ábra: Ábra

Boros Norbert, Kallós Gábor, Krankovits Melinda, Pukler Antal, Szörényi Miklós

Informatikai rendszerek alapjai

Szerkesztők: Krankovits Melinda

Bevezetés

Előszó

Az Informatikai rendszerek alapjai tantárgy 3 modult tartalmaz. Az első modul az informatikai alapismeretek modul, bevezető jellegű, és a klasszikus felépítést követi: az adatábrázolási ismeretek után a hardver témakör tárgyalása következik, majd az operációs rendszerekkel kapcsolatos fontos tudnivalókat tekintjük át.

A modul célja részben az általános informatikai műveltség kialakítása, ill. „csiszolása", részben pedig a következő anyagrészek megalapozása. Csak egy példát említünk ezzel kapcsolatban: az adatábrázolási ismeretek szükségesek az Excel számítások, a VBA, illetve a MATLAB modul feldolgozásához.

Fontosnak ítéltük azt is, hogy az elméleti leíráshoz tartozzon megfelelő gyakorlati feladatblokk, így a hardver és operációs rendszerek részt ezzel is kiegészítettük.

A második modul a táblázatkezelés. A műszaki, gazdasági életben gyakran előfordulnak olyan problémák, feladatok, amiket a leghatékonyabban egy táblázatkezelő alkalmazás segítségével lehet megoldani. A modulban egy ilyen program használatát fogjuk bemutatni az alapoktól indulva, hiszen fontos célunk az ismeretek korrekt elméleti megalapozása, tisztázása. A témakör végcélja nem csak az, hogy bemutassa a táblázatkezelő alapvető használatát, hanem hogy olyan trükköket is megtanítson, amiknek a használatát elsajátítva időt és energiát spórolhatunk a feladatok megoldásánál. A gyakorlati feladatokat a Microsoft Office Excel 2010 táblázatkezelővel fogjuk bemutatni, de fontos hangsúlyozni, hogy a táblázatkezelés „filozófiája" eszközfüggetlen. Ezért lehetőség szerint úgy tárgyaljuk a tananyagot, hogy a későbbiekben, szükség esetén át tudjon állni bármelyik más táblázatkezelőre a hallgató.

A harmadik modul a matematikai számítások Excellel. A műszaki, gazdasági élet területén munkálkodó szakember gyakran találkozik olyan problémákkal, amelyek megoldásához matematikai háttér és ilyen jellegű számítógépes támogatás szükséges. Gondolhatunk például arra, hogy egy konkrét vizsgált jelenség szinte minden esetben matematikai függvényekkel írható le (esetleg közelítő módon), vagy egy optimalizálási feladat megoldásához matematikai modell felállítása szükséges (megfelelő egyenlőtlenségrendszer, célérték, változó adatok, korlátozó feltételek) a tényleges – általában számítógéppel végrehajtott – megoldás előtt.

Az Excel használata ehhez a problémakörhöz nyilvánvalóan kézenfekvő választás: a programot széles körben használják (a kutatásoknál és az iparban is általánosan alkalmazott segédeszköz), ráadásul bármely felsőoktatásban tanuló hallgató számára ingyenesen elérhető. A rendszer univerzális – nagyon sokféle probléma megoldásához tud segítséget nyújtani –, és emellett kellően robusztus, mély tudású. (Bár, mint látni fogjuk, természetesen a rendszer határait is „elérhetjük", akár nem kimondottan komoly feladatok megoldása során is.)

Megemlítjük azt is, hogy a most tárgyalt részek közül több „visszaköszön" majd (nagyon hasonló formában) az Informatika II (Számítási módszerek) tárgy MATLAB moduljában is. Érdemes lesz majd visszalapozni, és a hasonlóságokat, eltéréseket elemezni.

Ahogy a modul címéből is következik, a most következő leckék feldolgozásához szükséges bizonyos alapvető matematikai háttér. Tananyagunk szervezése és előírt terjedelme nem teszi lehetővé azt, hogy teljes matematikai igényességgel leírjunk minden olyan fogalmat, állítást, tételt (bizonyítással),

amit fel fogunk használni. Ehelyett azt a megoldást alkalmazzuk, hogy röviden összefoglaljuk a szükséges elméleti matematikai ismereteket, és az érdeklődők számára megadjuk a megfelelő szakirodalmat, ahol bővebben utána lehet járni a részleteknek. Ilyen tankönyvet, szakkönyvet természetesen – a megadottakon túl – mindenki szabadon is választhat, a modul jellegéből következően az analízis, a lineáris algebra és az operációkutatás területéről.

A tárgyalásnál – a korábbi részekhez hasonlóan – elsősorban a probléma-orientált megközelítést alkalmaztuk: egy-egy kitűzött konkrét feladat kapcsán ismerkedünk meg a megoldás lehetőségeivel és a rendszer által nyújtott támogatással. Az érdeklődő olvasó gyakran a bemutatottakon túlmenően további megoldásokat is konstruálhat – erre többször utalunk majd.

Végezetül jó felkészülést kívánunk minden kedves hallgatónak!

1. modul: Informatikai alapismeretek

1. lecke: Alapok

Cél: Az informatikai tárgy keretein belül elsőként számrendszerekkel és adatábrázolással (kódolással) foglalkozunk. Ez szükséges már az alapszintű Excel feladatok értelmezésénél és megoldásánál, valamint a Visual Basic, ill. MATLAB tanulmányok megkezdésénél. (Számítási módszerek tárgy, Informatika II.)

A lecke kisebb terjedelmű olvasmányos részeket is tartalmaz.

Követelmények: Ön akkor sajátította el megfelelően a tananyagot, ha

- át tud váltani pozitív egész számokat tízesből kettesbe és fordítva;
- emlékezetből fel tudja írni a tanult logikai műveletek igazságtáblázatát;
- saját szavaival el tudja mondani a szövegek kódolásának főbb jellemzőit;
- képes az „A”, „a” és „0” jelek ASCII kódjának megadása után az angol nagy- és kisbetűk, ill. a számjegyek kódjainak meghatározására;
- elő tudja állítani negatív egész számok kettes komplemens kódját, illetve megadott előjeles kód alapján vissza tudja állítani a kódolt eredeti egész számot.

Időszükséglet: A tananyag elsajátításához (a feladatok megoldásával együtt) hozzávetőlegesen 3 órára lesz szüksége.

Kulcsfogalmak

- Kettes, tízes és tizenhatos számrendszer, átváltások.
- Boole-algebra, logikai műveletek, igazságtáblázat.
- ASCII kódtáblázat, karakterek kódja.
- Kettes komplemens kódolás.
- Gépi epszilon.

Számrendszerek

Út a helyi értékes számrendszerekig, a tízes és a kettes számrendszer

A számolás kialakulása és (korai) fejlődésének története az emberi kultúra rendkívül fontos és érdekes fejezete. Jegyzetünkben nincs mód arra, hogy ezt az izgalmas periódust kifejtve tárgyaljuk, ezért csak felsorolásszerűen közöljük azokat a főbb mérföldköveket, amelyek elvezettek odáig, hogy a ma használt – alapszintű – számolási módszerek, jelölések és rendszerek az élet természetes részévé váltak:

- A számolás kezdetei (ujjakon, apró tárgyakkal; Kr. e. 500 000 – Kr. e. 20 000), a számfogalom kezdetleges megjelenése (kezdetben csak: egy, kettő, sok).
- Fejlettebb számfogalom, de még csak „kapcsolatokban” (kezdetben a számokat nem önállóan, hanem valaminek a mennyiségét, nagyságát kifejezve használták; tört mennyiségek is).

- Önállóan is megjelennek a számok.
- Számrendszerek, kitűntetett alappal (hatvanas, tízes, tizenkettes stb., ókor), mai naptárunk és időmérésünk is az akkori konvenciókon alapul.
- A tízes számrendszer válik hivatalossá a világ kulturált részének „zömén” (ókor).
- Római számírás és számolás (tízes alapon, egyértelmű, de még nem valódi helyiértékes rendszer, Európában a 13. századig általánosan elterjedt forma; hátrányok: nehézkes aritmetika, nincs nulla jel).
- Az ókorban kialakult hindu tízes alapú rendszer (a nullát is használták) arab közvetítőkön keresztül eljut Európába és lassan elterjed.
- Más alapú számrendszerek leírása tudományos igényességgel (elsőként a kettes, az újkortól).

A következőkben áttekintjük a helyiértékes számrendszerek fontos jellemzőit.

Tekintsük először a decimális rendszert, azaz legyen a rendszer alapja a tíz. A számírás azon a tényen alapul, hogy minden nemnegatív valós szám felírható olyan tíz-hatványok összegeként, amelyben minden különböző tíz-hatvány maximum kilencszer szerepelhet. Ennek az összegnek a rövidített felírásából keletkezik a szám helyiértékes alakja (a képviselt érték tehát a sorban elfoglalt hellyel is meghatározott, ellentétben a római rendszerrel).

Nem kell mást tenni, csak az összegben szereplő tíz-hatványok együtthatóját jelentő számjegyeket a kitevők nagyság szerinti sorrendjében leírni, figyelembe véve, hogy amelyik kitevő nem szerepel az összegben, annak az együtthatója nyilván nulla. A nulladik kitevő együtthatója után vesszőt teszünk, ezzel jelezve, hogy befejeződött az egészrész, és a törtrész következik.

Példa: Az 1948,4 valójában az

$1\cdot 1000+9\cdot 100+4\cdot 10+8\cdot 1+4\cdot \frac{1}{10}=1\cdot 10^3+9\cdot 10^2+4\cdot 10^1+8\cdot 10^0+4\cdot 10^{-1}$ összeg rövidítése.

Ebben valóban maximum kilencszer szerepel minden előforduló tíz-hatvány.

Általánosan, ha a számot a -val jelöljük, az előzőek képletben:

$a=a_{n-1}10^{n-1}+a_{n-2}10^{n-2}+\ldots+a_0 10^0+a_{-1}10^{-1}+\ldots+a_{-m}10^{-m}=\sum_{i=0}^{n-1}a_i 10^i+\sum_{i=1}^{m}a_{-i}10^{-i}$, ahol n az egész rész számjegyeinek számát, m a törtrész számjegyeinek számát jelenti. Minden lehetséges i indexre $a_i\in\{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9\}$, azaz a_i valamelyik számjegy. Az összeg segítségével az a számot egyszerűen úgy lehet felírni, hogy az a_i együtthatókat az indexek szerint csökkenő sorrendben egymás után írjuk, az a_0 után tizedesvesszőt teszünk. Azaz

$a=a_{n-1}a_{n-2}\ldots a_0,a_{-1}\ldots a_{-m}$.

Ha a szám negatív, akkor igaz, hogy $a=-|a|$, de az $|a|$ már nemnegatív, így a hatványok összegére való bontás az előzőek szerint elvégezhető, és $a=-a_{n-1}a_{n-2}\ldots a_0,a_{-1}\ldots a_{-m}$. Ezzel a módszerrel tehát bármilyen valós szám tíz számjeggyel (plusz az esetleges előjellel és a tizedesvesszővel) felírható. Az alapműveletek a rendszer segítségével könnyen automatizálhatók, ezért elvégzésük könnyen tanulható (ez volt a tízes alapú számrendszer egyik igen nagy előnye a római számolással szemben, és ennek lett köszönhető elterjedése is).

Jegyezzük meg: a tízes számrendszerben a számok jeleinek helyiértékes felépítéséhez pontosan tíz számjegy kell.

Természetesen a helyiértékes felírási mód nemcsak a tízes, hanem tetszőleges számrendszer esetén is használható, csak a számjegyeket kell megfelelően megválasztani, továbbá meg kell mutatni, hogy az adott számrendszerben is felírható minden szám a számrendszer alapjának hatványait megfelelően összegezve; mégpedig úgy, hogy az összegben minden hatvány legfeljebb az alap értékénél eggyel kevesebbszer szerepelhet (azaz: a számjegyek száma minden számrendszerben pontosan az alapszám értékével egyezik meg).

A számjegyek kiválasztása tíznél kisebb alap esetén nem okoz fejtörést, egyszerűen elhagyjuk a tízes rendszer számjegyei közül a feleslegeseket (az éppen vizsgált alapnál nagyobb vagy egyenlő értékű jegyeket). Ha azonban az alap tíznél nagyobb, akkor új számjegyekre van szükség. A mai gyakorlat szerint nem új jeleket szokás új számjegyként kreálni, hanem az latin ábécé elejéről választunk annyi nagybetűt, amennyit az alap értéke megszab.

A kettes számrendszerben is – a tízes rendszerhez hasonlóan – igaz, hogy minden nemnegatív valós szám felírható az alap, azaz a kettő hatványainak olyan összegeként, amelyben minden kettőhatvány maximum egyszer szerepel. Képlettel: tetszőleges nemnegatív a valós számra igaz, hogy

$a = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_0 2^0 + a_{-1}2^{-1} + ... + a_{-m}2^{-m} = \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=1}^{m} a_{-i}2^{-i}$, ahol minden lehetséges i-re $a_i \in \{0,\ 1\}$. Az a szám kettes számrendszerbeli alakja pedig:

$a = a_{n-1}a_{n-2}...a_0, a_{-1}...a_{-m}$. Negatív szám esetén ugyanúgy járunk el, mint ahogyan a tízes számrendszernél láttuk.

A kettes számrendszer számjegyeit (a 0-t és az 1-et) angol nevük (binary digit) rövidítéséből bitnek is szokás nevezni.

Tevékenység: Írja fel kettes számrendszerben a következő számokat: 1, 2, 3, 4, 5.
Végezze el kettes számrendszerben a következő összeadási műveleteket: 1 + 1, 1 + 2, 1 + 3, 3 + 3.

Vizsgáljuk meg most a tizenhatos számrendszerbeli felírásokat! Alkalmazva a tízes és a kettes rendszernél már tárgyalt ismereteket, azt kapjuk, hogy minden nemnegatív valós a számra

$a = a_{n-1}16^{n-1} + a_{n-2}16^{n-2} + ... + a_0 16^0 + a_{-1}16^{-1} + ... + a_{-m}16^{-m} = \sum_{i=0}^{n-1} a_i 16^i + \sum_{i=1}^{m} a_{-i}16^{-i}$, ahol minden lehetséges i-re $a_i \in \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ \mathrm{A},\ \mathrm{B},\ \mathrm{C},\ \mathrm{D},\ \mathrm{E},\ \mathrm{F}\}$. Az A, B, C, D, E és F jelek tizenhatos rendszerben rendre a 10, 11, 12, 13, 14 és 15 tízes számrendszerbeli számnak megfelelő számot jelentik. Az a szám tizenhatos számrendszerbeli alakja pedig:

$a = a_{n-1}a_{n-2}...a_0, a_{-1}...a_{-m}$. Ha a szám negatív, akkor ugyanúgy járhatunk el, mint a tízes és kettes alapú rendszer esetében láttuk.

Példa egy egyszerű összeadásra a tizenhatos számrendszerben: $11_{(16)} + \mathrm{AF}_{(16)} = \mathrm{C0}_{(16)}$

Tevékenység: Írja fel tizenhatos számrendszerben a következő számokat: 14, 15, 16, 17, 18.
Végezze el tizenhatos számrendszerben a következő összeadási műveleteket: 12 + 5, 12 + 12.

Fontos leszögeznünk, hogy a számok helyiértékes leírása után a kapott jelsorozatból nem lehet egyértelműen eldönteni, hogy az milyen számrendszerben lett megadva (gyakran persze a környezet alapján ez mégis egyértelművé válik). Ezért – sok esetben – a számjegyek mellett fel kell tüntetni valamilyen módon azt is, hogy melyik számrendszert használtuk.

Mi a következőkben – ha szükséges – a szám jegyei után alsóindexben és zárójelben a számrendszer alapját (tízes számrendszerben) megadva jelezzük a használt rendszert. Például: $176_{(16)}$, $A1F_{(16)}$ tizenhatos, $1011_{(2)}$, $10_{(2)}$ kettes, $10_{(10)}$, $176_{(10)}$ tízes számrendszerben megadott számok. Vigyázzunk arra, hogy $10_{(2)}$ nem egyenlő $10_{(10)}$ -zel és $176_{(16)}$ nem egyenlő $176_{(10)}$ -tal.

A hétköznapi életben a tízes számrendszer a megszokott. Az alap nélkül leírt számokat mi is mindig tízes számrendszerben megadott számnak tekintjük, és a tízes alapot csak akkor írjuk ki, ha külön hangsúlyozni szeretnénk azt, hogy a szám ebben a rendszerben van megadva.

Tevékenység: A fentiek alapján gondolja át, hogy hogyan lehet leírni egy számot nyolcas (oktális) számrendszerben.

Számok átírása egyik számrendszerről másik számrendszerre

Ezek után felmerülhet az a kérdés, hogy hogyan kell – ha szükségessé válik – egyik számrendszerben leírt számot egy másik számrendszerbe átírni.

Mivel tetszőleges számrendszerből tízes számrendszerre való átírás az egyszerűbb, először ezzel foglalkozunk. Az átalakításhoz egyszerűen fel kell írni azt az összeget, amelynek rövidítéséből a szám jelsorozatát kaptuk. Azokat a számjegyeket, amelyek a tízes rendszerben nincsenek benne, helyettesíteni kell az értékük tízes rendszerbeli alakjával. Az így kialakult összegben már minden tízes számrendszerben van felírva, és ha elvégezzük a műveleteket az eredmény is tízes számrendszerbeli lesz (és ez éppen a szám tízes számrendszerben megadott alakja).

Példa:

$$A1F_{(16)} \rightarrow A\cdot 16^2+1\cdot 16^1+F\cdot 16^0 \rightarrow 10\cdot 16^2+1\cdot 16^1+15\cdot 16^0=10\cdot 256+1\cdot 16+15\cdot 1=2591_{(10)}$$

$$1011_{(2)} \rightarrow 1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0=8+2+1=11_{(10)}$$

$$AA,8_{(16)} \rightarrow A\cdot 16^1+A\cdot 16^0+8\cdot 16^{-1} \rightarrow 10\cdot 16^1+10\cdot 16^0+8\cdot 16^{-1}=10\cdot 16+10+8\cdot 1/16$$

Tevékenység: Adja meg a $8A3E1_{(16)}$, $726_{(8)}$, $1010101,01_{(2)}$, számok tízes számrendszerbeli alakját!

A feladat megoldása számológéppel, illetőleg számítógéppel (alkalmas programmal) automatizálható, felgyorsítható.

Megemlítjük azt is, hogy a mostanra már széles körben elérhető és elterjedt tudományos számológépek is meg tudnak oldani ilyen típusú feladatokat, de ezekkel a számolást segítő ügyes kis gépekkel jegyzetünkben nem foglalkozunk.

Átváltás kettes számrendszerbe

Az algoritmus népszerű változata az úgynevezett akasztófa módszer. Ennél az eljárásnál az átváltandó számot kell folyamatosan a számrendszer alapjával (jelen esetben kettővel) osztani mindaddig, amíg el nem érjük a nullát. Az osztás hányadosát lefelé egészre kerekítve írjuk az akasztófa bal oldalára, a maradékát pedig a jobbra. Papír-ceruza módszerként alkalmazva:

Hányados	Maradék
2014	0
1007	1
503	1
251	1
125	1
62	0
31	1
15	1
7	1
3	1
1	1
0	0

{á:m1e1a01.png}

1. ábra

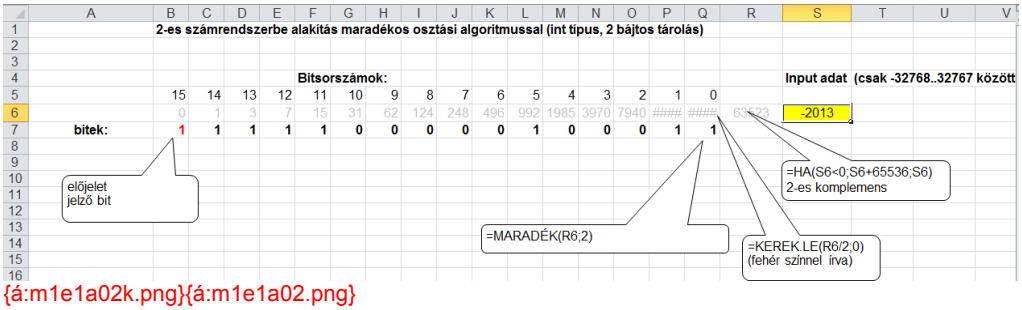
Az akasztófa módszer alkalmazása tízes számrendszerből kettesbe váltáshoz

A keletkezett nullákat és egyeseket alulról felfelé kell leírni a bináris számban.

Így végül: $2014_{(10)} = 011111011110_{(2)}$

Hasonlóan a tízesbe történő átváltáshoz, ez a feladat is automatizálható, felgyorsítható számológéppel, illetőleg számítógéppel (alkalmas programmal). Bemutatunk egy Excel-példát, amely megvalósítja az egyszerű átváltást tízes számrendszerből.

Tevékenység: Töltse le a gyak1.xls fájl, és nyissa meg az Excellel! Tanulmányozza a táblázatot. Az átváltandó szám az S6 (sárga hátterű) cellába írandó!



{á:m1e1a02k.png}{á:m1e1a02.png}

2. ábra

Átváltás tízesből kettes számrendszerbe Excellel

Tevékenység: Számítsa ki a 2015 kettes és tizenhatos számrendszerbeli alakját!

Valamely valós szám tetszőleges A alapról tetszőleges B alapra való átalakítása, ha sem A sem B nem tíz, két menetben történhet. Először alakítsuk át a számot tízes alapú formára, majd ezt alakítsuk tovább B-alapú alakra.

Természetesen a számokkal minden számrendszerben lehet műveleteket végezni. A négy alapművelet elvégzése nem is olyan nehéz, ha felismerjük, hogy ugyanolyan szabályok érvényesek, mint amiket a tízes számrendszerben megismertünk. A problémát legfeljebb ezeknek a szabályoknak az adaptálása okozza. Nehezebb lehet ugyanis a bonyolultabb műveletek hozzáigazítása tetszőleges, de nem tízes alapú számrendszerhez.

Fixpontos számábrázolás

Fixpontos számábrázolásnál a tizedes (kettedes) elválasztójel helye rögzített. Ilyen számábrázolást alkalmaznak például a pénztárgépekben, ahol csak két tizedes jegy pontosan jelenítik meg a fizetendő összeget például: 18325,40 Ft. Természetesen adott mennyiségű számjegy esetében az elválasztójel helye egyszerre van hatással az ábrázolható számok nagyságára és az ábrázolás pontosságára.

Egy tetszőleges b valós szám tízes számrendszerből tetszőleges A-alapú rendszerbe való átírásához nem kell mást tennünk, mint megkeresnünk A azon hatványainak összegét, amely (tízes számrendszerben) b-vel egyenlő. Azaz meg kell állapítani, hogy a

$$b = a_{n-1}A^{n-1} + a_{n-2}A^{n-2} + \ldots + a_0A^0 + a_{-1}A^{-1} + \ldots + a_{-m}A^{-m} = \sum_{i=0}^{n-1} a_iA^i + \sum_{i=1}^{m} a_{-i}A^{-i}$$ egyenlőség

milyen n, m és a_i értékekre teljesül. Ezekről az értékekről csak annyit tudunk, hogy az összes $a_i < A$ és nemnegatív.

Vizsgáljuk először az összeg első felét, azt, amelyben az a_i indexei nemnegatívok. Ez a részösszeg a b-nek A-alapú egész részét adja meg, és látható, hogy az utolsó tag kivételével minden más tag osztható A-val. Mivel $a_0 < A$, ezért az is nyilvánvaló, hogy az összeget – azaz b egészrészét – maradékos osztással A-val elosztva éppen a_0-t kapunk maradékul, a hányados pedig $a_{n-1}A^{n-2} + a_{n-2}A^{n-3} + \ldots + a_2A^1 + a_1$ lesz. Ezt A-val újra elosztva a_1-et kapunk maradékul. Ezeket a lépéseket az újabb és újabb hányadosra elvégezve rendre megkapjuk a többi pozitív indexű a_i-t. Az osztásokat akkor fejezzük be, ha a hányados nulla lesz. Ez éppen az n-edik lépésben következik be, és ekkor az utolsó együtthatót, az a_{n-1}-et is megkapjuk. Így nyilvánvaló, hogy a maradékok rendre az $a_0,\ldots,a_{n-1}$ értékeket adják, azaz a b-t előállító A számrendszerbeli alakot definiáló összeg első fele már felírható.

Az összeg második felét vizsgálva először azt kell megállapítanunk, hogy ez a részösszeg éppen b-nek A-alapú törtrészét adja meg. Vegyük észre, hogy ha ezt az összeget – azaz b törtrészét – A-val szorozzuk, akkor a kapott összeg egy olyan A-alapú számot határoz meg, amelynek egész része éppen a_{-1}, a törtrésze pedig az $a_{-2}A^{-1} + \ldots + a_{-m}A^{-m+1}$ összeg, amely ugyancsak egy törtszám, csak a számjegyeinek száma az A-alapú alakban eggyel kevesebb, mint az előző törtrészé volt. Ezt a törtszámot A-val szorozva az egészrész a_{-2}, a törtrész egy újabb törtszám, amelynek A-alapú alakjában a_{-3}-tól a_{-m}-ig szerepelnek a számjegyek. A szorzásokat addig folytatva, amíg a törtrész

nullává nem válik, megkaphatjuk a b A-alapú alakját megadó összeg összes negatív indexű a_i értékeit és m értékét is, ami nem lesz más, mint a nulla törtrész eléréséhez szükséges szorzások száma. Sajnos előfordulhat, hogy a törtrész sohasem válik nullává. Ilyen esetben a b szám A-alapú alakja végtelen sok számjeggyel leírható törtrészből áll. Ekkor a szorzást addig kell folytatni, amíg elő nem kerül egy olyan törtrész, amelyik már a szorzások eredményeként megjelent. Ettől kezdve ugyanis periodikusan ismétlődni fog minden, a két egyforma törtrész közti szorzat, így a b A-alapú alakja periodikus végtelen törtrészű lesz.

Ezek alapján az átalakítást a következők szerint kell elvégezni:

1. Válasszuk a számot egészrészre és törtrészre.
2. Határozzuk meg az egészrész új számrendszerbeli alakját.
3. Határozzuk meg a törtrész új számrendszerbeli alakját.
4. Vesszővel elválasztva írjuk az egészrész mögé a törtrészt.

Példa: $1848,4_{(10)}$ bináris, oktális és hexadecimális átalakítása.

1. Az egészrész: $1848_{(10)}$, a törtrész $0,4_{(10)}$
2. Az egészrész átalakítása. (A táblázatban a nulladik lépésnél/indexnél a kiindulási egészrészt adtuk meg):

	2		8		16	
index	hányados	maradék	hányados	maradék	hányados	maradék
0	1848(10)	0	1848(10)	0	1848(10)	8
1	924(10)	0	231(10)	7	115(10)	3
2	462(10)	0	28(10)	4	7(10)	7
3	231(10)	1	3(10)	3	0(10)	0
4	115(10)	1	0(10)	0		
5	57(10)	1				
6	28(10)	0				
7	14(10)	0				
8	7(10)	1				
9	3(10)	1				
10	1(10)	1				
11	0(10)	0				

{á:m 1e1s01.png}

Az egészrész alakjai tehát a megadott számrendszerekben: $11100111000_{(2)}$, $3470_{(8)}$, $738_{(16)}$.

3. A törtrész átalakítása. A törtrész átalakítása is hasonló, azzal a különbséggel, hogy hogy nem osztani, hanem szorozni kell, és az egészek közé „felcsúszó” számjegyek adják az átalakított értéket. Az eljárást egészen addig folytatjuk, amíg eredményül nullát nem kapunk, illetve el nem érjük a kívánt pontosságot. A táblázatban csak a szorzási eredmények vannak (az átalakítandó szám törtrésze, amit az első szorzáshoz használunk, nincs!):

	2		8		16	
	szorzat		szorzat		Szorzat	
index	egészrész	törtrész	egészrész	törtrész	egészrész	törtrész
−1	0	,8(10)	3	,2(10)	6	,4(10)
−2	1	,6(10)	1	,6(10)	innen ismétlődik	
−3	1	,2(10)	4	,8(10)		
−4	0	,4(10)	6	,4(10)		
	innen ismétlődik		innen ismétlődik			

{á:m1e1s02.png}

A törtrész új alakjai: $0,\dot{0}11\dot{0}_{(2)}$, $0,\dot{3}14\dot{6}_{(8)}$, $0,\dot{6}_{(16)}$. A számjegyek feletti pontok a periodikusan ismétlődő szakaszt jelölik ki.

4. Az új teljes alakok: $111000111000,\dot{0}11\dot{0}_{(2)}$, $3470,\dot{3}14\dot{6}_{(8)}$, $738,\dot{6}_{(16)}$.

Műveletek nem csak számokkal, a Boole-algebra

A 19. század közepétől a számokkal való műveletvégzés mellett megjelentek más objektumokon végezhető műveletek is, és kialakult az absztrakt algebra. Számítástechnikai szempontból ennek a tudományágnak legfontosabb területe a Boole-algebra, amit George Boole munkássága alapozott meg.

A Boole-algebra egy olyan struktúra, amely egy kételemű (logikai) halmazból és a rajta elvégezhető műveletekből épül fel. A halmaz egyik elemét I(gaz), másik elemét H(amis) értéknek tekintjük, ezeken az értékeken maximum 4 egyoperandusú és 16 kétoperandusú művelet definiálható. Mi ezek közül egy egyoperandusú műveletet és három kétoperandusú művelet fogunk megismerni.

A tárgyalandó egyoperandusú műveletet negációnak nevezzük. A műveleti jele $\neg$ jel legyen. A művelet elvégzése pedig a következő: ha az operandus értéke I, akkor az eredmény legyen H, ha az operandus értéke H, akkor az eredmény I. Formálisan: $\neg\mathrm{I}=\mathrm{H}$, $\neg\mathrm{H}=\mathrm{I}$. A művelet leírását ún. igazságtábla segítségével is megadhatjuk.

operandus: A	**eredmény:** $\neg A$
I	H
H	I

A kétoperandusú műveletek egyikét „és”, a másikat „vagy”, a harmadikat „kizáró vagy” műveletnek nevezzük, a műveleti jelük: $\wedge$, $\vee$, valamint $\neq$. A három művelet igazságtáblája:

1. operandus: A	**2. operandus:** B	**eredmény:** $A\wedge B$	**eredmény:** $A\vee B$	**eredmény:** $A\neq B$
I	I	I	I	H
I	H	H	I	I
H	I	H	I	I
H	H	H	H	H

A logikai műveletek számunkra a későbbiekben felhasználható fontosabb tulajdonságai:

Az „és” valamint a „vagy” művelet:

- asszociatív, azaz $(A \wedge B) \wedge C = A \wedge (B \wedge C) = A \wedge B \wedge C$ és $(A \vee B) \vee C = A \vee (B \vee C) = A \vee B \vee C$;
- kommutatív, azaz $A \wedge B = B \wedge A$ és $A \vee B = B \vee A$.

A „kizáró vagy” kommutatív azaz $(A \neq B) = (B \neq A)$, de nem asszociatív, azaz $((A \neq B) \neq C)$ nem egyenlő $(A \neq (B \neq C))$-vel.

Az „és” művelet disztributív a „vagy” műveletre, illetve a „vagy” az „és”-re, azaz:
$(A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C)$, illetve $(A \wedge B) \vee C = (A \vee C) \wedge (B \vee C)$.

Természetesen ezek csak a legegyszerűbb műveleti tulajdonságok, többre azonban nem lesz most szükségünk. Egy konkrét feladat megoldását tartalmazza a következő fájl: boole-példa.docx

Megjegyzés [M1]: beszúrni ezt a mondatot! + letölthető fájl mellékelve

Kódolás

A kódolás feladata, logikai és szöveges adatok kódolása

Kódolásnak hívjuk – tágabb értelemben – azt a módszert, amely segítségével mások számára is elérhetővé, érthetővé tesszük gondolatainkat. Ilyen kódolás a beszéd, az írás bármilyen formája, így a számok leírása is.

Szűkebb értelemben kódolásról akkor beszélünk, ha szokásos eszközökkel (számjegyekkel, írással, képpel, videóval, hanggal) megadott objektumokat valamilyen egységes rendszerben újra megadunk, leírunk. A továbbiakban a szűkebb értelmű kódolást értjük kódolás alatt.

A következőkben a jelen eszközeinek használatához nélkülözhetetlen kódolásokkal fogunk foglalkozni. Ezeknek a kódolásoknak alapja a kettes számrendszer, mivel eszközeink elektronikus eszközök, és a kettes számrendszer két különböző bitje elektronikus eszközökkel könnyen megjeleníthető vagy könnyen továbbkódolható úgy például, hogy az 1-nek egy magas, míg a 0-nak egy alacsony feszültségszintet feleltetünk meg.

A különböző kódrendszereket vizsgálva megkülönböztethetünk fix hosszúságú és változó hosszúságú kódokból álló kódrendszert. A kódok hosszát a kódban lévő jelek számával szokás definiálni. A számítástechnikában mindkét rendszert használják.

A Boole-algebra objektumainak kódolása a lehető legegyszerűbb kódolás, hiszen csak két elemnek kell kódot választani (I , H). Legfeljebb az okozhat fejtörést, hány jelből álljon a kód, ha ragaszkodunk ahhoz, hogy a kódokat bitekből építjük fel. Egyetlen jelből álló kód esetén például az I objektumot 1-gyel, a H objektumot 0-val kódolhatjuk, 8 bites kód esetén az I kódja a 11111111 jelsorozat, a H -é a 00000000 jelsorozat lehet.

A betűk és egyéb jelek kódolására fix hosszúságú rendszert használunk, a kód hossza általában 8 vagy 16 bináris jel, ritkábban előfordul a 24 és 32 kódhossz is.

Mivel 0 és 1 jelekből 8 hosszúságú kódot 256-féleképpen lehet előállítani, ezért ezzel a kódrendszerrel összesen 256 különböző jel kódolható. Ezeket a jeleket, illetve kódjaikat szabványok határozzák meg. A legelterjedtebb szabványok egyike az ASCII (American Standard Code for Information Interchange = amerikai szabványkód információcseréhez). Az első 128 kód sztenderd,

azaz állandósult jeleket tartalmaz, a második 128 kód az ún. kiterjesztett jelek kódjai. A kiterjesztés többféleképpen is történhet, a bemutatott példa a Latin1 kiterjesztésű változat. A kódrendszert az 1. és 2. táblázat tartalmazza, amelyben a bináris kód mellett megtaláljuk a kód decimális és hexadecimális értékét is (a felesleges nullák nélkül). (A bináris kódban fontosak a kódok kezdő nullái is, mivel ezek nélkül a kód nem lenne egyértelműen visszafejthető!)

Látható, hogy ebben a rendszerben összefüggő tömböt alkotnak az angol ábécé nagybetűi és a kisbetűi, valamint a számjegyek. Megtalálhatók a jelek között az írásjelek, a gyakran használt matematikai jelek, sok ékezetes betű jele és néhány speciális jel is.

Az ASCII rendszerben nem kódolhatók a távol-keleti nyelvek betűi, sőt némely európai ábécé különleges betűje sem. Ennek a problémának a kiküszöbölésére alkották meg az Unicode, az UTF-8 rendszereket, Ezekben a kódok 16, 24, 32, 40, 48 hosszúságú 0 és 1 jelből álló sorozatok. Ezekben a rendszerekben 16 bittel 2562, 24 bittel 2563 különböző jelet lehet kódolni.

Ha jeleket tudunk kódolni, akkor már szöveget is. Ekkor a szöveg betűit, számjegyeit, írásjeleit, betűközeit a jelek kódtáblázatát felhasználva kódoljuk, és a kódokat egymás mögé írva megkaphatjuk a kódolandó szöveg kódját. Vegyük észre, hogy a kapott kód hossza függ a jelek kódhosszától és a szöveg jeleinek számától is. Mivel a jelek száma más és más szövegben általában nem egyforma, ezért a szöveg ilyen kódolása változó hosszúságú kódot eredményez.

DEC	HEX	BIN	Jel	DEC	HEX	BIN	Jel	DEC	HEX	BIN	Jel	DEC	HEX	BIN	Jel
0	0	00000000	NUL	32	20	00100000		64	40	01000000	@	96	60	01100000	`
1	1	00000001	SOH	33	21	00100001	!	65	41	01000001	A	97	61	01100001	a
2	2	00000010	STX	34	22	00100010	"	66	42	01000010	B	98	62	01100010	b
3	3	00000011	ETX	35	23	00100011	#	67	43	01000011	C	99	63	01100011	c
4	4	00000100	EOT	36	24	00100100	$	68	44	01000100	D	100	64	01100100	d
5	5	00000101	ENQ	37	25	00100101	%	69	45	01000101	E	101	65	01100101	e
6	6	00000110	ACK	38	26	00100110	&	70	46	01000110	F	102	66	01100110	f
7	7	00000111	BEL	39	27	00100111	'	71	47	01000111	G	103	67	01100111	g
8	8	00001000	BS	40	28	00101000	(	72	48	01001000	H	104	68	01101000	h
9	9	00001001	HT	41	29	00101001	)	73	49	01001001	I	105	69	01101001	i
10	0A	00001010	LF	42	2A	00101010	*	74	4A	01001010	J	106	6A	01101010	j
11	0B	00001011	VT	43	2B	00101011	+	75	4B	01001011	K	107	6B	01101011	k
12	0C	00001100	FF	44	2C	00101100	,	76	4C	01001100	L	108	6C	01101100	l
13	0D	00001101	CR	45	2D	00101101	-	77	4D	01001101	M	109	6D	01101101	m
14	0E	00001110	SO	46	2E	00101110	.	78	4E	01001110	N	110	6E	01101110	n
15	0F	00001111	SI	47	2F	00101111	/	79	4F	01001111	O	111	6F	01101111	o
16	10	00010000	DLE	48	30	00110000	0	80	50	01010000	P	112	70	01110000	p
17	11	00010001	DC1	49	31	00110001	1	81	51	01010001	Q	113	71	01110001	q
18	12	00010010	DC2	50	32	00110010	2	82	52	01010010	R	114	72	01110010	r
19	13	00010011	DC3	51	33	00110011	3	83	53	01010011	S	115	73	01110011	s
20	14	00010100	DC4	52	34	00110100	4	84	54	01010100	T	116	74	01110100	t
21	15	00010101	NAK	53	35	00110101	5	85	55	01010101	U	117	75	01110101	u
22	16	00010110	SYN	54	36	00110110	6	86	56	01010110	V	118	76	01110110	v
23	17	00010111	ETB	55	37	00110111	7	87	57	01010111	W	119	77	01110111	w
24	18	00011000	CAN	56	38	00111000	8	88	58	01011000	X	120	78	01111000	x
25	19	00011001	EM	57	39	00111001	9	89	59	01011001	Y	121	79	01111001	y
26	1A	00011010	SUB	58	3A	00111010	:	90	5A	01011010	Z	122	7A	01111010	z
27	1B	00011011	ESC	59	3B	00111011	;	91	5B	01011011	[	123	7B	01111011	{
28	1C	00011100	FS	60	3C	00111100	<	92	5C	01011100	\	124	7C	01111100	|
29	1D	00011101	GS	61	3D	00111101	=	93	5D	01011101	]	125	7D	01111101	}
30	1E	00011110	RS	62	3E	00111110	>	94	5E	01011110	^	126	7E	01111110	~
31	1F	00011111	US	63	3F	00111111	?	95	5F	01011111	_	127	7F	01111111	

{á: m1e1t01.png}

1. táblázat

Az ASCII kódrendszer első 128 jele és kódjai

DEC	HEX	BIN	Jel	DEC	HEX	BIN	Jel	DEC	HEX	BIN	Jel	DEC	HEX	BIN	Jel
128	80	10000000	€	160	A0	10100000		192	C0	11000000	A	224	E0	11100000	á
129	81	10000001		161	A1	10100001	¡	193	C1	11000001	A	225	E1	11100001	á
130	82	10000010	‚	162	A2	10100010	¢	194	C2	11000010	A	226	E2	11100010	â
131	83	10000011	ƒ	163	A3	10100011	£	195	C3	11000011	A	227	E3	11100011	ã
132	84	10000100	„	164	A4	10100100	¤	196	C4	11000100	A	228	E4	11100100	ä
133	85	10000101	…	165	A5	10100101	¥	197	C5	11000101	A	229	E5	11100101	ă
134	86	10000110	†	166	A6	10100110	¦	198	C6	11000110	Æ	230	E6	11100110	æ
135	87	10000111	‡	167	A7	10100111	§	199	C7	11000111	Ç	231	E7	11100111	ç
136	88	10001000	ˆ	168	A8	10101000	¨	200	C8	11001000	E	232	E8	11101000	è
137	89	10001001	‰	169	A9	10101001	©	201	C9	11001001	E	233	E9	11101001	é
138	8A	10001010	S	170	AA	10101010	ª	202	CA	11001010	E	234	EA	11101010	ê
139	8B	10001011	‹	171	AB	10101011	«	203	CB	11001011	E	235	EB	11101011	ë
140	8C	10001100	Œ	172	AC	10101100	¬	204	CC	11001100	I	236	EC	11101100	ì
141	8D	10001101		173	AD	10101101		205	CD	11001101	I	237	ED	11101101	í
142	8E	10001110	Z	174	AE	10101110	®	206	CE	11001110	I	238	EE	11101110	î
143	8F	10001111		175	AF	10101111		207	CF	11001111	I	239	EF	11101111	ï
144	90	10010000		176	B0	10110000	°	208	D0	11010000	Đ	240	F0	11110000	đ
145	91	10010001	‘	177	B1	10110001	±	209	D1	11010001	N	241	F1	11110001	ñ
146	92	10010010	’	178	B2	10110010	²	210	D2	11010010	O	242	F2	11110010	ò
147	93	10010011	“	179	B3	10110011	³	211	D3	11010011	O	243	F3	11110011	ó
148	94	10010100	”	180	B4	10110100	´	212	D4	11010100	O	244	F4	11110100	ô
149	95	10010101	•	181	B5	10110101	µ	213	D5	11010101	O	245	F5	11110101	õ
150	96	10010110	–	182	B6	10110110	¶	214	D6	11010110	O	246	F6	11110110	ö
151	97	10010111	—	183	B7	10110111	·	215	D7	11010111	×	247	F7	11110111	÷
152	98	10011000	˜	184	B8	10111000	¸	216	D8	11011000	Ø	248	F8	11111000	ø
153	99	10011001	™	185	B9	10111001	¹	217	D9	11011001	U	249	F9	11111001	ù
154	9A	10011010	š	186	BA	10111010	º	218	DA	11011010	U	250	FA	11111010	ú
155	9B	10011011	›	187	BB	10111011	»	219	DB	11011011	U	251	FB	11111011	û
156	9C	10011100	œ	188	BC	10111100	¼	220	DC	11011100	U	252	FC	11111100	ü
157	9D	10011101		189	BD	10111101	½	221	DD	11011101	Y	253	FD	11111101	ý
158	9E	10011110	ž	190	BE	10111110	¾	222	DE	11011110	Þ	254	FE	11111110	þ
159	9F	10011111	Y	191	BF	10111111	¿	223	DF	11011111	ß	255	FF	11111111	ÿ

{á:m1e1t02.png}

2. táblázat

Az ASCII kódrendszer második (Latin1 kibővítés) 128 jele és kódjai

Számok kódolása

Természetesen az előző részben leírt kódolási módszer számok kódolására is jó, de az így kódolt számokkal a műveletvégzés nehézkes lesz, tehát alkalmasabb rendszert célszerű választani. Sokféle számkódolás lehetséges. A következőkben négyféle módszerrel fogunk foglalkozni.

Nemnegatív egész számok kódolása

Ezeket a számokat számítástechnikában előjeltelen (angolul unsigned) számoknak is szokás nevezni. Kódolásukra 8, 16, 32, 64 bites kódok a legelterjedtebbek. Nyolc bittel 0 és $2^8-1=255$, 16 bittel 0 és $2^{16}-1=65535$, 32 bittel 0 és $2^{32}-1=4294967295$, 64 bittel 0 és $2^{64}-1$ közé eső számokat szokás kódolni. A kód megállapítása nagyon egyszerű. Írjuk fel a kódolandó számot kettes számrendszerben, írjunk eléje annyi nullát, amennyi ahhoz kell, hogy a kód hossza megfelelő legyen. Amit kaptunk az a szám kódja! Fontos megjegyezni, hogy a fenti módszerekkel a megadott intervallumokon kívüli egész számok nem kódolhatók!

Egész számok kódolása kettes komplemens kóddal

Ezeket a számokat számítástechnikában előjeles (angolul signed) számoknak szokás nevezni. Szintén 8, 16, 32, 64 bites kódokat használunk a kódoláshoz. Nyolc bittel a $-2^7=-128$ és

$2^7-1=127$, 16 bittel a $-2^{15}=-32768$ és $2^{15}-1=32767$, 32 bittel a $-2^{31}=-2147483648$ és $2^{31}-1=2147483647$, 64 bittel -2^{63} és $2^{63}-1$ közé eső számokat kódoljuk. A megadott intervallumokon kívül eső számok ezzel a módszerrel nem kódolhatók!

A kódolás a következő (csak a nyolcbites változattal foglalkozunk, a másik három változatban ennek mintájára történik a kódok megállapítása). Ha a szám nemnegatív, akkor – ugyanúgy, mint az előző kódolási módszernél – vesszük a kettes számrendszerbeli alakot, eléje írunk annyi nullát, hogy nyolc jelből álljon, és már készen is van a kód. Ha a szám negatív, akkor hozzáadunk $2^8=256$-ot. Az eredmény 127-nél nagyobb, 256-nál kisebb pozitív szám lesz. Ennek vesszük a kettes számrendszerbeli alakját, és az lesz a negatív számunk kódja. Vegyük észre, a nemnegatív szám kódja mindig nullával, a negatívé mindig eggyel kezdődik, ezért a kód vezető bitje megadja azt is, hogy kód negatív vagy nemnegatív számot jelent-e. Ez a bitet emiatt előjelet jelző bitnek hívjuk, hiszen nem direkt módon az előjel kódolására használjuk (ezért az elterjedt „előjelbit” megnevezés nem pontos).

A negatív szám kódját kettes komplemens kódnak is nevezik.

A kettes komplemens kód megállapítására egy kevesebb számolást igénylő módszert is lehet használni. A módszer azon alapul, hogy a 2^n-1 szám kettes számrendszerben pontosan n darab egyesből áll, nulla nélkül. Ebből a számból könnyű kivonni bármilyen n-nél kevesebb bitből álló pozitív bináris számot, hiszen nem kell mást tenni, csak 0-t írni arra a helyiértékre, ahol a kivonandóban 1 van, és 1-et arra, ahol 0 van. Az eredmény nem más, mint az a szám, amit úgy kapunk, hogy a kivonandóban minden bitet az ellentettjére – 0-t 1-re, 1-et 0-ra – cserélünk, más szóval negáljuk, és annyi 1-gyel kiegészítjük a szám elején, hogy n jegyű legyen. Ez a szám eggyel kisebb, mint a kivonandó –1-szeresének kettes komplemens kódja, ezért a kettes komplemens eléréséhez 1-et hozzá kell adni. Amit kaptunk, az nem más, mint a kivonandó (pozitív) szám -1-szeresének (ez már negatív) kódja.

Példa: Számítsuk ki a -1848 kettes komplemens kódját (16 biten)! Az ún. 2-es komplemenst használjuk a következő módon:

1. Írjuk fel a negatív szám abszolút-értékének bináris alakját!
2. Fordítsunk meg ellenkezőjére minden bitet (negálás)!
3. Egészítsük ki vezető egyesekkel 16 bitre.
4. Adjunk hozzá 1-et!

A megoldás táblázatba foglalva:

1848	111 00111000
negálás	000 11000111
kiegészítés	11111000 11000111
+1	00000000 00000001
−1848 kódja	11111000 11001000

{á:m1e1s03.png}

Érdemes megjegyezni, hogy a fenti módszert mechanikusan alkalmazva a kettes komplemens kódra, a kódhoz tartozó negatív szám abszolút értékének kettes számrendszerbeli alakját, azaz a kódját kapjuk. -1848 esetén:

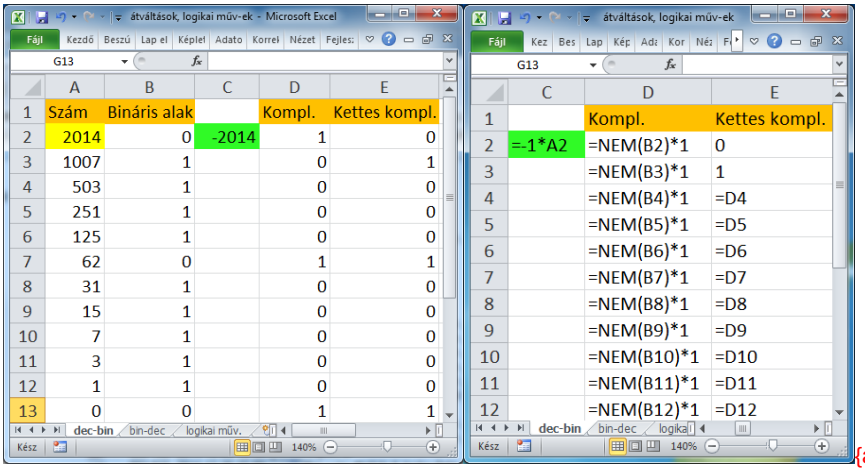
−1848 kódja	11111000 11001000
negálás	00000111 00110111
kiegészítés	nem szükséges
+1	00000000 00000001
1848 kódja	00000111 00111000

{á:m1e1s04.png}

Tevékenység: Számítsa ki a −1848 32 bites komplemens kódját!

Most is bemutatunk egy Excel példát, amely automatizáltan végrehajtja a kettes komplemens kód előállítását. A szám pozitív alakjából kapjuk meg a negatív számot reprezentáló kódot. A Nem függvénnyel megfordítjuk a bináris jegyeket, a „plusz 1” művelethez viszont most egyszerűen kézi átírást használunk.

A korábbiakhoz hasonlóan, erre a feladatra is biztosít az Excel beépített függvényt, de ennek az alkalmazhatósága is korlátozott (lásd később).



	A	B	C	D	E
1	Szám	Bináris alak		Kompl.	Kettes kompl.
2	2014	0	-2014	1	0
3	1007	1		0	1
4	503	1		0	0
5	251	1		0	0
6	125	1		0	0
7	62	0		1	1
8	31	1		0	0
9	15	1		0	0
10	7	1		0	0
11	3	1		0	0
12	1	1		0	0
13	0	0		1	1

	C	D	E
1		Kompl.	Kettes kompl.
2	=-1*A2	=NEM(B2)*1	0
3		=NEM(B3)*1	1
4		=NEM(B4)*1	=D4
5		=NEM(B5)*1	=D5
6		=NEM(B6)*1	=D6
7		=NEM(B7)*1	=D7
8		=NEM(B8)*1	=D8
9		=NEM(B9)*1	=D9
10		=NEM(B10)*1	=D10
11		=NEM(B11)*1	=D11
12		=NEM(B12)*1	=D12

{á:m1e1a03.png}

3. ábra

Kettes komplemens kód előállítása Excellel

Ebben a kódrendszerben két szám összegének kódját rögtön megkaphatjuk, ha a kódokat összeadjuk, tehát nem kell az összeadás elvégzéséhez oda-vissza kódolgatni, a számok összeadása helyett elég a kódokat összeadni. Ezen kívül megspórolhatjuk a kivonást is, ugyanis $a-b=a+(-b)$ tetszőleges a és b esetén, az a és a $-b$ kódjának összege ebben a kódrendszerben éppen az $a-b$ szám kódja lesz. A $-b$ kódja pedig az előbb megismert mechanikus módszerrel könnyedén előállítható.

Jegyezzük meg, hogy ha két szám összege vagy különbsége nagyobb, mint a kódolható számok intervallumának végpontja vagy kisebb, mint a kezdőpontja, akkor a kódok összege nem állít elő kódot, hiszen a kérdéses összeg vagy különbség ezzel a módszerrel nem kódolható!

Példa: $1592-1848=1592+(-1848)=-256$. Kódokkal:

1592	00000110 00111000
−1848 kódja	11111000 11001000
összeg	11111111 00000000

{á:m1e1s05.png}

Tevékenység: Ellenőrizze, hogy az eredmény valóban a -256 kódja-e!

Összetett objektumok (pl. képek) kódolása

Nem célunk, hogy az összetett objektumok néha mélyebb ismereteket is igénylő kódolását és kódrendszereiket ismertessük, de annak érzékeltetéséhez, hogy milyen elveket alkalmaznak egy ilyen eljárás során, bemutatunk egy vázlatos példát.

Legyen egy ilyen objektum egy kép. Az első lépés a kódolásban az, hogy a képet diszkrét egységek rendszerére bontjuk. Ezek az egységek az ún. képpontok (a képpont angolul pixel). Minden képpont egy-egy téglalap alakú képrész lesz. A felbontás annál jobb – és ennek következményeképpen a pontokból összerakott kép annál jobban hasonlít az eredeti képhez –, minél több pontból áll. Ezért szokás a felbontást egy adott hosszúságegységre (inch) eső pontok számával jellemezni. A mértékegység a pont per inch (angolul: Dot Per Inch), röviden DPI. Minél nagyobb a DPI érték, annál nagyobb és jobb a felbontás. Mivel a képek kétdimenziósak, és a vízszintes-függőleges irányú pontra bontás nem szükségképpen egyezik meg, ezért előfordulhat, hogy a vízszintes és a függőleges felbontást szorzat alakban adják meg, ahol az első tényező a vízszintes, a második a függőleges irányú egy inch-re eső pontszámot jelenti.

Az eljárás során tehát egy, a képhez tartozó pontrendszer áll elő. Ennek a rendszernek a pontjait a méretük és a színük jellemzi. A méret a DPI értékből minden pontra egységesen megállapítható, a szín minden pontban más és más lehet.

A kép bitekből álló kódját tehát úgy lehet megadni, hogy megadjuk a bitkódját annak, hogy hány pontból áll a kép vízszintesen, hány pontból függőlegesen, megadjuk a bitkódját a vízszintes és függőleges pontméretnek, majd egyenként kódoljuk a pontok színét is. A bitkódokat egymás után írva egy 0-1 sorozatot kapunk, és ez lesz a kép kódja. Érdemes megemlíteni, hogy ezzel a kódolási mechanizmussal visszakódoláskor az eredeti képnek egy egyszerűbb változatát kapjuk. Ennek megértéséhez elég azt látni, hogy más és más felbontásra más és más kód alakul ki ugyanarról a képről, és a visszaalakítás után a képek csak „hasonlítani” fognak egymásra is és az eredeti képre is.

Azt a folyamatot, amely egy valós, folytonos objektumhoz diszkrét, legtöbbször bináris kódot állít elő, digitalizálásnak nevezzük.

Természetesen ez a leírás egy elnagyolt leírása az egyik lehetséges kódolásnak (legjobban az úgynevezett pixelgrafikus, ill. rasztergrafikus kódoláshoz hasonlít), de ahhoz, hogy a bonyolultabb objektumok bináris kódolását megértsük – elegendő.

A pixelgrafikus tárolással kapcsolatban célszerű vázlatosan megismerni az RGB modellt. Ebben a modellben minden színt három alapszín összetételével állítunk elő. Ezek a színek: vörös (Reed), zöld (Green) kék (Blue). 16 bites színkód esetén a vörös összetevő kódolására 5, a zöldére ugyancsak 5, a kékre 6 bitet szokás használni. Ez 65536-féle szín kódolását teszi lehetővé, ami az emberi szem teljesítőképességét figyelembe véve egy hétköznapi kép kódolására bőségesen elegendő. Sok ember a képpontok színének 8 bittel (256 féle szín) való kódolásakor kapott kódból visszaállított képről sem tudja megmondani, hogy mi a különbség az eredeti és a visszaállított kép között.

A képpontokhoz tartozó direkt kódok mellett a kép kódjában általában még bizonyos kiegészítő vagy járulékos információt is kódolnak (pl. hibajavításra). Ez a kód végső hosszát, méretét csak minimális mértékben megnöveli.

A pixelgrafikus kódolást elsősorban fényképszerű képek kódolására használják. A vonalakból összerakott képek kódolására nem célszerű módszer. Az ilyen típusú ábrákat matematikai módszerek – vektorok – segítségével lényegesen rövidebb kóddal lehet leírni (vektorgrafikus kódolás).

teszt rész

Önellenőrző kérdések

1. Számítsa át tízes számrendszerre a következő előjeles számokat!

$AF5_{(16)}$ = 2805

$123_{(4)}$ = 27

$001010_{(2)}$ = 10

$1100101_{(2)}$ = -27

2. Alakítsa át a következő tízes számrendszerbeli számokat előjel nélküli 16 bites bináris alakba!

$413_{(10)}$ = 0000000110011101

$284_{(10)}$ = 0000000100011100

3. Alakítsa át a következő tízes számrendszerbeli számokat előjeles 16 bites bináris alakba!

$-413_{(10)}$ = 1111111001100011

$-284_{(10)}$ = 1111111011100100

4. Válassza ki a következő bináris számok közül a párosokat!

☐ 111010111

☑ 011100010

☑ 011011010

☐ 011110111

☑ 100010010

5. Döntse el az alábbi állításokról, hogy igaz vagy hamis!

- × Tízes számrendszerben leírt, véges sok tizedesjegyet tartalmazó szám kettes számrendszerben is véges sok számjegyet tartalmazó törtrészből áll.
- ✓ Kettes számrendszerben leírt szám, amelynek törtrésze véges sok számjegyet tartalmaz, tízes számrendszerben is véges sok tizedesjegyű lesz.
- × Tízes számrendszerben véges sok számjeggyel leírt szám kettes számrendszerben is véges sok számjeggyel leírható.
- ✓ Előfordulhat, hogy tízes számrendszerben véges sok számjeggyel leírt szám kettes számrendszerben végtelen sok számjeggyel lesz leírható.

6. Adja meg az ASCII kódját a Nemzeti dal első sorának!

Segítség:

Karakter	ASCII kód
A	65
a	97
szóköz	32
, (vessző)	44
! (felkiáltójel)	33

Karakter:	T	a	l	p	r	a		m	a	g	y	a	r	,
Kód:	84	97	108	112	114	97	32	109	97	103	121	97	114	44
Karakter:		h	í		a		h	a	z	a	!			
Kód:	32	104	237	32	97	32	104	97	122	97	33			

7. Döntse el az alábbi állításokról, hogy igaz vagy hamis!

- ✓ Minden egész számként kódolható szám kódolható valós számként is.
- ✓ A Boole-algebra elemeit egy bittel is lehet kódolni.
- ✓ Nem minden valós számot lehet valós számként kódolni.
- × Minden negatív szám kódolható egész számként.
- × Véges sok olyan valós szám van, amelynek kódja ugyanaz.

8. Döntse el az alábbi állításokról, hogy igaz vagy hamis!

- ✓ Van olyan szám amelynek nemnegatív számként megállapított kódja ugyanaz, mint az egész számként megállapított kód.
- ✓ Van olyan szám amelynek valamilyen kódja éppen az a betű ASCII kódjával egyenlő.
- × Van olyan szám amelynek valós számként megállapított kódja megegyezik az egész számként megállapított kódjával.

9. Adja meg a C változó értékét!

A = IGAZ, B = HAMIS

$$C = \neg\left(A \wedge B \vee \left(B \neq B\right)\right)$$

C = IGAZ

10. Adja meg a C változó értékét!

A = IGAZ, B = HAMIS

$$C = \neg\left(\neg A \vee \neg B\right) = \neg\left(\neg A \wedge \neg B\right)$$

C = IGAZ

11. Adja meg a D változó értékét!

A = IGAZ, B = HAMIS, C = IGAZ

$$D = \neg C \neq A \vee B \vee \left(A \wedge \neg A\right) \neq \left(C \neq B\right)$$

D = HAMIS

2. lecke: Hardver

Cél: Ebben a leckében az a célunk, hogy alapvető képet adjunk a számítógépek belső működéséről. Ehhez szükséges bizonyos történelmi áttekintés, amelynek leghangsúlyosabb része a Neumann-elvek tárgyalása. Ezután röviden ismertetjük a Neumann-elvű számítógép fontosabb alkotóelemeit, és azok működését. Bemutatjuk a számítógép-generációkat, megismerkedünk az egyes korszakok legfontosabb újításaival. Az áttekintés végén külön foglalkozunk a személyi számítógépekkel is. Ez az alfejezet a hardver tananyag egyik különösen fontos része, azonban csak önmagában, a korábbi ismeretek elsajátítása nélkül nem dolgozható fel.

Az elméleti ismeretek alkalmazását egy kis, egyszerű CPU-szimulátor programmal támogatjuk meg, amellyel nagyon egyszerű programozási feladatokat is megoldunk. Ez a rész tekinthető úgy is, mint a 2. féléves Informatika II. tárgy (Számítási módszerek) programozási részének (VBA) egyfajta megalapozása.

Fontos kiemelni, hogy a lecke egyes részei olvasmányos jellegűek, és direkt módon nem is kerülnek számonkérésre (elsősorban a történeti részek). Annak ellenére, hogy ez megnöveli az anyag terjedelmét, úgy gondoltuk, hogy teljesen nem tekinthetünk el a fejlődési folyamat bizonyos mértékű leíró bemutatásától.

Követelmények: Ön akkor sajátította el megfelelően a tananyagot, ha

- saját szavaival össze tudja foglalni az EDVAC gép kifejlesztéséig tartó út fontosabb állomásait;
- saját szavaival ismertetni tudja, hogy milyen tulajdonságok teljesülését várta el Neumann János az elektronikus számítógéptől;
- saját szavaival be tudja mutatni a Neumann-architektúrájú számítógép fő komponenseit, ismertetni tudja ezek főbb jellemezőit;
- nagyság szerint sorba tudja rendezni a bináris, illetve decimális rendszer szerint megadott memória-kapacitás értékeket;
- saját szavaival be tudja mutatni az egyes számítógép generációk jellegzetességeit (jellemző technikai megoldások, újdonságok);
- saját szavaival ismertetni tudja a PC-kben és a laptopokban alkalmazott főbb technikai megoldásokat;
- meg tud oldani egyszerű feladatokat a Hymn szimulátor program segítségével.

Időszükséglet: A tananyag elsajátításához (a feladatok megoldásával együtt) hozzávetőlegesen 4 órára lesz szüksége.

Kulcsfogalmak

- Elektronikus számítógép, Neumann-elvek.
- A Neumann-elvű számítógép fő részei: CPU (CU, ALU, regisztertömb), memória, buszrendszer, órajel-generátor, input-output eszközök.
- Memóriakapacitások mértékegységei.
- Számítógép-generációk, hardver és szoftver újítások (például: memóriatípusok, megszakításkezelés, mikroprocesszor).
- Moore törvény.

- A párhuzamos végrehajtás lehetőségei (pipeline technika, szuperskalár processzorok, vektorprocesszorok).
- Hymn CPU szimulátor.

Út a modern számítógépekig

A számítógépek fejlődésének vázlatos története 1940-ig

Minden művelt embernek fontos tisztán látnia, hogy a mostani modern számítástechnikai eszközök kifejlesztéséig nagyon hosszú és gyakran rögös út vezetett. Feltétlenül helyes, ha megemlékezünk azokról a neves kutatókról, felfedezőkről is, akik sokszor éveket, évtizedeket áldoztak az életükből azért, hogy egy-egy újítással közelebb kerüljön az emberiség a számológép, számítógép mai – kényelmes – formájához.

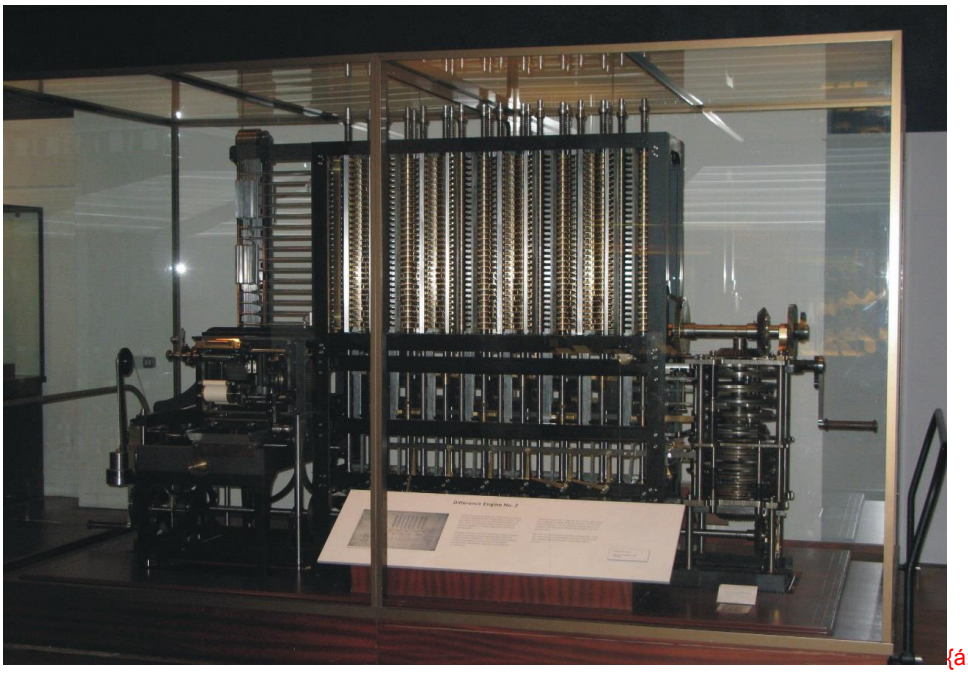
Az első, számolást segítő eszközök megjelenése még az őskor idejére tehető, és a későbbi – izgalmas – fejlődés rendkívül szerteágazó. Hasonlóan a számolás fejlődésének történetéhez (előző lecke), általában most is csak nagyon vázlatosan tudjuk ezen érdekes periódus főbb mérföldköveit bemutatni. Az elektromechanikus számítógépek megjelenésével kezdődő, és az EDVAC kifejlesztéséig tartó rendkívül érdekes és fontos időszakról ugyanakkor részletesebb áttekintést adunk.

1. felsorolás: a számolóeszközök fejlődése (az 1940 utáni időszakot is itt tárgyaljuk)

- A legrégebbi – régészek által megtalált – számolóeszközök megjelenése (vonalkák, pontrendszerek; Afrika, Kr. e. 20 000 körül).
- Elterjed a szorobán és az abakusz (Kr. e. 2000 tájékán); ezek az eszközök ma is használatosak.
- A tudomány és technika fejlődésével először jelentkezik komoly, feszítő igény nagyobb mennyiségű számítások gyors (kézi számításnál gyorsabb) elvégzésére (csillagászati és hajózással kapcsolatos táblázatok, 16-17. század).
- Új elméleti eredmény: logaritmus (a szorzás összeadásra, az osztás kivonásra vezethető vissza, táblázatok, John Napier, 1600-as évek eleje); erre épülve új számolási segédeszköz: Napier-csontocskák (a logarléc őse).
- Az első ténylegesen elkészült mechanikus számolóberendezés: összeadás, kivonás teljesen automatikusan, szorzás, osztás a kezelő segítségével (Wilhelm Schickard, 1620-as évek).
- „Pascaline" mechanikus számológép forgó alkatrészekkel, fogaskerekekkel: összeadás, kivonás közvetlenül, szorzás és osztás ismételt összeadással, illetve kivonással, több tucat példány (Blaise Pascal, 1640-es évek eleje).
- A Pascal-gép továbbfejlesztett változata (Leibniz-kerékkel): itt már a szorzás és az osztás elvégzése is teljesen automatikus, a Leibniz-kerék a 19. század végéig része marad a mechanikus számológépeknek (Wilhelm Gottfried Leibniz, 1670-es évek eleje).
- Az első szélesebb körben használható számológépek kifejlesztése, csak segítik a számolást, a felhasználó működteti a gépet, sokszor az eredmények leolvasása sem egyszerű – emiatt gyakori számolási hibák (Thomas de Colmar, ill. Willgodt Odhner; 1850-es, ill. 1870-es évek).
- A mechanikus berendezéseket lassan felváltják az elektromechanikus majd az elektronikus asztali, illetve zsebszámológépek (20. század).
- Fokozatosan az okostelefonok veszik át a zsebszámológépek szerepét (napjainkban).

2. felsorolás: az ős-számítógépek fejlődése (1940-ig)

- Első számítógép „előfutár”: programmal vezérelt gép a szövetgyártásban, amely előre megállapított minta szerint képes elvégezni a szövést, lyukkártyás irányítás (Joseph Marie Jacquard, 1800-as évek eleje).
- Differenciálgép: összeadási műveleteket teljesen automatikusan végrehajtó számológép gőzgép meghajtással, hajózási és csillagászati táblázatok újraszámolására, majdnem teljesen elkészült, működésére a technikai fejletlenség miatt nem volt lehetőség (Charles Babbage, 1830-as évek).
- Analitikus gép (terv): olyan elvek a tervezésben, amelyek a mai modern számítógép-építés során is használatosak. A gép programozható, műveletvégző egység a négy alapművelet elvégzésére, önálló részegység a részeredmények megőrzésére, bemeneti és kimeneti egység. Ha megvalósul, ez lett volna az első olyan gép, amit számítógépnek (computer) nevezhetünk (Charles Babbage, 1840-as évek).



{á:m1e2a01.png}

1. ábra

Babbage differenciálgépének (2. változat) működő rekonstrukciója

- Statisztikai adatok feldolgozásának gépesítése, adatfeldolgozó gép: népszámlálás adatainak feldolgozása, lyukkártyás módszer, a rögzített adatok csoportosítása, a kézi módszerhez szükséges idő töredéke alatt (Herman Hollerith, 1890-es évek – a gépek gyártására vállalkozás, ebből a cégből nőtt ki az IBM).
- Első működőképes mechanikus számítógép, Z1: a mechanikai elemek mozgatására már elektromotorok (Kondrad Zuse, 1938).

Tevékenység: Saját szavaival mutassa be a fejlődés eddig megismert főbb állomásait, a kutatók nevének megemlítésével!

Elektromechanikus számítógépek

A technikai fejlődés a 20. század elejétől felgyorsult, újabb és újabb felfedezések jelentek meg. Ezek lehetővé tették, hogy a mechanikai eszközök egy része elektromechanikussá, majd később elektronikussá váljon. Így történt ez a számítógép esetén is. A jelfogó (relé), majd az elektroncső megjelenése új lendületet adott a számítógép-építésnek.

Az első komoly eredményeket Zuse érte el a már említett Z1 gép elektromechanikus átalakításával: Z2 nevű gépe 1940-ben készült el. Az 1941-ben megépült Z3-ban már mindent megtalálunk, amit a mai modern számítógépek legalapvetőbb tulajdonságaiként elvárunk. Minimális változtatásokkal továbbfejlesztett változata a Z4 1945-ben készült el, és ez volt az első kereskedelmi forgalomba került számítógép.

Az utóbbi két gép bináris számrendszert használt, műveletvégző egységük alkalmas volt valós számok kezelésére is, memóriával rendelkeztek, programozhatók voltak. Fő egységeiket a telefonközpontokban is használt jelfogókból építette Zuse.

Az Egyesült Államokban a már említett IBM cégnél is megindult a számítógépek fejlesztése. 1944-ben befejezték és a Harvard Egyetemre szállították első számítógépüket az IBM Automatic Sequence Controlled Calculatort (ASCC). A tervező Howard Aiken elektromechanikus egységekből – kapcsolókból, relékből – építette gépét, amit a működtető egyetem később Mark I.-nek nevezett el. A gép az Amerikai Tengerészeti Hivatal megrendelésére végzett számításokat. A működését program vezérelte. A programlépések egy szalagra lyukasztott lyukkombinációval voltak kódolva, ezeket olvasta a számítógép, és lépésenként hajtotta végre. A számítások elvégzésére decimális kódolást használt.

Továbbfejlesztett változata a MARK II. 1946–1947-ben készült el, még mindig kapcsolók és jelfogók felhasználásával.

Az első elektronikus számítógépek

Az elektroncsövek modernebb változatai – bár melegedésük és ennek következményeként meghibásodásuk még mindig komoly problémát okozott a számítógép-tervezőknek és -építőknek –, a 20. század negyvenes éveire mégis alkalmassá váltak arra, hogy a számítógépekben alkalmazott jelfogókat helyettesítsék. Így megteremtődött a teljesen elektronikus számítógépek létrehozásának technikai feltétele.

Az elméleti megalapozás is megtörtént ugyanebben az időszakban. Itt csak Alan Turing munkásságát említjük meg, aki az 1930-as években publikált cikkeiben leírta a számítógép olyan elméleti modelljét – az ún. Turing-gépet –, amely mind a mai napig a legteljesebb leírása, ill. modellje az absztrakt számítógépnek. Erre a modellre épül az algoritmus mai értelmezése is, ugyanis minden olyan számítási eljárást, amit Turing-gép képes megvalósítani algoritmusnak nevezünk, és nincs olyan számítási eljárás, amit nem tudnánk Turing-géppel megvalósítani.

Az elektronikus számítógépek kifejlesztésének folyamatát nagy mértékben felgyorsította a második világháború. Az egymással szembenálló felek terveinek (titkos üzenetek) megismerése és megelőzése ugyanis fontos szerepet játszott a háború kimenetelében. A németek az ENIGMA nevű rejtjelező géppel kódolták üzeneteiket, és az így rejtjelezett üzenetet megfejthetetlennek tartották. A kriptoanalízissel foglalkozó angol tudósok – köztük Alan Turinggal –, már 1943–44-ben megépítették és használták a Colossus nevű elektronikus számítógépet, amit az ENIGMA-kódú üzenetek megfejtésére használtak – sikerrel. Több tucatot készítettek belőle, és csak kódfejtésre használták. A

gépet és terveit titokban tartották, és ez a titkosítás oly sikeres volt, hogy a németek még az 1950-es években is használták az ENIGMA-kódot, mert még akkor sem tudtak arról, hogy az angolok meg tudják fejteni.

A gépet lehetett programozni, de ez körülményes volt, mivel kapcsolók, kábelek segítségével lehetett beállítani a programot.

Háborús célok motiválták az amerikai fejlesztőket is. Az Egyesült Államok Hadseregének Ballisztikai Kutatások Laboratóriuma tüzérségi táblázatok számítására alkalmas számítógép tervezését és építését rendelte meg a Pennsylvaniai Egyetem Moore Intézetében (tervezők: John Mauchly és John Eckert). A katonai finanszírozás és felhasználás miatt a tervezés, az építés és a munkába állítás is titokban történt. Csak 1946-ban vált publikussá a projekt, és ekkor ismerhette meg a világ az ENIAC (Electronic Numerical Integrator And Computer, magyarul: Elektronikus Numerikus Integrátor és Számítógép) nevű gépet, amely decimális kódolást használt, programozható volt, bár a programozása a Colossuséhoz hasonlóan kapcsolókkal és kábelek segítségével történt. Elektronikus elemekből építették meg, de minimális mértékben reléket is tartalmazott. Fontos tulajdonsága volt a gépnek, hogy már a tervezéskor általános célokra készült, ami praktikusan azt jelentette, hogy nemcsak az eredetileg tervezett célszámításra volt alkalmas, hanem minden olyan feladatot képes volt megoldani, amire be tudták programozni. Ezt a tulajdonságot Neumann János tanácsadói munkájának eredményeképpen tartják számon. Ő a Los Alamosban zajló atomkísérletekhez szükséges számítások elvégzésére és tesztek kiértékelésére használta a gépet.

Itt megjegyezzük, hogy legelső elektronikus számítógépet is az USA-ban készítették, az Iowai Főiskolán (1937–1942), tudományos célfeladatokra, konkrétan lineáris egyenletrendszerek megoldására. A gép teszteredményei sikeresek voltak, de működése (különösen: a részeredmények tárolása) megbízhatatlan volt. A fejlesztés abbamaradt, és a gépről a tudományos társadalom alig tudott valamit egészen az 1960-as évekig.

Még az ENIAC-projekt befejezése előtt a gép tervezői – ugyancsak az Egyesült Államok Hadserege és a Pennsylvaniai Egyetem Moore Intézete közötti szerződés keretében – 1944 augusztusában új fejlesztésbe kezdtek. Munkájukhoz tanácsadóként csatlakozott Neumann János is. Céljuk egy új számítógép, az EDVAC (Electronic Discrete Variable Automatic Computer, magyarul: Elektronikus Diszkrét Változású Automatikus Számítógép) megtervezése és megépítése volt. Az ENIAC kifejlesztése során nyert tapasztalatok alapján egy teljesen elektronikus, belső programvezérlésű számítógép megépítésébe fogtak, amelyet 1948-ban már elkészítettek, de csak 1949-ben került a Ballisztikai Kutató Laboratórium birtokába. Elődjétől elsősorban abban különbözött, hogy bináris kódolású volt, nem voltak elektromechanikus alkatrészei, és a működését meghatározó programot a tárolóegységében kódolva tárolta.

Tevékenység: Saját szavaival foglalja össze, hogy mit tanultunk az elektromechanikus gépekről és az első elektronikus számítógépekről!

Neumann a tervezés eredményeit, illetve saját javaslatait az 1945. júniusi dátumú **„First Draft of a Report on the EDVAC”** (Az EDVAC-jelentés első vázlata) című tanulmányában foglalta össze. A jelentésben egy olyan számítógépet ír le, amely évtizedeken át szinte kizárólagosan meghatározta a számítógép-tervezés és -építés irányelveit (a mai számítógépek működésében is nagyon fontos szerepe van ennek az elvnek). Megjegyzendő, hogy a Neumann által leírt gép soha nem épült meg. A ténylegesen elkészült EDVAC, ha nem is sokkal, de eltér a vázlatban megadott számítógéptől.

A Neumann-elvek

Az EDVAC-jelentésben Neumann a számítógépének nemcsak a tervét adta meg, hanem azt is megfogalmazta (megfelelő indoklással), hogy mit vár el egy számítógéptől.

1. A gép digitális legyen, azaz diszkrét jeleket tudjon kezelni. Ezek a jelek számjegyeket, mégpedig bináris számjegyeket jelentsenek.
2. A számítógép teljesen elektronikus legyen, azaz a fő részegységei ne tartalmazzanak mozgó (mechanikus) alkatrészeket.
3. A gép a következő részegységekből épüljön fel:
    a) Legyen egy aritmetikai műveletek elvégzésére alkalmas egysége. Ezt nevezhetjük központi aritmetikai egységnek, Neumann elnevezése szerint „Central Arithmetical part”, azaz CA.
    b) Legyen egy, a gép működését vezérlő egység, ami egymás után hajtja végre az utasításokat. Az egymásutániság azt jelenti, hogy a sorra kerülő utasítással addig ne foglalkozzon a vezérlőegység, amíg az éppen feldolgozás alatt lévőt be nem fejezte. (Ennek a tulajdonságnak a megkövetelése a kor technikai lehetőségei miatt történhetett, nem pedig elvi megfontolásból, hiszen Neumann alig pár év múlva a párhuzamos végrehajtást, mint elvi lehetőséget is vizsgálta.) Az egységnek neve „Central Control”, azaz CC. A CC és a CA egység együtt a vezérlő (Control), röviden C.
    c) Legyen egy olyan egysége, amelyben tárolni lehet a működést meghatározó utasításoknak, valamint a számoláshoz szükséges adatoknak, részeredményeknek a kódjait. (Természetesen digitálisan!) Neve „Memory”, azaz M.
    d) A gépnek legyen egy olyan részegysége, amely képes a környezettel kommunikálni. A környezetből kapott jelek és az oda küldött jelek az R nevű egységcsoporton (outside Recording medium, azaz külső adathordozó) találhatók (kódolva). A külső adathordozókat soroljuk két csoportba. A bemenő jeleket kezelő egységeket nevezzük I-nek (Input, azaz bemenet), a kimenő jeleket (kódjaikat) tartalmazó egységeket pedig O-nak (Output, azaz kimenet).
    e) A fenti elemek működését szinkronizálandó legyen „órája” a gépnek, amely a szinkronizációhoz az úgynevezett órajelet generálja. A részegység által előállított jel segítségével oldható meg, hogy a CC működésében az utasítások végrehajtási lépései ne keveredjenek.
4. A gép működtetése bináris kódokkal leírt utasítások végrehajtásával történjen, amiket a memória tartalmaz. A memóriában lévő utasítások cserélhetők, így a gép minden olyan feladat végrehajtására képes, aminek megoldásához utasítássorozatot lehet készíteni. (Ez azt jelenti, hogy a tervezett berendezést Neumann univerzális, belső (program)vezérlésű számítógépként képzelte el.)
5. A memória mellett a C egység részeként legyenek a gépben olyan átmeneti tárolók, amelyek az utasításokhoz kapcsolható kódokat, az elvégzendő műveletek operandusainak kódjait és az eredmény kódját tartalmazzák.

Neumann részletesen leírja a tároláshoz használt kódolást is. Egyértelműen a bináris kódok mellett teszi le a voksot. Az egész számok kódolására a kettes komplemens kódot javasolja, megmutatva annak előnyeit. Megadja a valós számok egyfajta normalizált rendszerű kódolását is, amely hasonlít a már megismert IEEE szabványok által javasolt kódolásokhoz.

Felsorolja a CA által mindenképpen elvégzendő műveleteket. Ezek: összeadás, kivonás, szorzás és egészosztás. Felsorol még néhány más műveletet is, amelyek elvégezhetőségét fontosnak tartja. Megtervezi a CA elektronikus összetevőit, és megadja működési elvüket is.

Megadja azt is, hogyan legyen megszervezve a memória.

Végül kitér arra is, hogy a definiált egységek kapcsolata hogyan valósuljon meg, hogyan mozogjon az egyes egységek között a kód.

A tanulmány alkalmat adott az utókornak arra, hogy a benne összefoglalt tulajdonságokat Neumann-elveknek, az ezek alapján felépített számítógépeket Neumann-elvű számítógépnek tekintse. Sok kutató azonban Neumann-elvként egyetlen tulajdonságot szokott kiemelni, mégpedig a belső programvezérlés elvét, amit tárolt program néven is szokás emlegetni.

Tevékenység: Foglalja össze röviden (5-6 pontban) a Neumann János által megfogalmazott elveket az elektronikus számítógépek felépítéséről!

bővített normál rész

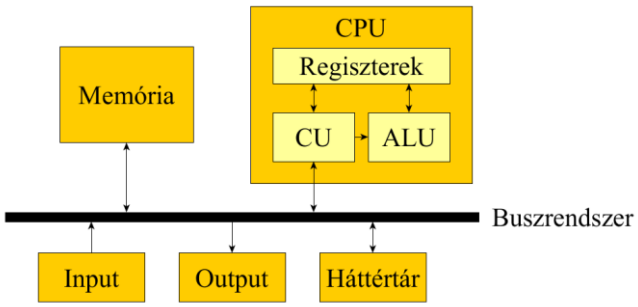
Segítség az alapelvek összefoglalásához:

- Teljesen elektronikus működés: A mechanikus gépek működési sebességét mindig korlátozza a bennük lévő alkatrészek tehetetlensége. Neumann, az addig szokásos mechanikus vagy elektromechanikus megvalósítás helyett, tisztán elektroncsöves megvalósítást javasolt a működési sebesség növeléséhez.
- Kettes számrendszert használata: Rámutatott, hogy a két diszkrét állapottal rendelkező (villanykapcsolóhoz hasonló) elektronikus megvalósításhoz jól illeszkedik a 0 és 1 számjegyeket tartalmazó kettes számrendszer használata.
- Belső memória használata, tárolt program elve: Nagy sebességű belső memória használata a lassú külső adattár helyett, a gép működését leíró programot is memóriában tároljuk.
- Soros utasítás végrehajtás: A program elemi utasításait egymástól időben elkülönülve hajtja végre az gép, emiatt egyszerűbb és egyben olcsóbb lehet.
- Univerzális gép: Képes legyen az algoritmikusan leírható feladatokat megoldani (figyelembe véve a korlátozottan rendelkezésre álló memória méretet).

normál rész

A Neumann-elvű számítógép felépítése és működése

Az 1940-es évek végén lezárult a számítógép-építés történetének őskora. Minden készen állt a nagyüzemi gyártáshoz, és elérkezett az idő, hogy a számítógép, ha lassan is, de a mindennapi élet eszközévé váljon.



{á:m1e2a02.png}

2. ábra
A Neumann-elvű számítógép felépítésének sematikus rajza

Neumann terveihez igazodva, némely praktikus változtatással a számítógép logikai felépítését a 2. ábra tartalmazza. Az építőelemeket, illetve a számítógépet alkotó berendezéseket közös néven hardvernek nevezzük.

A következőkben röviden bemutatjuk az ábrán megjelölt egységek funkcióját, felépítését és működését.

Memória

A memória a számítógép működéséhez szükséges kódokat (adatokat) tárolja. A Neumann elvekhez illeszkedve elektronikus működésű, és kettes számrendszerben megadott kódokat tartalmaz. Legkisebb elemi egysége két stabil állapottal rendelkezik. Ezek az állapotok egymásba átvihetők (külső hatásra). Az egyik állapot 0-nak, a másik 1-nek tekinthető, így az elemi egység egy bit tárolására alkalmas.

Az elemi cellákat, mivel egy bináris számjegy tárolására képesek, szintén bitnek nevezzük. (A bit kétféle jelentését ne keverjük!) A gyakorlatban az elemi cellákat nagyobb tárolási egységekbe szervezik. A leggyakoribb megvalósításnál nyolc elemi cella alkot egy egységet, ezt bájtnak (byte) hívjuk.

A kódolásnál megismert kódrendszerekben nyolc bitnél hosszabb kódokat is használtunk, így célszerű a memóriában a bájtnál nagyobb egységeket is definiálni. Ilyenek a félszó, ami két bájtból, a szó, ami négybájtos, a duplaszó, ami nyolc bájtból áll.

A memória méretét ugyancsak bájtokkal szokás kifejezni. A számítógépek fejlődésével a memóriaméret jócskán megnőtt, a megadáshoz ma már a bájt nagy többszöröseit kell használni. A számítástechnikai váltószám az egyes mértékegységek között az 1024, ami 2^{10}-nel egyenlő. Ugyanakkor a(z) SI (Système International d'Unités = Mértékegységek Nemzetközi Rendszere) rendszer tízes alapú. A megállapodások szerint a számítástechnikában is ezt kellene alkalmazni, de kompromisszumos megoldásként használatos a kettes alapú rendszer is (a hétköznapi életben ugyanakkor gyakran összekeverik a kétféle rendszert – méretek és mértékegységek).

A memória méretei általában a bináris rendszer szerint vannak megadva, a mértékegységet viszont a tízes rendszer szerint használják – helytelenül. (Megabyte helyett mebibyte-ot kellene mondani, illetve az MB helyett MiB jelet kellene használni.) A háttértárak méretét viszont általában SI egységekben adják meg, és ott – helyesen – megabyte, gigabyte, terabyte a mértékegység, MB, GB, TB a jel.

A mértékegység neve bináris rendszerben	Nagysága	Jele
Bit	egy bináris jegy tárolóegysége	b
bájt (byte)	8 bit (8 b)	B
kibibájt (kibibyte)	1024 bájt (1024 B)	KiB
mebibájt (mebibyte)	1024 kibibájt (1024 KiB)	MiB
gibibájt (gibibyte)	1024 mebibájt (1024 MiB)	GiB
tebibájt (tebibyte)	1024 gibibájt (1024 GiB)	TiB
pebibájt (pebibyte)	1024 tebibájt (1024 TiB)	PiB

1. táblázat

A memóriakapacitás bináris rendszere

új táblázat

A mértékegység neve SI-ben	Nagysága SI-ben	Jele
Bit	egy bináris jegy tárolóegysége	b
bájt (byte)	8 bit (8 b)	B
kilobájt (kilobyte)	1000 bájt (1000 B)	kB
megabájt (megabyte)	1000 kilobájt (1000 kB)	MB
gigabájt (gigabyte)	1000 megabájt (1000 MB)	GB
terabájt (terabyte)	1000 gigabájt (1000 GB)	TB
petabájt (petabyte)	1000 terabájt (1000 TB)	PB

2. táblázat

A memóriakapacitás SI rendszere

Az 1950-es évek számítógépeinek memóriamérete legfeljebb párszáz bájt volt, manapság a memória nagyságát gigabájtokban (helyesen gibibyte lenne) mérik.

Tevékenység: Becsülje meg, hogy 1 TiB hányszorosa 1 TB-nek!

Tevékenység: Jegyezze meg a táblázatokban szereplő mértékegységeket, és végezzen további hasonlításokat is különböző mennyiségek választásával!

A tárolóegységeket sorszámozással azonosítják (nullától kezdve). A bájtszervezésű memóriában bájtok, a szószervezésűben a szavak kapják a sorszámot. A sorszámokat bináris rendszerben többféle módszerrel is lehet kódolni. Az egyik ilyen módszer a direkt címkódolás, amelyben a címeket egyszerűen kettes számrendszerbe megadva kódolják. Fix hosszúságú kód esetén a kettes számrendszerbeli alak elé – ha szükséges – extra nullákat kell írni. Ha a nagy memória miatt ez a kód túl hosszú lenne (például 16 bitnél hosszabb), akkor másféle kódolással a cím kódja rövidíthető.

A memória szerepe: a kódok (adatok) tárolása. Neumann szerint ezek a kódok jelenthetik a számítógépet vezérlő utasításokat, de jelenthetik a számoláshoz szükséges adatokat is. Magáról a kódról – önmagában – általában nem eldönthető, hogy mit jelent, ezt a számítógép állapota határozza meg.

A memóriát kezdetben elektroncsövekből, később mágnesezhető apró gyűrűkből (ferritgyűrű) stb. építették. Később kizárólagossá vált az integrált áramkörös megvalósítás.

A programkódot is tartalmazó memóriát operatív memóriának szokás nevezni.

CPU

A CPU-t (Central Processing Unit, Központi Végrehajtó Egység; Neumann-terminológia szerint C) Neumann két fontos részből állónak tekintette. Célszerűnek látszik e két egység (CU – neumanni terminológia szerint CC – és CA) mellett kiemelni, és harmadik fontos összetevőként kezelni a CPU néhány bájtból álló tárolóegységét, a regisztertömböt is.

Mivel Neumann csak numerikus értékekkel végzendő számolásra gondolt, amikor a számítógépet tervezte, ezért a számolómű nevében is csak az aritmetikai jelzőt szerepeltette. Pár éven belül azonban már megjelentek olyan számítógépek, amelyek logikai műveletek elvégzésére is képesek voltak, ezért mi a CA helyett az ALU (Arithmetic and Logic Unit) megnevezést fogjuk használni.

A CPU tehát a CU és az ALU egységből áll, amik elérik a CPU részét képező regisztertömböt is.

A regisztertömb speciális tárolóegységek rendszere. A tárolóegységek mérete a tárolt kódok nagyságának, a memóriaszervezésnek, a műveletvégző egységnek paramétereitől függ, egy bájtnál nem rövidebbek, de a mai számítógépekben lehetnek akár nyolcbájtosak is.

Néhány fontosabb egység a regisztertömbben:

- Utasításregiszter, röviden IR (Instruction Register): az éppen végrehajtás alatt álló utasítás kódját tartalmazza.
- Utasításszámláló regiszter, röviden PC (Program Counter): egy memóriacím kódját tartalmazza. Aktuális értéke alapján kerül a megfelelő utasítás az utasításregiszterbe. (Ez a PC rövidítés nem tévesztendő össze a personal computer PC rövidítésével!)
- Akkumulátorregiszter, röviden A (Accumulator): Az ALU által előállított eredmény kódját tartalmazza.
- Státuszregiszter, röviden SR (Status Register): minden bitje más és más információt kódol. Egyik bitjéből azt lehet kiolvasni, hogy a művelet eredménye negatív-e vagy sem, a másikból azt, hogy nulla-e vagy sem, a harmadikból azt, hogy az eredmény beletartozik-e a kódolható tartományba vagy sem, stb.

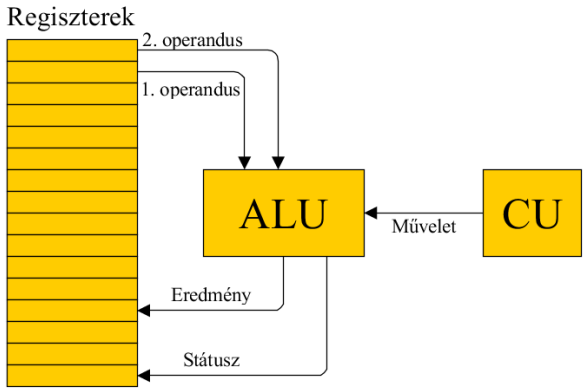
Mivel ezt a tárolót szinte folyamatosan használja a CU és az ALU, ezért gyorsnak és megbízhatónak kell lennie.

Az ALU azokat a műveleteket végzi, amelyek elvégzésére a CU-tól utasítást kap. A műveletekhez szükséges adatok kódjait szintén a CU készíti elő a regisztertömb megfelelő regisztereinek kitöltésével, és valamelyik regiszterbe kerül az eredmény is (a művelet elvégzése után). A folyamat során a státuszregiszterbe is kerülhetnek bitek.

Az ALU által elvégezhető műveleteket a következő csoportokba sorolhatjuk:

- Aritmetikai műveletek (lehetnek egész számok kódjaival elvégezhetők – fixpontos műveletek, és valós számok kódjaival elvégezhetők – lebegőpontos műveletek).
- Logikai műveletek. Ezeket a műveleteket ismertük meg a Boole-algebra című fejezetben.
- Léptető műveletek. Ezek a kódok bitjeit jobbra vagy balra tolják úgy, hogy a kódból kieső bitek a kódból elvesznek, az eltoltak helyébe nulla vagy egy kerül, illetve lehet az eltolás ciklikus is, ekkor a kipottyanó bit a kód másik végén jelenik meg.

Minél több és összetettebb művelet elvégzésére képes az ALU, annál bonyolultabb áramkörökből épül fel, és természetesen annál drágább is. Napjainkban választható az egyszerűbb, olcsóbb ALU (és így több művelet ugyanannak a számításnak az elvégzésére), vagy a bonyolultabb, drágább ALU (kevesebb műveletvégzéssel); régebben viszont még csak az egyszerű ALU-k léteztek.



{á:m1e2a03.png}

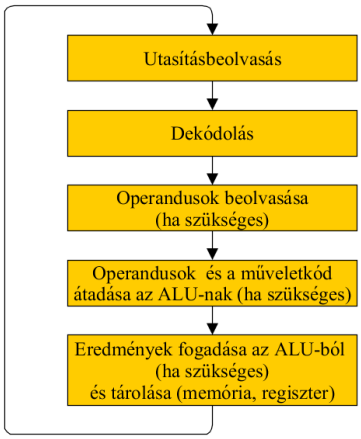
3. ábra

Egy kétoperandusú művelet elvégzésére beállított ALU működésének vázlata

A Control Unit (vezérlőegység) a számítógép alapvető fontosságú részegysége. Működése során három egyszerű lépést ismétel ciklikusan, ezek a következők: elérés (fetch), dekódolás (decode), végrehajtás (execute).

A vezérlőegység elsőként a memóriából, a PC regiszterben lévő címkód alapján beolvas egy utasításkódot, ami az utasításregiszterbe (IR) kerül. Miután beolvasta, értelmezi azt, vagyis eldönti, hogy mit kell csinálnia, végrehajtja – ha szükséges az ALU műveletvégző képességét is felhasználva – a felismert utasítást, az eredményt – ha az utasítás alapján erre szükség van – a memóriába juttatja, módosítja a regisztertömb megfelelő regisztereit, majd a következő utasításkód beolvasásával egy újabb ciklus kezdődik.

A folyamat pontosabb megértéséhez célszerű megismernünk a (gépi) utasítás fogalmát. Egy gépi utasítás nem más, mint egy elemi instrukció a CPU számára, amit az felismer, és pontosan úgy működik, ahogyan az instrukció szerint működnie kell. Természetesen egy adott CPU-hoz több különböző instrukció, gépi utasítás tartozik. Ezeknek az utasításoknak az összességét utasításkészletnek (instrucion set) nevezzük. Ez a készlet a CPU jellemző tulajdonsága, így más-más CPU-nak más-más utasításkészlete van.



{á:m1e2a04.png}

4. ábra
A CU működésének vázlata

Az utasítások binárisan vannak kódolva, ez a kód általában egy állandó és egy változtatható részből tevődik össze. Az állandó résszel lehet megadni, hogy milyen utasítást kódolt a jelsorozat, a változtatható rész adja meg azt, hogy a végrehajtáskor mik lesznek az esetleges műveletvégzés operandusai (operandus nélküli utasításnál ez a rész elmarad). Kezdetben csak ezt a kódot használták az utasítás megadására, később minden utasításnak rövid nevet (mnemonikot) adtak, így könnyebb volt őket megjegyezni, és könnyebb volt összeállítani a CPU egységet irányító utasítássorozatot is.

A gépi utasításokat a következő csoportokba célszerű sorolni.

- Memóriakezelő utasítások (regiszter vagy memória olvasása, írása, másolása stb.).
- Aritmetikai és logikai utasítások (összeadás, kivonás, osztás, szorzás; bitmanipulációs műveletek, például: eltolás, negáció; összehasonlítás, például: kisebb, nagyobb, egyenlő stb.).
- Vezérlőutasítások (feltételes ugrás, feltétel nélküli ugrás stb.).
- Egyéb utasítások (például energiagazdálkodást szabályzó utasítások, amelyek a mai modern számítógépeken már megtalálhatók).

Nézzük meg egy példán, hogy hogyan is történik a gépi utasítások végrehajtása. Tegyük fel, hogy valamilyen elképzelt számítógép memóriája bájtszervezésű, mérete 64 KiB, az adatok 8 bittel kódolt nemnegatív számok, és a processzor gépi utasításainak kódhossza egy- vagy néhány bájtos lehet. Ismeri az összeadást végző utasítást, aminek végrehajtásakor a CPU a CU és az ALU összehangolt működésével összead két értéket. Az összeadandók kódját vagy a kódok tárolóegységét az utasításban megadható két operandussal lehet kijelölni, és az eredmény az első operandusnak megfelelő helyen tárolódik.

Az utasítás mnemonikus formája a következő: ADD op1,op2.

Most tegyük fel, hogy az op1 egy memóriacím, az op2 pedig egy konstans. A cím és a konstans megkülönböztetéséhez a konstans elé tett aposztrófot használjuk.

A konkrét utasítás legyen: ADD 1024,'32.

Ezt binárisan 32 bittel kódolva a következő felépítésű kódot kapjuk: az első bájt az utasításkód, a második és harmadik bájt a direkt memóriacím, az utolsó bájt a konstans kódja lesz. Az utasításkód tehát négybájtos, amiben az első bájton a művelet és az operandusok típusa van kódolva. Esetünkben konkrétan a következő: 10000010 0000000100000000 00100000.

Amikor a gép végrehajtja az utasítást, ez a kód az IR regiszterbe jut. Az első nyolc bitből a CU megállapítja, hogy a művelet összeadás, hogy memóriában lévő kód az első operandus, megállapítja, hogy a második operandus kódja az utolsó bájtról olvasható le, és azt is, hogy az eredményt oda kell visszaírnia, ahonnan az első operandus kódját kapja. A PC regiszter értékét – mivel a feldolgozás alatt lévő utasítás négybájtos – 4-gyel megnöveli.

Ezután a 0000000100000000 (=1024) címen lévő kódot az ALU első operandusát tartalmazó regiszterbe, a 00100000 kódot a második operandust tartalmazó regiszterbe tölti, majd utasítást ad az ALU-nak az összeadás elvégzésére. Az eredmény szintén egy regiszterbe kerül.

A negyedik lépésben az eredményregiszter tartalmát (ez egy kód lesz!) a 0000000100000000 címre írja, beállítja a SR-t, majd elkezdi a következő ciklust.

A CPU működési sebességét az egy másodperc alatt végrehajtott utasítások számával mérik. Mértékegysége az IPS (az instructions per second rövidítése). Ennek ezerszerese a kIPS, milliószorosa az MIPS.

A fenti utasítás-végrehajtás példája jól mutatja, hogy hogyan is működik a CPU. Egy logikai szinttel feljebb lépve, a következőkben ezekből az utasításokból egy szerves egységet kell képezni abból a célból, hogy a CPU pontosan úgy működjön, ahogyan a felhasználója szeretné (működtető program).

Gépi programnak vagy másképpen gépi kódú programnak nevezzük azt az utasításkód-sorozatot, amit a számítógép memóriájába betölthetünk, és a betöltés után ezek az utasítások vezérlik, irányítják a számítógép CPU egységét, és ezáltal a számítógépet. Ennek a munkának az elvégzői, azaz a programok készítői a programozók.

A gépi kódokkal programot írni komoly munkát jelentett. Ennek a megkönnyítésére, a gépi kódokat helyettesítő mnemonikok és egyéb más szimbolikus jelek bevezetése az első és egyben legegyszerűbb programíró vagy más szóval programozási nyelv megalkotásához vezetett. Ezt a nyelvet assembly-nyelvnek nevezték, és annyi „nyelvjárása” volt (van), ahányféle CPU létezett. Az assembly nyelven megírt programot a számítógép már nem tudta közvetlenül felhasználni, ezért át kellett alakítani a gépi nyelvre. Az átalakításra programokat használtak. Ezek voltak az assemblerek.

A programozók munkáját tovább egyszerűsítette a magas szintű programozási nyelvek megjelenése. Ezzel a programozók munkája jelentősen leegyszerűsödött, hiszen egy ilyen nyelvi környezetben az írott nyelvekhez hasonlóan lehet megfogalmazni a programokat. Természetesen a magas szintű nyelven megírt programot sem képes a számítógép megérteni, ezért ezeket is át kell alakítani gépi kódra. Az átalakítást ugyancsak programok végzik, amiket fordítóprogramoknak hívunk. Manapság számtalan magas szintű programozási nyelv létezik.

Tevékenység: Keressen az interneten egy egyszerű C programot! Próbálja meg értelmezni a program sorait és működését!

A Számítási módszerek (Informatika II.) tantárgy keretein belül mi is megismerkedünk egy magas szintű programozási nyelvvel, a Visual Basic-kel. Ebben a leckében pedig röviden bemutatunk egy olyan alkalmazást (Hymn CPU szimulátor), amelyet gépi kódú, ill. assembly utasításokkal lehet vezérelni.

Buszrendszer

A számítógép fő részei közötti összeköttetést elektromos vezetékek biztosítják. Ezek a vezetékek juttatják el az egyik egységből a másikhoz a működéshez szükséges kódokat.

Ilyen vezetékek kötik össze a memóriát a CPU-val, az input, output eszközökkel stb. Kezdetben ezeken a rendszereken mindenféle kódot mozgattak, később külön vezetékrendszer készült a memóriacímek, az utasítások, az adatok kódjainak átvitelére. Egy-egy kódcsoporthoz tartozó vezetékrendszert busznak hívunk. Egy számítógép összes busza a buszrendszer. A buszrendszerben nem pont-pont kapcsolattal mennek a vezetékek, hanem minden busz egy fővezetékből áll, amelyre több egység is csatlakozhat.

A kódok bitjeit kétféleképpen lehet átvinni:

- Bitenként – egy vezetéken – egymás után. Ezt soros átvitelnek hívjuk.
- A teljes kódot, vagy annak egy jól meghatározott bitszámú részét annyi vezetéken, ahány bitből áll a megadott rész. Ehhez legalább annyi vezeték kell, ahány bitet egyszerre szeretnénk szállítani. Ezt az átvitelt párhuzamos átvitelnek nevezzük.

Órajel-generátor

Egy számítógép megfelelő működéséhez nem csak az fontos, hogy a gépet felépítő elemi áramkörök működjenek, hanem az is, hogy ezek az elemi működések vagy állapotváltozások szinkronban működjenek. Ennek biztosítására a legtöbb számítógépben egy szinkronizáló jelet, elterjedtebb nevén órajelet használnak, amit az órajel-generátor állít elő. Ez az apró áramkör (még az ősi számítógépeken sem volt jelentős méretű) a főbb egységektől függetlenül működik, az előállított jelet egy buszra teszi, amelyhez minden egység hozzáfér.

Az órajel egy alacsony (a bináris 0 kódnak felelhet meg) és egy magas (a bináris 1 kódnak felelhet meg) feszültségérték között periodikusan és ugrásszerűen váltakozó folytonos, elektromos jel. Fontos tulajdonsága a periódusidő – jele T. A T azt mutatja, hogy az órajel-generátor jele bármely időpillanathoz mérten (minimum) mennyi idő múlva lesz pontosan ugyanolyan értékű. A T reciproka az 1/T, az órajel frekvenciája, ami azt fejezi ki, hogy egy időegység alatt (pl. másodperc) hány periódus ismétlődik. A mai modern számítógépeken ez az érték már 10^9 (gigahertz, GHz) nagyságrendű is lehet.

Az áramkörök működését a felfutó és lefutó élnek nevezett 0-1 vagy 1-0 átmenetek szinkronizálják, azaz az órajelben bekövetkező hirtelen feszültségváltozást figyelve képesek a számítógép részei önmaguk és egymás közti összehangolt működésre.

Az órajel frekvenciája nagy mértékben meghatározza egy adott számítógép gyorsaságát is. (Kizárólagosságról azért nem beszélhetünk, mert itt egyéb szempontokat is mérlegelni kell, pl.: többmagos vs. egymagos gépek, ill. bizonyos eszközök egy adott szint feletti gyorsításra nem képesek).

Input és output eszközök

A számítógép csak akkor használható értelmes munkára, ha a környezetével megfelelő módon kapcsolatot tud teremteni. Ez a kapcsolat kétirányú, azaz a gépnek képesnek kell lennie a környezettől adatokat, kérdéseket stb. fogadni, és meg kell tudni jeleníteni az eredményeket, esetenként a számolás részeredményeit is. Ehhez megfelelő eszközökre van szükség van.

- Bemenő vagy inputeszköznek nevezzük azokat az eszközöket, amelyeken keresztül a számítógép adatokat, instrukciókat fogad. Ezeket inputperifériáknak is szokás nevezni.
- Kimenő vagy outputeszközöknek nevezzük azokat, amelyeken keresztül a számítógép adatokat küld a külvilág felé. Ezek az outputperifériák.
- Háttértárnak vagy input/output (I/O) eszköznek nevezzük azokat a berendezéseket, amelyekre a számítógép bináris kódokat képes küldeni, ezeket az eszköz megőrzi, és ha szükséges, vissza tudja olvasni. A háttértárak kódtároló részeit gyakran le lehet választani az egységről, és egy másik számítógép hasonló egységére téve lehetővé válik az adatok egyik gépről a másik gépre való „átszállítása" is. A háttértárat a memória kibővítésének is fel lehet fogni, hiszen azok a kódok (adatok), amelyekre a működéshez éppen nincs szükség, háttértárra küldhetők, amikor újra szükség lesz rájuk, visszatölthetők a memóriába.

Hétköznapi tapasztalataink alapján sokféle input vagy output, illetve I/O eszközt fel tudunk sorolni; ilyenek például a billentyűzet, az egér, a monitor, a hangszóró, a nyomtató, CD-olvasó, CD-író, a Winchester, stb. A mai modern számítógépekhez kifejlesztettek már olyan perifériákat is, amik nem háttértárak, mégis képesek input és output tevékenységek végrehajtására. Ezekre példa: érintőképernyő, többfunkciós nyomtató stb.

Ezeknek az eszközöknek a számítógéphez való csatolására kétfajta módszert alkalmaznak:

- A memóriába ágyazott I/O esetén az eszközöket a memória eléréséhez használt buszra kapcsolják. Ezzel a módszerrel csatolt periféria vagy háttértár ugyanúgy sorszámot (esetenként sorszámokat) kap, mint a memóriarekeszek, azaz címezhetővé válik. Kód írása vagy olvasása a memóriába való írással illetve a memóriából való olvasással megegyező módon, ugyanazokkal az adatmozgató utasításokkal történik.
- Dedikált I/O esetében az adatforgalom egy külön buszon, külön input, output utasítások hatására jut el a memóriába vagy a CPU regisztereibe.

Korunkban azt a berendezést hívjuk számítógépnek, amelyik struktúrájában és működési elvében megegyezik a fentebb vázlatosan ismertetett géppel, vagy ahhoz nagyon hasonló. (A számítógép megjelenése azonban nagyon sokféle, meglepően változatos lehet!)

Tevékenység: Saját szavaival mutassa be a Neumann-elvű számítógépek megismert főbb alkotóelemeit és működésüket!

A számítógép működése

Már láttuk, hogy a CU hogyan működik. Azt is láttuk, hogy a CU működése során hogyan szólítja meg az ALU-t, hogyan használja a memóriát, az input, az output, az I/O eszközöket. Most vázlatosan megismerjük a számítógép működését is.

Az operatív memória elemei általában elvesztik tartalmukat, ha a berendezést kikapcsolják. A belső programvezérlés elvén megépített számítógép viszont csak akkor működik, ha a memóriából a működéshez szükséges utasításhoz hozzájut. Ezért szükség van olyan tárolóegységre, amelyből a gép a bekapcsoláskor kiveheti az első működtető utasításokat.

Ez az első elektronikus számítógépeken pár bájtból álló kapcsolókból álló tábla volt, ahol a gép kezelője a kapcsolókon állította be az első utasításokat. Ezeket kapta meg a CU, amik hatására „betöltötte" a működtető gépi programot. A betöltés általában valamilyen háttértárról történt.

Manapság léteznek olyan memóriák, amelyek nem vesztik el tartalmukat a számítógép lekapcsolása után sem. Bekapcsoláskor erről a memóriatartományból veszi ki a CU az elsőként végrehajtandó utasítást, minden bekapcsoláskor ugyaninnen és mindig ugyanazt. Így a számítógép bekapcsolásakor mindig ugyanaz a cím kerül a PC regiszterbe. Ezen a címen egy olyan memóriarészt lehet elérni, amelynek tartalma kikapcsoláskor is megmaradt. Ide a gép gyártásakor beírják az indításkor végrehajtandó utasításokat. Ezeknek a végrehajtása során a CU felépíti a további működéshez szükséges memóriatartalmat. A megmaradó tartalommal rendelkező memória mérete a kezdetekben pár bájt volt, manapság több KiB, sőt GiB nagyságrendű.

A működés tehát a következő:

Bekapcsoláskor feszültség alá kerül a számítógép. Ennek a feszültségimpulzusnak a következményeképpen a CPU regisztereiben kezdőérték generálódik. A PC regiszterben lévő cím alapján bekerül az utasításregiszterbe az első gépi utasítás kódja, és a CU a már megismert módon elkezdi ciklikus működését, és ez az állandóan ismétlődő folyamat már a teljes számítógép-működést meghatározza. Ez a működés automatikus, csak akkor van szükség a kezelő beavatkozására, ha a program ezt előírja. (Természetesen a kezelő beavatkozhat a működésbe, amikor csak akar, de ennek gyakran az a következménye, hogy az éppen „futó” program megszakad, és esetleg nem is lesz folytatható.) A számítógép működése kikapcsolással állítható le. A kikapcsolás a fő egységek feszültséggel való ellátásának megszüntetésével lehetséges.

A mai modern gépek, ha éppen várakoznak a gépkezelő beavatkozására, és ez nem történik meg, akkor takarék üzemmódra kapcsolnak, sőt a kezelő utasítására már önmaguk kikapcsolására is képesek.

Számítógép-generációk

Az 1950-es évektől kezdődően a számítógép-tervezés és -készítés ipari méreteket öltött. Újabb és újabb eszközök és ötletek alapján egyre gyorsabb és megbízhatóbb gépeket építettek. A fejlődés fázisait az úgynevezett számítógép-generációkhoz tartozó számítógépek fontosabb technikai és logikai tulajdonságaival érzékeltethetjük.

Az első generáció

Az első generációs gépeket az elektroncsöves technika jellemezte. Aritmetikai egységeik – néhány kivételtől eltekintve – csak az egész számok kódjaival tudtak számolni. A lebegőpontos kódokkal végzendő műveletekre programot kellett készíteni. Ezek az egyéb, úgynevezett felhasználói programokkal együtt gépi nyelven készültek, ezt kellett az input eszköz segítségével bejuttatni az operatív memóriába. A tartalmat megőrző memória mérete pár bájtnyi volt, a gépet működtető programot jellemzően háttértárról kellett „betölteni” a memóriába. Gyakran előfordult olyan megoldás is, hogy a gép az első végrehajtandó utasításokat is a háttértárról kapta.

E korszak gépeit a processzorcentrikus felépítés jellemezte, a külső berendezéseket (input, output stb.) a CPU közvetlenül vezérelte. Emiatt a számítógép sebessége jóval alatta maradt az elektronikus alkatrészek (CPU, memória) sebességének, hiszen a perifériák sok mechanikus alkatrésszel rendelkeztek, és ezek működési sebessége több nagyságrenddel alatta maradt az elektromos egységekének.

Az első generációhoz tartozó gépek logikai felépítése alapvetően (kis változtatásokkal) a neumanni architektúrát követte. Ez a korszak az 1950-es évek közepéig tartott.

A második generáció

A második generációs gépekben (1955–65) a jóval kisebb méretű és sokkal gyorsabb félvezető diódák, tranzisztorok kiszorították az elektroncsöveket. Ezzel gyorsítani lehetett a CPU működését, és bonyolultabb áramkörök létrehozása is lehetségessé vált. Hasonlóan, a ferritgyűrűből (egy bit tárolására kiválóan alkalmas egység) felépített memória is nagyságrendben kisebb helyet foglalt el az elektroncsöveshez képest, ezért sokkal nagyobb kapacitású memóriát lehetett vele építeni.

Ha a számításokhoz kicsi volt a memória, akkor az éppen nem használt kódokat (adatokat) háttértárakra írták, ahonnan szükség esetén beolvashatók voltak. Az (egységről levehető) adathordozók mágnesezhető anyagból készültek, kivitelük szalag- vagy lemezforma volt.

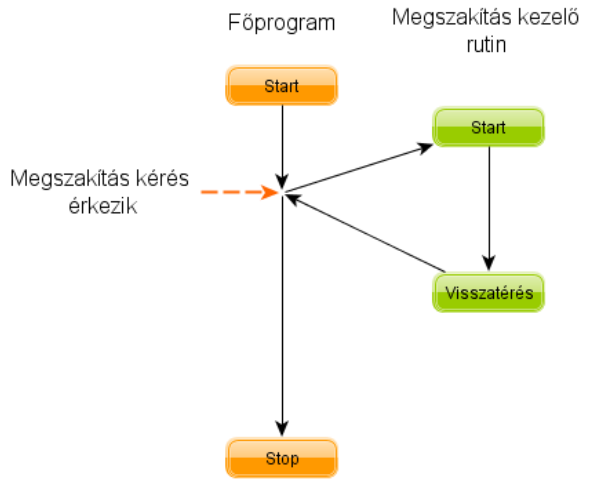
A második generációs gépek mérete jelentős mértékben lecsökkent, az üzembiztonságuk megnőtt. A teljesítményük, sebességük az 1 MIPS értéket is elérte.

A működési sebesség növekedésében jelentős szerepet játszott a külső berendezések (perifériák) vezérlésének függetlenítése a CPU-tól (a CPU tehermentesítése is általános cél volt, lásd még lent is). A gyorsabb központi egységet a lassú perifériák visszafogták, éppen ezért az I/O műveletekre a CPU helyett önállóan működő perifériaprocesszorokat építettek a gépbe. Ezek vezérelték az input és output tevékenységet, a CPU csak elindította a folyamatot. A művelet elvégzése után a perifériaprocesszor üzenetet küldött a CPU-nak. Ezzel a megoldással az I/O műveletek elvégzése alatt a CPU további utasításokat tudott végrehajtani.

A fenti megoldást követve az output folyamat befejezését a CPU csak akkor érzékelheti, ha erről jelzést kap. Ezt a jelzést, az úgynevezett megszakításkérést a processzornak fogadnia kell, amit csak akkor tud megtenni, ha az éppen feldolgozás alatt lévő programot megszakítja, és csak azt követően folytatja, ha elvégzi azokat az utasításokat, amelyek a kivitel befejezéséhez szükségesek. Bizonyos esetekben az input perifériák is kezdeményezhetnek kommunikációt. Ilyenkor a megfelelő perifériavezérlő ugyancsak megszakításkérést küld a CPU-hoz, és minden nagyon hasonlóan zajlik le, mint ahogy fent bemutattuk.

Ez a folyamat részletesebben (5. ábra):

1. Az I/O eszköz jelzi a tevékenységének kezdetét vagy végét, azaz megszakításkérést küld a CPU-nak.
2. A CPU észleli a megszakításkérést, az éppen működő programot a feldolgozás alatt lévő utasítás végrehajtását befejezve megszakítja.
3. Elindítja a megszakításkérésnek megfelelő feladat elvégzésére szolgáló eljárást.
4. A megszakításkérést kiváltó problémához tartozó eljárást befejezve folytatja a megszakított programot (a következő utasítással).



{á:m1e2a05.png}

5. ábra
Programvégrehajtás megszakítás esetén

A megszakításkérések kezelését a CPU végezte, de némelyik számítógépbe már külön megszakításvezérlő áramkört is építettek, ami gyorsabbá és biztonságosabbá tette a megszakításkezelést.

A megszakításkérések hierarchikusan is feldolgozhatók, ami azt jelenti, hogy a számítógépekben ezek feldolgozása nem sorban, egymás után történik, hanem egy magasabb prioritású (elsőbbséget élvező, valamilyen szempontból fontosabb) megszakításkérés megszakíthatott egy épp folyamatban lévő alacsonyabb prioritásút (kevésbé fontosat).

Tevékenység: Egyprocesszoros, egymagos gépet feltételezve gondolja végig, hogy a fentieken túlmenően milyen típusú feladatok megoldása történhet még megszakításkezeléssel!

Néhány gyártó megjelent a piacon olyan ALU-val is, amelyek már közvetlenül voltak képesek a lebegőpontos kódokkal való számolásra, így ezeken a gépeken nem kellett program a lebegőpontos kódolású számokkal való műveletvégzéshez.

A számítógép működéséhez, működtetéséhez segédprogramokat készítettek. Ezek a programok voltak a későbbi operációs rendszerek előfutárai. Operációs rendszernek egy olyan programrendszert nevezünk, amely képes arra, hogy a felhasználó minimális segítségével teljesen automatikusan működtesse a számítógépet, ill. olyan szolgáltatásokat adjon, amelyek kényelmessé teszik a felhasználók feladatait megoldó programok használatát is.

Megjelentek azok a programrendszerek, amelyek bizonyos feladatkör megoldását, kezelését tették lehetővé, azaz kialakult az ipari szoftvergyártás. A szoftver a különböző programok, programrendszerek gyűjtőneve. A megvásárolható programrendszerek mind-mind szoftvertermékek.

A termékek előállítására hatékony eszközökre volt szükség, ezért ebben korszakban dolgozták ki az első magas szintű programozási nyelveket is.

A harmadik generáció

A harmadik generációs gépek korszaka a 60-as évek közepétől a 70-es évek végéig tartott. Már néhány évvel a korszak kezdete előtt megjelentek az integrált alapú áramkörök, amelyekbe miniatűr méretben akár több száz dióda, tranzisztor volt beépítve. A miniatürizálás következménye a számítógépekben: csökkent a méret, arányosan nőtt a működési sebességük és négyzetes arányban csökkent az energiafogyasztás. A tárolási kapacitás megsokszorozódott.

Megjelentek olyan operációs rendszerek, amelyek képesek voltak egyszerre több program futtatására is. Kétféle technológiát alkalmaztak arra, hogy több program egyidőben futtatható legyen.

A multiprogramozás elvét követve az operatív memóriában egyszerre több felhasználói program is lehet, és ezeket az operációs rendszer felügyeletével a számítógép egy adott prioritási sorrend szerint futtatja. A forgatókönyv tehát a következő: Az első program futtatásával kezdődik a munka. Ha ez a program I/O tevékenységet kezd, akkor ennek befejeztéig a gép a második program utasításait hajtja végre egészen addig, amíg az első program I/O tevékenysége be nem fejeződik, vagy a második el nem indít egy I/O tevékenységet. Az első esetben az első programot folytatja a gép, a második esetben a harmadik program végrehajtására kapcsol és így tovább.

A multitaszking (többfeladatos rendszer) filozófia szintén több program futását teszi lehetővé úgy, hogy számukra „fair” kiszolgálást biztosít. A rendszer minden programhoz hozzárendel egy időszeletet – többnyire ugyanannyit –, és amikor az egyik programra szánt idő lejár, akkor a következő program utasításai kerülnek sorra egészen addig, amíg az utolsó programra szánt idő is letelik. Ekkor újra az első program utasításai kerülnek sorra. I/O tevékenység végrehajtása esetén természetesen rögtön a következő program utasításaira kerül sor, azaz a multiprogramozási technika itt is érvényesül. Ha az egyes programokra szánt idő elegendően rövid, akkor a felhasználók úgy érzékelik, mintha a programjaik párhuzamosan működnének. Ezt a működési módot időosztásos (time sharing) technikának nevezzük.

Tevékenység: Windows operációs rendszerének működését elemezve írjon össze néhány tipikus feladatot, amit a rendszer elvégez (pl. egérkattintás észlelése és megfelelő reakció, idő kijelzése, telep állapotának kijelzése stb.). Gondolja végig, hogy mi lenne a feladatokhoz tartozó helyes prioritási sorrend (az teljes rendszer gyors és biztonságos működésének biztosítása a cél)!

A gépek kihasználtságát azzal is fokozták, hogy a gépen egyidőben több felhasználó dolgozhatott. Ezek a felhasználók akár több programot is elindíthattak. Ezek futását multiprogramozott vagy multitaszking módon az operációs rendszer felügyelte. Mivel a gépen egyszerre több program is futhatott, gondoskodni kellett a memória megfelelő kezeléséről. Nem szabadott ugyanis megengedni, hogy az együtt futó programok egymás memóriaterületére írjanak, esetleg olvassanak.

Az is előfordulhatott, hogy a memória mérete önmagában nem lett volna elég az elindított programok futtatásához. Ennek a problémának a megoldására kialakult a virtuális memóriakezelés.

Ennek lényege az, hogy a memóriát logikailag egyenlő részekre, úgynevezett lapokra osztják, és ezeket a lapokat rendelik a futtatandó programokhoz. Ha a program nagyobb területet igényel, mint amit a kiosztott lapok meghatároznak, akkor a programnak és a futáskor szükséges munkaterületnek csak egy része kerül az operatív memóriába, a többi a háttértáron marad. Ha szükség van a háttértáron lévő adatokra, akkor az operációs rendszer kicseréli az éppen nem használt lapot a memóriában arra a részre, amire a program folytatásához szükség van. Ezt a működési módot memórialapozásnak hívjuk. Ezzel a technikával látszólag az operatív tár méreténél nagyobb (virtuális vagy látszólagos) memória használható.

Mind a multiprogramozott, mind a többfeladatos, mind a többfelhasználós használatot az operációs rendszerek biztosították. Ez mindhárom esetben olyan memóriamenedzselést jelentett, amely nélkül a biztonságos működés lehetetlen lett volna.

Az operációs rendszerek új szolgáltatásai mellett szerkezetük is jelentősen fejlődött. Megjelent a modularitás, és ezek a modulok piramisszerűen egymásra épülve a hétköznapi felhasználó elől elrejtették a gép valódi működését, olyannyira, hogy ő csak a piramis tetején lévő modulhoz férhetett hozzá. Ezzel megnövekedett a működés biztonsága is.

Megjelentek a modulok létrehozását támogató, sőt speciális feladatok programozására alkalmas magas szintű programozási nyelvek, ezáltal tovább növekedett a programozói munka hatékonysága.

A negyedik generáció – napjaink számítógépe

A negyedik generációs gépek megjelenését a 70-es évek közepétől számítjuk. A technikai fejlődés következményeként lecserélhetők lettek a még mindig elég nagyméretűnek számító elektronikus és mágneses elven működő alkatrészekből álló egységek az integrált áramkörökkel megvalósított egységekre. Ez a számítógépek méretének jelentős csökkenésével járt, és lehetőséget adott arra is, hogy olyan egységek is megjelenjenek a számítógépeken, amelyeknek megépítésére eddig nem volt lehetőség (ill. megvalósításuk időközben új igényként merült fel). Ezek az új részek egyrészt az operációs rendszerben eddig szoftveres úton megoldott funkciókat váltottak le hardver megvalósításra, másrészt új szolgáltatásokat nyújtottak a számítógép biztonságos működéséhez.

Ilyen új egységek:

- Nagy kapacitású megmaradó tartalmú memória.
- Kisméretű, nagykapacitású, közvetlen hozzáférésű írható, olvasható memória-áramkörök.
- Direkt memória-hozzáférést biztosító áramkörök.
- Memória-hozzáférést szabályozó áramkörök.
- Programozható perifériavezérlők.
- Új periféria- és háttértár-csatlakozók.
- Mikroprocesszor.

Ezek az egységek az integrált áramkörök gyártástechnológiájának továbbfejlődésének (az LSI – Large Scale Integration: magas integráltsági fokú áramkör megjelenése) következményeképpen egyre kisebb és kisebb méretűek lettek, és manapság már a zsebben is elfér egy olyan teljesítményű számítógép, amely a generáció kezdetén még szobaméretű volt.

Itt külön megemlítjük Gordon E. Moore észrevételét (ez Moore-törvényként vált ismertté), amely szerint az egységnyi területű áramköri lapra beépített diódák, tranzisztorok száma a technikai fejlődés következtében (az 1960-as évektől kezdődően) 18-24 havonta megduplázódik. A miniatürizálás folyamata az informatikával alig foglalkozó külső szemlélők számára is rendkívül látványosan bemutatható ennek a törvénynek a felhasználásával.

{á:m1e2a06.png}

6. ábra

Chip-pagoda – a Moore-törvény szemléltetése (minden lapka az alatta levőhöz képest feleakkora területen tartalmaz azonos számú tranzisztort; Paderborn, Heinz-Nixdorf múzeum)

Tevékenység: Becsülje meg, hogy a Moore-törvény betartásával 10 év alatt nagyjából hányadrészére csökken annak a lapkának a területe, amely kezdetben 1 m^2-es!

Memóriák, memória-hozzáférés és -menedzselés

A tartalmukat megőrző (csak olvasható) memóriákból az évek során többféle típus alakult ki. A klasszikus változat a ROM memória (Read Only Memory, csak olvasható memória). Tartalmát a gyártáskor rögzítik. Ez a tartalom nem változtatható, megmarad akkor is, ha a gép nem kap áramellátást, tápfeszültséget. Felhasználása elvileg korlátlan idejű, de a gyártási technológia függvényében ez az idő véges is lehet (pl. 25 évre garantáltan megmarad a tartalom). A memóriacellák általában direkt elérésűek, ami azt jelenti, hogy bármelyik memóriaegység tartalma közvetlenül kiolvasható, függetlenül attól, hogy az előtte vagy utána lévő memóriaegységek tartalmát elolvasta-e a számítógép vagy sem.

A PROM (Programmable Read Only Memory) típusba nem gyárilag került bele a tartalom, hanem speciális programíró berendezéssel a felhasználója egyszer beégette, és ezután az eszköz ROM-ként volt használható (mára elavult). A típus továbbfejlesztett változata az EPROM (Erasable Programmable Read Only Memory, törölhető programozható csak olvasható memória). Ez a memóriafajta ugyan törölhető és újraprogramozható volt, de csak speciális eszközzel (UV-fénnyel), a gazdaszámítógéppel nem. A gyártáskor általában ebbe a memóriába is beleégették a kezdő tartalmat (mára szintén elavult típus).

A legújabb változatok az EEPROM memóriák (Electrically Erasable Programmable Read Only Memory, elektromosan törölhető programozható csak olvasható memória). A csak olvasható kitétel valójában itt arra utal, hogy a memória tápfeszültség nélkül is megőrzi tartalmát, nem szükséges azt

újraírással frissíteni. Ha kell, a tartalom kicserélhető, átírható. Ezt az átírást – az ős-típusokkal ellentétben – a gazdaszámítógéppel is elvégezhetjük. Az EEPROM memóriák egy speciális változatát, a flashmemóriát háttértárként is lehet használni. Ilyen memória található a mai pendrive-okban is.

A tartalmukat elvesztő (írható-olvasható) memóriákat RAM-nak (Random Access Memory rövidítése; szó szerinti fordítás: véletlen elérésű memória) nevezzük. Az elnevezés nem igazán szerencsés, mert itt valójában arról van szó, hogy a memóriaegységek tartalmát direkt módon, a cím alapján lehet kiolvasni vagy megváltoztatni. Ezeknek a feladatoknak az elvégzésére fordított idő – más szóval az elérési idő – nem függ a memóriaegység helyétől. Nem pontos az elnevezés azért sem, mert a ROM-típusú memóriák tárolóegységeit is így érhetjük el. Ennek ellenére a RAM megnevezés csak az írható-olvasható memóriákat jelenti. Három típusról tanulunk.

Az első a dinamikus RAM vagy DRAM. Bitjeit olyan elektronikus elemek (1 kondenzátor és 1 tranzisztor) alkotják, amelyek tartalma kiolvasáskor, illetve az idő előrehaladtával akkor is elveszik, ha feszültség alatt van a memória. Éppen ezért kiolvasás után és bizonyos idő elteltével a tartalmat frissíteni kell (ezért nevezik dinamikus RAM-nak). Az állandó frissítések miatt működése relatíve lassú, de viszonylag olcsó előállítási költsége miatt általánosan elterjedt operatívmemória-fajta.

A második a szinkron DRAM, röviden SDRAM (Synchronous Dynamic Random Access Memory) olyan memória, amit az órajel segítségével a többi egységgel szinkronban lehet elérni. Ez a memória csak akkor fogad, ill. küld, ha az órajel ezt lehetővé teszi (például egy felszálló ág megjelenésekor).

A harmadik típus a statikus RAM vagy SRAM, amelynek bitjei – az eltérő felépítés miatt – csak akkor vesztik el a tartalmukat, ha a tápfeszültség megszűnik, így normál működés közben nincs szükség állandó frissítésre. Az elérési idejük így jóval kisebb, mint a DRAM memóriáké. Előállítási költségük viszont nagyságrenddel nagyobb, ezért elsősorban speciális (nem operatív) memóriaként használják őket, például a CPU regisztereinek megvalósítására. Szintén SRAM-elemekből épülnek meg a gyorsítótárak (cache-memóriák). Ezek viszonylag kiskapacitású (kezdetben pár száz B, manapság akár pár KiB is lehet) memóriák, amelyek arra szolgálnak, hogy ide a lassúbb operatív memóriából (a CPU működésével párhuzamosan) adatokat másoljunk, hogy azokat a CPU innen gyorsan kiolvashassa, ha szüksége lesz rá. Használhatnak gyorsítótárat a perifériák, a háttértárak és az operatív memória közti adatforgalom lebonyolítására is.

Tevékenység: Saját szavaival mutassa be összefoglalóan a ROM és RAM memóriák megismert típusait!

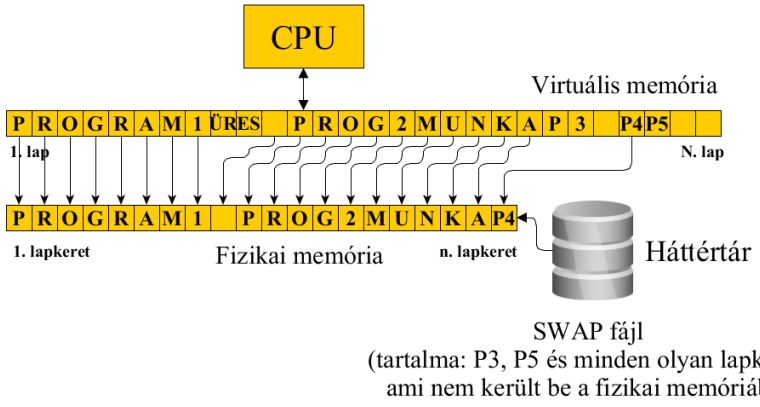
Ahogy a második generációnál már megemlítettük, a számítógépek gyorsításánál általános trendként jelentkezett a lassúbb külső(bb) egységek vezérlésének a (gyors) CPU-tól való minél nagyobb függetlenítése, illetve a CPU ilyen téren történő tehermentesítése. Ezen célból fejlesztették ki azt az áramkört is, amely a közvetlen memória-hozzáférést és ennek vezérlését megvalósítja meg (DMA, Direct Memory Access).

Gyakran előfordul ugyanis, hogy nagymennyiségű adatot kell másolni I/O eszközök és a memóriatartomány, valamint csak memóriatartományok között. Gazdaságtalan lenne, ha ezt teljes egészében a CPU végezné, ehelyett elegendő csak elindítania a feladatot, és érzékelni a befejezését. Az indításkor a CPU beállítja a DMA vezérlőjében a másolandó adatok jellemzőit (forrás, cél, méret), majd folytatja a működését más (a másolástól független) utasítások végrehajtásával. A DMA-vezérlő elvégzi a másolási feladatot, és ha elkészült, akkor megszakításkéréssel jelez a CPU-nak, az pedig visszatérhet a másolt adatokkal kapcsolatos további tevékenységekhez.

Ezzel a memóriaelérési technológiával természetesen a DMA átveheti az I/O eszközök elindításának és az I/O eszközök megszakításkérésének feldolgozását is (ez persze nem szükséges azokban az esetekben, ha a mozgatandó adatmennyiség kicsi; például: billentyűk leütéséből, egér mozgatásából keletkező input adatok).

Ahogy a negyedik generációs gépek bevezetésénél említettük, a számítógépek fejlődésében, gyorsításában kulcsszerepet játszott az a filozófia is, hogy az operációs rendszerben korábban szoftveres úton megoldott funkciókat váltottak le hardver megvalósításra (új beépített áramkörök, de a technológia fejlődésével ez nem járt a gépek méretének növekedésével). Egy ilyen eszköz a MMU (Memory Management Unit, memória-menedzselő egység) is. Segítségével a virtuális memória kezelése jóval gyorsabbá vált, hiszen áramkörökkel valósítottak meg olyan programrészeket, amelyek eddig az operációs rendszer részét képezték (ezzel az operációs rendszer leterheltsége is csökkent).

Az MMU az operatív memóriát kisméretű – általában 2, 4, 8 KiB nagyságú – blokkokból, lapokból álló rendszerként kezeli (a lapok sorszámmal azonosíthatók). Amikor egy programot a memóriába töltünk, az MMU elhelyezi valamely lap(ok)ra, végeredményben az MMU a CPU számára virtuális memóriaként működik (7. ábra). A virtuális memória mérete általában nagyobb a fizikai memóriaméretnél, így az MMU a lapozási technika segítségével az operatív tárban mindig azt a kódrészt tárolja, amire éppen szükség van.



{á:m1e2a07.png}

7. ábra
Az MMU virtuális memóriaszolgáltatása

Perifériavezérlők és -portok

A fent ismertetett elv – lassúbb külső egységek vezérlésének a CPU-tól való minél nagyobb függetlenítése – viszonylag korán, már az 1960-as években megfogalmazódott a perifériák és a háttértárak esetében, mivel ezek működési sebessége nagyságrendekkel elmaradt a CPU működésétől. Ezt azonban akkor a technológiai fejletlenség miatt még nem lehetett megfelelő módon megvalósítani.

A miniatürizálás előrehaladtával és a gyors memóriatípusok megjelenésével az 1970-es, 80-as években a gátló tényezők megszűntek, így lehetővé vált perifériavezérlés önálló egységekkel történő megoldása. Ezek kezdetben egyszerű vezérlők voltak, amelyek az adott periféria és a memória közti kódforgalmat lebonyolító utasítások végrehajtására voltak képesek (megszakításkéréssel). Később a vezérlőkhöz memóriát, sőt műveletvégző egységet is építettek. Némelyiket ilyen eszközt fixen a gépbe

építették, másokat bővítményként, a gép felhasználójának kívánságára – akár utólag – lehetett a számítógépbe építeni.

Később megjelentek az intelligens perifériák is, amik a számítógéptől független memóriával, vezérlővel és műveletvégző egységgel rendelkeznek. Mondhatnánk, hogy ezek speciális célra készült számítógépek. Manapság szinte mindegyik periféria (háttértár) ilyen.

A számítógépbe épített perifériavezérlők csatlakozóin (portjain) lehet a perifériákat a számítógéphez csatlakoztatni.

Kezdetben minden perifériának külön vezérlője volt, később a perifériacsoportok kaptak vezérlőt, és szofveres úton, meghajtóprogrammal volt képes a vezérlő a perifériát kezelni.

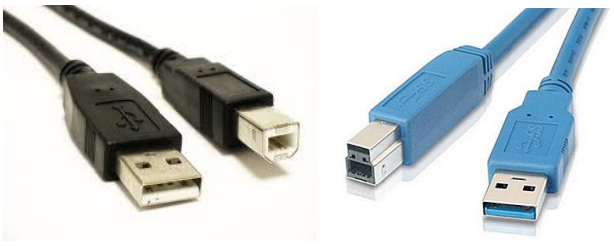
Napjainkban általánosan elterjedt az USB (Universal Serial Bus, általános soros busz), amelynek portjára (csatlakozó pontjára) megfelelő szoftvertámogatással szinte mindenféle periféria csatlakoztatható. Ennek megfelelően az USB-t használó eszközök köre igen széles: billentyűzet, egér, játékvezérlő, lapolvasó, nyomtató, digitális fényképezőgép, webkamera, külső merevlemez stb. Az USB soros, pont-pont közötti kapcsolatot biztosít a számítógép (host) és a külső eszköz (device) között. Az USB egy számítógéphez legfeljebb 127 eszköz csatlakozását teszi lehetővé. Ennyi csatlakozási pontot persze egyetlen számítógéphez sem építenek, de speciális elosztóval, az USB-hubbal a csatlakozási pontok sokszorozhatók.

A manapság használatos gépeken és eszközö0kön az USB három verziójának valamelyike található meg. Működésük lehetséges sebességfokozatait az alábbi táblázat tartalmazza. A sebességet megabit/másodperccel (egy másodperc alatt átvitt bitek száma Mb-ben) jellemezzük.

Sebességfokozat	USB 1.1	USB 2.0	USB 3.0
Alacsony sebesség (low speed)	1,5 Mb/s	1,5 Mb/s	1,5 Mb/s
Teljes sebesség (full speed)	12 Mb/s	12 Mb/s	12 Mb/s
Magas sebességfokozat) hi-speed	x	480 Mb/s	480 Mb/s
Szupersebesség (superspeed)	x	x	5 Gb/s

3. táblázat
USB szabványok maximális átviteli sebességei

Az USB szabvány 1.1-es és 2.0-ás változatában azonos csatlakozókat és kábeleket használnak, a 3.0-ás változatban azonban változtatni kellett a megnövekedett sebesség miatt. Az ilyen 3.0-ás csatlakozókat kék színnel különböztetik meg a hagyományos 1.1-es és 2.0-ás csatlakozóktól, amelyekkel felülről kompatibilis.



{á:m1e2a08.png}

8. ábra
USB 2.0 és 3.0-ás csatlakozók

Tevékenység: Ellenőrizze saját számítógépét és saját eszközeit, hogy milyen USB csatlakozókkal rendelkeznek!

Processzorok

Azzal, hogy az I/O vezérlése önállóvá vált, a CPU-ra csak a memória (esetenként ennek is csak a virtuális változata) kezelése maradt. Az időszak elején, amikor a programok futási idejét a memória gyorsasága határozta meg elsősorban, olyan CPU-kat alkalmaztak, amiknek utasításkészlete bonyolult és sok utasításból állt. A lassú memória miatt a bonyolult utasításokhoz szükséges viszonylag hosszú végrehajtási idő sem növelte a programok futási idejét. Az ilyen számítógépeket összetett utasításkészletű számítógépnek hívták (CISC, Complex Instruction Set Computer).

Mivel a memóriák gyorsasága növekedett, ez lehetővé tette az utasítások egyszerűsítését, és a számuk is csökkent. Igaz, hogy ugyanannak az eredménynek eléréséhez így akár négy-öt egyszerűbb utasítás is kellett, de ezeket az utasításokat akár tízszer gyorsabban lehetett végrehajtani, ami CPU működésében jelentős gyorsulást eredményezett. Az ilyen számítógépet csökkentett utasításkészletűnek nevezték (RISC, Reduced Instruction Set Computer).

A miniatürizálás növekedésével lehetőség nyílott arra is, hogy a különálló regisztertömböt, CU-t és ALU-t egybeépítsék. Ezt az egybeépített egységet hívják processzornak. Az egyetlen chipben megvalósított processzor a mikroprocesszor.

A párhuzamos végrehajtás hardveres megoldásai

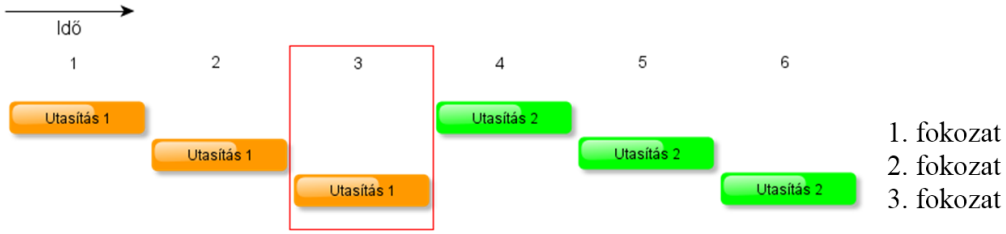
A memóriák további gyorsulása, illetve a különböző memóriakezelési technikák alkalmazása és egyéb más okok következményeképpen a processzorok sebességének növelése fontos kutatási feladattá vált. Az egyik ígéretes lehetőségnek a párhuzamos végrehajtás tűnt.

Neumann – mint tudjuk –, úgy képzelte el számítógépének utasítás-végrehajtását, hogy a CU csak akkor fog neki a következő utasításnak, ha az előzőt már befejezte. Ez a technológia, amely a korai számítógépek CPU-egységét jellemezte, nem volt hatékony módszer. Érezte ezt Neumann János is, hiszen nevezetes jelentése után nem sokkal – más számítástechnikával foglalkozó kutatókkal egy időben – máris foglalkozni kezdett a párhuzamos végrehajtás lehetőségével.

Az alapprobléma az volt, hogyan is lehetne megoldani, hogy az utasítás-végrehajtáskor a CPU éppen nem működő részeit hogyan lássuk el munkával, illetve milyen módon lehetne egyszerre több utasítást is végrehajtani. Többféle javaslat is született, megvalósításukhoz viszont akkor még nem volt megfelelő technikai háttér. Az integrált áramköri technológia kialakulása már lehetővé tette ezeknek a javaslatoknak megvalósítását, sőt újabb megoldásokat is eredményezett.

A modern számítógépek szinte mindegyike alkalmaz valamilyen párhuzamos technológiát, annak ellenére, hogy tudjuk: bizonyos utasítások nem végezhetők el párhuzamosan. Ilyenek például a függő értékadások, ahol elő- és utóidejűség megkövetelt.

A legegyszerűbb, leggyakrabban használt lehetőség az utasítások párhuzamos végrehajtására a pipeline vagy csővezeték-technika. Pipeline nélkül az utasítások végrehajtása időben teljesen „szeparáltan” történik: az utasítás végighalad a beolvasás, dekódolás és végrehajtás fázisain, majd ezután következhet egy újabb utasítás végrehajtása. Bármely időpillanatban vizsgáljuk a processzort, annak csak egyetlen egysége dolgozik, a többi kihasználatlanul áll.

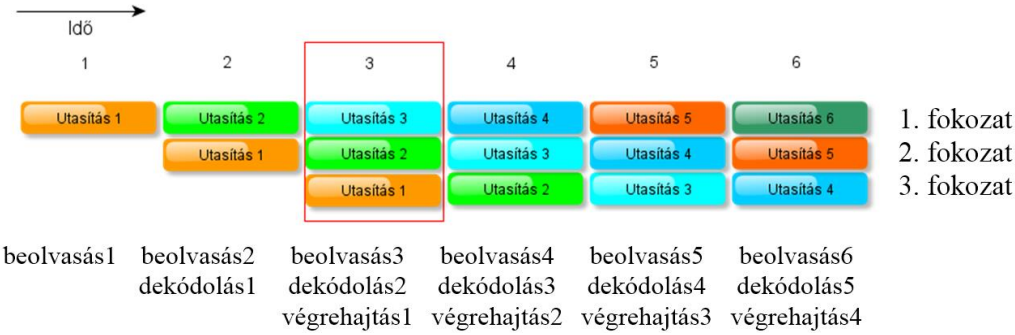


{á:m1e2a09.png}

9. ábra

Végrehajtás pipeline nélkül

A pipeline ezzel szemben úgy működik, mint egy szerelőszalag a gyárban. Az első utasítás dekódolása majd végrehajtása közben a felszabadult beolvasó egység már olvassa az éppen feldolgozás alatt lévő utasítást követőt, majd ha lehetséges az ezt követőt stb. A bekapcsolás utáni pár ciklustól eltekintve bármely időpillanatot vizsgálva azt tapasztaljuk, hogy a processzor összes egysége dolgozik.



{á:m1e2a10.png}

10. ábra

Végrehajtás pipeline használatával

Az előző két ábráról látható, hogy a soros végrehajtás hat időegység alatt két utasítást, a pipeline ugyanennyi időegység alatt négy utasítást fejez be, azaz jóval nagyobb teljesítményre képes, így rövidebb idő alatt futtatja le ugyanazt a programot.

A pipeline annál hatékonyabb, minél több részre, fokozatra (angolul stage) sikerül szétosztani egy utasítás végrehajtását. A több pipeline fokozat ugyanakkor nagyobb, bonyolultabb processzort jelent, és az utasítások végrehajtására is több órajel-periódusidő kell.

További problémát okoznak az elágazó utasítások. Elágazáskor csak a végrehajtás egy későbbi fázisban derül ki, hogy a programot melyik programág utasításainak végrehajtásával kell folytatni, tehát elképzelhető, hogy az addig betöltött, részben végrehajtott utasításokat ki kell dobni, és más utasítások végrehajtását kell elkezdeni. Az ilyen újrakezdések természetesen jelentős teljesítménycsökkenést okozhatnak. Elkerülésükre további ötletek megvalósítására van szükség (például az elágazások előrebecslése). Hasonlóan, ha a párhuzamosan végrehajtott utasításokban az operandusok egymástól nem függetlenek, akkor késleltetést kell alkalmazni. Ez szintén ronthatja a processzor teljesítményét.

Azonban mindezen problémák ellenére is a pipeline-technológia hatékonyabb programfuttatást tesz lehetővé, mint az egyszerű soros utasítás-végrehajtás.

Mivel az utasítás-végrehajtáskor a legtöbb időt a művelet-végrehajtás, azaz az ALU működési ideje teszi ki, ezért kézenfekvő megoldás a teljesítménynövelésre az, hogy a processzorba több ALU-t is beépítenek. Az ilyen több ALU-val rendelkező processzorok a szuperskalár-processzorok. Ezek jelentik párhuzamos végrehajtás következő szintjét. Az ilyen CPU-k az órajel egy periódusidején akár több utasítás végrehajtását is befejezhetik. Az adatfüggőségek itt is késleltetéssel küszöbölhetők ki. A késleltetés okozta késedelmek megszüntetésére az ilyen processzorokban gyakran tartalmaznak soron kívüli utasítás-végrehajtást (out-of-order execution). Ez azt jelenti, hogy a függőséget okozó utasítás helyett, a processzor egy független utasításban megadott műveletet hajtat végre az éppen szabad ALU-n, a felfüggesztett utasítást pedig akkor folytatja, amikor az adatfüggőség megszűnik.

A vektorprocesszorok olyan speciális célra kifejlesztett processzorok, amelyeknek az adatregiszterei többszörözve vannak, és regisztervektort alkotnak. Innen kerülnek az adatok a vektorművelet elvégzésére képes ALU-hoz, ami ugyanazt a műveletet hajtja végre mindegyik regisztertartalmon. Az eredmény is egy regisztervektorba kerül. A regisztervektorok feltöltését és az eredmény kiolvasását, általában a kódok átvitelét gyors eszközökkel végzik, így a processzor rendkívül gyors. Általános célú processzorral kombinálva, illetve speciális egységekben (például videokártyákban) használják. Nagy mennyiségű kód átkódolása (például titkosítása) nagyon gyorsan elvégezhető ezzel a processzorral.

Az eddig bemutatott megoldásokban a sebességnövelést a processzor bizonyos részeinek megsokszorozásával érték el. A multiprocesszoros rendszerek ezzel szemben több teljes értékű processzort tartalmaznak. Ezek a processzorok lehetnek egyenrangúak, azaz ilyenkor mindegyik processzorra rá lehet bízni bármelyik folyamat végrehajtását. Ez a szimmetrikus multiprocesszoros rendszer, röviden SMP (Symmetric MultiProcessing). Ha a processzorok nem egyenrangúak, azaz mindegyik más-más feladat elvégzésére alkalmas, akkor aszimmetrikus multiprocesszor-rendszerről beszélünk, rövidítve AMP (Asymmetric MultiProcessing). Az AMP rendszerekkel kezdődött a multiprocesszorok korszaka, és a szimmetrikus rendszerek csak később jelentek meg a számítógépeken.

Tevékenység: Olvasson utána az interneten, hogy népszerű játékprogramokban hogyan alkalmazzák a most megismert párhuzamos végrehajtási lehetőségeket!

A szoftverek és a szoftvertermékeket előállító eszközök

Az operációs rendszerek bonyolultsága, szolgáltatásai legalább olyan mértékben fejlődtek, mint a hardver eszközök. A felhasználó-gép kapcsolatot lehetővé tevő karakteres parancs-vezérlést felváltotta a grafikus felület, aminek kezelése sokkal könnyebb, és könnyebben is tanulható (operációs rendszerek fejlődése).

Ebben az időszakban fejlesztették ki a strukturált programozást támogató programozási nyelveket. Strukturált programtervezésről akkor beszélünk, ha a programjainkat olyan struktúrákból (programelemekből) építjük fel, amelyeknek végrehajtása az első utasításukkal kezdődik, és az utolsó utasításukkal fejeződik be. Az így megtervezett és leírt programot tekintjük strukturált programnak.

Ekkor jelent meg a programozásban az objektumszemlélet is, és ezzel együtt az objektumorientált programozási nyelvek és használatuk. Ebben a szemléletben a programot objektumok rendszerének kell elképzelni, és a működés során ezeknek az objektumoknak a kölcsönhatása, „beszélgetése” szolgáltatja annak a feladatnak a megoldását, amihez a program készült. Az objektumorientált programozási nyelvekhez definiáltak olyan objektum osztályokat is, amelyekből – mintegy

mintakollekcióból – a feladatunk megoldásához szükséges objektumok létrehozhatók. Természetesen a programozó saját objektumosztályokat is megalkothat, és ezek alapján is kreálhatók objektumok.

Teljesen új szemléletű programozás a logikai programozás. Ezt a szemléletet a szakértői rendszerek kifejlesztésének igénye inspirálta. Ezt a filozófiát követve az instrukciókra, utasításokra épülő programozási rendszereket felváltották azok a programfejlesztési eszközök, amik nem utasításokat fogalmaztak meg, hanem következtetési szabályokat definiáltak, és axiómákat írtak le. Ezek gyűjteménye lett a program. Majd a felhasználó által feltett eldöntendő kérdésre, az axiómákra támaszkodva a következtetési szabályokat felhasználva a programot futtató rendszer megpróbálta megkeresni a választ, ami kétféle lehet; az egyik – nincs válasz, a másik megadta, hogy a kérdésre a válasz igen vagy nem.

A programozási nyelvekben megjelentek a párhuzamos működést támogató programozási eszközök. Ezek segítségével a programozó is képes volt olyan programot készíteni, amiben megadhatta, hogy a program mely részeit lehet párhuzamosan végrehajtani, és azt is, hogy ezt a párhuzamos végrehajtást hogyan kell vezérelni. A hagyományosnak mondható, egyetlen munkafolyamot – szálat – meghatározó algoritmusok mellett megjelentek olyan algoritmusok, amik a feladatot párhuzamosan végrehajtható folyamatokkal oldották meg. Ezzel teljessé vált a párhuzamos technológia: párhuzamos algoritmus, program és végrehajtás.

Az ötödik generáció

Az ötödik generációs számítógépek elveinek kidolgozása, ezek alapján a gép megtervezése és megépítése napjaink feladata. A próbálkozások és a kutatások ígéretesek, de a jegyzet írásakor még nem történt meg az az áttörés, amely lényegesen megváltoztatná a számítógépet, és ezzel megteremtené azt a berendezést, amit az ötödik generációhoz sorolhatnánk.

Személyi számítógépek – PC-k

Tágabb értelemben személyi számítógépen vagy PC-n (personal computer) olyan általános célú számítógépet értünk, amelynek mérete, ára és szolgáltatásai lehetővé teszik, hogy bárki számára megvásárolható legyen, és különösebb mély szakmai tudás nélkül bárki, a munkahelyen, otthon vagy némelyik típusát akár utazás, ill. az utcán séta közben is használhassa.

{á:m1e2a11.png}

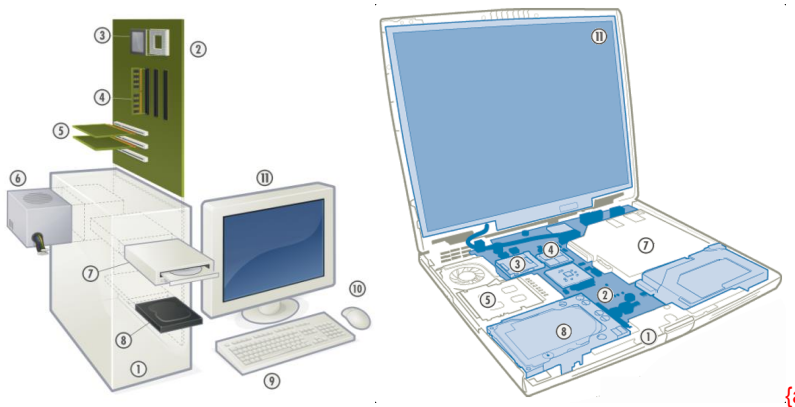
11. ábra

Személyi számítógépek

Szűkebb értelemben PC-nek nevezzük a személyi számítógépek egy családját, az „IBM kompatibilis” PC-ket. Az IBM – a világ egyik legnagyobb informatikai vállalata – egyaránt foglalkozik hardvergyártással és szoftverfejlesztéssel is. A PC-család alapját képező olcsó, moduláris felépítésű számítógépet az 1981 augusztusában dobta piacra, ami azonnal sikert aratott. Később – látva a kedvező eladási adatokat – az IBM újabb és újabb modellekkel jelentkezett, míg végül más cégek is gyártani kezdték az ilyen gépeket, tovább csökkentve azok árát, és növelve elterjedtségüket. Paradox módon napjainkban épp az IBM nem gyárt személyi számítógépeket, miután az IBM-PC üzletágát 2005-ben eladta a kínai Lenovonak.

A PC-k általános felépítése

Az PC asztali változatának fizikai felépítése nem sokat változott a megjelenése óta.



{á:m1e2a12.png}

12. ábra

Az asztali PC és a laptop felépítése

A különböző részegységek az alaplaphoz (2) csatlakoznak. Ezen helyezkedik el a processzor (3) és a memóriamodulok (4) csatlakoztatására szolgáló foglalatok, valamint a gép rugalmas bővíthetőségét biztosító csatlakozók. Bővíthetjük a gépet például szabványos bővítőkártyákkal (5), merevlemezes – HD – (8) vagy optikai – CD, DVD – (7) meghajtókkal. Az előbb említett eszközök a tápellátást biztosító tápegységgel (6) együtt a számítógépházban (1) kaptak helyet. A gépház hátulján további csatlakozók állnak rendelkezésre a külső perifériák, billentyűzet (9), egér (10), monitor (11) csatlakoztatására. Ezek a perifériacsatlakozók vagy közvetlenül az alaplapra vannak építve, vagy bővítőkártyákra szereltek, és ezeken keresztül csatlakoznak az alaplaphoz.

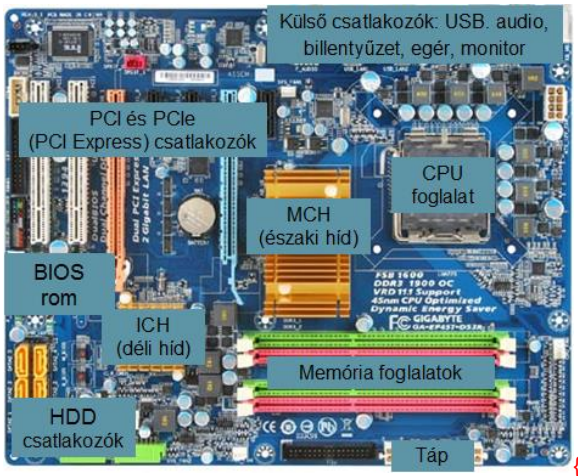
A moduláris felépítésnek köszönhetően a részegységek mindegyike cserélhető akár egy másik gyártó kompatibilis termékére is. A hordozható számítógépek tervezésekor a mobilitást helyezik előtérbe a bővíthetőséggel szemben. Ezekben a gépekben legfeljebb az operatív memória bővíthető, más nem. Az egyes részegységek cseréje is csak korlátozottan lehetséges, mivel a legtöbbjük felépítése a gyártóra jellemző. A bővíthetőség is hasonlóan támogatott (asztali gépeknél nagyon széles körűen, laptopoknál erősen korlátozottan).

Alaplap

Az alaplap (mainboard, motherboard) egy nyomtatott- és integráltáramkör-rendszert tartalmazó lap. Áramköreinek egy része a tápegységről kapja az elektromos feszültséget a működéséhez, más részüket állandó feszültséggel látja el egy kis elem. Az alaplap legfontosabb (beépített) részei: memóriavezérlő (MMU), megszakításkezelő, DMA-vezérlő, órajel-generátor, valós idejű óra és flashmemória. Ezeknek az áramköröknek egy részét önálló integrált áramköri lapkán (chipben), más részét összevontan, úgynevezett multichipekben szokták megvalósítani. Az alaplap integrált áramköri lapkáinak összessége a lapkakészlet (chipset).

Két fontos kapcsolatot megvalósító áramkör is található az alaplapon. A CPU-t a memóriával és a gyors PCI-busszal (Peripheral Component Interconnect) vagy a PCIe-busszal (Peripheral Component Interconnect express) összekötő egység az északi híd. A PCI-buszt a lassú ISA-busszal (Industry Standard Architecture) a déli híd köti össze. A híd (bridge) feladata a különböző működésű adattovábbító rendszerek összehangolása. A hidakba manapság már beépítik azokat az áramköröket, amelyek a kommunikációt segítik. Az északi híd például tartalmazhatja az MMU-t, a DMA-t, a megszakítás-vezérlőt is. A PCI buszra általában a háttértárak és a gyors perifériák, míg az ISA buszra a lassú perifériák vezérlői csatlakoznak. A modernebb PC-ken már nem használnak ISA buszt. Ezeken a déli híd egy ICH (I/O Controller Hub) azaz I/O-vezérlő csatlakozójának elosztója.

A processzor és az operatív memória nem része az alaplapnak, szabványos foglalatokba illesztve kapcsolhatók hozzá. Ugyancsak szabványos csatlakozón kapcsolódnak a különböző bővítőkártyák, amelyekkel a PC-hez kapcsolható perifériák körét bővíthetjük. Ezek a kártyák a PCI processzorfüggetlen buszra csatlakoznak. Régebbi személyi számítógépeken szinte az összes periféria ilyen kártyákkal volt csatlakoztatható, a mai modern gépeken a perifériacsatlakozók egy részét az alaplapra integrálták.



{á:m1e2a13.png}

13. ábra
Az asztali PC alaplapja

Az alaplapon található a PC elindításához szükséges ROM is. Ez a chip a régebbi gépeken a BIOS-t (Basic Input Output System, magyarul alap beviteli, kiviteli rendszer) tartalmazta. Ez a programrendszer indult el a gép bekacsolásakor. Szolgáltatásaihoz tartozott a hardvereszközök állapotának ellenőrzése, és az operációs rendszer betöltése az operatív memóriába. A legújabb PC-kben az a ROM egy kisméretű flashmemória, ami csak az operációs rendszer betöltését indítja el, az összes többi szolgáltatás már ennek a dolga.

Az alaplap beépített áramkörei meghatározzák azt is, hogy milyen processzor, memória és bővítőkártya használható. A csatlakozók kialakítása is kizárja némelyik típust a bővítésből, de ez még nem elegendő a valóban alkalmazható elemek kiválasztására. A pontos meghatározás az alaplap kézikönyvéből, illetve internetes kereséssel végezhető el.

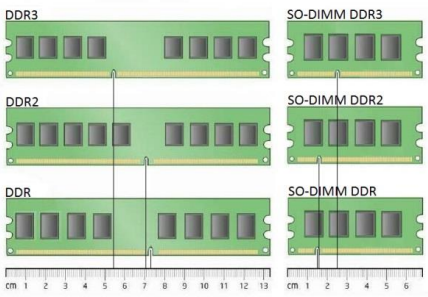
Processzor és memória

A legelső IBM PC az Intel Corporation által 1979-ben megjelentetett 8088-as processzora köré épült. Ez az egy évvel korábban kifejlesztett 8086-os processzor egyszerűsített változata. Mivel a fejlesztése során alapvető fontosságú volt a kompatibilitás megőrzése, ezért a későbbi gépek is a 8086 processzorcsalád tagjaira (286, 386, 486, Pentium stb.) – összefoglaló nevükön – az x86 architektúrájú processzorokra épültek. Nemcsak az Intel gyárt x86 architektúrájú processzort. A legnagyobb konkurense az AMD, de több más kisebb gyártó is készít vagy készített ilyen processzorokat például a VIA, a Transmeta vagy a Cyrix.

Egy tipikus modern x86 architektúrájú asztali PC-be szánt processzor 64 bites (x86-64 architektúra). Ez azt jelenti, hogy a regisztereinek mérete 64 bit. 2-3 GHz frekvenciájú órajelen működik, és az ismertetett technikai megoldások szinte mindegyikét (20-30 fokozatú pipeline, szuperskalár-működés, out-of-order utasítás-végrehajtás, vektorutasítás-készlet stb.) alkalmazza.

Ahogy a legtöbb általános célú számítógépben, az IBM PC-kben is az SDRAM-ot használják operatív tárként. Bár az alapvető működési elv változatlan, ennek a memóriának az évek során több, egymással nem kompatibilis típusát fejlesztették ki. Napjainkban leginkább a DDR2 és DDR3 SDRAM memóriákkal találkozhatunk a gépekben (DDR = Double Data Rate, kétszeres adatátviteli sebesség; a szám a változatok sorszáma). Különböző formátumú modulokat használnak az asztali gépekben és a laptopokban. Ezek a SO-DIMM (Small Outline Dual In-line Memory Module, kisméretű kétoldali

oldalanként önálló csatlakozójú memóriamodul) DDR RAM-ok. Hogy a különböző RAM-ok a látszólag egyforma kivitel ellenére megkülönböztethetőek legyenek, a csatlakozó felület kiképzését is megváltoztatták.



{á:m1e2a14.png}

14. ábra

Asztali számítógépekbe és laptopokba való DDR memóriatípusok

Háttértárak

A merevlemezegység, a HDD (Hard Disk Drive), adathordozója a merevlemez, a HD (Hard Disk). Működése során egy elektromágneses tekerccsel vezérelt mágnesező részt/indukciós részt tartalmazó egység (író/olvasó fej) írja fel/olvassa le a tárolandó/leolvasandó kódokat egy kör alakú, gyorsan forgó lemez mágnesezhető felületére/felületéről. Egy lemezmeghajtón több, egy tengelyre, egymás fölé szerelt lemez is lehet. Általában mindegyikhez két író/olvasó fej tartozik. Az adatok koncentrikus körök, vagy más néven sávok (angolul track-ek) mentén vannak tárolva. Az egymás alatt lévő sávokat cilindernek hívják. A sávok szektorokból állnak. Egy szektor adatai egyetlen művelettel írhatók, illetve olvashatók. A mai HDD-k már több szektort tudnak együtt kezelni. A szektor mérete az első HDD-kben általában 256 bájt volt, a mai szektorméret 512 bájt. Az átvitel gyorsítására itt is alkalmaznak cache-t, amit a vezérlő elektronikával és a csatolófelülettel együtt a merevlemez-meghajtóba építenek.

A merevlemezen tárolt adatok (kódok) logikai egységeket alkotnak. Ezek a logikai egységek a fájlok (file). Kialakításukat a felhasználók által használt programok és az operációs rendszerek végzik. A fájlokat a lemezen egy hierarchikus rendszerbe (könyvtárrendszerbe, mapparendszerbe) szervezik. Ezt a rendszert alapvetően az operációs rendszer segítségével a felhasználók alakítják ki.

A HDD-k legújabban az SCSI (Small Computer System Interface) buszra, korábban a SATA-buszra (Serial Advanced Technology Attachment) csatlakoznak, ami soros átvitelt biztosít. Az SCSI busz vezérlője általában a PCI vagy a PCIe buszok valamelyikére kapcsolódik.

{á:m1e2a15.png}

15. ábra

Merevlemezes meghajtó szétszedett állapotban

A merevlemez-meghajtók általában a számítógép belső egységei, de léteznek külső HDD-k is.

HDD-t használva tartsuk szem előtt, hogy érzékeny a külső behatásokra (tönkremehet). Bár a modern meghajtók belső jellemzőik folyamatos mérésével képesek előre jelezni a közelgő meghibásodást, a merevlemezről mindig készítsünk biztonsági másolatot.

A legújabb fejlesztések következményeképpen a HDD-k mellett megjelentek a flashmemóriás háttértárak is. Ezek a teljesen elektronikus működésű SSD (Solid-State Drive, szilárdtest meghajtó) tárolók. Mivel csak korlátozott számú újraírást viselnek el, kiegészítő áramkörök gondoskodnak a cellák egyforma terheléséről és a hibás cellák helyettesítéséről. Általában a SATA-buszra csatlakoztathatók, de készülhetnek más buszra, például a PCIe-buszra vagy USB-re csatlakoztatható SSD-k is. Nagy előnyük a HDD-kkel szemben, hogy mozgó alkatrészt nem tartalmaznak. Kezelésükhöz nem kell külön meghajtó egység, vezérlőjük a memóriával egy egységet alkot. Hátrányuk, hogy kevésbé megbízhatók, azaz jóval hamarabb tönkremennek, mint a HDD-k. Egyik fajtájuk a külső egységként használható pendrive.

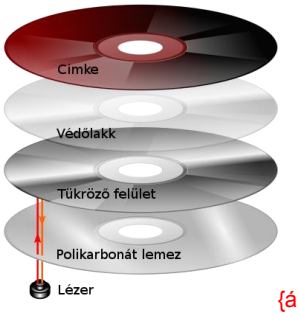
{á:m1e2a16.png}

16. ábra

SATA buszra csatlakoztatható SSD meghajtók és belső felépítésük

A háttértárak egy másik csoportját alkotják az optikai elven működő tárolóegységek, ezek archiválásra, nagymennyiségű adat megőrzésére alkalmasak. Ide tartoznak a különböző CD (Compact Disc, kompaktlemez), DVD (Digital Versatile Disc, digitális sokoldalú lemez) és a BD (Blu-Ray Disc, kék sugár lemez, a Blu-t az angol blue szóból alkották) adathordozó-lemezek írására és olvasására egyaránt képes meghajtók. Mindhárom típus rétegzett szerkezetű lemezeket használ az adatok (kódok) tárolására. Az adathordozó egy fényvisszaverő képességű vékony fémréteg, amin

létrehozott gödrök (pit) és púpok (bump) jelenítik meg a tárolandó adatokat. Léteznek többrétegű adathordozók is, ekkor az egyik réteg a fényt féligáteresztő tulajdonságú. Ezek a rétegek egy fényáteresztő lemezre van rögzítve, föléjük lakkréteg, majd arra címke kerül.



{á:m1e2a17.png}

17. ábra
Az optikai lemez rétegei és felépítése

Az adatok leolvasása úgy történik, hogy egy lézersugárral átvilágítjuk a fényáteresztő lemezt, és a sugár visszaverődik a felette lévő tükröződő felületről. A gödrök és púpok helyén a visszavert lézerfény intenzitása más és más lesz. Ezt az intenzitáskülönbséget érzékeli egy detektor, amelynek kimenetéből erősítés és hibajavítás után megkapható a tárolt adat (kód) elektronikus megfelelője.

Az optikai lemezeken az adatok egy spirális pálya mentén helyezkednek el. A tárolható adatmennyiség függ a tároló rétegen lévő púpok, gödrök sűrűségétől, illetve arányos a lemez adathordozó rétegeinek számával. A következő táblázatban bemutatjuk a tipikus tárolási kapacitásokat egyoldalas, egyrétegű 12 cm átmérőjű lemezek esetében.

CD	700 MB
DVD	4,7 GB
BD	25 GB

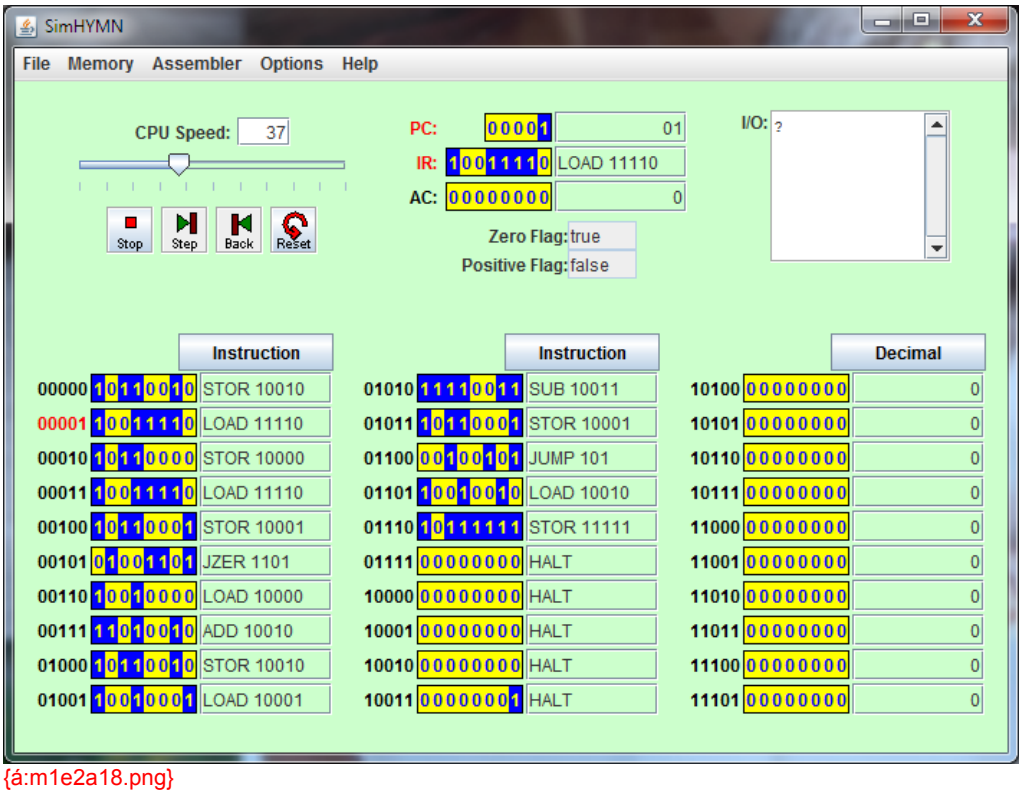
4. táblázat
Az optikai tárolók tipikus tárolási kapacitása

Valódi háttértárként való alkalmazásuk (még) nem elterjedt, mivel a meghajtók lassúak (különösen az írási műveleteknél), és adathordozóik nem is mindig újraírhatók.

SimHYMN

Ahogy fent már említettük, a számítógép − leegyszerűsített − működését egy szimulátor program segítségével bemutatjuk.

Tevékenység: Nyissa meg a SimHYMN CPU szimulátort! (A program használatához szükséges az Ön számítógépén a Java futtatókörnyezet. Ha ez nincs telepítve, akkor töltse le a http://java.com/en/download helyről, és értelemszerűen telepítse.)
Programfájlra hivatkozni!



{á:m1e2a18.png}

18. ábra

A SimHYMN felülete

A szimulátor ablakának felépítése

Vizsgáljuk meg először a program ablakának felépítését. A képernyő felső felében látjuk a vezérléshez szükséges eszközöket, a regisztereket, és az input, output kezelésére szolgáló ablakot. Az alsó részen a memóriarekeszek és a hozzájuk tartozó címek láthatóak. Megfigyelhető, hogy ennek az egyszerű számítógépnek nincs elkülönített adat- és programmemóriája. (Tehát a gép Neumann architektúrájúnak tekinthető.)

A számítógép modellünk három (két speciális és egy általános célú) regiszterrel rendelkezik.

A speciális célú PC (Program Counter, utasítás számláló) regiszter funkcióját már ismerjük: ez jelzi, hogy a számítógép hol tart az utasítások végrehajtásában. A PC mindig a következő végrehajtandó utasítás címét tartalmazza.

Tevékenység: A PC regiszter 5 bites. Számolja ki, hány memóriacímet lehet kezelni vele!

5 bit segítségével 2^5, azaz 32 memóriacímet tudunk megkülönböztetni.

Tevékenység: Számolja meg, hány rekeszből áll a memória!

A szimulátornak memóriába ágyazott I/O-ja van, ezért csak az első 30 (0-tól 29-ig sorszámozott) memóriacímet használja az adatok, programok tárolására, az utolsó kettőt a be- és kimenet (30, 31) számára tartja fenn.

A másik speciális célú regiszter az IR (Instruction Register, utasításregiszter). Ebbe a regiszterbe olvassa be a gép a memóriából a végrehajtandó utasítást, majd ez alapján végzi el az adott műveletet.

A számítógépnek egy általános célú regisztere van, ez az AC (Accumulator). A szimulátor az akkumulátorba olvassa be az aritmetikai és logikai műveletekhez szükséges adatokat (a műveletek egyik operandusát), illetve itt tárolja az aktuálisan elvégzett számítás eredményét.

A szimulátor a(z egész) számokat előjelesen tárolja.

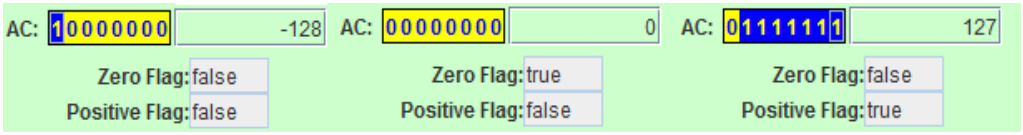
Tevékenység: Melyik a legkisebb és a legnagyobb tárolható szám a Hymn szimulátorban?

Mivel a memória bájtos szervezésű (8 bit = 1 bájt), ezért a program 2^8 = 256 féle számmal tud dolgozni. A legkisebb ábrázolható szám a -128, a legnagyobb a $+127$.

Tevékenység: Állítsa át az AC regiszter bitjeit úgy, hogy a legkisebb, majd a legnagyobb 8 bites előjeles szám szerepeljen benne! (A biteket kattintással lehet módosítani.)

Két mutatóval is rendelkezik ez az egyszerű számítógép, a Zero és a Positive Flag-gel. A Zero Flag értéke akkor válik igazzá (true), ha az akkumulátorban lévő adat értéke 0, a Positive Flag pedig értelemszerűen azt jelzi, hogy az akkumulátorban pozitív szám van-e.

Tevékenység: Módosítsa az akkumulátor értékét, és figyelje meg, hogyan változnak a flag-ek!



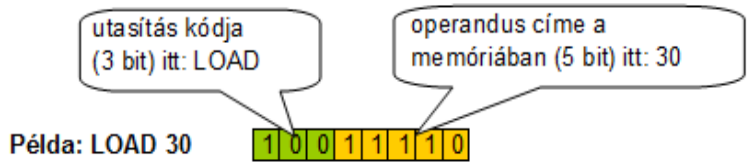
{á:m1e2a19.png}

19. ábra

A legkisebb és a legnagyobb 8 bites szám

A szimulátor utasításai

A Hymn Cpu szimulátor utasításai egységesen úgy épülnek fel, hogy az első 3 bit maga az utasítás, a következő 5 pedig a művelethez tartozó operandus címe a memóriában. Ennek az egyszerű számítógépnek így tehát 2^3 = 8 utasítása van.



{á:m1e2a20.png}

20. ábra

Példa egy utasítás felépítésére

A lehetséges utasításokat a következő táblázat tartalmazza.

Kód	Mnemonik	Leírás
000	HALT	nem történik semmi
001	JUMP	PC := operandus (ugrás az operandus címre)
010	JZER	ha AC = 0, akkor PC := data (ugrás az op. címre) egyébként pedig PC := PC + 1 (következő utasítás végrehajtása)
011	JPOS	ha AC > 0, akkor PC := data (ugrás az op. címre) egyébként pedig PC := PC + 1 (következő utasítás végrehajtása)
100	LOAD	AC := M[data]; PC := PC + 1 (az operandus által megcímzett memória tartalmának betöltése AC-be)
101	STOR	M[data]:=AC; PC := PC + 1 (AC tartalmának kiírása az ope-randussal által megcímzett memória cellába)
110	ADD	AC := AC + M[data]; PC := PC + 1 (AC regiszter tartalmához hozzáadja az operandussal megcímzett memória tartalmát)
111	SUB	AC := AC - M[data]; PC := PC + 1 (AC regiszter tartalmából kivonja az operandussal megcímzett memória tartalmát)

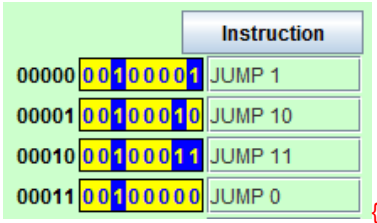
5. táblázat
A szimulátor utasításkészlete

Egyszerű programok készítése

Első programunkat egérkattintással, a megfelelő bitek átbillentésével készítjük el. Ez a program még semmilyen számítás műveletet nem fog végrehajtani, csak ugrálni fog az egyes memóriacímek között.

Feladat: A nullás memóriacímről ugorjunk át az egyesre, a kettesre, a hármasra, majd végül vissza a nullásra!

Tevékenység: Készítse el a programot a következő ábra alapján!



{á:m1e2a21.png}

21. ábra
Első programunk

Tevékenység: Futtassa a programot lépésenként a Step nyomógomb segítségével, és figyelje meg, hogy változnak az egyes regiszterek értékei.

A program pirossal jelöli azt a memóriacímet, amivel az adott lépésben éppen dolgozik.

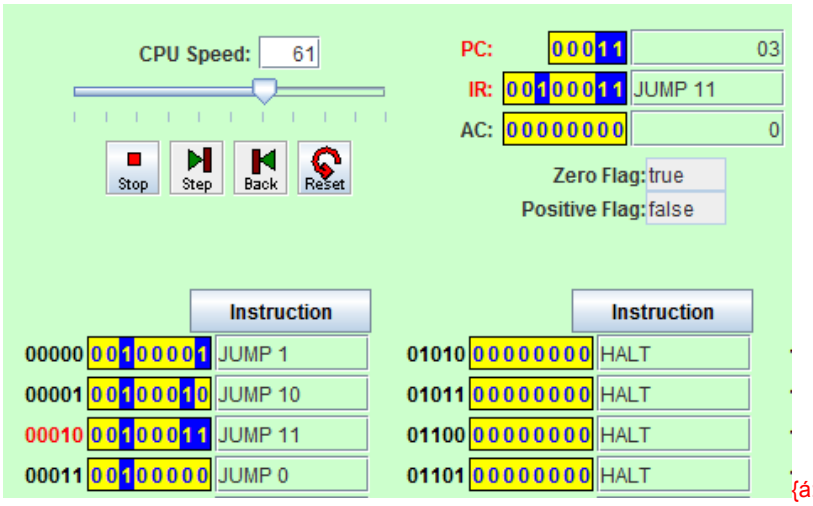
Mivel logikai, aritmetikai és adatmozgató műveletet nem végzünk, az AC regiszter értéke nem változik a futás során. Az IR regiszterben mindig az aktuálisan pirossal jelölt memóriacímről beolvasott utasítás található, a PC regiszterben pedig az a memóriacím, amire az adott lépésben ugrani akarunk.

Ezeknek a regisztereknek a bitjeit kézzel is át tudjuk állítani.

Tevékenység: Módosítsa a PC, illetve az IR regiszter bitjeit és nézze meg, hogy mi történik a következő lépésben!

A regisztereket alaphelyzetbe tudjuk állítani a Reset nyomógombbal, a programot lefuttatni pedig a Play gomb segítségével lehet.

Tevékenység: Állítsa alaphelyzetbe a regisztereket, majd futtassa a programot! A futás közben módosítsa a CPU Speed értékét a csúszka segítségével!



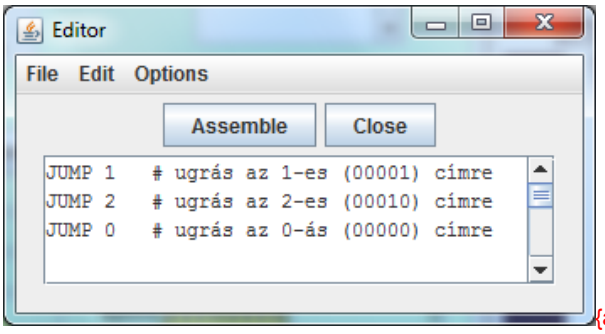
{á:m1e2a22.png}

22. ábra
A memóriacímek között ugráló program futás közben

Ez a program egy végtelen ciklust valósít meg. (A ciklusokkal bővebben a Számítási módszerek tárgyban fogunk foglalkozni.)

A programot nem csak bitek átbillentésével, hanem mnemonikok segítségével (assembly nyelven) is el lehet készíteni (ez a módszer legalább 6-8 utasításnál már biztosan egyszerűbb, és így a kód is könnyebben értelmezhető). Ehhez nyissuk meg a kódszerkesztő ablakot az Assembler/Show Editor paranccsal. Figyeljünk arra, hogy az Editor ablakban a memóriacímeket már nem binárisan, hanem decimálisan kell megadni!

Tevékenység: Írja be az Editor ablakba a következő ábrán szereplő programot, majd a töltse be a memóriába az Assemble nyomógomb segítségével! Figyelje meg, hogyan kerülnek be az utasítások a memóriába!



{á:m1e2a23.png}

23. ábra

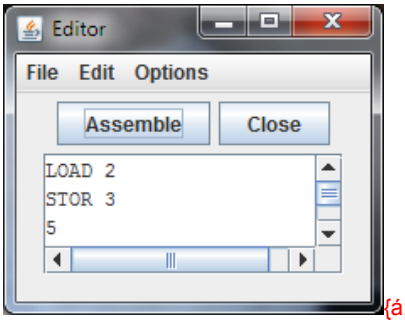
Ugráló program az Editor ablakban

Az Editor ablakban kommenteket is írhatunk az utasítások után, ezek a # jellel kezdődnek.

Második programunk már adatmozgatást is fog végezni.

Feladat: A kettes memóriacímről töltsünk át egy adatot a hármas memóriacímre!

Tevékenység: Készítse el a következő ábrán lévő programot, majd futtassa le!



{á:m1e2a24.png}

24. ábra

Adatmozgatás

A LOAD 2 utasítás betölti a kettes (00010) memóriacímen lévő adatot (jelen esetben 5) az AC regiszterbe, majd a STOR 3 kiírja az AC regiszter értékét a hármas (00011) memóriacímre.

Tevékenység: Futtassa le az előző programot lépésenként úgy, hogy a betöltés után, a STOR 3 parancs előtt módosítja az AC regiszter értékét!

Az előző tevékenységgel bizonyítható, hogy a STOR parancs az AC regiszter értékét írja ki az adott memóriacímre.

Tevékenység: Módosítsa úgy az előző programot, hogy az 5 helyére −92-t ír, majd futtassa le!

Mivel a számítógép együtt, közös memóriában tárolja az utasításokat és az adatokat, a −92 lefutott egy STOR 4 utasításként.

	Decimal
00000 10000010	-126
00001 10100011	-93
00010 10100100	-92
00011 10100100	-92
00100 10100100	-92
00101 00000000	0

	Instruction
00000 10000010	LOAD 10
00001 10100011	STOR 11
00010 10100100	STOR 100
00011 10100100	STOR 100
00100 10100100	STOR 100
00101 00000000	HALT

{á:m1e2a25.png}

25. ábra

Utasításként kezelt adat

Azért, hogy ezt a hibát kikerüljük, egy HALT utasítással le kell állítani a program futását a STOR utasítás után. Ha beírjuk a HALT utasítást, akkor az adat egy rekesszel hátrébb kerül a memóriában, ezért a memóriacímeket is át kell írni a programban.

Tevékenység: Javítsa ki a programot egy megfelelően beszúrt HALT utasítással!

Az Assembler Editorban memóriacímre mutató hivatkozásokat, címkéket is elhelyezhetünk. A címkék használatával már nem kell „bedrótozni” a programba azt, hogy melyik memóriacímen van az adat, ez leegyszerűsíti a kód elkészítését és javítását.

Tevékenység: Egészítse ki az előző programot megfelelő címkékkel (pl. a és b)!

```
LOAD a
STOR b
HALT
a: -92
b: 0
```

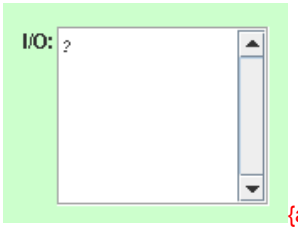
source_code="y"

Ahogy a fejezet elején már említettük, a számítógép memóriába ágyazott I/O-val rendelkezik, a 30-as memóriacím a bemenet, a 31-es pedig a kimenet kezelésére szolgál.

Tevékenység: Alakítsa át úgy az előző programot, hogy az „a” címkével jelölt memóriarekesz tartalmát írja ki a kimenetre, majd olvasson be egy számot és tárolja el az „a” címen.

```
LOAD a
STOR 31
LOAD 30
STOR a
HALT
a: -92
```

source_code="y"



{á:m1e2a26.png}

26. ábra
I/O ablak

Az I/O ablakban a program futása során a kérdőjel után kell írni a bemenetnek szánt értéket.

A LOAD 30 utasítást a READ, a STOR 31-et pedig a WRITE paranccsal is lehet helyettesíteni.

```
LOAD a
WRITE
READ
STOR a
HALT
a: -92
```

source_code="y"

Feladat: Az ADD (összeadás) és a SUB (kivonás) művelet segítségével készítsünk programot a $12-23+31$ műveletsor elvégzésére.

Tevékenység: Próbálja ki a következő programot!

```
LOAD a # Az "a" memóriacímen lévő adat betöltése az AC regiszterbe.
SUB b  # Az AC regiszter tartalmából kivonjuk a "b" memóriacímen lévő értéket.
ADD c  # Az AC regiszter tartalmához hozzáadjuk a "c" címen lévő értéket.
WRITE  # Az AC regiszter értékét kiírjuk az outputra.
HALT
a: 12
b: 23
c: 31
```

source_code="y"

Tevékenység: Írja át úgy az előző programot, hogy az a, b és c érték az inputról érkezzen!

Tevékenység: Írja ki címkék segítségével a születési dátumát számjegyenként a kimenetre!

Utolsó kis programunkban a feltételes ugrás (JPOS, JZER) parancsait próbáljuk ki.

Feladat: Írjunk olyan programot, ami bekér az inputról egy számot, majd megvizsgálja annak előjelét: ha pozitív szám érkezik, akkor $+1$-et, ha negatív, akkor -1-et, nulla esetén pedig 0-t ír ki.

Tevékenység: Értelmezze, és futtassa le soronként a következő programot! Nézze meg, hogy hogyan változnak a flagek a futás során!

```
READ            # Adat beolvasása az inputról az AC regiszterbe.
JZER kiir       # A kiir címre ugrás, ha az AC regiszter értéke 0.
```

```
JPOS pluszegybe # A pluszegybe címre ugrás, ha az AC regiszter értéke pozitív.
LOAD minegy     # A -1 betöltése az AC regiszterbe.
JUMP kiir
pluszegybe:
LOAD pluszegy   # A +1 betöltése az AC regiszterbe.
kiir:
WRITE           # Az AC regiszter tartalmának kiírása az outputra.
HALT
minegy: -1
pluszegy: 1
```

source_code="y"

Zárásként újból hangsúlyozzuk, hogy a fejezet célja nem a programozás megtanítása volt, hanem a számítógép belső működésének a bemutatása néhány egyszerű program segítségével. A programozással a következő félévben a Számítási módszerek tárgy keretein belül fogunk részletesebben foglalkozni.

teszt rész

Önellenőrző kérdések

1. Rendezze nagyság szerint növekvő sorrendbe az alábbi értékeket!

1 8 bájt

2 0,01 kB

3 0,04 kB

4 0,05 KiB

5 100 bájt

2. Rendezze nagyság szerint növekvő sorrendbe az alábbi értékeket!

1 0,4 kB

2 500 bájt

3 2 kB

4 2 KiB

5 3000 bájt

3. Melyik állítás igaz, melyik a hamis?

✓ Az 1 KiB nagyobb mint 1 KB.

✗ Az 1 kB ugyanakkora, mint az 1 KiB.

× A memória nagyságát Kib-ben szokás megadni.

✓ A CU működési sebességét MIPS mértékegységben is meg lehet adni.

4. Melyik állítás az igaz, melyik a hamis?

× A soros átvitelhez több szálú vezeték kell, mint a párhuzamoshoz.

× A soros átvitelhez ugyanannyi szálú vezeték kell, mint a párhuzamoshoz.

✓ A soros átvitelhez kevesebb szálú vezeték kell, mint a párhuzamoshoz.

× Az átviteli sebesség mértékegysége a MIPS.

5. Melyik állítás az igaz, melyik a hamis?

× Az ALU önállóan állapítja meg, hogy milyen műveletet kell elvégeznie egy utasítás végrehajtása közben.

✓ A CPU a számítógép egyik fő egysége, ami a CU-t és az ALU-t is tartalmazza.

× A regisztertömb a memória része.

✓ A memóriába ágyazott input és output eszközök kezeléséhez nincs szükség külön gépi input és output műveletekre.

6. Melyik állítás az igaz, melyik a hamis?

× Neumann tanulmányának elkészülését megelőző időkben nem is gondoltak arra, hogy a kettes számrendszer alkalmas lehet a számítógépes kódolásra.

× Az EDVAC megépítését megelőző időkben csak elektromechanikus alkatrészekből (relékből) építettek számítógépet.

✓ Az EDVAC elektroncsöves számítógép volt.

× Az első mechanikus számítógép megtervezője Jackard volt.

× Az első elektronikus számítógépet Babbage tervezte.

7. Melyik állítás az igaz, melyik a hamis?

× Az első műveletvégző egységek egyike sem tudott közvetlenül valós számokkal műveletet végezni.

✓ Neumann számítógéptervében nem szerepelt logikai műveletek elvégzésére képes műveletvégző egység.

× Az ALU csak aritmetikai műveletek tud végezni.

8. Melyik állítás az igaz, melyik a hamis?

- × Neumann nem tervezett órajelet előállító áramkört a számítógépéhez.
- × Neumann János Alan Turinggal közösen tervezte az első elektronikus számítógépet.
- × Neumann számítógépének memóriája kikapcsolás után is megőrizte tartalmát.

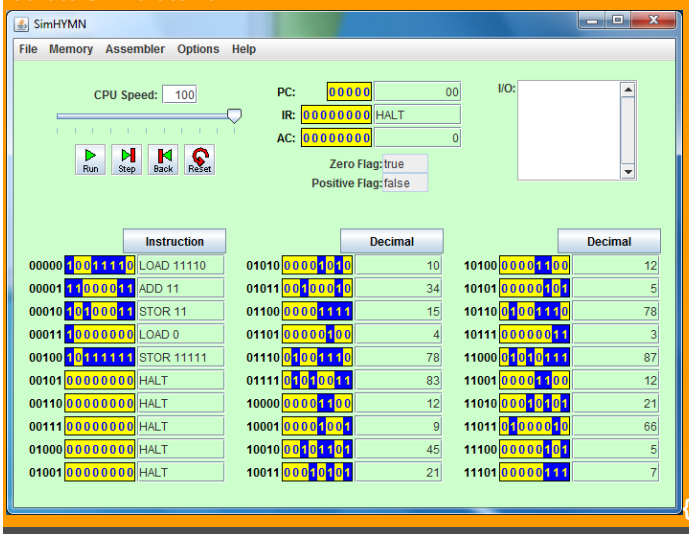
9. Melyik állítás az igaz, melyik a hamis?

- × A PCI és a PCIe buszokkal a memóriából közvetlenül lehet a perifériákra (például egy nyomtatóra) kódot továbbítani.
- ✓ Az USB-re akár 127 periféria is csatlakoztatható, és ezekre a perifériákra az USB-n sorosan érkeznek a kódok.
- × Az operatív memória is az USB-re csatlakozik.

10. Mennyi lesz az alábbi kifejezés eredménye, ha számítógépünk csak egybájtos előjeles adatípust használ (Hymn CPU szimulátor)?

Kifejezés: $-19-102-111$
Eredmény: 24

11. Az alábbiak közül jelölje meg azokat a műszaki megoldásokat, amelyek jellemzőek a képen látható szimulátorra!



{á:m1e2f11.png}

☑ utasításregiszter

☐ Hyper-Threading technológia

☑ belső memória

- ☐ háttértár
- ☐ túlcsordulást jelző flag
- ☑ soros utasítás-végrehajtás
- ☐ párhuzamos utasítás-végrehajtás
- ☐ előjel nélküli egész aritmetika

3. lecke: Operációs rendszerek

Cél: Ebben a leckében áttekintjük azokat az ismereteket, amelyeket az operációs rendszerekkel kapcsolatban szükséges ismerni ahhoz, hogy az Informatika I. és II. tárgy további feldolgozása „gördülékeny” lehessen. Ez a bemutatás viszonylag tömör, csak a legfontosabb elemekre összpontosít. Általános bevezetést követően megismerkedünk az operációs rendszerek feladataival (többek között: folyamatok kezelése, fájl- és mapparendszer menedzselése), áttekintjük azt a folyamatot, ahogy az operációs rendszer elindul, és bemutatunk néhány jelentősebb operációs rendszert is. Hasonlóan az előző leckéhez, egyes részek itt is olvasmányos jellegűek (főként a konkrét operációs rendszerek fejlődéstörténete – a rendszerek kiemelt fontossága miatt azonban nem akartuk elhagyni a leíró részt).

Szintén az előző lecke felépítését követve, az elméleti ismeretek alkalmazását most is egy gyakorló feladatblokkal segítjük. Ez a feladatcsoport a Total Commander program segítségével végigvezeti a hallgatót mindazokon a tevékenységeken, amelyekre a további informatikai tanulmányok során szükség lesz.

Követelmények: Ön akkor sajátította el megfelelően a tananyagot, ha

- összefoglalóan ismertetni tudja az operációs rendszer főbb jellemzőit (virtuális környezet, kezelési biztonság, erőforrások kezelése);
- képes felsorolni, ill. listából kiválasztani az operációs rendszer feladatait;
- saját szavaival képes ismertetni az operációs rendszer elindulásának a folyamatát;
- saját szavaival el tudja mondani a megismert fájlrendszerek főbb jellemzőit;
- végre tudja hajtani a Total Commander program segítségével a leckében bemutatott fájl- és könyvtárműveleteket;
- saját szavaival be tudja mutatni a megismert operációs rendszereket (főbb jellemzők).

Időszükséglet: A tananyag elsajátításához (a feladatok megoldásával együtt) hozzávetőlegesen 3 órára lesz szüksége.

Kulcsfogalmak

- Virtuális gép.
- Az operációs rendszer feladatai.
- BIOS és kernel, felhasználói interfészek.
- Fájl- és mapparendszer (aktuális könyvtár, szülőkönyvtár, elérési utak stb.), műveletek (mappa létrehozása, fájlok másolása, mozgatása stb.).
- Commander programok.
- Windows, UNIX és Linux operációs rendszerek.
- Az operációs rendszerek általános tulajdonságai.

Az operációs rendszerekről általában

Képzeljük el egy pillanatra, hogy az újonnan megkapott számítógépünkhöz nincs szoftveres támogatás, azaz a hardvert teljes egészében nekünk kell működésre bírni! Megfelelő hardver

ismeretek birtokában a gép normális működési folyamatának szinte bármely részét kiemelhetjük, és elemezhetjük, hogy a háttérben milyen történések zajlanak – ezeket kellene megvalósítanunk. Sorra végigmehetünk az indítás, hardver eszközök ellenőrzése stb. folyamatokon, de nagyon hamar adódik az a nyilvánvaló következtetés, hogy egészen egyszerű tevékenységek megvalósítása is rendkívül komoly befektetett munkát kívánna. (Nyomtatásnál például döntenünk kellene arról, hogy miféle elektronikus jeleket küldjünk a lézernyomtató kábelére ahhoz, hogy az épp a nekünk tetsző mintázatot égesse a papírra.)

Szükség van tehát olyan programrendszerre, amely ezt a munkát leveszi a vállunkról.

Meghatározás: Az operációs rendszer (Operation System, OS) olyan alapvető fontosságú programcsomag, ami a hardvert közvetlenül kezeli, a felhasználó alkalmazásainak pedig egységes környezetet biztosít.

Tevékenység: Idézze fel, hogy milyen korábbi definíciót tanultunk az operációs rendszerről az előző leckében! Mit gondol, problémát okoz az, hogy a két megfogalmazás között különbség van?

Az operációs rendszer használata a mai számítógépeken – lényegében – megkerülhetetlen. Ez a program a gép bekapcsolásakor elindul, és a kikapcsolás (pontosabban a szabályos leállítás) előtt ez az utolsó program, ami más programok leállítása után utolsóként a processzort is leállítja.

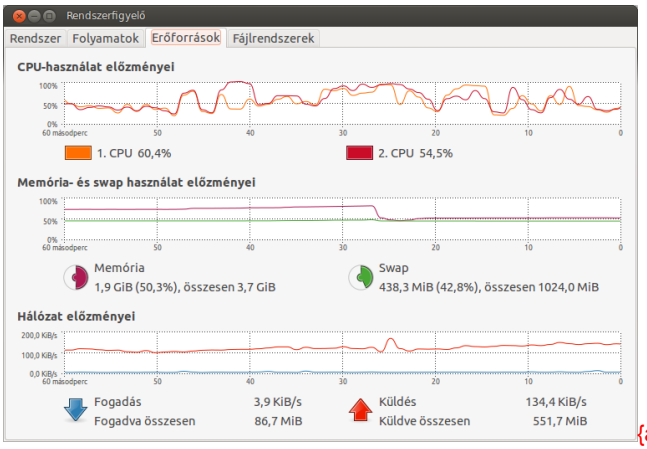
Az operációs rendszer a hardver eszközök kezelése közben elrejti azok – felhasználó számára érdektelen – részleteit. A felhasználó – egy magasabb absztrakciós szinten – a valódi hardver eszközök helyett egy egységes, kényelmes virtuális környezetet lát és ezt is kezeli. Ezen tulajdonsága alapján az operációs rendszert egy virtuális gépnek tekinthetjük. Az eszközök és a fájlok (lásd később) könnyen megjegyezhető, szimbolikus nevekkel kezelhetők, a programok kényelmesen, a technikai részletek ismerete nélkül indíthatók, leállíthatók, és sok-sok magas szintű parancs segíti a munkát (például: adatok másolására).

A belső működési részletek elrejtésével és a virtuális felület alkalmas kialakításával a kezelési biztonság is jelentősen fokozható. Itt nemcsak az adatok biztonságára gondolunk, hanem a gép védelmére is. Míg például egy autót ügyetlen vezetője könnyen tönkre tud tenni, addig egy számítógép (a fizikai megsemmisítés esetét kivéve) gyakorlatilag elronthatatlan. A felhasználó legfeljebb csak a saját adatait tudja elveszíteni, hiszen ahhoz – és tipikusan csak ahhoz – biztosít az operációs rendszer teljes felügyeleti jogot számára.

Ha az operációs rendszer feladatát „alulról felfelé”, azaz a gép fizikai alkotórészeinek, valamint a különböző feladatok megoldására szolgáló programok ismeretében közelítjük meg, akkor azt mondhatjuk, hogy az operációs rendszer célja, hogy egy összetett rendszer részeit menedzselje. Ezek a részek a hardver és szoftver erőforrások.

Meghatározás: A számítógép erőforrásainak azokat a szolgáltatásokat nevezzük, amelyeket az operációs rendszer használatakor a felhasználó igénybe vehet.

Egy számítógépen több tucatnyi hardver és még több szoftver eszközt kell kezelni, és ezen erőforrásokat megfelelően elosztani a gépen futó, erőforrásokért versengő programok között. A megfelelő elosztás bonyolult feladat, mert sok, gyakran egymásnak feszülő szempontot kell figyelembe venni: például legyen igazságos és optimális. Optimális, de kinek a szempontjai alapján?



{á:m1e3a01.png}

1. ábra

Az operációs rendszer felügyeli az erőforrásokat (Rendszerfigyelő)

Tevékenység: Sorolja fel azokat az erőforrásokat, amelyeket az ábra szerint az operációs rendszer felügyel!

Mint a történeti részben már bemutattuk, a számítógépek hőskorában (1950-es évek) még nem léteztek operációs rendszerek. Ekkor az operátorok (gépkezelők) végezték a mai operációs rendszerek alapfeladatainak jelentős részét. Az operátor segítségével volt lehetséges egy megírt program futtatása, neki kellett odaadni a kódot, a kor technikai színvonalának megfelelő adathordozón (például lyukkártyán). Az operátor döntött arról, hogy a programot mikor lehet és kell futtatni, a futáshoz szükséges erőforrásokat is ő biztosította a program számára. A program futása után az eredmények olvashatóvá tételében (nyomtatásában) is segédkezett. Ilyen munkastílusban természetesen elsősorban nagyobb méretű számítási feladatok elvégzését volt érdemes a számítógépre bízni. Az operációs rendszerek elterjedésével az operátori munka – legalábbis eredeti formájában – nagyrészt megszűnt.

Az alfejezet lezárásaként végül felsoroljuk azokat a fontos feladatokat (funkciókat), amelyeket az operációs rendszerek elvégeznek. Ezek a következők:

- Folyamatkezelés,
- Fájl- és könyvtárkezelés,
- Memóriakezelés,
- I/O kezelés,
- Hálózatkezelés.

Az utolsó kivételével mindegyikkel részletesen foglalkozunk a továbbiakban.

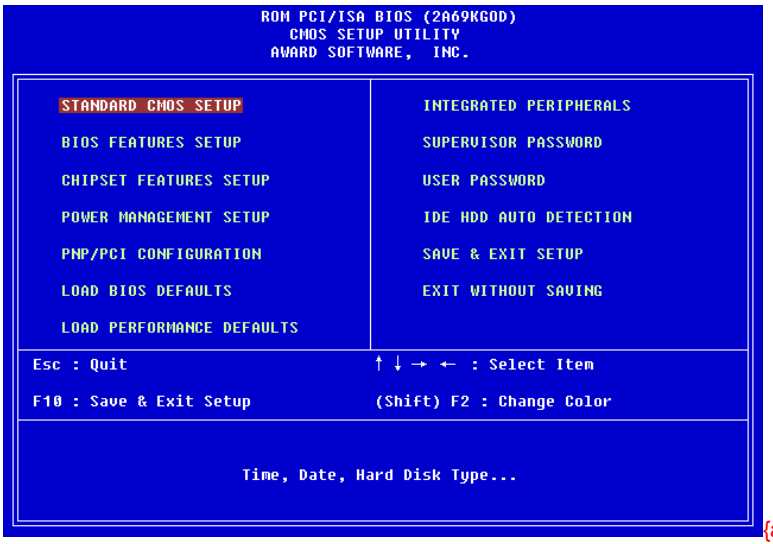
Az operációs rendszer indulása, a BIOS és a kernel

Mint már tanultuk, bizonyos programok bele vannak építve a számítógép hardver elemeibe. Ezek az apró programok képesek többek között arra, hogy önmaguk épségét leellenőrizzék és felismerjék a saját környezetüket. A legfontosabb ilyen program az alaplaphoz és a videokártyához tartozó BIOS (Basic Input-Output System). A BIOS néhány MB méretű és a mai számítógépek zömében

flashmemórián található. Ezen program alapvető feladata, hogy kapcsolatot biztosítson a hardver és szoftver részek között.

A gép bekapcsolásakor, illetve némi késleltetéssel újraindítás után is a BIOS lép működésbe, és különböző gyorstesztekkel feltérképezi a rendelkezésre álló hardverelemeket. Az egyes elemektől kapott válaszok alapján azt is igyekszik eldönteni, hogy azok működőképesek-e (csak alapvető ellenőrzésről van szó). Ha ennek során komolyabb problémát talál, akkor a kiírt hibaüzenet mellett különböző ütemű csipogásokkal jelzi az egyes hibákat. Egyes esetekben csak a csipogás segíthet a hiba azonosításában, hiszen ebben a fázisban még az sem biztos, hogy a videokártya elindult, így előfordulhat, hogy a képernyőn nem látunk semmit.

A BIOS-ba belépve a hardverkörnyezet beállítása egy egyszerű felületen némileg befolyásolható. Például letilthatók egyes eszközök (DVD, alaplapi videokártya), cserélgethető a háttértárak hivatkozási sorrendje, beállítható a pontos idő stb. A BIOS szerepe azonban ezzel együtt is egyre csökken (ez egy régi, több évtizedes technológia terméke), mert ezek a beállítások (egy-két kivételtől eltekintve) már az operációs rendszerből is elvégezhetők.



{á:m1e3a02.png}

2. ábra
Tipikus BIOS képernyő

Tevékenység: Ha lehetősége van rá, lépjen be a gépe BIOS-ába és tekintse meg, hogy milyen beállítási lehetőségek vannak. Indokolatlanul semmit ne módosítson!

Ha a BIOS túljut a hardverek gyorstesztjén, akkor az előre beállított helyeken elkezdi keresni az operációs rendszert. Ha ténylegesen indíthatót talál, akkor azt elkezdi betölteni. Maga az operációs rendszer a rendszermag (kernel) betöltésével indul, ez a rendszer legalapvetőbb szolgáltatásait tartalmazó modul (a hardver erőforrások, többek között a processzor, a memória kezeléséért felel).

{á:m1e3a03.png}

3. ábra
Az operációs rendszer indulása

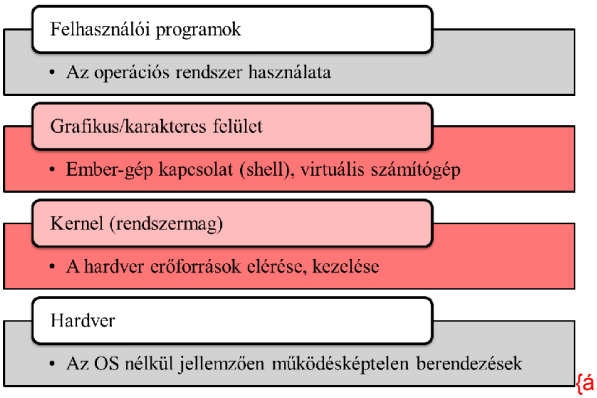
A kernel inicializálja az eszközöket, majd elindítja a különböző speciális feladatokat elvégző, később folyamatosan futó programokat. Jellemzően több tucatnyi olyan program elindul ekkor, amelyek mindegyike egy-egy konkrét funkcióval (szolgáltatással) bővíti az operációs rendszer képességeit (például: nyomtatási sorok kezelése, különböző hálózati és biztonsági funkciók).

A szolgáltatások „felállásával" az operációs rendszer gyakorlatilag működőképessé vált. Az automatizált működés mellett még a felhasználó utasítására is vár (aki esetleg névvel és jelszóval kell, hogy azonosítsa magát – belépés). A felhasználó bejelentkezése után akár még további programok is elindulhatnak (jellemzően apró segédprogramok, például képernyőre kitett óra, időjárás-előrejelző piktogram).

Tevékenység: Windows operációs rendszerében állítsa be, hogy bejelentkezés után automatikusan induljon a számológép (Calc). Ki- és bejelentkezéssel ellenőrizze, hogy beállítás valóban helyes!

Felhasználói interfészek

Egy átlagos felhasználó számára a számítógéppel végzett munka legnagyobbrészt két fő aktivitásból áll: vagy direkt módon az operációs rendszer szolgáltatásait veszi igénybe, vagy valamilyen felhasználói programot futtat és használ. Ezek a programok viszont szintén az operációs rendszerre épülnek rá (4. ábra), a felhasználó tehát végeredményben mindkét esetben az operációs rendszert (is) használja.



{á:m1e3a04.png}

4. ábra
Az operációs rendszer a hardver és a felhasználói programok között

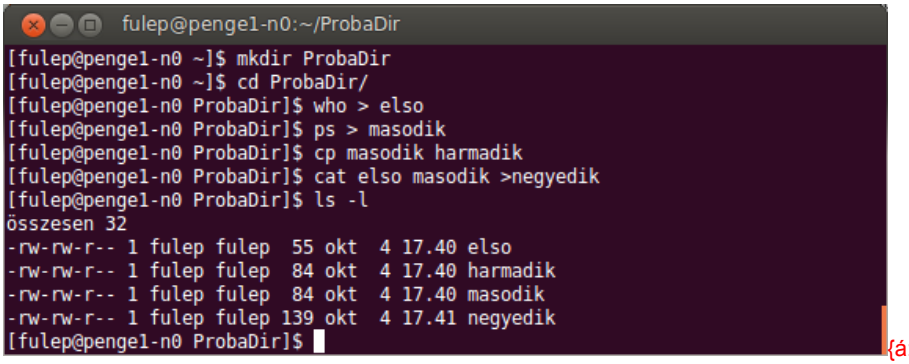
Kijelenthetjük tehát, hogy az operációs rendszer kiemelten fontos részét képezi a felhasználói felület (UI, User Interface, interfész) hiszen ezen keresztül zajlik a kommunikáció az ember és az operációs rendszer között. A felhasználói interfész kidolgozottsága (főként: kényelem) döntően meghatározza a felhasználóban az egész operációs rendszerről kialakult képet is, hiszen az átlagos géphasználó keveset tud az OS memória- és folyamatkezeléséről, biztonsági kérdéseiről és egyéb fontos részletekről – a felület alapján viszont képes ítéletet mondani.

Természetesen a szoftvergyártók is felismerték ezt, ezért szebbnél szebb, egyre praktikusabb és egyre kevesebb szakismerettel kezelhető felületekkel igyekeznek felhasználóikat elbűvölni (és persze újabbakat toborozni). Régebben parancsokat kellett tudni precízen begépelni, később egyre több mindent tettek egérrel irányíthatóvá, manapság pedig az a fő cél (érintőképernyő esetén), hogy természetes ujjmozdulatokkal, rámutatással lehessen irányítani az operációs rendszert.

A használat jellege alapján három alapvető felhasználói felülettípust különböztetünk meg.

A parancssori interfész (Command Line Interface, CLI) a legrégebbi és a technikailag legegyszerűbb típus. Jellemzői a fix méretű monitorkarakterek és a billentyűzet kizárólagos használata. A kommunikáció parancsok begépeléséből és az üzenetek elolvasásából áll. A kisebb erőforrásigény miatt a régebbi rendszereknek csak parancssori felületük volt, ezért manapság többen hajlamosak a karakteres felületet lebecsülni és korszerűtlennek tekinteni. Ez azonban tévedés (és tipikusan a nem kellően hozzáértő felhasználókra jellemző)! A karakteres felület minden korszerű operációs rendszernek is szerves része, mert sokszor jóval hatékonyabb és mindenképpen teljesebb, mint a szebb, grafikus felület!

Illusztrációs példa: amíg a grafikus fájlkezelőkben néhány állítási lehetőség van (pl. ikonos vagy listás megjelenítés, 3-4 rendezési szempont), addig a Linux fájllistázásra való ls parancsának kb. 60 olyan kapcsolója van, amivel a megjelenítést szabályozhatjuk.



{á:m1e3a05.png}

5. ábra

Munka parancssori felületen

Tevékenység: Az irodalomjegyzékben szereplő megfelelő hivatkozások és internetes források alapján tájékozódjon, hogy az ábrán kiadott Linux parancsok hogyan működnek és milyen paraméterekkel (argumentumokkal), ill. kapcsolókkal hívhatók. Ezek után ismertesse, hogy a munkafolyamat (session) végrehajtása során mi történt, mi az eredmény!

Hangsúlyozottan nem a törzsanyag részeként (hanem csak érdekességként) bemutatunk egy olyan feladatot is, ami meggyőzően szemlélteti, hogy a parancssori felület nagy segítségére lehet a hozzáértőknek.

Tegyük fel, hogy le kell töltenünk az internetről egy tömörített (zip) fájlt, amiben egy szövegfájl található, és ennek első háromszáz sorát email-ben el kell küldenünk valakinek. Ezt bárki hamar elvégezheti, de a feladat nagyon unalmassá válik, ha kiderül, hogy ezentúl óránként kell ezt elvégezni a munkahelyünkön. A kedves Olvasó hogyan oldaná ezt meg automatizáltan?

Linux alatt a következő parancs gépelhető be:

```
while true; do wget -q -O - http://itt.hu/ez.zip | gunzip | head -300 | mail
ide@cim.hu; sleep 3600; done
```

source_code="y"

Ezzel a feladatot megoldottuk, parancsunk óránként el fogja küldeni email-ben az újra és újra letöltött fájl megfelelő részét. Nyilván sokkal kényelmesebb és hibamentesebb ezt a parancsot egyetlen alkalommal kigondolni és kiadni, mint óránként, hónapokon keresztül ugyanazt a feladatot alkalmanként 2-3 perc alatt „kézzel”, webböngésző, tömörítő, szövegszerkesztő és levelező program segítségével végrehajtani.

A szöveges interfész (Text User Interface, TUI, „álgrafikus interfész”) olyan fejlettebb felület, amely a képernyőt már nemcsak írógép-stílusban használja, hanem karakteres felületen ablakos technikára emlékeztető képernyőképeket valósít meg. Ezek az interfészek önálló OS-felületként nem terjedtek el, tipikusan a meglévő operációs rendszerekhez készítettek ilyen felületű felhasználói segédprogramokat.

Fénykorukat a PC-s környezetben, a DOS parancssori operációs rendszer idején élték. Első igazán sikeres képviselőjük a Norton Commander volt, ami aztán egész lavinát indított el a fájlkezelő- és mindenféle segédprogramok fejlesztésében. Ezek a programok közvetlenül az operációs rendszer funkcióit bővítették és tették kényelmesebbé, időnként csupán egy jó ötleten alapuló megjelenítési móddal, máskor az OS egyes funkcióinak átvételével.

A kései utód, a Total Commander, a mi Informatika tananyagunkban is fontos szerepet kap, mivel ez a javasolt (elvárt) univerzális eszköz a fájl- és könyvtárkezeléssel kapcsolatos feladatok megoldásához.

A felhasználói felületek harmadik típusát, a grafikus interfészt (Graphic User Interface, GUI) különösebben senkinek nem kell bemutatni, hiszen a modern számítógépeken és mobil eszközökön ez az alapértelmezett. Az ilyen rendszerekben grafikus képernyőfelületen jelennek meg az információk, nem különülnek el élesen a grafikus és szöveges elemek. A felhasználó a billentyűzet mellett különböző mutatóeszközöket, leggyakrabban egeret használhat, vagy épp a saját ujjait érintőképernyők esetében. A grafikus felület jellemző, közismert velejárói az ablakok, ikonok, gombok, gördítősávok és a rövid üzeneteket megjelenítő, időnként felpattanó „buborékok”.

Fontos világosan látni azt, hogy a grafikus felület a komoly hardveres erőforrásigény miatt régebben nem lett volna kifejleszthető, még akkor sem, hogyha valaki akkortájt (mondjuk az 1970-es években) a most elfogadott elveket pontosan meg tudta volna fogalmazni.

A mostani erős hardver képességek megléte mellett viszont már szabadon szárnyalhat a fejlesztők fantáziája: sok-sok új ötlet, lehetőség segíti a szebb, használhatóbb felületek kialakítását. Talán egyedül a felhasználói megszokás a gyakori és gyors változások komoly korlátja: a felhasználókat általában nehéz a megszokottól jelentősen eltérő felületre átszoktatni.

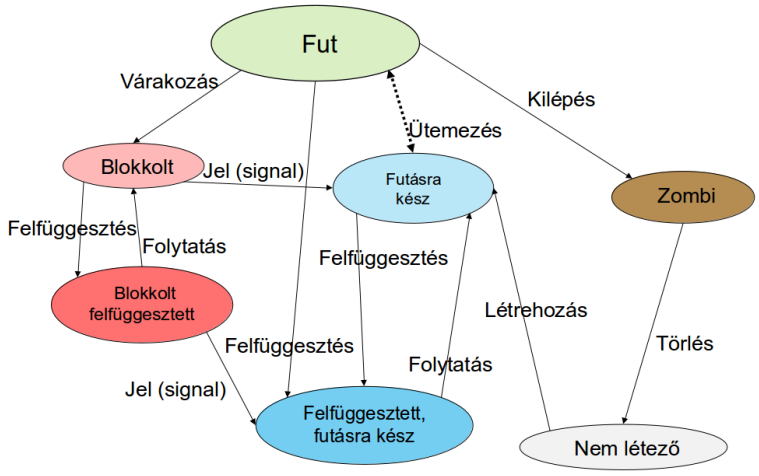
Folyamatok, életciklus és ütemezés

Programjaink háttértárakon helyezkednek el, így statikusnak tekinthetők: sok fájlból állhatnak, van méretük, ennek megfelelően foglalnak el megadott helyet az adathordozókon, egyikről a másikra másolhatjuk őket stb. A folyamatok (processzek) ezeknek a programoknak a futó példányai (jegyzetünkben nem foglalkozunk a szálakkal, a párhuzamosan futó folyamatrészekkel). Egy folyamatnak különböző állapotai vannak: megszületik („épp indul”'), fut, várakozik, befejeződik és megszűnik (lásd lent is). Az állapotátmenetről a folyamat sokszor nem maga dönt, hanem ezt az operációs rendszer szabályozza (ütemezés). A folyamat futás közben használja a processzort (a processzor őt futtatja), és használ egyéb erőforrásokat is, például a memóriát.

Folyamatot csak folyamat indíthat – az operációs rendszer indulásakor elindul egy ún. ősszülő folyamat, ami az összes többit elindítja. Ezekből aztán egy hierarchikus rendszer alakul ki.

Az operációs rendszer az éppen futó folyamatokról nyilvántartást vezet (processz tábla). Ebben a valóban táblázatszerűnek képzelhető adathalmazban található minden, ami a folyamatok futásának adminisztrációjához kell: a futtatandó kód és a hozzá tartozó adatrész elhelyezkedése, regisztertartalmak, menedzsment (pl. jogosultság) információk stb. Ezen adatok egyetlen folyamathoz tartozó összességét folyamat tartalomnak (process context) hívjuk.

Egy folyamat életciklusa vázlatosan a következő lehet. Ha egy új folyamat létrejön, akkor a nemlétező (Non-existent) állapotból indul és bekerül a futásra kész (Ready), tehát a futásra várakozó folyamatok listájába. (Egyszerű, egyprocesszoros/egymagos esetben a „Ready” folyamatok közül mindig csak egy futhat, ezt pedig az ütemező választja ki.)



{á:m1e3a06.png}

6. ábra
Folyamat állapotok és állapotátmenetek

Az éppen futó (Running) folyamatot vagy az ütemező teszi vissza egy idő után a futásra várakozók sorába, vagy „saját akaratából” kerül blokkolt állapotba, mert például egy számára szükséges erőforrásra vár.

A folyamatokat futásukban fel lehet függeszteni (Suspended), ez természetesen blokkolt, futó és futásra váró állapotban is megtörténhet. Blokkolt állapotból egy folyamat úgy kerülhet ki, hogy egy jelzés (Signal) értesíti arról, hogy a blokkolást kiváltó körülmény megszűnt. Ekkor a folyamat

visszakerül a blokkolást megelőző állapotába, kivéve, ha a felfüggesztés éppen futó állapotban történt, ekkor ugyanis csak a „Ready” állapotba van visszatérés.

Egy folyamat befejezésekor annak futása megszűnik (Zombie), de valamely idő szükséges a folyamattartalom törléséhez, majd ezután – ekkor már nem is beszélhetünk folyamatról – újra a nemlétező állapotba kerül.

Tevékenység: Mutassa be saját szavaival a folyamatok életciklusát! (Állapotok, lehetséges átmenetek.)

A futásra képes folyamatok közötti menedzsmentfeladatokat az ütemező látja el. Az ütemezés egy erőforrás-kiosztási feladat (itt a processzor a fő erőforrás), amelynek során az OS a processzorra várakozó („futni akaró”) folyamatok közül kiválaszt egyet, amelyik a következő időszakban futhat.

Az ütemezés főbb céljai a következők:

- Minden folyamat fusson le (triviális cél annak megszervezése, hogy valamilyen módon minden folyamat kapjon processzoridőt, és tudjon futni).
- A válaszidő csökkentése, azaz a felhasználó aktivitása és a rá adott első válasz között eltelt idő minimalizálása (ha például egy felhasználó két ablakban két külön programot futtat, elvárja, hogy úgy érezze, hogy mindkét program „azonnal'” reagál a lenyomott gombokra, még akkor is, ha a gép csak egyprocesszoros).
- A rendszer hatásfokának a növelése, azaz: a processzor minél jobb kihasználása és az átbocsátó képesség növelése (egységnyi idő alatt több befejezett feladat).

Az ütemezés alaptevékenysége a rövid időközönként végrehajtott folyamatváltás (process context switch). Ilyenkor az OS ütemezésért felelős része, az ütemező (scheduler) megállítja az éppen futó folyamatot, elment minden adatot (ami majd a futás folytatásához szükséges; processz tartalom), és a mentett adatok helyére betölti a futásra kiválasztott következő folyamat adatait.

Az ütemező a fenti célokat bonyolult, ún. ütemezési algoritmusok segítségével igyekszik elérni. Ezeknek az algoritmusoknak két alapvető feladata van:

- A folyamatváltás időzítésének megtervezése,
- A következő futtatandó folyamat kiválasztása.

Az ütemező algoritmus különböző optimalizációs stratégiákat valósít meg a célok összehangolására – ezek ugyanis sokszor ellentétesek (ha valaki sokat foglalja a processzort, akkor a többiek „éheznek”). Az egyik ilyen stratégia lehet a prioritások beállítása és kezelése. Eszerint a folyamatok fontossági sorrend szerint kategorizálhatók, és ez a besorolás a futás során dinamikusan változhat (akár a felhasználó beavatkozásra is).

Szintén a folyamatokhoz kapcsolódóan, az operációs rendszer rendkívül fontos feladata a memória kezelése is (virtuális memória, lapok, lapkeretek). Mivel ennek részleteit az előző leckében már bemutattuk, ezért most ismételten nem tárgyaljuk.

Tevékenység: Saját gépének operációs rendszerében (a megoldás előtt javasolt a Súgó áttekintése, böngészése!) tájékozódjon a következőkről:

- Milyen programok és folyamatok futnak most a gépen?
- Milyen egyéb, most éppen nem futó folyamatok találhatók még a folyamatlistában?

- Mekkora az éppen használt memória, és mennyi a még szabad?
- Használ-e a rendszer virtuális memóriát, és ha igen, akkor mennyit?

(Windows alatt használja a megoldáshoz a Feladatkezelőt, az Erőforrás-figyelőt és a systeminfo parancsot.)

Fájlkezelés, fájlrendszerek

Mint már jól tudjuk, adatainkat a háttértáron fájlokba rendezve tároljuk (a fájl vagy állomány a számítógép háttértárolóin lévő összetartozó, azonosítóval ellátott kódok tárolási egysége). Az operációs rendszer a fájlok használatához nagyon kényelmes eszközrendszert biztosít. Ezzel óriási terhet levesz a vállunkról, hiszen nem kell törődnünk az összetartozó adatok egyben tartásával, az adatok bitenkénti letárolásával, a használható, szabad hely kiszámításával és sok más technikai részletkérdéssel.

A fájlrendszer feladata, hogy lehetővé tegye a fájlok tárolását, használatát a felhasználó által elvárt kényelmes, hatékony, biztonságos módon. Természetesen a sok különböző igényre készült számtalan (részben egyedi) megoldás miatt sokféle fájlrendszer létezik, mindegyiknek megfogalmazhatók előnyei és hátrányai (lásd később).

A fájlrendszer kereteit a háttértárakon az első használat előtt ki kell alakítani (formázás). Megjegyezzük, hogy teljesen tetszőleges fájlrendszer ilyenkor sem választható, csak olyan, amit az általunk használt operációs rendszer támogat. Napjainkban egy átlagos felhasználó a formázás műveletével csak nagyon ritkán találkozik, mivel az adathordozók túlnyomó többsége gyárilag formázott.

A különböző adathordozókon tárolt adatok kezelésére, manipulálására egységes logikai szemléletet dolgoztak ki, ez a fájlkezelés. Ez a koncepció általánosabb, mint a konkrét fájlrendszerek, hiszen valamennyi fájlrendszer ugyanezen elvet követi.

Egy átlagos célra használt, otthoni számítógépen százezres vagy milliós (!) nagyságrendű fájl található. Ennyi fájlt csak megfelelően csoportosítva lehet kezelni, ezért a fájlrendszerek teljesen általános, „kötelező” tulajdonsága a mapparendszer (könyvtárrendszer) támogatása. Eszerint a fájlok csoportokba sorolhatók (ezek a mappák vagy könyvtárak), és a mappákban újabb almappák hozhatók létre. A rendszer felépítését a felhasználó a fájlkezelő rendszer segítségével lényegében a saját elképzelése szerint végezheti el (persze bizonyos fizikai és logikai korlátozások betartásával).

Tevékenység: Gondolkozzon el azon, hogyan tudná megszámolni, hány fájl van a számítógépén.

Természetesen az adattárolás konkrét megvalósításából adódóan minden fájlrendszernek vannak korlátai, a mappák és fájlok maximális számára, a mappák maximális egymásba ágyazhatóságára, a maximális névhosszúságra (fájlok és mappák), illetve a teljes fájlrendszer maximális méretére vonatkozóan. Régebbi tervezésű fájlrendszereknél mindezek ma már komoly korlátot jelenthetnek, azonban a modern fájlrendszereket úgy tervezik, hogy a felhasználó ezekbe a korlátokba egyáltalán ne, vagy csak szélsőséges esetben „fusson bele”.

Fájlrendszerünk kiválasztásánál érdemes megfontolni még mindezek mellett a hatékonyság és a biztonság kérdését is. Előbbi csoportba tartozik például a fájlok töredezettségének a problémája, amely akkor jelentkezik, ha korábbi törlések, illetve méretváltoztatások következtében az új fájlok már nem feltétlenül folyamatos tárolással kerülnek a háttértárra. Ez rosszabb esetekben érzékelhető problémákhoz vezethet, mivel a fájlok olvasási és írási sebessége így jelentősen csökkenhet. Ügyes

algoritmusokkal a töredezettség csökkenthető (lásd EXT4 fájlrendszer). A biztonsággal kapcsolatban fontos érv lehet a Unix és a Linux rendszer mellett (a Windows rendszerrel szemben) az, hogy a vírusfertőzés alapja szinte mindig egy futtatható fájl módosítása. A Windows ezt megengedi, a Unix és a Linux nem.

Az operációs rendszer a fájlok és könyvtárak kezelésével kapcsolatban a következő fontos szolgáltatásokat, ill. támogatást nyújtja:

- Fájlok egyedi elnevezése (a fájl azonosítója névből és kiterjesztésből áll),
- Sok fájl strukturált tárolása (mappákba szervezve),
- Fájlok visszakereshetősége, olvashatósága,
- Fájlok létrehozása,
- Fájlok másolása és mozgatása,
- Fájlok módosítása,
- Fájlok törlése, kukába helyezése,
- Fájlok jogosultságainak kezelése (ahol meghatározhatjuk, hogy ki végezhet olvasási, írási, törlési, futtatási műveletet az egyes fájlokon),
- Fájltípusok meghatározása, kezelő programokhoz kapcsolása (például beállíthatjuk, hogy a fájlkezelőben egy zenei fájlra kattintva azt valamely zenelejátszó program elkezdje lejátszani),
- Fájlok fájltípusnak megfelelő megtekintése,
- Fájlok tömörítése,
- Fájlrendszer létrehozása az adathordozókon.

Tevékenység: Indítsa el operációs rendszerének intézőjét (pl. Windows Intéző), továbbá egy Commander programot! Nézze meg, hogy ezek a programok milyen támogatást nyújtanak a fenti listában szereplő aktivitások elvégzéséhez!

A fájlok és mappák kezeléséhez – a fent bemutatottakon túl még – a következő fontos fogalmak kapcsolódnak (amelyeket minden felhasználónak ismernie kell):

- Aktuális mappa (azonosítása a pont karakterrel),
- Szülőmappa (azonosítása: két pont karakter),
- Gyermekmappa (azonosítása: névvel),
- Meghajtó (azonosítása betűjellel, például C:, D: stb.),
- Elérési út (relatív, ha az aktuális mappából indul; abszolút, ha a gyökérből indul),
- Dzsóker karakterek (?, * – egy, illetve több karakter helyettesítésére használhatók),
- Maszkok (dzsóker jelekkel és egyéb karakterekkel felépítve).

A fájlok és a mappák műveleteit egy összetett gyakorló feladatban tekintjük át a lecke elméleti része után.

A következőkben a sok-sok fájlrendszer közül három jellemző típus legfőbb tulajdonságait tekintjük át: egy egyszerű, de a mai apró digitális eszközeink (MP3 lejátszó, fényképezőgép, telefon) által széles körben elterjedt rendszert, majd a Windows és Linux egy-egy képviselőjét.

- FAT, FAT32: Régebbi, de nagyon elterjedt, kompatibilitási célú fájlrendszer. Eredetileg a 80-as években, a DOS operációs rendszerhez készült, a kor színvonalán. Nevét arról a táblázatról kapta, amiben az egyes fájlok elhelyezkedését tartja nyilván (FAT – File Allocation Table). Eredetileg csak 8+3 karakteres fájlnevek használatára volt képes és több más komoly korlátja is volt. Manapság a FAT32-nek nevezett verziót használjuk (32 bites belső címzés). A FAT32-ben a maximális fájlméret 4 GB (és a teljes rendszer mérete 100 GB alatti). A 4 GB régebben óriásinak látszott, de ma már a közönséges médiafájlok is nagyobbak ennél, többek között ezért is szorul ki a rendszer a használatból (a Windows rendszerek kb. 2000-ig főként FAT és FAT32 fájlrendszereket használtak). Manapság már semmilyen operációs rendszer nem részesíti ezeket előnyben (de mind képes használni), azonban a hordozható digitális eszközökön a mai napig elterjedt, ennek oka a széles kompatibilitás. Így ez a fájlrendszer kerül a pendrive-okra, MP3 lejátszókra, a fényképezőgépek és okostelefonok flash háttértáraira. A FAT eszközök külön-külön kezelhetők, a jól ismert meghajtó betűjelek (A:, C:, stb.) segítségével.
- NTFS: A Windows NT-hez készített NTFS fájlrendszert a korábbi FAT rendszerre építve hozták létre, kimondottan ambiciózus célkitűzéssel (hibatűrés, biztonság, adatok helyreállíthatósága stb.). Az NTFS 2 TB maximális méretű lehet, de ez egybefűzhető, ún. dinamikus kötetek segítségével 16 TB-re növelhető. Ez a méret már kielégíti a mai igényeket. A rendszerben nyilvántartható az egyes felhasználók helyfoglalása, és beállítható, hogy bizonyos tárhelyméretet senki ne léphessen túl.
- EXT4: Az EXT4 (EXTended filesystem version 4 – kiterjesztett fájlrendszer, 4. verzió) modern fájlrendszert a Linuxhoz fejlesztették ki a 90-es évek elején. Méretproblémákba az átlagos felhasználó aligha futhat bele: a maximális rendszerméret 1000 PB (petabyte, 1 PB = 1000 TB; azaz kb. félmilliószorosa egy mai tipikus winchester kapacitásának), a fájlméretre vonatkozó korlát pedig 16 TB, ami a legtöbb feladathoz szintén bőven megfelel. Egy mappába 64000 almappa kerülhet. Az EXT4-ben alkalmazott extent (kb. terjedelem) szolgáltatás megbecsüli a fájlok jövőbeli méretváltozásait, és ennek megfelelően foglal le a meghajtón a fájl számára fizikai területet. Ez az alrendszer képes nagyon alacsony szinten tartani a fájlok széttagoltságát, így Linux alatt a töredezettség-mentesítés problémájával gyakorlatilag nem kell foglalkoznia a felhasználónak. Szintén csökkenti a széttagoltságot a késleltetett elhelyezés (Delayed Allocation) összetevő használata. Ez a módszer a fizikai blokk elhelyezését visszatartja: csak akkor keres helyet a letárolandó adatnak, amikor az ténylegesen fizikailag tárolásra kerül (a gyakran írt fájloknál mindez komoly teljesítménynövekedést eredményezhet).

Tevékenység: Esetleges internetes kutakodást (gyűjtőmunkát) is beépítve, hallgatótársával együtt végezze el a következő feladatot. Jelölje meg kedvenc operációs rendszerét és fájlrendszerét! Érveljen a saját rendszere mellett, és a kollégája rendszere ellen (természetesen szakmai érvekkel)!

Fájl- és könyvtárkezelés gyakorlat

Parancssori felület

Az operációs rendszerek egyik legfontosabb szolgáltatása a fájl- és könyvtárkezelés. Ezeket a feladatokat az operációs rendszerek többségében parancssorból is el lehet végezni. A következő táblázatban bemutatjuk a Linux és a Windows operációs rendszerek legfontosabb fájlkezelő parancsait.

Tevékenység: Próbálja ki a parancsokat a saját operációs rendszerében!

Feladat	Windows	Linux
Könyvtár létrehozása	md	mkdir
Könyvtár törlése	rd	rmdir
Könyvtárváltás	cd	cd
Fájlok másolása	copy	cp
Fájlok és könyvtárak másolása	xcopy	cp
Fájlok, könyvtárak listázása	dir	ls

6. táblázat
Fájl- és könyvtárkezelő parancsok

Total Commander

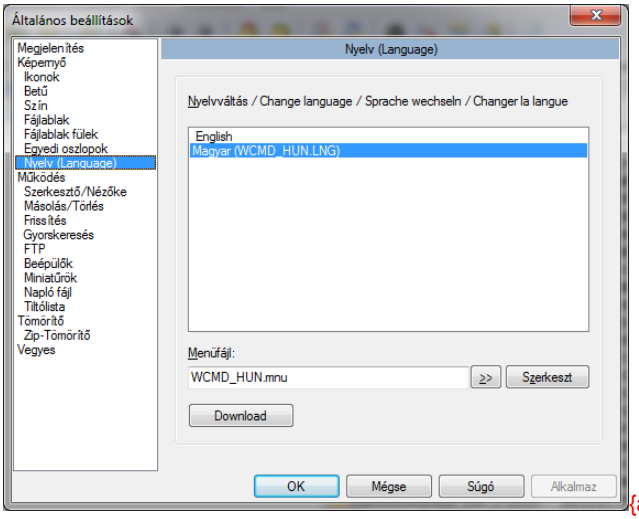
Alapok

Az operációs rendszerek beépített funkciói mellett számos fájlkezelő alkalmazás segítheti a munkánkat. Ahogy fent már említettük, a Commanderek is ilyen, nagyon sikeres segédprogramok. A következőkben egy gyakorlati feladatblokk megoldásán keresztül alaposabban is megismerkedünk a Total Commander rendszerrel.

Bár ezt a tananyagrészt Windows környezetben készítettük el, más ismert operációs rendszerekben is használható mindig megfelelő Commander változat (vagy éppen pontosan ugyanez).

Tevékenység: Nyissa meg a Total Commander alkalmazást és ismerkedjen meg a felületével!

Szükség esetén a Beállítások/Általános beállítások (angolra állított program esetén: Configuration/Options) ablak Nyelv (Language) fülén állítsuk be a kívánt nyelvet.



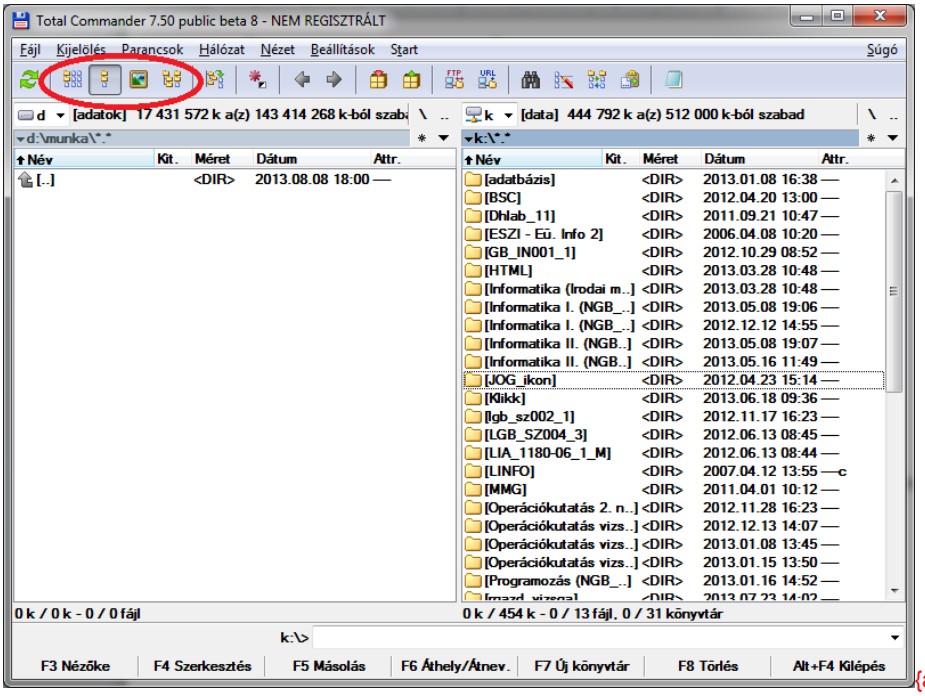
{á:m1e3a07.png}

7. ábra
Nyelvi beállítások

Tevékenység: A megfelelő nyelv kiválasztása után próbálja ki a különböző nézeteket az eszköztár

- Csak a fájl neve,

- Minden fájl adat,
- Miniatűrök nézet és a
- Váltás a könyvtárablakon keresztül nyomógombjai segítségével.



{á:m1e3a08.png}

8. ábra

A Total Commander felülete

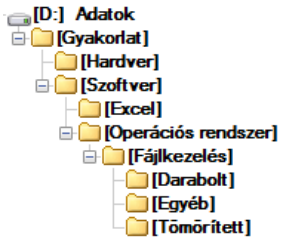
Mappák létrehozása

Feladat: Hozzunk létre egy új mappát Gyakorlat néven a számítógép egyik meghajtójára!

A továbbiakban mi a D:\ meghajtón fogunk dolgozni. Az új mappa létrehozása az F7-es funkcióbillentyűvel vagy az Új könyvtár nyomógombbal történhet.

Tevékenység: Hozza létre az ábra szerinti mappastruktúrát! A munka végén ellenőrizze, hogy valóban pontosan az előírt struktúrát kapta meg!

Megjegyzés: Ha nincs D: meghajtója akkor dolgozzon az Ön által használt gépen bármely írható meghajtó gyökerébe, (pl. C: meghajtó), és a további feladatokat is értelemszerűen ezen oldja meg!



{á:m1e3a09.png}

9. ábra

Mappastruktúra

A Windows rendszerben figyeljünk arra, hogy a fájlnevekben megadhatók ugyan kis- és nagybetűk, de használatukra nincs hivatalos előírás. Előfordulhat, hogy különböző programok különböző formában jelenítik meg az általunk létrehozott neveket (például úgy, hogy a mappaneveket csupa nagybetűvel, a normál fájlokat csupa kicsivel írják), és maga az operációs rendszer sem tesz általában különbséget a kis- és nagybetűk között.

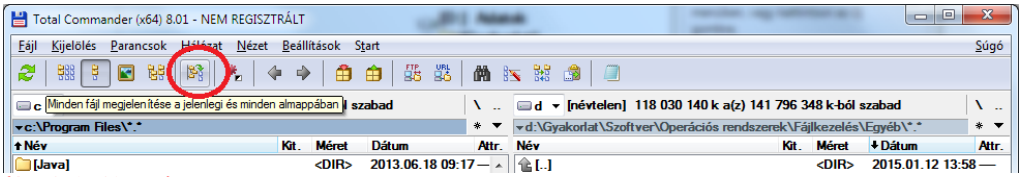
Fájlok másolása

Tevékenység: Oldja meg velünk közösen a következő feladatokat!

Feladat: Másoljuk át a D:\Gyakorlat\Szoftver\Operációs rendszer\Egyéb mappába a C:\Program Files\ mappából (és almappáiból) az összes exe és bat kiterjesztésű fájlt.

Használjuk a Total Commander bal oldali ablakát forrásnak, a jobb oldalit pedig célnak. A forrás oldalon menjünk bele a Program Files könyvtárba, a másikon pedig az Egyéb mappába.

A „Minden fájl megjelenítése a jelenlegi és minden alkönyvtárban” nyomógombbal vagy a Ctrl+B billentyűparanccsal listázzuk ki a Program Files könyvtárban (és alkönyvtáraiban) található fájlokat.



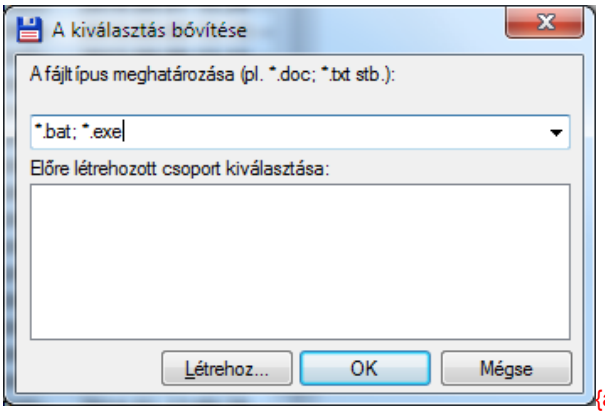
{á:m1e3a10.png}

10. ábra

Minden fájl megjelenítése

A Kijelölés/Coportkijelölés… vagy a numerikus billentyűzet + jele (szürke +) segítségével jelöljük ki a bat és exe kiterjesztésű fájlokat. Ehhez a „*.exe; *.bat” maszk használható.

A *-gal bármennyi bármilyen, a ?-lel pedig pontosan egy tetszőleges karakter helyettesíthető.



{á:m1e3a11.png}

11. ábra
Csoport kijelölése

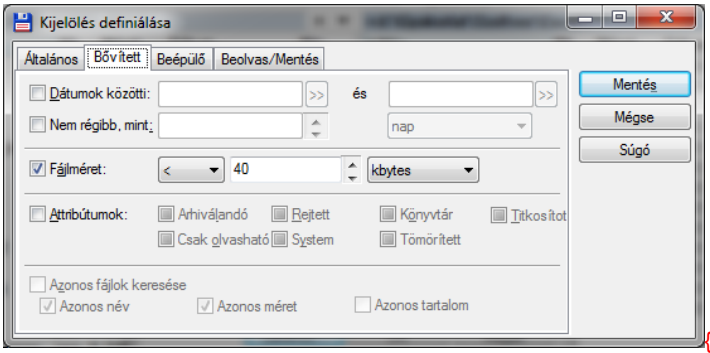
A kijelölés után az F5 funkcióbillentyűvel vagy a Másolás nyomógombbal lehet átmásolni a kijelölt fájlokat a célmappába.

Az egyforma nevű fájlok esetében ne a felülírást, hanem az átnevezést válasszuk és lássuk el a fájlokat megfelelő sorszámmal, pl. program2.exe.

Törlés

Feladat: Töröljük ki az Egyéb mappából a 40 KB-nál kisebb fájlok közül a legrégebbit.

A Csoportkijelölés ablak Létrehoz… nyomógombjának segítségével hívjuk elő a Kijelölés definiálása ablakot. Ennek a Bővített fülén állítsuk be a méretre vonatkozó feltételt, majd mentsük el az új kijelölést „40” néven.



{á:m1e3a12.png}

12. ábra
Bővített kijelölés

Ezután rendezzük a fájlokat dátum szerint növekvő sorrendbe és válasszuk ki a listából legelső kijelölt fájlt.

{á:m1e3a13.png}

13. ábra
Rendezés

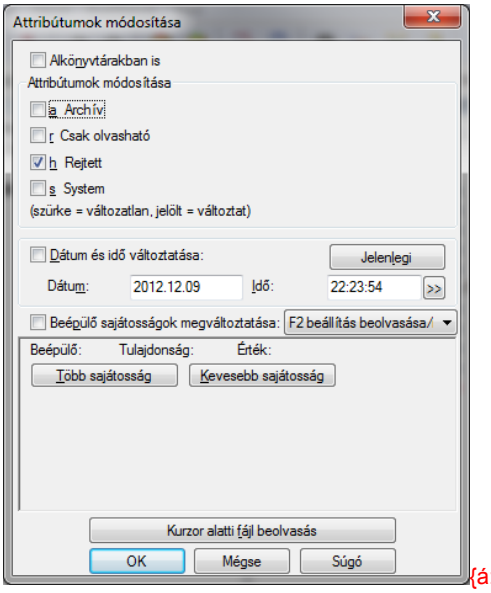
Ha most kattintanánk a törlésre, akkor a rendszer nem csak azt a fájlt törölné ki, amin állunk, hanem az összes kijelöltet. Ezért a numerikus billentyűzet – jelével (szürke –) vagy a Kijelölés/Csoportkijelölés megszüntetése… paranccsal szüntessük meg a kijelölést. Végezetül az F8-as funkcióbillentyűvel vagy a Delete nyomógombbal vagy a Törlés parancsra kattintva távolítsuk el a kiválasztott fájlt.

Átnevezés, áthelyezés és attribútumok

Feladat: Az Egyéb mappa legnagyobb méretű fájlját nevezzük át „Rejtett”-nek úgy, hogy a kiterjesztését meghagyjuk, majd tegyük rejtetté.

Válasszuk ki az ablakban a kritériumnak megfelelő állományt, majd kattintsunk rá és nevezzük át. Rejtetté tenni végül a Fájl/Attribútumok módosítása… menüpont alatt tudjuk.

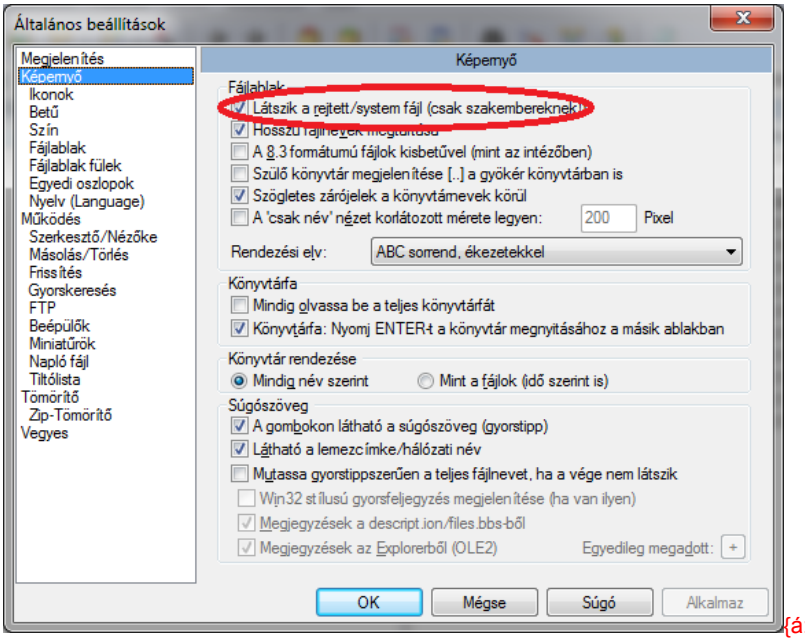
Tevékenység: Figyelje meg, hogy milyen attribútumokat és egyéb tulajdonságokat lehet még átállítani ebben az ablakban!



{á:m1e3a14.png}

14. ábra
Attribútumok módosítása

A rejtett fájlokat úgy tehetjük láthatóvá, hogy a Beállítások/Általános beállítások ablak Képernyő fülén bepipáljuk a megfelelő kapcsolót.



{á:m1e3a15.png}

15. ábra

Beállítások, rejtett fájl láthatósága

A bekapcsolás után úgy ismerjük fel a rejtetté tett fájlt, hogy annak megváltozott az ikonja.

{á:m1e3a16.png}

16. ábra

Rejtett fájl

Feladat: A rejtetté tett fájlt helyezzük át a Hardver mappába.

Ezt kijelölés után az F6-os funkcióbillentyűvel vagy az Áthely/Átnev. nyomógombbal tudjuk elvégezni.

Keresés

Feladat: Keressünk a C:\ meghajtón olyan png kiterjesztésű fájlokat, amelyek nevének második karaktere „a”.

Az Alt+F7 billentyűparanccsal vagy a Keresés nyomógombra {á:m1e3s01.png} kattintva nyissuk meg a keresés ablakot. Ez ugyanúgy épül fel, mint a Kijelölés definiálása ablak.

Tevékenység: Készítse el a kereső kifejezést!

Tevékenység: Hozzon létre Képek azonosítóval egy új mappát a Szoftver mappába!

Az Ablakba gombbal jelenítsük meg a fájlokat egy ablakban, majd másoljuk át a Képek mappába.

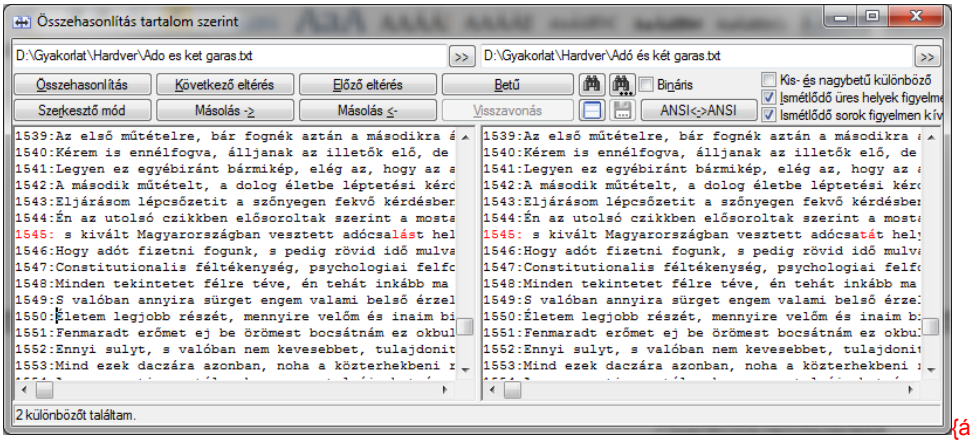
Fájlok összehasonlítása

Ebben a feladatban két nagyon hasonló szövegfájl apró eltéréseit kell megtalálnunk.

Tevékenység: Töltse le a Hardver mappába az Ado es ket garas.txt és az Adó és két garas.txt azonosítójú fájlokat!
Fájlokra hivatkozni!

Jelöljük ki a fájlokat, majd vizsgáljuk meg a Fájl/Összehasonlítás tartalomra… parancs segítségével, hogy a két fájl szövege megegyezik-e.

Az összehasonlító pirossal jelöli azokat a sorokat, ahol eltérést talált, és szintén pirossal jelöli az eltérő karaktereket is.



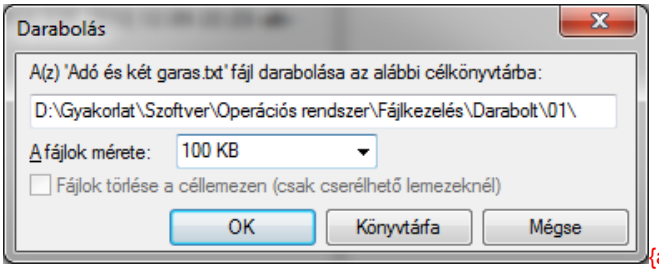
{á:m1e3a17.png}
17. ábra
Fájlok összehasonlítása

Fájldarabolás

Feladat: Daraboljuk fel 100 KB-os darabokra az Adó és két garas.txt fájlt.
Fájlra hivatkozni!

Tevékenység: Hozzon létre egy 01 és egy 02 nevű mappát a Darabolt könyvtárba!

A létrehozás után az egyik ablakban jelenítsük meg a 01, a másikban pedig a Hardver mappát. Jelöljük ki az Adó és két garas.txt-t, majd válasszuk ki a Fájl/Fájldarabolás… parancsot. A megjelenő párbeszédablakba gépeljük be a 100 KB-ot, majd daraboljuk fel a fájlt az OK gomb lenyomásával.



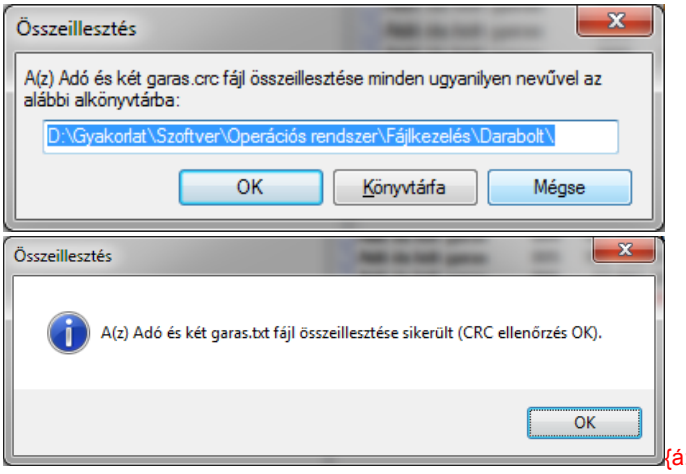
{á:m1e3a18.png}
18. ábra
Fájldarabolás

A darabolás után több sorszámozott fájldarab és egy CRC (Cyclic Redundancy Check; ellenőrző összeg, kontrollszumma) kiterjesztésű önellenőrző fájl jön létre. A CRC állomány tartalmazza az eredeti fájl nevét, méretét és egy ellenőrző kódot. A tartalmat az F3-as funkcióbillentyűvel vagy a Nézőke nyomógombbal meg is tudjuk nézni.

Tevékenység: Nézze meg az egyik sorszámozott fájl tartalmát!

Tevékenység: Másolja át a 01 mappa tartalmát a 02 mappába!

Álljunk a Total Commander egyik ablakában a Darabolt, a másikban pedig a 01 mappába és egyesítsük a fájldarabokat. Az egyesítést a CRC fájlra való dupla kattintással vagy a CRC kijelölése után a Fájl/Fájlegyesítés paranccsal lehet elvégezni.



{á:m1e3a19.png}

19. ábra
Fájlegyesítés

Tevékenység: Hasonlítsa össze az eredeti és a most egyesített fájl tartalmát!

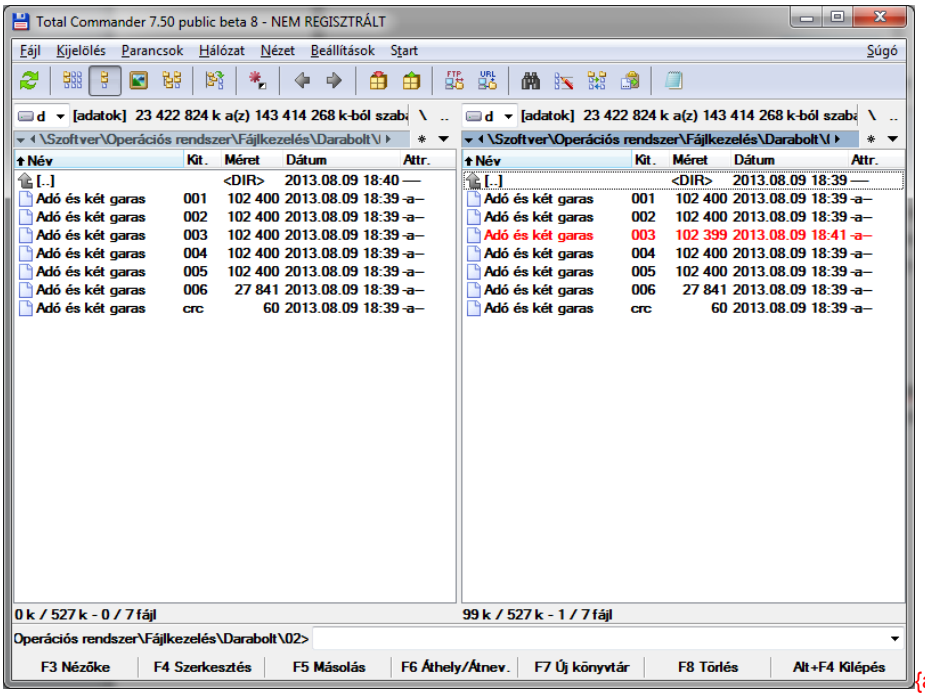
Feladat: Próbáljuk ki, hogy valamelyik fájldarab apró módosítása után már az összeillesztést nem lehet elvégezni!

A 02 azonosítójú mappában található fájldarabok közül nyissuk meg a 3-as sorszámút és töröljünk ki belőle egy karaktert. A fájlt szerkesztésre megnyitni az F4-es funkcióbillentyűvel vagy a Szerkesztés nyomógombbal lehet. A törlés után próbáljuk meg újra egyesíteni a fájlt és figyeljük meg, hogy milyen hibaüzenetet kapunk.

Mappák összehasonlítása

Feladat: Hasonlítsuk össze a 01 és 02 mappát, jelöljük meg az eltéréseket.

A mappákat nyissuk meg a Total Commander két oldalán. Ezután a Kijelölés/Mappák összehasonlítása vagy a Shift+F2 billentyűparancs hatására a rendszer piros színnel kiemelve megjeleníti az eltéréseket (most csak egy eltérő fájl van). Az összehasonlítás csak a fájlokra vonatkozik, ha lenne mappa is a könyvtárakban, akkor azt a rendszer a vizsgálat során nem venné figyelembe.



{á:m1e3a20.png}

20. ábra

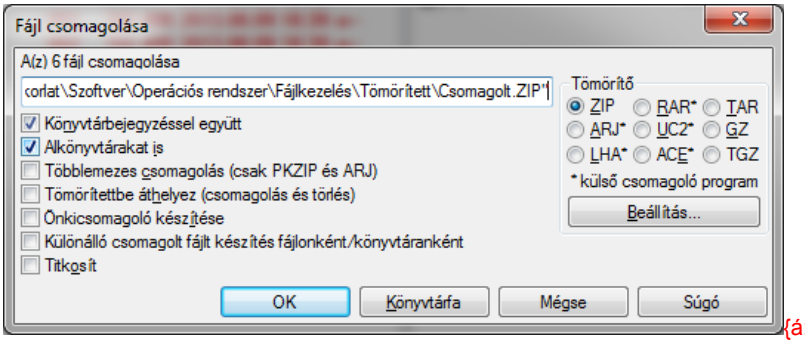
Könyvtárak összehasonlítása

Tömörítés

Feladat: Csomagoljuk be a 01 könyvtárban lévő fájldarabokat Csomagolt.zip néven a Tömörített mappába.

A Total Commader egyik ablakában jelenítsük meg a 01, a másikban pedig a Tömörített mappát. Jelöljük ki a CRC fájlt, majd invertáljuk a kijelölést a Kijelölés/Kijelölés megfordítása vagy a * gomb segítségével (így az összes fájldarab lesz kijelölt). Végezetül a Fájl/Becsomagolás… vagy az Alt+F5 billentyűparanccsal vagy a Tömörítés nyomógombbal {á:m1e3s02.png} tömörítsük be a fájlokat. A megjelenő párbeszédablakban ne felejtsük el átírni a keletkező fájl nevét.

> FONTOS! A fájl átnevezésekor ne módosítsuk az elérési utat, a kiterjesztést és a tömörítő algoritmust!



{á:m1e3a21.png}

21. ábra
Tömörítés

{á:m1e3a22.png}

22. ábra
Tömörített fájl

A Fájl csomagolása ablakban a Titkosít lehetőséget bepipálva jelszóval védett zip állományt is lehet készíteni.

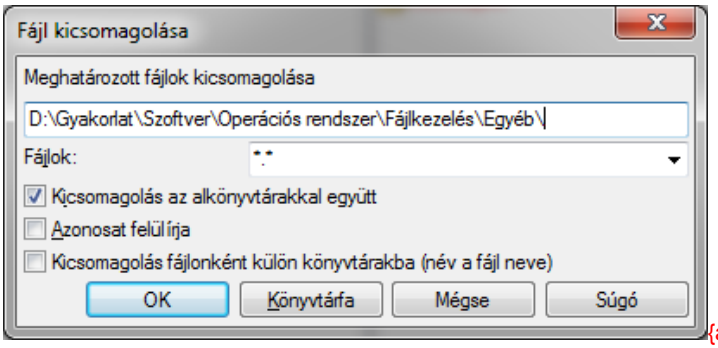
Tevékenység: Hasonlítsa össze az eredeti fájlok méretét a csomagolt fájl méretével!

Érdekesség!

Tevékenység: A veszteségmentes tömörítés nem mindig jár méretcsökkenéssel. Készítsen egy olyan szöveges fájlt, ami csak a „SZE” karaktereket tartalmazza, majd tömörítse be! Hasonlítsa össze az eredeti fájl méretét a csomagolt fájl méretével!

Feladat: Csomagoljuk ki az előbb létrehozott Csomagolt.zip azonosítójú fájlt az Egyéb mappába.

A már megszokott módon állítsuk be a Commandert úgy, hogy az egyik oldalon legyen az Egyéb mappa, a másikon pedig a zip fájl mappája. Jelöljük ki a tömörített fájlt, majd a Fájl/Kicsomagolás… vagy az Alt+F9 billentyűparanccsal vagy a Kicsomagolás nyomógombbal {á:m1e3s03.png} csomagoljuk ki a fájlokat.



{á:m1e3a23.png}

23. ábra
Kicsomagolás

Tevékenység: Hasonlítsa össze a 01 és az Egyéb mappa tartalmát!

Tevékenység: Tömörítse be a Gyakorlat mappát jelszóval ellátott (jelszó: infó) zip állományként Vezetéknév_Keresztnév_NEPTUNKÓD.zip néven!

Figyeljünk arra, hogy hiába állítunk be bármilyen fájlkiterjesztést, attól technikailag független, hogy milyen tömörítő algoritmust használunk. Tehát, ha pl. zip tömörítést szeretnénk használni, a Tömörítő algoritmusnál mindenképp a ZIP legyen kiválasztva (és logikus, hogy a fájl kiterjesztése is utaljon erre, azaz ZIP legyen).

teszt rész

Önellenőrző kérdések

1. Mi jellemző a parancssori interfészre?

☑ A monitoron fix méretű karakterek, billentyűzet használata.

☐ Monitoron grafikus + szöveges elemek, felhasználói mutatóeszközök.

☐ A grafikus felület elemeinek kezelése – ablak, gomb, görgetősáv.

2. Tegye sorba az operációs rendszer indításának egyes lépéseit!

1 Hardver önteszt.

2 BIOS betölti a kernelt.

3 Kernel: eszközök inicializálása.

4 Szolgáltatások indítása.

5 Felhasználói bejelentkezés.

6 Automatikusan induló programok indítása.

3. Jelölje be az alábbiak közül az operációs rendszer funkcióit!

☑ fájl- és könyvtárkezelés

☑ folyamatkezelés

☐ keresés az interneten

☑ I/O kezelés

☐ fényképek mappákba rendezése

☐ algoritmusok hatékonyságának a mérése

4. Melyek azok a BIOS-funkciók, paraméterek, amelyeket a mai operációs rendszerek önállóan is el tudnak végezni, be tudnak állítani?

- ☑ Pontos idő beállítása.
- ☑ Energiagazdálkodási funkciók.
- ☑ Processzor teljesítmény.
- ☑ Különböző eszközök letiltása.
- ☑ Különböző interfészek letiltása.
- ☐ Vírusos e-mailek blokkolása.

5. Készítsen olyan keresőkifejezést, aminek a segítségével azokat a fájlokat lehet megkeresni, amiknek a második karaktere „x” és „xlsx” a kiterjesztése!

Kifejezés: x*.xlsx

Megjegyzés [M2]: javítani: ?x*.xlsx

6. Készítsen olyan keresőkifejezést, aminek a segítségével azokat a fájlokat lehet megkeresni, amiknek a neve három karakter és pdf a kiterjesztése! (Pl.: abc.pdf)

Kifejezés: ???.pdf

Modulzáró feladatsor

1. Tegye értékük szerint nemcsökkenő sorrendbe az alábbi számokat! A zárójelben a számrendszer alapja található.

1 100 (2)

2 110 (2)

3 09 (16)

4 012 (8)

5 11 (10)

1 pont

2. Alakítsa át a következő decimális számot előjel nélküli 16 bites bináris alakba: 551!

0000001000100111

1 pont

3. Alakítsa át a következő decimális számot előjeles 16 bites bináris alakba: -118!

1111111110001010

1 pont

4. Alakítsa át a következő előjelesen ábrázolt 16 bites bináris számot decimális alakba: 0000001000010111!

535

1 pont

5. Alakítsa át a következő előjelesen ábrázolt 16 bites bináris számot decimális alakba: 1111111110001011!

-117

1 pont

6. Alakítsa át a következő 8 bites bináris számot hexadecimális alakba: 01011001!

59

1 pont

7. Válassza ki a következő bináris számok közül a negatívokat!

☑ 111010111

☐ 011100010

☑ 111011010

☐ 011110111

☑ 100010010

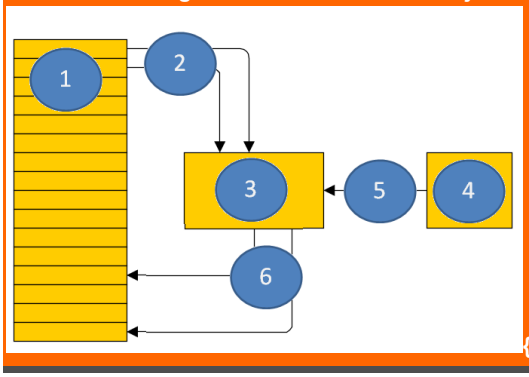
1 pont

8. Adja meg a logikai kifejezések értékét!

A	B	C	**Kifejezés**	**Eredmény**
IGAZ	IGAZ	IGAZ	$(A \vee \neg B) \neq C$	HAMIS
IGAZ	HAMIS	HAMIS	$(A \vee B) \neq \neg(A \wedge C)$	HAMIS
HAMIS	IGAZ	HAMIS	$\neg(A \neq (B \neq C)) \vee \neg(A \neq C)$	IGAZ

1 pont

9. Párosítsa a fogalmakat az ábra számmal jelölt részeihez!

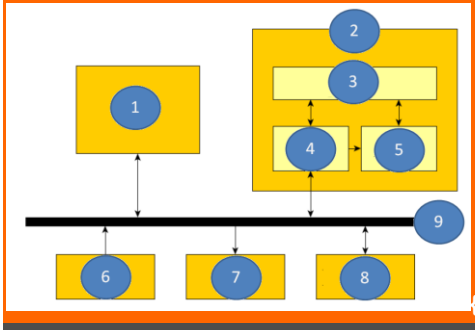


{á:m1f09.png}

1 Regiszterek
2 Operandusok
3 ALU
4 CU
5 Művelet
6 Státusz és eredmény

1 pont

10. Párosítsa a fogalmakat az ábra számmal jelölt részeihez!



{á:m1f10.png}

1 Memória
2 CPU
3 Regiszterek
4 CU
5 ALU
6 Input
7 Output
8 Háttértár
9 Buszrendszer

1 pont

11. Mennyi lesz az alábbi kifejezés eredménye, ha számítógépünk csak egybájtos előjeles adatípust használ (Hymn CPU szimulátor)?

Kifejezés: $-93+22+45$
Eredmény: -26

1 pont

12. Válassza ki a következő listából a Neumann-elveket!

☑ A gép digitális legyen, azaz diszkrét jeleket tudjon kezelni (ezek a jelek bináris számjegyek legyenek).

☑ A gép teljesen elektronikus legyen, azaz fő részegységei ne tartalmazzanak mozgó (mechanikus) alkatrészeket.

☐ A gép Floppy vagy CD meghajtóval rendelkezzen.

☐ A gép háttértárán legalább egy operációs rendszer legyen.

1 pont

13. Zip tömörítőt alkalmazva csomagolja be a mellékelt alma.txt fájlt!
Fájlra hivatkozni!

Mekkora méretű a keletkezett állomány?

120 B

1 pont

14. Csomagolja ki a mellékelt Étlap.zip fájlt, majd oldja meg az alábbi feladatokat.
Fájlra hivatkozni!

Egyes ételek leírása véletlenül rossz helyre került. Melyik mappában található a palacsinta.txt azonosítójú fájl?

Hal

1 pont

Az Előétel mappában egy darabolt fájl található. Mi a szétdarabolt fájl eredeti azonosítója?

sajt.pdf

1 pont

Hány darab txt kiterjesztésű fájl van a Desszert\Palacsinta\Kakaós mappában?

62

1 pont

Hasonlítsa össze a Desszert\Beigli\Diós és a Desszert\Beigli\Mákos mappák tartalmát! Melyik fájl tér el a két mappában?

341.bat

1 pont

Nevezze át a Saláta mappában található sali.bat fájlt sali.png-re, majd nyissa meg! Milyen szöveg látható a képen?

UBORKA

1 pont

A Pizza mappában található recept1.txt fájl a recept.txt fájl másolata, de egy szót töröltek belőle. Melyik ez a szó?

erdei

1 pont