijelentéspár ekvivalenciáját többféleképpen is! (b) Határozzuk meg a logikai értékét!

p:= Az n egész szám 0-ra végződik.

q:= Az n egész szám páros.

3.1.13. A Boole-függvények megvalósítása

Ahogy az előbb említettük, a Boole-függvény szorzatok legfeljebb 2^n tagú összegével történő leírása közvetlenül elvezet egy lehetséges megvalósításhoz. Az ábrát használva példaként, láthatjuk, hogyan hajtható végre a megvalósítás. Az ábrán az A, B és C bemeneteket a bal szélen, a kimeneti függvényt, M-et, a jobb szélen ábrázoltuk. Mivel szükségünk van a bemeneti változók invertált értékére (komplemensére), a bemenetek elágaztatásával, az 1-gyel, 2-vel és 3-mal jelzett inverteren áteresztve előállítjuk ezeket. Annak érdekében, hogy ne legyen zavaros az ábra, berajzoltunk hat függőleges vonalat, amelyből három össze van kötve a bemeneti változókkal, a másik három ezek komplemenseivel.



A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)

6. ábra: A háromváltozós többségi függvény igazságtáblázata b. Áramkör (a) megvalósítására

Ezek a vonalak szolgáltatják a bemeneteket a kapuk számára. Például az 5, 6, 7 kapuk mindegyike használja A-t bemenetként. Egy tényleges áramkörben ezek a kapuk valószínűleg közvetlenül hozzá vannak kötve A-hoz anélkül, hogy közbülső „függőleges” vezetéket használnának.

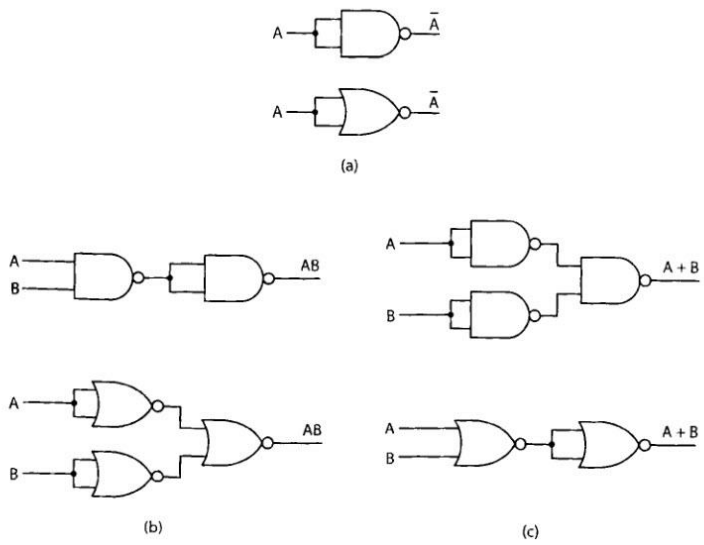
Az áramkör négy És-kaput tartalmaz, az M függvény minden tagjához egyet (az igazságtáblázat minden olyan sorához, ahol 1-es bit van az eredményoszlopban). Minden És-kapu kiszámolja az igazságtáblázatnak egy sorát, ahogy jeleztük. Végül a szorzatok eredményei VAGY-olásra kerülnek, és ez adja a végeredményt.

A ábrán látható áramkör egy olyan megállapodást használ, amelyre többször szükségünk lesz az egész könyvben: amikor két vonal metszi egymást, nincs kapcsolat közöttük, hacsak egy nagy pont nem jelzi ezt a kereszteződésben. Például a 3-as kapu kimenete keresztezi mind a hat függőleges vonalat, de csak a C-hez csatlakozik.

A ábrán megadott példából világosnak kell lennie, hogy miként valósítható meg áramkörrel bármilyen Boole-függvény:

Bár megmutattuk, hogyan lehet bármilyen Boole-függvényt megvalósítani NEM, ÉS, illetve VAGY kapukkal, gyakran kényelmesebb az áramköröket csak egyfajta kapuval megvalósítani. Szerencsére egyszerű módja van annak, hogy az

előző algoritmus által létrehozott áramkört átalakítsuk tisztán NEM-ÉS vagy tisztán NEM-VAGY formájúvá. Ahhoz hogy megtegyünk egy ilyen átalakítást, csak arra van szükség, hogy a NEM, ÉS és VAGY kaput egyetlen kaputípus felhasználásával valósítsuk meg. Az ábra felső sora mutatja, hogyan valósíthatjuk meg mindhárom kaput csak NEM-ÉS kapukkal; az alsó sor mutatja, hogyan tehetjük meg csak NEM-VAGY kapukat használva. (Ez így egyszerű, de más mód is lehetséges.)



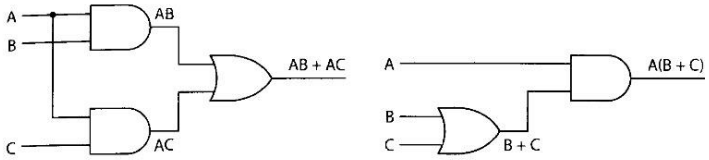
7. ábra: (a) nem (b) és (c)vagy kapuk cask nem – és vagy csak nem- vagy kapukkal történő megvalósítása

A Boole-függvények megvalósításának egyik módja, ha csak NEM-ÉS, vagy csak NEM-VAGY kaput akarunk használni, hogy először a fenti megvalósítási lépéseket végezzük el NEM, ÉS és VAGY kapukkal. Ezután cseréljük ki a többszörös bemenettel rendelkező kapukat két bemenetű kapukból álló ekvivalens áramkörökkel. Például: A + B + C + D-t kiszámíthatjuk, mint (A + B) + (C + D) úgy, hogy három darab kétbemenetes VAGY-kaput használunk. Végül a NEM-, ÉS- és VAGY-kapukat kicseréljük a 3.4. ábra áramköreivel.

Bár ez az eljárás nem vezet optimális áramkörhöz abban az értelemben, hogy minimális számú kaput használjon, de az kimutatható, hogy ez a megoldás mindig megvalósítható. A NEM-ÉS és a NEM-VAGY kapukról azt mondjuk, hogy teljesek (complete), mert bármely Boole-függvény kiszámítható ezek bármelyikének kizárólagos felhasználásával. Más kapunak nincs meg ez a tulajdonsága, ami miatt gyakran előnyben részesítik ezeket az áramköri blokkok tervezésénél.

3.1.14. Áramköri ekvivalencia

A hálózattervezők gyakran próbálják csökkenteni termékeikben a kapuk számát, hogy csökkentsék az alkatrészek árát, a nyomtatott áramköri lap nagyságát, az áramfogyasztást és így tovább. Az áramkör bonyolultságának csökkentéséhez a tervezőknek találnia kell egy olyan áramkört, amely ugyanazt a függvényt számolja ki, mint az eredeti, de kevesebb kapuból áll (vagy egyszerűbb kapukból, például kétbemenetes kapukból a négybemenetesek helyett). Az ekvivalens áramkörök keresésében a Boole-algebra nagyon értékes eszköz.



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)

A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

(b)

8. ábra: Két ekvivalens függvény (a) AB+ AC (b) A(B+C)

A Boole-algebra felhasználására példaként tekintsük az (a) ábrán bemutatott AB + AC áramkört és igazságtáblázatot. Bár még nem tárgyaltuk, de a hagyományos algebra nagyon sok szabályát megtartja a Boole-algebra is. Speciálisan, az AB + AC a disztribúciós szabály használatával felírható A(B + C) alakban is. A (b) ábra mutatja az A(B + C) áramkörét és igazságtáblázatát. Mivel két függvény akkor és csak akkor ekvivalens, ha az összes lehetséges bemenetre a két függvény ugyanazt a kimenetet adja; nagyon egyszerűen ellenőrizhetjük az ábra igazságtáblázataiból, hogy A(B + C) ekvivalens AB +AC-vel. Az ekvivalencia ellenére világosan látható, hogy a (b) áramkör jobb, mint a (a), mert kevesebb kaput tartalmaz.

(b) (c)

(a) NEM, (b) ÉS, (c) VAGY kapuk csak NEM-ÉS vagy csak NEM-VAGY kapukkal történő megvalósítása

Általában az áramkörtervezők Boole-függvénnyel kezdenek, és aztán alkalmazzák a Boole-algebra szabályait, és próbálnak egyszerűbb, de ekvivalens függvényt találni. A végleges formából azután létrehozzák az áramkört.

Ahhoz, hogy ezt a megközelítést használjuk, szükségünk van a Boole-algebra néhány azonosságára. Az ábra mutatja a legfontosabbakat.

Név	**ÉS forma**	**OR forma**
Identitásszabály	$1A = A$	$0 + A = A$
Nullszabály	$0A = 0$	$1 + A = 1$
Idempotens szabály	$AA = A$	$A + A = A$
Inverz szabály	$A\bar{A} = 0$	$A + \bar{A} = 1$
Kommutatív szabály	$AB = BA$	$A + B = B + A$
Asszociatív szabály	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Disztribúciós szabály	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Abszorpciós szabály	$A(A + B) = A$	$A + AB = A$
De Morgan-szabály	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\,\bar{B}$

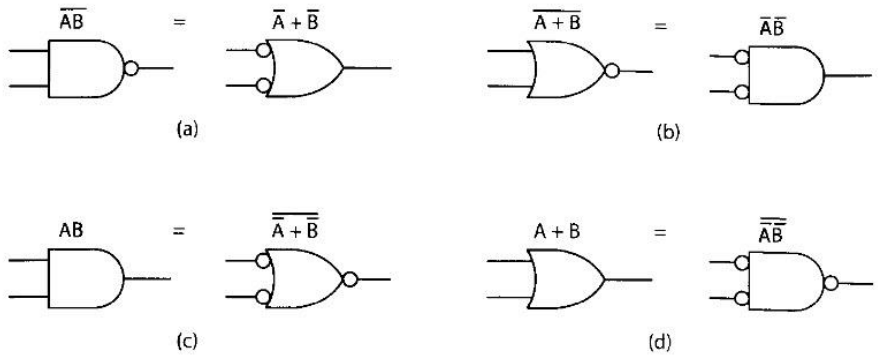
9. ábra: A Boole algebra néhány azonossága

Érdekes megjegyezni, hogy minden szabálynak két formája van, amelyek egymásnak duáljai. Az ES és VAGY, valamint a 0 és az 1 egyidejű cseréjével bármelyik forma előállítható a duálisából. Az összes szabály könnyen bizonyítható igazságtáblázatuk megalkotásával.

Név	ÉS forma	OR forma
Identitásszabály	1A = A	0+ A = A
Nullszabály	OA=O	1+A=1
Idempotens szabály	AA = A	A + A = A
Inverz szabály	AA=O	A+A=1
Kommutatív szabály	AB = BA	A + B = B + A
Asszociatív szabály	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Disztribúciós szabály	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Abszorpciós szabály	A(A + B) = A	A + AB = A
De Morgan-szabály	AB = A+ B	A+ B= A B

A Boole-algebra néhány azonossága

Néhány kapu alternatív jelölése: (a) NEM-ÉS. (b) NEM-VAGY. (c) ÉS. (cl) VAGY

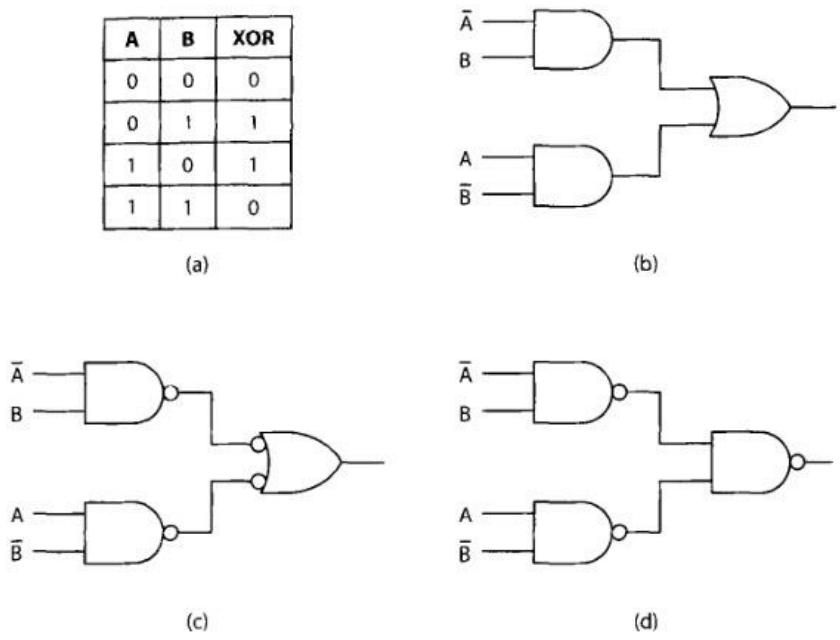


10. ábra: Néhány kapu alternatív jelölése: (a) NEM-ÉS. (b) NEM-VAGY. (c) ÉS. (cl) VAGY

A de Morgan-szabály, az abszorpciós szabály és a disztribúciós szabály ES formája kivételével az eredmények meglehetősen intuitívak. A de Morgan-szabályt kiterjeszthetjük két változónál többre is, például ABC = A + B + C.

A de Morgan-szabály egy alternatív jelölést sugall. Az (a) ábra az ÉS alakot mutatja, a negációt inverziós gömb jelöli mind a bemeneten, mind a kimeneten. Így egy VAGY kapu invertált bemenettel ckvivalcns egy NEM-ÉS kapuval. A (b) ábrából, a de Morgan-szabály duál formájából világos, hogy egy NEM-VAGY kapu meg-rajzolható egy invertált bemenetekkel rendelkező ÉS kapuval. A de Morgan-szabály mindkét formájának negálásával jutunk el a (c) és (d) ábrához, amelyek az ÉS- és VAGY-kapuk ekvivalens reprezentációit mutatják. Analóg szimbólumok léteznek a de Morgan-szabály többváltozós formáira (például egy n bemenettel rendelkező NEM-ÉS kapuból egy n invertált bemenettel rendelkező VAGY-kapu lesz).

Az ábra azonosságait és a hasonló több bemenetű kapukat használva könynyű átalakítani egy igazságtáblázat összeg-szorzat ábrázolását tisztán NEM-ÉS vagy tisztán NEM-VAGY formájúvá.



11. ábra: kizáró vagy (XOR) függvény igazságtáblázata. (b) – (d) Három áramkör ezen függvény kiszámítására

Példaként tekintsük a 11. (a) ábra KIZÁRÓ VAGY (EXCLUSIVE OR, XOR) függvényét. A szabvány szorzat-összeg áramkört a 11. (b) ábra mutatja. Azért, hogy NEM-ÉS formájúvá konvertáljuk, a vonalakat, amelyek az ÉS-kapuk kimenetét kötik össze a VAGY-kapuk bemenetével, újra kell rajzolni két inverziós gömbbel, ahogy a 11. (c) ábra mutatja. Végül a 10. (a) ábrát használva eljutunk a 11. (d) ábrához. Az A és B változókat A és B-ből NEM-ÉS és NEM-VAGY kapukkal úgy generálhatjuk, hogy a bemeneteket összekötjük. Megjegyezzük, hogy az inverziós gömbök oda mozdíthatók a vonalak mentén, ahova akarjuk, például a 11. (d) ábra bemeneti kapuinak kimeneteiről elmozdít-hatjuk a kimeneti kapu bemeneteihez.

Végezetül az áramköri ekvivalenciával kapcsolatban egy meglepő eredményt fogunk bemutatni, mely szerint azonos fizikai kapukkal különböző függvényt tudunk kiszámolni, attól függően, hogy milyen konvenciót használunk. A 12. (a) ábrán bemutatjuk egy bizonyos F kapu kimenetét különböző bemeneti kombinációkkal. Mind a bemenetet, mind a kimenetet feszültségben (volt) adjuk meg. Ha azt a konvenciót használjuk, hogy a 0 volt logikai 0-nak és a 3,3 volt vagy az 5 volt

A	B	F
0^V	0^V	0^V
0^V	5^V	0^V
5^V	0^V	0^V
5^V	5^V	5^V

(a)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

(b)

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

(c)

12. ábra: (a) Eszköz elektromos jellemzői. (b) pozitív logika (c) negativ logika

3.2. ÖSSZEFOGLALÁS

A logika tudománya a gondolkodással foglalkozik. A gondolkodást természetesen nem mint lelki folyamatot vizsgálja, hanem csak az úgynevezett gondolatformákat, ezeknek egymással való kapcsolatát és ezt a kapcsolatot létrehozó logikai műveletet. Központi problémája a helyes következtetések fogalmának szabatos tisztázása.

Állításon vagy kijelentésen olyan kijelentő mondatot értünk, amely egyértelműen igaz vagy hamis; tehát

a) nem lehet igaz is, hamis is (az ellentmondástalanság elve),

b) nem lehet az, hogy se nem igaz, se nem hamis (a kizárt harmadik elve).

3.3. ÖNELLENŐRZŐ KÉRDÉSEK

1. Fogalmazzuk meg az ellentmondástalanság és a harmadik kizárásának az elvét!
2. Mit értünk logikai műveleten?
3. Definiáljuk a negációt és adjuk meg logikai értéktáblázatát!

4. MATEMATIKAI ALAPOK – KÓDELMÉLETI ALAPOK

4.1. KÓDELMÉLET

Legyen $X = \{x_1, \ldots, x_n\}$ egy véges, nemüres halmaz. X-et ábécének, elemeit betűknek hívjuk. Az X elemeiből képzett $v = y_1 \ldots y_m$ sorozatokat X feletti szavaknak nevezzük; egy szó hosszán a benne szereplő betűk számát értjük. (A 0 hosszúságú szót üres szónak hívjuk).

Jelölje X* az X feletti véges hosszú szavak halmazát.

Legyen $Y = \{y_1, \ldots, y_m\}$ egy ábécé. Az x_* nyelv Y ábécé feletti kódolásán egy $f: X^* \to Y^*$ leképezést értünk; azaz minden X-beli szóhoz hozzárendelünk Y-beli szót úgy, hogy különböző X-beli szavakhoz különböző Y-beli szavakat rendelünk hozzá.

A továbbiakban Y = {0, 1}, azaz minden szóhoz egy bináris sorozatot rendelünk hozzá; ezt nevezzük bináris kódolásnak.

Rendeljük hozzá az $x_1, \ldots x_n$ betűkhöz rendre az $\alpha_1, \ldots, \alpha_n$ bináris sorozatok; egy tetszőleges szónak pedig feleltessük meg azt a bináris sorozatot, melyet a szó betűinek megfeleltetett bináris sorozatok egymás után írásából kapunk. Ezt nevezzük betű szerinti kódolásnak. $A\ K = \{\alpha_1, \ldots, \alpha_n\}$ halmaz a kód, $a\ z\ \alpha_i$-k a kódszavak. A továbbiakban betű szerinti kódolással foglalkozunk.

Példa: *Legyen* $X = \{a, b, c\}$, $K = \{00; 11; 011\}$. *Ekkor a baac szónak az 110000011 bináris sorozat felel meg.*

Egy kód felbontható, ha tetszőleges bináris sorozat legfeljebb egyféleképpen bontható fel kódszavakra.

Példák: $K_1 = \{110; 111; 001\}$. *A* K_1 *kód felbontható, mivel minden kódszó 3 hosszú, így tetszőleges üzenetet „hármasával" kell felbontani.*

K_2 = {10; 100;1000}. A K_2 kód is felbontható, mert az 1-esek jelzik az új kódszó kezdetét.

K_3 = {10;101; 010}. K_3 nem felbontható, mert az 101010 sorozatot felbonthatjuk 101010 vagy 101010 alakban is.

Egy K kód prefix kód, ha egyik kódszó sem kezdődik egy másik kódszóval. A fenti példákban a K_1 kód prefix volt, a K_2 és K_3 viszont nem, hiszen például a K_2 kódban az 100 kódszó az 10 kódszóval kezdődött.

Tétel: Minden prefix kód felbontható

(Megjegyzés: A tétel megfordítása viszont nem igaz. Ha egy kód nem prefix, attól még lehet felbontható; például a fenti K_2 kód ilyen.) Mivel egy kódról egyszerűen el lehet dönteni, hogy prefix-e, ezért a továbbiakban prefix kódokkal foglalkozunk.

Tétel: Legyen a *K* prefix kódban a kódszavak hossza rendre $l_1,\ldots,l_n$ Ekkor teljesül a McMillan-egyenlőtlenség:

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1 \ .$$

13. ábra:

Tétel: Legyenek (1 1 ,. . . , *ln)* olyan pozitív egész számok, melyekre

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1 \ .$$

14. ábra:

Ekkor létezik olyan prefix kód, melyben a kódszavak hossza rendre (l_i ..., ln).

A következő eljárás megkonstruál egy ilyen prefix kódot: feltehetjük, hogy az l_i-k nagyság szerint növekvő sorrendbe vannak rendezve (ha nem, akkor átrendezzük őket). Legyen:

$$a_1 = 0, \text{ és } a_i = \sum_{j=1}^{i-1} 2^{-l_j}, \text{ ahol } 2 \leq i \leq n.$$

15. ábra:

Irjuk fel az a_i-ket (ez n darab szám) kettes számrendszerbe, legyen *az* az α_i kettes számrendszerbeli alakjában a_i „kettedesvessző" utáni l_i bit. Ez egy prefix kód lesz, melyben a kódszavak hossza (l_i ,. .. , l_n).

Példa: Adjunk példát prefix kódra, melyben a kódszavak hossza (3,2,2,3,2)! Megoldás: Mivel $2^{-3} + 2^{-2} + 2^{-2} + 2^{-3} + 2^{-2} < 1$, ezért létezik ilyen prefix kód.

Rendezzük a hosszakat nagyság szerint növekvő sorrendbe: (2,2,2,3,3). Legyen $a_1 = 0 = 0{,}00$

$a_2 = 2^{-2} = 0, 01$

$a_3 = 2^{-2} + 2^{-2} = 0{,}10$

a 4 $2^{-2}+2^{-2}+2^{-2}$0,110

a_5 $2^{-2}+2^{-2}+2^{-2}+2^{-3}$=0,111

ahol az a_i-ket átírtuk kettes számrendszerbe. Legyen α_i 00, *α2* = 01, α_3 10, α_4 110, α_5=111. A keresett kód: *K*={110; *00;* 01; 111; 10}.

Amikor egy üzenetet szeretnénk kódolni, akkor az egyes betűk többnyire nem egyforma valószínűséggel fordulnak elő: például egy magyar szövegben az „e” betű sokkal gyakrabban fordul elő, mint a „w” betű. Tegyük fel, hogy azF forrás $a z\ x_1,\ldots,\mathbf{X}_n$ betűket rendre $p_l,\ldots,p_n$ valószínűségekkel bocsájtja ki.

Ha a *K* kódban a kódszavak hossza $l_1, \ldots, l_n,$ akkor egy *M* hosszú üzenet átlagos kódolt közlésének a hossza:

$$M \sum_{i=1}^{n} p_i l_i$$

16. ábra:

lesz. Ezért ha a kódolt üzenet átlagos hosszát szeretnénk minimalizálni, akkor azt a kódot kell megkeresnünk, melyre

$$\sum_{i=1}^{n} p_i l_i$$

17. ábra:

minimális.

Tegyük fel, hogy az *F* forrás az $x_1, \ldots, x_n$ betűket rendre $p_l, \ldots, p_n$ valószínűségekkel bocsájtja ki. A *K* kódban a kódszavak hossza pedig $l_l \ldots, l_n$. Az *F* forrás entrópiáján a

$$H(F) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

18. ábra:

mennyiséget értjük. A *K* kód költsége az

$$L(K) = \sum_{i=1}^{n} p_i l_i$$

19. ábra:

mennyiség

Definíció:A *K* felbontható kód optimális, ha a felbontható kódok között minimális a költsége.

Tétel: Mindig létezik optimális prefix kód.

Tétel: Az *F* forráshoz tartozó optimális K_{opt}

$$H(F) \leq L(K_{opt}) \leq H(F) + 1.$$

20. ábra:

Tétel: Legyen *a* $K=\{ \alpha_1, \ldots, \alpha_m \}$ kód optimális *a* $\{p_1, \ldots, p_m\}$ gyakorisági listára nézve, ahol $p_1 \geq p_2 \geq \ldots p_{j-1} \geq p_j \geq \ldots p_m$. Ha $p_j = q_m + q_{m+1}$ *és* $p_1 \geq p_2 \geq \ldots p_{j-1} \geq p_{j+1} \geq \ldots p_{m-1} \geq q_m \geq q_{m+1}$ akkor

K' $\{\alpha_1 \ldots, \alpha_{j-1}, \alpha_{j+1} \ldots, \alpha_{m-q}, \alpha 0, \alpha 1\}$ kód optimális a $\{p_1, p_2 \ldots, p_{j-1}, p_{j+1} \ldots, p_{m-1}, q_m, q_{m+1}\}$ gyakorisági listára nézve.

Ez a tétel lehetőséget ad az optimális kód kiszámítására. Az így előállított optimális kódot nevezzük Huffmann-kódnak.

A Huffmann-kód megkonstruálása: rendezzük a gyakoriságokat csökkenő sorrendbe; hagyjuk el a két utolsó elemét, az összegüket pedig írjuk be a listába úgy, hogy a rendezés megmaradjon. Folytassuk ezt az eljárást, amíg már csak 2 elem nem marad. Két betű esetén az optimális kód biztosan az, ha az egyikhez a 0-t, a másikhoz az egyest rendelünk hozzá. Ezt követően az előző tétel segítségével megkaphatjuk a 3 szavas optimális kódot, ebből a 4 szavasat végül az eredeti gyakorisági listához tartozó optimális kódot.

Példa: Az F forrás $a, b, c, d,$ *e betűket bocsátja ki* $0,1;\ 0,2;\ 0,2;\ 0,15;\ 0,35$ *valószínűségekkel. Adjunk meg egy optimális kódot!*

e	*0.35*	*0.35*	0.4	0.6
b	*0.2*	*0.25*	*0.35*	0.4
c	*0.2*	*0.2*	*0.25*	
d	0.15	*0.2*		
a	0.1			

21. ábra:

A táblázat első oszlopába beírtuk csökkenő sorrendbe a valószínűségeket, és mindegyik mellé megjegyeztük, hogy melyik betűhöz tartozik. Ezután a két legkisebb értéket elhagytuk, az összegüket pedig beírtuk a sorrendbe; így kaptuk a második oszlopot. Aláhúzással jelöltük, hogy melyik tagot kaptuk meg összegként. Ezután a második oszlop két legkisebb értékét elhagytuk, az összegüket pedig beírtuk a sorrendbe; így kaptuk a harmadik oszlopot. Ugyanígy jutottunk el az utolsó oszlophoz is. Most visszafelé haladva megkonstruálhatjuk az optimális kódot:

e	00	00	1	0	0
b	10	01	*00*	1	*1*
c	11	10	01		
d	010	11			
a	011				

22. ábra:

Az utolsó oszlopban a 2 betűhöz tartozó optimális kód a 0 és az 1. Mivel a 0-nak megfelelő 0,6-et kaptuk meg összegként, ezért a 0-ból csinálunk két új kódszót úgy, hogy egyszer 0-t, egyszer pedig 1-est írunk utána; az így kapott 2 új kódszót (00; 01) beírjuk az előző oszlop aljára(!), a többi kódszót (jelen esetben az 1-est) pedig változatlan sorrendben az előző oszlop „tetejére" írjuk. Ebben az oszlopban most az 1 kódszóhoz tartozó valószínűséget kaptuk meg összegként, ezért az 1-ből csinálunk két új kódszót, az 10-t és az 11-et; ezeket írjuk be a második oszlop „aljára"; a 00 és a 01 kódszavak pedig ebben a sorrendben az első oszlop tetejére kerülnek.

Végül a második oszlopban a 01-nek megfelelő valószínűséget kaptuk összegként; ezért a 01-ből 2 új kódszót, a 010-t és a 011-t készítjük, és tesszük az első oszlop végére; a 00, 10, 11 kódszavak pedig ebben a sorrendben az első oszlop tetejére kerülnek.

Így megkaptuk az optimális kódot: K (011; 10; 11; 010; 00)

4.2. HIBAFELISMERŐ ÉS HIBAJAVÍTÓ KÓDOLÁS

Az üzenetek továbbítása során az üzenetek sajnos megsérülhetnek; előfordulhat, *hogy* bizonyos bitek meghibásodnak, és 1-es helyett 0, vagy 0 helyett 1-es lesz belőlük. Hogyan lehetne úgy üzenetet küldeni, hogy a vevő észlelje, ha hiba keletkezett, vagy ki is tudja javítani?

Tegyük fel, *hogy egy* M üzenetet (bináris sorozatot) szeretnénk továbbítani úgy, hogy ha abban legfeljebb 1 db hiba keletkezik, akkor azt vevő észrevegye. Ez egyszerűen megoldható az alábbi módon: duplázzuk meg az üzenetet, azaz például a 0111 sorozat helyett küldjük el 01110111 sorozatot. Ekkor a vevő összehasonlítja a részt; ha azok megegyeznek, akkor nem keletkezett hiba, ha eltérnek, akkor meghibásodott az üzenet (azt persze nem tudjuk megmondani, hogy hol; viszont visszaírhatunk a küldőnek, hogy küldje el újra az üzenetet). Persze azért, *hogy* felismerjük, hogy hibás az üzenet, árat kellet fizetnünk: az üzenet 2-szer hosszabb lett az eredetinél.

Ennél sokkal „gazdaságosabb” a következő eljárás: tegyük az üzenetünk végére egy darab bitet, egy ún. paritásellenőrző bitet úgy, hogy az így kapott üzenetben páros darab 1-es legyen. Ha feltesszük, hogy legfeljebb 1 hiba keletkezik küldés közben, akkor egyszerű eldönteni, hogy hibás-e az üzenet, vagy sem: ha páros darab 1-es van benne, akkor nem keletkezett hiba, ha páratlan, akkor valahol hiba történt.

Mi a helyzet akkor, ha azt szeretnénk, hogy ha legfeljebb 1 hiba keletkezik az üzenetben, akkor azt a vevő ki is tudja javítani?!

Ez is megoldható többféleképpen: ha például az üzenetünket „megháromszorozzuk”, akkor 1 hiba esetén, többségi alapon el tudom dönteni, hogy mi volt az eredeti üzenet: pl. ha a 001101001 sorozatot kapjuk, akkor az eredeti üzenet a 001 sorozat volt. Persze most minden üzenet háromszor hosszabb lett!

A Hamming-kódolás segítségével egy 2^k—k—1 hosszú bináris sorozatot 2^k — 1 hosszú sorozatként elküldve egy hibát ki tudunk javítani. Az eljárás a következő: Tegyük fel például, *hogy* $k = 4$, azaz egy 11 hosszú bináris sorozatot szeretnénk továbbítani; legyen ez mondjuk a 01110000001. Ebből csinálunk egy 15 hosszú sorozatot; a sorozat 2 hatvány sorszámú helyeire – azaz első, második, negyedik, nyolcadik – helyen paritásellenőrző biteket fogunk írni, az eredeti üzenetet a többi helyre írjuk be: ..0.111.0000001

A pontoknak megfelelő helyeken fognak állni a paritásellenőrző bitek. Írjuk fel egymás alá azoknak a helyeknek a sorszámát(!) kettes számrendszerben, ahol 1-sek állnak:

0101

0110

0111

1111

Azt szeretnénk, *hogy* minden egyes oszlopban páros darab 1-es álljon. Ez az első oszlopban nem stimmel; ezt viszont a 8. helyen álló paritásellenőrző bittel ki tudjuk javítani, ha oda is 1-est írunk, hiszen ekkor a 8-as kettes számrendszerbeli alakját, az 1000-át is be kell írni, és így az első oszlop „megjavul".

Mivel a második oszlopban páros darab 1-es áll, ezért az üzenet 4. helyére 0-t kell írnunk.

A harmadik oszlopban páratlan darab 1-es van, ezért a 2. helyen álló paritásellenőrző bitnek 1-nek kell lennie.

A negyedik oszlopban is páratlan darab egyes áll, ezért az 1. helyen álló parításellenőrző bitnek 1-esnek kell lennie.

Vagyis az elküldött 15 hosszú üzenet az 110011110000001.

Most tegyük fel, hogy kapunk egy 15 hosszú üzenetet a fenti Hamming-kódolással kódolva. Hogyan lehet kitalálni, *hogy* melyik bit hibásodott meg, ha egyáltalán történt hiba?

Mondjuk a 011100000111000 üzenetet kapjuk. Írjuk fel ismét a kettes számrendszerben azoknak a helyeknek a sorszámát, ahol 1-es áll:

0101

0110

0111

1111

Látható, *hogy* az elő oszlopban páratlan darab 1-es van, a többiben páros; ez csak úgy lehet, ha az 1000 helyen, azaz a 8.helyen álló bit meghibásodott, azaz az a „jó" üzenet a 011100010111000 üzenet volt; ebből ha elhagyjuk a paritásellenőrző biteket, akkor megkapjuk az eredeti 10000111000 üzenetet.

4.3. KAKTEREK ÉS KÓDOLÁSUK

4.3.1. Karakterek és karakterkészletek

A karaktert [character] (a számhoz hasonlóan) fogalomnak tekintjük, amit meg kell tudni jeleníteni írott formában, illetve el kell tudni tárolni a memóriában vagy bármely más tárolóeszközön, fájlban, illetve továbbítani kell tudni egy informatikai hálózaton. Karakter lehet az ábécé egy betűje, egy szám, egy írásjel (a szóközt is beleértve) vagy egyéb más írásrendszerben használt jel illetve vezérlő karakter (például soremelés) is.

Karakterkészlet [character set, charset] alatt karakterek kiválasztott csoportját értjük (a kiválasztás lehet tetszőleges, de általában valamely ország, nemzetiség, nyelv, régió alapján történik).

4.3.2. Karakterek kódolása

Karakterek. kódolása [character encoding, coded character set] alatt a karakterekhez valamilyen érték (kód) rendelését értjük. Ilyen például a Morze-kód, ami karakterekhez hosszabb és rövidebb impulzusokból álló kódokat rendel (amiket aztán könnyen lehet továbbítani például egy rádió adó-vevő segtségével), vagy a Braille-kód, ami karakterekhez 3D objektumokat rendel (amit aztán megfelelő technológiával „kinyomtatva” látásukban sérült emberek is képesek elolvasni). Az informatikában a karakterkódolás általában egy szám karakterhez rendelését jelenti (amit aztán valamilyen módszerrel tárolunk).

> ***Karakterek tárolási formáján*** **[characer encoding form & character encoding scheme] a karakterkódok (számok) konkrét tárolási móriját értjük. Ez lehet triviális, például hogy egy megfelelő hosszúságú, egészek tárolására használt ábrázolást alkalmazunk, vagy lehet szofisztikáltabb, például egy tömörebb — de bonyolultabb — változó kódhosszúságú kódolás esetében.**

Egyes kódolási módszerek egyszerre meghatározzák a karakterek kódolását és a kódok tárolási formáját. Tipikusan ilyen kódolási módszerek a klasszikus kódtáblák [code page, character map, charmap], amelyek általában egy bájton ábrázolnak egy karaktert.

4.3.3. Klasszikus kódtáblák

Az ASCII-kódtábla

Az Arnerican Standard Code for Information Interchange (ASCII) 7-bites kódtábla látható az 23. ábrán. Az egyes karakterek kódja a karakter sorának és oszlopának fejlécéből adódik: például a @ kódja 0x40, a W kódja 0x57.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

23. ábra: Az ASCII 7-bites kódtábla (forrás: wikipedia.org/wilci/ASCII)

Nagyon fontos kiemelni, hogy az ASCII kódtábla 7-bites, azaz 128 karakter kódolására alkalmas. Ha 8 biten tároljuk vagy továbbítjuk, a legnagyobb helyiértólc i bitet nullára állítjuk.

Az ISO 8859-X kódtáblák

Az ISO/IEC 8859-X kódtáblák az ISO és az IEC szabványosító testületek közös kódtáblái, céljuk, hogy minél több regionális karaktert tartalmazzanak. Az ASCII kódtábla az ISO 8859-X kódtáblák része, azaz miniden ASCII-kód egyben érvényes ISO 8859-X kód is. Mi a leggyakrabban az ISO 8859-1 (nyugat-európai) és az ISO 8859-2 (közép-európai) kódtáblákkal találkozhatunk, ezeket szokás latin-1 és latin-2 kódtábláknak is nevezni. Az ISO 8859-1 kódtáblát ISO 8859-15 (latin-9) néven frissítették (többek között belekerült az euro karakter), és hasonló történt az ISO 8859-2-vel is: ISO 8859-16 (latin-10) néven frissítették.

A Windows kódtáblái

Az ASCII kiterjesztésével a Microsoft megalkotta, saját 8-bites kódtábláit, külön-külön az egyes régiókra. Mi a leggyakrabban a Windows-1252 (közép-európai) és a Windows-1252 (nyugat-európai) kódkészletekkel találkozhatunk. Ezeket szokás Windows latin-2 illetve Windows latin-1 kódtábláknak is nevezni.

Szintén fontos tulajdonsága ezeknek a kódoknak is, hogy az ASCII-t módosítás nélkül tartalmazzák, azaz annak csak kiegészítései. Az ISO 8859-X kódtáblák és a Windows-xxxx kódtáblák nagy részben egyeznek (nem kizárólag a közös ASCII részek), de nem teljes mértékben kompatibilisek egymással.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	€ 20AC		‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152		Ž 017D	
90		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153		ž 017E	Ÿ 0178
A0	NBSP 00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C0	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D0	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E0	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F0	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

24. ábra:

Feladatok

1. feladat Megaltozik-e egy adott ASCII-karakter kódja, ha előjeles vagy előjel nélküli egészként tároljuk?

2. feladat Keressük meg az ISO 8859-1, az ISO 8859-2, a Windows-1250 és a Windows-1252 kódtáblák kiosztásait!

3. feladat Helyes eredményt kapunk, ha egy ASCII-vel kódolt karakterekből álló fájlt a) Windows-1250 b) Windows-1252 kódtáblák alapján próbálom értelmezni?

4. feladat Mi lehet a magyarázata annak, hogy régebben (sajnos sokszor még ma is) az ő illetve (J betűk helyett (hibásan) ő vagy O szerepelt?

5. feladat Tudunk-e több, különböző kódkészlethez tartozó karaktert egyetlen szövegfájlban tárolni (és azokat helyesen megjeleníteni olvasáskor)?

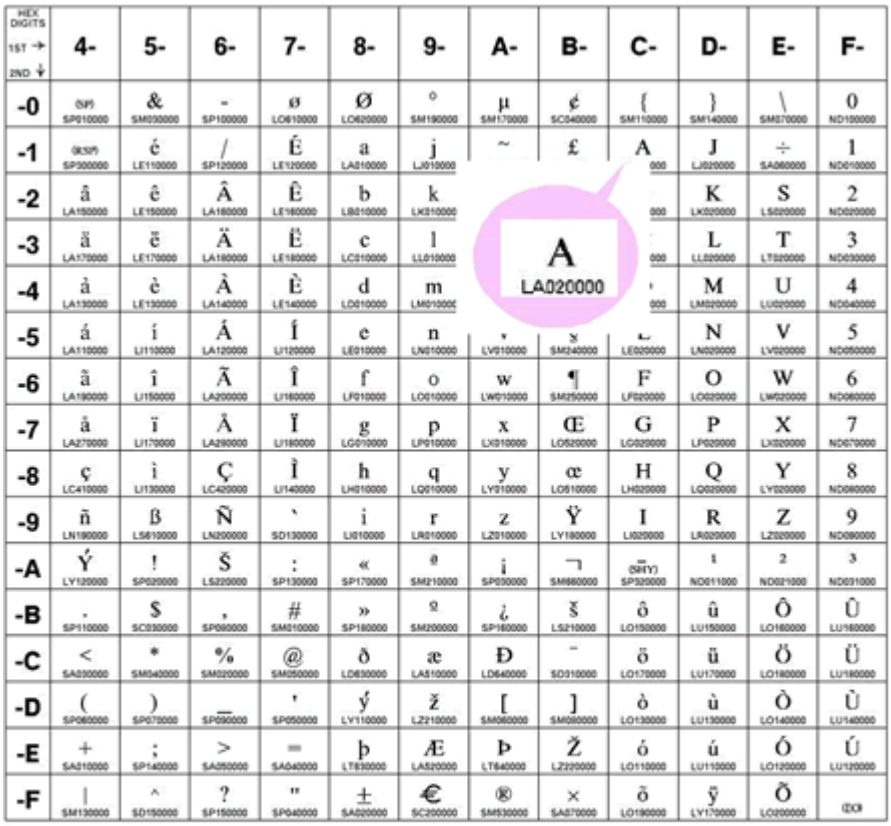
6. feladat Adjunk példát olyan szövegfájlra, amely nemcsak ASCII-karaktereket tartalmaz, de mé1s azonos módon értelmezhető a) ISO 8859-2 és windows-1250 b) ISO 8859-1 és windows-1252 kódtáblákkal!

4.3.4. A Unicode

Az előzőekben ismertetett kódtáblák közös problémája, hogy önmagukban nem képesek többnyelvű szövegek tárolására (sőt, egyes esetekben még egyetlen nyelv esetében sem, például: kínai, japán), mivel a — 8 bites tárolásból adódó — lehetséges 256 különböző karakter nyilván nem elegendő a világ összes nyelvében használt betű és írásjel ábrázolására. Ennek a megoldására jött létre a Unicode konzorcium, amely létrehozta és karbantartja a Unicode ajánlást. Jelenleg (Unicode 6.0) 93 Másrendszert és több mint százezer karaktert tartalmaz, amelyek között élő és holt nyelvek mellett megtalálhatók matematikai és egyéb szimbólumok is.

1. feladat A tivww.unicode.org/charts/ oldalon nézzük meg az Unicode által támogatott karaktereket!

A Unicode azonos az ISO/IEC 10646 szabvánnyal. A Unicode egy karakter kódolási szabvány (figyelem, önmagában nem kódtábla!), amely meghatározza, hogy egy konkrét karakternek mennyi a Unicode-értéke [code point]. Általában U+hexa-kód formában jelöljük: például az euro jel kódja: U+20AC. Ezt a Unicode-értéket különböző módokon lehet tárolni.



25. ábra:

Az UTF-8

Az UTF-8 a Unicode egy tárolási formája, (Unicode Transformation Format) ami a Unicode kódokat változó hosszon, 1–4 bájton tárolja, a kód értékétől függően. Legfontosabb tulajdonságai:

- ASCII kompatibilis, azaz minden ASCII-szöveg egyben helyes UTF-8 szöveg is,
- önszinkronizáló, azaz nem kell az UTF-8 bájtsorozat elejéről kezdeni az olvasást, hogy pontosan el lehessen határolni az egyes karaktereket reprezentáló UTF-8 byte-csoportokat.

Az UTF-16

Az UTF-16 tárolási forma változó hosszon, 2–4 bájton tárolja a Unicode kódokat. Nem ASCII-kompatibilis.

Az UTF-32

Az UTF-32 tárolási forma fix hosszon, 4 bájton tárolja a Unicode kódokat. A kódolás nagyon egyszerű: az Unicode kódokat kell 4 bájtos egészekként tárolni. Nem ASCII-kompatibilis.

Feladatok

1. feladat Hogyan befolyásolják a tárolási méretet a használt karakterek? Mikor érelemes az UTF-8 és mikor az UTF-16 kórtolási formát választanunk?

2. feladat Az UTF-8 önszinkronizáló tulajdonságának milyen szerepe van a véletlenül kiválasztott pozícióból történő megjelenítésre, és a hibás átvitelből adódó értelmezési problémákra?

3. feladat Az UTF-8, UTF-16, UTF-32 kódolások közül melyek alkalmasak szövegek véletlenszerű elérésére (azaz például ha az x. karakterhez akarok közvetlenül ugrani)?

Szövegfájlok

Ha egy fájlban csak szöveget akarunk tárolni (pontosabban csak olyan karaktereket, amelyek mindegyike megtalálható egy kiválasztott karakterkészletben), akkor nincs más dolgunk, mint a karakterek (kódtáblája vagy valamely Unicode tárolási formája szerinti) bájtjait egymás után írni, és ezt eltárolni. Valójában is ez történik, ezeket a fájlokat nevezzük *egyszerű szövegfájloknak* [plain text file], kiterjesztésük (általában): TXT.

Feladatok

1. feladat Hasonlítsuk össze a 7 tárolási formáit: a.) előjel nélküli egészként, b.) kettes komplemens ábrázolású előjeles egészként, c.) ASCII-kódolással d.) UTF-8 kódolással!

2. feladat Honnan tudjuk, hogy egy szövegfájl beolvasásakor a kódokat melyik kódtábla szerint kell értelmeznünk?

3. feladat Pusztán a szövegfájlba történő beleolvasással eldönthető-e általános esetben, hogy az milyen kódtábla szerint értelmezendő?

4. feladat Töltsünk be a böngészőnkbe egy szövegfájlt, majd módosítsuk a kódtáblát! Kódoljuk át a szövegfájlt az iconv parancs segítségével, és nézzük meg az eredményt a böngészővel!

5. feladat Mi a mojibake? (Keressünk rá!)

6. feladat Ha ékezetes karaktereket használunk egy SMS-ben, akkor van-e különbség az egy SMS-ben elküldhető karakterek száma között, ha magyar (jobbra dőlő) ékezetekkel rendelkező karaktereket vagy csak balra dőlő ékezetekkel rendelkező karaktereket használunk? Indokoljuk meg!

5. DIGITÁLIS, GÉPI SZÁMÁBRÁZOLÁS

5.1. MIT JELENT A GÉPI SZÁMÁBRÁZOLÁS?

A gépi számábrázolás a számok (számító)gépek memóriájában vagy háttértárán történő tárolását jelenti.

5.1.1. Előjel nélküli egész számok ábrázolása

Az előjel nélküli (nemnegatív) egész számok [unsigned integer] ábrázolása megegyezik a bináris számrendszernél megismert leírással: $86[_{10}] = 01010110_{[2]}$, azaz egy nemnegatív egész számot a kettes számrendszerbe átváltott formájában tárolunk. A tömörebb írásmód miatt ugyanakkor ezt legtöbbször nem bináris, hanem hexadecimális formában írjuk le. (Ne feledjük, hogy a bináris-hexadecimális átváltás nem más, mint négy bitenkénti csoportosítás.

Mivel a gyakorlatban általában csak egész byte méretű ábrázolásokat használunk, az előjel nélküli egészek is legtöbbször 1, 2, 4, 8, 16 byte (8, 16, 32, 64, 128 bit) hosszúak lehetnek. Így is hívjuk ezeket: 8 bites előjel nélküli egész, 16 bites előjel nélküli egész stb.

A $_{46[10]}$ számot a memóriában a következőképpen tároljuk 1 bájton: 00101110.

5.1.2. Előjeles egész számok ábrázolása

Az előjeles egész számok [signed integer] ábrázolására az első gondolatunk az lehet, hogy az előjel nélküli egészek ábrázolásához egy előjelet. jelentő bit hozzáadásával (ami például 0 ha pozitív az előjel és 1, ha negatív az előjel) egyszerűen meg lehet oldani a problémát. Sajnos azonban ez a megoldás sok szempontból nem megfelelő: a legkézenfekvőbb probléma, hogy ezzel a módszerrel lehetséges a +0 és a -0 ábrázolása is, ami zavarhoz vezet (például a „nulla-e” vizsgálatot így két különböző értékre kell megtenni), továbbá az ilyen módon felírt számokkal végzett műveletek bonyolultabbak, mint amennyire az feltétlenül szükséges lenne.

Sokkal jobb eredményre vezet a *kettes komplemens* ábrázolás: ahelyett, hogy egy előjelbittel jelölnénk az előjelet, a negatív számokat úgy ábrázoljuk, hogy hozzáadjuk őket egy nagy pozitív számhoz, és az eredményt ábrázoljuk, mint egy előjel nélküli egész számot. Ahhoz, hogy az eredmény biztosan pozitív

legyen (hogy előjel nélküli egészként felírhassuk) de ugyanakkor ne is legyen túl nagy (hogy minél kevesebb biten felírható legyen az eredmény), a következő módszert alkalmazzuk: ha N biten akarjuk tárolni a kettes *komplemens* számot, akkor a negatív számhoz hozzáadandó nagy pozitív szám legyen a következő: 2^N.

A 2 kettes komplemens ábrázolása 8 biten: $2^8 + (-2) = 256 - 2 = 254$, azaz: 11111110.

A –17 kettes komplemens ábrázolása 8 biten: $2^8 + (-17) = 256 - 17 = 239$, azaz: 11101111.
A kettes komplemens ábrázolást az előző módszeren kívül megkaphatjuk úgy is, hogy a negatív számhoz egyet hozzáadunk, az eredmény abszolút értékét binárisan ábrázoljuk a megadott biten, és a számjegyeket invertáljuk.

A –2 kettes komplemens ábrázolása 8 biten: – 2 + 1 = –1 ennek abszolút értéke: 1 = 00000001, invertálva: 11111110.

A –17 kettes komplemens ábrázolása 8 biten: – 1 7 + 1 = –16 ennek abszolút értéke: 16 00010000, invertálva: 11101111.

Fontos tudnivalók:

- Kettes komplemens ábrázolásban is lehetséges nemnegatív számok ábrázolása, amelynek módja megegyezik az előjel nélküli egészek tárolási módjával. (Azaz ebben az esetben nem kell az előzőekben ismertetett műveleteket elvégeznünk.)
- A kettes komplemens ábrázolásban már csak egyetlen ábrázolási módja van a nullának.

7. feladat Adjuk meg a 0 kettes komplemens ábrázolását 8, 16, 32, 64 biten!

8. feladat Adjuk meg a – 1 kettes komplemens ábrázolását 8, 16, 32, 64 biten! Adjuk meg az 1 kettes komplemens ábrázolását 8 biten!

5.1.3. *Egész számok* ábrázolási határai

Az N biten történő, előjel nélküli egész számábrázolás esetén a tárolható legkisebb érték: 0, a tárolható legnagyobb érték: $2^N - 1$.

Ha 8 bites előjel nélküli egész ábrázolást használunk, akkor a legkisebb ábrázolható szám a 00000000 (értéke 0), a legnagyobb ábrázolható szám az 11111111 (értéke $2^8 – 1 = 255$).

1. feladat Mennyi a legnagyobb tárolható érték 8, 16, 32, 64 bites előjel nélküli egész esetében?

2. feladat Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, előjel nélküli egész számábrázolás esetében?

Ha kettes komplemens módon ábrázolunk egy egész számot és ehhez N bit áll rendelkezésre, akkor a tárolható legkisebb érték: -2^{N-1}, a tárolható legnagyobb érték: $2^{N-1}-1$.

Ha 8 bites kettes komplemens ábrázolást használunk, akkor a legkisebb ábrázolható szánt az 10000000 (értéke –128), a legnagyobb ábrázolható szám a 01111111 (értéke 127).

3. feladat Kettes komplemens ábrázolás esetén miért nem ugyanannyi szám tárolható a pozitív és a negatív tartományban? (Azaz miért nem –127 és 127 vagy –128 és 128 a két határ?)

4. feladat Mennyi az értéke a kettes komplemens ábrázolással, 8 biten tárolt 11111111 illetve a 00000000 számoknak?

5. feladat Eldönthető-e egyszerűen (ránézésre) egy kettes komplemens módon ábrázolt számról, hogy az negatív vagy pozitív?

6. feladat Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, kettes komplemens számábrázolás esetében?

5.2.4. Túlcsordulás

Az egész számok véges biten történő ábrázolása miatt mindig van legkisebb és legnagyobb ábrázolható szám. Amikor műveletet végzünk, elképzelhető, hogy a művelet eredménye már nem ábrázolható az operandusokkal megegyező méretben. Ezt a jelenséget túlcsordulásnak [overflow] nevezzük.

Ha 8 bites előjel nélküli egészekkel dolgozunk, a 156 + 172 = 328 összeget már nem tudjuk 8 biten tárolni (mert a legnagyobb tárolható érték a 255).

Ha 8 bites előjeles egészekkel dolgozunk, a –84 + (–79) = –163 összeget már nem tudjuk 8 biten tárolni (mert a legkisebb tárolható érték a –127).

Túlcsordulás esetén – megvalósítástól függően – lehetséges

- *levágás:* a túlcsordult eredmény még ábrázolható részét tároljuk (a számláló kvázi körbefordul, mint például egy gázóránál, amelynél fix számú helyiértéken történik a mérés),
- *a szaturáció:* a túlcsordult eredmény helyett a legnagyobb illetve legkisebb ábrázolható értéket tároljuk.

Levágás: ha 8 bites előjel nélküli egészekkel dolgozunk, a $156_{[10]}$ = $10011100_{[2]}$ és a $172_{[10]}$ = $10101100_{[2]}$ valódi összege ($328_{[10]}$ = $101001000_{[2]}$) helyett annak a 8 utolsó bitjét tároljuk: 01001000.

Szaturáció: ha 8 bites előjel nélküli egészekkel dolgozunk, a $156_{[10]}$ = $10011100_{[2]}$ és a. $172_{[10]}$ = $10101100_{[2]}$ valódi összege ($328_{[10]}$ = $101001000_{[2]}$) helyett az ábrázolható legnagyobb számot tároljuk: 11111111.

1. feladat Mi az Y2K-probléma? Mi a kapcsolat a túlcsordulás és az Y2K-probléma között?

5.2.5. Lebegőpontos számábrázolás

Nem egész számok gépi ábrázolására a lebegőpontos [floating point] ábrázolást használjuk: a számot először átalakítjuk normalizált alakba, és az így kapott alak különböző részeit külön-külön tároljuk.

Egy szám normalizált alakján olyan szorzatra bontását értjük, ahol a második tag a számrendszer alapjának valamely hatványa (ezt a szám nagyságrendjének is nevezzük), az első tag értéke pedig annyi, hogy a második taggal megszorozva az eredeti számot kapjuk, továbbá az első tag legalább 1, de a számrendszer alapjánál kisebb értékű.

Tízes számrendszerben a 382 normalizált alakja: 3.82×10^2, a 3.875 normalizált alakja 3.875 _ 10^0, a 0.00000651 normalizált alakja 6.51×10^{-6}.

Az előzőekben definiált első tagot a szám *mantisszájának* [mantissa, significand], a hatványkitevőt (a nagyságrendet) a szám *karakterisztikáidnak*

vagy exponensének [exponent] nevezzük. Negatív számok tárolásához szükség van még az előjelre (ami 1 bit méretű, értéke 0, ha pozitív a szám és 1, ha negatív a szám). Egy szám ábrázolásához tehát ezeket az értékeket kell eltárolni.

Kettes számrendszerben ábrázolva a számot, a normalizált alak tovább egyszerűsödik, hiszen a mantissza tizedespontja előtt mindig 1 áll („az első tag legalább 1, de a számrendszer alapjánál kisebb értékű" feltétel miatt), amit így nem kell eltárolni, és ezt a megtakarított bitet a mantissza pontosabb tárolására lehet fordítani. Fontos, hogy a lebegőpontos szám értelmezésekor ezt az el nem tárolt számjegyet is figyelembe vegyük?

A lebegőpontos elnevezés abból adódik, hogy az ábrázolható számok nem fix számú tizedesjeggyel vannak tárolva[4], hanem a karakterisztika alapján a mantisszában tárolt érték tizedespontja „mozog".

A karakterisztika is lehet pozitív vagy negatív, de ebben az esetben nem az egész számoknál megismert előjeles (kettes komplemens) tárolást használjuk, hanem az *eltolt* [excess] tárolást: a tárolandó karakterisztikához hozzáadunk egy kellően nagy, fix értéket, és az így kapott eredményt tároljuk el, ami már biztosan pozitív egész szám.

A $382_{[10]}101111110_{[2]}$ normalizált alakja: $1.01111110_{[2]}\ 2^{8}{}_{[10]}$= $1.01111110_{[2]}\ 2^{1000}{}_{[2]}$, azaz tárolandó a 0 előjelbit, a 01111110 mantissza és az 1000 karakterisztika (a megfelelő eltolással).

A $-3{,}375[_{10}] = -11{,}011_{[2]}$ normalizált alakja $-1{,}1011_{[2]} x -2^{1}$, azaz tárolandó az 1 előjelbit, a 1011 mantissza és az 1 karakterisztika (a megfelelő eltolással).

A mantissza 10 biten, az exponens 5 biten történő excess -15 ábrázolása esetén a tízezer ábrázolása: 10000 = $1.0011100010\ 8192$ = $1.0011100010\ 2^{13}$, azaz tárolandó a 0 előjel, a. 0011100010 mantissza és az 11100 exponens.

1. feladat Hasonlítsuk össze a 8 bites előjel nélküli egész, az előjelbites egész, a kettes komplemes és a 127-tel eltolt (excess-127) számábrázolásokat! (Táblázatosan foglaljuk össze: egy sor legyen a tárolt 8 bit, az oszlopok legyenek a vizsgált ábrázolási módok, egy adott mezőbe írjuk be a mező sorának megfelelő bitsorozat értelmezését az oszlopnak megfelelő számábrázolás esetében!)

[4] ezt a módot fixpontos ábrázolásnak nevezzük

5.2.6. Lebegőpontos számábrázolás határai és pontossága

Az előjeles vagy előjel nélküli egész számok ábrázolásakor és a lebegőpontos számábrázolás esetében is csak fix értékek tárolhatók, de míg ezek az egészek esetében pontosan megegyeznek a tárolni kívánt egész számokkal, a lebegőpontos számábrázolás esetében ez nincs így, mivel végtelen sok valós szám nyilván nem ábrázolható véges helyen. Ezt úgy is értelmezhetjük, hogy a tárolás során kényszerű kerekítés történik. Mindezek miatt az ábrázolási határok mellett a pontosság is jellemez egy-egy konkrét lebegőpontos számábrázolást, ami megadja, hogy egy adott szám tárolása esetén a tárolni kívánt és a tárolt szám értéke legfeljebb milyen távol lehet egymástól. A lebegőpontos számok normalizált alakú tárolásából következik, hogy a (relatív) pontosságot a mantissza tárolási mérete határozza meg, az ábrázolási határok pedig elsődlegesen a karakterisztika méretéből adódnak. Fontos azt is kiemelni, hogy, a pontosság az ábrázolási tartományban abszolút értelemben nem egyenletes, azaz függ az ábrázolni kívánt számtól.

A mantissza. 10 biten, az exponens 5 biten történő excess -15 ábrázolása esetén a tízezernél nagyobb, pontosan ábrázolható számok közül a legkisebb a 0 előjelű, a 0011100011 tárolt mantisszájú és a 11100 tárolt exponensű szám: $1.0011100011_{[2]} \times 2^{13} = 1.2216796875\ 8192 = 10008$, a, tízezernél kisebb, pontosan ábrázolható számok közül a legnagyobb a 0 előjelű, 0011100001 tárolt mantisszájú és a 11100 tárolt exponensű szám: $1.0011100001_{[2]} \cdot 2^{13} = 1.2197265625\ 8192 = 9992$, azaz tízezer körül a hiba kevesebb, mint 8.

Az IEEE binary16 számábrázolás esetén az egy tízezrednél kisebb, pontosan ábrázolható számok közül a legnagyobb a 0 előjelű, 1010001101 tárolt mantisszájú és 00001 tárolt exponensű szám: $1.1010001101 \times 2^{-14} = 1.6376953125 \times 0.00006103515625 = 0.00009995698929$, a tízezrednél nagyobb, pontosan ábrázolható számok közül a legkisebb a 0 előjelű, a 1010001110 tárolt mantisszájú és a 00001 tárolt exponensű szám: $1.1010001110\ 2^{-14} = 1.638671875$

$0.00006103515625 = 0.00010001659393$, azaz egy tízezred körül a hiba kevesebb, mint egy tízmilliomod.

2. feladat A mantissza 10 biten, az exponens 5 biten történő ábrázolása esetén adjuk meg (az előző két példához hasonló módon) az ezernél, a

száznál, a tíznél, az egy tizednél, az egy századnál és az egy ezrednél nagyobb számok közül a le„ll„kisebb ábrázolhatót illetve a kisebb számok közül a legnagyobb ábrázolhatót, és számítsuk ki (közelítőleg) a hibát.
A lebegőpontos számok ábrázolásának a gyakorlatban is alkalmazott nemzetközi szabványa a az IEEE 754 = IEC 599 = ISO/IEC 60559.

5.2.7. Alulcsordulás

A túlcsorduláshoz (ami pozitív vagy negatív irányban túl nagy szám ábrázolásának kísérletét jelenti, azaz a számábrázolási határt léptük túl) hasonló az alulcsordulás [underflow]: itt a számábrázolás másik paramétere, a pontosság szenved csorbát: olyan kis számot akarunk ábrázolni, ami már csak nullaként ábrázolható.

elnevezés	mantissza méret	karakterisztika méret	karakterisztika eltolás
binary16	10	5	15
binary32	23	8	127
binary64	52	11	1023
binary128	112	15	16383

26. ábra: IEEE 754 =IEC 599 = ISO/IEC 60559 bináris lebegőpontos típusok jellemzői

Az alulcsordulási hiba csökkentésére a szabványnak megfelelve a nulla exponens értéket. speciálisan kell kezelni: a mantissza legnagyobb helyiértékű bitjét az előzőekben megismert fix 1 helyett nullának kell értelmezni, és az exponens excess értékéhez is egyet hozzá kell adni:

Az IEEE binary16 formátumban tárolt 0000000000000001 bitminta értelmezése: előjel: 0, exponens: 00000 (az eredeti 0 – 15 értelmezés helyett a módosított exponens: -14), mantissza: 0000000001, azaz: $0.0000000001 \; x.2^{-14} = 2^{-24} = 5.9604645 \; 10^{-8}$

1. feladat A mantissza 10 biten, az exponens 5 biten történő ábrázolása esetén adjuk meg a legnagyobb ábrázolható negatív számot.

5.2.8. Végtelenek és a NaN

A szabványos lebegőpontos számábrázolások a valós számokon kívül képesek tárolni a ∞-t és –∞-t, továbbá a speciális NaN (Not a Number) értéket. Ez utóbbit kapjuk eredményül (többek között) akkor, ha nullát nullával osztunk vagy ha negatív számból vonunk négyzetgyököt.

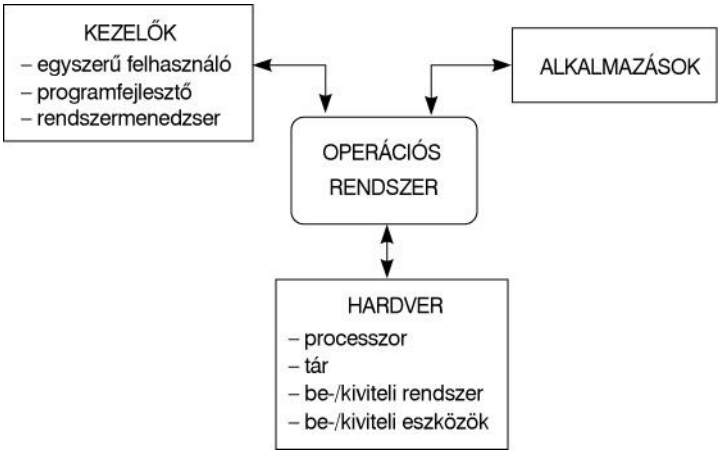
Ha az exponens minden bitje 1 és a mantissza nulla, akkor az (előjeltől függően) $\pm \infty$ az ábrázolt érték, ha a mantissza nem nulla, akkor NaN az ábrázolt érték.

1. feladat Adjuk meg a legnagyobb binary16-ban ábrázolható számot!

6. RENDSZERSZINTEK ÉS A RENDSZERSZOFTVER GÉPI SZINTJE

6.1. STRUKTURÁLT SZÁMÍTÓGÉP FELÉPÍTÉS

A rendszermodell felállításához vizsgáljuk meg a következő ábrát, amely az operációs rendszer úgynevezett kontextdiagramját mutatja be.



27. ábra:

A kontextdiagram alapján az operációs rendszer fő környezeti kapcsolatai:

- a kezelők (operátorok),
- alkalmazások vagy alkalmazói programok,
- a számítógéphardver.

Az operációs rendszernek a működése során ezek felé kell csatlakozási felületet nyújtania, és alapvető funkcióit a környezeti kapcsolatokban mutatott viselkedésével jellemezhetjük.

6.2. AZ OPERÁCIÓS RENDSZEREK FUNKCIÓI

Röviden fogalmazva az operációs rendszernek két alapvető feladata van:

- *kényelmesen* használható virtuális gép megvalósítása a kezelők és az alkalmazások felé,
- a számítógéphardver *hatékony és biztonságos* működtetése.

A kényelmesen használható virtuális gép azt jelenti, hogy a felhasználók számára áttekinthető modelleket kell kínálni, és azokat meg kell valósítani a fizikai rendszeren. Másként fogalmazva: a fizikai rendszer kényelmes és célszerű absztrakcióit kell implementálni az adott számítógép-konfiguráción. Ilyen absztrakciók például az önálló logikai processzorral és tárterülettel rendelkező, egymástól védett, de egymással kommunikálni tudó, párhuzamosan végrehajtódó *folyamatok,* a fizikai címtartomány méretét meghaladó címezhető tártartomány *(virtuális tár)*, a *fájl* és a katalogizált *fájlrendszer,* ahol megoldott a biztonságos és védett adattárolás stb.

A számítógéphardver hatékony működtetése azt jelenti, hogy az alkalmazások futtatását úgy kell koordinálni, hogy a fizikai eszközök kihasználtsága minél jobb legyen, és a felhasználó által előírt prioritások teljesüljenek.

A biztonságos működtetés pedig azt jelenti, hogy az operációs rendszer korrekt megoldásokat tartalmaz a megosztott erőforrás-használatból eredő problémákra (például közös fizikai processzor használata, a fizikai tár kiosztása, ugyanazon készülékre indított több, egyidejű be-kiviteli művelet végrehajtása), valamint elfedi a tranziens hardverhibákat (például hibatűrő, hibajavító kódolás adattárolások, adatátvitelek esetén).

6.2.1. Csatlakozási felületek

A következőkben az operációs rendszer kapcsolódási felületeit jellemezzük.

Kezelői (operátori) felület (Operator Interface, User Interface)

A kezelői felület egyfajta ember-gép kapcsolat. Arra szolgál, hogy az operációs rendszer ezen keresztül működtethető legyen, illetve működéséről a felhasználó tájékoztatást kapjon.

A rendszert kezelő felhasználók a következő jellegzetes csoportokra oszthatók: *egyszerű felhasználókra, alkalmazásfejlesztőkre,* és *rendszermenedzserekre*.

Az *egyszerű felhasználók* jellegzetes tevékenysége, hogy programokat (alkalmazásokat) futtatnak, amelyek segítségével szokásos napi feladataikat látják el (például irodában dokumentumkezelés, táblázatszerkesztés; automatizált üzemben diszpécserfeladatok stb.).

Az egyszerű felhasználó számára tehát az operációs rendszer olyan gép, amelyik egy felhasználói körnek lehetőséget ad adat- és programfájlok védett és rendezett tárolására, valamint alkalmazások futtatására. A rendszer legfontosabb szolgáltatásai: bejelentkezés, a rendelkezésre álló alkalmazások áttekintése, alkalmazások indítása, esetleges együttműködése és leállítása, fájlműveletek (másolás, mozgatás, törlés, tartalomfüggő feldolgozás).

Az *alkalmazásfejlesztők* részletes ismeretekkel rendelkeznek az operációs rendszer alkalmazói programok számára nyújtott szolgáltatásairól. Ezeket az ismereteiket a programkészítésben használják fel. Mint kezelők, az általuk készített alkalmazások tesztelésével, teljesítményelemzésével is foglalkoznak.

Az alkalmazásfejlesztőknek nyújtott szolgáltatások tehát már bizonyos mértékű bepillantást engednek az operációs rendszer belső szerkezetébe, a futó programokhoz rendelt fizikai eszközök állapotát is láthatóvá, esetleg közvetlenül módosíthatóvá teszik.

A *rendszermenedzser* feladata az operációs rendszer üzemeltetése, valamennyi ezzel kapcsolatos probléma megoldása. Ez magában foglalja egyrészt a *rendszergenerálás* feladatát, ami azt jelenti, hogy a rendszert a rendelkezésre álló hardverhez és az ellátandó feladatokhoz illesztett kiépítésben kell telepíteni. Másrészt *adminisztrációs* feladatokat, ami a rendszer felhasználóinak, a rendszerhez kapcsolódó alkalmazásoknak a nyilvántartását, jogosultságaik kiosztását, a rendszer üzemeltetési szabályainak, biztonsági előírásainak meghatározását, azok betartásának felügyeletét jelenti. Harmadrészt *hangolási* feladatokat, ami a hardver lehetőségeit, a tipikus alkalmazásokat és a rendszer statisztikáit figyelembe véve azoknak a rendszerparamétereknek beállítását jelenti, amelyek az üzemeltetés hatékonyságát befolyásolják (pufferméretek, ütemezési és kiosztási algoritmusok stb.). Negyedrészt *rendszerfelügyeletet* takar, ami a zavartalan, folyamatos működés biztosítását, az esetleges rendellenességek észlelését, elhárítását, az ennek érdekében szükséges időszakos feladatok ellátását (karbantartások, mentések stb.) jelenti.

Alkalmazási (programozói) felület (Application Interface, Program Interface)

A számítógéprendszeren futó, adott feladatokat megoldó programok valamilyen programozási nyelven (pl. FORTRAN, Pascal, C, C++ stb.) készülnek. Előkészítési időben (fordítás, szerkesztés) történik meg a program átalakítása a processzor által végrehajtható formára (gépi kód). A program készítője általában feltételezi, hogy a program valamilyen operációs rendszer felügyelete alatt fut, az operációs rendszer pedig kész, előre programozott megoldásokat tartalmaz például a be- és kiviteli műveletekre, az időkezelésre, a dinamikus tár-

igények kielégítésére, a programok együttműködésének és információcseréjének megoldására.

Az alkalmazások számára az operációs rendszer egy olyan gép, amelyik kiterjeszti a processzor utasításkészletét. A számítógép és az operációs rendszer együtt egy kiterjesztett utasításkészletű, új gépet alkot, amelyik a processzor utasításkészletén kívül tartalmazza az operációs rendszer műveleteit is. Ez a kiterjesztett gép a futtatható programok rendelkezésére áll.

Hardverfelület (Hardware Interface)

A hardver fejlődésének köszönhetően egyre újabb, nagyobb teljesítményű processzorok, be- és kiviteli eszközök, továbbá összekapcsolási módok, architektúrák alakulnak ki. Az operációs rendszerek működtetik a hardvert, igyekeznek hatékonyan kihasználni a hardver lehetőségeit. Az operációs rendszer és a hardver kapcsolódási felülete több ponton valósul meg.

- Az operációs rendszer maga is program, ami az adott számítógéprendszeren fut, tehát a processzor és az architektúra lehetőségei (utasításkészlet, regiszterkészlet, a rendszerhívás megvalósítása, megszakítási rendszer, címzési módok, be- és kiviteli rendszer) azok, amiket az operációs rendszer felhasználhat.
- Az operációs rendszer kezeli a hardvereszközöket. Az alkalmazások számára tárat, processzor használatot, lemezterületet biztosít, végrehajtja az alkalmazások által kezdeményezett be- és kiviteli műveleteket.
- Az operációs rendszernek kezelnie kell a rendszerhez kapcsolódó be- és kiviteli eszközöket, amelyek igen heterogén csoportokból kerülhetnek ki, és a rendszer élettartama során új eszközök beillesztésére is sor kerülhet.

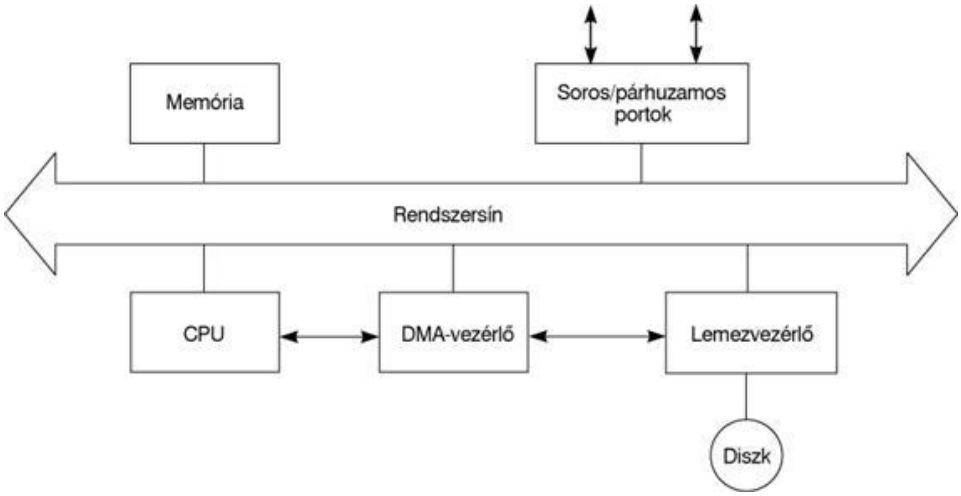
6.3. SZÁMÍTÓGÉP-ARCHITEKTÚRÁK

A mikrokontrollerektől a szuperszámítógépekig a különböző méretű, teljesítményű és szerkezetű hardvereket operációs rendszerrel kell működtetni. Meglehetősen nehéz megtalálni ezekben a rendszerekben azokat a közös vonásokat, amelyek meghatározzák, hogy mit kell tennie az operációs rendszernek a hatékony működtetés érdekében.

Illusztrációként három jellegzetes hardverfelépítést mutatunk be: egy egyszerű mikrogépet, egy tipikus személyi számítógépet, valamint egy szuperszámítógépet.

6.3.1. Egyszerű mikrogép

Az egyszerű mikroszámítógép felépítését a következő ábra mutatja.

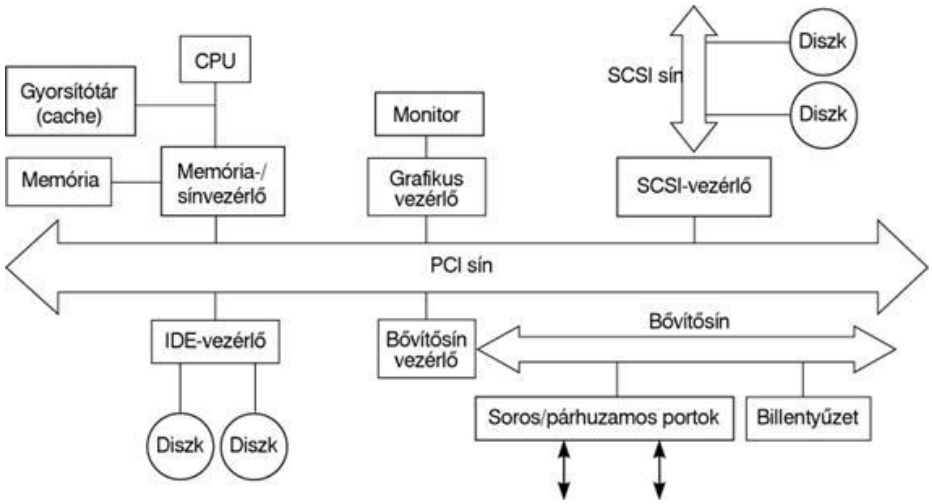


28. ábra: Egyszerű mikrogép architektúrája

Az egyszerű mikrogép egyetlen sínnel rendelkezik, amelyet mind a CPU-memória, mind a B/K (beviteli/kiviteli) forgalom terhel. A CPU egyben a sínvezérlő szerepét is betölti. Esetleges kezelőszervek a soros/párhuzamos portokra csatlakoztathatók. A DMA-vezérlő képes önállóan szervezni a lemez – memória közötti blokkátvitelt, az adatforgalom azonban a rendszersínt veszi igénybe. A rendszersín használatának szervezésére a DMA-vezérlő és a CPU között külön jelforgalom van. A sín lehetőséget ad megszakításkérések továbbítására is.

6.3.2. Jellegzetes személyi számítógép

Ez az architektúra már lényegesen bonyolultabb, és olyan megoldásokat tartalmaz, amelyeket korábban csak a nagyszámítógépekben alkalmaztak. A gyorsítótár megfelelő találati arány esetén lehetővé teszi, hogy a memória a CPU-utasítás végrehajtásával egyidejűleg a B/K-adatforgalom számára is rendelkezésre álljon. A grafikus vezérlő saját videomemóriával és intelligens processzorral rendelkezhet, aminek hatása, hogy a grafikus műveletek nem vesznek igénybe nagy sávszélességet a PCI-sínen.

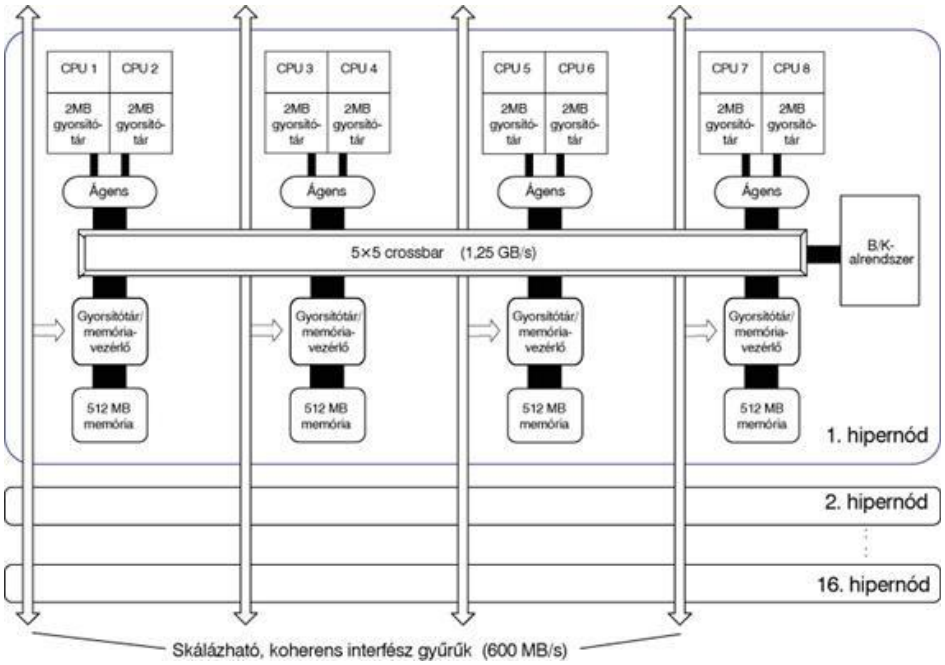


29. ábra: Személyi számítógép architektúrája

A bővítősín általában lassabb külső készülékek csatlakoztatására ad lehetőséget, külön vezérlő hajtja meg, biztosítva befelé a megfelelő hatékonyságú sínfoglalást. Természetesen a sín megszakításkéréseket is tud továbbítani. Két további vezérlő (IDE és SCSI) ad lehetőséget jellegzetesen lemezegységek csatlakoztatására, ugyancsak autonóm blokkátviteli képességekkel.

6.3.3. Szuperszámítógép

A következő ábrán a Convex Exemplar számítógépcsalád belső felépítését látjuk, amelyik lehetővé teszi, hogy 128 processzor (HP PA-RISC típusúak) hatékony együttműködésével alakuljon ki igen nagy teljesítményű számítási rendszer. Az építkezés hierarchikus, kettős processzoregységekből négy építhető egy *hipernódba*. Minden CPU saját gyorsító utasítás- és adattárral *(cache)* rendelkezik. A *hipernódhoz* egy B/K-alrendszer is tartozik. A crossbar kapcsolóhálózat a processzorpárok, valamint a B/K-alrendszer hatékony és átlapolt kapcsolatát teszi lehetővé a négy memóriablokkal. A memóriablokkok tartalmazzák a *hipernód* saját adatterületét, a közös tár egy adatterületét, valamint a *hipernódok* kapcsolatának hatékonyságát fokozó belső hálózati gyorsítótárak területét. A *hipernódokat* négy SCI *(Scalable Coherent Interface)* szabvány szerint kialakított gyűrű kapcsolja össze. A rendszerben valamennyi processzor számára rendelkezésre áll a memóriában egy közös címtartomány, amely valójában egy elosztott, koherens gyorsítótárral támogatott memóriával valósul meg.



30. ábra: Szuperszámítógép-architektúra

Ha általános következtetéseket akarunk levonni a bemutatott architektúrákból, azt állapíthatjuk meg, hogy egy bizonyos számítási teljesítmény alatt az architektúrák megtartják a gépi utasításkészlet szintjén a Neumann-modell szerinti szekvenciális utasítás-végrehajtást, de legalábbis ennek a látszatát, és ennek, valamint az ezzel átlapolódó B/K-műveleteknek a támogatására adnak egyre hatékonyabb eszközöket. A valódi működés elosztottá és többszálúvá válik, azonban ez a programozó elől rejtve marad.

6.4. RÉTEGEK ÉS MODULOK

A strukturált programozás alapelveinek következetes betartása rétegszerkezetű programrendszert eredményez. A rétegek hierarchikusan egymásra épülnek. Minden réteg úgy fogható fel, hogy az alatta lévő réteget – mint **virtuális gépet** – használva egy bonyolultabb virtuális gépet valósít meg a felette elhelyezkedő réteg számára. Ebből következik, hogy minden réteg csak a közvetlen alatta elhelyezkedő réteg „utasításkészletét” használhatja.

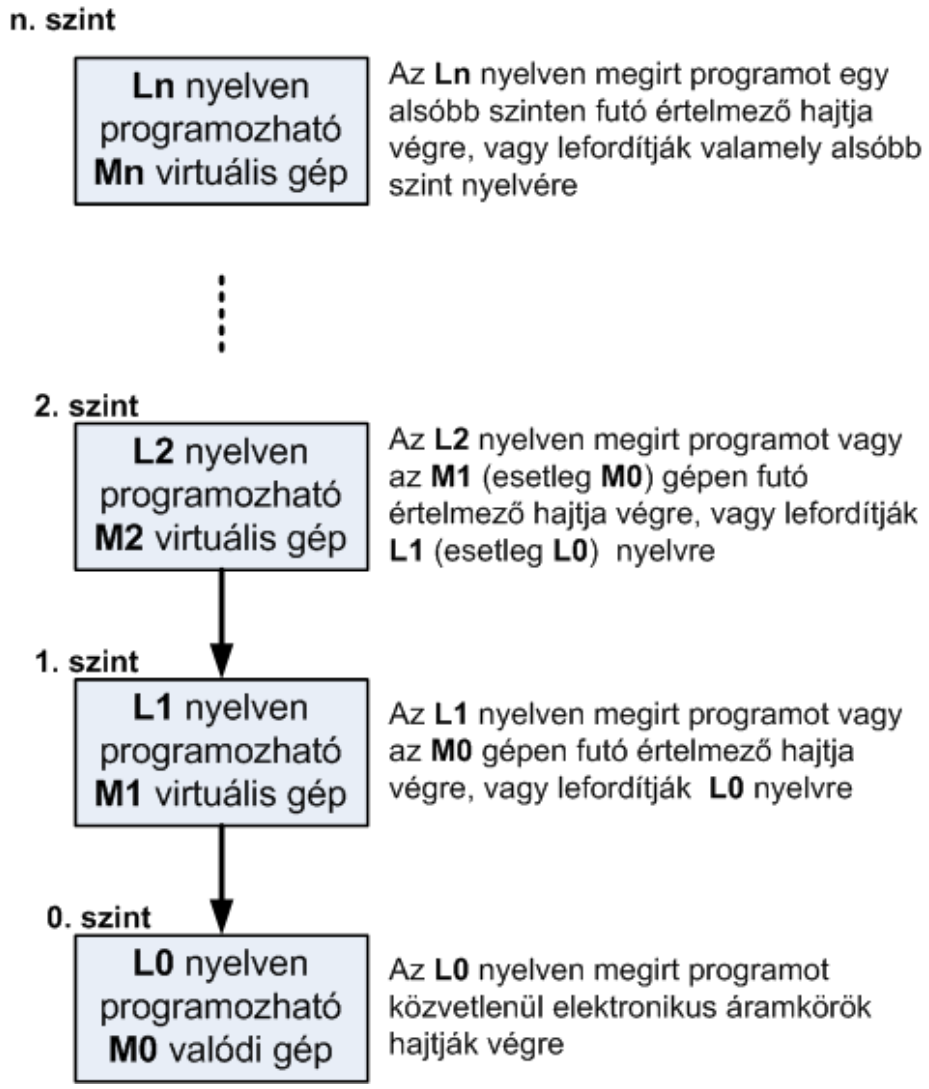
A továbbiakban egy általánosított modellen keresztül mutatjuk be a számítógép-architektúrák szintjeit.

A rétegek száma legalább kettő, de általában a 6 rétegű strukturális vagy 6 rétegű architektúrák szerint választják szét a különböző működési vagy absztrakciós szinteket.

A szintek bevezetésével együtt bevezettünk egy új fogalmat is: a virtuális gépet. A virtuális géphez (nevezzük **Mx**-nek) saját gépi nyelv (legyen ez **Lx**) tartozik, amelyet értelmezni és végrehajtani képes. A virtuálisgép-modellel tetszés szerinti bonyolultságú, többszintű modellt lehet felépíteni, mindössze arra kell csak figyelni, hogy az egymásra épülő szintek – a különböző virtuális gépek – „kimenete" olyan legyen, hogy azt képes legyen az alsóbb szint értelmezni és végrehajtani. Könnyen belátható, hogy e modell legalján van az **M0** virtuális gép, az **L0** nyelvével, ami valójában a valódi gép. E szint fölött helyezkedik el az **M1** virtuális gép, amelyet **L1** nyelven programozhatunk, amelyet vagy az **M1** virtuális gép fordít le **L0** nyelvre, vagy az **M0** virtuális gépen futó értelmező (interpreter) hajt végre. A következő szinten lévő **M2** virtuális gép **L2** nyelven programozható, amely programot vagy az **M1** (ritkábban az **M0**) gépen futó értelmező hajt végre, vagy lefordítják **L1** (ritkábban **L0**) nyelvre. Ezen meggondolások alapján a szintek száma tetszés szerint növelhető.

A modell biztosítja, hogy az *n.* szintű virtuális gépre írt programnak nem kell törődnie az alsóbb szinten futó fordítókkal (translator) és értelmezőkkel, számára ugyanis érdektelen, hogy a virtuális gép programját hány és milyen fordító vagy értelmező hajtja végre. Egy dologban biztos lehet a program írója: a program végrehajtása meg fog történni.

Gyakorlati okokból – költséghatékonyság és végrehajtási idő – a szintek számát nem célszerű nagyon megnövelni, ugyanakkor nagyon lecsökkenteni sem.



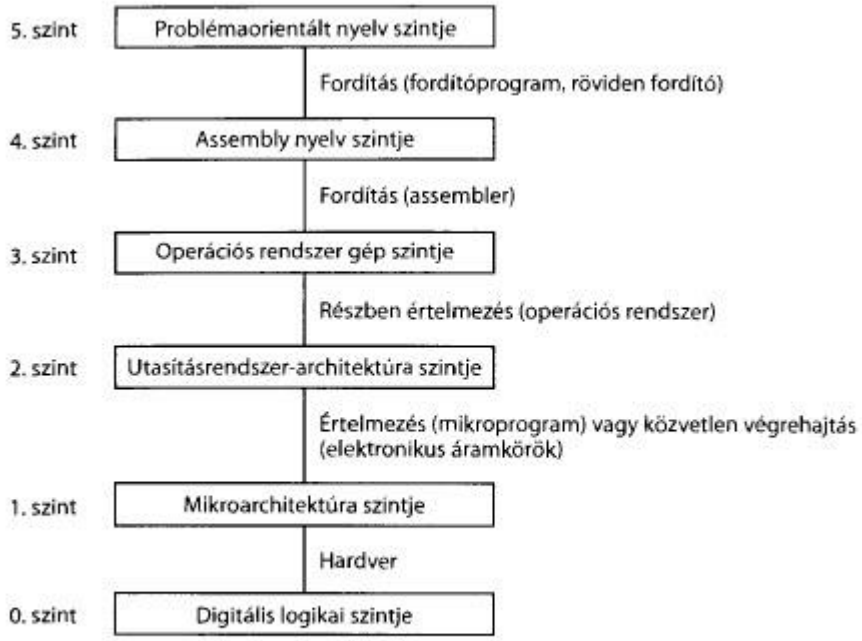
31. ábra: Többszintű gép architektúrája

6.4.1. Korszerű többszintű számítógépek

A legtöbb mai számítógép két- vagy többszintű.

Alul a 0. szint a gép valódi hardvere. Ennek az áram körei hajtják végre az 1. szintű gépi nyelvű programokat. A teljesség kedvéért meg kell jegyeznünk, hogy a mi 0. szintünk alatt is van még egy szint. Ez az úgynevezett **eszközszint.** Ezen a szinten találja a tervező az egyes tranzisztorokat, mint a számítógép-tervezés legalsó szintű építőköveit.

Az általunk vizsgált legalsó, **digitális logika szintjén** a **kapuk** a lényeges elemek. Bár a kapuk olyan analóg alkatrészekből épülnek fel, mint például a tranzisztorok, szerepük szerint digitális eszközöknek tekinthetők. Minden kapunak egy vagy több digitális bemenete van (a 0 vagy az 1 értéket reprezentáló jelek), kimenetként pedig ezekből egyszerű függvényértékeket számolnak ki, mint amilyen az AND (logikai „és") vagy az OR (logikai „vagy"). Egy kapu legfeljebb néhány tranzisztorból áll. Néhány kapuból összeállítható egy 1 bites memória, amely a 0 vagy az 1 értéket képes tárolni. Az 1 bites memóriákat 16-os, 32-es vagy 64-es csoportokba rendezve készíthetünk például regisztereket. Minden **regiszterben** meghatározott értékhatárig egy bináris számot tárolhatunk. Kapukból építhetjük fel magát az aritmetikai egységet is.



Egy hatszintű számítógép. Az egyes szinteket megvalósító módszert a szint alatt jelöltük (zárójelben a megvalósító program nevével)

32. ábra:

A következő felsőbb szint a **mikroarchitektúra szintje.** Ezen a szinten találjuk az (általában) 8–32 elemű, lokális memóriaként használt regiszterkészletet és az ún. **aritmetikai-logikai egységet (Arithmetic Logic Unit, ALU),** amely az egyszerű aritmetikai műveletek elvégzésére képes. A regiszterek az ALU-hoz kapcsolódnak, az adatok áramlásának útja az **adatút.** Az adatút alap feladata az, hogy kiválasszon egy vagy két regisztert, az ALU-val műveletet végeztessen el

rajtuk (például adja össze a tartalmukat), az eredményt pedig valamelyik regiszterben tárolja.

Egyes gépeken az adatút működését az ún. **mikroprogram** vezérli, míg más gépeken a vezérlés közvetlenül a hardver feladata Ezt a szintet korábban „mikroprogram szintnek" is nevezték, mert ezen a szinten a régebben szinte kivétel nélkül szoftverértelmezőt használtak. Az adat utat manapság gyakran (legalább is részben) közvetlenül a hardver vezérli, emiatt változott az elnevezés.

Azokon a gépeken, amelyeken az adatút szoftvervezérlésű, a mikroprogram egy értelmező program, amely a 2. szintű utasításokat egyenként betölti, elemzi és az adatutat használva végrehajtja. Az ADD utasítás esetében például betölti az utasítást, megkeresi és regiszterekbe helyezi az operandusait, az ALU kiszámolja az összeget, végül az eredményt a megfelelő helyre elküldi. Ha a gépen az adatút hardver vezérlésű, akkor is hasonlók az egyes lépések, csak akkor nem tárolt program vezérli a 2. szintű utasítások értelmezését.

A 2. szintet **utasításrendszer-architektúra szintjének (Instruction Set Arehitecture, ISA-szint)** nevezzük. Minden számítógépgyártó vállalat az általa forgalmazott gépekhez ad egy kézikönyvet. „Ezek általában az ISA-szintről szólnak, az alacsonyabb szinteket nem tárgyalják. A gépi utasításrendszer leírása tulajdonképpen nem más, mint azoknak az utasításoknak a leírása, amelyeket a mikroprogram vagy a hardvervégrehajtó áramkör értelmez. Ha a gyártó kétféle ISA-szintű értelmezőt biztosít ugyanahhoz a számítógépéhez, két „gépi nyelv" referencia kézikönyvet kell adnia.

A következő szint általában egy kevert szint. A szint nyelvéhez tartozó utasítások többsége az ISA-szinten is megvan. (Semmi akadálya annak, hogy valamely szint utasításai más szintek utasításai között is szerepeljenek.) Ezen felül a szint új utasításokkal, eltérően memóriaszervezéssel, több program egyidejű futtatásának képességével és egyéb tulajdonságokkal rendelkezik.

A 3. szinten sokkal változatosabbak a konstrukciók, mint az 1. vagy a 2. szinten. A 3. szint új szolgáltatásait a 2. szinten futó értelmező biztosítja, amelyet hagyományosan operációs rendszernek szoktunk nevezni. A 2. szintről örökölt 3. szintű utasításokat nem az operációs rendszer, hanem közvetlenül a mikroprogram (vagy a hardver) hajtja végre. Másként fogalmazva, a 3. szintű utasítások egy részét az operációs rendszer, más részét közvetlenül a mikro program értelmezi. Ezt értjük a „kevert" szint elnevezés alatt. A mellékelt ábrán ezt a szintet **operációs rendszer gép szintjének** nevezzük.

A 3. és a 4. szint között alapvető eltérés van. Az alsó három szintet nem egyszerű hétköznapi programozóknak találták ki, azok elsősorban a magasabb

szinteken szükséges értelmezők és fordítók futtatására szolgálnak. Ezeket az értelmezőket és fordítókat **rendszerprogramozók** írják, akik új virtuális gépek tervezésére és megvalósítására szakosodtak. A 4. és az e fölötti szinteket az alkalmazási feladatokat megoldó programozóknak szánták.

A 4. szint és a magasabb szintek megvalósításának módjában is változás van. A 2. és a 3. szintet mindig értelmezővel, míg a 4. szintet és az e fölöttieket általában – de nem mindig – fordítóval valósítják meg.

Az 1., 2. és 3. szint, illetve a 4., 5. és magasabb szintek nyelveinek természetében találhatjuk a további különbséget. Az 1., 2. és 3. szint gépi nyelvei numerikusak, a rajtuk írt programok hosszú számsorozatok, amely kedvező a gépek, de kedvezőtlen az ember számára.

A 4. szinttől kezdődően a nyelvek szavakból és az ember számára is jelentéssel bíró rövidítésekből állnak.

A 4. az **assembly nyelv szintje** valójában az alsóbb szintekhez tartozó nyelvek szimbolikus formája. Ezen a szinten lehet az 1., 2. és 3. szintekre programot írni azok virtuális gépeinek saját nyelveinél kényelmesebb nyelven. Az assembly nyelvű programokat először lefordítjuk az 1., 2. vagy 3. szint nyelvére, majd értelmeztetjük a megfelelő virtuális vagy valódi géppel. A fordítást végző programot **assemblernek** nevezzük.

Az 5. szint nyelveit az alkalmazási feladatokat megoldó programozóknak tervezik.

Az ilyen nyelveket szokták **magas szintű nyelveknek** nevezni, és több száz van belőlük. Az ezeken írt programokat általában a 3. vagy a 4. szint nyelvére fordítják az **ún. fordítóprogramok,** de esetenként találkozunk értelmezőkkel is. A Java-programokat például először egy ISA-szerű nyelvre, Java-bájtkódra szokták fordítani, amelyet azután egy értelmező hajt végre.

Néha az 5. szint egy speciális alkalmazási területre, például a szimbolikus matematikára kidolgozott értelmező. Ez a területen felmerülő problémák megoldásához az adatokat és műveleteket olyan formában biztosítja, amelyet a területen jártas szakemberek könnyen megértenek.

Összefoglalásként emlékeztetünk arra, hogy a számítógépeket egymásra épülő szintek sorozataként tervezik. Minden szint egy önálló absztrakciónak felel meg különböző elemekkel és műveletekkel. A számítógépek ilyen tervezése és tanulmányozása során időlegesen eltekinthetünk a lényegtelen részletektől, ezzel egy bonyolult tárgyat könnyebben érthetőre szűkíthetünk.

Egy-egy szint adattípusainak, műveleteinek és szolgáltatásainak összességét a szint **architektúrájának** nevezzük. Az architektúra a szint használója által

látható tulajdonságokat foglalja egységbe. A programozó által látható tulajdonságok, mint például, hogy mennyi a rendelkezésre álló memória, az architektúrához tartoznak.

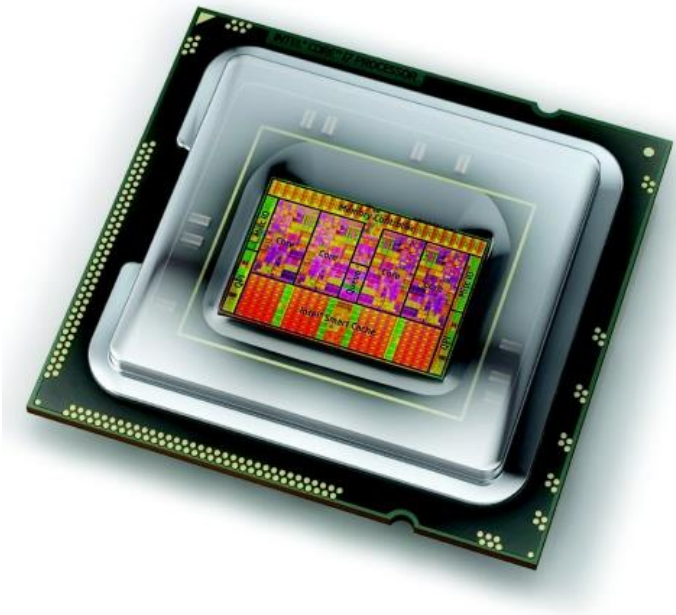
A megvalósítás részletei – például, hogy milyen áramköri elemek valósítják meg a memóriát – nem része az architektúrának. A programozó által látható számítógépes rendszerelemek tervezésével a **számítógép-architektúra** foglalkozik.

A hétköznapi gyakorlatban a számítógép architektúra és a számítógépek felépítése lényegében ugyanazt jelentik.

7. A BELSŐ ADATTÁROLÁS ARCHITEKTÚRÁJA

7.1. A KÖZPONTI TÁR FELADATAI[5]

A multiprogramozott rendszerekben a CPU-t több folyamat **megosztottan** használja. Ahhoz, hogy megfelelő működési sebességet tudjunk biztosítani, egyszerre több folyamatot is a **központi tárban (main storage, memory)** kell tartanunk, hiszen, mint korábban láttuk, a háttértárak elérési ideje – gyorsító tár alkalmazása ellenére is – sokkal nagyobb a főtárénál, így nagyon lassú lenne, ha környezetváltásnál a háttértárról kellene behozni, illetve a háttértárra kellene kivinni a folyamatokat. Vagyis a központi tár igen fontos erőforrás, melyért több folyamat verseng, és szervezése, kezelése az operációs rendszerek tervezését, megvalósítását és teljesítményét befolyásoló egyik legfontosabb tényező.



33. ábra:

[5] http://www.tankonyvtar.hu/hu/tartalom/tkt/operacios-rendszerek/ch05s04.html

7.2. A FŐTÁR MEGOSZTÁSA A FOLYAMATOK KÖZÖTT

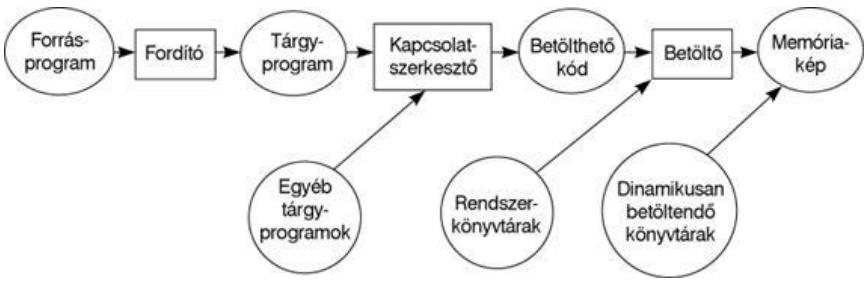
Ebben a részben a „klasszikus” tárkezelés két legfontosabb kérdését, a program címeinek kötését és a tárallokációt vizsgáljuk meg részleteiben.

7.2.1. A program címeinek kötése

A CPU közvetlenül csak a központi tárhoz tud hozzáférni, így a folyamatok aktuálisan végrehajtandó utasításának és az operandusoknak az operatív tárban kell elhelyezkedniük. Ez azt jelenti, hogy a programokat, adatokat végrehajtás előtt a másodlagos (háttér-) tárakból be kell tölteni a központi (fizikai) memória valamilyen címére.

Egy programhoz általában lineáris, folytonos címtartományt szoktunk képzelni, amely 0-tól kezdődően (a program eleje) a program maximális céméig terjed. Ezt hívjuk **logikai címtartománynak (logical address space)**. A mai rendszerekben a programok végrehajtása gyakorlatilag soha nem a 0. címtől kezdve, azaz a logikaival egyező **fizikai címtartományban (physical address space)** történik.

Egy-egy program lefordításakor, szerkesztésekor a fordító és a kapcsolatszerkesztő program első menetben általában a logikai címtartományban ad értékeket a szimbolikus hivatkozásoknak. Ugyanakkor a végrehajtáshoz már a fizikai címekre van szükség. A **logikai és fizikai címtartomány közötti megfeleltetés, leképezés (mapping)** elvileg bármelyik lépés alatt elvégezhető, de valamikor biztosan meg kell tenni. Nézzük meg tehát, hogy mikor (a program előkészítésének melyik fázisában) milyen lehetőségeink vannak a leképezés elvégzésére (34. ábra).



34. ábra: Logikai-fizikai címtranszformáció a felhasználói programok többlépcsős feldolgozása során

Statikus logikai-fizikai címleképezés

A címkonverzió történhet **fordítás** közben **(compile time)**. Ha ismeretesek a program fizikai címei, akkor a leképezés elvégezhető a fordítás alatt. Ha a későbbiek során a program (fizikai) kezdőcíme megváltozik, akkor azt újra le kell fordítani. Éppen ezért, ezt a megoldást – merevsége miatt – csak speciális esetekben, a ROM (Read Only Memory, csak olvasható tár) memóriába kerülő programok esetén használják, egyébként pedig nem a végleges fizikai cím, hanem logikai cím rendelődik a tárgykódban a hivatkozásokhoz.

A végleges fizikai címek kötésére a következő lehetőség **szerkesztés** közben adódik **(link time)**. Szerkesztésre akkor van szükség, ha egy program több, egymástól függetlenül lefordított modulból áll, amelyek hivatkoznak más modulban definiált logikai címre. A kapcsolatszerkesztő (linker) program feladata a modulok közötti kereszthivatkozások feloldása, és a modulok egymás mögé helyezése. Ennek eredményeképpen a tárgykódban a logikai címek helyére fizikai címek helyettesítődhetnek, azonban az esetek többségében csak a modulok logikai címtartományainak egyesítése történik meg, és egy **áthelyezhető kódot (relocatable code)** kapunk eredményül.

Ha a végleges címkonverzió **betöltés** közben **(load time)** történik, akkor a korábban kapott áthelyezhető kód címhivatkozásait módosítja a betöltő program (loader) az aktuális címkiosztás szerint.

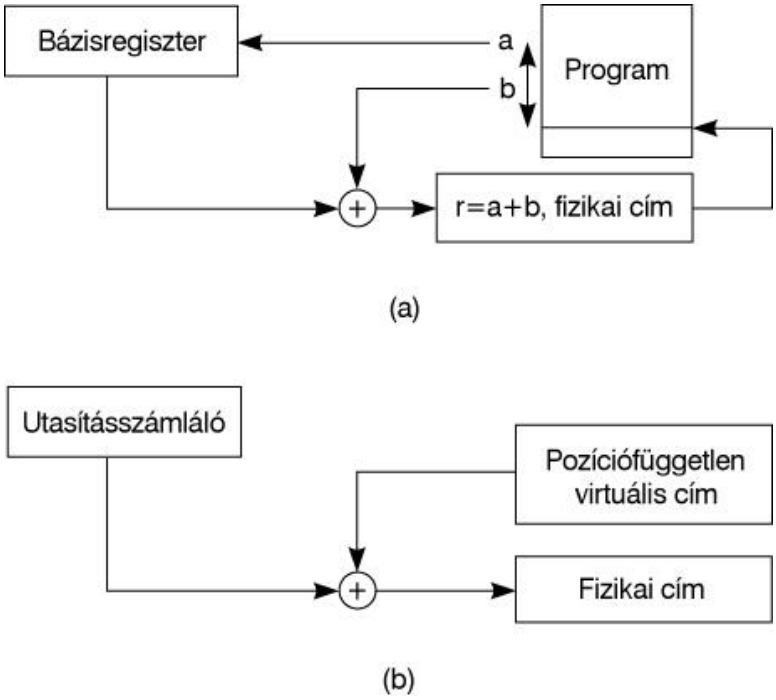
Az eddig felsorolt megoldások ún. **statikus leképezést** valósítanak meg, azaz a fizikai címek hozzárendelése a program élete során egyszer, végrehajtásuk előtt következik be, és az értékek többé nem változnak.

7.2.1. Dinamikus logikai-fizikai címleképezés

A tárgazdálkodás szempontjából igen előnyös lehet, ha a program módosítás nélkül futtatható tetszőleges szabad tárterületre töltve, és helye akár végrehajtás közben is megváltoztatható. Ilyenkor **dinamikus címleképezésről** és dinamikus áthelyezhetőségről beszélünk. Dinamikus címleképezésnél a program memóriaképe logikai címeket tartalmaz, és a konkrét fizikai címhozzárendelés csak az **utasítás-végrehajtás** során következik be (speciális hardverelemek segítségével).

A legegyszerűbb dinamikus címleképezést a **bázisrelatív címzés** valósítja meg. Ha a program valamennyi címhivatkozása bázisrelatív címzési módú, a program tetszőleges helyre betölthető, a bázisregisztert a betöltési kezdőcímre állítva a program végrehajtható (35.**(a)** ábra). Ekkor annak sincs akadálya, hogy a program végrehajtásának megszakadása után a teljes programot egy új kezdőcímre töltsük vissza, vagy másoljuk át, és így folytatódjék a végrehajtás.

A bázisrelatív címzéshez igen hasonló (akár annak alesetének is tekinthető) dinamikus címleképezési mód az utasításszámláló-relatív címzés, melynek eredményeképpen **pozíciófüggetlen kódot (PIC: Position Independent Code)** kapunk, amelyet ugyancsak végrehajthatunk tetszőleges címre töltve (35.**(b)** ábra).

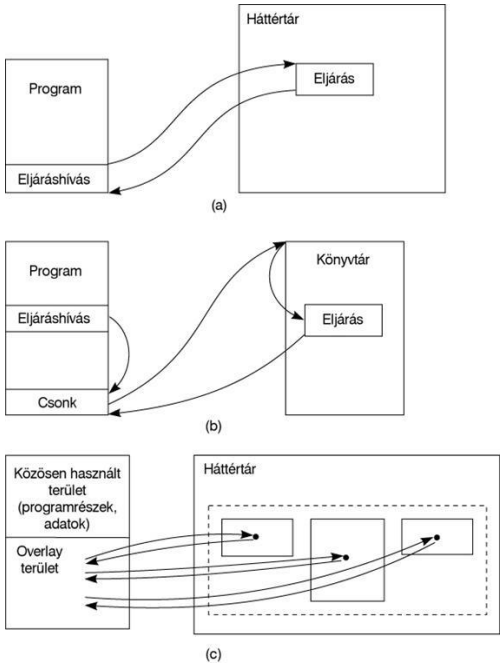


35. ábra: Dinamikus címleképezési módok (a) bázisrelatív címzés (a – a program fizikai kezdőcíme, b – virtuális (logikai) cím, r – fizikai cím), (b) utasításszámláló relatív címzés

7.3. TÁRSZERVEZÉSI MÓDSZEREK

A számítógépek történetének korai időszakában a memória viszonylag drága erőforrásnak számított. Ezért a rendszertervezők komoly erőfeszítéseket tettek a központi tár használatának optimalizálására. A mai tárak ára egyre alacsonyabb, ezért a „gazdaságos” tárkihasználás némileg mást jelent, mint korábban. (A „gazdaságosság” mögött húzódó **költségfüggvényben** a hangsúlyok eltolódtak a mai élet kihívásai által diktált követelmények – például felhasználói kényelem, nagy komplexitású, beágyazott rendszerek összetett feladatai, valósidejű működés – felé.)

Társzervezés alatt azt a módot értjük, ahogyan a központi tárat a felhasználók (folyamatok) között megosztjuk. Az „optimális” tárhasználatra való törekvés során felmerült kérdések közül az első megválaszolandó mindjárt az volt, hogy egyszerre csak egyetlen, vagy több felhasználó is használhassa-e a főmemóriát. Később az vetődött fel, hogy ha egy időben párhuzamosan több felhasználó között osztjuk meg a memóriát, akkor vajon mindegyikhez ugyanakkora területet rendeljünk-e, vagy különböző méretű részekre, **partíciókra (partition)** osszuk azt. És ezeknek a partícióknak a hossza állandó vagy pedig változó legyen-e? Eldöntendő az is, hogy a programok a (felhasználói) memórián belül bárhol, vagy csak bizonyos részeken futhatnak-e, továbbá, hogy a folyamatokhoz rendelt fizikai memória területnek egybefüggőnek kell-e lennie, vagy össze nem függő részekből is állhat.



36. ábra: Késleltetett betöltési módok (a) dinamikus betöltés (b) dinamikus könyvtárbetöltés (c) átfedő programrészek

A fenti kérdésekkel is érzékeltethető, egyre táguló lehetőségek a történeti fejlődés során nyíltak meg. Mindegyik változatra találhatunk működő rendszert, ezért ennek a pontnak további részében azt tekintjük át, hogy ezek milyen sajátosságokkal rendelkeznek, és hogyan is lehet megvalósítani őket.

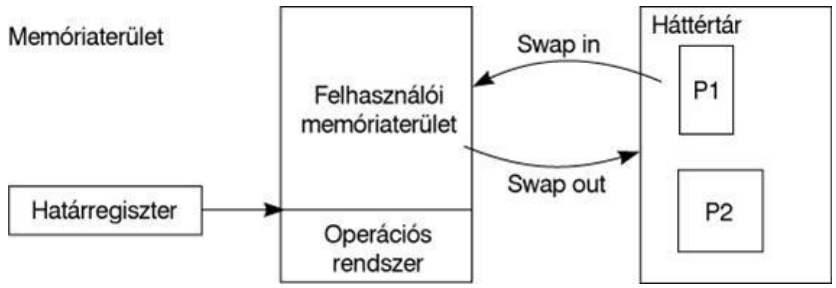
7.4. EGYPARTÍCIÓS RENDSZER

Az **egypartíciós rendszereknél** egy időben egyszerre egyetlen folyamat használhatja a központi tárat, így az operációs rendszeren – és a speciális tárterületeken, mint például a megszakításvektorok, a periféria-címtartományok – felüli folytonos címtartományon teljes egészében ez a folyamat futhat.

A betöltő a program indításakor azt az első szabad címre hozza be. Ha a program futása során az operációs rendszernek több tárra van szüksége, akkor azt vagy a program által nem használt területről – a tár másik végéről – „lopja" el, vagy a programot kell áthelyezni, ami hardvertámogatás nélkül nehézkes.

Az operációs rendszer védelmére elegendő egyetlen regiszter, amely a program legkisebb címét tartalmazza. A folyamat futása közben – felhasználói módban – a tárkezelő hardver minden egyes kiadott memóriacímet összehasonlít a határregiszter értékével, és ellenőrzi, hogy minden hivatkozás a tárolt cím felett legyen.

A folyamatok váltása a központi tár és a háttértár közötti a későbbiekben részletezett tárcsere (swapping) segítségével történhet.



37. ábra: Egypartíciós memóriaszervezés

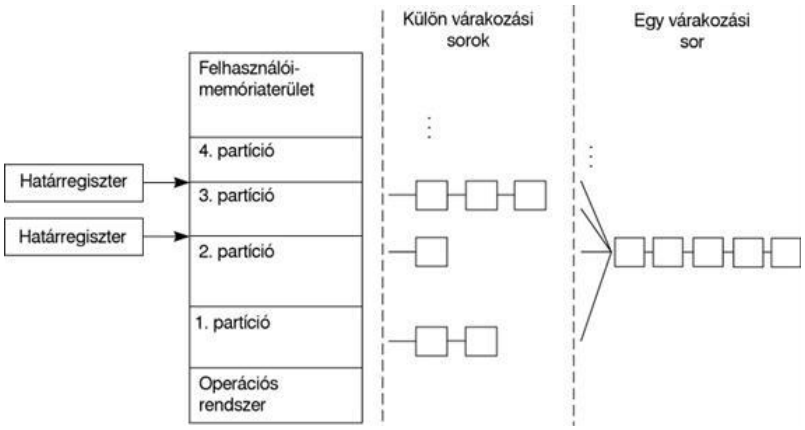
7.5. TÖBBPARTÍCIÓS RENDSZER

Bár swapping segítségével egypartíciós rendszerekben is megvalósítható valamiféle multiprogramozás, azonban ez a mágneslemezes átvitel nagy időigénye miatt jelentősen megnöveli a környezetváltási időt, így legfeljebb a lemeznél még sokkal lassabb periferiális műveletek esetén, vagy a felhasználói esélyegyenlőség biztosítása érdekében volt érdemes átkapcsolni más folyamat végrehajtására. A hatékony multiprogramozás megkövetelte, hogy egy időben egyszerre több folyamat is a tárban tartózkodjék.

7.5.1. Multiprogramozás rögzített partíciókkal

A korai rendszerekben a felhasználói tárterületet olyan különböző méretű partíciókra osztották, melyek mérete a rendszer működése során nem változhatott **(rögzített partíciók, fixed partition)**. A programok a méretüknek megfelelő partícióban futtathatók. Minden partícióban egyetlen folyamat tartózkodhat. Így a multiprogramozás fokát alapvetően a partíciók száma korlátozza.

Egy-egy folyamat befejeződése után az operációs rendszer egy következő folyamatot választ ki futásra a szabad partícióhoz. Vagy az egyetlen közös várakozási sorból (ilyenkor nyilván figyelembe kell venni, hogy a programok csak a maximális memóriaigényüknek megfelelő méretű partícióban futtathatók), vagy pedig a már eleve ehhez a partícióhoz kötött várakozási sorból (ilyenkor a hosszú távú ütemező a belépő folyamatokat közvetlenül a különböző partíciókhoz tartozó várakozási sorokba sorolja) valamilyen – általában a legjobb illeszkedést kereső – stratégia szerint (38. ábra).



38. ábra: Többpartíciós memória szervezés

Ez a tárkezelési mód nagyon egyszerű, ugyanakkor nagyon merev, így végeredményben rossz tárkihasználást biztosít. A folyamatok nem használják ki teljesen a partícióban rendelkezésükre álló tárterületet, minden partícióban a benne futó folyamathoz rendelt, ám kihasználatlan memóriaterület, „lyuk" marad. Ezt hívjuk **belső tördelődésnek (internal fragmentation)**.

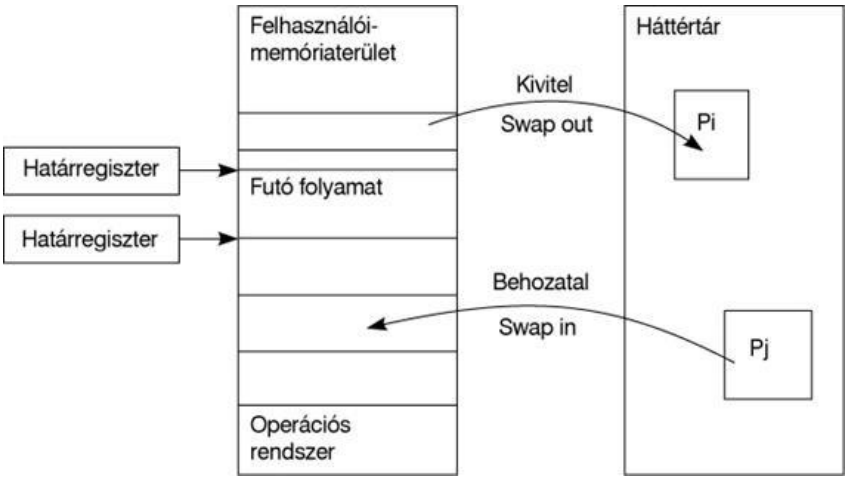
7.6. TÁRCSERE

A folyamatoknak a háttértárról a memóriába való bevitelét, illetve a memóriából a háttértárra való kivitelét a **tárcsere (swapping)** technika segítségével oldhatjuk meg. Ennek során az operációs rendszer egy-egy folyamat teljes

tárterületét a háttértárra másolja, így szabadítva fel a területet más folyamatok számára. Ehhez természetesen az operációs rendszernek pontosan ismernie kell a folyamatok aktuális tárigényét.

A tárcsere a perifériás átvitel miatt időigényes – egyszerre nagy tárterületet kell mozgatni –, így jóval hosszabb időt vesz igénybe, mint egy szokásos környezetváltás. A tárcserével töltött idő jelentősen megnövelheti a futás idejét, ezért mindenképpen célszerű a tárcsere idejét viszonylag alacsonyan tartani az effektív futási időkhöz, valamint lehetőség szerint minimalizálni kell a tárcserék számát és optimalizálni magát a műveletet.

Az ütemezőnek célszerű a tárban levő futásra kész folyamatok közül – ha van ilyen – választani. A tárterület elmentésénél pedig figyelhetünk arra is, hogy egy változatlan, a háttértáron meglevő tartományt nem kell újra kiírni. További gyorsítást tesz lehetővé az **átlapolt tárcsere (overlapped swapping)**, vagyis amikor az éppen futó folyamat végrehajtásával párhuzamosan történik egy másik folyamat kivitele, illetve egy harmadik behozatala (3.15. ábra).



39. ábra: Átlapolt tárcsere

Ha szükség van további memóriaterületre, akkor el kell dönteni, melyik folyamat legyen az áldozat. Ehhez figyelembe vehetjük a folyamatok állapotát, prioritását, futási, illetve várakozási idejét stb. Behozatalnál pedig azt kell megválaszolni, hogy a háttértáron levők közül mikor és kit hozzunk be, szintén a fenti szempontok esetleges figyelembevételével. Mindkét esetben figyelnünk kell azonban arra, hogy a folyamatokat feleslegesen ne pakolgassuk, valamint, hogy a választási algoritmus ne vezessen éhezésre.

Statikus logikai-fizikai címleképezésnél egy további megkötést jelent, hogy a háttértárra kivitt folyamatokat ugyanarra a memóriaterületre kell behozni, amelyet korábban elfoglaltak. Dinamikus – futás közbeni – címtranszformáció alkalmazása esetén a folyamatok minden további nélkül betölthetők az előzőtől eltérő memóriaterületre.

8. AZ OPERÁCIÓS RENDSZER KAPCSOLATA A GÉPI SZINT ALATTI ARCHITEKTÚRÁVAL

8.1. VIRTUÁLIS TÁRKEZELÉS

Az előző részben különböző memóriakezelési stratégiákat vizsgáltunk. Mindegyik módszernek az volt az alapvető célja, hogy lehetőleg kellően sok folyamatot tudjunk a memóriában tartani egyszerre, és így megfelelő szintű multiprogramozást tudjunk biztosítani. Azonban a fenti technikák alapvetően megkövetelték, illetve törekedtek arra, hogy a programok végrehajtásuk előtt minél teljesebb terjedelmükben a memóriába kerüljenek. A késleltetett, illetve overlay betöltések, tárcserék, a felhasználók és a programfejlesztési segédeszközök szintjén tették lehetővé a tártakarékos futtatást, általános megoldást nem kínáltak.

Ezzel szemben a **virtuális tárkezelés (virtual memory management)** – a lap-, illetve szegmensszervezésre épülően – olyan szervezési elvek és az operációs rendszer olyan algoritmusainak összességét takarja, mely **megengedi és biztosítja, hogy a rendszer folyamataihoz tartozó logikai címtartományoknak csak egy – a folyamat futásához éppen szükséges – része legyen a központi tárban**, **de ennek ellenére bármelyik folyamat szabadon hivatkozhasson bármilyen, tartományában szereplő logikai címre.**

8.1.1. A működés alapjai

A korábbi algoritmusok tárgyalása során láttuk, hogy nem kerülhető meg az, hogy a végrehajtandó utasítások a fizikai memóriában legyenek. Első megközelítésként a teljes logikai címtartományt a fizikai memóriába olvasták. Az átfedő programrészek alkalmazása és a dinamikus betöltés valamelyest lazított a fenti követelményen, azonban e technikák sok óvatosságot és erőfeszítést követeltek a programozótól. A virtuális tárkezelés ezzel szemben a programozó elől is rejtett megoldást kínál, amelyet az operációs rendszer nyújt, és amely teljes szabadságot biztosít akár a fizikai memória méretét sokszorosan meghaladó logikai (virtuális) címtartományok használatában.

A virtuális tárkezelés előzményeit tekintve különböző programok vizsgálata során bebizonyosodott, hogy **nem szükséges**, hogy a programok teljes logikai

memóriaterülete a központi tárban legyen, mert a programok nem használják ki a teljes címtartományukat:

- tartalmaznak ritkán futó kódrészleteket (például a hibakezelő rutinok),
- a statikus adatszerkezetek általában túlméretezettek (például vektorok, tömbök, szimbólumtáblák definiálásánál a ténylegesen szükségesnél sokkal nagyobb területeket foglalhatunk le),
- a program különböző részei időben egymástól elkülönülten működhetnek (ezt már az overlay technika is kihasználta),
- a programok az összetartozó részek címtartományát is időben egyenetlenül használják, azaz az időben egymáshoz közeli utasítások és adatok általában a térben is egymáshoz közel helyezkednek el **(lokalitás)**.

A korábbi vizsgálatokból az is világossá vált, hogy **nem is célszerű**, hogy a programok teljes logikai memóriaterülete a központi tárban legyen, hiszen

- ekkor a futtatható programok méretét nem korlátozza a központi memória nagysága, azaz a ténylegesen meglevő tárterületnél nagyobb tárigényű (hosszabb) programokat is futtathatunk,
- az egyes folyamatok tárigényének csökkenésével a memóriában tartott folyamatok száma növelhető, azaz növelhető a multiprogramozás foka, és ezzel javítható a CPU-kihasználtság és átbocsátóképesség anélkül, hogy a körülfordulási idő, vagy a válaszidő növekednék,
- a programok betöltéséhez, illetve a folyamatok háttértárba mentéséhez kevesebb perifériás műveletre van szükség, azaz a betöltés/kivitel gyorsabb lesz.

Jól látható tehát, hogy mind a rendszer, mind pedig a felhasználó szempontjából előnyös, ha a folyamatok tárterületének csak egy részét tartjuk egyidejűleg a memóriában.

Amikor egy folyamat érvénytelen – azaz nem a valós memóriában levő – címre hivatkozik, a címképző hardver hibamegszakítást okoz, amelyet az operációs rendszer kezel, és behozza a háttértárról a szükséges blokkot. Ennek logikai lépései a következők:

- Az operációs rendszer megszakítást kiszolgáló programrésze kapja meg a vezérlést, amely
  - elmenti a folyamat környezetét,
  - elágazik a megfelelő kiszolgáló rutinra,
  - eldönti, hogy a megszakítást programhiba (például kicímzés) vagy logikailag érvényes, de nem betöltött blokkra való hivatkozás okozta-e.

- Ez utóbbi esetben
- Az operációs rendszer behozza a kívánt blokkot a központi tárba:
  - a blokknak helyet keres a tárban; ha nincs szabad terület, fel kell szabadítania egy megfelelő méretű címtartományt, ennek tartalmát esetleg a háttértárba mentve,
  - beolvassa a kívánt blokkot.
- Az operációs rendszer átadja a vezérlést a folyamatnak, amely ismételten végrehajtja, vagy folytatja a megszakított utasítást.

A virtuális tárkezelés megvalósítása teljesen új feladatok elé állítja az operációs rendszert, amelynek megoldásához megfelelő hardvertámogatás szükséges. Legelőször is biztosítani kell, hogy az érvénytelen címre való hivatkozás megszakítást okozzon. Ezenfelül a megszakított utasítások újraindíthatók, vagy folytathatók kell hogy legyenek. A folytathatóság megoldása ritkább, mert utasítások végrehajtása közben fennálló állapotot kellene elmenteni, ami meglehetősen bonyolulttá tenné a processzorokat.

A virtuális tárkezelést az IBM/360 számítógépcsalád megjelenésével kezdték elterjedten alkalmazni, és elmondhatjuk, hogy ma már a fejlettebb mikroprocesszorok mindegyike használ valamifajta virtuális tárkezelést. A korszerű hardverek szinte kivétel nélkül lap- vagy kombinált szervezésűek, ezért a virtuális tárkezelés is csaknem mindig lapok mozgatásán alapul, így a továbbiakban mi is lapszervezésű rendszerekkel foglalkozunk.

A virtuális tárkezelés lapcsere algoritmusait egyéb, a blokkokhoz tartozó – később részletesen tárgyalt – bitek is támogatják.

A különböző rendszerek minősítését alapvetően befolyásolja a működési sebességük. A folyamatok futásának sebességét pedig döntően a tárhoz férés effektív ideje határozza meg. Ez virtuális tárkezelés esetén a megfelelő előfordulási valószínűséggel súlyozott memória hozzáférés, illetve (laphiba esetén) lapbehozatali időből számítható. Ha **p** jelöli a laphiba előfordulás valószínűségét, akkor

Effektív hozzáférési idő = (1-p) * Memória hozzáférési idő + p * Laphiba idő.

A laphiba kiszolgálási ideje több mint öt nagyságrenddel is nagyobb lehet a valós memória hozzáférési időnél (a lemezműveletek időigénye a mai rendszerekben is 10 msec körül van), ezért **p**-nek nagyon kicsinek (10^{-6}–10^{-8}) kell lennie ahhoz, hogy a folyamatok futása ne lassuljon le túlságosan.

A virtuális tárkezelés tárgyalásánál három alapvető kérdést kell megválaszolnunk:

- Melyik lapot hozzuk be a tárba (betöltendő lap kiválasztása)?

- Ha nincs szabad hely a memóriában, akkor melyik lapot cseréljük le (lapcsere, replacement strategy)?
- Hogyan gazdálkodjunk a fizikai tárral, azaz melyik folyamat számára hány lapot biztosítsunk?

A továbbiakban e három kérdést vizsgáljuk meg kicsit részletesebben.

8.1.2. Betöltendő lap kiválasztása (fetch-strategy)

Alapvetően két stratégiát követhetünk.

Igény szerinti lapozás (demand paging) esetén csak a laphibánál hivatkozott lapot hozzuk be a tárba. A módszer előnye, hogy a betöltendő lap kiválasztása nagyon egyszerű, továbbá, hogy csak a biztosan szükséges lapok kerülnek be a tárba. Ugyanakkor az új lapokra való hivatkozás mindig laphibát okoz, ami lassítja a működést.

Előretekintő lapozásnál (anticipatory paging) ezzel szemben az operációs rendszer megpróbálja „kitalálni”, hogy a folyamatnak a közeljövőben mely lapokra lesz szüksége, és azokat szabad idejében – amikor a lapcseréhez használt háttértár szabad – betölti.

Ha a jóslás helyes volt, akkor az előre behozott lapok miatt jelentősen lecsökken a laphibák száma, és ezzel a folyamat futási sebessége nagyban felgyorsul. Ha a döntés hibás volt, akkor felesleges lapok foglalják el a tárat.

Mivel manapság a központi tár ára drasztikusan csökken és ezzel párhuzamosan mérete egyre nő, ezért az előretekintő lapozás egyre népszerűbb, hiszen a hibás döntés – azaz a felesleges tárfoglalás – „ára” egyre kisebb.

8.1.3. Gazdálkodás a fizikai tárral

A folyamatok lapigénye

A fizikai tárgazdálkodás legalapvetőbb kérdése, hogy hány lapot adjunk az egyes folyamatoknak. A folyamatok szempontjából nézve az a jó, ha minél több lapjuk van a tárban, hiszen nagy valószínűséggel annál kevesebb laphibát okoznak. Túl kevés lap esetén állandóan laphiba lép fel. Ha a rendszerben átlagosan nem tud befejeződni egy laphiba kiszolgálása, mielőtt egy újabb laphiba fellép, a folyamatok felgyűlnek a mágneslemez várakozási sorában, és a CPU-nak nem lesz futtatható folyamata, **CPU-tétlenség** alakul ki. A rendszer teljesítménye leromlik. (Fennáll annak a veszélye is, hogy a hosszú távú ütemező ezt a B/K-intenzív folyamatok túlsúlyaként értékeli, és újabb folyamatokat enged be, ami természetesen tovább rontja a helyzetet.) A gyakori laphibák által okozott teljesítménycsökkenést **vergődésnek (thrashing)** nevezzük.

A rendszer szempontjából nézve, ha egy folyamatnak sok lapot adunk, azt jelenti, hogy kevesebb folyamatot tudunk a tárban tartani, így alacsonyabb lesz a multiprogramozás foka és ezzel várhatóan a CPU-kihasználtság is. A görbe kezdeti szakaszán kevés folyamat van a rendszerben. Minél kevesebben vannak, annál nagyobb annak a valószínűsége, hogy mindegyikük várakozik, a CPU pedig tétlen. A folyamatok számának a növekedésével a CPU-kihasználás aszimptotikusan közelít az adminisztrációs (például környezetváltási) veszteségekkel csökkentett 100%-os maximumhoz. A folyamatok számának további növekedésekor azonban – mivel az egy folyamatra jutó tárterület csökken – belép a tárkezelés hatása és kialakul a vergődés, a CPU-kihasználás pedig a folyamatok számának további növelésével meredeken leesik. El kell kerülni, hogy a rendszerben ilyen üzemállapot alakulhasson ki, azonban az optimumot lehetőleg meg szeretnénk közelíteni. Fontos tehát, hogy meg tudjuk becsülni, milyen **laphiba-gyakoriság (PFF: Page Fault Frequency)** mellett marad még a rendszer egyensúlyban.

Könnyen beláthatjuk, hogy a rendszer egészét tekintve a CPU akkor nem válik tétlenné, ha átlagosan egy laphiba kezelése közben nem következik be újabb laphiba (ha meggondoljuk, ugyanerre a következtetésre jutottunk az effektív memóriahozzáférési idő kiszámítása kapcsán).

A teljes rendszerre vonatkozó egyensúlyi feltételt vonatkoztathatjuk az egyes folyamatokra is, azaz előírhatjuk, hogy két laphiba közötti átlagos futási idejük haladja meg a laphiba átlagos kiszolgálási idejét. Ha ez minden folyamatra teljesül, akkor az egész rendszer is egyensúlyban lesz.

Visszatérve az eredeti kérdésre, célszerű tehát egy folyamatnak annyi lapot adni, amennyi szükséges az egyensúlyhoz, azaz ahány lapra hivatkozik a laphiba kiszolgálás átlagos ideje alatt (ugyanakkor nem sokkal többet, mert akkor leromlik a multiprogramozás foka). Ezt az értéket a **munkahalmaz méretének (working-set size)** nevezzük**.**

8.2. FÁJLRENDSZEREK

A klasszikus operációs rendszerek felhasználók által leginkább használt absztrakciója a fájl, állomány.

> **Fájlnak a felhasználó, vagy a rendszer szempontjából összetartozó információk perzisztens, a létrehozó programot „túlélő” gyűjteményét nevezzük.**

A fájlokat a rendszer többnyire valamilyen háttértáron tárolja, amely tartalmát megőrzi még akkor is, amikor a rendszer áramellátását kikapcsolták.

Egy operációs rendszer általában nagyszámú fájlt tárol, kezel. Ezeket egymástól meg kell különböztetni, el kell különíteni. Az egyes állományokat tipikusan egy hozzájuk rendelt név azonosítja, a nagyszámú fájlt könnyebb kezelhetőség érdekében könyvtárakba csoportosítják. Az állományok az operációs rendszer felhasználói számára közös, logikai erőforráshalmazt képeznek, az ebből fakadó erőforrás-gazdálkodási feladatok, mint például a hozzáférések ütemezése, koordinálása, szabályozása, korlátozása is a fájlkezelő rendszerek feladata.

A fájlabsztrakció a rendszer használója, programozója számára kényelmes hozzáférést biztosít a fájl tartalmához, elrejtve a tárolás és kezelés konkrét részleteit. A fájlkezelő rendszer tipikusan a következő magas szintű feladatokat valósítja meg:

- hozzáférést biztosít az állomány részeihez, megvalósítja az állományok tartalmának átvitelét a háttértár és a központi tár, illetve a háttértár és egyéb perifériák – például nyomtató – között,
- műveleteket biztosít az egyes fájlokon, illetve a fájlokat összefoglaló könyvtárakon,
- osztott hozzáférések esetén koordinálja, időzíti az egyes folyamatok fájlhasználatát,
- szabályozza a felhasználóknak, vagy azok programjainak a fájlokhoz hozzáférést, különválasztva kezeli az egyes felhasználók állományait és könyvtárait, megakadályozza, hogy a fájlokon illetéktelen felhasználók műveleteket végezhessenek, esetlegesen védi a fájlokban tárolt információkat az illetéktelen olvasások ellen, kódolva, titkosítva annak tartalmát,
- gyakori a fájlokban tárolt információk sérülése, hardver- vagy kezelési hiba miatt bekövetkező elvesztése elleni védelem, a fájlok időszakonkénti megbízhatóbb háttértárra mentése is.

A fájlok kezelését különböző, a konkrét hardvertől egyre távolodó, a logikai szervezéshez egyre közeledő, egymásra épülő programrétegek valósítják meg.

A perifériás eszközhöz legközelebb az ún. készülékkezelő programok (**device driver,** meghajtó) állnak, amelyek közvetlenül a berendezéseket vezérlik. Ezen réteg feladata az adatok központi tár és a periféria közötti átvitelének kezdeményezése, vezérlése, lebonyolítása. A jelenlegi hardverrendszerekben tipikusan megszakításosan vezérelt perifériák megszakításai is ide futnak be.

A meghajtókra épül az elemi átviteli műveletek lebonyolítására szolgáló réteg. Itt történik meg a fájlok tartalmának a periféria által használt címzési rendszerre való leképzése, a periféria által megkövetelt, avagy csak a rendszer telje-

sítményét javító, az átvitelek számát csökkentő blokkok képzése. Mivel a háttértár elérése még a korszerű, nagy sebességű mágneslemezes tárak esetén is több nagyságrenddel lassabb, mint a központi tár elérése, e réteg gyakori feladata gyorsító tárak (cache) képzése, a gyakran használt blokkok központi tárból történő újrafelhasználásának megvalósítása is. Egyidejű hozzáférés kérelmek esetén a réteg sorba állíthatja a kéréseket, optimalizálva a fizikai erőforrás kihasználását.

Az elemi átvitelekre épül az állományszervezés rétege, amely nyilvántartja a háttértár szabad és lefoglalt területeit, gondoskodik az egyes fájlokhoz tartozó blokkok logikai összefűzéséről, a dinamikus helygazdálkodásról. Ide tartozik új fájlok létrehozásakor, vagy a fájlokban tárolt információ bővülésekor további szabad helyek lefoglalása, illetve állományok törlésekor a korábban lefoglalt területek felszabadítása. Tipikusan ez a réteg biztosít eljárásokat a rendszer programozói számára az állomány tartalmának eléréséhez, módosításához.

A logikai állományszervezés rétege ismeri és kezeli a nyilvántartásokat, az állomány neve alapján megtalálja annak helyét. Ide tartozik az állományok felhasználónkénti hozzáférésének szabályozása, de gyakran a konkurens elérések időbeli ütemezése, szinkronizálása is e réteg feladata.

8.2.1. Az állományok tárolása a lemezen

Az operációs rendszer a lemezterületet a hardverhez hasonlóan nagyobb egységekben, blokkokban kezeli. Ez az adatmozgatás, lefoglalás, felszabadítás alapegysége, ennél kisebb területekkel az operációs rendszer nem foglalkozik (néhány kivételtől eltekintve). Egy logikai – operációs rendszer által képzett – blokk tipikusan 1 vagy néhány hardverblokkot – szektort – tartalmaz. A blokkok méretének meghatározását szervezési, tárolási és hatékonysági szempontok befolyásolják. Minél nagyobb blokkokat használunk, annál kevesebb az egységnyi információ átviteléhez szükséges többlet művelet, viszont a nagy blokkokat ki nem töltő információ miatt feleslegesen foglalunk mind a háttértárban, mind a központi tárban területet, jelentős belső tördelődési veszteség lép fel.

A blokkok méretét gyakorta meghatározza az adatszerkezetekben a blokkokat azonosító címek számára fenntartott hely mérete. Főleg kisebb, kezdetlegesebb rendszerekben a blokkok címzésére viszonylag kevés helyet, mondjuk 16 bitet tartottak fenn. Eleinte a 16 biten ábrázolható különböző értékek bőven elegendőnek bizonyultak az összes, a lemezen előforduló fizikai szektor egyenkénti megcímzésére. Azonban a lemezegységek kapacitásának ugrásszerű növekedésével ez a 16 bit hamarosan kevésnek bizonyult, fizikai szektorokat össze kellett vonni az operációs rendszer által egyben kezelt logikai blokkokká, hiszen ezek száma nem haladhatta meg a 64 K-t. Ezért aztán a logikai blokkok mérete

egyre növekedett, így gyakran előfordulhatott, hogy egy tipikusan néhány száz bájtból álló szöveges fájl a lemezen több megabájtnyi területet kényszerült elfoglalni. A csapdából természetesen egyszerűnek tűnik a kiút, nagyobb címtartományt kell fenntartani a blokkok számára. Ám ez nemcsak a blokkok adminisztrációja számára fenntartott területeket növeli, elvéve a helyet az „értékes" adatoktól, de a háttértárak rohamos fejlődésével a ma még bőségesen elegendőnek tűnő címtartomány 1-2 éven belül szűknek bizonyulhat. Az egyes rendszerváltozatok kompatibilitásának fenntartása is nehezíti a probléma megoldását, hiszen a rendszer belső adatszerkezeteit érintő módosításokat nem könnyű zökkenőmentesen lebonyolítani. Másik lehetőség a háttértár particionálása, a fizikailag nagyszámú blokkot tartalmazó lemezegységek logikailag kisebb egységekre, ún. kötetekre „darabolása", ám a szétszabdalt tárterület kezelése is problémákat okoz.

A fájlszervezés rétegének a lemez minden egyes blokkját nyilván kell tartania, tudni kell, mely blokkok tartoznak az egyes fájlokhoz, melyek az éppen szabad, fel nem használt blokkok. Ismernie kell a különböző alacsony (például szabad területek), illetve magas szintű (például könyvtárak) nyilvántartások tárolására fenntartott blokkokat is, és gazdálkodnia kell velük.

Alacsony szinten kétfajta adatszerkezetet különíthetünk el, a lemez szabad blokkjainak, illetve az egyes fájlokhoz tartozó blokkoknak a leírását. Mivel mind információtartalmában, mind kezelésében más jellegű nyilvántartással van dolgunk, e célokra más és más adatszerkezetek honosodtak meg.

A szabad blokkok nyilvántartására a különböző adatszerkezetek használatosak:

- **Bittérképes ábrázolás.** A lemez mindegyik logikai blokkjához egy bitben tárolható, hogy szabad-e. Ezekből a bitekből képzett vektort a lemez kijelölt helyén tároljuk. A bitvektoros szervezés esetén egymás melletti szabad blokkok kiválasztása nagyon egyszerű, de a vektor kezelése csak akkor hatékony, ha a teljes vektort a központi tárban tudjuk tartani.
- **Láncolt listás ábrázolás.** A szabad blokkokból „lecsípett" bájtokban egy másik szabad blokk sorszámát, címét tárolhatjuk. Így a szabad blokkokat mintegy láncra fűzzük, csak a lánc elejét kell „kézben tartani" – egy jól definiált helyen tárolni –, innen kiindulva az összes szabad blokk elérhető. Mivel a szabad blokkban értékes információ nincs, ezért a lecsípett bájtok nem okoznak problémát. Az ábrázolás viszont nem hatékony, a lista bejárása lassú, mert több lemezműveletet igényel.
- **Szabad helyek csoportjainak listája.** A fenti ábrázolás teljesítménye könnyen javítható. Ahelyett, hogy minden egyes blokkból csak egyetlen címnyi területet csippentenénk le, kihasználhatjuk a teljes blokkot arra,

hogy más szabad blokkok címét tároljuk. Mivel a szabad blokkok száma dinamikusan változhat, az egyik címet a fenti láncolt lista képzésére használjuk, ám a többi **n-1** címnyi terület 1-1 szabad blokkot jelöl ki.

- **Egybefüggő szabad területek nyilvántartása.** A szabad blokkok egyesével történő tárolása helyett egy táblázatban az egymás mellett lévő szabad blokkokról az első sorszámát és a terület hosszát tároljuk. A módszer akkor előnyös, ha a szabad területek átlagos hossza (jóval) nagyobb egynél, valamint ha az allokálásnál egymás melletti szabad területeket akarunk kiválasztani. Ha a táblázat elemei a kezdő blokk szerint rendezettek, az egymás melletti szabad területek összevonása egyszerű.

Az egyes állományokhoz tartozó blokkok tárolására, lefoglalására a következő általános megoldások alakultak ki:

- **Folytonos terület lefoglalása.** A fájlhoz tartozó információt egymás melletti szabad blokkokban tároljuk. A rendszernek csupán az első blokk sorszámát, valamint a blokkok számát kell tárolni az állományt leíró adatok között.

A folytonos terület foglalásának előnye, hogy a tárolt információnak mind soros, mind közvetlen elérése egyszerű, gyors. Napjaink rendszerében ez a tárolási forma általában akkor használt, ha előre tudjuk a szükséges blokkok számát, illetve igény az összes blokk egyidejű, gyors tárba, illetve tárból való mozgatása. Tipikus eset a virtuális tárkezelés esetén a tárcsere (swapping) megvalósítása.

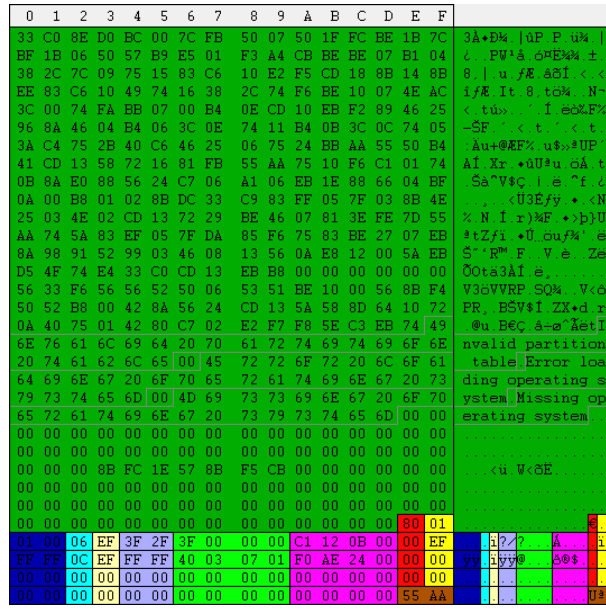
- **Láncolt tárolás.** A szükséges blokkokat egyesével allokáljuk, minden blokkban fenntartva egy helyet a következő blokk sorszámának. Általában a rendszer az állományleíróban az első és esetleg az utolsó blokk sorszámát tárolja.

A tárolási mód nagy előnye, hogy ez egy dinamikus adatszerkezet, az állomány mérete szükség szerint tetszőlegesen növekedhet, határt csak a szabad blokkok száma szab. A fenti módszer külső tördelődéstől is mentes, a szabad területeket nem kell tömöríteni.

Hátrány, hogy a láncolt tárolás csak a tárolt információ soros elérést támogatja, a közvetlen eléréshez a listát az elejéről végig kell járni.

Külön problémát jelent, hogy a lánc szervezéséhez szükséges címet a blokkok hasznos területéből kell lecsípni. Így egy-egy blokk központi tárba másolásakor ezeket a darabokat ki kell hagyni. Ezen a problémán segít a láncolt tárolás egy változata, ahol a láncokat az állományoktól elkülönülten, egy **fájl allokációs táblában (FAT)** tároljuk. A táblában a lemez minden blokkjához tartozik egy bejegyzés (pointer), amelyik – amennyiben a blokk fájlhoz tartozik – a fájl kö-

vetkező blokkjára mutat. Az egyes állományokhoz csak a lánc első blokkjának sorszámát kell tárolni az állományleíróban, a következő blokkok a FAT-ból megtalálhatók. Ezt az allokációs táblát egyidejűleg a szabad blokkok tárolására is fel lehet használni. A tárolt információ közvetlen elérése is egyszerűbb, nem kell az egyes blokkokat végigjárni, elég e FAT-ban „lépkedni”. A FAT, vagy annak jelentős része a tárban tartható, így ez a keresés gyors. A sérülékenység ellen a FAT kettőzött tárolásával védekezhetünk. Ezt a megoldást alkalmazzák az MS-DOS-rendszerek.



40. ábra: Master Boot record diagram

- **Indexelt tárolás.** Az indexelt ábrázolás alapötlete, hogy az állományhoz tartozó blokkokat leíró címeket külön adatterületre, az indextáblába gyűjtjük. Az állományleíróban az indextáblát tartalmazó blokk(ok) címét kell tárolni.

A módszer előnyei, hogy a közvetlen hozzáférés megvalósítása egyszerű, hiszen az indextábla segítségével minden blokk közvetlenül megtalálható. Lehetőség van „lyukas” állományok tárolására, azaz olyan állományokra, amelyek nem minden blokkja tartalmaz információt. Ebben az ábrázolási formában a nem használt területekhez nem kell lemezblokkot lefoglalni, azok helyét az indextáblában üresen hagyjuk.

Hátránya, hogy a blokkok címeinek tárolásához akkor is legalább egy teljes (index)blokkot kell lefoglalni, ha az állomány csak kevés blokkot tartalmaz. A

tárolási mód pazarló. Mivel az indextábla mérete előre nem ismert, ezért meg kell oldani, hogy a táblázat is dinamikusan növekedhessen, például az index-blokkokat láncba fűzzük, hasonlóan a szabad helyek csoportjainak tárolásához, vagy többszintű indextáblákat használunk.

- **Kombinált módszerek.** Egyes rendszerek az állomány hozzáférési módja, vagy mérete függvényében más és más módszert alkalmazhatnak. Például kis állományokat folytonosan, míg nagyokat indexelten tárolhatunk. Vagy soros hozzáférés igénye esetén láncolt, közvetlen hozzáférés esetén összefüggő blokkok tárolását választhatjuk.

Mind a szabad blokkok, mind az állományok tárolásának fent említett módszerei többé-kevésbé érzékenyek a lemezen tárolt nyilvántartások, adatszerkezetek sérülésére. Például egy láncolt blokkos szabadhely-nyilvántartásnál egy blokk sérülése a lánc megszakadásához vezethet, ezzel a lánc végén álló blokkok elvesznek a rendszer számára. Még nagyobb problémát jelenthet az egyes állományokhoz tartozó blokkok leírásának elvesztése, megsérülése.

Ezt a problémát az operációs rendszerek általában redundáns adatszerkezetekkel igyekeznek enyhíteni. Például használhatunk kétirányú láncokat, ha a lánc mindkét vége még megvan, egy-egy blokk kiesését könnyen elviselhetjük, mindkét irányból elindulva megkeressük a „szakadást” és összefűzzük a láncot.

A redundáns adatszerkezetek árát a csökkenő értékes tárolási kapacitással és a lassabb kezelő algoritmusokkal fizetjük meg. Egy rendszer tervezőjének gondosan mérlegelnie kell tehát az egyes tényezőket, például azt, hogy a perifériás eszköz várhatóan milyen arányú meghibásodásokat produkál, mekkora katasztrófát okoz számunkra egyes szabad blokkok, vagy az állományok blokkjainak elvesztése. A sérülések hatásai kiküszöbölésének egyszerű eszköze a háttértár tartalmának rendszeres, gyakori mentése.

A fájlkezelő rendszernek – vagy az erre szorosan épülő segédprogramoknak – tipikus feladata a különböző lemezen tárolt nyilvántartások konzisztenciájának ellenőrzése, az esetleges sérülések kijavítása. A javítás ügyesen megválasztott adatszerkezetek esetén, bonyolult – nagy lemezegységek esetén lassú – ellenőrző algoritmusokat használva gyakran automatikusan történhet, ám néha a rendszergazda kézi beavatkozására is szükség lehet. Legrosszabb esetben a rendszergazdának egyes blokkok tartalmát „szemrevételeznie” kell, megtippelni, hogy vajon ez szabad blokk lehetett-e, vagy valamelyik – és főleg melyik – állományhoz tartozhatott. Ez a tevékenység szöveges információk esetén még úgy-ahogy sikerrel kecsegtet, de bináris állományok esetén nagyon nehezen használható.

8.3. FÁJLRENDSZEREK

A fájlrendszer feladata, hogy az eltárolandó fájlokat és könyvtárakat a winchester egy partícióján a megfelelő helyen elhelyezze, garantálja annak visszaolvashatóságát, valamint a változásokat adminisztrálja. Tehát a fájlrendszer funkciója kettős: egyrészt tárolja egy adott partíción lévő) adatsávok (fájlok) helyét, másrészt kezeli az ezekhez kapcsolódó metaadatokat.

Minden, a fájlrendszerben tárolt adathoz (egy adott fájl fizikai elhelyezkedése a lemezen) tartozik egy metaadat-bejegyzés is, ez tartalmazza a fájl vagy könyvtár nevét, létrehozás, módosítás dátumát, a tulajdonos adatait, valamint a hozzáférési jogosultságokat. Fontos tudni, hogy a könyvtárak fizikailag nem jelennek meg a lemezen, azok csak a fájlrendszer adatbázisában lévő bejegyzésként vannak tárolva (tehát amennyiben egy lemezről elveszítjük a fájlrendszert leíró adatbázist, úgy a nyers adatok visszanyerhetőek, de nem fogjuk tudni, hogy melyik fájl hol kezdődött és hol van vége és nem fogijuk tudni rekonstruálni a könyvtárrendszert.

Látható, hogy a fájlrendszer helyes működése létfontosságú a tárolt adatok használhatóságának szempontjából, éppen ezért a fájlrendszer adatbázisa a lemezen általában több példányban, sokszorosítva lesz eltárolva, csökkentve a megsérülés valószínűségét.

Mivel a számítógépeknek számos különböző feladatra használhatóak, ezért az adatok tároláskor is különböző igények léphetnek fel mind teljesítmény, mind biztonság tekintetében. Ezen igényekre kielégítésére rengeteg fájlrendszer megvalósítás létezik

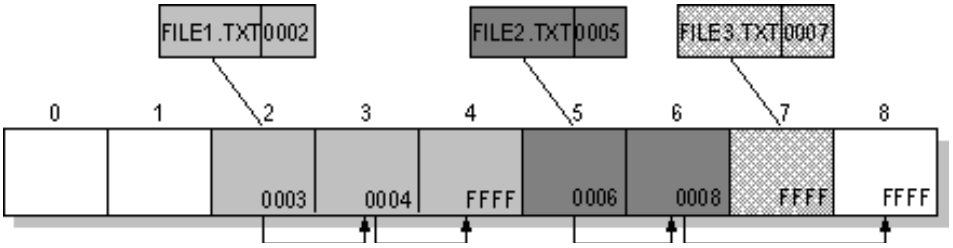
Fájlrendszereket általában az operációs rendszerrel együtt szokták szállítani, például MS Windows operációs rendszerhez két fájlrendszer érhető el: a FAT 16-32, illetve az NTFS. Míg Linux alatt a legelterjedtebb az ext3 fájlrendszer, de számos egyedi igényéket kielégítő megvalósítás érhető el. Továbbá léteznek hálózati fájlrendszerek is, amelyek az operációs rendszer szempontjából átlagos partíciónak tűnnek, de fizikailag az adatok nem az adat számítógép merevlemezén tárolódnak.

A fájlrendszerek egy fontos tulajdonsága, a legkisebb foglalási egység, azaz ha létrehozunk egy üres fájlt, akkor fizikailag mekkora tárterület foglalódik le ennek a fájlnak a winchesteren.

8.3.1. FAT-fájlrendszer

A File Allocation Table (FAT) fájlrendszert a Microsoft fejlesztette és a Windows XP operációs rendszerig használták. Régen ez volt a Windows operá-

ciós rendszerek által kizárólagosan használt fájlrendszer. A FAT-fájlrendszer klasztereket tart számon, és a különböző FAT-verziók főként abban különböznek, hogy hány biten tárolják a klaszterek sorszámait (FAT12, FAT16, FAT32). Ezen bitszám határozza meg, hogy összesen mekkora lehet egy FAT-partíció mérete. Az operációs rendszert is tartalmazó FAT-partíciók tartalmaznak egy boot sectort is, ez a szektor kerül beolvasásra memóriába az operációs rendszer bootolásának első lépéseként. Az alábbiakban részletesen megnézzük, hogyan tárolja a fájlokat a FAT-fájlrendszer. A 41. ábrán látható a partíció egy darabja: a tárolt fájlok, valamint a hozzájuk tartozó allokálós táblabeli bejegyzések. Az allokációs tábla – többek között – tartalmazza a fájl nevét és annak a klaszterek a számát, ahol a fájl kezdődik. Minden klaszter végén található egy cím amely a fájl többi darabját tároló klaszterre mutat vagy egy 0xFFFF jelzés, ami azt jelenti, hogy ez az utolsó klaszter, amiben a fájl részlete volt eltárolva. Fontos észrevennünk, hogy a rendszer nem követeli meg, hogy egy nagyobb méretű fájl egymás utáni klaszterek sorozataként legyen a lemezen: az operációs rendszer utasításaitól függően akár rengeteg, a lemez különböző pontjain elhelyezkedő klaszterbe is kerülhet a fájl egy-egy darabja. A FAT fájl rendszerekben a legnagyobb tárolható fájlméret 4 GB.



41. ábra: Egy FAT partíció darabja Forrás: http://www.ntłs.com/images/recover-FAT-structure.gif

8.3.2. NTFS

Az NTFS (vagy New Technology File System) a Microsoft Windows NT és utódainak (Windows 2000, Windows XP, Windows Server 2003) szabványos fájlrendszere. A korábbi Windows 95, 98, 98SE és ME nem képesek natív módon olvasni az NTFS-fájlrendszert, bár léteznek programok erre a célra is.

Az NTFS a Microsoft korábbi FAT-fájlrendszerét váltotta le, melyet az MS-DOS és a korábbi Windows verziók esetén használtak. Az NTFS több újdonsággal rendelkezik a FAT-fájlrendszerrel szemben, mint például a metaadatok támogatása, fejlettebb adatstruktúrák támogatása a sebesség, a megbízhatóság és lemezterület-felhasználás érdekében, valamint már rendelkezik hozzáférés-

védelmi listával és megtalálható benne a naplózás is. A fő hátránya a korlátozott támogatottsága a nem-Microsoft operációs rendszerek oldaláról, mivel a pontos specifikáció a Microsoft szabadalma.

Az NTFS-nek három verziója létezik:

1. v1.2 – NT 3.51, NT 4
2. v3.0 – Windows 2000
3. v3.1 – Windows XP, Windows Server 2003, Windows Vista

Az NTFS-en belül minden fájlokkal kapcsolatos információt (fájlnév, létrehozás dátuma, hozzáférési jogok, tartalom) metaadatként tárolnak. Ez az elegáns, bár absztrakt megközelítés lehetővé tette újabb fájlrendszer-funkciók létrehozását a Windows NT fejlesztése során – egy érdekes példa az Active Directory által használt indexelő mezők hozzáadása. A fájlnevek Unicode (UTF-16) formátumban vannak tárolva, azzal a változtatással, hogy a fájlrendszer nem ellenőrzi az UTF-16 szerinti szabványosságot.

Az NTFS B+-fákat használ a fájlrendszer adat tárolására. Bár bonyolult megvalósítani, rövidebb hozzáférési időt biztosít bizonyos esetekben. Egy fájlrendszer naplót használnak magának a fájlrendszer integritásának (de nem az egyes fájloknak) a biztosítására. Az NTFS-t használó rendszerek biztonságosabbak, ami egy kiemelten fontos követelmény a Windows NT-k korábbi verzióinak instabil mivolta miatt.

A megvalósítás részletei nem nyilvánosak, így külső gyártóknak nagyon nehéz NTFS-t kezelő eszközöket előállítani.

8.3.3. exFAT fájlrendszer

A Microsoft a Windows Vista SP1 megjelenésével egy új fájlrendszert vezetett be. Az Extended File Allocation Table (exFAT) a FAT32 fájlrendszer utódja. Az exFAT bizonyos helyzetekben jobb választás, mint az NTFS.

FAT32 hátrányok az exFAT-tal szemben

Alapértelmezés szerint csak maximum 32 GB-os lemezeket formázhatunk FAT32-re. Külső szoftverek ezt áthidalhatják, ám nem ajánlott.

A maximális fájlméret 4 GB.

A töredezettség-mentesítés és szabad terület kalkulációk nagyon erőforrás-igényesek lehetnek a méretesebb lemezeken.

Egy FAT32 mappa csak bizonyos számú almappát és fájlt tartalmazhat (összesen 65536 bejegyzést). Egy-egy almappa vagy fájl több bejegyzést is lefoglalhat magának (függ a fájlnevének hosszától).

exFAT előnyök

A fájlméret korlátozásban az 512 terabájt az ajánlott, de 64 zettabájt az elméleti maximum.

A formázás- és mappatartalom-limitálást eltörölték.

HPFS mintájára, az exFAT is free space bitmap táblát használ a szabad területek villámgyors lefoglalásához, így gyorsítva a fájlműveleteket.

HTFS mintájára támogatja az Access Control List (ACL)-t, melynek segítségével különböző fájlrendszer jogosultságok használhatók. (SP1 még nem támogatja.)

Az így formázott lemezeket a Mac OS X is támogatja írni és olvasni is tudja ezeket.

exFAT hátrányok

A Windows Vista SP1 előtti operációs rendszerek nem támogatják (kiegészítés: a Windows XP rendszerhez letölthető egy fájl, amely alkalmassá teszi a rendszert az exFAT használatára).

ReadyBoost inkompatibilis. A Windows 7 már támogatja.

8.3.4. SWAP-fájlrendszer

Egy adott pillanatban nem minden programot használunk, amit elindítottunk a számítógépünkön, Eleve az adott programnak sem használjuk minden részét. Ebből kifolyólag a nem aktív programokat, valamint programrészeket az operációs rendszer nem a viszonylag szökős memóriában tartja, hanem a winchesteren, az úgynevezett swap (csere) területen. A tárolás olyan formátumban történik, hogy a kért programrészleteket, ill. adatokat szükség esetén külön keresés-konvertálás nélkül a memóriába tudja visszatölteni. (Például: amikor a tálcára leteszünk egy programot és sokáig nem foglalkozunk vele, majd később elővesszük, azt tapasztaljuk, hogy elég lassan reagál a kérésünkre, de a winchester nagy tempóban dolgozik, ekkor kerülnek vissza a swap (csere) területről a memóriába az adott programba tartozó adatok és program részletek). MS Windows alatt a fájlrendszerben egy reguláris fájlként jelenik meg a swap terület, amit Pagefile-nak hív a rendszer. Ez komoly hátrány, hiszen mint reguláris

fájl, ez is ki van téve a töredezettségnek, hasonlóan a többi, fájlrendszerbéli fájlhoz.

GNU/Linux rendszereknél a swap tárterület egy linux-s-,vap típusú fájlrendszerrel rendelkező külön partíció. Ez a megoldás természetesen nem szenved az előbb említett, a fájlrendszerek töredezettségéből fakadó hátránytól.

9. A SZÁMÍTÓGÉP RENDSZEREK ÁLTALÁNOS FELÉPÍTÉSE

9.1. ALAPFOGALMAK

A számítógép különböző eszközök együttese, amelyeket nap, mint nap láthatunk a számítógépes munka közben, de a számítógép házában egyes eszközök rejtve maradnak a szemünk elől. A számítógép napjainkra (2010) használati tárgy lett, és két dolog miatt szeretjük igazán: hatalmas méretű adatsereg feldolgozását képes rövid idő alatt elvégezni, ezen túl a szórakozás egyik kedvelt eszköze. De mit jelent pontosan ez a kifejezés? A fogalmat két megközelítésben ismertetjük. A számítógép működési mechanizmusára épülő fogalom:

- **A számítógép olyan eszközrendszer, amely bizonyos, jól meghatározott utasítások sorozatát egyértelműen képes végrehajtani. Ezek a műveletek matematikai logikai számítások, de ide értendő az adatok ki- és bevitele is.**

A második megközelítésben az ember-számítógép viszonyt emeljük ki:

- **A számítógép egy információátalakító eszköz, amely a bemeneti eszközein közölt információkat átalakítva, a kimeneti eszközein jeleníti meg valamilyen, az ember számára érzékelhető formában.**

A számítógép felosztható további elemekre, az azt alkotó, fizikailag megfogható részek összességére, amit összefoglaló néven hardvernek neveznek.

- **A számítógépet alkotó elektronikus-, elekromechanikus- és mechanikus berendezések összességét nevezzük hardvernek.**

Önt, amikor számítógéppel dolgozik, felhasználónak nevezzük. A felhasználó szívesen venné, ha a számítógép bekapcsolás után máris munkára fogható lenne, de a valóságban nem ez történik. A berendezés önmagában használhatatlan, működtetéséhez szükséges egy vagy több program, amelyek segítségével a felhasználó a számítógépnek kiadhatja a parancsait.

- **Szoftvernek nevezzük azon szellemi termékek összességét, amelyekkel egy adott számítógépet működtetni lehet, konkrétan a programokat, a hozzá tartozó adatokat, leírásokat.**

A programozó szakemberek alkotta programok nélkül a számítógépek használhatatlanok lennének. A program – mint ismeretes – utasítások rendezett, véges halmazából áll. A számítógép működésének alapja az utasítások precíz értelmezése és végrehajtása. A programok használatának leírását egy dokumentációban (felhasználói kézikönyv) teszik közzé, az átlagos felhasználó a programot csupán használja, sohasem készíti.

A fentieket összegzéseként felsoroljuk a számítógépés munka kiemelt erőforrásait: hardver, szoftver és ezek együttesét működtetni tudó ember.

A számítógépek birtoklása hosszú időn át a nagy kutatóbázisok, katonai létesítmények, később nagyobb cégek, illetve egyetemek kiváltsága volt az ún. mikroszámítógépek megjelenéséig. A hetvenes években – szerényebb teljesítmény mellett – eljutottak az átlagemberhez is. Ezt a változatot házi számítógépnek (Home Computer) hívtak.

Pár évvel később, 1981-ben megjelent az IBM cég mikroszámítógépe, amit most már a kínai Lenovo cég gyárt. A személyi számítógép, vagyis a PC (Personal Computer) a mai napig fejlődik, és „beköltözött" az iskolákba, családi otthonokba is. Több PC-gyártó cég létezik, így az nem biztos, hogy az a program, ami egy adott PC-n működik az működni fog egy másikon is. Ha mégis működik egy másik számítógépen is a program, akkor a két számítógép kompatibilistásáról beszélünk. Lehet kompatibilis két szoftver is, hiszen amikor elkészül egy program újabb változata, akkor elvárjuk, hogy a régivel készült dokumentumaink kezelhetőek legyenek az új változattal is.

> **Két számítógép vagy program, vagy számítógépes rendszer kompatibilis egymással, ha kicserélhető az egyik a másikkal az eredeti funkció teljes megőrzése mellett.**

A számítógép alatt ezután egy mai, 2012-ben beszerezhető PC-t értünk, amelyre sokáig IBM-kompatibilis PC néven hivatkoztak.

9.2. SZÁMÍTÓGÉP-ARCHITEKTÚRÁK

Egy számítógépet alapvetően három szinten vizsgálhatunk.

A felhasználó általában csak az operációs rendszer szolgáltatásain keresztül látja a számítógépet, melyek többé-kevésbé elfedik előle a hardvert (célszerű úgy felfogni a dolgot, hogy egy vagy több magas szintű nyelvet lát, és az operációs rendszerrel egy parancsnyelven keresztül kommunikál).

Az operációs rendszer (beleértve a fordítóprogramokat is) a felhasználói programot és a hardvert adatként tekinti és dolgozza fel. Ezt a tárolt programú vezérlés elve biztosítja. Ez az elv többszintű hierarchia kialakítását teszi lehetővé (például mikroprogramozás). Itt utalunk arra az általános rendszertervezői

elvre, hogy egy feladat megoldásánál a megoldás algoritmusa a fontos, a tényleges hardver/szoftver arány megválasztása a konkrét körülményektől függ (ár, megbízhatóság, sebesség stb.).

A hardvertervező elemi áramkörökkel, azokból felépített funkcionális egységekkel, azok megvalósításával foglalkozik.

Az architektúrális vizsgálat szintje a fenti két szint közé esik: a rendszer funkcionális moduljaival, azok kapcsolatával foglalkozik az operációs rendszer alatti, de az áramköri részletek feletti szinten. A számítógép-architektúra tehát az illesztési felület a hardver- és a szoftvertervezők között.

Ismert, hogy megfelelően nagy tárolókapacitás esetén bármilyen számítógép képes szimulálni bármilyen másik létező vagy jövőbeni számítógépet. Ez nem jelenti azt, hogy minden számítástechnikai probléma megoldható lenne elfogadható időn belül bármelyik számítógéppel. Így a különböző számítástechnikai feladatcsoportokra – pontosabban: a különböző információfeldolgozási folyamattípusokra – különböző számítógéptípusokat alakítanak ki.

Minden egyes számítógép-architektúra az információfeldolgozási folyamat egy meghatározott elméleti modelljének konkrét megvalósítása. Mivel a különböző számítógépek gyakran eltérő modelleket testesítenek meg, azokat nehéz egymással összehasonlítani. Néhány példa a különböző elméleti modelltípusokra:

- adatáramlás a számítógép regiszterei között;
- adatáramlás a számítógép beviteli és kiviteli egységein keresztül;
- információfeldolgozási szolgáltató hálózat;
- erőforrások hozzárendelése versengő igényekhez.

A továbbiakban architektúrán a számítógép (vagy számítógéprendszer) térbeli, időbeli (beleértve az események sorrendezési módját is) és funkcionális elrendezését értjük.

A számítógépgyártó cégek általában nem egyes gépeket, hanem kompatibilis gépekből álló családokat terveznek, melyek architektúrája azonos (vagy nagyon hasonló), és egy sebesség-, kapacitás- és ártartományt fednek le. Az azonos architektúra itt a gépcsalád tagjaira azonos utasításkészletet, az egyes tagok között átvihető perifériális és háttértár egységeket jelent. A gépcsalád a felhasználó számára az igény növekedésével a könnyű bővíthetőséget biztosítja, míg a gyártó többek között a szoftver fejlesztési terheit csökkenti (mivel ugyanazt a szoftvert a család több gépén változatlanul futtathatjuk). A gépcsalád fogalma azonban általában azt is magában foglalja, hogy a gyártó minimalizálni akarja az áttérés nehézségeit saját régebbi gépcsaládjáról az új családra. Ennek

egyik módja az új család architektúrájának a régebbivel felülről kompatibilissé tétele, másik módja a régi család jellemzőinek emulációja az új családban mikroprogramozás segítségével.

9.3. INPUT-PROCESS-OUTPUT MODELL

A számítógépes rendszerek működése három fontos lépésre bontható fel:

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

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

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

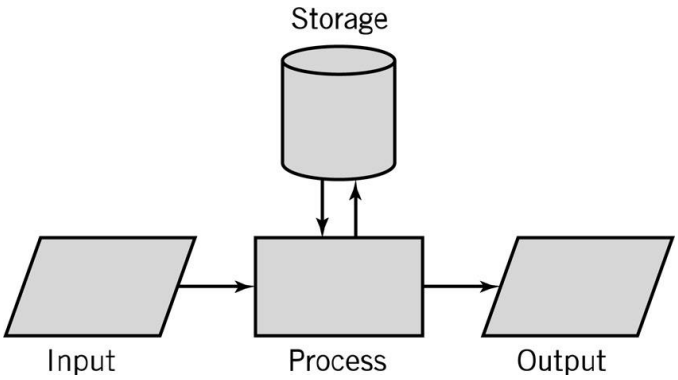
Általános célú számítógépes környezetben a leggyakrabban használt eszközök az egyes lépések végrehajtására a következők:

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

A számítógépes rendszerek működéséhez, a feldolgozás lépés megvalósításához a számítógépbe bevitt adatok átmeneti és hosszú távú tárolására van szükség. A számítógépben leggyakrabban használt tároló eszközök: merevlemez, optikai lemez, hajlékonylemezek, mágneskazetta.

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

Adattároló



Beolvasás (Input) Feldolgozás (Process) Kiírás (Output)

42. ábra: Input-Process-Output modell

A beolvasás, feldolgozás és kiírás lépések további részfeladatokra bonthatók, például adatállományok mozgatása, konvertálása, keresés bennük stb.

A számítógépes programok készítésekor ezeket a részfeladatokat kell továbbpontosítani, lebontani a számítógép által közvetlenül végrehajtható műveletekre.

A számítógép-programozás nem más, mint annak a számítógép által végrehajtható műveleteket tartalmazó műveletsornak a definiálása, mely a felhasználó által kívánt feladatot megvalósítja.

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

- adatbeolvasási/kiviteli utasítások, adatállomány írása, olvasása;
- aritmetikai műveletek, logikai műveletek, értékadó műveletek különböző típusú adatelemeken;
- feltételes elágazások utasításai;
- ciklusutasítások.

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

tű programozási nyelvi műveletet egyértelműen meg tudunk feleltetni egy számítógép által közvetlenül végrehajtható műveletsorozatnak.

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

A számítógép programok készítésében az ún. objektumorientált (OO) programozás lényeges változást hozott. Az egyes programok utasítássorozatait OO-környezetben már utasításblokkonként fejlesztik és tárolják.

Ennek számos előnye van, a kód-újrafelhasználás lehetősége, áttekinthetőbb programszerkezet az összetartozó adatelemek és utasításblokkok együttes kezelése miatt, vagy az egyszerűbb programkarbantartás és továbbfejlesztés lehetősége. Fontos kiemelni, hogy az OO-szemlélet lényeges változást hozott a szoftverfejlesztésben, de nem változtatott az IPO-modell érvényességén. Az OO-környezetben fejlesztett programok is hasonlóan hajtódnak végre, mint a más programozási környezetben fejlesztett társaik, a különbség a mi szempontunkból talán annyi, hogy az OO-forráskódban nehezebben követhető az egymás után végrehajtott utasítások sorozata.

9.4. SZÁMÍTÓGÉPES RENDSZEREK ÁLTALÁNOS FELÉPÍTÉSE

A különböző számítógépes rendszereket feloszthatjuk négy fontos komponensre:

1. Számítógép hardver. Ezek az elemek biztosítják az adatbevitel, adattárolás, adatfeldolgozás és adatmegjelenítés fizikai megvalósítását. Jellemzően több hardver komponensből áll egy számítógép, és hozzá tartozik az a vezérlő elektronika is, mely az elemeket külön-külön, ill. azokat közösen vezérli.
2. Számítógépes szoftver. A számítógépes szoftver utasítások formájában definiálja, hogy egy-egy szolgáltatás megvalósításához (egy adott feladat végrehajtásához) a számítógép hardver milyen műveleteket és milyen sorrendben kell, hogy végrehajtson.
3. A számítógép által tárolt és feldolgozott adatok. Különböző típusú (egész szám, lebegőpontos szám, karakteres szöveg, grafikus kép stb.)

információ ábrázolása olyan formában, melyet a számítógép közvetlenül fel tud dolgozni.

4. Számítógépes kommunikációt lehetővé tevő elemek. Ezek az elemek hardver- és szoftverrészeket is tartalmaznak. Lehetővé teszik, hogy a számítógép más számítógépekkel kapcsolatba léphessen, azokkal adatcserét folytasson. Ezeket az elemeket a többi elemtől nem az elemek jellege (szoftver, hardver stb.), hanem azok funkciója különbözteti meg a többi elemtől.

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

A számítógép architektúráját az első két komponens, vagyis a számítógéphardver és a számítógépszoftver alkotja. Amikor a számítógép architektúrájáról beszélünk, akkor ennek a két komponensnek a felépítéséről, részeiről és működéséről beszélünk. A számítógép architektúráját meghatározó komponensek közé a szoftver elemek közül csak az ún. operációs rendszerhez tartozó komponenseket soroljuk. Ezek a szoftverkomponensek működtetik a hardver elemeket, teszik lehetővé további alkalmazások futtatását. Az alkalmazói szoftverek és a számítógép által tárolt és feldolgozott adatok a számítógépes rendszerek működéséhez alapvetően szükséges elemek, nem függenek a számítógép architektúrájától.

9.5. SZÁMÍTÓGÉP HARDVERELEMEI

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

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

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

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

A CPU ALU és CU egysége a CPU interfészegységén keresztül kapcsolódik a számítógép további részeihez. Az összeköttetés jellemzően ún. adat sínen (buszon) keresztül valósul meg.

A sín (busz) a számítógépekben általánosan használt kapcsoló elem. A sín elektromos jeleket továbbító kábelek együttese a kommunikációt lehetővé tevő vezérlő logikával kiegészítve.

A sín egy vagy több számítógép komponens között teremt kapcsolatot, tesz lehetővé egy vagy kétirányú adatcserét a sínre kapcsolt egységek között.

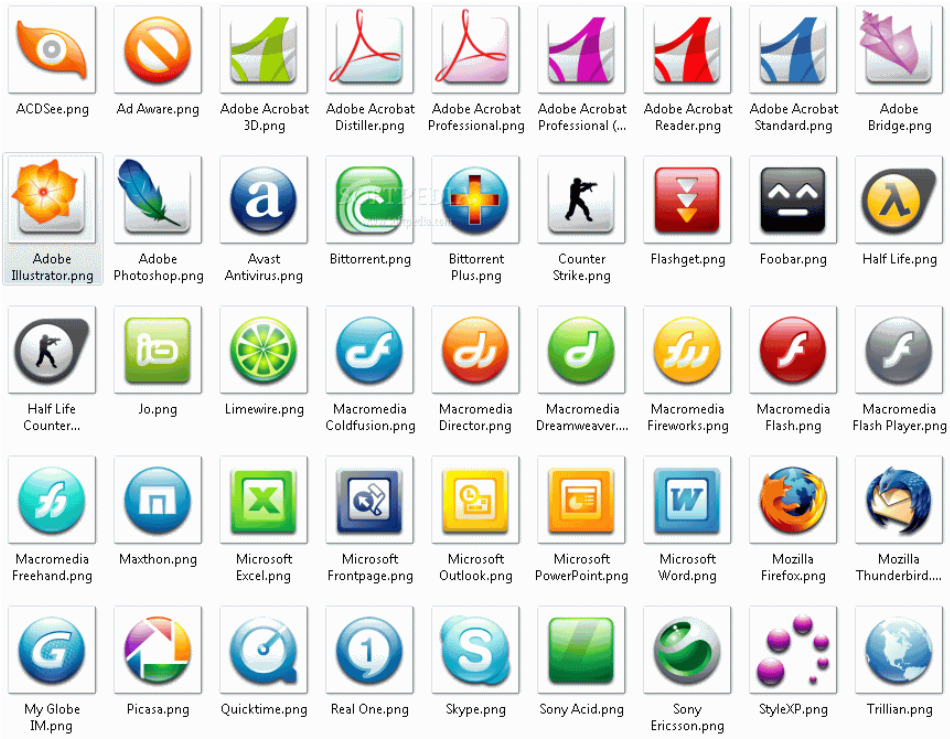
A sínen szállított jelek lehetnek adatértékek, a kommunikáció irányításához használt vezérlőjelek vagy tápfeszültség. Sín kapcsolatot használunk – mint említettük – perifériák csatlakoztatására, de például a CPU-n belüli komponensek összekapcsolására is.

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

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

lódnak a számítógép magjához, tehát a másodlagos tárolók a perifériák közé tartoznak.

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

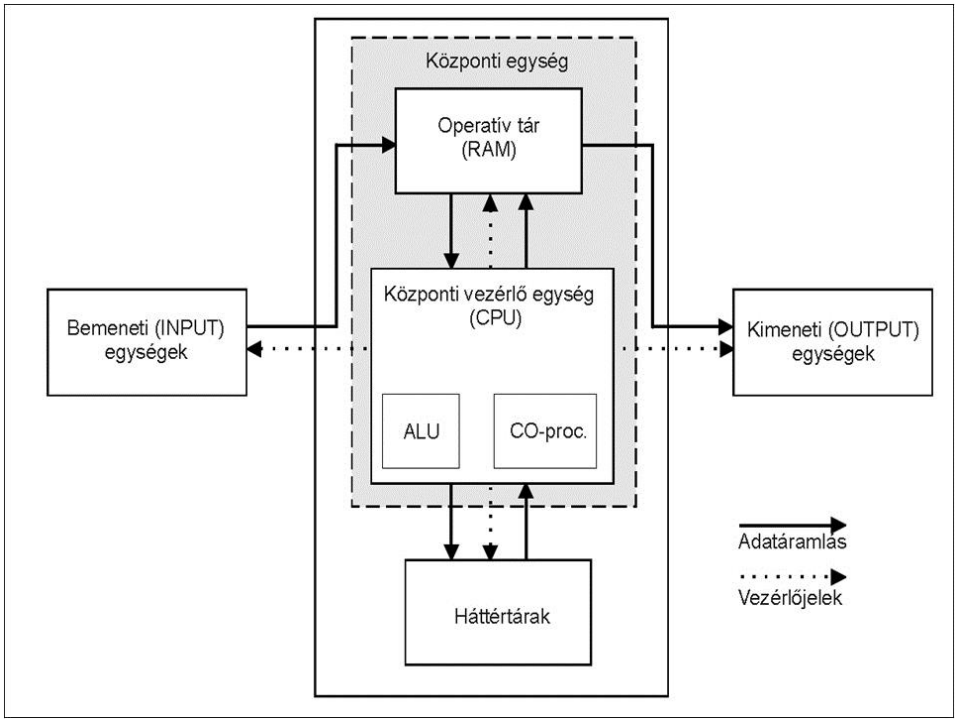


43. ábra: Számítógépek általános felépítése

9.6. SZÁMÍTÓGÉP SZOFTVER ELEMEI

A szoftver fogalmának pontos meghatározására a számítógép-tudomány számos definíciót készített. Számunkra elegendő az az egyszerű meghatározás:

- **Szoftvernek nevezzük azon szellemi termékek összességét, amelyekkel egy adott számítógépet működtetni lehet. Konkrétan a programokat, a hozzá tartozó adatokat, leírásokat.**



44. ábra: Windows 8

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

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

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

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

1. Felhasználói interfész. A felhasználói interfész segítségével a felhasználók közvetlenül tudják az operációs rendszer szolgáltatásait aktivizálni.
2. Például adatállományokat tudnak létrehozni, tartalmukat megjeleníteni, esetleg azokat módosítani, alkalmazásokat indítani, a számítógépen futó alkalmazásokat kilistázni stb.
3. Programozói interfész. A programozói interfész a programok futtatását teszi lehetővé az adott számítógépen. Az operációs rendszer tartalmazza a hardver elemek közvetlen kezelését megvalósító funkciókat.

10. AZ OPERÁCIÓS RENDSZEREK SZERKEZETE ÉS SZOLGÁLTATÁSAI

10.1. OPERÁCIÓS RENDSZEREK FELÉPÍTÉSE

Az **operációs rendszer** a számítógépes szoftver része, olyan programrendszer, amely betölti és vezérli a gépen futó programokat (alkalmazásokat), elosztja, ütemezi az erőforrásokat, kezeli a hardvert, biztosítja a felhasználó és a számítógéprendszer közötti kommunikációt.

Az operációs rendszer szolgáltatásai:

- Lemezkezelés
- Könyvtár- és fájlkezelés
- Rendszerbeállítások

Az operációs rendszer komponensei (részei):

1. Rendszermag (kernel)
2. Alkalmazói programozási interfész (API: Application Programming Interface)
3. Rendszerhéj (shell)
4. Szervizprogramok (Utility)

1. **A rendszermag:** A legfontosabb és legbonyolultabb komponens. Feladata a rendszer vezérlése a hardver lehetőség szerinti optimális kihasználása, az alkalmazói alrendszer kéréseinek kiszolgálása, a kért programok futtatása.

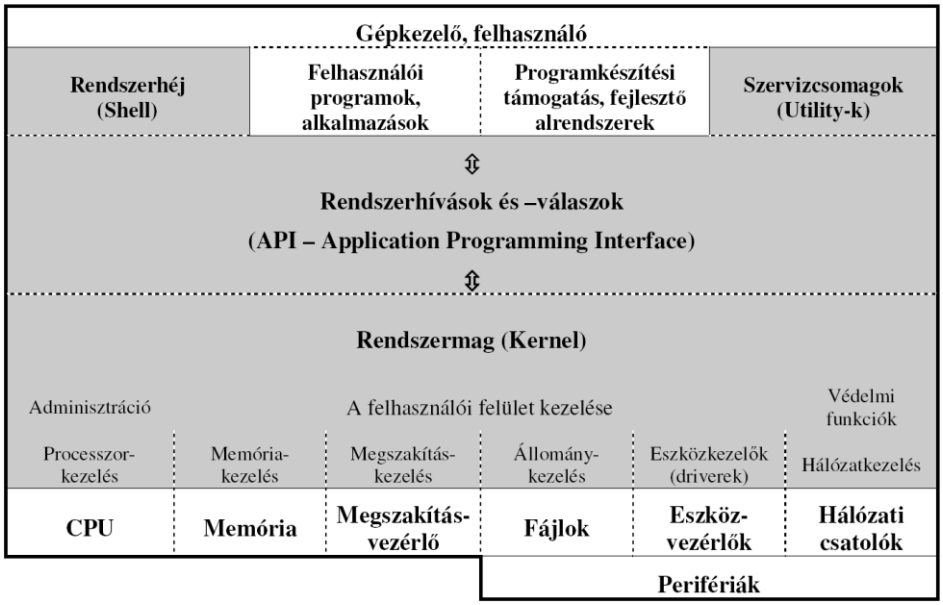
2. Az API: Egy illesztési felület, interfész a rendszermag és az alkalmazási alrendszer között. Azon szabályok összessége, amelyek megadják, hogy hogyan kell kérni a rendszermagtól szolgáltatásokat, illetve hogyan kaphatjuk meg a kérésre adott választ. Az API-t a rendszerhéj és a szervizprogramok is használják. Grafikus változatai objektumorientáltak.

3. A rendszerhéj: Feladata az operációs rendszer és a felhasználó (gépkezelő) kapcsolatának biztosítása. Grafikus vagy parancssor típusú lehet.

A grafikus típusú (GUI: Graphics User Interface) képi metaforákkal – ikonokkal – és valamilyen mutatóeszközzel (pl.: egérmutatóval), a parancssor típu-

sú pedig egy egyszerű karakteres szövegszerkesztővel segített párbeszédet biztosít. Az API-n keresztül tart kapcsolatot a rendszermaggal.

4. Szervizcsomagok: A ritkán szükséges funkciókat tartalmazza, melyeket így nem kell a rendszermagba beépíteni, és ezáltal a rendszer rugalmasságát lehet velük növelni. A szervizcsomagok a rendszermagtól függetlenek, több változatban is elkészülhetnek.



45. ábra:

10.2. RENDSZERMAG (KERNEL)

Rendszermag (angolul *kernel):* az operációs rendszer alapja (magja), amely felelős a hardver erőforrásainak kezeléséért, beleértve a memóriát és a processzort is.

A többfeladatos rendszerekben – ahol egyszerre több program is futhat – a kernel felelős azért, hogy megszabja, hogy melyik program és mennyi ideig használhatja a hardver egy adott részét (ezen módszer neve a multiplexálás). A hardver elemek használata gyakran bonyolult programrészeket igényel, ezért ezt a feladatot gyakran egységes, absztrakt hardverelérést biztosító részekkel támogatja. Ezek a részek elrejtik a bonyolult módszereket és egy tiszta, egyszerű felületet biztosítanak, amivel megkönnyítik a hardverelemeket használó programozók munkáját.

Egy számítógép működéséhez nem feltétlenül szükséges operációs rendszer és annak magja. Az egyes programok közvetlenül betölthetőek és használhatóak a „csupasz vason”, feltéve, hogy a programozó vállalja azt, hogy mindent közvetlenül, operációs rendszeri segítség nélkül fog kezelni. A kezdeti számítógépek esetén ez volt a normális működési mód: minden egyes új program elindításához a gépet újra kellett indítani. Az idő előrehaladtával apró segédprogramok, rutinok állandósulni kezdtek, azokat több programhoz is használták, és kialakultak azok a szokásos programrészek, melyeket újraindítás után újra használni szerettek volna, mint például egyes betöltő (indító, *boot)* programok vagy hibakeresők. Ezekből alakultak ki a kezdeti operációs rendszerek.

A kerneleknek négy fő kategóriáját különböztethetjük meg (eltekintve azon programkörnyezetektől, melyek kernel nélkül futnak):

- a *monolitikus kernelek* gazdag és hatékony absztrakciókat biztosítanak az alattuk található hardverelemekhez;
- a mikrokernelek egy kisméretű alapkészletet biztosítanak a hardver kezeléséhez, és számos alkalmazással – amelyeket „szervereknek” nevezünk – biztosítják a további, részletesebb funkcionalitást;
- a hibrid vagy módosított mikrokernelek hasonlóak a színtiszta mikrokernelekhez de több, részletesebb kódot tartalmaznak a kernelmagban, hogy nagyobb sebességet érjenek el;
- az exokernelek (vagy rendszer rutinkönyvtárak) nem biztosítanak absztrakciókat vagy állandó rendszermagot, hanem egy programokban használható rutinkönyvtárból állnak, ami a hardver közvetlen vagy közvetett elérését biztosítja.

10.2.1. A rendszermag feladatai, rendszervezérlési feladatok

Adminisztráció

Az adminisztrációs feladatok részben a felhasználók számára lehetnek fontosak, másrészt az optimális működés beállítása és a hibajavítás segítése céljából vannak beépítve.

Védelmi funkciók

A védelmi funkciók sokrétűek: védeni kell a rendszert a futó felhasználó programoktól, a felhasználói programokat egymástól, az adatokat a sérüléstől és az illetéktelen hozzáféréstől. Ez különösen fontos feladat egy hálózatos környezetben.

A CPU kezelése (processzorkezelés)

Az operációs rendszer felügyelete alatt a gépen futó programot folyamatnak (process) nevezzük. A felhasználói folyamatokat feladatnak vagy taszknak hívjuk, megkülönböztetve őket a rendszer folyamataitól. A processzor folyamatokhoz rendelése, használatának üzemezése a rendszermag ütemezőnek (sheduler) nevezett komponensének a feladata, ami maga is egy rendszerfolyamat.

Sokszor növelni lehet a rendszer hatékonyságát úgy, hogy a folyamatokat is több, önállóan is végrehajtható szálra (thread) bontjuk. A többszálú (multithreading) operációs rendszer ütemezője a folyamatok helyett tehát folyamatszálakat kezel. A többszálú folyamatok is egyetlen szállal indulnak a betöltés után. Ezt elsődleges (primary) szálnak nevezzük. A szálak futás közben létrehozhatnak további szálakat is.

Az operatív tár (operatív memória) kezelése

A rendszermag memóriakezelő részeinek a többfeladatos rendszerekben biztosítaniuk kell a folyamatok számára szükséges tárterületet, illetve meg kell oldaniuk az ilyenkor jelentkező védelmi problémákat. A tárkezelési filozófiák hosszabb fejlődés után mára szinte egységesen virtuális tárakat használnak. A virtuális memóriakezelés a futtatható programok méretét az operatív memóriánál nagyságrendekkel nagyobb, a merevlemezes háttértár kapacitásával összevethető méretűre terjeszti ki.

A legtöbb operációs rendszer esetén a rendelkezésre álló főtár (memória) nem elegendően nagy ahhoz, hogy az összes aktív folyamat egyidejűleg a memóriában tartózkodjon. Ezért gondoskodni kell arról, hogy a megfelelő időben az inaktív folyamatokat a főtárból eltávolítsuk, és helyükbe újakat hozhassunk. Az átmenetileg kipakolt folyamatokat háttértárolókon helyezzük el. Azt sem nehéz elképzelni, hogy egy nagyméretű program egyedül sem fér el a memóriában, és ezért kisebb részekre kell bontani. A rendszer feladata tehát, hogy megoldja a programrészek cseréjét a kellő pillanatban, illetve kezelje azt az esetet, amikor a mozgatni kívánt memóriaterület részben vagy egészben nem található a főtárban. Az áthelyezés automatikus, nincs szükség arra, hogy a program speciális parancsokat tartalmazzon a végrehajtásához.

A megszakítások feldolgozása

A megszakítások kiszolgálásakor a futó programok által használt eszközök igényeit kell kielégíteni. Megszakításra lehet szükség például egy, a billentyűzetről küldött karakter fogadásához, vagy a belső időzítő óra kérésére.

Az eszközkezelők

Az eszközkezelők (illesztő programok, driverek) egy részét az eszköz vagy a periféria gyártója szokta elkészíteni, és külön telepítést (installálást, setupot) igényelnek. A telepítés során egy feljegyzés készül róluk egy rendszerállományba (a config.sys-be vagy a registry-be), amelynek segítségével rendszertöltéskor a kernelbe épülhetnek. Mivel ilyenkor a korábbi rendszermag megváltozhat, az operációs rendszert a telepítés után újra kell tölteni (a gépet újra kell indítani). Ma már, a „csatlakoztasd és használd" (Plug and Play, rövidítve PnP), módszert alkalmazzák, ami automatikussá teszi az illesztett periféria felismerését és a megfelelő illesztőprogram kernelbe építését. A hibátlan működést ekkor az operációs rendszer gyártója garantálja. Az eszközkezelőkre további, a felhasználó számára a periféria kezelését, tesztelési és beállítási lehetőségeket biztosító programok épülhetnek. Ezek már alkalmazásnak tekinthetőek.

Állománykezelés

Az operációs rendszer feladata az adatok gyors elérhetőségének biztosítása, a megbízható adatmegőrzés, a tároló hellyel való takarékos gazdálkodás és a felhasználói igényekhez alkalmazkodó adatszerkezetek kialakítása. A fájlrendszerek egy része az operációs rendszerre, más része az adathordozó típusára jellemző.

Hálózatkezelés

A hálózati kommunikáció az operációs rendszerektől független szabványokra épül. Az ISO OSI-albizottsága dolgozta ki azt a jól strukturált hétrétegű modellt, amit a létező protokollok többé-kevésbé követnek. A hálózati kapcsolatokat kezelő operációs rendszerek tervezésekor figyelembe kell venni az alábbiakat: bizonyos funkciókat szét lehet osztani a hálózati erőforrások között, valamint a hálózatokkal együtt járó adatvédelmi, biztonsági követelmények egy része csak kernel szinten biztosítható megnyugtatóan.

10.3. A RENDSZERHÉJ

A rendszerhéj a rendszerbetöltés utolsó lépéseként kerül az operatív tárba, és az első olyan program lesz, amelynek segítségével a felhasználó kapcsolatba léphet az operációs rendszerrel. Az állományok és könyvtárak kezelésében is közreműködik. A rendszerhéj akár több változatban is elkészíthető és cserélhető. Hiba, ha a rendszerhéjat magával az operációs rendszerrel azonosítjuk. A rendszert a rendszermag határozza meg!

Ma kétféle, széleskörűen elterjedt változata létezik a rendszerhéjnak, a **parancssor** és a GUI-t használó, **grafikus**.

A parancssoros rendszerhéj egy szöveges felületen tart kapcsolatot a felhasználóval. Jellegzetes eleme alapestben a prompt, amely után villogó szövegkurzor jelzi, hogy az operációs rendszer fogadja a felhasználó parancsait. A felhasználói parancs az ENTER billentyű leütéséig szerkeszthető. Azután a rendszer értelmezi, lebontja rendszerhívásokra és megkísérli a végrehajtását, majd az eredményről értesíti a felhasználót szöveges formában a képernyőn. Megjelenhetnek akár menük és használhatunk akár egérkurzort is.

A grafikus rendszerhéj és a GUI elemeit **WIMP** mozaikszóval foghatjuk össze: **W**indows, **I**cons, **M**enus, **P**ointing devices, azaz Ablakok, Ikonok, Menük és Mutató eszközök.

A grafikus felületeket kezelő programok és programrendszerek objektumorientáltak.

Objektumaik üzenetekkel folytathatnak egymással párbeszédet. A felhasználó, az alkalmazói és rendszerprogramok is üzeneteket használhatnak az információcseréhez. A felhasználó eseménnyel válthat ki üzeneteket. Ilyen lehet az egér elmozdítása.

10.3.1. Üzenetkezelés

A folyamatok szintjén az üzenet eljárás- vagy függvényhívás, az üzenetre adott válasz pedig az eljárás vagy függvény végrehajtása lesz. Egy folyamatszál üzenhet más szálnak, de saját magának is, esetleg egyszerre többnek is (broadcast). A felhasználót tekintve az üzenetcsere eszközei a billentyűzet, a képernyő, az egér, esetleg a hangkártya és a mikrofon lehet. A párbeszéd a GUI ablakain keresztül folyik.

A GUI komponenseit tehetjük egyetlen számítógépre, amit hálózatba kötve ilyenkor **grafikus munkaállomás**nak nevezünk. Elképzelhető, hogy a grafikus megjelenítő rendszer egy távoli számítógépen futó alkalmazói programot szolgál ki. A felhasználói oldal gépét ilyenkor **grafikus terminál**nak hívjuk. A grafikus munkaállomás inkább a Windows-ra, a grafikus terminál pedig XWindows néven a Unixra és a Linuxra jellemző.

Az egérrel vagy a billentyűzetről a felhasználó üzeneteket küld, melyek a rendszer üzenetsorába kerülnek, majd egy ütemező szétosztja őket a folyamatszálak között. Amelyik szál kezeli az üzenet ablakát, az kapja meg feldolgozásra is. A szálak üzenetkezelője ciklusban, folyamatosan figyeli a saját üzenetsorát. Ha nem üres, kivesz egy várakozót a sorból, és az ablakkezelőnek továbbítja feldolgozásra. Ha nincs üzenet a sorban, akkor az 1. vagy a 2. pont szerint folytatódik a szál futása.

1. A folyamatszálat a rendszer „elaltatja”. Az alvó szálat valamilyen hardveresemény, egér-, billentyűüzenet, vagy egy másik szál által küldött üzenet ébresztheti fel.
2. A szál nem alszik el. Ha vannak a háttérben várakozó feladatok, azok végrehajtása folytatódhat a következő üzenet vételéig. Ha ilyen nincs, a szál „ébren” várakozhat, miközben átengedi más szálaknak a processzszort. Ez a szituáció veszélyeket hordoz magában.

Az üzenetsorokba (**puffer**) helyezett üzenetek feldolgozását aszinkronnak nevezzük, ha a küldő az üzenetet magára hagyja a feladás után, nem foglalkozik tovább vele. A rendszer persze megenged szinkron párbeszédet is. A küldő ilyenkor felfüggeszti a futását, míg a választ meg nem kapja. A szinkron üzenetek súlyos gondokat okozhatnak, amennyiben nem érkezik meg a válasz. Ezt a szituációt az operációs rendszerekben **holtpont**nak (**deadlock**) hívják. Ez az állapot elkerülhető, ha a holtpontba jutott folyamatot töröljük az üzenetsorból. Ha van **feladatkezel**ő szervizprogramunk, akkor ezt megtehetjük, és a többi alkalmazás futhat tovább. Persze csak akkor, ha nem maga a rendszer jutott holtpontba. (Ezt mondjuk úgy, hogy „lefagyott a rendszer”).

10.4. FÁJLOK

Fájlon külső adathordozón elhelyezett, valamilyen katalógusban névvel nyilvántartott, logikailag összetartozó adathalmazt értünk.

Milyen lehet a fájlnév, milyen a katalógusok belső szerkezete, hogyan helyezkednek el az adatok az adathordozón, hogyan kezeli őket az operációs rendszer? Ezeknek a szabályoknak az összességét **fájlrendszer**nek hívjuk.

Az adathordozók egységei, blokkjai címezhetőek. Az operációs rendszer által címezhet legkisebb tárterületet a **blokk** mellett **tárolóegység**nek vagy **klaszter**nek (cluster) is nevezik. A hardverek által megengedett legkisebb méret viszont a **szektor**, amely különbözhet a blokk méretétől. Ha a tároló kapacitását jó hatásfokkal szeretnénk kihasználni, akkor fel kell adni a fájlok folytonos elhelyezését az adathordozón. Ez viszont következményekkel jár: A katalógusban el kell helyezni a fájl kezdőcímét, de a többi hozzá tartozó blokk helyét és logikai sorrendjét is fel kell jegyezni.

A nem egymás melletti blokkokban elhelyezett állományt **töredezett**nek mondjuk. A fájl logikailag egymást követő blokkjainak helyét és sorrendjét indextáblákkal vagy foglaltsági listákkal lehet megadni. Emellett szükség lehet az adathordozó szabad és foglalt blokkjainak nyilvántartására is, amit foglaltsági térképnek nevezünk.

A nem folytonos elhelyezéssel a hatásfok növelhető, de más gondok jelentkezhetnek. A gyakran használt és nagyon feldarabolt fájlok sok fejmozgatást okozhatnak a merevlemezen, ami lassítja a rendszert. Ideiglenesen összefüggővé tehetjük az állományainkat a **töredezettségmentesítő**nek vagy **defragmenter**nek nevezett szervizprogrammal.

10.4.1. Fájlok csoportosítása

A fájlokat számtalan módon csoportosíthatjuk. Az egyik lehetőség a fájlok **felhasználás szerinti** besorolása, amelyet a kiterjesztéssel, a fájlnevet követő utolsó pont után adhatunk meg. Bár nem kötelező, de ajánlott a kiterjesztést következetesen használni.

A másik, a fájlhoz való felhasználói hozzáférés szerinti csoportosítást a katalógusokban vagy leíró táblákban szokták megadni. A tulajdonságokat itt attribútumoknak hívjuk. Tipikus fájltulajdonságok: csak olvasható, írható és olvasható, végrehajtható, rejtett, rendszerfájl stb.

A fájlok használatának szabályozása minden felhasználóra, illetve egyedenként is megadható. A hozzáférést szabályozó lista neve ACL (Access Control List). Ebben található, hogy az egyes felhasználóknak vagy felhasználói csoportoknak milyen jogaik vannak a fájlhoz. Az ACL-listákat a különböző operációs rendszerek más-más módon építik fel, és mindig titkosan kezelik. A leggyakoribb felhasználói jogok: olvasási, írási, törlési, létrehozási, keresési, futtatási jog. Ha az ACL-lista a fájlrendszerből hiányzik, elfogadható védelmi rendszert nem lehet felépíteni. A nagyon sok állomány elhelyezés szerinti csoportosítását megadhatjuk egy hierarchikus, bővíthető és módosítható, fastruktúrájú katalógusszerkezettel. Hogy a katalógusokba újabb katalógusokat lehessen bejegyezni, ezeket is fájlként, speciális belső szerkezettel rendelkező fájlként kezeljük.

11. AZ OPERÁCIÓS RENDSZEREK TÍPUSAI ÉS LEHETŐSÉGEI

11.1. AZ OPERÁCIÓS RENDSZEREK TÍPUSAI

Az operációs rendszereket nagyon sokféle módon csoportosíthatjuk. Hagyományosan az alábbi felosztásokat szokás használni.

<table>
<tr><th colspan="4">Operációs rendszerek felosztása</th></tr>
<tr><td rowspan="5">I. Funkció (feladat) szerint</td><td rowspan="4">1. Általános célú operációs rendszerek</td><td rowspan="2">a) Egy felhasználós (monouser)</td><td>Egyfeladatos: pl. DOS</td></tr>
<tr><td>Többfeladatos (multitasking): pl. Windows</td></tr>
<tr><td rowspan="2">b) Több felhasználós</td><td>Egyfeladatos (kötegelt vagy batch feldolgozás)</td></tr>
<tr><td>Többfeladatos (multi programozás)</td></tr>
<tr><td rowspan="2">2. Speciális célú operációs rendszerek</td><td colspan="2">a Hálózati: pl. Novell Netware</td></tr>
<tr><td colspan="2">b.) Valós idejű (real time)</td></tr>
<tr><td rowspan="2">II. Kommunikáció szerint</td><td colspan="3">1. Interaktív operációs rendszerek (párbeszédes)</td></tr>
<tr><td colspan="3">2. Nem interaktív operációs rendszerek</td></tr>
<tr><td rowspan="4">III. Hardver mérete szerint</td><td colspan="3">1. Mobil operációs rendszerek</td></tr>
<tr><td colspan="3">2. Mikrogépes operációs rendszerek</td></tr>
<tr><td colspan="3">3. Kisgépes operációs rendszerek</td></tr>
<tr><td colspan="3">4. Nagygépes operációs rendszerek</td></tr>
</table>

46. ábra:

Kronológiai szempontból először az egyfeladatos egy felhasználós rendszerek jelentek meg. Ezek voltak az első operációs rendszerek.

A Többfeladatos (multitasking) felépítés lehetővé teszi, egy felhasználó több feladatának időben párhuzamos (konkurens) végrehajtását, egyetlen gépen végrehajtva.

Az egyfeladatos (kötegelt vagy batch) feldolgozás egymástól független munkák egymást követően való végrehajtását teszi lehetővé parancskötegekben. Tartozéka a munkavezérlő nyelv (Job Control Language), melynek feladata a kötegekbe foglalt munkák futtatása.

A Többfeladatos (multiprogramozott) operációs rendszerek a központi egységet egy ütemezési stratégia alapján rendelik hozzá a programokhoz (folyamatokhoz). Ezek több felhasználó feladatát is képesek látszólag egyidejűleg elvégezni, akár egyetlen CPU-val is. Védelmi feladatai nagyon fontosak és sokrétűek, emiatt az ilyen mai rendszerek szinte kizárólag egyben hálózatos rendszerek is.

A speciális célú operációs rendszerek nevükhöz méltóan valamilyen speciális célra lettek kifejlesztve. A speciális célú hálózati operációs rendszerek közül kiemelhetjük a Novell Netware operációs rendszert, amely feladatokat képes megosztani illetve menedzselni a hálózat gépei között.

A valós idejű (real time) operációs rendszerek folyamatvezérlési feladatokra lettek kifejlesztve (pl.: gyártási folyamatok, közlekedési lámpák vezérlésére). Jellemzőjük, hogy a feladatokat szigorú időkorlátok kötik.

11.2. KONKRÉT OPERÁCIÓS RENDSZEREK

11.2.1. Mobil rendszerek

Egy mobil operációs rendszer, másként mobil OS az operációs rendszerek azon csoportja, amelyek egy hordozható, kézi eszközt (például mobiltelefont, okostelefont vagy táblagépet) működtetnek. Működésük és felépítésük nagyban hasonlít „asztali” társaikéra, mint a Windows, OS X vagy a különböző Linux-disztribúciók. Valamelyest mégis egyszerűbbek és főképp a vezeték nélküli kapcsolattartásra és a multimédiára koncentrálnak, a kommunikációs eszköz funkciója mellett a felhasználó szemszögéből inkább szórakoztatóelektronikai, mint informatikai eszközök, és általában más beviteli megoldásokat alkalmaznak.

11.2.2. A leggyakoribb mobil operációs rendszerek, szoftverplatformok

Android a Google Inc.-től

Az Android nevű operációs rendszert kezdetben egy kis cég fejlesztette, amelyet a Google Inc. megvásárolt, és folytatta a rendszer fejlesztését. Az Android egy a Google szolgáltatást biztosító programokat leszámítva nyílt forráskódú, Linux alapú, operációs rendszer

Az első kiadása 2007. november 5-én jelent meg. Az Android főverziók sorban: 1.0, 1.5, 1.6, 2.0, 2.1, 2.2, 2.3, 3.0 (csak tabletekre) és érkezőben a 4.1 (tabletekre és okostelefonokra egyaránt optimalizált).

Blackberry OS a RIM-től

Ez az operációs rendszer a könnyű kezelhetőségre és üzleti felhasználásra fókuszál. Az utóbbi időben az alkalmazásellátottsága nagyon ingadozik, emellett multimédiatámogatást is kapott. Jelenleg a Blackberry App World 15 000 feletti letölthető alkalmazást tudhat magáénak. A RIM jövőbeni stratégiája a Blackberry OS mellett a QNX-re is fókuszál, amit már el is indított a Blackberry Playbook fedélzetén. Az első QNX okos telefonokra 2012 elején lehet számítani.

iOS az Apple-től

Az iOS az Apple iPhone, iPod Touch és iPad operációs rendszere, amit az OS X-ből készítettek. Külső fejlesztőtől származó alkalmazások nem voltak elérhetők a második verziójáig (ekkor még iPhone OS-nek hívták), tehát az első iPhone nem tekinthető valós okos telefonnak vagy PDA-nak, habár egy úgynevezett jailbraik eljárással lehetőség volt az akkor még nem túl sok alkalmazás telepítésére. Ez az eljárás még mindig elérhető, az újabb és újabb operációsrendszer-verziókhoz elkészítik, újabb és újabb lehetőségeket adva a merészebb iPhone felhasználók kezébe. Jelenleg minden iOS-t futtató eszközt az Apple fejleszt és a Foxconn vagy más Apple-lel szerződött partner gyárt.

Symbian OS

A Symbian rendelkezett hosszú ideig a legnagyobb piaci részesedéssel az okostelefonok piacán. Eleinte a Nokián kívül más gyártók, mint az LG, a Mitsubishi, a Motorola, a Samsung a Sharp és a Sony Ericsson is gyártottak Symbiant futtató okos telefonokat. Ekkor még a híres S60 felület mellett egyebek közt egy UIQ felület is létezett, ez azonban háttérbe szorult, majd eltűnt. Végül a Nokia felvásárolta a Symbiant és nyílt forráskódúvá tette jelentős részét. Az iPhone és az Androidot futtató eszközök megjelenésével a részesedésük zuhan-

ni kezdett. A Nokia végül szövetségre lépett a Microsofttal Windows Phone alapú telefonok gyártása miatt. A Symbian fejlesztését az Accenture vette át, aki megerősítette annak támogatását 2016-ig.

Windows Phone a Microsofttól

2010. február 15-én a Microsoft bemutatta a következő generációs mobil OS-ét, a Windows Phone 7-et. Az új operációs rendszer egy teljesen átdolgozott GUI-t kapott. Teljes integrációt kapott a Microsoft szolgáltatásokkal, mint a Windows Live, a Zune, az XBox Live és a Bing, de szintén integráltak sok nem Microsoft elemet, mint Facebook- vagy Google-fiókok hozzáadásának lehetőségét.

11.2.3. Mikrogépes rendszerek

Az első általános célú, egyfelhasználós, monoprogramozott mikrogépes operációs rendszer a **CP/M** volt. A személyi számítógépek első széles körűen elterjedt operációs rendszere az IBM és a Microsoft terméke, a **DOS** lett. Adatkezelése részben a CP/M-ből, részben a Unix-ból származik.

A grafikus felhasználói felület, a GUI a Xerox cégtől származik, és az Apple alkalmazta elsőként a Machintosh típusú gépein.

Az 1980-as évek közepén az IBM és a Microsoft egy új operációs rendszert hozott létre, az **OS/2-t**, mely grafikus felhasználó felülettel is rendelkezett már. Közben a Microsoft (elszakadva az IBM-től) még a DOS-ra építve egy **Windows**-nak nevezett új rendszer kidolgozásába fogott, mely már szintén rendelkezett grafikus felhasználó felülettel (**Windows 3.0, Windows 3.1, Windows 3.11**). 1995-re a Windows önálló (már DOS nélküli) többfeladatos rendszerré fejlődött, és ellátták már egyszerű hálózatkezelési funkciókkal is (**Windows 95, Windows 98**).

Az IBM sem tétlenkedett, immár egyedül kifejlesztette az **OS/2 Warp** operációs rendszert, ami a Windows-hoz hasonló grafikus felülettel rendelkezett. A rendszer magja megbízhatóbb volt, a Windows 95 megjelenése mégis visszaszorította.

A Microsoft 1993-ban adta ki az első sorozatban gyártott szerverekre optimalizált rendszerét, a **Windows NT**-t, mely jelentős hardverigénnyel rendelkezett. 1998-ban jelent meg a 9x és az NT összeolvasztásával a **Windows 2000**. A 90-es évek elején az általános célú operációs rendszerek két ágra szakadtak: **szerver és asztali változat**ra. Az OS/2-ből a már csak szerver változat készül.

A Windows-nál a legújabb szerver a **Windows 2012**, az asztali pedig a **Win8**.

Az 1990-es évekre Linus Benedict Torvalds finn egyetemista elkészítette a Unix-szerű **Linux**ot. Ez teljesen szabadon terjeszthető és fejleszthető operációs rendszer. A Linux fejlesztésébe később több nagy szoftvergyár is beszállt. Vannak külön szerver és asztali változatok is. Ismertebb Linux-változatok: **Debian, Mandrake, Red Hat** és a magyar nyelvre is felkészített **SuSE és Uhu**.

A Machintosh-ok operációs rendszere a **MacOS**. Ezek a számítógépek kezdetben nem az Intel, hanem a Motorola processzorcsaládra épített rendszerek, de az utóbbi néhány évben újra Intel-processzorokat építenek be. Legfrissebb operációs rendszerük az **OS X Mountain Lion.**

11.2.4. Kisgépes rendszerek

Az első ilyen időosztásos operációs rendszer a Unix volt. Ez egy „hordozható" operációs rendszer. Alkalmas helyi és távolsági számítógép-hálózatok kezelésére. Alkalmazási területe elsősorban a nagyobb hardverre jellemző: IBM Power PC, SUN Sparc stb.

11.2.5. Nagygépes rendszerek

A nagygépek közé a mainframe és szuperszámítógépek tartoznak. Ezek sokprocesszoros felépítésű gépek, amelyek több felhasználós hálózatokat és kisebb szervereket képesek kiszolgálni.

Az első operációs rendszerek az 1960-as években készültek ilyen gépekre. Fontos volt a jó hardverkezelés és a felhasználóbarát felület. Két egymástól független fejlesztés készült, melyek a mai napig életben maradtak, az **IBM MVS** és a **Unix**-alapú nagygépes rendszerek (**Solaris, HPUX**).

Később megjelent még az **OS/400** és az **OS/390** is. **Linux**-változat is készült. A mai nagygépes rendszerek főbb jellemzői: jelentősen továbbfejlesztett biztonsági alrendszer, helyi hálózatos és Internetes szolgáltatások, a Unix-szerű kernel grafikus felhasználói felülettel,és széles körű virtuális gépi lehetőségekkel.

11.2.6. Virtuális gépek

A virtuális gép (**VM**, Virtual Machine) fogalmát az IBM vezette be. Azt jelenti, hogy egy **hoszt**nak (**host**) nevezett operációs rendszeren szimulálni tudunk más, esetleg több komplex számítógéprendszert, a hardver és az operációs rendszer együttesét. Virtuális gépnek egy olyan különleges módon vezérelt memóriatartományt szokás nevezni, amelyet az alkalmazás különálló gépként érzékel. A virtuális gép tartalmazza mindazokat a valóságos (fizikai) gépen is elérhető erőforrásokat, amelyek szükségesek az adott alkalmazás futásához. A

virtuális gépek kialakításából származó előnyök közül kiemelést érdemel a rendszer nagyobb biztonsága. A látszólag külön gépeken futó folyamatok nem zavarhatják meg egymás működését.

Legismertebb virtuális gép a **Java VM**, melyet a SUN Microsystem vezetett be és az internet hálózati környezetére tervezte. Ez egy valós számítógéprendszerre telepített bájtkód-értelmező, mely „hordozható”.

12. HÁLÓZATI OPERÁCIÓS RENDSZEREK

12.1. A HÁLÓZATI OPERÁCIÓS RENDSZEREK TULAJDONSÁGAI

A hálózatok felügyeletéhez, működtetéséhez szükség van megfelelő operációs rendszerre. Legtöbb esetben külön beszélhetünk a szerver operációs rendszeréről és külön a munkaállomások operációs rendszeréről.

A hálózati operációs rendszereknek különböző szolgáltatásaik vannak. Ilyenek a memória- és egyéb erőforrások megosztása a felhasználók között, folyamatok menedzselése, kommunikáció megvalósítása, fájlkezelés. Az operációs rendszerek hierarchikus felépítése lehetővé teszi a különböző funkcióknak a megfelelő szintekhez való rendelését. Az alacsonyabb szintek szoftverjei az egyfelhasználós funkciókat, míg a magasabb szintek szoftverei a hálózati működtetéssel kapcsolatos feladatokat látják el.

12.1.1. A hálózati operációs rendszerek biztonsági rendszerei

A korszerű hálózati operációs rendszerek mindegyike legalább négyszintű biztonsági rendszerrel rendelkezik. Ezek:

- bejelentkezési védelem
- jogosultságok védelmi rendszere
- attribútumok védelmi rendszere
- szerver (kiszolgáló) védelem

12.1.2. Bejelentkezési védelem (Login Security)

A hálózatba csak olyan felhasználó jelentkezhet be, aki a rendszer számára „ismert”. Bejelentkezéskor minden felhasználónak névvel és jelszóval kell azonosítania magát a rendszer felé. A hálózati jelszót minden esetben vakon kell begépelni, hogy a képernyőről ne tudja senki leolvasni. A hálózat tervezésekor a

felhasználókat több kategóriába sorolják, ezek közül az alábbiak szinte minden rendszerben megjelennek.

- **Egyszerű felhasználók** (userek): Semmilyen rendszer-karbantartási műveletet nem végezhetnek el, csak használhatják a hálózati erőforrásokat.
- **Kiemelt felhasználók**: Általában valamilyen rendszeradminisztrációs tevékenységgel megbízott felhasználók.
- **Rendszergazda**: Az egész rendszerre kiterjedő felügyeleti joggal rendelkezik, az ő feladata a hálózat biztonságos és zökkenőmentes működtetése.

Installáláskor „automatikusan” keletkezik két felhasználó: a rendszergazda (Administrator, Supervisor) és a **vendég** (Guest). A vendég egy korlátozott jogokkal rendelkező egyszerű felhasználó, aki általában jelszó nélkül használhatja a rendszert. A rendszergazda szintű használat minden esetben jelszóval védett.

Az azonos feladatokat végző felhasználókat célszerű csoportokba (groupokba) szervezi. Ezzel leegyszerűsíthető a rendszeradminisztrációs tevékenység. A hálózat biztonságosabbá tétele érdekében a felhasználói nevekhez tartozó jelszavakra számos előírás adható meg. Ilyenek például:

- Előírható a jelszó minimális hossza. (Ajánlott érték: minimum 6-8 karakter)
- Előírható, hogy a jelszó feltétlenül tartalmazzon betűket és számokat is.
- Megadható, hogy a felhasználónak milyen időközönként kelljen megváltoztatni a jelszavát.

A bejelentkezési védelmet tovább szigoríthatjuk azzal, hogy előírjuk, hogy az adott felhasználó milyen időpontokban és melyik munkaállomásokról léphet csak be a hálózatba.

12.1.3. Jogosultságok védelmi rendszere (Rights Security)

A jogosultsági rendszer ellenőrzi, hogy az adott felhasználó mely könyvtárakkal, alkönyvtárakkal, fájlokkal milyen műveleteket végezhet el. A jogosultsági rendszer konkrét megvalósítása a különböző hálózati operációs rendszerek esetében eltérő lehet, de mindegyik tartalmazza az alábbi alapvető jogosultságokat:

- Olvasás: A felhasználó megtekintheti az adott fájl tartalmát.
- Írás: A felhasználó módosíthatja az adott fájl tartalmát.

- Végrehajtás: A felhasználó futtathatja az adott fájlt.
- Törlés: A felhasználó törölheti az adott fájlt.
- Engedélyek módosítása: A felhasználó módosíthatja a fájl jogosultsági információit.

Ezek a jogosultsági információk egyaránt beállíthatók fájlokra és könyvtárakra is. A könyvtárakra beállított jogosultságokat az adott könyvtárban lévő alkönyvtárak és fájlok is öröklik.

12.1.4. Attribútumok védelmi rendszere (Attribute Security)

A könyvtárakhoz vagy fájlokhoz attribútumokat adhatunk meg, amelyek „erősebbek” az előbb tárgyalt jogosultságoknál. Egy adott fájl vagy könyvtár attribútumait azok a felhasználók változtathatják meg, akik a fájlra, vagy könyvtárra módosítási joggal rendelkeznek.

Szerver védelem

A szerverek minden hálózati operációs rendszer esetén kitüntetett szereppel bírnak, ezért a szerverhez való hozzáférést külön is lehet korlátozni. A szerver konzolja minden esetben jelszóval védhető, megadható, hogy mely felhasználók jogosultak bejelentkezni a szerveren, kik férhetnek hozzá az eseménynaplóhoz, kik kezdeményezhetik a rendszer leállítását stb.

12.2. HÁLÓZATI OPERÁCIÓS RENDSZEREK

Novell NetWare

A hálózati operációs rendszereknek két fajtája van, az egyik egy olyan szoftver, amelynek egyetlen feladata, hogy a hálózat működését felügyelje, irányítsa. Ez a szoftver található a hálózat kiszolgálóján, mint operációs rendszer. Ilyen hálózati kiszolgáló például a Novell NetWare.

A Novell NetWare elsősorban LAN-ok kiszolgálóihoz készített szoftver, amely lehetővé teszi a munkaállomások számára az állományok megosztását. Ehhez a munkaállomáson kezdetben DOS, később Windows illetve OS/2 futott.

A kiszolgáló az állományok közös elérését hálózati meghajtókon keresztül teszi lehetővé. Ezek a meghajtók a gép saját háttértáraihoz hasonlóan használhatók. Tehát valójában egy központi háttértár szerepét látja el a kiszolgáló. Ezt a funkciót nevezik a Novellnél file servernek.

Másik szolgáltatás a hálózati nyomtató, vagyis a print server. A munkaállomáson kezdeményezett nyomtatás hatására a kinyomtatandó dokumentum a kiszolgáló nyomtatási sorába kerül, ahonnan a kiszolgáló már a munkaállomástól függetlenül végzi a nyomtatást. A nyomtató csatlakozhat közvetlenül a központi géphez, vagy a hálózathoz egyaránt.

Harmadik szolgáltatása a Novel hálózatoknak a mail server. Ez egy belső levéltovábbítási rendszer a felhasználók között. Egyszerű szöveges leveleket tud továbbítani a felhasználó azonosítója alapján.

Unix és Linux rendszerek

A másik típusú hálózati operációs rendszer hasonló egy hagyományos értelemben vett operációs rendszerhez, amely hálózatkezelő funkciókkal is rendelkezik. Ebben az esetben a számítógépet egy hagyományos munkaállomáshoz hasonlóan lehet használni, miközben az más számítógépeket is kiszolgál a hálózaton keresztül. Erre az egyik legjobb példa a Unix operációs rendszer, amely a nagygépek operációs rendszereként már a 60-as évek óta ellát hálózatkezelő funkciókat.

A Unix szolgáltatásainak ismertetése tulajdonképpen megegyezik az internetes szolgáltatások ismertetésével, hiszen a Unix operációs rendszer alatt dolgozták ki az internetes szolgáltatásokat (pl. a korábban már említett Telnet, FTP, e-mail stb.).

A Unix rendszerekben a jelszót úgy tárolják, hogy egy titkos algoritmussal a jelszóból előállítanak egy teljesen új jelsorozatot, és ezt tárolják el. Az algoritmus vissza nem fordítható módon végzi mindezt, így még ha más hozzá is fér a jelszó tárolt alakjához, akkor sem képes a jelszót kideríteni. A bejelentkezésnél a begépelt jelszót ugyanazzal a módszerrel elkódolják, és akkor tekinti a rendszer helyesnek a jelszót, ha a két kódolás eredménye ugyanaz a jelsorozat.

A megfejthetetlenséghez azonban bizonyos szabályokat be kell tartani:

- A jelszó ne legyen értelmes szöveg.
- Tartalmazzon számjegyeket és betűket egyaránt.
- Ne legyen könnyen megjegyezhető, hogy ha valaki netalán mégis rájön, minél könnyebben elfelejtse:-)
- Időnként (2 hét) le kell cserélni.
- Senkinek ne áruljuk el jelszavunkat.

Felhasználói jogosultságok

Miután bejelentkeztünk, a hálózat által számunkra biztosított jogoknak megfelelően tevékenykedhetünk. A Unix rendszereken egységes jogosultsági rendszer használatos, amely három kategóriába osztja a felhasználókat minden egyes állomány esetében:

- Az állomány tulajdonosa
- Az a csoport, amelyhez az állomány tulajdonosa tartozik (pontosabban amely csoportba ezek közül az állományt a tulajdonosa sorolta)
- Mindenki egyéb felhasználó

Ezen három kategórián belül ugyanazok a jogok adhatók, illetve vonhatók meg:

- **olvasási jog**: Az illetőnek joga van-e olvasni az állomány tartalmát.
- **írási jog**: joga van-e módosítani a tartalmát az állománynak.
- **futtatási jog**: joga van-e programként elindítani. Csak azt az állományt érdemes ezzel a joggal ellátni, amely futtatható programot tartalmaz.

12.3. SZABVÁNYOK ÉS PROTOKOLLOK

A szabványok használata elengedhetetlen globalizálódó világunkban. Azokban az időkben, amikor a számítógépes hálózatok világában még nem alkalmazták a szabványokat a különböző gyártók által készített eszközök képtelenek voltak egymással kommunikálni.

Ennek megváltoztatásához olyan szabályokat hoztak létre, amelyek elősegítették a hálózati eszközök közötti hatékony kommunikáció létrejöttét. Ezeknek a szabályoknak és konvencióknak az összességét protokolloknak nevezzük.

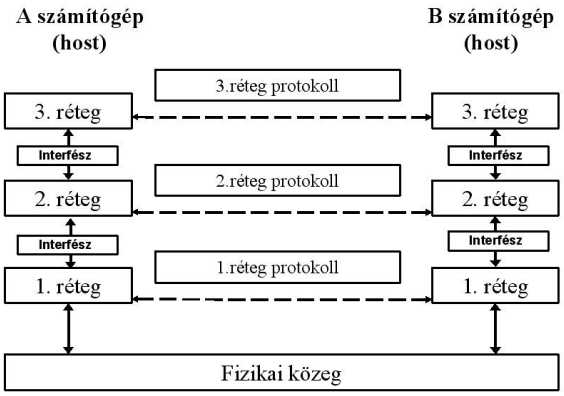
- Számos szervezet foglalkozik pl. a hálózati szabványok létrehozásával és vizsgálatával, a teljesség igénye nélkül íme néhány:
- ISO (International Organization for Standardization): Nemzetközi Szabványosítási Szervezet. Több mint 100 szabványügyi szervezetet magába foglaló nemzetközi testület (ne firtassuk, hogy miért nem IOS :-).
- IEEE (Institute of Electrical and Electronics Engineers): Az elektronika tárgyköréhez köthető nemzetközi mérnöki szervezet, amely számos szabvány kidolgozásában játszott jelentős szerepet, pl. Ethernet-szabvány stb.
- Interne**t Engineering Task Force (IETF)**: Az interneten használt protokollokért felelős szervezetek egyike.

- **World Wide Web Consortium (W3C)**: Nemzetközi szervezet, amely www-re wontakozó szabványok fejlesztését koordinálja.

12.3.1. Protokollhierarchiák

A számítógéphálózatokat tervezésük összetettségének csökkentése érdekében **rétegekbe** szervezik

- •Réteg: jól definiált szolgáltatásokat biztosítva eltakarja a nyújtott szolgáltatások megvalósításának részleteit a felsőbb rétegektől, minden réteg az alatta levőre épül
- •Egyik gép n. rétege a másik gép n. rétegével kommunikál.
- •Minden réteg adat és vezérlőinfomációt ad át az alatta levő rétegeknek, egészen a legalsó rétegig
- •Interfész: rétegek közötti határfelület, a felső rétegnek nyújtott elemi műveleteket és szolgálatokat definiálja



47. ábra: Protokoll-hierarchia

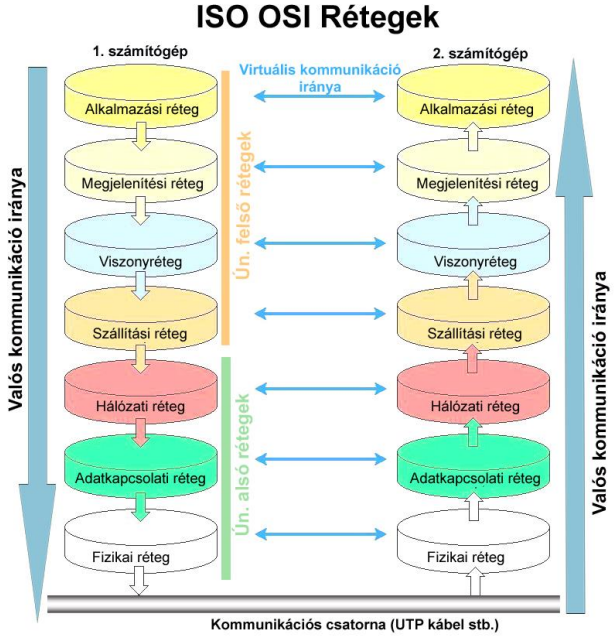
12.3.2. ISO-OSI hivatkozási modell

Az OSI modellje a különböző protokollok által nyújtott funkciókat egymásra épülő rétegekbe sorolja. Minden réteg csak és kizárólag az **alsóbb rétegek által nyújtott funkciók**ra támaszkodhat, és az általa megvalósított funkciókat pedig csak felette lévő réteg számára nyújthatja. A rendszert, amelyben a protokollok viselkedését az egymásra épülő rétegek valósítják meg, gyakran nevezik „protokoll veremnek" vagy „veremnek". A protokoll verem mind hardver szinten, mind pedig szoftveresen is megvalósítható, vagy a két megoldás keverékeként is. Tipikusan csak az alsóbb rétegek azok, amelyeket hardver szinten (is) megvalósítanak, míg a felsőbb rétegek szoftveresen kerülnek megvalósításra.

Ez az OSI-modell alapvetően meghatározó volt a számítástechnika és hálózatokkal foglalkozó ipar számára. A legfontosabb eredmény az volt, hogy olyan specifikációkat határoztak meg, amelyek pontosan leírták, hogyan léphet egy réteg kapcsolatba egy másik réteggel. Ez azt jelenti a gyakorlatban, hogy egy gyártó által írt réteg programja együtt tud működni egy másik gyártó által készített programmal (feltéve, hogy az előírásokat mindketten pontosan betartották). Az említett specifikációkat a TCP/IP közösség a Requests for Comments vagy „RFC”-k néven ismeri. Az OSI közösségben használt szabványokat itt lehet megtalálni: ISO szabványok.

Alapelvek:

- A rétegek különböző absztrakciós szinteket képviseljenek
- Minden réteg jól definiált feladatot hajtson végre
- A rétegek feladatának megválasztásakor nemzetközileg elfogadott szabványok kialakítására kell törekedni
- A rétegek száma megfelelő legyen
- Különböző feladatok ne kerüljenek egy rétegbe
- A szerkezet ne legyen nehezen kezelhető



48. ábra:

12.3.2. Fizikai réteg

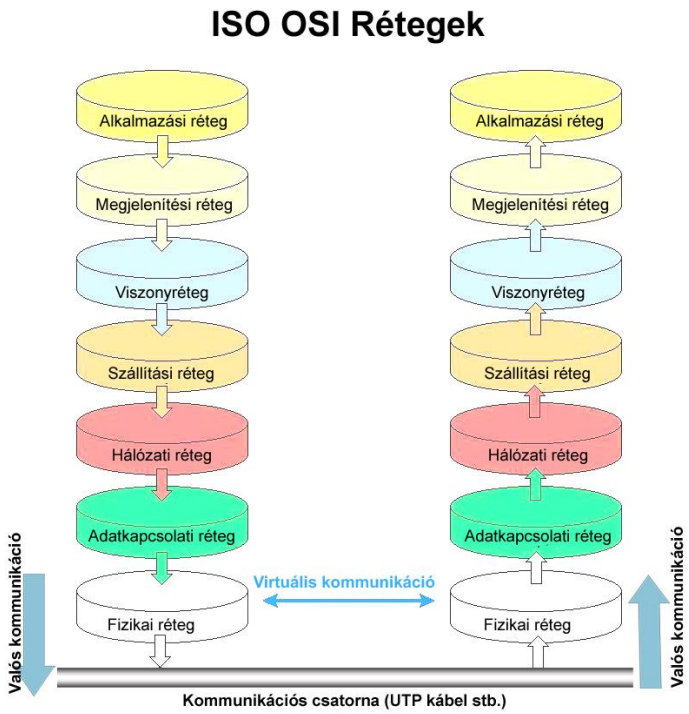
A bitek kommunikációs csatornára bocsátásáért felel

- Mechanikai, elektromos, fizikai jellemzők

Az OSI modell alján helyezkedik el a fizikai réteg, amely a hálózat fizikai jellemzőivel áll kapcsolatban: milyen kábelek és csatlakozók használhatóak, a kábelek milyen hosszúak lehetnek stb.

A fizikai réteg másik aspektusa a hálózaton továbbított elektronikus jelek leírása, azonban ezeknek a jeleknek a jelentésével nem foglalkozik, csak pl. a logikai 1 és 0 szintjének meghatározására szorítkozik.

A fizikai réteg eszközei közül megemlíthetjük pl. a repeatert (jelismétlő) és a hálózati csatolókártyát. Az előbbi eszközt akkor használjuk, ha a fizikai réteg által megadott hossznál hosszabb kábelt kívánunk használni a hálózat kiterjesztése érdekében. Működése során a bemenetére érkező jeleket a repeater minden vizsgálat nélkül továbbítja a kimenetére.



49. ábra:

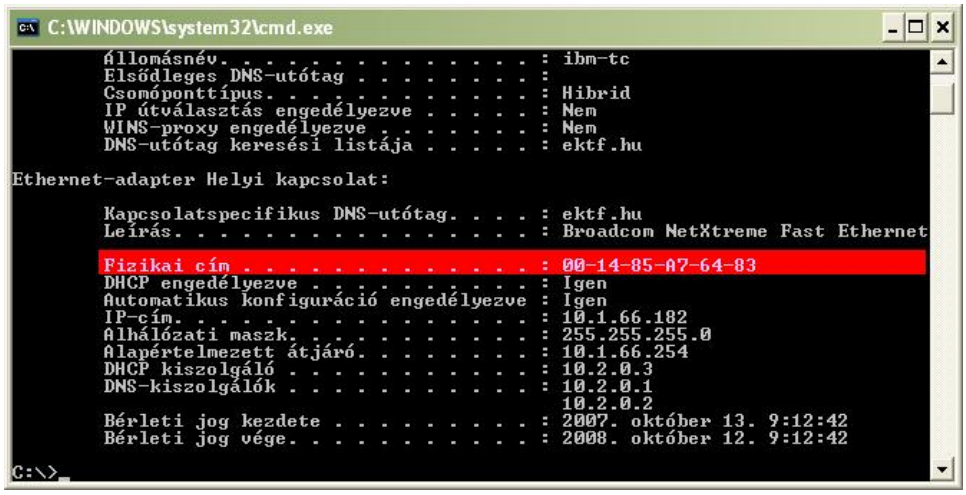
13.3.3 Adatkapcsolati réteg

Az adatkapcsolati réteg elsődleges feladata, hogy a hálózat csomópontjai között hibamentes átvitelt biztosítson illetve az adatkeretek kezelése.

Míg a fizikai réteg csak az egyes jelekkel (0,1) foglalkozott, addig az adatkapcsolati réteg a jelek sorozatát ún. adatkeretekben vizsgálja. Az adatkeretek többnyire n×100 byte hosszúságúak. A küldő gép adatkapcsolati rétege a keret tartalmát felhasználva elvégez egy számítást az adatkereten, melynek eredményét hozzácsatolja a kerethez. A fogadó gép adatkapcsolati rétege megismétli ezt a műveletet és a kapott eredményt összehasonlítja a kerethez csatolt értékkel, ezzel biztosítva a keretek sérülésmentességének ellenőrzését. Ha a keret sérülésmentes, akkor a fogadó gép adatkapcsolati rétege küld egy ún. nyugtajelet a küldő gép adatkapcsolati rétegének.

Az adatkapcsolati réteg szintjén minden egyes hálózati eszköz egyedi azonosítóval rendelkezik, amelyet az eszköz gyártása során megváltoztathatatlanul rendelnek az eszközhöz. Ez az azonosító az ún. MAC (Media Access Controll).

A MAC magyar fordítása rendszerint Fizikai cím. Ha a Start/Futtatás-ra klikkelünk, majd a megjelenő ablakba beírjuk a cmd parancsot, a karakteres felületű parancssori ablakban az ipconfig /all parancs kiadásával megjeleníthetjük gépünk bizonyos hálózati adatait.



50. ábra:

Az adatkapcsolati réteg eszközei közé tartozik pl. a bridge (híd) és a switch. Az előbbi a nevét onnak kapta, hogy két hálózati elem között (egyfajta hídként) azok MAC-azonosítója alapján tartja a kapcsolatot és biztosítja az adatok megfelelő áramlását. A bridge eszközt szokták intelligens repepater-nek is nevezni.

A switch (amelyet intelligens hubnak is neveznek), megvizsgálja az érkező adatsorozatok MAC-azonosítóját annak érdekében, hogy a megfelelő portra küldhesse az adatokat.

13.3.4. Hálózati réteg

Kommunikációs alhálózatok működését vezérli

- Csomagok forrás- és célállomás közötti útvonalának meghatározása

A hálózati réteg végzi a számítógépes hálózatokban a számítógépek közötti kommunikáció során az adatok útvonalának megválasztását (természetesen ez csak akkor lehetséges, ha több útvonal is rendelkezésre áll az adatok továbbítására). Az itt alkalmazott protokoll feladata tahát az útválasztás és a logikai címzés.

Ahogy az előző fejezetben már említettük, a hálózati eszközök rendelkeznek egy olyan azonosítóval, amelyet az előállításuk során rendeltek hozzá az eszközhöz és amelyet nem tudunk megváltoztatni (MAC). Ha az adatokat többféle úton (esetleg különböző típusú hálózatokon) kell továbbítani, akkor a MAC alapján ez nem lehetséges, mert a MAC nem tartalmaz információt arról, hogy az adott eszköz melyik hálózatban található.

Ha olyan címzési módszert akarunk használni, ahol az eszköz azonosítója információt szolgáltat arról is, hogy az adott eszköz melyik hálózatban helyezkedik el, illetve amelyeknél mi határozhatjuk meg az eszközök azonosítására szolgáló címet, akkor a logikai címzést kell használnunk.

A logikai címzést a hálózati réteg protokolljaival valósíthatjuk meg, ilyen pl. az IP (Internet Protocol), amelyet rendszerint párban használnak a TCP-vel (Transmission Control Protocol). A hálózati protokoll feladata, hogy a MAC azonosítókhoz hozzárendeljék a megfelelő logikai címet, amely alapján már azonosítható nem csak az eszköz, de az a hálózat is, amelyben az eszköz található.

13.3.5. Szállítási réteg

Adatokat fogad a felső rétegtől, kisebb darabokra vágja szét, átadja hálózati rétegnek, biztosítja, hogy minden darab megérkezzen

A szállítási réteg azért felel, hogy az adatcsomagok megbízhatóan és hibamentesen eljussanak az egyik számítógéptől a másikig. Ezt úgy teszi, hogy kapcsolatot teremt a hálózati eszközök között, nyugtázza a csomagok kézbesítését és újra elküldi az elveszett vagy sérült csomagokat.

A szállítási réteg a nagyméretű adacsomagokat kisebb méretű csomagokká darabolja a hatékonyabb szállítás érdekében. A fogadó számítógép ezeket a

csomagokat összeilleszti, és megvizsgálja, hogy minden adatcsomag megérkezett-e.

Bizonyos esetekben a sebesség és a hatékonyság fontosabb, mint a megbízhatóság. Ilyenkor a küldő fél nem foglalkozik az adatok elküldése előtt a kapcsolat felépítésével, csupán elküldi a csomagokat.

13.3.6. Viszonyréteg

Lehetővé teszi kapcsolati viszony létesítését két számítógép között

A kapcsolati viszony (session) azt jelenti, hogy két számítógép között kiépül a kommunikációs kapcsolat, adatok jutnak el egyik géptől a másikig (szállítási réteg feladata), majd az adatáramlás befejezésével a kapcsolat megszűnik a két gép között.

A viszonyréteg háromféle kommunikációs módot tesz lehetővé:

1. Simplex: Az adatok továbbítása csupán egy irányban történhet

2. Half-duplex: Az adatok továbbítása kétirányú, de adott időpillanatban vagy csak az egyik, vagy csak a másik irányba történhet.

3. Duplex: Az adatok továbbítása egyszerre lehetséges mindkét irányban.

13.3.7. Megjelenítési réteg

Foglalkozik az átviendő információ szintaktikájával, szemantikájával

- •Különböző ábrázolásmódú számítógépek is képesek egymással kommunikálni (ASCII-EBCDIC)
- •Adattömörítés, hitelesítés, titkosítás

A megjelenítési (prezentációs) réteg felel azért, hogyan jelennek meg az adatok az alkalmazások számára.

Pl. a legtöbb számítógép és operációs rendszer (Windows, Unix, Macintosh) ún. ASCII (American Standard Code for Information Interchange) kódolást használ az adatok kódolására. Azonban más számítógépek (pl. IBM mainframe-ek) EBCDIC (Extended Binary Coded Decimal Interchange Code) kódolást használnak. Az ASCII és az EBDIC nem kompatibilisek egymással, ezért ha egy Windows-t futtató számítógép kapcsolódni akar egy IBM mainframe-hez, a megjelenítési réteg feladata az adatkonverzió elvégzése.

A megjelenítési réteg az adatkonverzión túl képes az adatok tömörítésére és titkosítására is.

13.3.8. Alkalmazási réteg

Széles körben igényelt protokollokat tartalmaz

- •Virtuális terminál, elektronikus levelezés, távoli terminál elérés

Az alkalmazási réteg (az OSI-modell legfelső rétege) a felhasználói programok számára biztosítja a hálózati kommunikációt. Az elnevezés megtévesztő lehet, mert pl. az Outlook Express nem része a rétegnek, de a kimenő levelek továbbításáért felelős protokoll (SMTP, Simple Mail Transfer Protocoll) igen.

Néhány ismertebb protokoll, amely ide tartozik:

- - FTP (File Transfer Protocoll), fájlok átvitele
- - Telnet (távoli bejelentkezés) kimenő levelek továbbítása
- - SMTP (Simple Mail Transfer Protocoll)
- - stb.

13.3.9. Az OSI-modell kritikája

Rossz időzítés

Egy szabvány megjelentetésének időpontja rendkívül erősen befolyásolhatja annak sikerét. A szabványosításnak a kutatások befejezése után és a beruházások megkezdése előtt kell megtörténnie. Ez azért fontos, hogy a kellő tapasztalat birtokában egységes szabványt hozhassunk létre. Mire az OSI protokollok megjelentek, addigra a versenytárs TCP/IP-protokollok már széles körben elterjedtek a kutatóegyetemeken.

Rossz technológia

A protokollok nem voltak tökéletesek. A viszony réteget alig használja a legtöbb alkalmazás, a megjelenítési réteg pedig szinte teljesen üres. Eredetileg csak öt réteg volt, de mivel a nagy befolyású IBM-nek már volt egy 7 rétegű modellje ezért 7 rétegűre módosították.

Rosszul implementálható

A modell és a protokollok rendkívüli bonyolultsága miatt az implementációk kezdetben terjedelmesek, kezelhetetlenek és lassúak voltak. Mindenki megbukott, aki próbálkozott vele. Nem telt bele sok idő, és az „OSI”-ról mindenkinek a „gyenge minőség” jutott az eszébe. Bár az idők során egyre jobbak lettek

a termékek, a kialakult kép nem változott. Ugyanakkor a TCP/IP egyik első implementációja a Berkeley-féle Unix része volt, és nem csak nagyon jó, de még ingyenes is volt. Az emberek gyorsan rászoktak, így komoly felhasználói tábora alakult ki. Ennek köszönhetően egyre jobb lett a termék, ami tovább növelte a felhasználók körét.

Kis túlzással azt mondhatnánk, hogy az OSI-modellre épülő protokollokat szinte egyetlen hálózat sem használja, bár a tervezéskor mindenki az OSI-modell alapján gondolkodik.

13.3.10. A TCP/IP

A TCP/IP nem más, mint egy protokollkészlet, amelyet arra dolgoztak ki, hogy hálózatba kapcsolt számítógépek megoszthassák egymás között az erőforrásaikat. A fejlesztés az *ARPAnet* köré csoportosult kutatók munkája. Valószínűleg az ARPAnet a legismertebb TCP/IP-alapú hálózat.

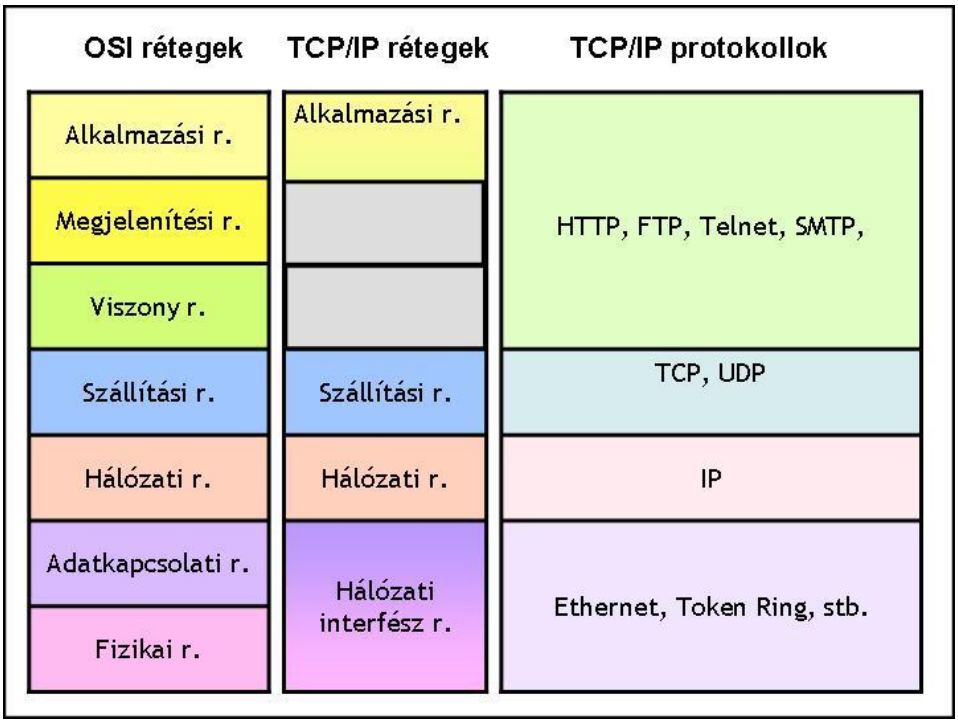
A TCP/IP protokollcsomag négy rétegű szerkezete részben hasonló az ISO OSI hivatkozási modellhez.

TCP/IP protokoll, amelyre az internet épül, valójában nem egyetlen protokoll, hanem egy protokoll-csomag. A TCP/IP-modell az Amerikai Nemzetvédelmi Minisztérium szárnyai alatt született meg 1969-ben, létrehozásának az volt a célja, hogy heterogén hálózatokat úgy lehessen összekötni, hogy a célállomások között mindig több útvonal legyen kialakítható, bármiféle központ nélkül.

TCP/IP igazi jelentőségét az a tény adja, hogy az internet szabványos és elfogadott kommunikációs protokolljává nőtte ki magát. Így az internet és a helyi hálózatok problémamentes összekapcsolása miatt mára a helyi hálózatok de facto szabványává is vált.

De facto, de jure szabványok

Alapvetően a szabványoknak két családja van: a de facto és a de jure. Ez utóbbi esetben hivatalos szervezetek deklarálják és hivatalosan dokumentálják a szabványokat, míg az előbbi egy széles körben használt konkrét megoldásból alakul ki, amelyet aztán célszerű de jure szabványokká alakítani.



51. ábra:

http://www.tankonyvtar.hu/hu/tartalom/tkt/operacios-rendszerek/ch05s04.html