Általános
INFORMATIKAI
Tanszék

Számítógépek, számítógép rendszerek

Jegyzet

Dr. Vadász Dénes

Miskolc, 2005. február

TARTALOM

TARTALOM.. a

A tárgy célja 1

1. Számítógép történet... 3

1.1. Számológépek ókortól az ötvenes évekig 3

1.2. A Neumann elv 5

1.3. Az architektúra fogalma 6

1.3.1. Egy számítógép hardver architektúrája 6

1.3.2. A szoftver architektúra 8

1.3.3. Rétegezettség (Layered architecture) 8

1.3.4. A virtualitás fogalma 8

1.3.5. A transzparencia fogalma 9

2. Számítógép használati módok 10

2.1. Mit lát egy felhasználó, ha leül egy számítógép elé? 11

2.1.1. A folyamat (processz) 12

2.1.2. A felhasználói kezelői felület 13

2.1.3. Az eszköz és fájlrendszer 14

2.1.4. A felhasználók 17

2.2. Számítógépes hálózatok 18

2.2.1. A hálózatosodás motivációi 18

2.2.2. A hálózatok összetevői 19

2.2.3. Számítógéprendszer igénybevétele 19

3. A UNIX operációs rendszer használata 24

3.1. A UNIX filozófia 24

3.2. Honnan tanulhatjuk a UNIX használatot? 26

3.3. Fontos parancsok, csoportosítva 27

3.3.1. Manipulációk fájlokon, jegyzékeken 27

3.3.2. Állapotok (státusok), információk lekérdezése, beállítása 28

3.3.3. Processz indítás, vezérlés 29

3.3.4. Kommunikáció a világgal, felhasználókkal 29

3.3.5. Hasznos szűrők 29

3.3.6. Parancsok a tanuláshoz 29

3.4. A Bourne shell (sh) 30

3.5. Az sh burok, mint parancsértelmező 30

3.5.1 Alapfogalmak 30

3.5.2. Parancs, cső, lista csoportosítás, zárójelezés 33

3.5.3. A parancs végrehajtás 34

3.5.4. Az adatfolyamok átirányítása 35

3.5.5. Fájlnév kifejtés 36

3.5.6. A metakarakterek semlegesítése, az ún. quótázás 38

3.6. Burokprogramozás 39

3.6.1. A shell változók 39

3.6.2. A shell változók osztályai 39

3.6.3. Hivatkozások shell változókra, kifejtésük 41

3.6.4. Parancs behelyettesítés 42

3.6.5. Változók érvényessége (scope-ja) 43

3.7. Vezérlési szerkezetek a sh shellben 44

3.7.1. Szekvenciális szerkezet 44

3.7.2. Elágazás: az if 44

3.7.3. Elágazás: a case 44

3.7.4. Ciklus: a for 45

3.7.5. Ciklus: a while 46

3.7.6. Az if-hez, while-hoz jó dolog a test 46

3.7.7. További jó dolog: az expr parancs 47

3.7.8. A rekurzió lehetősége 48

3.7.9. A read parancs 49

3.7.10. Fontos tanácsok 50

A startup file-ok összefoglalása: 51

3.8. Az awk mintakereső és feldolgozó 52

4. Hálózatok, az Internet 59

4.1 Az Internet története 59

4.1.1. Az ARPANet 59

4.1.2. Az RFC-k (Request for Comments) 59

4.1.3. A korai protokollok 60

4.1.4. A történet folytatódik 61

4.1.5. További protokollok 61

4.2. Az Internet Magyarországon 62

4.3. Csomópontok azonosítása az Internet-en 64

4.4. Az elektronikus levelezés alapfogalmai 67

4.5. Az Internet gopher 70

4.6. A File Transfer Protocol FTP 71

4.7. Az Archie szolgáltatás 71

4.8. Hogyan kereshetünk személyeket, számítógépeket? 72

4.9. A World Wide Web (WWW) és nézegetőik 72

A WWW programozási nyelve: a Java 74

5. Hardver architektúrák, a központi egység működése.. 75

5.1. Az ALU (Aritmetikai logikai egység) 75

5.2. A regiszterek, regiszterkészlet 76

5.3. A vezérlő és dekódoló egység 77

5.4. A címképző és buszcsatoló egység 77

5.5. A CPU belső sínje, sínjei 77

5.6. Az utasításkészlet 77

5.7. Címzési módok 78

5.8. Instrukciókészletek, instrukciók csoportjai 79

5.9. Processzorok működési módjai 81

A. függelék: A verem (stack) adattípus formális specifikációja 81

6. Processzor teljesítmény növelés.. 83

6.1. A processzorok ciklusideje 83

6.2. Processzor teljesítmények (Processor performance) 83

6.3. Teljesítmény értékelési módok 83

6.3.1. MIPS: Million Instruction per Second 83

6.3.2. Korszerűbb sebesség-meghatározások 84

6.3.3. Whetstone Benchmark 84

6.3.4. Livermore Loops Benchmark 84

6.3.5. Dhrystone Benchmark 85

6.3.6. Linpack 100 *100 és 1000 * 1000 Benchmark 85

6.3.7. TPC Benchmark A 85

6.3.8. Dongarra teszteredmények 85

6.3.9. SPEC Benchmark Suites (SPEC készletek) 86

6.3.10. Többprocesszoros gépek összevetése 89

6.3.11. A SPEC teljesítménymérés továbbfejlesztése 89

6.4. A CPU teljesítmény növelés módszerei 90

6.5. A CISC és a RISC architektúrák. 90

6.5.1. CISC: Complex Instruction Set Computer 90

6.5.2. RISC: Reduced Instruction Set Computer 91

6.6. Párhuzamos architektúrák, átlapolás 93

6.6.1. Párhuzamosítás a CPU-n belül: futószalag (csővezeték, pipe-line) feldolgozás 93

6.6.2. Szuperskalaritás 95

6.6.3. Párhuzamosítás a processzoron kívül (Multiprocesszoros rendszerek) 96

7. A sínek.. 98

7.1. Az IBM PC-k sínrendszerei 100

7.2. A PCI sín 101

7.3. További híres sínek 102

8. A memória.. 104

8.1. A félvezető tárolók 105

8.2. Az alapvető DRAM operációk 106

8.3. A Page Mode, Fast Page Mode, Hyper Page Mode elérések 106

8.4. Az SDRAM-ok (Synhronous DRAM Operation) 107

8.5. A memória modulokról 108

8.6. Hogyan csökkenthető a hozzáférési idő? 108

8.7. A gyorsító tárak (cache) 109

8.8. Az asszociatív tárak (CAM Content Addressable Memory) 110

9.Perifériák, eszközök 111

9.1. A vezérlők (controllerek, adapterek) 111

9.1.1. A vezérlők feladatai (általánosan) 111

9.1.2. Egyszerűsített forgatókönyv (scenario) diszkes adatátvitelre 112
9.2. Megszakítás (interrupt, IT) 114
9.3. Az eszközök osztályai 115
9.4. Mágneslemezes tárolók 116
9.5. CD (Compact Disk) lemezek 120
9.6. Terminálok 121

A tárgy célja

A *Számítógépek, számítógép rendszerek* c. tantárgy az informatika tantárgycsoporthoz tartozik. Szokásosan mind a tavaszi, mind az őszi félévben meghirdetjük heti 2 óra előadás és 2 óra gyakorlat időtartamban. A tantárgy tananyaga a Számítástechnika alaptárgyi szigorlat anyaga is egyben. Lezárása a félév végén vizsgával történik. Felvételéhez ajánlott a Programozás alapjai c. tantárgy tananyagának ismerete.

A tárgy céljai

A tárgy célja, hogy a hallgatók megismerjék a számítógépek, számítógép rendszerek használatát. Feltételezem, hogy a tárgyat hallgatók már eddig is használtak számítógépeket, ismernek különböző parancsnyelveket, grafikus kezelőket is használtak már. Nos, a tárgyban a számítógép használatot - ezen belül is a kezelő szoftverek használatát fogjuk rendszerezni, gyakorolni. Nem a működtető szoftverek mélységeinek megismerése a cél most (azt tanulhatják az Operációs rendszerek c. tantárgyban), hanem a géphasználat: itt is van tanulni, rendszerezni való!

Egy másik cél a számítógépek felépítésének, a részegységeknek, a működésüknek megismerése, ha úgy tetszik, architektúra, hardver ismeret. Persze, nem villamosmérnöki szintű ismeretről van szó. Erre nincs is előképzettségük (még nem tanultak elég elektronikát) és nincs is szükség olyan mély ismeretekre a tanulmányaik folytatására. Azt azonban megtanulhatják, hogy funkcionális szinten hogyan épül fel egy számítógép, milyen komponensei lehetnek, azok hogy működnek stb.

A módszerek

Az előadások egy rövid történelmi bevezetés után eleinte a felhasználói felületekkel, azaz a számítógép használattal foglakoznak. A gyakorlatokon különféle rendszerek interaktív használatát, parancsnyelvek megismerését, a parancsnyelvi programozást kapják feladatul. Főleg Unix-os rendszerek kapcsolattartóit (burkait) kell megismerniük, de valamennyi időt töltünk a hálózati szolgáltatások használatával is. Az előadások később a számítógép architektúrákról, a részegységekről, ezek működési elveiről szólnak, a gyakorlatokon ezekhez bemutató és egyéni feladat kapcsolódik majd. Végül grafikus felhasználói felületekkel is foglakozunk egy kicsit. Ennek a sorrendnek egyszerű oka van: minél előbb ki akarjuk adni a burokprogramozási feladatokat.

A tárgy e félévi ütemtervét hirdetőtáblán (és a számítógépes hálózaton) közzétettem.

Az ütemtervben is olvasható, hogy több feladatot kapnak a félév során. A feladatokat a gyakorlatvezetők adják ki, de a hallgatók felelőssége is, hogy a feladatokat időben megkapják, átvegyék: a gyakorlatvezetők nem fognak a hallgatók után szaladgálni, hogy kiadják azokat! A feladatokat viszonylag rövid határidőkkel kell elkészíteni.

Az aláírás feltételei: gyakorlatokra járni kell, ott dolgozni. A feladatokat elkészíteni, azokat a gyakorlatvezetőnek beadni, aki ellenőrzi azokat. Végül – az informatikus hallgatóknak – néhány kisebb és egy nagyobb évközi zárthelyi dolgozatot eredményesen meg kell írniuk.

A vizsga rövid írásbeli és szóbeli vizsga lesz. A félév végén a vizsgakérdések listáját megkapják. A vizsgán a gyakorlatvezetők véleményét figyelembe szoktuk venni. A vizsgán kap-

hatnak kérdéseket az évközi feladataikkal kapcsolatosan is, esetleg be kell mutatniuk, hogy képesek számítógéprendszerekre bejelentkezni, ott dolgozni.

A tantárgyhoz javasolt irodalom listája az ütemtervben megtekinthető.

Miskolc, 2005. február

1. Számítógép történet

A számítógépek történetét két szakaszban tárgyalhatjuk. Az első szakasz az ókortól a századunk közepéig tartott, a második szakasz a század közepétől napjainkig. Javaslom, pillantsanak be a számítástechnika virtuális múzeumába! Az alábbi fejezetben említett személyekről, és persze sok más híres személyről találhatnak információkat a *Pioneers of Computing* kiindulóponttól!

1.1. Számológépek ókortól az ötvenes évekig

Az ember nem szeret számolni. Régesrégi kívánsága volt az emberiségnek, hogy álljanak rendelkezésére számológépek, melyekkel tévedés nélkül, gyorsan tudnak számolni. Nos, voltak is gépek a történelemben.

Többezer éves a (kínai eredetű) *abakusz*. Ez a digitális eszköz primitív volta ellenére nagyon hasznos lehetett. Egyes helyeken még ma is használják: az ún. „golyós" számológép hasonlít az abakuszra. Ma azonban már többnyire csak játék.

Blaise *Pascal* (1623-1662) [lásd a *Pioneers of Computing-ban*] francia matematikus, fizikus, filozófus és feltaláló 1642-ben 6 digiten számoló *összeadó-kivonó gép*et készített (ez volt a *Pascaline*). Az utókor - elismerésképpen - modern programnyelvet nevezett el róla.

Gottfried Wilhelm *Leibniz* (1646-1716) szorozni és osztani is tudó mechanikus gépe (1694-ben készült), úgy tetszik, ez a tegnapi mechanikus számológépek őse.

Charles *Babbage* (1792-1871) [*Pioneers of Computing*] angol matematikus és feltaláló az általános célú számítógépek „atyja".

Difference Engine nevű mechanikus gépe az ún. differencia módszer segítségével számolta volna ki polinomok értékeit diszkrét lépésekben változtatott független változó érték mellett. A terv jó volt, a gépet el is készítették, csak éppen nem működött.

Babbage a Difference Engine kudarca után belefogott az *Analytical Engine* elkészítésébe, ami az általános célú számítógépek előfutárának tekinthető. Tízes számrendszert használó mechanikus gép lett volna: a tervek itt is jók voltak, a megvalósítás az akkori technológiai körülmények között még reménytelenebb volt. A tervezett gép fő részei:

- a malom (CPU),
- a tár (memória),
- nyomtató,
- lyukkártyás bemeneti egység.

Azt lehet mondani, Babbage megelőzte korát! Állítólag utóbb elkészítették Babbage tervei alapján a gépet és az működött.

Ada Byron, Lady Lovelace (1815-1852) [lásd *Pioneers of Computing*], aki Babbage „múzsájának" tekinthető, s aki Lord Byron, a híres költő - szép és okos - leánya volt, felismerte Babbage jelentőségét. *Megfigyelések Babbage Analytical Engine*-jéről címmel írt munkájában ismertette a gép működését, jelentőségét, és *programokat* is közölt a nem létező gépre! Ő volt tehát a történelem első programozója, hálából róla nevezték el az *ADA* nyelvet!

Ugorjunk a XIX. század végére, a XX. század elejére: ebben az időszakban a mechanikus számológépek rohamosan fejlődtek. A legjelentősebb neveket említsük meg:

Roman *Verea* spanyol feltaláló szorzótáblát használó gépet készített.

Otto *Steiger* ugyanilyen elvű gépe a *millionare*, több tízezer példányban készült és kelt el.

Hermann *Hollerith* (1860-1929) [Pioneers of Computing] neve kiemelendő. Az 1880-as amerikai népszámlálás adatainak feldolgozása 1887-re fejeződött be, és folyamatosan növekvő bevándorlás miatt az 1890-es népszámlálás feldolgozása reménytelennek tűnt hagyományos módszerekkel. Hollerith *elektromos lyukkártya feldolgozó gépe* segítségével 6 hét alatt sikerült a feldolgozás! Hollerith 1896-ban céget alapított, ami 1924-től IBM-ként vált ismertté.

Claude Elwood *Shannon* már században dolgozta ki a kommunikáció- és információelmélet alapjait. Bemutatta, hogy bináris elektromos relékből összeadásra, kivonásra, szorzásra és osztásra alkalmas áramköröket lehet építeni, és hogy ezek tervezéséhez a matematikai logika formális leírása jó eszköz.

Konrád *Zuse* (1910-1995) [Pioneers of Computing] 1938-ban készítette *Z1* nevű gépét meccano fémépítő játék elemekből, mechanikus elemekből készült memóriával, villanykörték sora volt a kijelzője; a *Z2* gépében jelfogós (relés) memória volt; 1941-ben a *Z3* relés lebegőpontos aritmetikai egységgel rendelkezett.

Howard *Aiken* vezetésével készült a *MARK I* az IBM támogatásával a Harvard egyetemen (USA, 1943-44). Telefon-relékből épült a gép, eredeti célja a telefonbeszélgetések számlázása volt, de a háborús viszonyok miatt lőelem-táblázatok számítására használták.

A II. világháború nagy lökést adott a fejlődésnek. A kódfejtés és a logisztika számításigényes feladataira az angolok több számítógépet is kifejlesztettek és használtak, csak azok a titoktartás miatt nem váltak ismertté. A fejlesztés központi alakja Alan *Turing* (1912-1954) matematikus [Pioneers of Computing] volt. Ekkor készültek a *Robinson* számítógépcsalád tagjai, és 1943 decemberében már működött a *Colossus I*, a világ első elektroncsöves számítógépe. Churchill szerint a kódfejtő számítógépek hozzásegítették Angliát a győzelemhez.

1939-től kezdve az USA-ban is dolgoztak elektroncsöves számítógép fejlesztésen (Presper *Eckert* és John *Mauchly* [Pioneers of Computing] a Pennsylvaniai Egyetem Műszaki Karán). 1946-ra készült el az *ENIAC* (Electronic Numerical Integrator and Computer), mely ún. külső programvezérlésű gép volt. A programot lyukkártyákra lyukasztották, az adatokat 20 db tízjegyű regiszterben tárolták. A gép 18 ezer elektroncsövet tartalmazott, elképzelhetjük az áramfelvételét! Működési sebessége viszont ezerszer gyorsabb volt, mint a MARK I sebessége.

Neumann János (1903-1957) [Pioneers of Computing] magyar származású matematikus és vegyész Herman *Goldstine* kollégájával együtt 1946-ban megfogalmazta, 1948-ban egy konferencián előadta az elektronikus digitális számítógépekkel szembeni követelményeket. A *Neumann elv* hosszú időre meghatározta a számítógépek fejlesztési irányát. Az első *tárolt programú* számítógépet (EDSAC) mégsem a Neumann által vezetett csoport készítette (csak 1951-re fejezték be Neumannék az EDVAC-ot), hanem az angliai Cambridge University-n Maurice *Wilkes*.

Tovább most nem megyünk a számítógép történelemben, mert a századunk második felében lezajló hardveres fejlődést célszerű a működtető rendszerek (operációs rendszerek) és a számítástechnikai munkamegosztás fejlődésével, a specializálódással párhuzamosan tárgyalni. Annyit megelőlegezhetünk, hogy szokásos az 1945-55 közötti időszak gépeit az *első generációsnak*, az 1955-65 közötti időszak fejlesztéseit a *második generációs rendszereknek* (tranzisztorok és kötegelt rendszerek), az 1965-80 közötti időszakot a *harmadik generációnak* (integrált áramkörök és multiprogramozás), végül az 1980-tól napjainkig is terjedő időszakot a *negyedik generációs* gépek korszakának (személyi számítógépek és LSI) nevezni.

1.2. A Neumann elv

Neumannék híres cikkének lényege - az elektronikus számítógépekkel szembeni követelmények - lényegében 3 pontba foglalhatók össze. Az 1. pont a számítógép fő részeit és az azokkal szembeni kívánalmakat fogalmazza meg, a második pont a tárolt program elvet rögzíti, a harmadik pedig az automatikus, az emberi beavatkozás nélküli működési követelményt rögzíti. Nézzük ezeket a pontokat:

1. A fő funkcionális részek a következők:
   - a vezérlő egység (control unit),
   - az aritmetikai és logikai egység (ALU),
   - a tár (memory), ami címezhető és újraírható tároló-elemekkel rendelkezik, továbbá
   - a ki/bemeneti egységek.
   - A részegységek elektronikusak legyenek és bináris számrendszert használjanak. Az ALU képes legyen elvégezni az alapvető logikai és aritmetikai műveleteket (néhány elemi matematikai és logikai művelet segítségével elvileg bármely számítási feladat elvégezhető).
2. Tárolt program elvű legyen a számítógép, azaz a program és az adatok ugyanabban a tárban tárolódjanak, ebből következően a programokat tartalmazó rekeszek is újraírhatók.
3. A vezérlő egység határozza meg a működést a tárból kiolvasott utasítások alapján, emberi beavatkozás nélkül, azaz közvetlen vezérlésűek a számítógépek.

A harmadik pont azt jelenti, hogy létezik *utasítás készlet* (*instruction set*), melyek utasításait a vezérlő képes felismerni és az ALU-val elvégeztetni. Az utasításhalmaz elemeinek célszerű sorozata a tár (rendszerint egymásutáni) címezhető celláiban van. Úgyis fogalmazhatunk, hogy adott és a memóriában tárolt egy *utasításfolyam (instruction stream)*: a gépi kódú program (kód: code, program text stb.). A vezérlőegység egy mutatója jelöli ki a soron következő végrehajtható utasítást (instruction) az utasításfolyamban. A soron következő gépi utasítást a vezérlő egység értelmezi. A gépi utasításokban kódolva vannak az elvégzendő operációk, kódolva vannak az operandusok, azaz az adatok, vagy az adatok tárbeli címei. Ezeket a vezérlő egység a tárból előveszi, az ALU pedig elvégzi rajtuk az operációkat. A tárolási helyek címezhetők, a tárolási helyeken a tárolt értékek változtathatók. Egy instrukció végrehajtása után a vezérlőegység mutatója automatikusan - emberi beavatkozás nélkül - a soron következő instrukcióra mutat, a vezérlő egység veszi ezt az instrukciót s. í. t. Neumannék nem használták a CPU (Central Processing Unit) kifejezést, de mi mondhatjuk, hogy a CPU az *utasítás-számláló regisztere (PC: Program Counter, IP Instruction Pointer*) mutatja a soron következő instrukció címét.

A második pontból következik, hogy maga a program is feldolgozható, módosítható.

A három pont együtt azt mondja: a számítógép architektúra hardver és szoftver architektúrák együttese, hiszen működését nemcsak a hardver szabja majd meg.

1.3. Az architektúra fogalma

Az architektúra alatt kétféle dolgot értünk, a számítógép architektúra fogalmat két, eltérő nézőpontból értelmezhetjük. A leírás irányultságában van különbség a két értelmezésben.

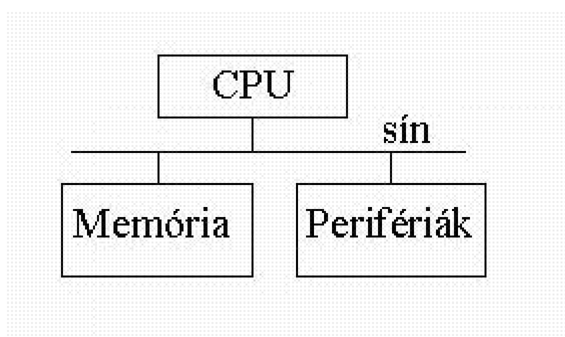
Irányulhat funkcionális *specifikációra* a leírás. Ekkor a gépet (esetleg egyes részeit) fekete doboz szemlélettel nézzük. Egy digitális számítógép bizonyos szintű *általános specifikáció-jára* gondolunk ekkor, például a processzor utasításkészletének, társzervezésének és címzési módjainak, a B/K műveleteknek (és vezérlésüknek) felhasználói (programozói) leírását stb. Ebben az értelemben lehetnek *közös (hasonló) architektúrával rendelkező számítógépcsalá-dok*, melyeknél a megvalósítás (az implementáció) különbözhet. A felhasználó (programozó) szempontjából az *architektúra azonossága* (hasonlósága) biztosítja a *kompatibilitást* (helyet-tesíthetőséget, áthelyezhetőséget, csatlakoztathatóságot), például egy adott programnak a család minden tagján működnie kell.

Irányulhat *megvalósítási* célra a leírás. Ez egy másik (villamosmérnöki, hardvertervezői) szempont, itt az architektúra egy számítógép (rendszer) lényeges részei, fő funkcionális ele-mei kapcsolódásának leírását jelenti valamilyen szinten. Ebben a leírásban is lehetnek funk-cionális elemek, de nemcsak a funkciók felsorolása, specifikálása van itt, hanem a funkciók kapcsolódása is. Ez lehet blokkdiagram, kapcsolási rajz (különböző részletességben), de lehet a felépítés (részben) szöveges leírása is.

És persze, bármi is volt a leírás irányultsága, az architektúra fogalmat különböző szinteken - amik persze egymásra épülhetnek - értelmezhetjük. Így beszélhetünk mikrogép-szintű archi-tektúrákról (mikroprogramozott processzoroknál), processzorszint is van, illetve számítógép-rendszer szinten is értelmezhetjük az architektúrát.

1.3.1. Egy számítógép hardver architektúrája

A legáltalánosabb architektúrát (a második értelemben, ahol is a részeket és kapcsolódásukat tekintjük) az 1.1. ábrán láthatjuk.



1.1. ábra. Egy számítógép architektúrája

E szerint egy számítógép a *sín*re kapcsoló-dó *központi egység*ből (CPU, Central Processing Unit), a *központi tár*ból (Central Memory) és a *perifériák*ból áll. Mindezeket a sín köti össze.

A memória adatokat (bit, bájt, szó, blokk, rekord mezőkkel) és gépi instrukciókat tároló címezhető cellák készlete. A számí-tógép működése során állapotokat vesz fel, állapotátmenetek történnek az állapotok között. Az állapotot egy adott pillanatban a memóriacellák pillanatnyi állapota és a perifériák pillanatnyi állapota adja. Egy gépi instrukció végrehajtása változtat az állapoton: azzal például, hogy valamelyik memória

cella értékét megváltoztatja. Ez egy sokdimenziós állapottér. Magának a programnak (az instrukciók sorozatának) futása állapotátmenetek láncolatát jelenti. A sokdimenziós állapottér átmeneteinek megragadható egy kulcsjellemzője: a programszámláló regiszter (PC/IP) „állapotainak átmenete”, e regiszter egymásutáni értékeinek sorozata. Nevezhetjük ezt a sorozatot a „programvezérlés menetének” (flow of control).

Az imperatív programnyelvek - ezekkel a programvezérlés menetét manipuláljuk - jól megfelelnek a Neumann elvű gépeknek.

A Neumann elvű gépeknél jellegzetes a hiba- és eseménykezelés. Az elgondolás a hiba- és eseménykezelés módja mögött a következő: van egy program rész a memóriában, mely az esemény kezelője (handler). Esemény bekövetkeztekor a programvezérlés (normális) menete megszakad, a vezérlés a kezelőre ugrik, és ha abból vissza lehet térni, folytatódik a normális programfutás a megszakítási pont után. Vagyis a hibakezelés is a programvezérlés menetének manipulálásával történik.

Kontrasztként, hogy igazán megértsük a Neumann gép működését, érdemes összevetni azt más elvű számítógéppel, például az adatfolyam géppel.

Az adatfolyam gépnél szeparált processzorok vannak minden operációra (az operáció kb., megfelel a Neumann gép instrukciójának). A processzorok (az operációk) „várnak”, míg operandusaik értékei előállnak, s mikor azok előállnak, azonnal „működnek”, adják az eredményüket. Az eredményüket további processzorok/operációk ezután már használhatják. Az operációk végrehajtási sorrendjét nem imperatív jellegű (tedd ezt, aztán ezt s.í.t.) program szabja meg, hanem az az adatfolyamból adódik. A hibakezelés is jellegzetes. Az egyes processzorok/operációk operandusainak explicit hibaértéke is van. Ha egy operáció „hibás” (pl. túlcsordulás következik be aritmetikai operáció során), akkor az a hibaértéket állítja elő eredményül. Mindenképp kell legyen eredménye egy-egy operációnak, hiszen azt (azokat) más processzorok/operációk „várják”. Nos, ha egy processzor hibás értéket kap operandusaként, akkor ő is hibaértéket fog továbbítani.

További összehasonlításként nézzük a két gép számítási modelljének összetevőit (a számítások alapelemeit, a problémaleírás jellegét és módszerét, a végrehajtás modelljét: szemantikáját és kontrollját)!

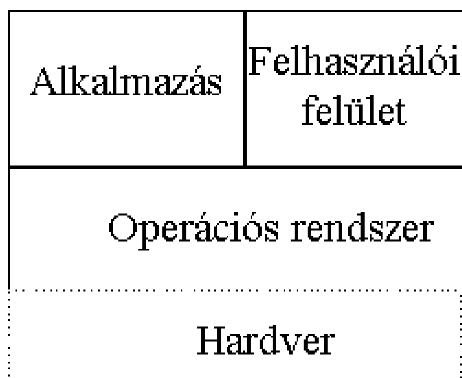
A Neumann gépnél az alapelemek azonosítható entitásokhoz rendelt (típusos) adatok: változók, amik többszörösen vehetnek fel értékeket. A problémaleírás procedurális/imperatív: lépésenként van megadva, mit kell tenni. A végrehajtás szemantikája: állapotátmenet szemantika. A kontroll: közvetlen vezérlés (beszélhetünk a vezérlés menetéről...).

Az adatfolyam gépnél az alapelemek azonosítható entitásokhoz rendelt (típusos) adatok, de egyszeri azokon az értékadás. Az adatoknak lehet neve, de azok nem változók! Ha változók lennének az operandusok, nem tudná a processzor, melyik értékkel kell végrehajtani az operációt. A problémaleírás deklaratív: a „program” operációk felsorolása, halmaza, leírásuk sorrendje nem befolyásolja a processzorok működési sorrendjét! A végrehajtási szemantika applikatív. A kontroll: adatfolyam vezérelt.

A későbbiekben részletesebben is tárgyaljuk Neumann gép hardver architektúráját, az egyes részeket.

1.3.2. A szoftver architektúra

Ugyancsak általánosan és az architektúra fogalom második értelmében a szoftver architektúra az 1.2. ábrán látható.



1.2. ábra. A szoftver architektúra

Az ábra elvi jelentőségű és nagyon általános, természetesen lehetnek más, az ábrán nem szereplő szoftver komponensek is egy számítógép szoftver architektúrájában. A feltüntetett komponensek talán a legfontosabbak: a *felhasználói kapcsolattartó* (User Interface), a *segédprogramok* (Utilities), az *alkalmazások* (Applications) és hát persze maga az *operációs rendszer* (Operating System). A legfontosabb, amit megfigyelhetünk az egyes részek kapcsolódásában a *rétegezettség*!

1.3.3. Rétegezettség (Layered architecture)

A réteges szerveződés általános alapelv, sok helyütt megfigyelhető a számítástechnikában (vö. strukturált programozás, hálózati protokollok rétegei stb.). A lényege:

- Egy alsóbb réteg szolgáltatásokat biztosít a felső rétegnek. Biztosít egy magasabb absztrakciós szintű *virtuális utasításkészletet*.
- A felső réteg nem látja az alatta lévő réteg megvalósítását, annak részleteit, csak a virtuális utasításkészletét. A még lejjebb lévő rétegek részletei pedig teljesen el vannak rejtve a felső réteg elől: a közvetlen alatta lévő réteg elszigetel.
- Jól meghatározott *protokollok* és *interfészek* kellenek az egyes rétegek között.

Az ábránkon az operációs rendszer - mint egy réteg - elválasztja az alkalmazásokat, a segédprogramokat, sőt a felhasználói kapcsolattartó felületet is a hardvertől. Az elválasztás valahogy függetleníti ezeket, az a képzetünk - és ez bizonyos mértékig igaz is -, hogy a hardver akár „le is cserélhető” az operációs rendszer alatt. Másrészt - ez az ábrából azonban nemigen derül ki -, ezt az elszigetelést az operációs rendszer úgy valósítja meg, hogy szolgáltatásokat biztosít a felette lévő réteg számára. A szolgáltatások hívhatók a felsőbb rétegből: akár azt is mondhatjuk, hogy az operációs rendszer egy virtuális gépet emulál, és ennek a virtuális gépnek a szolgáltatások hívásai az utasításai. Ezek az utasítások virtuálisak. Egy érdekesség: ha az operációs rendszer szolgáltatásait specifikáljuk, mondjuk felsorolva, hogy milyen virtuális instrukciókat biztosít az alkalmazások számára, akkor az már az architektúra az első értelemben.

Gyakran fogjuk használni a *virtuális* (virtuális objektum, virtualitás) és a *transzparens* (transzparens objektum, transzparencia) jelzőket (fogalmakat). Mit jelentenek ezek?

1.3.4. A virtualitás fogalma

Virtuális egy objektum, ha valójában nem létezik, de úgy tűnik, mintha (ott) volna.

Példák:

- virtuális diszk eszköz, amit egy hálózati fájl-szerver biztosít.
- virtuális egy terminál eszköz, ha azt pl. emulálja egy szoftver.
- virtuális memória, ami egy-egy futó program rendelkezésére áll, míg a valóságban annak a gépnek sokkal kisebb a központi memóriája stb.

1.3.5. A transzparencia fogalma

Transzparens (átlátszó) egy objektum, ha valójában ott van, de nem látszik, nem vesszük észre, nem kell vele törődnünk.

Példa:

- Mialatt a fájl-szerver biztosít egy virtuális diszk eszközt, maga a hálózat, a hálózati szolgáltatások transzparensek, nem látszódnak. A virtuális diszkre ugyanúgy a nevével hivatkozhatunk, mint egy valódi (reális) diszkre, nem törődünk közben a hálózattal, nem is vesszük észre, hogy hálózaton is dolgozunk (legföljebb ha a virtuális diszk lassú :-)).

2. Számítógép használati módok

Használható-e a számítógép operációs rendszer (működtető rendszer) nélkül? Egyáltalán, hogyan használhatunk egy számítógépet? Nyilván, kellenek programok hozzá, melyek valahogy betöltődnek a memóriába és elindulnak, de milyen szinteket tudunk a használatban elképzelni? A válaszunk: legalább három szint elképzelhető.

- **Direkt futtatás a hardveren**, azaz a gép használata működtető rendszer nélkül. Ma ez a használat csak kis bitszélességű mikrokontrollereknél szokásos, de régebben ez természetes használati mód volt. Természetesen ekkor minden felelősség a felhasználóé. Teljes részletességgel ismernie kell a hardvert, gépi utasítás szinten kell tudni programoznia. Felmerül a kérdés ekkor: hogyan programozhatunk egyáltalán? (Rövid válasz erre: külön berendezéseken programozunk és az elkészített programot "beleégetjük" a memóriába. A gép bekapcsolásakor a beégetett program elindul.)
- **"Monitor" program segítségével**. A monitor egy egyszerű működtető szoftver, ami képes
  - memória rekeszek lekérdezésére, beállítására;
  - tesztekre;
  - esetleg primitív eszközkezelésre (pl. egy konzol és egy diszk kezelésére).
- **Operációs rendszer segítségével**. Ma ez az általános!

Mi az operációs rendszer?

Szoftver. Működtető rendszer. *B. Hansen* szerint az operációs rendszer egy csomó kézikönyv és egy automatikusan működő eljárás, amelyek lehetővé teszik, hogy különböző személyek *hatékonyan* használjanak egy számítógéprendszert. Mit jelent itt a hatékonyság? Hogy mérhetjük tárgyilagosan? Nos, néhány lehetőség:

- Milyen funkciókat biztosít?
- Milyen a teljesítménye (performance): a válaszidő, a fordulási idő, a CPU felhasználási idő (vagy fizetendő forint), a memóriamennyiség, a kommunikáció, a csatornák teljesítménye stb.
- Kényelmesség.

Láthatók az ellentmondások! (Pl. funkcionalitás és CPU idő ára között, funkcionalitás és válaszidő között.)

Az operációs rendszer fogalmat két szempontból definiálhatjuk (egy harmadik szempontot pedig megemlítünk):

I. Az operációs rendszer kiterjesztett/virtuális gép (Extended/Virtual Machine)

- Magasabb szintű, absztraktabb "utasításokat" és "objektumokat" (pl. fájl nevek) biztosít;
- elrejti a részleteket;
- kezelhetővé teszi a gépet, anélkül, hogy a hardvert ismerni kellene.

II: Az operációs rendszer erőforrás menedzser (Resource Manager)

- Védi, kiosztja, összehangolja az erőforrásokat.

III. Az operációs rendszer egy válaszoló rendszer (Response System)

Az operációs rendszer egy válaszoló rendszer, mondhatná egy definíció. Hiszen válaszokat ad kérelmekre. A kérelmek jöhetnek

a felhasználó parancsaiból;

alkalmazásokból (rendszerhívással, kivételes eseménnyel);

a hardverből (megszakítással).

Nem tartom elég jónak ezt a definíciót. Azért, mert ezzel a definícióval hajlamosak leszünk összekeverni a felhasználói és programozói felületeket az operációs rendszerrel.

2.1. Mit lát egy felhasználó, ha leül egy számítógép elé?

Tessék idézőjelben venni a „mit lát" kifejezést. Arról van szó, hogy mivel kell foglakoznia, mit kell megtanulnia, megismernie. Úgy is feltehetjük a kérdést, mit nem *lát* a felhasználó, mi van elrejtve előle, mivel nem kell foglakoznia.

A felhasználó valóságban egy végberendezést, egy *terminált* lát.

Ma ez egy képernyő (megjelenítő), a billentyűzet és a mutató (utóbbiak a beviteli eszközök) együttese. A megjelenítő lehet grafikus. Lehet a terminál emulált: egy futó program biztosít a képernyőn egy „ablakot", az ablakhoz tartozik billentyűzet és mutató. A felhasználó a beviteli eszközöket használva egy *parancsnyelvvel* vezérli, kezeli a gépet (a gépen futó programokat), nézi, mi jelenik meg a megjelenítőn, értelmezi a *válasznyelvi* elemeket.

Mikor a gépet vezérli, valójában egy parancsértelmező processz fut számára, ami az operációs rendszer szolgáltatásain át, azokat kérve vezérel.

A végberendezés a *terminál.* Ez a géppel való kommunikáció alapvető hardver eszköze. Nem érdemes most sokat foglakozni vele, mert van ennél több és fontosabb absztrakt „látnivaló"! Később persze foglakozunk a terminálokkal is., de most soroljuk fel az absztrakt látnivalókat. Ezek a következők: a felhasználói felület, vagy parancsnyelv; a processzek, vagyis a futó programok; az eszközök és a fájlok; más felhasználók és hozzáférési kategóriák; gazdagépek és szolgáltatásaik.

A felhasználó lát egy *felhasználói kezelői felületet.* (User Interface) Manapság ez kétféle lehet:

- parancsértelmező (Command Language Interpreter, CLI) (esetleg menüfeldolgozó), vagy
- egy grafikus felhasználói felület (GUI, Graphical User Interface).

Vegyük az elsőt, a parancsértelmezőt!

A parancsértelmező képes adni a megjelenítőn egy *készenléti jelet* (prompt), ami mutatja, hogy most billentyűzet beviteli eszközön begépelhetünk egy *parancsot.* A parancsokat a parancsértelmező elfogadja, elemzi, átalakítja és végrehajtja. A végrehajtással „válaszol".

„Látunk” tehát *parancsokat*. Megkell ismernünk a parancsokat, szintaxisukat, értelmezésüket.

Néha a parancsértelmező elindít számunkra egy programot, készít számunkra egy processzt, aminek *saját felhasználói kapcsolattartó rendszere* van. Ez utóbbi is lehet parancsértelmezős (esetleg menüs) jellegű, vagy GUI jellegű. Tudnunk kell ("látnunk" kell), éppen mivel tartjuk a kapcsolatot, mert a szabályok eltérhetnek!

Látunk tehát futó programokat: *folyamatokat* (processzeket).

Menjünk vissza a parancsértelmezőhöz. Miközben parancsokat adunk, argumentumokként sokszor használunk eszköz- ill. fájlneveket. Ezeket is *látjuk*.

Eszközöket ismerünk a (szimbolikus) nevükön, *fájlokat* látunk a nevükön át. Használjuk ezeket a neveket.

Némely rendszerben némely parancsban személyek nevét vagy címét kell szerepeltetni (pl. levelező rendszerben az e-mail címet).

Látunk tehát személyeket, *felhasználókat* (user), azok postaláda címeit stb. A személyekhez kapcsolódóan *látunk tulajdonossági* kategóriákat. Úgy látjuk, hogy ez és ez az objektum (pl. fájl) ezé és ezé a személyé. Később látni fogunk *védelmi, hozzáférési* kategóriákat is. Ezt és ezt a fájlt használhatjuk *olvasásra*, de nem változtathatjuk meg. Másokat viszont *írhatunk*, hozzáfűzéses módon írhatjuk, esetleg teljesen *újraírhatjuk*. Vannak bináris adatokat tartalmazó fájlok, melyek végrehajtható programokat tartalmaznak, és ezeket mi *futtathatjuk*, másokat viszont *nem futtathatunk*.

Néha kapcsolatot létesítünk más számítógépekkel. Ehhez *látunk csomópontokat*, *gazda* gépeket (host) és látjuk azok *szolgáltatásait*.

Azt nem biztos, hogy látjuk, hogyan valósult meg a kapcsolat, milyen áramkörökön (virtual circuit) kapcsolódnak össze a gépek, milyen üzenetváltások mennek közöttük, de tudjuk, hogy vannak más gépek, és ezekkel tudunk kapcsolatot létesíteni.

Nem látunk viszont egy sor dolgot!

- magát az operációs rendszer *nem látjuk* igazán;
- *nem látjuk* a CPU-t, a memóriát, ezek kapcsolódásait;
- *nem látjuk* a diszkeket, azt, hogy azokon hogy szerveződik az adattárolás stb.

A következőkben a "látnivalókat " részletezzük.

2.1.1. A folyamat (processz)

A processz egy (párhuzamos programszerkezeteket nem tartalmazó) végrehajtható program futás közben. Vegyük észre a végrehajtható program és a processz közti különbséget!

A *végrehajtható program* egy fájl. Statikus, van mérete, helyet foglal egy fájlrendszeren.

A *folyamat* egy végrehajtható program futási példánya. Dinamikus, időbeli tulajdonságai is vannak, sőt, azok a fontosabbak. Ugyanannak a végrehajtható programnak több futási példánya is lehet ugyanazon rendszerben.

A folyamatoknak vannak menedzselési információi, ezekből a felhasználó számára legfontosabbak az azonosítási információk: a *pid* (process identifier), a *pname* (process name).

A *pid*-et bizonyos parancsok argumentumaiban használjuk, jó tehát tudni azokat. A parancsértelmező valamely parancsával lekérdezhető, hogy milyen azonosítójú folyamatok élnek a rendszeren. A Unix burokban ilyen lekérdező parancs a ps parancs (process state szavak rövidítéséből jött a parancs neve).

Egyes rendszerekben a folyamatok szekvenciálisan futnak (nem párhuzamosan, single tasking módon). Ekkor, bár több folyamat lehet a memóriában, egyszerre csak egy folyamat aktív. Ilyen rendszer volt pl. az MS-DOS.

Vannak rendszerek, melyeken a folyamatok virtuális párhuzamosságban (multi tasking) futnak. Az ilyen rendszerekben kevesebb processzor van, mint processz. Az operációs rendszer ezt az egyetlen processzort megosztja a folyamatok között. Ilyen rendszerek voltak a korai Unix-szerű operációs rendszerek, a VAX/VMS stb. egyetlen processzorral.

Folyamatok futhatnak valós párhuzamosságban (multi processing) is. A multi processing rendszerekben több processzor (CPU) van. Persze, itt is előfordulhat, hogy több a processz, mint a processzor. A korszerű operációs rendszerek korszerű hardvereken multi-processing jelleget adnak.

Kérdezhetnénk, miért kell az egyszerű felhasználónak a processzekkel foglakozni, például azonosítójukat ismerni? Nos, azért, mert vezérelni akarjuk a folyamatokat! Ha pl. a gépünk „nagyon lassú”, lehet, hogy túl sok processzt futtatunk egyszerre. A kevésbé fontosakat jó lenne terminálni, „lelőni”. Lehet, hogy szinkronizálni akarjuk a folyamataink futását. Egyik eredményét fel akarhatjuk használni a másik bemenetelére. Hasznos, ha „látjuk” a processzeinket.

Jegyezzük még meg, hogy grafikus felhasználói felületekkel dolgozva is "*látjuk*" a processzeket! Ikonként látjuk őket, kinyitott ablakukat látjuk, tudjuk, hogy léteznek, kezeljük őket.

2.1.2. A felhasználói kezelői felület

A felhasználói felület (a gépkezelő felület, a parancsértelmező) is egy (esetleg több) processz. Manapság alapvetően két osztályuk van:

- **(Interaktív/kötegelt) parancsértelmező processz a felület**. Ilyen pl., az
  - MS-DOS parancsértelmező processze (a command.com fut benne), a
  - VAX/VMS DCL-je (Digital Command Language), ilyenek a
  - Unix burkok (sh, ksh, bash, csh, tcsh stb.)

A parancsértelmezők használhatók alfanumerikus terminálokról (akár emulált terminálokról is). A parancsértelmezős felület alesete a menüvezérelt felület.

- **Grafikus felhasználói felület** (GUI, Graphical User Interface) Pl. az
  - MS-DOS MS-Windows 3.1, a Windows 2000 grafikus felülete stb., a
  - VAX/VMS DECWindows felülete, a
  - Unix munkaasztalok (Desktop), a CDE (Common Desktop Environment) stb.

Ezekhez természetesen grafikus terminálok kellenek!

Lássuk be a következőket

A gép kezelése során van egy "parancsnyelvünk" (command language), amin megfogalmazzuk a parancsainkat. A kezelő folyamatnak pedig van egy "válasznyelve" (respond language), amin adja a válaszokat: ez lehet egy hibaüzenet, egy nyugtázás stb. Tulajdonképpen a grafikus objektumok manipulálásán alapuló kapcsolattartáskor is felfedezzük a parancsnyelvi és a válasznyelvi elemeket! Talán egyszerűbbek itt a válasznyelvi elemek: az ablakok, nyomógombok, legördülő menük, ikonok stb. A parancsnyelvi elemek pedig itt: a kijelölések, kiválasztások (kattintások, kettős kattintások, a vonszolás stb.).

Mi a különbség a *kötegelt* és az *interaktív* kapcsolattartás között? Történelmi sorrend szerint a kötegelt kapcsolattartás alakult ki előbb. A kötegelt kapcsolattartásban a felhasználó valahogy el van választva a processzei (taszkjai) futásától. Nem tudja, mikor futnak azok, nem tud a futásuk közben beavatkozni. Előre meg kell mondania, hogy a processzei milyen bemeneteket kérnek, az esetleges változatokra is felkészülve. Az eredményeket pedig a processzek fájlokba, esetleg nyomtatott listákba teszi, amiket a felhasználó a futási időn kívül böngészhet. Az interaktív kapcsolattartás során a processzek futása közben - a parancsnyelvi-válasznyelvi elemek segítségével - akár közbe is avatkozhatunk, dönthetünk a válaszokról, felelhetünk feltett kérdésekre, akár *eseményeket* generálhatunk a futó processzeink számára. A mai kapcsolattartó felületektől elvárjuk, hogy mind kötegelt (háttérben futó), mind interaktív használatot biztosítsanak.

Vegyük észre: az operációs rendszer *nem a kapcsolattartó felület* és fordítva, *a kapcsolattartó nem az operációs rendszer*! Az operációs rendszer híd az alkalmazások - többek között a kapcsolattartó felületek - és a hardver között, míg a kapcsolattartó a felhasználót segíti, hogy a gépet kezelje, persze, az operációs rendszer szolgáltatásai segítségével. Sok rendszerben a kapcsolattartó, a burok, szorosan kapcsolódik az operációs rendszerhez (része annak), máshol pedig meglehetősen független magától az operációs rendszertől!

2.1.3. Az eszköz és fájlrendszer

A felhasználó a parancsokban gyakran használ - argumentumokként - eszköz- és fájlneveket.

Az eszköz (device): szimbolikus névvel ellátott periféria, komponens.

Pl.

```
MS-DOS         A:        floppy eszköz
               COM1:     soros vonal
               stb.
VAX/VMS        TT:       terminál
               stb.
```

Többféle eszközosztályt ismerünk. Vannak

- strukturált (blokkorientált) eszközök (diszkek, szalagok stb.) Fájlrendszer (file system) képezhető rajtuk.
- Vannak karakter orientált eszközök (terminálok, printer vonalak stb.)
- Végül említenünk kell a különleges eszközöket. Ilyen pl. az óraeszköz, mely a számítógépen az idő kezelését teszi lehetővé.

Az eszközök mellet „látunk” fájlokat is.

A felhasználó (és az operációs rendszer) szemszögéből **a fájl** *névvel ellátott, valamilyen szempontból összetartozó adatelemek együttese.* A nevekre vonatkozóan lehetnek konvenciók és korlátozások. A fájloknak vannak tulajdonságaik (attribútumaik).

A fájlokat alkotó **adatelemek** lehet bájtok (byte) vagy karakterek, szavak, rekordok.

Lehet egy fájl különböző *szervezettségű* (organisation). Ez az adatelemeknek az adathordozón való tényleges rendezettségét jelenti. Az operációs rendszer biztosít(hat)ja a szervezettséget, és jó, ha erről a felhasználónak tudomása van.

A fájl a "tartalma" szerint is osztályozható: vannak egyszerű szövegfájlok, vannak dokumentum fájlok (a szöveg mellett szerkesztésükre is vonatkozó információkkal), vannak bináris fájlok, ezen belül mondjuk végrehajtható programok, kép- és hang-fájlok, adatfájlok stb.

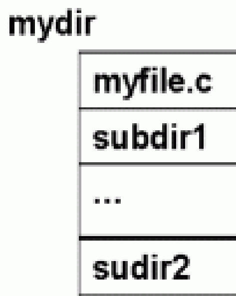
Az operációs rendszer rutinjai különböző operációkat biztosít a fájlok elérésére a processzek számára: ezek a *fájl elérések*. A programozóknak kell ismerniük a különböző elérési módszereket, ráadásul az elérési lehetőségek függhetnek a szervezettségtől. Az egyszerű felhasználó nem feltétlenül foglakozik a fájl elérésekkel.

Minden fájlról tudjuk, hogy melyik eszközön van.

Névkonvenciók és korlátozások (restrikciók a név hosszúságra, karakterkészletre stb.) vannak az egyes operációs rendszerekben. Ezeket is ismernie kell a felhasználóknak.

Ha az eddig elmondottakat átgondoljuk, a következő elképzelésünk lehet: valamely strukturált eszközön vannak fájlok, van egy fájl-halmaz. A halmazhoz fájlok tartoznak, azokat azonosíthatjuk a neveikkel. Az eddig elmondottakból nem következik az, hogy rendezni tudnánk a fájljainkat. Márpedig szeretnénk valahogy rendezni, strukturálni ezt a halmazt! Például csoportosítani, együtt kezelni bizonyos fájlokat. Ez az igény vezetett a jegyzék koncepció kialakításához.

A jegyzék (katalógus, directory): egy *fájl, ami bejegyzéseket tartalmaz más fájlokról*. (Nevüket, elhelyezésükre utaló információkat, esetleg fájl tulajdonságokat: attribútumokat stb.). Ugyanabba a jegyzékbe bejegyzett fájlok egy csoportot alkotnak, elkülöníthetők ezzel más jegyzékbe bejegyzett fájloktól.



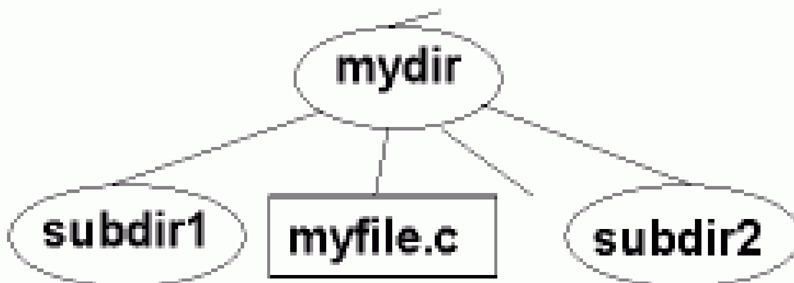
2.1. ábra. Bejegyzések egy jegyzékbe

A mai magyar számítástechnikai nyelvben *könyvtáraknak* nevezik a jegyzékeket. Én nem helyeslem ezt az elnevezést, mert a *könyvtár* (library) fogalom egy különleges fájlra vonatkozik, amiben *könyvek/tagok* (members) az összetevők. Az ilyen fájlokat a *könyvtárkezelő segédprogramok* (librarian/ar) képesek kezelni, illetve a tárgyprogramokat, mint tagokat tartalmazó könyvtárból a futtaható programokat építő segédprogramok (linker/ task builder) képes hivatkozásokat feloldani, a makro-könytárakból az assemblerek képesek makrókat beépíteni a forrásprogramunkba. Azt hiszem azonban, szélmalomharc, ha a directory fogalom helyes magyar elnevezésért, a könyvtár szóhasználat ellen szólok, annyira elterjedt ez a szóhasználat.

A 2.1. ábrán a *mydir* jegyzék tartalma néhány bejegyzés, többek között a *myfile.c* nevű fájlról, a *subdir1* és a *subdir2* nevű jegyzékről.

Néha szokás úgy fogalmazni, hogy a myfile.c fájl a mydir *jegyzékben van*. Valójában a *jegyzék nem tartalmazza a bejegyzett fájlokat*, hanem csak a nevüket, az elhelyezkedésükre utaló információkat, esetleg egyéb attribútumaikat (méretüket, készítési dátumaikat, tulajdonosuk azonosítóit stb.).

Mindenesetre a jegyzékbe való bejegyzés (a "benne van" reláció) egy *szülő-gyermek reláció* (lásd 2.2. ábra)!



2.2. ábra. Szülő gyermek reláció a jegyzékbe való bejegyzés

A szülő reláció egy-az-egy reláció: minden fájlnak (látjuk majd az egyetlen kivételt) egy szülője van. A gyermek reláció egy-a-több reláció: egy jegyzéknek több gyermeke lehet.

Korszerű operációs rendszerekben - egy kivételével - minden fájl be van jegyezve egy jegyzékbe.

A szülő-gyermek reláció kiterjesztésével többszintű hierarchikus faszerkezet alakul ki.

A gyökér jegyzék az eszköz kitüntetett jegyzéke. Az a jegyzék, ami nincs bejegyezve más jegyzékbe. Kiindulópontja a hierarchikus faszerkezetnek. Kitüntetett "helyen" van az eszközön.

Van szimbolikus neve, ez operációs rendszer függő:

```
MS-DOS        \
VAX/VMS       [000000]
Unix          /
```

A fájl-rendszer (File System): blokk-orientált eszközre képzett, hierarchikus struktúra, melyben

- a fájlok azonosíthatók, kezelhetők; attribútumaik, blokkjaik elérhetők;
- az eszköz blokkfoglaltsága menedzselt.

A fájl-rendszeren azonosíthatók a fájlok az ösvény fogalom segítségével.

Az ösvény (path): szülő-gyermek relációban lévő jegyzék-nevek listája (a lista vége lehet fájlnév is), mely valamely jegyzékből kiindulva másik jegyzéket, fájlt azonosít.

A listaelválasztó karakter operációs rendszer függő:

```
MS-DOS    \
VAX/VMS   .
Unix      /
```

Az ösvény indulhat

- a munkajegyzékből: ez a **relatív ösvény**;
- a gyökérjegyzékből: ez **az abszolút ösvény**.

A munkajegyzék (aktuális jegyzék, Working Directory, default directory): *az operációs rendszer által feljegyzett és ezzel kitüntetett jegyzék.* Miután az operációs rendszer feljegyzi, erre nem szükséges explicite hivatkozni, ebben gyorsan tud keresni. A relatív ösvény kiindulópontja. A munkajegyzék váltható (beállítható), van erre külön parancs (rendszerint a cd parancs).

Ha mégis szeretnénk hivatkozni rá, van külön szimbolikus neve. Ez a név a fájlrendszerbeli nevétől független hivatkozási lehetőséget biztosít, mindig a pillanatnyi munkajegyzéket azonosítja. A munkajegyzék szimbolikus név is operációs rendszer függő, de a három eddig említett operációs rendszerben ugyanaz: . (a dot).

A szülő jegyzék (Parent Directory): *egy fájl szülője*. Munkajegyzékre vonatkoztatva a relatív ösvény kijelölését egyszerűsíti. A fájlrendszerbeli nevétől függetlenül, azon kívül van szimbolikus neve:

```
MS-DOS        ..
VAX/VMS       -
Unix          ..
```

Még egy fogalom hasznos lehet, **az aktuális eszköz** (Default Device) fogalma. Ez az operációs rendszer által feljegyzett, ezzel kitüntetett eszköz, melyre nem kell explicite hivatkozni. Szimbolikus neve a VAX/VMS-ben van csak.

2.1.4. A felhasználók

Többfelhasználós operációs rendszerekben saját magunkon kívül "látunk" más felhasználókat is. Sőt, azt is látjuk, hogy a felhasználók csoportokhoz tartoznak.

A felhasználókkal való kommunikációhoz ismernünk kell mások neveit, elektronikus levélcímüket stb.

Látunk a felhasználókhoz, a csoportokhoz tartozó *tulajdonossági kategóriákat* (xy tulajdonosa ennek és ennek, xq csoport csoport-tulajdonosa ennek és ennek stb.). Továbbá látunk *hozzáférési kategóriákat* (olvasható ez és ez, írható, törölhető, kiegészíthető, futtatható stb.). A tulajdonosságok és hozzáférések szabályozottak lehetnek. Ezeket a viszonyokat ismernünk kell.

2.2. Számítógépes hálózatok

Az önálló számítógép hasznos, hálózatba kapcsolva azonban még hasznosabb. Hálózatosítva számítógéprendszerekről beszélhetünk, nemcsak számítógépekről.

Milyen számítógép hálózati osztályokat ismerünk? Az alábbiakban felsorolunk hálózati osztályokat:

- GAN (Global Area Network). Világra kiterjedő hálózat, nagy sávszélességgel, viszonylag nagy késleltetési időkkel jellemezhető.
- WAN (Wide Area Network). Sok ezer km-re kiterjedő; sávszélessége néhány Mbps-től néhány Gbps-ig terjedhet. Késleltetési ideje kisebb az GAN hálózatétól.
- MAN (Metropolitan Area Network). Nagyvárosra, városra kiterjedő; sávszélessége néhány 100 Mbps-től néhány Gbps-ig terjedhet.
- LAN (Local Area Network), 10 km-es nagyságrendben a kiterjedése, 10-100 Mbps, 1-2 Gbps sávszélességek a szokásosak, kis késleltetési időkkel.
- VLAN (Very Local Area Network) hálózatoknak cm-m a kiterjedése; igen nagyok a sebességek.

2.2.1. A hálózatosodás motivációi

Számítógép hálózatok már régen kialakultak. Eleinte a hálózatosodás mozgatórúgója az erőforrás megosztás és összevonás volt. Felsorolunk jellegzetes erőforrás megosztás-összetétel elemeket:

- **Data Compound/Sharing** (adat összetétel/megosztás). Igény a közös állományok használatára.

Tipikus példa az osztott adatbázisok használata (distributed DB).

- **Function Compound/Sharing** (feladat összetétel/megosztás). Igény a költséges perifériák közös használatára. Példák:
  - fájl szolgáltatás (file-server);
  - drága periféria megosztása;
  - speciális CPU igénye.
- **Aviability Compound** (lehetőség biztosítás). Igény a megbízhatóság növelésére. Pl.
  - helyettesítő erőforrások biztosítása.
- **Power Compound** (erő összevonás). Igény a teljesítmény növelésére. Tipikus példa:
  - párhuzamos feldolgozással gyorsítás.
- **Load Compound** (terhelés elosztás). Igény a teljesítmény növelésére. Tipikus példa:

- erőforrás csúcsterhelés esetére nem terhelt erőforrások bevonása.

Az erőforrás megosztás-összevonás mellet jelentős hajtóerő a számítógépes hálózatok fejlődésében a számítógépes kommunikáció (Computer Mediated Communication). Ma már ez szinte nagyobb hajtóerő, mint az erőforrás megosztás-összetétel! Igény van a kommunikációra. Az elektronikus levelezés, fájlok átvitele, a WEB böngészés és információszerzés ma már természetes igénye a művelt embernek, és ez piacot, igényt jelent a hálózatok fejlesztőinek.

2.2.2. A hálózatok összetevői

Minden számítógép hálózatban vannak számítógépek, melyek a felhasználói programokat (alkalmazásokat) futtatják. Ezeket *gazda* (host) gépeknek nevezhetjük. Lehetnek bennük *végberendezések* (terminálok), amik a kapcsolattartást segítik. Gyakori, hogy a végberendezésen magát a gazda gépet értik, ilyenkor a terminál a gazda géphez tartozó valami. Az egyes végberendezéseket *alhálózatok* kötik össze, amik *átviteli vonalakból* és *kapcsolóelemekből* tevődnek össze. A kapcsolóelemek speciális számítógépek (vagy szokásos gazdagépek, de speciális feladattal is), melyek a hozzájuk befutó vonalak logikai vagy fizikai összekapcsolását végzik. Mindenesetre, mikor hálózatba kapcsolt számítógéprendszert akarunk "igénybe venni", valójában egy terminál (vagy egy számítógép, aminek vannak terminálként viselkedő perifériái) ülünk, és annak segítségével használjuk a rendszert.

2.2.3. Számítógéprendszer igénybevétele

Leülve egy *számítógéprendszer* (valójában terminál) elé két dolgot kell csinálnunk. Néha az egyik, vagy akár mindkettő "elmarad", de éreznünk kell, hogy van ez a két dolog! A végső klasszikus célunk az, hogy ellenőrzött módon bejelentkezzünk egy gazda gépre, és annak szolgáltatásait, erőforrásait egy felhasználói felületen (parancsértelmező folyamaton, vagy grafikus interfészen át) használjuk. Az a két dolog a következő két gondból fakad:

- Melyik gazda gépre akarunk bejelentkezni? Milyen szolgáltatását igényeljük?
- Hogy lehet ellenőrzötten bejelentkezni?

Ezek után a két dolog: *kapcsolat létesítése* és *ülés* (viszony) *létesítése*.

2.2.3.1. Kapcsolat létesítése (Connection)

Célja: létesüljön vonal (kapcsolat) egy géphez (induljanak processzek, amik támogatják ezt a kapcsolatot, segítik a későbbi "ülés" (session, viszony) létesítését), hogy ezen a vonalon (ülés létesítése után) igénybe vehessük a szolgáltatást.

Az indult processzek "kezelik" a létesített *virtuális vonalat*, a kapcsolatot. A kapcsolat létesítéséhez meg kell mondani, melyik gazda géppel akarjuk létesíteni a kapcsolatot, és milyen szolgáltatást akarunk igénybe venni. A kapcsolat létesítésében fontosak a hálózat kapcsolóelemein futó processzek is, de végső soron legfontosabb a cél gazda gépen futó, a kapcsolatot fogadó és gondozó processz. Ez a processz Unix-nál: az *init* processz, illetve ennek *tty* és *login* gyermeke.

Az egyik leggyakoribb kapcsolatlétesítési indító ok a *távoli géphasználat* (távolról akarunk egy gépet kezelni a rendszer parancsértelmezőjén át). Tudomásul véve, hogy létezik más célú

kapcsolatlétesítés is, most itt elsősorban a távoli géphasználati kapcsolatépítésről (később az ugyanolyan célú ülés létesítésről) szólunk.

Hogyan kezdeményezhető kapcsolat?

a) Elszigetelt gépen, a gépre közvetlenül kapcsolt terminálon: bekapcsolással, a terminál bekapcsolásával. Ez helyi géphasználat.

Vegyük észre, a legtöbb személyi számítógéppel - azzal, hogy éppen az elé ülünk, azt választjuk ki, azt bekapcsoljuk - már meg is született a kapcsolat, nincs kifejezett kapcsolatépítés.

b) Már létező ülés (session) alól: indítunk egy terminálemulációt és abban kapcsolatfelvevő processzt. A kapcsolatfelvevő processz a mi kliens processzünk, helyben fut. Meg kell neki adni a távoli gép azonosítóját, nem szükséges (bár lehet) explicite megadni a szolgáltatás (a port és ezzel a protokoll) azonosítót: vannak ui. konvenciók a legfontosabb szolgáltatásokra (pl. a távoli géphasználatra), és a kliensünkbe "be lehet programozva", hogy a távoli gép mely portját (mely szolgáltatását) szólítsa meg.

Nézzünk néhány távoli géphasználatot kezdeményező klienst!

Pl. MS-DOS "ülésből" TCP/IP hálózati protokoll segítségével

```
> tn  host_id
```

vagy

```
> rlogint  host_id
```

Unix, VAX/VMS ülésből TCP/IP alatt a

```
> telnet  host_id
```

terminál-szerver ülésből TCP/IP alatt a

```
> connect  host_id
```

vagy

```
> telnet host_id
```

DECNET alatt

```
> set_host  host_id
```

parancsok így kiadva, távoli géphasználati kapcsolatot kezdeményeznek. Általában a sikeres kapcsolat kiépítése után a vonalon a távoli rendszert ismertető szöveg jelenik meg (milyen gép, milyen operációs rendszer stb.), majd a távoli gépre való üléslétesítéshez a login üzenet. Ekkor már élő a kapcsolat, folytatódhat rajta az üléslétesítés (megadjuk bejelentkezési nevünket, jelszavunkat).

Fontos megjegyeznünk, hogy a telnet, rsh, rlogin távoli géphasználati célú kapcsolatépítők nem biztonságosak! A telnet protokoll egyszerű, de könnyen lehallgatható, a kapcsolaton átmenő üzenetek hamisíthatók. Egyes rendszergazdák letiltják, hogy rendszerükre telnet protokollos kapcsolatot építsünk ki. Helyette az ssh (secure shell) klienssel történő kapcsolatkezdeményezést ajánlják. Ez ugyanis biztonságos, nem lehet hamisítani az üzeneteket, nem lehet lehallgatni azokat (vagy legalább is nagyon nehéz!) Javasoljuk tehát meglévő ülésből az ssh kapcsolatindítást:

```
> ssh  host_id
```

Gondunk lehet, vajon van-e a helyi rendszerünkön ssh kliens installálva. Sok Unixos rendszeren ma már van. MS Windows-os rendszerekre az ssh beszerezhető, installálható.

Nézzék a következő helyeket:

Miért használjunk ssh-t a telnet és az rsh helyett?

"Gyengéd" bevezető ismeretek ...

A kapcsolat bontása néha nem is egyszerű!

a) Kapcsolaton létesített ülés bontása, megszüntetése bontja a kapcsolatot is. Lásd tehát ott a kapcsolat bontást!

b) Előfordul, hogy a kapcsolat létrejött, de rajta az ülés nem (pl. elfelejtettük a jelszót). Ilyenkor nem működik az a) megoldás! Ilyenkor jó, ha megjegyeztük a kapcsolatlétesítő (telnet, rloginvt, ssh) ún. *menekülési* (escape*) szekvenciáját*! Ennek begépelése bontja a kapcsolatot.

2.2.3.2. Ülés létesítése kapcsolaton

Az ülés (session) létesítés célja az, hogy azonosított (ellenőrzött) módon használjuk a rendszert, akár általános, akár speciális célra.

Az ülés (session) a login - logout közötti idő. Az ülés alatt egy felhasználói kapcsolattartó rendszer segítségével kezeljük a rendszert. (Ne feledjük, most a távoli géphasználati célú kapcsolatokról és ülésekről szólunk!)

```
login: username
passwd:
os_prompt>              # ez itt a session
. . .
. . .
os_prompt> logout | EOF jel   # egeszen eddig
```

Az üléslétesítéshez ismerni kell a felhasználói nevet (username): ez a *számlaszám* (account). Tudni kell a hozzátartozó *jelszót* (password)

Vannak *nyilvános számlaszámok*, ezek jelszó nélküliek, vagy a jelszót nem az eredeti célú azonosítás ellenőrzésre, hanem pl. statisztikai célú azonosításra használják.

Szokásos nyilvános számlaszámok: *anonymus*, *guest.* A rendszergazdák ma már nem nagyon engedélyezik ezeknek használatát távoli géphasználati bejelentkezésre, az anonymus számlaszámot ftp-s kapcsolat és üléslétesítésre szokták csak engedélyezni.

A nyilvános számlaszámokkal kapcsolatban ügyelni az *etikus viselkedésre*!

Sok személyi számítógépnél az üléslétesítés is elmarad: a bekapcsolással létesült kapcsolaton ellenőrzés nélkül indul az ülésünk, valamilyen felhasználói kapcsolattartó burkot vagy GUI-t használhatunk, mert az az operációs rendszer betöltése után azonnal indul. MS DOS alatt azonnal indul a command.com burok, máris van ülésünk. Persze, mi, "beavatottak", tudjuk, ilyenkor is van kapcsolatunk és rajta ülésünk.

A géphasználat végén bontanunk kell az ülésünket.

Sok rendszerben a kapcsolattartó burok program ismeri a *logout* parancsot. Ezzel bonthatjuk az ülést, és ezzel bontjuk a kapcsolatot is. Lehetnek más ülést befejező parancsok is. Gyakori a *bye*, néha az *exit* stb. Tanuljuk meg, hol milyen parancs az ülésbontó parancs. Grafikus kapcsolattartónál találhatunk valamilyen ülést bontó nyomógombot, legördülő menüelemet, esetleg billentyű-kombinációt.

Régebbi Unix rendszerekben néha nincs is ülés bontó parancs. Ekkor kihasználhatjuk a következő tényt: a burok valójában egy processz, ami a Unix szokások szerint, ha a szabványos bemenetén fájlvég (EOF) jelet kap, befejezi futását (terminálódik). A burok szabványos bemenete - szintén szokás szerint - a kapcsolatot indító terminálra (emulált terminálra) van leképezve. Ha itt EOF jelet tudunk a billentyűzeten előállítani, az terminálja a burkot, ezzel megszünteti az ülést, bontja a kapcsolatot. Most már csak az a kérdés, hogyan produkáljunk EOF-et. Nos, ez az *stty* parancs segítségével kideríthető! (Sajnos, különböző billentyűkombináció lehet a különböző rendszerekben!)

Kezdőként általános problémájuk lehet a következő: a kapcsolatot valamilyen terminál (esetleg emulált terminál) alól indítják. Később fogunk tanulni a terminálokról, arról pl., hogy vannak különböző terminál szabványok, amik azt foglalják össze, hogyan viselkedik egy terminál (az akár "igazi", akár emulált.). A kapcsolatot fogadó gazdagép operációs rendszere, az egyes alkalmazások a kapcsolat a valós vagy virtuális vonalán valamilyen terminált képzelnek el, és az néha nem az a szabvány, mint ahonnan a kapcsolatot létesítettük. Ezekkel a gondokkal, feloldásukkal feltétlenül foglakozniuk kell majd a félév során! Mindenesetre a feloldás mindkét oldalon kereshető, lehetőleg általánosan ismert szabványú terminált használjunk (valódi termináloknál is van beállítási lehetőség sokszor), azt emuláljunk! Másrészt előbb utóbb meg kell tanulnunk, hogy az egyes operációs rendszerekben hogyan hangolhatjuk a "terminál driver"-eket, azaz a gazda oldalon hogyan tudunk alkalmazkodni a különböző terminálokhoz.

Jegyezzük meg, a klasszikus (ssh, telnet, tn, rloginvt) kapcsolat- és üléslétesítésen kívül vannak célirányos kapcsolat és üléslétesítések is! Az ftp-zés is kapcsolat és üléslétesítés, csak az ftp kapcsolaton nem a szokásos burok processz a felhasználói felület, hanem egy szolgáltató processz, aminek csakis ftp parancsokat adhatunk. Ugyanígy: gopher kliens is kapcsolatot kezdeményez egy gopher szerverrel. Rendszerint itt az üléslétesítés eliminálódik. Másik érdekesség a gopher kapcsolaton, hogy a kapcsolat csak arra az időre teremtődik meg, amíg a gopher szerver egy dokumentumot leküld a kliensnek megjelenítésre, utána a kapcsolat bom-

lik, *kíméli* a hálózatot. Hasonlóan csakis a dokumentumok transzferére létesül kapcsolat a WWW szerver és a WWW böngésző kliensek (Explorer, netscape) között.

Gyakorlatok:

Létesítsünk kapcsolatot és ülést a legkülönbözőbb rendszerekből, ahová tudunk, ahol van számlaszámunk. Minden informatikus hallgatótól elvárjuk, hogy rendszereinken, illetve az Egyetemi Számítóközpont gazda gépein (gold, silver stb.) legyen számlaszáma! Akinek még nincs, jelentkezzen a rendszegazdánál, kérjen segítséget.

Különböző számítógéprendszerek használata során ügyeljenek az etikus viselkedésre! Kérem, tanulmányozzák a laboratóriumainkban kifüggesztett szabályzatot.

Ha a kapcsolat- és üléslétesítésben van gyakorlatunk, könnyen megoldhatjuk az első évközi feladatot!

Kezdjék tanulni az *ftp* és a WWW böngésző kliensek használatát!

Tanulmányozzák a Unix *stty* parancsát.

3. A UNIX operációs rendszer használata

A Unix

- nagyon elterjedt,
- multi-tasking, multi processing,
- időosztásos,
- általános célú operációs rendszer.

Használatának megtanulását sok, kiváló könyv segítheti [lásd az ütemterv irodalomajánlatát]. Nyájas bevezetést ad az Orlando Unix iskola. A Unix rövid történetét olvashatjuk itt.

Találunk könyvet a MEK-ben is: Rideg Márton: Unix alapismertek címmel.

Szokásos parancsértelmezős kezelői (kapcsolattartói) a ***burkok*** (shell). Különféle burkokkal kezelhetjük a Unixos gépeket!

Neve	programja	szokásos promptja[1]	előnye
Bourne shell	sh	$	Mindenütt! Shell programozásra!
Bourne again shell	bash	$	Interaktív használatra!
C shell	csh	%	Mindenütt! Interaktív használatra!
TC shell	tcsh	%	Interaktív használatra!
Korn shell	ksh	$	SVR4-ben ez az ajánlott!
Superuser shellje	sh	#	

3.1. A UNIX filozófia

Build on the work of others! Sok-sok kész segédprogram, szűrő létezik, amiből építkezhetünk. Nem írunk mindent újra, hanem használjuk a kész megoldásokat.

Az ezt segítő koncepciók

1.Egy Unixhoz készült program processzként általában a szabványos bementről (stdin) olvas, a szabványos kimenetekre (stdout/stderr) ír. Így szokás programozni. A segédprogramok, szűrők mind ilyenek. A szabványos ki/bementetek általában, alapértelemben a terminál kép-

[1] Megjegyzés: a fejezetben a példákban $ promptot fogok írni, ha hangsúlyozni akarom, hogy Bourne buroknak kell adni a parancsot. A > prompttal jelzem, ha mindegy, milyen burkot használunk.

ernyőjére ill. billentyűzetére vannak hozzárendelve. Nagyon sokszor az első fájlnévként adott argumentum a szabványos bemenet.

Ezt kihasználhatjuk. Ezt a konvenciót használhattuk pl. az ülés megszüntetésére, mikor is az üléshez tartozó burok processznek fájlvég (EOF) jelet adva azt termináltuk.

Egy másik példa a következő: a Unix burkaiban tulajdonképpen nincs az MS DOS-ban megszokott *type* parancs (ami egy szövegállományt a képernyőre listáz). Használhatjuk helyette a *cat* nevű, valójában fájlok összefűzésére való (concatenation) parancsot!

Íme a példa:

```
> cat f1 f2 f3  # oszefuzo, stdout-ra irja a fajlokat
.....
> cat f1        # type helyett! Most nincs "összefűzés",
                # csak a stdout-ra íródik az f1 fájl!
....
```

Egy másik példa (itt is a hiányzó type parancsot "helyettesítjük"):

```
> more <file
```

Sőt:

```
> more file
```

2. A standard adatfolyamok átirányíthatók! (Az átirányító operátorokat később összefoglaljuk!)

Ezt persze már megszokhattuk az MS DOS-ban is. Jegyezzük meg azonban, hogy az átirányítást a DOS örökölte, az volt a későbbi! Az alábbi példában a már ismert *cat* segítségével szövegsorokat írunk az *f1* fájlba. Lássuk be, hogy itt a *cat* a szabványos bemenetről olvas egészen a fájlvég jelig, amit a CTRL/D billentyűkombinációval generáltunk. A cat szinte egy kis szövegrögzítőként viselkedi, csak azért nem szövegszerkesztő, mert ha egy sort bevittünk (a billentyűzet bufferből elküldtük), akkor az már nem javítható!

A példa:

```
> cat > f1
első sor
második sor
...
utolsó sor
CTRL/D
```

3. Csővezeték képezhető!

Ez is ismerős lehet. Ezt is tárgyaljuk később részletesebben. Mindenesetre ez a tulajdonság segíti, hogy különböző, már meglévő szűrőket használjunk a feldolgozásokban. Az alábbi példában a már ismert cat kiírná a képernyőre az f1 fájl tartalmát, de azt a grep mintakereső

szűrővel megszűrjük. Ennek eredménye sem kerül a képernyőre, mert tovább szűrjük a wc sor-szó-karakterszámlálóval. Csakis ennek az eredményei fognak kiíródni.

A példa:

```
> cat f1 | grep minta | wc
...
```

4. Az ún. *daemon* processzek szolgáltatásokat biztosítanak!

Például nyomtatási kérelmeket szolgálnak ki, a levelezést segítik stb. Sok démon futhat a használt Unix rendszerben, ezek szolgáltatásait kihasználhatjuk.

3.2. Honnan tanulhatjuk a UNIX használatot?

- Könyvekből.
- Kézikönyvekből, dokumentációkból.
- Segédletekből.
- Az *on-line manual*ból.
- Saját jegyzeteinkből, társainktól.
- A WWW lapjaiból (Orlando iskola, shell összefoglaló stb).

Az on-line kézikönyv, a *man*

A *man* parancs megjeleníti a kézikönyv (on-line manual) lapjait.

A kézikönyvekben tömörítve, formátumozó direktívákkal tárolt dokumentumok vannak. A man parancs több szűrőn (végül a *more* szűrőn) keresztül jeleníti meg a dokumentumokat, a bejegyzéseket.

A *more* legfontosabb parancsai:

- *space*: lapot dob,
- *Return:* sort dob,
- *q*: - kilép (quit).

A *man* hívásának szintaxisa

```
> man [ -opciok] [section] bejegyzes
```

Ha bővebben meg akarunk ismerkedni a man-nal, hívjuk a man-t a man bejegyzésre!

```
> man man
```

Ha pl. a parancsokat, azok szintaktikáját és szemantikáját akarjuk megismerni, miután minden parancsról van bejegyzés, ismerkedhetünk a parancsokkal a *man* segítségével. Sajnos, a bejegyzésekhez a klasszikus *man*-ban nincsenek "dzsókerek"! Ebből következően tudni kell a pontos bejegyzés neveket!

Segít, ahol van:

az *apropos* adatbázis,

a whatis,

a whereis.

Segít az X11 GUI felületén a szebb formátumú, kezelhetőbb *xman*. Segít a gold-on az X11-es környezetű *info* (hypertext ismertető). Segít SGI-n a GL-es környezetű *insight* (könyvespolc). Segít SUNOS, SOLARIS környezetben az *Answerbook*.

Tanácsok

Miután a kézikönyv lapjai angol szövegek,

- tudni kell angolul olvasni.
- A fontos parancsok nevét pontosan tanuljuk meg. Használjunk parancs-kártyát, készítsünk jegyzeteket,
- A man lapok végén az utalások vannak kapcsolatos lapokra. Nézzük ezeket is!

3.3. Fontos parancsok, csoportosítva

A következőkben felsorolok fontos parancsokat, megemlítve a parancs nevét és rövid feladatát, jellemzőjét. A csoportosítás is segíthet egy-egy parancs pontos nevének, feladatának megtalálásában.

3.3.1. Manipulációk fájlokon, jegyzékeken

1. Editorok, szövegszerkesztők

ed	sororientált
vi (vim)	képernyő-orientált
e (emacs)	képernyő-orientált
pico	egyszerű, sok rendszeren

2. "Kiírók"

cat	concatenál, stdout-ra
pr	printel, stdout-ra
head	fájl első sorait, stdout-ra
tail	fájl utolsó sorait
more	lapokra tördelő szűrő
od	oktális dump (ömlesztés)

3. Jegyzékekkel kapcsolatos parancsok

ls	jegyzék tartalom lista (dir helyett)
mkdir	jegyzék készítés
rmdir	jegyzék törlés
cd	munkajegyzék váltás
pwd	munkajegyzék lekérdezés
chmod	fájl védelmi maszk váltás (nemcsak jegyzékre)
chown	fájl tulajdonos váltás (nemcsak jegyzékre)
file	fájl típus lekérdezés (nemcsak jegyzékre)

4. Másolások, mozgatások

cp	copy, másolás
mv	move, mozgatás (rename helyett is!)
ln (link)	"linkel"
rm (unlink)	"linket" töröl, remove: fájl törlés
find	keres fájlt egy fán és csinál is valamit (bonyolult, de nagyon hasznos!)

3.3.2. Állapotok (státusok), információk lekérdezése, beállítása

ps	processzek listázása
file, ls, pwd	ld. fönn
date	dátum, idő lekérdezés
who, w, rwho, rusers	ki van bejelentkezve?
rup	mely rendszerek élnek?
top, gr_top	erőforrás-használat csúcsok
osview, gr_osview	erőforrás-használat
last	utolsó bejelentkezések
finger	ki kicsoda?
passwd, yppasswd	jelszóállítás
chsh, chfn, ypchpass	név, induló burok stb. beállítás
ypcat	NIS (yellow pages) adatbázis lekérdezés
xhost	X11 munka engedélyezése
set	környezet (environment) lekérdezése
du, df	diszk használat

3.3.3. Processz indítás, vezérlés

sh, bash, csh, ksh, tcsh	shell idítás
exec	processz indítás
kill	processz "megölése", szignálküldés
sleep	processz altatása
wait	processz várakoztatás
at	processz indítása egy adott időpontban
nohup	kilépéskor ne ölje meg
test	kifejezés tesztelése
expr	kifejezés kiértékeltetése
if, case, for, do while	vezérlő szerkezetek
break, continue	vezérlő szerkezetek
echo	argumentumai echoja (meglepően hasznos valami)

3.3.4. Kommunikáció a világgal, felhasználókkal

ssh, telnet, rlogin, rsh	kapcsolatlétesítés,
rwho, rusers, finger	lásd fönt
write	üzenet konzolokra
talk, xtalk	interaktív "beszélgetés"
mail, Mail, pine, zmail	elektronikus levelezés kliense
ftp	fájl átvitel kliense
lynx, netscape, mozilla	WWW böngésző (kliens)

3.3.5. Hasznos szűrők

grep	mintakereső
awk, nawk	mintakereső feldolgozó
wc	sor, szó, karakterszámláló
sed	áradatszerkesztő
head, tail	ld. fönn
cut	mezőkivágó

3.3.6. Parancsok a tanuláshoz

man	laplekérdezés a kézikönyvből
apropos	kézikönyvben kulcsszó

whereis	hol van egy parancs
whatis	man lap leírás
xman	X11-es kézikönyv

stb.

Egy kis segítség a DOS-ból UNIX-ba áttérőknek.

3.4. A Bourne shell (sh)

A shell (burok) szót meghallva, kétféle értelmezésre kell gondolnunk. Hogy melyik értelemben használjuk a burok szót, az a szövegkörnyezetből derül ki.

A burok (shell) egy parancsértelmező processz

Tehát egy futó program. Van azonosítója (pid), ami lekérdezhető. Készíthet gyermek processzeket. A feladata:

- *készenléti jelet* (prompt) ad, ami azt jelzi, a szabványos bemeneti csatornán képes beolvasni parancsot (csövet, listát);
- *parancsot, csövet, listát* elfogad, elemez, esetleg átalakításokat végez, behelyettesít, végrehajt.

A shell egy programnyelv

Mint programnyelv,

- van vezérlési szerkezete;
- vannak (egyszerű) adatszerkezetei, változói.

Szövegszerkesztővel írhatunk ún. burok programokat (shell-szkripteket), később ezeket "odaadhatjuk" egy shell parancsértelmezőnek, hogy azt dolgozza fel.

3.5. Az sh burok, mint parancsértelmező

Tárgyalásához meg kell tanulnunk néhány alapfogalmat.

3.5.1 Alapfogalmak

3.5.1.1. A parancs fogalma

A *parancs* "fehér"[2] karakterekkel határolt szavak sora. A sorban az első szó a parancs neve, a többi szó a parancs argumentumai (általában opciók és módosítók, fájlnevek gazdagép és felhasználó azonosítók stb.).

---

[2] Fehér karakterek: a szóköz, tabulátor, sorvég karakterek.

A parancsot a burok beolvassa, elemzi, átalakítja és végrehajtja (a parancsnak megfelelően csinál/csináltat valamit). A parancs vagy külön processzben fut (a burok gyermek processzeként, szeparált processzként), vagy végrehajtja maga a burok (ekkor nem készül gyermek processz).

A parancsnak, akár külön processzben fut, akár maga a burok hajtja végre,

- van visszatérési értéke!

A visszatérési értéke lehet

  - normális (0) visszatérés,
  - nem normális (nem 0) visszatérés.

A visszatérési értéket a burok használhatja a vezérlés menetének szabályozására.

A burok processznek (ami a parancsot végrehajtja) van legalább három *nyitott adatfolyama*.

Leírójuk	Nevük	Szokásos leképzésük
0	stdin	billentyűzet
1	stdout	képernyő, ablak
2	stderr	képernyő, ablak

Láttuk, a parancsban *szavak* vannak. A ***szó*** az, amit *fehér karakter* határol. Idézőjelbe (" " ' ') tett szöveglánc (quótázott szöveglánc) csak egy szónak számít (az idézőjel semlegesíti a fehér karakterek szóhatároló funkcióját).

Ügyelni kell a speciális karakterekre! Ezek szerepe különleges! (Ilyenek a * $ [ ] { } \ . stb. karakterek).

Egy példa parancsra:

```
> find . -name a.c -print
   0   1     2    3    4
```

azaz, a fenti parancs 5 szóból áll.

A parancsokban a parancsértelmező *adatfolyamai* átirányíthatók. Ekkor a parancs – ha nem kellene is - szeparált processzben fut. Miért? Mert az indító shell processz szabványos adatfolyamainak leképzését nem változtatják (hibalehetőségekhez vezetne). Ilyenkor új processzt készítenek, és abban végzik az új leképzést.

Példa adatfolyam átirányításra:

```
> ls  >mylist.txt
```

Ebben a parancsban az átirányítás miatt az *ls* szeparált processzben fut. Az *ls* a burkokban rendszerint belső parancs, nem kellene neki feltétlenül szeparált processz. Kérdezhetnénk, milyen *program* fut ekkor a szeparált processzben? ls program nincs, hiszen az az sh/bash/tcsh/ksh burkok belső parancsa? Nos, a válasz: a gyermek processzben is a burok fut, ennél viszont a szabványos (standard) kimenet a *mylist.txt* fájlba van leképezve, és ez a gyermek burok processz fogja az *ls*-t végrehajtani!

3.5.1.2 A csővezeték (pipe) fogalma

A *csővezeték* parancsok sora | -vel (cső operátorral) szeparálva. A | a csővezeték operátor.

A csővezeték szintaxisa:

```
> parancs_bal   |   parancs_jobb
```

A szemantikája:

Végrehajtódik a *parancs_bal* és szabványos kimenete leképződik az utána végrehajtódó *parancs_jobb* szabványos kimenetére.

A csővezeték parancsai szeparált processzekben futnak Miért? Mert itt is szükséges a szabványos adatfolyamok leképzésének megváltoztatása!

A *csővezetéknek is van visszatérési értéke*: a parancs_jobb visszatérési értéke.

A parancs degenerált csővezeték. Ezentúl, ha valahol csővezetéket írunk, oda parancsot is írhatnánk.

Példa:

```
> cat /etc/passwd   |   grep  valaki
```

Az *cat* itt a számlaszámokat tartalmazó állományt teszi a szabványos kimenetére, a csővezetékbe. Ezt „megszűrjük” a *grep* mintakereső szűrővel, keresve a *valaki* mintát tartalmazó sort. A grep a csővezetékből olvas: arra képzi szabványos bemenetét.

3.5.1.3.A parancslista fogalma

A *parancslista* csővezetékek sora, szeparálva a következő operátorokkal:

```
   &&    ||     # magasabb precedencia
    &   ; \n    # alacsonyabb precedencia
```

A parancslista operátorainak precedenciája alacsonyabb, mint a csővezeték | -jé!

A *parancslista* szintaxisa:

```
> csőbal  listaoperátor  csőjobb
```

A szemantika:

;	soros végrehajtása a csöveknek
&	aszinkron végrehajtása a csőbal-nak (ez a háttérben fut, és azonnal indul a csőjobb is, vagy visszatér az indító shell)
||	csak akkor folytatja a listát, ha csőbal nem normális visszatérési értékű
&&	csak akkor folytatja a listát, ha csőbal normális visszatérési értékű

Először látjuk a visszatérési érték értelmét, az valóban megszabhatja a „vezérlés menetét”!

A csővezeték degenerált lista. Ezentúl, ha valahová parancslistát írunk, az lehet csővezeték, sőt parancs is!

A *parancslista* visszatérési értéke az utolsó csővezeték visszatérési értéke. Háttérben futó csővezeték visszatérési értéke külön kezelhető.

3.5.2. Parancs, cső, lista csoportosítás, zárójelezés

A csoportosítás, zárójelezés oka kettős lehet:

az operátorok precedenciájának átértékelését akarjuk elérni;

processz szeparálást akarunk elérni.

Lehetséges zárójelek: ( ) { }

A szintaxis:

```
{ lista }   vagy   ( lista )
```

3.5.2.1. Zárójelezés a precedencia átértékelés miatt

Emlékezzünk a csővezeték operátor és a listaoperátorok precedencia sorrendjére. Ezt a precedenciát tudjuk zárójelezéssel átértékeltetni.

Beláthatjuk, hogy az alábbi példákban eltérő eredményeket kapunk! Emlékeztetek arra, hogy a *date* parancs dátumot és időt ír a szabványos kimenetre, a *who* parancs a bejelentkezettek listáját teszi a kimenetre, a *wc* parancs pedig sor-, szó- és karakterszámláló.

Példa:

```
$
$ date ; who | wc        # mast ad ez
...
$ ( date ; who ) | wc    # mint ez
...
```

Házi feladatként magyarázzák meg, miért ad mást a két lista!

3.5.2.2. A processz szeparálás miatti zárójelezés

{ *lista* } zárójelezéssel csoportosított parancsnál, - hacsak más ok miatt (pl. átirányítás van, csővezeték van, külső parancsot kell végrehajtani) nem kell szeparált processzben végrehajtani - ugyanabban a processzben fut a lista.

(*lista*) zárójelezéssel a parancslista mindenképp szeparált processzben fut!

Megpróbálom megmagyarázni példákkal. A megértéshez érteni kellene a *processz környezet* (process environment) fogalmat, amit később részletezünk. Mindenesetre a környezethez tartozó információ a *munkajegyzék* (working directory). A pillanatnyi munkajegyzék lekérdezhető a *pwd* paranccsal, munkajegyzék váltható a *cd* paranccsal. Az *rm* parancs fájltörlésre való. A két példa ugyanabból a kiinduló helyzetből induljon; munkajegyzék a *vhol*, ebben be van jegyezve az *ide* jegyzék, utóbbiban van *junk* fájl.

1. példa:

```
$ pwd                   # hol vagyunk?
vhol
$ cd ide ; rm junk      # törli vhol/ide/junk-ot
$ pwd
vhol/ide                # most ez a munkajegyzék
$
```

2. példa:

```
$ pwd
vhol                    # hol vagyunk?
$ ( cd ide ; rm junk )  # u.azt torli
$ pwd                   # mivel szeparalt processzben
vhol                    # futott, a cd csak
                        # „ideiglenes” volt.
```

3.5.3. A parancs végrehajtás

Általában az *sh burok* készít új processzt a parancs számára, ebbe betölti a parancshoz tartozó végrehajtható fájlt, átadja az argumentumokat az így készült processznek. Ez az általános szabály, ami alól vannak kivételek.

Nem készül új processz az ún.

- *belső parancsoknak* (special commands, built in commands),

- a *vezérlő parancsoknak* (*for, while, case* stb.),
- a *definiált függvényeknek* (sh makróknak),

de a kivételeknek is vannak kivételei:

- hacsak nem zárójeleztünk ( ) -vel,
- hacsak nincs átirányítás, csővezeték (> >> < << |) .

Biztos készül új processz a *külső parancsoknak*. Ezek lehetnek:

- V*égrehajtható* (compilált, linkelt executable) *fájlok* . (A burok ezeket a fork/exec villával indítja. Az argumentumok itt is átadódnak! Lásd a C-ben a main függvény argumentum-átvételét!)
- *Burok programok* (shell eljárások, shell szkriptek). A burok ekkor is a fork/exec villával indít szeparált processzt, ebbe burkot tölt és ennek adja a burokprogramot feldolgozásra.) (az argumentumok átadódnak!)

Mind a végrehajtható fájlok, mind a burokprogramok futtatására jellemző:

- PATH szerinti keresés,
- kell hozzájuk az x (executable) elérési mód,
- a burokprogramokra kell az r (readable) elérési mód is,
- a gyermek processz örökli a környezetet (environment, lásd később).

Külön érdekesség: vajon milyen végrehajtható fájl fut a szeparált processzben, ha belső parancsot indítunk, de kikényszerítjük (vagy kikényszerül), hogy mégis szeparált processzben fusson? Nos a válasz erre: akkor a gyermek processzben is a burok fut!

3.5.4. Az adatfolyamok átirányítása

Fontos szereplőek az 0/1/2 leírókkal azonosított szabványos adatfolyamok. Ahogy említettük, a parancsok általában az *stdin*-ről olvasnak, az *stdout*/*stderr*-re írnak.

Mielőtt a parancs végrehajtódik, a végrehajtó shell megnézi, van-e átirányítás a parancs sorában.

Ehhez a szavakban

```
<    >    <<-vmi    >>
```

átirányító operátorokat használhatjuk. Ha ilyen operátorokat talál a burok, akkor - szeparált processz(eke)t készítve, azokban leképezve az adatfolyamokat futtatja a parancsot.

Az átirányító operátorok szemantikája:

```
< file          # file legyen az stdin
> file          # file legyen az stdout (rewrite)
>> file         # file legyen az stdout (append)
<<[-]eddig      # here document: beagyazott input
```

A beágyazott inputnál a - elmaradhat, ezt jelzi a szintaxishoz nem tartozó [ ] zárójelpár.

Az átirányítás szintaxisát és szemantikáját lásd bővebben az on-line kézikönyv *sh* lapján! Az *append* hozzáfűzést, a *rewrite* újraírást jelent.

Legnehezebb megérteni a *beágyazott input* fogalmat. A burokprogramokban parancsokat, csöveket, listákat szoktunk írni, néha azonban jó lenne a feldolgozandó *adatokat* is oda írni. Jelezni kellene azonban, hogy ezek nem parancsok, hanem feldolgozandó adatok. A végrehajtó burok ugyanarról a szabványos bemeneti csatornáról (a szkriptből) kell, hogy olvassa ezeket is, mint a parancsokat! Vagyis a bemeneti csatornát akarjuk leképezni magára a burokprogramra, annak a soron következő soraira. Persze, azt is kell jelezni, hogy meddig tartanak az adatok, hol kezdődnek újra a parancsok! Nos, ezt a problémát oldja meg a beágyazott input, adatok beágyazása a burokprogramba.

Egy kis példa a beágyazott inputra, ahol is létezik az *a.script* szövegfájl, (futtatható és olvasható,) a tartalma az alábbi:

```
a.script
-------------------------------
grep  ezt  <<!
elso sorban van ezt
2. sor, ebben nincs
3. sor
!
echo ' na mi van?'
-------------------------------
```

Így indíthatjuk, és az alábbi az eredmény:

```
$ a.script
elso sorban van ezt
 na mi van?
$
```

A fenti példában a ! (felkiáltójel) használatos az adatsorok végének jelzésére. Olyan karakterkombinációt válasszunk, ami nincs az adatsorok között, hiszen ez fogja jelezni, meddig tartottak az adatok, hol kezdődnek újra a parancsok.

3.5.5. Fájlnév kifejtés

A parancsok argumentumai gyakran fájlnevek. Ezekre van "behelyettesítési" lehetőség, alkalmazhatunk *dzsókereket*.

Parancsok argumentumaként, argumentumaiban használhatunk ún. fájlbehelyettesítési dzsóker *karaktereket*. Ilyenek a kérdőjel (?), a csillag (*), a szögletes zárójelek [ ].

Ha ezek előfordulnak a parancs szavaiban (általában ott, ahol a burok fájl nevet várna), akkor a *szót* (amiben szerepelnek), *minta*ként (pattern) veszi a burok! A *minta* a hívó shellben behelyettesítődik (kifejtődik)

alfabetikus sorrendű fájl nevek listájává, olyan nevekre, melyek illeszkednek a fájlnév-térben a mintára

A fájlnév-teret a hierarchikus fájlrendszer ösvénynevei alkotják, beleértve az abszolút és a relatív ösvényneveket is.

Az illeszkedés szabályaiból néhányat felsorolunk:

- A nem dzsóker karakterek önmagukra illeszkednek
- A ? bármely, egyetlen karakterre illeszkedik
- A * tetszőleges számú és tetszőleges karakterre illeszkedik
- A szögletes zárójelbe írt karaktersorozat [...] illeszkedik egy, valamelyik bezárt karakterre (a pontok helyére képzeljünk karaktereket.
- A [!.] illeszkedik bármely, kivéve a ! utáni karakterre.

További érdekes minta szintaktika van! Érdemes utánanézni!!

Példa:

Tegyük fel, az aktuális jegyzékben van 4 fájl, a nevük:

```
a   abc   abc.d   xyz
```

Ekkor (a -> itt azt jelzi, mivé helyettesítődik az eredeti parancs):

```
$ ls *          -> ls a abc abc.d xyz
$ ls a*         -> ls a abc abc.d
$ ls [a]??      -> ls abc
$ ls [!a]??     -> ls xyz
```

Vegyük észre, hogy a fenti példa soraiban a fájlnév behelyettesítés megtörténik, és csak utána hívódik az *ls* parancs! Vagyis az sh burok nagyban különbözik az MS DOS parancsértelmezőjétől, bár ott is használhatók dzsókerek, de azokat a command.com nem helyettesíti be, hanem átadja a parancsnak, és az, ha tudja, majd behelyettesít.

Fájlnév behelyettesítés történik ott is, ahol tulajdonképpen nem fájlneveket várnánk, pl. az echo argumentumában! Ezért pl. az előző példa aktuális jegyzékét feltételezve a következő parancs

```
$ echo [a]??
abc
$
```

eredményt adja., miután az sh burok előbb fájlneve(ke)t helyettesít be, aztán hívja az *echo*-t.

3.5.6. A metakarakterek semlegesítése, az ún. quótázás

Láttuk a fájl-behelyettesítés dzsóker karaktereit, és tudjuk, hogy további metakarakterek is vannak (a fehér karakterek, a cső és lista operátorok, a változóbehelyettesítés operátora stb.).

```
      ;      &      ( )      ^      < >      $      space     |
stb.
```

Némelyiknek tudjuk a szerepét (pl. szeparátorok, operátorok), némelyiket később tanulhatjuk meg. Látni fogjuk, némelyiknek több szerepe is lehet.

Mindezeket a burok különlegesen kezeli (pl. fájlnév behelyettesítéshez mintaként a *-ot, a space karaktert szóelválasztóként stb..

Ha mégis szükségünk van rájuk: *semlegesítsük* (*quótázzuk) őket*!

- Egyetlen karakter quótázása
  `\spec_karakter`
- Több karakter quótázása:

```
'karaktersorozat' # Minden bezárt karakter quotázott, kivéve '
"karaktersorozat" # Ezen belül a változó/paraméter és
                  # parancsbehelyettesítés megtörténik
                  # (lásd később, most csak jegyezd meg!),
                  # de a fájlnév behelyettesítés nem!
                  # Ha mindenképp kell, a \ quotázással
                  # semlegesítsd a \ ` " $ karaktereket!
```

Példa (előlegezett a *burokváltozó* és a *változóbehelyettesítés* fogalma):

```
# a - sh változó, $a - kifejtése
$ a=abc        # értéket kap az a
$ echo  '$a'   # semlegesítve a $ kifejtő operátor
$a
$ echo  "$a"   # hatásos a $ operátor
abc
$ echo  "$\a"  # a \a-val az a karaktert értjük
$\a
$
```

Próbáljuk megérteni, megmagyarázni a fentieket!

3.6. Burokprogramozás

A *burok program* (shell szkript) szövegszerkesztővel készült fájl. Egy program, ami parancsokat tartalmaz soraiban (esetleg a beágyazott input szerkezetben adatsorokat is). A burok programot a parancsértelmező processz olvassa sorról-sorra, elemezi a sorokat és sorról-sorra végrehajtja/végrehajtatja a program parancsait.

A burokprogramozásnak meglehetősen szigorú a szintaktikus szabályai vannak.

Egy nagyon egyszerű példa, melyben az a.script szövegfájl 2 sort tartalmaz:

```
a.script
---------------
who  > kik
ps  >> kik
---------------
```

A szövegszerkesztővel készített burok-programot végrehajthatóvá és olvashatóvá kell tenni! Utóbbi az szövegszerkesztők (editorok) kimenetének alapértelmezése szokott lenni, előbbit explicite írjuk elő!

```
> chmod  +x  a.script
```

Ezután hívható:

```
> a.script      # magyarazd, mi tortenik
```

Figyelem! Az előadáson részletesebben tárgyaljuk, hogy milyen kivételes esetben elegendő a burokprogram csakis olvashatósági elérése.

Ha a *shell szkript* program, akkor

- vannak (egyszerű) adatszerkezetei (változók, konstansok: szöveglánc jellegűek, de néha numerikus adatként is kezelődnek)
- van (egyszerű) végrehajtási szerkezete (soros, elágazás, hurok);
- kommentározzuk (a # után a sor maradéka kommentár).

3.6.1. A shell változók

- van nevük,
- vehetnek fel értékeket (szövegeket),
- kifejthetők a pillanatnyi értékei.

3.6.2. A shell változók osztályai

3.6.2.1. Pozícionális változók (parancs argumentumok)

A nevük kötött, rendre a 0 1 2 ... 9

Rendre a parancssor 0., 1., 2. stb. aktuális argumentumát veszik fel értékként. A 0 nevű változó mindig a parancs nevét kapja, az 1-es nevű az első szót s.í.t. Annyi pozícionális változó definiálódik, amennyi argumentum van a parancs sorában, maximum persze 9. Ha több mint 9 aktuális argumentummal hívjuk a parancsot, a shift paranccsal a változók „eltolhatók”!

Példa:

```
> script  alfa  beta     # ez a szkript hivasa
```

Ekkor a script-en belül

```
   0     ->     script     értékű
   1     ->     alfa       értékű
   2     ->     beta       értékű
   3 - 9                   nincs definiálva.
```

3.6.2.2. Kulcsszós shell változók

a) Felhasználó által definiált kulcsszós változók

A felhasználó általi definícióval a felhasználó választja ki a változó nevét. A definiálás szintaxisa:

```
valtozo=string
```

Vigyázz!

```
valtozo  =  string    # nem jo! Miert?
                      # Mert a space szóelválasztó karakter!
```

A *string* lehet 0 hosszú is! Ekkor a változó ugyan definiált, de 0 hosszúságú.

b) A rendszergazda és a shell által definiált kulcsszós shell változók

b1) Rendszergazda által definiált változók

A rendszergazda által definiált változók rendszerint konvencionális nevűek. Szintén konvenció, hogy ezek nagybetűsek. Ilyenek pl. a

```
    PATH
    HOME
    MAIL    stb., változók.
```

Valahol a rendszergazda "leírta" a definíciót és a változót „exportálta”, azaz láthatóságát kiterjesztette (rendszerint egy startup fájlban): `PATH=string ; export  PATH`

A rendszergazda által definiált változókat a segédprogramok, szűrők stb. használják, nevük ezek miatt konvencionális.

b2) A shell által definiált változók

Ezek neveit a shell programozók választották, nevük ezért kötött, konvencionális. Ilyen nevek pl. a #, a * (meglepő. a – shell változó is, nemcsak dzsóker!). A változókifejtés alfejezetben további shell által definiált változókat (és pillanatnyi értéküket is) adunk meg.

3.6.3. Hivatkozások shell változókra, kifejtésük

A legegyszerűbb hivatkozás, a parancs-sorba írt

```
$valtozonev
```

Itt a $ a kifejtő operátor. Jegyezzük meg, hogy a nem definiált, vagy 0 hosszú változó hibajelzés nélkül „kifejtődik”, természetesen 0 sztringgé. Később láthatjuk, hogy a definiálás hiánya, vagy a 0 sztring definíció „ellenőrizhető”.

Példákon bemutatunk néhány előre definiált változót (pozícionális változót, shell által definiáltat, rendszergazda által definiáltat) a kifejtésükkel:

```
$0   a parancsnév
$1   az első aktuális argumentum
$9   a kilencedik argumentum kifejtve
$*   minden definiált argumentum kifejtve
$#   a pozicionális paraméterek száma decimálisan
$?   az utolsó parancs exit státusa (visszatérési értéke)
$$   a processz azonosítója: a pid értéke
$!   az utolsó háttérben futó processz pid-je
$HOME  a bejelentkezési katalógus  stb.
```

A változóbehelyettesítés teljes szintaktikája, szemantikája

`${valt}` szerkezet is egyszerű behelyettesítés. A kapcsos zárójelek hozzátartoznak a szintaktikához. A {valt} bármikor használható, de csak akkor kell feltétlenül, ha az egyértelműséghez a változónév pontos elválasztása szükséges. Ha a változónév „folytatódik” szöveggel, akkor jelentkezhet az egyértelműsítési igény:

```
$ nagy=kis
$echo  ${nagy}kutya
kiskutya
$ echo $nagykutya


# Miert?  Mert nem definialt a nagykutya valtozo!
```

A változókifejtés teljes szabályrendszere:

Az alábbiakban a

valt shell változó
szo szövegkifejezés (pl. szöveg-konstans)
: A kettőspont (colon) önmaga, de elmaradhat.

${valt:-szo}	Ha *valt* definiált és nem 0 string, akkor kifejtődik pillanatnyi értéke, különben kifejtődik a *szo*.
${valt:=szo}	Ha *valt* nem definiált vagy 0 string, akkor felveszi a *szo*-t, különben nem veszi fel. Ezután kifejtődik a *valt*.
${valt:?szo}	Ha a *valt* definiált és nem 0 string, akkor kifejtődik, különben kiíródik a *szo* és exitál a shell. A *szo* hiányozhat, ilyenkor default üzenet íródik ki.
${*valt*:+szo}	Ha a *valt* definiált és nem 0 string, akkor behelyettesítődik a *szo* (nem a *valt*!), különben semmi sem fejtődik ki.

Ha a : (colon) hiányzik, csak az ellenőrződik le, vajon a definiált-e a *valt*.

3.6.4. Parancs behelyettesítés

A parancsbehelyettesítés szintaxisa: (vegyük észre, hogy a ` grave accent, más mint a '):

```
`parancs`
```

A szemantikája: végrehajtódik a *parancs*, és amit a szabványos kimenetre (stdout) írna, az oda, ahova a `parancs` szerkezetet írtuk, behelyettesítődik (kifejtődik). Használhatjuk a kifej-tett füzért. burokváltozóhoz érték adásra, de más célra is.

Példa:

```
$ valt=`pwd`
$ echo $valt
/home/student/kovacs
$
```

Jegyezzük meg!

Minden adat füzér (string) jellegű. A füzérben lehetnek fehér karakterek is, ilyenkor quázi szavak vannak benne!

Példa:

```
$ szamharmas=`who | wc`
$ echo  $szamharmas
3 15 11
$
```

3.6.5. Változók érvényessége (scope-ja)

A processzeknek van *környezetük* (environment), amit megkülönböztetünk a *process context*-től. (A processz kontextus fogalmat az Operációs rendszerek tárgyban részletezzük.)

A környezet (environment) szerkezete, implementációja

A környezet a processz (itt a shell) kontextusához tartozó *szövegsorokból álló* tábla.

Egy sor ebben

```
 valt=string
```

alakú.

Mikor egy shell indul, végigolvassa a környezetét, és definálja magának azokat a változókat, melyeket a környezetben megtalál, olyan értékkel, amit ott talál. Ugyanennek a shellnek aztán további definíciók is adhatók: sőt, a környezetből az induláskor definiált változók átdefiniálhatók, meg is szüntethetők. Környezeti változó átdefiniálása nemcsak az aktuális shellnek, hanem a környezetnek is szól.

A környezet lekérdezhető a *set* paranccsal.

```
 $ set
...
```

A környezetbe tehető egy változó az export paranccsal. Ezzel tulajdonképpen a leszármazott processzekben (shellekben) is láthatóvá tesszük a változókat.

Szintaxis:

```
 $ export valt
```

(Ezzel a technikával "öröklődik" a HOME, MAIL, PATH stb. Ki definiálta és exportálta ezeket? Az ülés létesítés (login) és az ülésben a burok (shell) indítás során végrehajtott *startup shell* programok!)

Kérdés merülhet fel: mi történik, ha még nem definiált változót exportálok? Vajon ekkor definiálttá válik? Válasz: nem! Ha (újra)definiálom, marad exportált? Válasz: igen!

Jegyezzük meg!

1. Exportálással csakis a gyermek (és unoka) processzek öröklik a változókat! A szülő processzek nem látják a gyermekei exportált változóit!

2. Nem exportált, de definiált változó a gyermek processzekben nem látható. Visszatérve arra a burokra, amiben definiálták: újra látható! Miért? Mert a szülő processz átélte a gyermekei életét.

Pozícionális paraméterek láthatósága

A pozicionális változók csak abban a burokban láthatók, ahová adódnak. A gyermek processzeknek új pozícionális paraméterek adódnak át.

3.7. Vezérlési szerkezetek a sh shellben

3.7.1. Szekvenciális szerkezet

Szekvenciális, legegyszerűbb vezérlési szerkezetek a parancslisták.

3.7.2. Elágazás: az if

Szintaxis (a [ ] szögletes zárójel itt nem része a szintaktikának, csak azt jelzi, hogy elmaradhat a bezárt rész):

```
 if list1 then
        list2
[elif list3 then
        list4]
[else list5]
fi
```

Szemantika:

Az if, elif predikátuma a *list1*, ill. *list3* visszatérési értéke. A predikátum igaz, ha a visszatérési érték 0, azaz normális. (Lám, láthatjuk már a visszatérési érték értelmét!)

Értelemszerűen: ha *list1* igaz, akkor végrehajtódik *list2*, különben ha *list3* igaz, akkor *list4*, máskülönben *list5*.

Vegyük észre az új neveket (kulcsszavakat: if, then, elif, else, fi). Külön érdekes a fi lezáró!

3.7.3. Elágazás: a case

Szintaxis:

```
 case  szo  in
         pattern1 ) list1 ;;
        [pattern2 ) list2 ;;]
        ...
esac
```

Ahol:

```
 * )                       akármi, default
pattern | pattern )        alternatíva
[patt patt] )              alternatíva]
stb.,
```

Vagyis a mintaképzés hasonló a fájl-név generáció mintaképzéséhez.

Az alternatívákra adok két példát:

```
 -x|-y          vagy -x, vagy -y
-[xy]           vagy -x, vagy -y
```

Figyelem! A [ ] a szintaxis harmadik sorában annak jele, hogy valamilyen szerkezet elmaradhat, a mintákban a [ ] viszont hozzátartozik a szintaktikához!

Vegyük észre az új neveket, az érdekes case lezárót! Vegyük észre a mintát lezáró) zárójelet és a listákat lezáró ;; jelpárt!

Az értelmezését legáltalánosabban a következő: kifejtődik a *szo* és összevetődik a mintákkal, olyan sorrendben, ahogy azok le vannak írva.. Ha egyezés van, végrehajtódik a mintához tartozó lista, és befejeződik a case. Egy példán keresztül bemutatom ezt. Képzeljük el, hogy van egy *append* nevű shell programunk:

```
 append
-----------------------------------------
case  $#  in     # a # a poz. param. szama
        1 ) cat  >> $1 ;;
        2 ) cat  >> $2 ;;
        * ) echo 'usage: append from to'
esac
-----------------------------------------
```

Ezt hívhatjuk:

```
 $ append
usage: append from to
$ append f1
...
...
CTRL/D
$ # elkeszult az f1
$ append f1 f2  # f2-hoz fuzodott az f1
```

3.7.4. Ciklus: a for

Szintaxis:

```
for  valt  [in szolista...]
   do
      lista
   done
```

Ahol *szolista...*: szavak fehér karakterekkel elválasztott listája, ami el is maradhat (jelzi ezt a [ ] szintaktikához nem tartozó zárójelpár).

Hiányzó in *szolista* esetén a pozícionális paraméterek szólistája az alapértelmezés (ugyanaz, mintha in $*-t írtunk volna!).

Új neveket jegyezhetünk meg, köztük a do-done parancszárójel párt!

Szemantika: a *valt* rendre felveszi a *szolista* elemeit, és mindegyik értékkel végrehajtódik a do és done zárójelpár közötti *lista* (azaz annyiszor hajtódik végre a test, ahány eleme van a szólistának).

Példa:

```
 tel
----------------------------------------------------------
for i in $*
      do
        grep  Si ${HOME}/telnotes
      done
----------------------------------------------------------
```

Magyarázzák meg, mi történik, ha így hívom:

```
$ tel  kiss nagy kovacs
```

3.7.5. Ciklus: a while

Szintaxis:

```
while  lista1
    do
       lista2
    done
```

Szemantika: ha *lista1* exit státusa 0 (normális visszatérési értékű, ami azt jelent, igaz), akkor ismételten végrehajtja do done zárójelpárral közrezárt *lista2*-t, majd újra a *lista1* végrehajtása következik s.í.t.

3.7.6. Az if-hez, while-hoz jó dolog a test

A test parancs szemantikai igaz tesztelésre normális visszatérési értéket ad. Jól, értelemszerű-en használható vezérlési szerkezetekben.

Kétféle szintaxisa van (lásd bővebben a kézikönyvben: man test). A másodikhoz szintaktikai formához kellenek a [ ] zárójelek! A

```
test  expression
```

az egyik, a

```
[  expression  ]
```

a másik szintaxis. Ügyeljenek a szóelválasztó helyközökre (space-ekre)!

Szemantika: 0-val (normálisan) tér vissza a test, ha az *expression* igaz. Tudjuk tehát az *if* és a *while* predikátumaként használni.

Az expression lehetőségek

I. csoport: fájlokkal kapcsolatos tesztelő kifejezések. Példák (nem teljes):

```
test  -f  file  # igaz, ha file letezik és "plain file"
                # (nem jegyzék, nem fifo stb.).
test  -r  file  # a file olvasható
test  -w  file  # a file írható
test  -d  file  # a file létezik és jegyzék, stb.
```

II. csoport: shell változók/adatszerkezetek relációi.

Ezt is csak példákkal mutatom be, tehát ez sem teljes! És a másik szintaktikát használom!

```
[ s1 ]             # igaz, ha s1 nem 0 sztring
[ $v  -gt  ertek ] # algebrai nagyobb v. egyenlo
[ $v  -eq  ertek ] # algebrailag egyenlő
[     -z  s1  ]    # az s1 0 hosszú
[     -n  s1  ]    # az s1 nem 0 hosszú
[ $v  =  ertek  ]  # fűzérként egyforma
```

3.7.7. További jó dolog: az expr parancs

Szintaxis:

```
expr  ertek  operator  ertek
```

Szemantika: Kiértékel és az eredményt az *stdout*-ra írja.

Az operátorokat lásd a *man expr*-ben. Mindenesetre: vannak algebrai operátorok, ekkor az *ertek*-ek numerikus stringek kell, hogy legyenek.

Példák:

1. példa:

```
$ expr 1 + 2
3
$
```

2. példa:

```
$ sum=0
$ sum=`expr $sum + 1`
$ echo $sum
1
$
```

3. példa:

```
bell
------------------------------------------------------
n=${1-1}              # ha nincs arg, akkor is 1 legyen
while [ $n -gt 0 ]
do
        echo  '\07\c'   # quotazas, \c szerepe
        n=`expr $n - 1`
        sleep 1         # alszik 1 sec-ot
done
-------------------------------------------------------
```

Hívása:

```
> bell 3      # harmat sipol
```

3.7.8. A rekurzió lehetősége

A burok programok rekurzívan hívhatók. Legyen a HOME jegyzékünkben a dw burokprogram:

```
dw
------------------------------------------------------
cd  $1  ;  echo  $1
ls  -l
for  i  in  *
do  if  test  -f  $i
    then  :
    else  $HOME/dw  $i
    fi
done
------------------------------------------------------
```

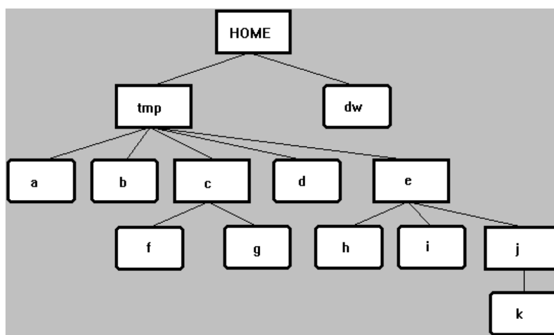
Ami itt új, az a : (kettőspont, colon) parancs, a ne csinálj semmit (*do-nothing)* parancs. A : parancsnak mindig 0 (normális, igaz) exit státusa van, bár itt ez nem érdekes.

Értermezzük, magyarázzuk, mit is csinál a dw burok-program! Bevallom, kisebb hibák vannak benne. Az egyik, induláskor nem ellenőrződik, vajon a $1 jegyzék-e (értelmes-e rá a cd

parancs). Mielőtt a 6. sorban rekurzívan újra hívódik, már „ellenőrizzük", vajon a $i jegyzék-e. De ez az ellenőrzés a `test -f   $i` szerkezettel történik, és itt lehet a másik hiba! A –f a "sima fájl" (plain file) létet ellenőrzi, nem azt, hogy jegyzék-e a fájl! (Utóbbit a test -d file parancssal tehettük volna.) Sajnos, ha a $i nem sima fájl, még nem biztos, hogy jegyzék! Vannak ui. más típusú fájlok, pl. pipe-ok, speciális fájlok, amikre a rekurziónak már nem szabadna menni. Sebaj, legalább megtanultuk a *do-nothing* parancsot! Szóval, óvatosan ezzel a példával!

Mindenesetre elemezzük, hogy mit csinálna, ha a 3.1. ábrán látható hierarchikus faszerkezetű részfájl rendszer volna és így indítanám:

```
>  dw  tmp
```



3.1. ábra. Példaprogramhoz fájlrendszer

- Belép tmp-be, echózza, hosszú listát készít róla.
- Újrahívja dw c-vel;
  - belép c-be, echózza, listát készít róla;
  - visszalép.
- Újrahívja dw e-vel;
  - belép e-be, echózza, listát készít róla. Újrahívja dw j-vel;
    - belép j-be, echózza, listát készít róla;
    - visszalép.
  - visszalép.
- Visszalép.
- Kilép.

3.7.9. A read parancs

Kulcsszós változók definiálására az értékadáson kívül még jó a *read* parancs is

Szintaxis:

```
read val1 val2 val3 ...
```

Szemantika: beolvas egy sort az stdin-ről, és a első szót a *val1*-be, második szót a *val2*-be teszi. Ha kevesebb szó van a beolvasott sorban, mint ahány változónk van, akkor a „maradék" definiálatlan marad. Ha kevesebb a változók száma, akkor az utolsó változóba a sor maradéka kerül, akár többszavas sztringként is.

A visszatérési érték 0, hacsak *end-of file* nem jött.

Beállítható a szóelválasztó karakter (alapértelmezés: fehér karakterek), az IFS shell változó értékadásával.

Tapasztalatom szerint némely burokban a read parancshoz az stdin nem irányítható át.

3.7.10. Fontos tanácsok

1.) Régebben az iit tartomány felhasználói alapértelmezési interaktív burokként a *tcsh*-t használták. Ez interaktív használatra a z SGI Irix rendszernél igen jó volt. Ha a tcsh-nak adjuk az *sh* szintaxisú parancsokat, gondunk lehet. Javasoljuk áttérni az sh-ra (ez ugyan nem kényelmes interaktív használatra), vagy a bash-ra! Utóbbi felülről kompatibilis az sh-val, de sokkal kényelmesebb az interaktív használata! Szerencsére a *gold ksh*-ja (Korn shellje) felülről kompatibilis az *sh*-val. Szerencsére, a *segédprogramok* (*utility*), *szűrők* bármely shellből ugyanúgy (v. nagyon hasonlóan) hívhatók. Tulajdonképpen csak a shell programozási vezérlési szerkezetek és a változó definíciók a tcsh-ban!

2.) Általános szokás, hogy a shell szkripteket Bourne shellben írják, az *sh*-val dolgoztatják fel. Újabban ksh-s és bash-os szkriptek is előfordulnak.

Kérdés merülhet fel, hogyan szabályozzuk azt, hogy shell szkript futtatásnál mindenképpen az *sh* dolgozza fel a programot?

Megoldások:

- Az *sh* program első sorába tegyük be

```
#! /bin/sh
```

Ez ugyan kommentár a burok processznek, de a rendszer „kitalálja" ebből, hogy az *sh*-val kell feldolgoztatnia a burokprogramot. További lehetőségek:

- Az *sh* szkript első sora semmiképp ne legyen ettől eltérő kommentár. Ha ui. az első sor kommentár, de nem a fenti, akkor a hívó shell típusával azonos shellel fogja feldolgoztatni a kernel a szkriptet. Nem lesz baj, ha a hívó shell is *sh*, de lesz, ha az *tcsh* vagy *csh*.
- A szkript első sora egyáltalán ne legyen kommentár. Ekkor ui. bármilyen shellből hívták, mindenképp az *sh* dolgozza fel. Apró példaprogramokhoz javasolható ez a megoldás, de komoly programokat illik kommentározni, méghozzá fej-kommentárral kezdeni, nem a jó tehát.
- Hívjunk előbb interaktív *sh*-t, bash-ot, ekkor a programértelmezéshez is az *sh* vagy bash dolgozik.

Interaktív Bourne shellből *csh* szkriptek feldolgoztatásához éppen az javasolt, hogy az első sor a következő kommentár legyen:

```
#! /bin/csh
```

De ritka az az emberpéldány, aki *csh* szkripteket ír!

További tanácsok

A különböző shellek indulásakor különböző *startup file*-ok "kézből" végrehajtódnak. Céljuk a globális változók definiálása, egy-két adminisztratív feladat elvégzése. Ráadásul:

Vannak közös *startup szkriptek*: ezeket a rendszergazda gondozza, írja, felügyeli.

Vannak a HOME-ban *saját startup szkriptek*. Ezeket mi is módosíthatjuk.

Egyes *szkriptek* csak a login során indulnak. De egy új terminálemuláció indítás transzparens login lehet, és ekkor ebben is elindul ez a típusú *shell szkript*).

Más *szkriptek* mindenképp futnak, ha indul egy shell (pl. > sh-ra).

A startup file-ok összefoglalása:

Bourne shell:	/etc/profile	# közös
	$HOME/.profile	# saját
C, TC shell:	/etc/cshrc	# közös
	$HOME/.cshrc	# saját, minden csh-ra
	$HOME/.login	# saját, login-re
Korn shell:	/etc/profile	# közös
	$HOME/.profile	# saját, továbbá a
	$ENV	# saját, ami szokásosan a $HOME/.kshrc-re van definiálva.

Mi, hol található a Unix-ban?

Hol vannak a végrehajtható programok, a dokumentumok, a könyvtárak (nem a jegyzékek, hanem az ar fájlok!), a konfigurációs fájlok stb., ez a kérdés. Jó összefoglaló van a Bartók-Laufer könyv 72. oldalán a konvencionális helyekre! Érdemes ezt tanulmányozni, érdemes a rendszerünkben körülnézni!

A reguláris kifejezések

Az *awk*, *ed*, *grep*, *lex, sed* segédprogramok szövegfájlokat olvasva reguláris kifejezésekkel választanak ki sorokat feldolgozásra. A kiválasztás: minta illesztése a sorra. A minta lehet reguláris kifejezés. Különböztessük meg a fájlnév-behelyettesítés minta fogalmától! A fájlnév-behelyettesítési minta az *sh*-nak szól, a reguláris kifejezés a segédprogramoknak (ezért, ha olyan dzsóker van a reguláris kifejezésben, ami fájlnév behelyettesítő karakter is egyben, quótázni kell, nehogy a burok kifejtse azt!)

A *minta* (pattern): füzérek halmaza (set of strings). A halmaz definíció az alkalmazástól függ. A *minta* valamire illeszkedhet (match). Ha illeszkedik, azt kiválasztottuk.

c	a c látható karakter önmagára illeszkedik;
\c	itt a c quótázott karakter, önmagára illeszkedik;
.	a dot bármely nem új sor karakterre illeszkedik;
[...]	füzér bezárva [ ]-be illeszkedik egyetlen karakterre, ami a füzérben van; [^...] negálás;
[a-c]	illeszkedik egyetlen, a bezárt tartományba eső karakterre;
[^a-c]	nem illeszkedik a tartomány karaktereire;
e*	illeszkedik 0, vagy több,
e+	1, vagy több,

e-	0, vagy 1 előfordulására e füzérnek;
e1e2	két összefűzött reguláris kifejezés illeszkedik az elsőre, majd a másodikra.
^|$	illeszkedik a sor kezdetére|végére.

Csakis az *awk*, *lex* és *grep* esetén továbbá:

e1|e2	akár e1-re, akár e2-re illeszkedik;
(...)	illeszkedik a bezárt reguláris kifejezésre.

További alapfogalmak:

sor/rekord	line
mező/szó	field
mezőelválasztó karakter	field separator: fehér karakter, : (colon) stb lehet.

3.8. Az awk mintakereső és feldolgozó

Az awk-t szűrőként szoktuk használni. Alapgondolata: szövegfájl sorokat olvas, minden sorban keres *mintákat* és a mintákhoz tartozó *akciókat* végrehajtja.

A szintaxisa:

```
awk  [-Fc]  [program]  [parameterek]  [file-lista]

    szóelválasztó                      input, ha nincs
      kijelölés                           az stdin
```

A szemantika:

Beolvassa az input sorait. A sorok szavait (szóelválasztó karakter alapértelmezésben a fehér karakterek egyike, megadható a -F opcióval, vagy a FS belső változó értékadásával beállítható) az *awk* 1, 2, 3 stb. nevű *mezőkbe* teszi (ezekre a programban lehet hivatkozni). Minden sorra nézi a *programban* megfogalmazott *minták* (reguláris kifejezés) illeszkedését, és amelyik *minta* illeszkedik a sorra, az ahhoz tartozó *akciót* végrehajtja. Vagyis, a program

```
minta {akciok}
minta {akciok}
...
```

formájú.

A program megjelenhet literális programként:

```
'program'
```

vagy egy fájlban:

```
-f  filename
```

A *paraméterek* segítségével további adatokat vihetünk az *awk*-ba (nézz utána!).

Akkor most egy példa, a *program* literális (egy soros és a minta hiányzik belőle, ami azt jelenti, minden sorra illeszkedik), *szóelválasztó* kijelölés nincs, *paraméterek* nincsenek, *input* az stdin:

```
> who | awk '{print $3,$4,$5,$1,$2}'
```

A példában a *who* kimenetét szűrjük az *awk*-val, a who kimenetének minden sorában a szavakat egy másik sorrendben íratjuk ki.

A program lehet több elemű, minden elem

```
minta {akciok}
```

formájú.

A programban az *akciók*: C-szerű utasítások, utasításblokkok. A fenti példában a print az egyetlen utasítás, hogy C-szerű-e, vagy sem, döntse el az olvasó. Mindenesetre, használhatók az akciók programrészben a C-szerű

if (feltétel) utasítás [else utasítás];

while (feltétel) utasítás;

for (kif1; felt_kif2; kif3) utasítás;

break, continue ;

{*utasítás; utasítás*} # összetett utasítás

printf formátum, kifejezéslista;

Ezeken kívül vannak jellegzetes *awk* utasítások is

print kifejezéslista;

for (name in array) utasítás;

next; # vedd a következő minta {akció} elemet;

exit; # exitálj

stb.

Végül változó definíciós utasítások is lehetségesek;

name értékadó-operátor kifejezés;

name[kifejezés] értékadó-operátor kifejezés;

A következőkben látunk majd példákat a programok *akciós* részére, ebben mező- és változó hivatkozásokra stb.

Érdekes a programok *minta* része is. Ezek tehát *reguláris kifejezések*, vagy *awk kifejezések* lehetnek. Legegyszerűbb az üres minta: ez minden sorra illeszkedik, a hozzá tartozó akció minden sorra végrehajtódik.

```
{print "Minden sorrra kiirni!"}
```

A következő programban a minta illeszkedik, ha a sorban valahol megtalálható a valami szó:

```
/valami/ {print "Megtalaltam valami-t."}
```

Ha valamely sorban a második mező a valami, akkor ezt jelzi az alábbi program:

```
$2=="valami"{print "Megtalaltam valamit a masodik mezoben"}
```

Vagyis a mintában a szokásos relációkkal is kapcsolhatunk mezőhivatkozásokat, awk belső változó (lásd ezeket később) hivatkozásokat is.

NF > 5 illeszkedik olyan sorokra, melyek mezőszáma nagyobb 5-nél.

$1~/valami|VALAMI/ illeszkedik azokra a sorokra, melyek első szava kis- vagy nagybetűs valami.

$1~/[Ss]treet/ illeszkedik azokra a sorokra, melyek első szava street, vagy Street.

Lehetséges tartományokat is kijelölni! Az alábbi minta illeszkedik a begin és az end szavakat tartalmazó sorok közötti sortartományra, azaz minden sorra a begin-t tartalmazó sortól az end-et tartalmazó sorig végrehajtódik az akció:

```
/begin/,/end/ {akcio}
```

Akkor most néhány további példát, melyek a /etc/*passwd* fájl sorait dolgozzák fel. Ez a fájl mindenki által olvasható, sorokból áll, a sorokban : (comma) elválasztóval hét mező található, rendre: login-név, titkos-jelszó, uid, gid, teljes-név, home-dir és induló-program. (NIS-es rendszerben az ypcat passwd parancs ilyen sorokat generál.)

1. példa:

```
> ypcat passwd | awk -F: '$4==105 {print NR,$4,$1}'
```

Kiírja a 105-ös csoporthoz tartozó számlaszámok sorszámár (NR), csoportszámát ($4) és bejelentkezési nevét ($1). A program literális, a minta a $4==105 formájú.

NR az awk belső változója, felveszi az éppen feldolgozott sor sorszámát.

2. példa, írjuk ki a passwd fájl 5. sorát!

```
> awk 'NR==5 {print $0} /etc/passwd
```

Ami új: a $0 nem mezőre, hanem az egész sorra vonatkozik. Literális a program. Van input, ez az /etc/passwd fájl.

3. példa, minden név (1-es mező) és gid (4-es mező) kiírandó:

```
> awk '{print  $1, $4}' /etc/passwd
```

4. példa, a gid, a név és az induló shell formázva írandó ki:

```
> awk -F: '{printf "%8s %4s %s \n",$1,$4,$7}' /etc/passwd
```

Láthatjuk, a printf hasonlít a C printf-hez (Nézz utána!)

Különleges minták: BEGIN és END

Különleges minta a BEGIN és az END. Ezek nem *utasítás zárójelek*, hanem *minták*:

A BEGIN illeszkedik minden sor előtt, az END a sorok feldolgozása után. Lehet tehát a BEGIN-nel inicializálni, az END-del a feldolgozás végén összegezni.

Az alábbi példákon mindjárt bemutatjuk használatukat, de a példákhoz bemutatjuk az awk további belső változóját, az FS-t (Field Separator). Az FS a mezőelválasztó karakter alapértelmezés szerint fehér karakter, beállítható a -Fc opcióval, illetve az awk programban FS=c értékadással is.

5. példa, számláljuk az input sorait. (Bár ez egyszerűbben is megoldható, most így csináljuk!) A példa bemutatja azt is, hogy a program lehet egy fájlban is:

Legyen a *prog* fájl tartalma:

```
------------------------------------
BEGIN { s=0}
      { s = s + 1}
END   { print "osszeg: ", s}
------------------------------------
```

Illetve, legyen *prog1* tartalma:

```
------------------------------------
BEGIN {FS=":";  s = 0}
      { if ($4 == 105) s = s + 1 }
END   { printf "Osszeg: ", s}
------------------------------------
```

Akkor

```
> ypcat passwd | awk -f prog
 osszeg: ddd
```

kiadja a számlaszámok számát, míg a

```
> ypcat passwd | awk -f prog1
Osszeg: nnn
```

kiadja a 105-ös csoportba tartozók számát.

A programokhoz még: látható az s = 0 értékadás, ami egyben egy belső változó definíciója is. Mindkét program három *minta {akció}* szerkezetet tartalmaz.

A számlaszámok számának egyszerűbb kiíratása:

```
> ypcat passwd | awk 'END { print NR}'
```

Mielőtt további példákat adnánk, foglaljuk össze az awk legfontosabb **beépített változóit:**

FILENAME	az aktuális input fájl neve, akár meg is változtatható;
FS	mezőelválasztó;
NF	mezőszám egy sorban;
NR	a pillanatnyi sorszám;
RS	input sorelválasztó (default: újsor-karakter);
OFMT	output formátum (default: %g);
OFS	output mezőelválasztó (default: szóköz);
ORS	output sorelválasztó (default: újsor)
stb.	

Az awk operátorok (csökkenő precedencia):

++, --	pre/postfix inkrementáció, dekrementáció;
*, /, %	multiplikatív operátorok;
+, -	additív operátorok;
	karakterlánc összekapcsolás (semmi operátor);
<,<=,>,>=,==, !=, ~, !~	relációs operátorok, ~ az egyezés, !~ a nem egyezés operátora;
!	kifejezés értékének tagadása;
&&	és;
||	vagy;
=, +=, -=, *=, /=, %=	értékadó operátorok.

Az awk-nak van néhány **beépített függvénye** is:

sqrt, log, exp, int	matematikai függvények.
length(s)	szöveghosszt adja vissza.

substr(s,m,n)	substring s-ből, m-től, n hosszan.
index(s,t)	s füzérben t első előfordulásának indexe.

Tömbök az awk-ban

Lehetséges az alábbi tömbdefiníció:

```
tomb_name[konst_kifejezes]  ertekado_oerator  kifejezes;
```

6. példa:

prog2

```
---------------------------------
    { line[NR] = $0 }
END { for (i=NR; i > 0; i--)
     print line[i]
    }
---------------------------------
```

Ez a program két komponensű. Első komponensében nincs minta, minden sorra illeszkedik tehát. Az akció részében tömbdefiníció van: line[NR] = $0 formában, szöveglánc elemeket tartalmazó *társtömböket* definiál, amiknek indexe a sorszám!

Hogy hogyan helyezi el őket, az nem érdekes. Mindenesetre lehet minden definiált elemére később hivatkozni!

Mit csinál ez a program? Fordított sorrendben kiírja a sorokat! Vigyázzunk, sok-sok sorból álló inputra veszélyes elereszteni, mert elfogy a memória!

A társtömbök indexei és az awk speciális for-ja

Társtömb index akármilyen konstans kifejezés lehet.

Használható a for (*name* in *tömbname*) ciklus is.

A 7. példán mutatom be a két új koncepciót. A példához tételezzük fel, van egy szövegfájlunk (ez lesz a feldolgozandó input), amiben név-érték párok vannak. A nevek ismétlődhetnek:

```
szovegfile
--------------------------
joe             400
mary            200
joe             200
john            300
susie   500
mary            200
-------------------------
```

Összegezzük az egyes nevekhez tartozó összegeket. A *prog* fájl:

```
prog
-----------------------------------
    { sum[$1] += $2 }
END { for (name in sum)
      print name, sum[name]
    }
-----------------------------------
```

Így hívjuk:

```
> awk -f prog  szovegfile
...
```

Az awk rendre társtömböket definiál

```
sum[joe]
sum[mary]
sum[john]
sum[susie]
```

nevekkel, ezekben összegzi az értékeket.

Figyeljük meg a for ciklust! A for és az in kulcsszó, a *name* általunk választott változónév.

És egy utolsó példa, szószámlálás. Vigyázzunk erre is, sok szóból álló szövegre veszélyes elereszteni. Csak a programot adom meg:

```
    { for (i=1; i<= NF, i++) num[$i]++}
END { for (word in num)
      print word,num[word]
    }
```

Meg tudják magyarázni?

4. Hálózatok, az Internet

Az egyedülálló számítógép hasznos az élet minden területén. Hálózatba kapcsolva a számítógépek még hasznosabbak. Korábban a hálózatosodás legfőbb oka az erőforrás-megosztás, az erőforrás-összevonás volt, manapság ezt kiegészíti a számítógépes kommunikáció (Computer Mediated Communication). A Internet világ kiteljesíti ezt a paradigmát, hiszen azt mondhatjuk az Internet a hálózatok hálózata, ahol egy hálózat egy csomópontjának felhasználója földrajzi és politikai határokat figyelmen kívül hagyva kommunikálhat bármelyik hálózat felhasználójával, bármelyik csomópont kérhet vagy biztosíthat szolgáltatásokat másik csomóponttól, csomópontnak (hacsak biztonsági okokból nem tiltják külön a hozzáféréseket).

A kapcsolatban lévő hálózatokat sem lehet felsorolni, nemhogy az Internetre csatlakozó gépeket. Azt sem lehet megállapítani, hogy hány hálózat, hány csomópont tartozik az Internethez, hány Internet felhasználó van a világon, csak az mondható biztosan, hogy állandóan növekszik. Illetve az is biztosan állítható, hogy a kapcsolatban lévő csomópontok, illetve a felhasználók számát tekintve az Internet a legnagyobb hálózat a világon.

4.1 Az Internet története

4.1.1. Az ARPANet

A 60-as években az USA-ban a Defense Advanced Research Projects Agency (DARPA) támogatásával kutatás indult, melynek kettős célja volt. Az egyik cél az volt, hogy telefonhálózaton keresztül csomagkapcsolással lehetne-e számítógépeket hálózatba kapcsolni, hogy egy vonalat több felhasználó is használhasson "egyidőben". A másik cél - nem kevésbé fontos - pedig, olyan hálózatot készítsenek, amelyik működőképes marad akkor is, ha háborús események miatt a hálózat egy része el is pusztulna: ha kiesnek bizonyos csomópontok, vonalak, automatikus átirányítással más vonalakon, csomópontokon továbbíthatók legyenek az üzenetek.

1969-re a kutatás-fejlesztések eredményeként - néhány (University of California at Los Angeles, Standford Research Institute, University of California at Santa Barbara, University of Utah) helyszín összeköttetésével - kialakult az ARPANet. 1971-re 15, 1972-re 37 növekedett ez a szám, ami az ARPANet sikerét jelezte. Bár a támogató hadügyminisztériumának (Department of Defense, DoD) eredeti célja a távoli helyszínek közötti megbízható, robosztus összeköttetések megvalósítása volt, a kutatás-fejlesztésben résztvevők - mivel nagyon kényelmesnek találták - elkezdték a hálózatot személyes üzenetváltásokra is használni, és például az elektronikus levelezés az ARPANet segítségével hallatlanul népszerűvé vált.

Az az elv, hogy nincs a hálózatnak külön központi üzemeltető gépe (vagy gépei) valóban hozta a tervezett előnyt: nem omlik össze a hálózat részeinek meghibásodásával. Hátrány is következett persze ebből: meglehetősen nehéz a hálózaton történő navigálás. A különböző gépek különböző operációs rendszerei és felhasználói felületei, parancsértelmezői gondot jelentettek. Ebben a korai időszakban még messze nem voltak meg a ma jól ismert egységesített navigációs eszközök, böngészők, keresők.

4.1.2. Az RFC-k (Request for Comments)

Az ARPANet kezdeti fejlesztési időszakában nemigen voltak még hálózati szabványok. Ezért a kutató-fejlesztők kitaláltak egy meglehetősen informális módszert a "szabványosításra", az

RFC-k módszerét. Ha valakinek volt valamilyen javaslata valamilyen megoldásra, akkor közzétette ezt egy ún. előzetes RFC-ben (draft RFC), az ARPANet társadalom megvitatta, kommentálta, javította a javaslatot, és végül megegyezéssel elfogadta az RFC-t. Ekkor az RFC sorszámot kapott és ezzel szabvánnyá (Internet Standard) vált: a számával lehet hivatkozni rá. Az első RFC-t 1969-ben S. Crocker publikálta. Manapság az RFC-k száma meghaladja a 2000-et. A módszer sikerét az akadémiai kutatók együttműködésre való hajlama segítette: bárki tehetett előzetes javaslatot, a közzétett javaslatot bárki kritizálhatta, javíthatta, széleskörű volt e megegyezés a végleges szabványról. Maga az ARPANet pedig a "szabványosítás" folyamatát gyorsította, a vita és megegyezés a hálózat segítségével történhetett.

4.1.3. A korai protokollok

A korai 70-es években az ARPANet-re meglehetősen különböző számítógépeket kapcsoltak. Minden helyszínen volt egy *interface message processor* (IPM) csomópont, egy kiskapacitású miniszámítógép, a helyszínek közötti összeköttetéshez, de a helyszín bármelyik számítógépét rákapcsolhatták a hálózatra. Eleinte a TCP/IP protokollszövet még nem volt kialakítva, egy korlátozott lehetőségeket biztosító hálózatvezérlő programcsomagot használtak az ARPANet-ben. A szállítási réteghez tartozó protokoll az Network Control Protocol (NCP) volt.

1974 májusában V. Cerf és R. Cahn cikke az IIIE Transactions on Communication-ban, A Protocol for Packet Network Interconnecting címmel az első javaslat a TCP protokollra. Ezután következett az RFC szabványosítás: a TCP szabvány az RFC-793-ban található. Az *átvitelvezérlő protokoll* (Transmission Control Protocol, TCP) megbízható csatlakozásorientált protokoll.

A *hálózatok közötti protokoll* (Internet Protocol, IP) a üzenetcsomagok továbbításra szolgáló protokoll, az RFC-791-ben rögzített. A TCP és az IP az Internet legismertebb protokolljai, összefoglaló betűszavuk a TCP/IP. Függetlenek a hálózat alsóbb rétegeitől, a fizikai közegtől.

A TCP/IP protokollszövet annyira összekapcsolódott az Internettel, hogy néha el is felejtkezünk arról, hogy más protokollok is tartoznak az Internet protokoll szövethez. Nem részletezve tekintsük át a legfontosabbakat, a "hagyományos" szolgáltatásokra való pillantással.

Egyik korai szolgáltatás az már az ARPANet korában a *számítógépes levelezés*. A SMTP (Simple Mail Transfer Protocol, RFC-821) és a TMP (Text Message Protocol, RFC-822) a legalapvetőbb dokumentumok (1982-ben keletkeztek). Az RFC-937 már a mikroszámítógépes személyi levelező ügynök és a kiszolgáló gép közötti kapcsolatot szabványosítja, az RFC-1521 pedig a MIME (Multipurpose Internet Mail Extension) leírása.

Szintén korai szolgáltatás a *távoli bejelentkezés*. A *telnet* hálózati terminál protokoll az RFC-854--860 szabványokban rögzített. Általában egy ismert terminál emuláció és egy kapcsolatépítő a legfontosabb alkalmazási szoftver elemek. A kapcsolatépítéshez a telnet-nek a távoli csomópont azonosítóját (címét vagy nevét) megadva a kiépülő kapcsolat nagyjából egy közvetlen vagy modemes vonali kapcsolathoz hasonlít: minden begépelt karakter a távoli rendszerhez kerül. A távoli rendszer a viszony (ülés, session) létesítéshez rendszerint a bejelentkezési eljárást (login procedure) biztosítja, ha van számlaszámunk a távoli gépen, akkor azonosítónk és jelszavunk megadásával létesíthetünk viszonyt, ülést a kapcsolaton.

További alapvető szolgáltatás az *állománytovábbítás*. Protokollja az állománytovábbító protokoll (File Transfer Protocol, FTP, RFC-959). A szolgáltatásokat az *ftp ügynök* programok segítségével igénybe vehetjük.

4.1.4. A történet folytatódik

1975-ben az ARPANet fejlesztés a Defense Communication Agency (DCA) felügyelete alá felügyelete alá került. 1882-ben elkezdve, 1983 január 1-jére az ARPANet teljesen áttért a TCP/IP protokollszövetre. Ekkor kezdték használni az *Internet* kifejezést - az IP protokoll nevéből származtatva a szót - azt is jelezve ezzel, hogy az Internet hálózatok hálózata, hiszen az ARPANet-hez egyre több hálózat kapcsolódott, most már átjárók nélkül, mert azok is a TCP/IP szövetet használták (pl. a Bitnet). Megkezdődött az Internet rohamos, megállíthatatlan növekedése.

1983-ban az ARPANet-Internet két hálózatra vált szét: MILNET-re és ARPANET-re. Talán ez az esemény az Internet igazi születésnapja, hiszen a szétválás lehetővé tette az USA hadügyminisztérium által eddig nem támogatott intézmények Internetre való csatlakozását is.

1884-ben állították be az első névszolgáltatót (domain name server). Kialakult a japán JUNET, az Egyesült Királyságban a JANET. 1986-ban az NFSNET is elindult. 1988 az Internet *féreg* éve: az Internet mintegy 6000 számítógépét "támadta meg" a *worm* nevű számítógép program. A hatására alakították ki az RFC-1087 azonosítójú Ethics and the Internet dokumentumot.

1989-ben alakult az Internet Engineering Task Force (IETF) az Internet Activities Board (IAB) alá rendelődve. 1990-ben az ARPANet hivatalosan megszűnt. Ez az év azért is jelentős, mert megjelent az Archie: az ftp-ző felhasználók ennek segítségével kereshették, hol találják meg az érdeklődésükre számot tartó állományokat. 1991-ben alakult a Commercial Internet Exchange Association (CIX): tagjainak a saját hálózataikon ingyenes adattovábbításokat biztosítottak, az USA kormányzat által támogatott NFSNET korlátozásokat elkerülhették. 1991 a *gopher* megjelenésének is az éve. Említsük meg szerzői nevét: Paul Linder és Mark McCahill, University of Minnesota. Ugyanekkor jelenik meg az Interneten a WAIS (Wide Area Information Service)

1992-ben az Internetre kapcsolódó gépek száma meghaladta az 1 milliót. Megalakult az Internet Society (ISOC), azzal a céllal, hogy az Internet technológiák fejlesztését és használatát, a szabványosításokat kézben tarthassák. Ez az év az első World Wide Web (WWW) böngésző kibocsájtásának éve. A CERN Europen Laboratory for Particle Physics, Genf, Svájc kutatója Tim Berners-Lee a WWW koncepció szülője.

1993-ban alakult az Internet Network Information Center (InterNIC), feladata a regisztráció, a szabványok RFC-k gondozása, információszolgáltatás az Internetről. Megjelenik az első grafikus WWW böngésző, a Mosaic. A képmegjelenítés, a hanglejátszás élménye miatt 1994 áprilisára a WWW forgalom felülmúlja a gopher forgalmat.

4.1.5. További protokollok

A már eddig említett alapvető szolgáltatásokhoz tartozó protokollok mellet további protokollok alkotják az Internet protokoll szövetet. A fontosabbakat megemlítjük ezek közül, megjegyezve, hogy ezek ügynök-szolgáltató (klient-server) jellegű szolgáltatási körbe tartoznak.

A *hálózati állományrendszer* (network file system) szolgáltatás segítségével az FTP módszert meghaladva lehet fájl-szolgáltatást biztosítani. Segítségével egy számítógép számára virtuális meghajtókat biztosítnak más rendszerek állományaiból. Takarékossági előnyök, közös állomány hozzáférés, könnyebb rendszerkarbantartás, archiválás lehetősége adott így, hogy csak a legfontosabbakat említsük az előnyeiből. A TCP alapú PC-kre való NetBIOS leírását az RFC-1001-1002 közli. Unix-os munkaállomásokhoz, szolgáltató gépekhez ma leggyakoribb a Sun Network File System (SUN NFS) hálózat fájlrendszere. A protokollokat a Sun Microsytem fejlesztette, szolgáltatja.

A *távoli nyomtatás* legszélesebb körben használt protokollja a BSD távoli sornyomtató protokollja. Sajnos, dokumentált leírása nem létezik, de C nyelvű forráskódokhoz a megvalósításhoz hozzá lehet jutni.

A *távoli futtatás* hasznossága nyilvánvaló ha adott munka erőforrás-igényes részét erőforrás-gazdagabb gépen akarjuk elvégezni. A leggyakoribb megoldások a BSD *rsh* és *rexec* kiszolgálói, a man lapokon leírásuk megtalálható. A távoli eljáráshívás legelterjedtebb protokolljai a Xerox cég Curier, ill. a Sun RPC. Leírásukat az adott cégektől kell megszerezni, de megemlítjük, hogy a BSD 4.3-tól kezdve ezek megvalósítása megvan (a Sun RPC csak részben).

A *névszolgáltatás*. Miután rengeteg nevet kell kezelni a nagykiterjedésű rendszerekben, ma minden TCP/IP megvalósításnak része kell legyen ez a szolgáltatás. Az érintett protokollokat az RFC-822-823 írja le. A Sun Yellow Pages rendszere - ma már Network Informatin Service (NIS) néven a felhasználók neveinek számlaszámainak, csoportjainak menedzselése mellett további szintet ad a névfeloldásra, Unix rendszerek által használt adatbázis kezelésre, szolgáltatás nevek kezelésére.

Terminál szerver szolgáltatás. A terminálszerverek kisebb teljesítményű speciális célú operációs rendszer alatt működő számítógépek. A már említett telnet protokoll mellett a névszolgáltatási protokollokat is kezelik, gyakran más protokollokat is ismernek (pl. távoli nyomtatási protokollokat, nem TCP/IP szövetet stb.). A terminálokat sokszor nem közvetlenül kapcsolják a gazdagépekre, hanem a terminálszerver portjaira, a terminálszerver végzi a kapcsolást, és miután rendszerint lehetséges egyszerre több aktív kapcsolat létesítése is, a szerver az aktív kapcsolatok közötti váltásokat is segítheti.

A *hálózati alapú grafikus megjelenítés* jelentősége azért nagy, mert így nem a gazdagéphez közvetlenül kapcsolt bit-térképes grafikus képernyőn is lehetséges a grafikus megjelenítés. Legszélesebb körben elfogadott szabvány az X11. Leírásukat több helyről is beszerezhetjük, az X11 ma a UNIX rendszerek természetes része. Azt is meg kell említeni, hogy az X11 a HTML terjedésével veszít jelentőségéből.

És végül megemlítjük, hogy az Internet szabványok aktív listáját megkaphatjuk az RFC-1011-ből.

4.2. Az Internet Magyarországon

Az Internet magyarországi története teljesen összefonódott az Információs Infrastruktúra Program és az azt követő Nemzeti Információs Infrastruktúra program történetével.

A program gondolata 1985-ben született, Vámos Tibor akadémikus merész kezdeményezésére. Gondoljuk meg, abban az időben, amikor minden hálózati termék szigorú embargó alá

esett, egy országos kutatói számítógépes hálózat létrehozását javasolta Vámos Tibor. Szerencsére, az OMFB, a Magyar Tudományos Akadémia vezetői felismerve a kezdeményezés jelentőségét biztosítottak megfelelő szervezeti és pénzügyi feltételeket: elindulhatott az IIF program, és első szakasza 1986 és 1990 között sikeresnek bizonyult.

1988-ra a Magyar Postánál üzembe helyezték a hazai fejlesztésű 80 vonalas csomagkapcsoló központot, és elkészült az ELLA elektronikus levelező program. 1989-ben az adathálózaton hazai és nemzetközi szolgáltatásokat biztosítanak, mintegy 100 végrendszer kapcsolódik a hálózatra. A kiépülő rendszer illeszkedett nemzetközi szabványokhoz (OSI, X.25). A szolgáltatások hazai (ELLA) és nemzetközi (EUnet) elektronikus levelezésre és távoli számítógép használatra, ezen keresztül adatbázisok lekérdezésére terjedtek ki.. Más nemzetközi kapcsolatokat, szolgáltatásokat az embargó miatt nem lehetett igénybe venni.

1990-ben a politikai változások lehetővé tették a nemzetközi kapcsolatok bővülését, csatlakozhattunk az EARN-hoz (European Academic and Research Network). A Magyar Posta az adathálózat kapacitását növelte, megnyitotta a nyilvános csomagkapcsolt adatszolgáltatást. Az embargó a hálózati elemek behozatalát még mindig akadályozza, a helyi hálózatok összekapcsolásának eszközei hazai fejlesztéssel alakultak ki. Mintegy 200 intézmény, felsőoktatási intézmények, kutatóhelyek, könyvtárak és múzeumok, alkotják az IIF intézeteket. Tulajdonképpen sikerrel lezárul a program első szakasza.

A sikerre való tekintettel a program tovább erősödött, 1991-ben kibővült a támogatók köre, az OMFB-hez és az MTA-hoz támogatóként csatlakozott a Művelődésügyi és Közoktatási Minisztérium és az Országos Tudományos Kutatási Alap (OTKA). Az IIF második fázisa indul, kissé módosuló fejlesztési céllal. Most már nemcsak hazai fejlesztésű hálózati elemekre lehet építeni, és most már az egész országot lefedő hálózatra is lehet gondolni. Mivel 1988-tól az USA engedélyezte Európa számára is az Internet technológiákat, a meglévő európai felsőoktatási és kutatói hálózatok Internetre való csatlakozását, az európai infrastruktúrán keresztül nekünk is lehetőségünk nyílott az Internet használat. Megkezdődhetett a kísérleti Internet kapcsolatok felépítése, a névszolgáltatás biztosítása, az elektronikus levelezésben az Internet címzések használata. A MATÁV korszerűbb, import csomagkapcsoló központokat szerezhetett be, melyekkel az adathálózat minőségét, teljesítményét és a szolgáltatások körét bővíthették.

1992-re Unix-os gazdagépek kerültek a rendszerbe. Ezekhez "alapértelmezés" szerint tartozik a TCP/IP protokollszövet és a névszolgáltató szoftverrendszer. 1993-ra Világbanki támogatással szinte az összes felsőoktatási intézményben elterjedtek a UNIX konfigurációk, korszerű helyi hálózatok alakultak ki. Elindult a HBONE, a hazai IP protokollú gerinchálózat kialakítása. Az IIF program finanszírozásából nem csak a fejlesztések folytak, hanem az intézetek ingyenesen vehették igénybe a hálózati szolgáltatásokat. Az IIF programban ekkor mintegy 450 intézmény vett részt. Sikerét elismerték mind a hazai, mind külföldi körökben.

1993-ban az IIF - ismét megújulva - a HBONE (országos bérelt vonalas, IP technológiájú gerinchálózat) megerősítését, az IP technológiákra épülő szolgáltatások (telnet, ftp, smtp, gopher, wais, archie, news stb.) hozzáférhetővé tételét, terítését tűzte ki célul. Az elgondolás az volt, hogy a szigetszerűen már kialakult vagy kialakuló helyi, nagy-forgalmú IP hálózatokat közvetlenül a gerinchálózatra kapcsolják, míg a többi intézmény, a kisebb elszórt felhasználók a nyilvános X.25. hálózat, legrosszabb esetben a nyilvános telefonhálózat közvetítésével érhesse el a gerinchálózatot. Nem elhanyagolható cél továbbá: a HBONE és a nagy nemzetközi hálózatok megbízható, nagy kapacitású vonalakkal való összekapcsolása is.

1995 folyamán a HBONE kialakult. A nagy-megbízhatóságú gerinchálózati mag Budapesten a KFKI-ban, az IIF központban és a BME-én elhelyezett router-ekből, az azokat összekötő 2Mbps sebességű mikrohullámú kapcsolatokból, a MATÁV Városház utcai központjában lévő router-ekből és kapcsolatrendszerükből állt. Utóbbinál kapcsolódnak a HBONE nemzetközi vonalai. Ugyancsak a MATÁV-nál hozták létre a Budapest Internet eXchange (BIX) csomópontot, melyet már tízegynéhány profitorientált Internet szolgáltató is finanszírozott. Ennek célja az volt, hogy a profitorientált szolgáltatók által menedzselt hazai felhasználók közötti forgalom a BIX-en keresztül cserélődjön ki, és ne terhelje egyik szolgáltató nemzetközi vonalait. Ez egy lényeges mozzanat: az IIF intézmények mellett a profitorientált intézmények is kacsintgatnak az Internetre. A HBONE a budapesti magból és mintegy 20 regionális központból állt 1995-96 fordulóján, és csak néhány regionális központ bekapcsolása volt hátra, hogy az összes megyeszékhelyen csomóponttal rendelkezzen. A vonalszélességekben is volt fejlődés, az öt legnagyobb forgalmú regionális központhoz (Veszprém, Pécs, Szeged, Debrecen, Miskolc) az 512 Kbps kapcsolatot 1996 őszén biztosították. A fejlődést mutatja, hogy 1996 áprilisára a HBONE névszolgáltatóinak száma meghaladta az 500-at, ugyanekkor a bejelentett elérhető gazdagépek száma meghaladta a 22000-et [Martos-Tétényi96].

Az 1996-os év több szempontból is jelentős. Egyrészt az IIF program átalakult NIIF programmá: Nemzeti Információs Infrastruktúra Programmá. Másrészt ez az év, amikor a profitorientált Internet szolgáltatók színre léptek és nagy sikereket értek el. (Feltétlenül meg kell említeni, hogy az IIF nélkül sikerük nem lett volna ilyen átütő, fejlődésük nem lett volna ilyen rohamos. Az IIF és a HBONE fejlesztés Magyarországon kialakította azt a szakembergárdát, amelyik az Internet technológiákat ismeri, a rendszereket működtetni tudja. Másrészt e rövid idő alatt is kialakult az Internet felhasználók széles köre: a mai főiskolai és egyetemi hallgatók képzésének eleme az Internet használat, egyre többen úgy lépnek ki az iskolákból, hogy ismerik és igénylik a hálózat szolgáltatásait. Sőt, ma már a középiskolák vannak a soron, megkezdődött ezek rákapcsolása is a Hálózatra.

4.3. Csomópontok azonosítása az Internet-en

A forgalomirányítók az IP címeket használják a csomagok továbbítása során. Az IP címek egyediek a csomópontokra (node-kra).

Példa:

```
193.6.10.1      gold     IBM R/6000 970      AIX
193.6.5.33      zeus     SGI Power Series    Irix
```

Hasznos feljegyezni fontos csomópontok IP címeit. A csomópontok neveit a rendszergazdák választják, nem egyediek.

Példa:

```
kuka az IIT-n:  VAX 2000,     VMS (már nincs!)
kuka a BME-n:       ?         ? (152.66.81.30)
```

A címek megjegyzése kellemetlen feladat. Jobban szeretjük a neveket használni parancsainkban. A megoldás tehát: név - IP cím feloldást kell biztosítani!

A névfeloldás alapgondolata: egy táblázatban rendeljük össze az IP címeket és a neveket. Ez a táblázat lehet az /etc/hosts fájl, vagy NIS rendszerben a megfelelő adatbázis. Felmerülnek azonban gondok! Ha van több mint 1 millió csomópont, mekkora lesz ez a tábla? És mi lesz az ismétlődő nevekkel (name collision)? És ha cserélődnek a nevek, címek, hogy biztosítható a naprakész állapot (consistency)?

A továbbfejlesztett megoldás célja (kissé leegyszerűsítve most) a következő: a helyi adminisztráció engedélyezett legyen (azaz a neveket lehessen viszonylag szabadon megválasztani), mégis biztosítsanak globális eléréseket.

1984-ben javasolták (RFC 822, 883; később RFC 1535,1536,1537) a **tartomány-név rendszer** (Domain Name System, **DNS**) hierarchikus osztott adatbázist, melynek helyi adminisztráció biztosított egy-egy szegmensén, de globális az elérése. Kliens-szerver koncepciójú. Ehhez bevezették a *névszolgáltató* (name server) és a *névfeloldó* (resolver) fogalmakat.

A koncepció szerint felosztották a világot *tartományokra* (domain), *nevet* is adva a tartományoknak. A tartományok szervezésében az összefogó erő lehet a földrajzi elhelyezkedés, lehet az politikai összetartozás, gazdasági, kulturális, társadalmi, szervezeti hasonlóság is. *Csúcstartománynak* nevezzük a legfelső szintű tartományokat (ezeknél az összefogó erő legtöbbször a közös államiság). Egy tartományon belül lehetnek

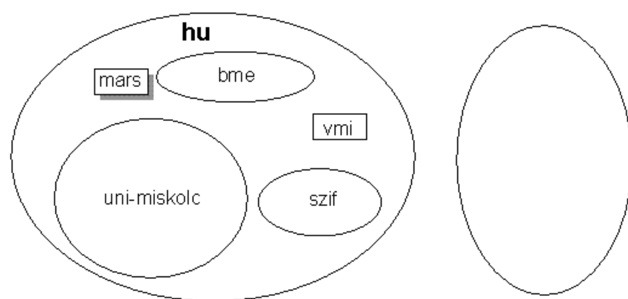
- *altartományok* (subdomain) a nevükkel,
- *csomópontok* (node-k) a nevükkel. Ezeknek IP címük is van.

Fontos, hogy a nevek a tartományon belül egyediek legyenek.

Az 4.1. ábrán láthatjuk a *hu* tartományt, benne *uni-miskolc*, és *bme* altartományokat, illetve a *mars* nevű csomópontot (természetesen a hu tartományban sokkal több altartomány és csomópont van. mint amit az ábrán láthatunk). A *hu* csúcstartomány.

A nevek szintaktikája ezután:

```
node-név.altartomány-név.csúcs-tartomány-név
node-név.csúcs-tartomány-név
altartomány-név.csúcs-tartomány-név
```



4.1. ábra. Példa tartományra

Látható, ez a névtér (domain name space) hierarchikus (max. 127 mélységig mehet). Egy egyszerű név (simple name) max. 63 karakteres lehet. A teljes tartománynév, csomópontnév pedig egyszerű nevek listája, a pont (dot) elválasztóval, egy tartománytól, csomóponttól a gyökér tartományig (csúcsig). (Ebben az absztrakcióban egy csomópont egy tartománynak felel meg!)

A tartománynév indexeli a DNS adatbázist! Az adatbázis pedig információ-

kat tartalmaz! Pl. egy gazdagépet indexelve a gazdagép (host) a hálózati címét (IP cím), HW jellemzőit, elektronikus levelezéshez útvonal-irányítási információit (e-mail routing) tartalmazza. Egy tartományt indexelve az adatbázis strukturált információkat tartalmaz a gyermekeiről!

Az információk hierachikusan leosztottak és decentralizáltak: a decentralizálás eszköze a felelősség leosztás (delegation).

Az információkat a névszolgáltatók biztosítják, melyek a delegált szervezet felügyelete alatti gazdagépen futó programok (továbbiakban gazdagépet említünk névszolgáltatóként). Egy névszolgáltató teljes információkat szolgáltat a névtartomány egy részéről (zónájáról). Egy szolgáltató több zónáról is szolgáltathat adatokat. Bár a zóna sokszor egybeesik egy tartománnyal, nem egy tartomány! A zóna lehet nagyobb, mint egy tartomány (ez azt jelenti a névszolgáltató nemcsak egy tartományért felel), lehet kisebb is (Pl. egy tartomány altartományaiért való felelősséget delegálták, akkor a delegált altartományért más névszolgáltató felel, de az altartomány megmaradt a felettes tartomány részének). Úgy is mondhatjuk, hogy a zónába beletartozik a delegáltakon kívül minden (gazdagépek is, altartományok is).

Válasszunk ki egy tartományon belül egy csomópontot, ami a *névszolgáltató* gép (name server) lesz! Határozzuk meg ennek a zónáját!

Példa: legyen az uni-miskolc tartomány névszolgáltatója a gold és legyen a tartományban az 4.2. ábra szerint néhány altartomány, néhány csomópont. Legyen a zónája a uni-miskolc tartomány minden gépe és altartománya, kivéve az iit altartományt (azt delegáltuk az Informatikai Intézethez).

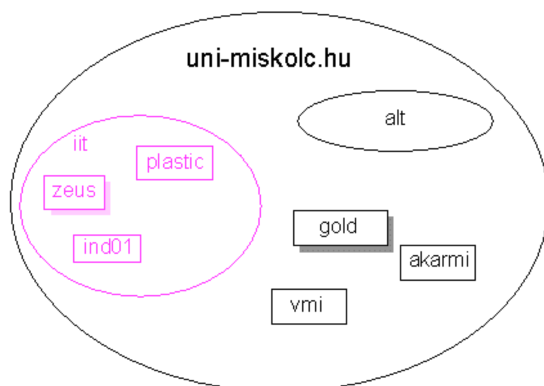
Tételezzük fel, hogy akarmi-ról keresem vmi-t (egy kliens program hív névfeloldó - resolver - rutint), és az nincs az akarmi saját névfeloldó táblájában.

Ekkor *akarmi* megszólítja a névszolgáltatót (name-server), itt a *gold*-ot, és kéri a név feloldását.

gold köteles ismerni a zóna összes nevét, és azt is, hogy az egyes nevekhez tartomány tartozik-e, vagy csomópont. Utóbbi esetben tudja az IP címét is: ezt tehát feloldhatja.

- A névszolgáltató köteles ismerni a delegált tartomány név-szolgáltatóját is., hogy feloldáshoz segítséget kérjen.

- A névszolgáltató köteles ismerni a csúcstartomány névszolgáltatóját is, onnan is kérhet segítséget a feloldáshoz. A csúcstól lefelé haladva a hierarchián, előbb-utóbb feloldható a név.



4.2. ábra. Az uni-miskolc tartomány (csak képzeletben!)

A névszolgáltatók ideiglenesen tárolnak (cache-elnek) információkat (akár másik névszolgáltatóról, akár csomópontról; ez gyorsítja a névfeloldást, csökkenti a hálózati forgalmat. Az ideiglenes tárolás "ideje" beállítható (konzisztencia így biztosítható változások esetén is).

Összefoglalva:

Vannak csúcs-tartományok. (pl. *.hu*, *.de*, *.com*, *.edu* stb.), ezeknek névszolgáltatóik.

A tartományok névszolgáltatói ismerik "felfelé" a csúcs névszolgáltatót (root name server), "lefelé" az zónájuk altartomány neveit és a delegált tartományok névszolgáltatóit, valamint a saját zónájuk csomópontjainak név-IP cím párjait.

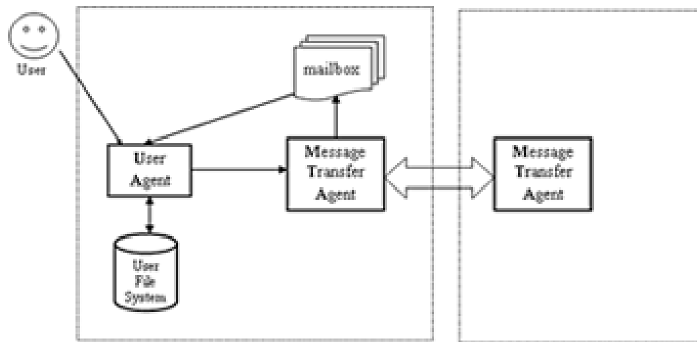
Mindig csak lefelé kell *névegyeztetést* biztosítani: ez még kézbentartható organizáció!

A tartományon belül a névkiosztás elég szabad, csak a névszolgáltató menedzserével kell egyeztetni. Ő ugyanekkor az IP címek kiosztását is adminisztrálja (neki kiadhatnak egy címkészletet, ebből adhat címeket).

4.4. Az elektronikus levelezés alapfogalmai

A hagyományos postai küldeményekhez továbbításához hasonlíthatjuk. Ott vannak postaládák (piros színű szekrények), amelyekbe bedobhatjuk a megcímzett, felbélyegezett leveleinket. Van postaszolgálat, ami kiveszi, osztályozza, továbbítja a leveleket, és végül kikézbesíti: bedobja a címzett postafiókjába (leveles szekrényébe). Innen a címzett kiemelheti és olvashatja az üzenetet.

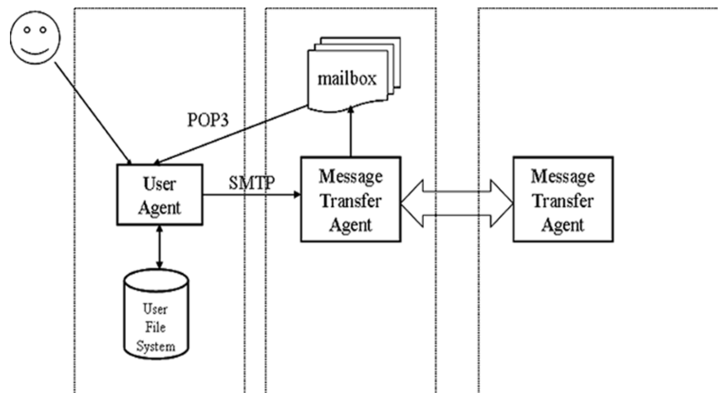
Az elektronikus levelezés egyszerűsített sémája az 4.3. ábrán látható. A *személyek* a helyi gazdagépen (host) *levelező ügynök* (User Agent) programot futtathatnak, és van a gazdagépen egy *postafiókjuk* is. A postafiók az *elektronikus levelezési címmel* (e-mail address) azonosított. A hálózat csomópontjain futnak *levél továbbító programok* (MTA: Message Transfer Agent) is.



4.3. ábra. Elektronikus levelezés sémája

UA - a helyi gazdagépen fut. Itt van a "postafiókunk" is. Ez biztosítja:

- a postafiók vizsgálatát,
- a levél megtekintését (view funkció),
- a levél törlését, áthelyezését *irattartóba* (jegyzékbe), kinyomtatását, lementését egy fájlba stb.
- Biztosítja továbbá feladandó levél szerkesztését (edit funkció),
- levél megcímzését ("fej" készítését),
- levél feladását.
- (Némely ügynök a helyi leveleket képes letenni a postafiókba!)



4.4. ábra. A levelező kliens PC-n fut

A User Agent csak akkor fut, ha a felhasználó elindítja! A felhasználói ügynök futhat személyi számítógépen is, ekkor a modell a 4.4. ábrán látható

MTA - a helyi gazdagépen és további csomópontokon is fut. Ún. "daemon", tehát aktív, a rendszermenedzser indítja., menedzseli.

Biztosítja:

- a feladott levél továbbítását (sending funkció),
- átfutó levelek továbbítását (relay funkció),
- bejövő levél elhelyezését a megfelelő postafiókba. (Némely ügynök erre külön processzt hív, nem maga végzi a lehelyezést).

Egy közbenső csomóponton futó MTA ideiglenesen sorokban (queue) tárolja a leveleket, amíg nem tudja azokat továbbítani, vagy postafiókba letenni. Sokszor gondot jelent a nagyméretű levelek ideiglenes tárolása: diszkkapacitást köt le, vagy gond lehet egy-egy olyan csomópont kiesése, ahol nagy számban vannak postafiókok, és nem lehet letenni ezekbe a leveleket. Gondot jelent egy rossz címzés is: sokszor ez csak a cél közelében derül ki, illetve, ha kiderült, értesíteni kell a feladót a kézbesítés lehetetlenségéről, indulnak tehát visszafelé is a levelek (amiket az MTA-k generáltak). Ügyeljünk az etikus viselkedésre!

Egy egyszerű elektronikus levél két részből áll: a *fejrészből* és a levél *testéből* (tartalmából).

A *fejrész* sorai megmondják *kitől*, *kinek*, milyen *tárgykörű* levelet kell továbbítani, milyen ügynökök mikor, hová továbbították (*bélyegzők*) stb. Szerepelhet a fejben *kinek még* jellegű cím is. Egyes levelező ügynökök *csatolt* részeket is képesek a levélhez fűzni, ekkor ez is sze-

repel a fejen. Az egyszerű levelezési szabványban a levél *teste* kizárólag ASCII karakterekből álló szövegsorokat tartalmaz.

Az induló UA állítja elő az induló *fejrészt*, többnyire képes a *testet* is előállítani: felolvassa azt egy fájlból, meghív egy szövegszerkesztőt a tartalom szerkesztésére. A nálunk működő *pine* UA pl. alapértelmezés szerint a *pico* nevű szövegszerkesztőt hívja meg levéltartalom szerkesztésre, de lehet neki más szövegszerkesztőt is adni. A UA képes feladni (send) a megcímzett levelet: ekkor tulajdonképpen átadja egy MTA-nak a levelet.

Az MTA-k a továbbiakban a fejrészt vizsgálják, az kiegészítik és továbbpasszolják a levelet, az utolsó MTA (vagy egy általa hívott processz) leteszi a címzett postafiókjába. (Ha úgy tetszik, az MTA a fejrészből borítékot, borítékokat készít. A borítékban már csak a címzett információi vannak, ha több címzett is van, több boríték is "készül". Az MTA beállításától függően felveszi a kapcsolatot a címzett MTA-val, egyezteti, létezik-e a címzett egyáltalán (igenlő válaszra továbbítja a levelet), vagy csak továbbítja a borítékolt levelet (benne a fejet és testet), és más MTA-ra bízza a címzett létezésének ellenőrzését. Vegyük észre a boríték és a fejrész közti különbséget! Leegyszerűsítve: a fejrészben lehet több címzett, mindegyiknek külön boríték készül!

E-mail címek az Internet-en

(1) Adott egy *gazdagép* (host, cluster) a saját *felhasználói azonosítóival*. Pl.:

gold.uni-miskolc.hu gazdagépen

iitvd bejegyzett felhasználó név (számlaszám).

ekkor:

```
iitvd@gold.uni-miskolc.hu
```

egy *e-mail* cím.

(2) Adott egy *altartomány* (subdomain), a *névszolgáltatója*, (esetleg cluster), és adott egy *felhasználói név* a az *altartományban* (clusterben):

iit.uni-miskolc.hu egy subdomain,

zeus a névszolgáltatója,

vadasz egy felhasználói név a tartományban,

ekkor

```
vadasz@iit.uni-miskolc.hu
```

egy *e-mail* cím. (Ekkor a névszolgáltató levelezési irányító információt is biztosít, ami megmondja, hogy melyik gazdagépen is van a címzett postafiókja.)

Vagyis

```
user-name@host.sub-domain...
```

illetve

```
user-name@sub-domain...
```

alakú lehet egy e-mail cím!

A protokollok

SMTP (Simple Mail Transfer Protocol) RFC-821, 1982, aug.

MIME (Multipurpose Internet Mail Extension) RFC-1521, 1993, szept.

4.5. Az Internet gopher

Minnesotai Egyetem, 1991, nyilvános adatbázis és lekérdező rendszer.

Kliens-szerver filozófiájú.

Gopher szerver: többfeladatos (multi tasking, rendszerint multi user) operációs rendszer alatt futó információ gyűjtő, szolgáltató. Daemon.

Képes:

- hierarchikus struktúrában információkat (lapokat) tárolni,
- kliensektől kiinduló kapcsolatkérésre kapcsolatokat létesíteni,
- kliens kérésére "lépni" föl/le a hierarchikus struktúrán,
- kliens kérésére lapot (fájlt) letölteni a kliens számára,
- kliens kérésére "szolgáltatást" biztosítani.

Gopher kliens: szinte minden operációs rendszer alatt futhat. Felhasználó indíthatja.

Képes:

- kapcsolatot létesíteni adott szerverrel,
- egy-egy "lapot" fogadni és azt "kezelni".

Mi lehet egy "lap"?

- *egy "menü"* ami nem más, mint az adott szinten egy jegyzéklista. Föl/le léphetünk rajta. Kiválaszthatjuk egy elemét. Egy eleme: további "lap".
- *egy szolgáltatás*: Pl. egy keresés, egy átkapcsolás másik gopher szerverre, egy processz elindítása stb.
- egy *fájl*, ami véglevél a hierarchián. Lehet szöveg: ekkor a kliens megjelenítheti (viewer), lehet egy kép, ekkor is megjelenítheti, egy hangfájl, ekkor "lejátszhatja" egy lejátszóval (ha a kliens képes rá), egy video stb. A fájlokat le is töltheti a helyi fájl-rendszerbe, esetleg postázhatja stb.

Ismerjük meg a *gopher világot*! Ismerd meg a tanszék laboratóriumaiban a gopher lehetőségeket!

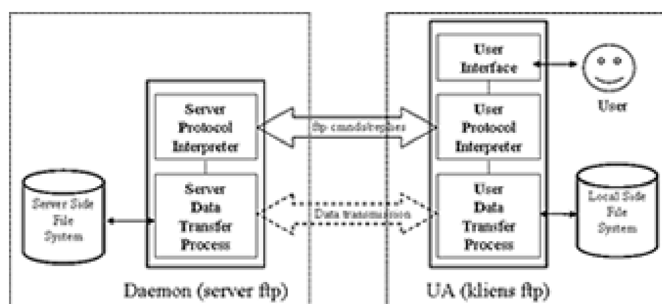
Jegyezzük meg: a kapcsolat a kliens és a szerver között csak addig él, amíg a "lap lejön". A letöltött lapot a kliens tárolja, megjeleníti. Nem terheljük a hálózatot, ha egy-egy lapot sokáig nézegetünk, csakis a kliens gépe erőforrásait használjuk ez alatt. Ez nagy különbség a telenet-es kapcsolatépítéshez képest, hiszen ott a kapcsolat addig él, amíg az ülésünk él. Persze, megtehetjük, hogy telnet-es kapcsolatot építünk arra a csomópontra, amelyiken a gopher szerver fut, és e kapcsolaton futtatunk egy gopher klienst (vannak csomópontok, ahol a *gopher* login névre jelszó nélküli bejelentkezés a gopher klienst indítja). Ekkor a kliens egyszerű processzek közti kommunikációs mechanizmusokkal építi fel a gopher kapcsolatot, a telnetes kapcsolat végig terheli a hálózatot.

4.6. A File Transfer Protocol FTP

Egy ftp kliens indítása:

```
> ftp
```

Kliens-szerver alapú fájltovábbítási (letöltés, felvitel) célú szolgáltatás. A helyi gépen indított ftp kliens kapcsolat létesít a távoli csomóponton az *ftp daemon*-nal (lásd 4.5. ábra).



4.5. ábra. Az ftp kliens-szerver modellje

Kell hozzá számlaszám (login név: uid) a távoli gépen.

Szokásos az : *anonymus* login név az ún. anonymus ftp szervereken. Ezekhez rendszerint nincs jelszó, vagy ha van, az a kezdeményező azonosítását szolgálja.

Az ftp kliens

- "lépegethet" a távoli csomópont fájlrendszerén, lekérdezheti a jegyzékek listáját stb.,
- fájl transzfer valósít meg ide (get) vagy oda (put). Anonymus szervereken rendszerint nem engedélyezik az oda továbbítást .

Szinte minden operációs rendszerben, ahol TCP/IP protokoll működik, van ftp kliens. A Unixok alatt rendszerint van ftp szerver is. Anonymus szerver persze nem mindenütt van.

Nézz utána az on-line kézikönyvben, milyen parancsai vannak az ftp kliensnek.

4.7. Az Archie szolgáltatás

Segít, hogy találjunk *anonymus ftp szervereket* az Internet-en.

Nagyon sok ftp szolgáltató név van a különböző archie szervereken. Tudni kell az archie szerverek (hostok) neveit. (Pl. archie.th-darmstadt.de).

Az archie-t 3 módon érhetjük el:

helyi klienssel: > archie

telnet -üléssel > telnet host + login: archie

e-mail-lel: mail to archie@archie-server, és a levél testében a parancsok (pl. help).

4.8. Hogyan kereshetünk személyeket, számítógépeket?

1) a finger segítségével (lásd: man finger).

2) a WHOIS szolgáltatással.

3) az X.500 ISO szabványú directory szolgáltató lekérdezésével.

4.9. A World Wide Web (WWW) és nézegetőik

A WWW (W^3) *hipermédia* jellegű, az Internet-re kiterjedő szolgáltatás.

A WWW az Internet világban forradalmi változást hozott. Hatására az Internet akadémiai, kutatói hálózatból üzleti és hobby hálózattá vált, szerepet kapott a szórakoztatás világában, a tájékoztató médiák körében, a pénzforgalom és kereskedelem, a reklám világában, az üzleti alkalmazások motorjává vált. Hatása akkora, hogy sokan, mikor az Internet kifejezést meghallják, csakis a WWW világra gondolnak.

A WWW koncepciójában a már jól ismert kliens-szerver koncepció mellett három - tulajdonképpen eddig szintén ismert - paradigma fonódik össze. Ezek a hypertext paradigmája, a hypertext utalások kiterjesztése IP hálózatokra gondolat és a multimédia paradigmája.

A *hypertext paradigma* lényege olyan szövegmegjelenítés, melyben a lineáris vagy a hierarchikus rendszerű, rendezett szöveg olvasás korláta megszűnik. Elektronikus szövegek lineáris olvasásához elegendő egy egyszerű szövegnézegető (viewer). Már a legegyszerűbb szövegszerkesztő is megfelel, melynek segítségével előre, hátra lapozhatunk a szövegben, sőt, egy esetleges kereső (search) funkcióval már-már átléphetünk egy szinttel feljebb, közelíthetjük a rendezett szövegek olvasásához. A rendezett olvasást biztosítanak a szótárprogramok, adatbázis lekérdezők. A hypertext jellegű rendszerekben a szövegdokumentumokban valamilyen szövegrészekhez rögzítettek kapcsolódó dokumentumaik is. A megjelenítő valamilyen módon kiemelten jelenít meg ezeket a szövegrészeket. Ezek a kiemelt részek utalások (kapcsolatok, linkek) más dokumentumokra, más szövegekre, szövegrészekre. A hypertext böngésző nem csak kiemelten jeleníti meg a szövegrészeket, hanem lehetőséget ad azok kiválasztására is (pl. mutatóval rákattinthatunk). A kiemelt rész kiválasztásával az utalt, a hivatkozott (linked) dokumentum betöltődik a nézegetőbe, folytatható az olvasás, természetesen itt ugyancsak lehetnek utalások, akár közvetlenül, akár közvetetten már előzőleg nézegetett dokumentumra is. Az így biztosított információs rendszer jellegzetesen hálós szerkezetű. Léteznek hypertext szövegeket létrehozó, azokat kezelni tudó információs rendszerek, bár jelentőségük a WWW terjedésével egyre szűkebb.

A *hypertext IP hálózatra való kiterjesztése* megszünteti azt a korlátozást, hogy az utalások csak ugyanarra a helyszínre, számítógéprendszerre vonatkozhatnak. Egy-egy kapcsolódó

dokumentum helye a hálózaton "akárhol" lehet, ha az utalások megfelelnek az Uniform Resource Locator (URL) szabványnak.

Végül a *multimédia* paradigma megszünteti a szövegekre való korlátozást: nemcsak hypertext háló, hanem hypermédia háló alakulhat ki. Hivatkozott dokumentum lehet kép, hanganyag, mozgókép, adatfájl, szolgáltatás stb. is. Ráadásul a kép dokumentumokban könnyű elhelyezni további utalásokat is, onnan tovább folytatható a láncolás.

Végül szóljunk a WWW jellegzetes kliens-szerver koncepciójáról is. A WWW kliensek a böngészőprogramok, a tallózók. Képesek a Hyper Text Markup Language (HTML) direktíváival kiegészített szövegek megjelenítésére, bennük az utalásokhoz rendelt szövegrészek kiemelt kezelésére, a kiemelt szövegek kiválasztására. Képesek bizonyos kép dokumentumok megjelenítésére, ezekben kiemelések kiválasztására, hangfájlok, videók lejátszására, vagy közvetlenül, vagy valamilyen segédprogram aktiválásával. A szerverek pedig képesek szöveg-, kép-, hang- és videó-fájlokat megkeresni saját fájlrendszerükben, és azokat elküldeni a kliensnek megjelenítésre. A kliens és szerver között üzenetváltások jellegzetesen négy lépéses forgatókönyv szerint történnek a Hyper Text Transport Protocol (HTTP) szabályozása alatt.

Az első lépés a kapcsolat-létesítés (connection): ezt a kliens kezdeményezi, hozzá legfontosabb információ a szerver azonosítója. A második lépésben a kliens kérelmet (request) küld a kapcsolaton a szervernek, ebben közli, hogy milyen protokollal, melyik dokumentumot kéri (nem részletezzük, de az átviteli eljárás, a *method* is paramétere a kérelemnek). Ezután a szerver megkeresi a kért dokumentumot és válaszol (response): a kapcsolaton leküldi a kért dokumentumot. Végül a kapcsolat lezárul (close). Mindezek után a kliens felelőssége, hogy mit is csinál a leküldött dokumentummal. Mindenesetre ideiglenesen tárolja a saját memóriájában és/vagy fájl-rendszerén, és a dokumentum fajtájától függően megjeleníti azt, esetleg elindítva külső lejátszót, annak átadva dokumentumot közvetve jeleníti meg, lehetőséget ad a felhasználónak végleges lementésre stb. Már a programozás kérdéskörébe tartozik, hogy ha olyan dokumentumot kap a böngésző, melyet közvetlenül nem tud megjeleníteni, lejátszani (futtatni), milyen segédprogramot hívjon meg a megjelenítésre. A felhasználó a MIME szabványoknak megfelelő lejátszókat beállíthat, rendszerint a böngésző konfigurációs menüjében a segítőprogramok (helpers) almenüben.

A manapság legismertebb WWW böngészők nem csak a HTTP protokollt ismerik, hanem más protokollok segítségével nemcsak WWW szolgáltatókkal tudnak kapcsolatot létesíteni, azoktól szolgáltatásokat kérni. Hogy csak a legfontosabbakat említsük, rendszerint képesek ftp protokollon keresztül állomány átvitel szolgáltatások igénylésére (ekkor a kapcsolat végigéli az ftp ülést), telnet protokollal távoli elérésre (ugyancsak végig van kapcsolat az ülés alatt), gopher protokollal gopher szolgáltatás és böngészés végzésére, POP3 protokollal levélszekrények vizsgálatára, letöltésére, SMTP vagy MIME protokollal levelek feladására (kapcsolat levéltovábbító ügynök szolgáltatóhoz), a USENET news levelek olvasására. Mindezekhez viszonylag egységes felhasználói felületet biztosítanak, innen adódik tehát az a téveszme, hogy az Internet az a WWW, vagy fordítva: hiszen egy jó WWW tallózó szinte minden szolgáltatást biztosít, amit az Interneten elérhetünk.

Amit eddig elmondtunk a WWW világról, az még mindig nem biztosítja igazán a programozhatóságot. A WWW szolgáltatóknak rendszerint van még további szolgáltatásuk is. A legegyszerűbb "programozási" lehetőség az, hogy bizonyos szolgáltatók megengedik, hogy különben kommentárnak számító HTML direktíva a szolgáltató parancs-értelmezőjének szó-

ló burok parancs legyen. A szerver elindítja a parancsértelmezőt, végrehajtatja a parancsot, az eredményeit pedig szövegfájl válaszként elküldi a kliensnek megjelenítésre. A Common Gateway Interface (CGI) protokoll szerint akár paramétereket is küldhetünk a kliensből a CGI programnak, a CGI program akár bele is írhat az utoljára megjelenített dokumentumba. Maga a CGI program pedig akármilyen nyelvű is lehet, gyakran egyszerű burokprogramok (shell script), többnyire lefordított és szerkesztett futtaható fájlok. (Ne feledjük: a CGI nem egy programnyelv, hanem egy interfész, azt szabályozza, hogy kap és ad információkat, paramétereket a CGI program.) Leggyakoribb alkalmazási területük a számlálók és vendégkönyvek elhelyezése a WWW nyitólapokon, pontos idő szolgáltatás, keresések a helyi vagy akár távoli WWW rendszerekben, átjárók adatbázis lekérdező rendszerekhez, kérdőívek, szavazólapok kitöltetése, de egyéb programozási megoldásokra is alkalmasak.

A WWW programozási nyelve: a Java

Mint említettük, a WWW böngészőkkel egységes, felhasználóbarát felületet kapott a WWW, ezzel részben az Internet is. A programozás eszközeit - korlátozottan - igénybe lehet venni. A CGI programokkal, melyek a szerver oldalon futnak, bizonyos feladatokat megoldhatunk, bizonyos alkalmazásokat készíthetünk, vagy készíthetnek számunkra. A Sun Microsystem fejlesztői felismerve az eddigi programnyelvek korlátozásait egy tejesen új programnyelvet dolgoztak ki a WWW programozáshoz, a Java nyelvet. Ezzel párhuzamosan a WWW tallózók fejlesztői olyan böngészőt készítettek, amelyik a Java nyelven írt programokat képes értelmezni és futtatni. Az ilyen tallózók *Java virtuális gépként* viselkednek. A HTML dokumentumokban a Java programokra való hivatkozások ugyanúgy találhatók meg, mint a más, pl. kép hivatkozások, és a dokumentum letöltése során akár ezek is letöltődnek. Az a tény, hogy a program nem a szerver oldalon fut (mint a CGI programoknál történik), hanem letöltődik a böngészőhöz és a böngésző hajtja azt végre több előnyt is eredményezett. Egyik előny az, hogy tehermentesítik a szervert, esetlegesen a hálózatot. Másik, talán még nagyobb előny, hogy a nem kell a különböző operációs rendszerekhez, géptípusokhoz illeszteni az alkalmazást, a "szabványos" Java kódot a Java virtuális gép, a böngésző végre tudja hajtani, a böngésző feladata az adott hardver, operációs rendszer adottságaihoz való illesztés. Hátrány is jelentkezik azonban, elsősorban biztonsági kérdések merülnek fel a Java alkalmazások (applet) futtatásánál. Miután a helyi gépen futtatunk, akár bizonytalan eredetű programokat, külön gondot kellet fordítani arra, hogy ne legyen lehetséges vírus- vagy féregprogramokat készíteni a Java nyelv segítségével. Ennek következtében a Java programocskák nem képesek a számukra kijelölt területen túllépni, maguk a böngészők pedig külön kérésünkre további biztonsági szintként nem fogadnak Java alkalmazásokat (amivel el is vesztjük a programozhatóságot). A Java nyelv könnyen megtanulható, különösen C++ ismeretek birtokában.

Tessék használni, felfedezni a WWW világot!

IRODALOM

Barry M. Leiner, *Vinton G. Cerf*, *David D. Clark*, *Robert E. Kahn*, *Leonard Kleinrock*, *Daniel C. Lynch*, *Jon Postel*, *Larry G. Roberts*, *Stephen Wolff* : A Brief History of the Internet,
http://www.isoc.org/internet/history/brief.shtml

5. Hardver architektúrák, a központi egység működése

Láttuk az 1.1. ábrán az általános (Neumann elvből következő) architektúrát. Ezek szerint egy számítógép fő hardver komponensei - megemlítve a feladataikat is - a következők:

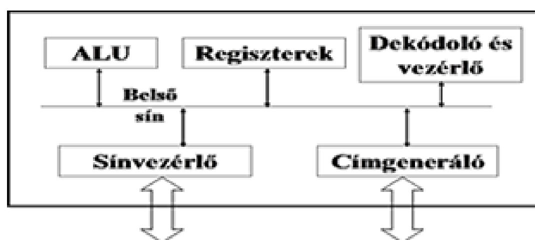
- A központi egység (CPU: Central Processing Unit). A CPU az általános vezérlő, műveletvégző és adatmozgató egység.
- A központi tár (Central Memory), a memória. A programok és az adatok tárolására szolgál.
- A sín, sínek (bus). A sínek adatmozgatást biztosító áramkörök.
- Az I/O perifériák, eszközök (device). Feladatuk a "másodlagos", "harmadlagos" tárolás és a külvilággal való kapcsolattartás.

A *számítógép működése* nagyon leegyszerűsítve és általánosan a következő:

A CPU "veszi" a tárból a soron következő gépi utasítást (Machine Instruction) és az esetlegesen szükséges adatokat. Elemzi az instrukciót és végrehajtja. Eredményét a CPU-ban tartja, vagy mozgatja a memóriába, majd folytatja a működését a soron következő instrukció feldolgozásával. Egyes instrukciók képesek a perifériákat kezelni, ugyanakkor egyes perifériák meglehetős önállósággal is képesek működni. A működés összehangolása a megszakításrendszeren keresztül biztosított.

A következőkben kicsit részletesebben vizsgáljuk az általános strukturális elemeket. Először a CPU-t, annak architektúráját, működését vesszük.

Manapság a CPU rendszerint egy mikroprocesszor, ami egy nyomtatott áramköri lapon van. Lehet persze a CPU maga különböző aktív és passzív elektronikai elemekből álló nyomtatott áramkör is. A funkcionális felépítése nagyon általánosan a következő (5.1. ábra):



5.1. ábra. Egy CPU architektúra

Nézzük a CPU általános strukturális elemeit.

5.1. Az ALU (Aritmetikai logikai egység)

Ha úgy tetszik, ez a CPU - egyben a számítógép - "kalkulátora", ami néhány alapvető műveletet képes végrehajtani.

- Összeadás és kivonás. Kezeli a helyiérték átviteli biteket.
- Fixpontos szorzás és osztás.
- Léptetések (shift), bitek mozgatása jobbra/balra (ami már a fixpontos szorzás/osztáshoz úgyis kell).
- Lebegőpontos aritmetikai műveletek. Ezeket nem minden ALU képes elvégezni. Néha a processzoron kívül, néha azon belül külön komponens végzi ezeket a műveleteket.
- Egyszerű logikai műveleteket.

5.2. A regiszterek, regiszterkészlet

A regiszterek a CPU belső tároló elemei. Tartalmuk gyorsan (a leggyorsabban) és egyszerűen elérhetők a CPU elemei (ALU, dekódoló, stb.) számára. „Munkamemóriát” biztosítanak az ALU számára, ideiglenes tárolást biztosítanak, segítik a címképzést, tárolnak állapotjellemzőket, státusokat (ezzel a vezérlést segítik).

A legtöbb regiszternek van neve - ezeket az assembly programozó használhatja.

Különböző hosszúságúak (bitszélességűek) lehetnek (1 bájtos, 2 bájtos szó stb.), ezeken belül lehetnek „átlapolások”.

A regisztereket többféle módon osztályozhatjuk. A programozási felhasználási lehetőségek szerint vannak a *programozó számára látható* (user visible) regiszterek. Ezeket mind az alkalmazások, mind a rendszerprogramok használhatják. Ezen az osztályon belül a felhasználási mód szerint vannak

- *általános célú regiszterek,* melyeknek felhasználási módjuk nem kötött, melyek a gépi instrukciók argumentumaiban általánosan szerepelhetnek. Vannak ezen kívül
- *speciális célú regiszterek*, melyek korlátozottan használhatók. A korlátozás azt jelenti, hogy csak bizonyos instrukciók argumentumaiként szerepelhetnek.

A *programozó számára nem látható regisztereket* a processzor használja saját működésének kontrolljához.

A kimondottan a felhasználás célja szerint is osztályozhatunk. Ekkor vannak

- *adatregiszterek* (R0-Rxx, AX stb.*)*. Adatelemek tárolására szolgálnak. Az ALU a „kalkulációkat” (részben) az adatregisztereken tudja végrehajtani.
  - Jó, ha sok van belőlük.
  - Különböző hosszúságúak lehetnek, különböző adattárolási formátumuk lehet (8, 16, 32 stb. bites, fix- és lebegőpontos regiszterek).
- Vannak *címregiszterek* is. Adatok és instrukciók memóriabeli címeinek tárolására szolgálnak, a címzés segítik. Több alosztályuk lehet.
  - *Általános célú címregiszterek* azok, melyeknek általában a címzésekkel kapcsolatosan használhatók. Ilyen lehet az *indexregiszter* (az indexregiszteres címzéshez a bázis címhez adandó idex értéket tartalmazhatja), a *szegmensregiszterek* (a tartalmukhoz adandó eltolás érték adja a címet).
  - Az *utasításmutató regiszter* (PC: Program Counter, v. IP: Instruction Pointer), ami mindig a soron következő instrukció tárbeli címét tárolja.
    Saját hardver inkrementációja van, az instrukció feldolgozása során automatikusan növekszik a tartalma (nem szükséges gépi instrukcióval léptetni). Hallatlanul fontos a Neumann elvben! Az „ugrások” (jump, branch) implementációja pedig éppen a PC/IP megváltoztatásával előállítható, és lehetnek utasítások, melyeknek argumentuma éppen a PC/IP.
  - A *verem-mutató regiszter* (SP: Stack Pointer) szintén fontos címregiszter. Miután több szintű veremtár létezhet, több SP is lehet. A „veremkezelő instrukciók” (PUSH, POP) automatikusan hivatkoznak rá és automatikusan állítják.
  - Általában a programozó számára nem látható címregiszterek a virtuális címzéshez használandó *címleképzési táblákat mutató regiszterek.*

- Speciális célú regiszterek az
  - *állapotregiszter(ek)*. A processzor belső állapotát jellemző biteket tartalmaz.
    Ilyen bitek: C - átvitel (carry) bit,
    Z - zero bit,
    S - előjel (sign) bit,
    O - túlcsordulás (owerflow) bit,
    P - paritás bit,
    H - half carry bit stb.
    A jelzőbitek az instrukciók végrehajtása során bebillennek, vagy törlődnek: jeleznek egy-egy állapotot. A feltételes ugró instrukciók éppen a jelzőbiteket használják a feltételre: lesz tehát pl. *jump on Z bit* instrukció.
  - Egyéb *vezérlőregiszter(ek)* is lehetnek. Az üzemmód állapotot (lásd később: felhasználói mód - kernel mód) illetve a megszakítás (interrupt) maszkot tartalmazhatják.
    Sok processzorban ez a két (állapot és vezérlő) regiszter együtt a PSW (Program Status Word), a *program állapot leíró* szó. Néhol e két regiszter és a PC/IP (Program Counter/Instruction Pointer) együtt a PSLW (Program Status Longword), a *program állapot leíró hosszúszó*.

5.3. A vezérlő és dekódoló egység

Feladata a „felhozott" (fetched) gépi instrukció elemzése, dekódolása, és a CPU többi elemének, különösképpen a *végrehajtó egységnek* (ALU és regiszterek, esetleges *védelmi egységnek*) összehangolt működtetése.

5.4. A címképző és buszcsatoló egység

A *címképző egység* alapfeladata az ún. *virtuális címek* leképzése *valós címekre*. Ezt szoros együttműködésben végzi az *operációs rendszer* megfelelő komponenseivel, ha van a processzorban *védelmi egység*, akkor ezzel is. A leképzésekkel az Operációs rendszerek c. tantárgy keretein belül foglalkozunk.

A *buszcsatoló egység* kezeli a sínt (síneket), adatforgalmat bonyolít le.

5.5. A CPU belső sínje, sínjei

Ez a processzoron belüli adatforgalmat biztosító áramkörök összessége. (A sínekről általánosan később még lesz szó.)

5.6. Az utasításkészlet

A gépi utasítások (Machine Instructions) (továbbiakban instrukciók) általános szerkezete:

Műveleti kód	Címrész

Az instrukciók címrésze határozza meg, mi az instrukció operandusa. Természetesen vannak két, esetleg lehetnek három operandusú instrukciók is, ezeknek rendre két illetve három cím-

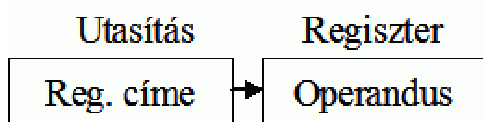
részük van. A CPU architektúra (első értelem!) specifikálja, milyen instrukciókat képes a CPU végrehajtani.

A fejlesztők szempontjai az instrukciókészlet kialakításához:

- Kódsűrűség növelése: adott instrukcióméret mellett minél több instrukció kódolható legyen.
- Ortogonalitás: bármely instrukció mellett bármely címzési mód lehetősége biztosított legyen.
- Szisztematikus kódolás: az instrukcióban lehetőleg egyforma mezők legyenek, melyeknek szerepe minden instrukcióban ugyanaz.
- Kompatibilitás meglévő rendszerekhez: architektúra családokhoz tartózó processzorok így alakulhatnak ki, korábbi programok is használhatók, de túlhajszolása hátráltatja a fejlődést.
- Operációs rendszerek, compiler-ek támogatása.
- Növekvő bitszám, ekkor
  - könnyebb az instrukciók szisztematikus kódolása,
  - gyorsabb az ALU működése,
  - nagyobb a címtartomány.
- Milyenek legyenek az adattípusok (bit, BCD, byte, szó, hosszú szó, lebegőpontos ábrázolások, szöveglánc stb.).

5.7. Címzési módok

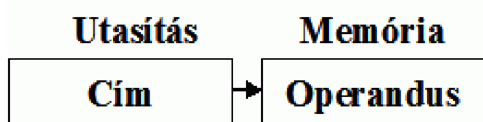
Az instrukciókban lévő címrész többféle címzési móddal határozhatja meg az instrukció operandusát. A „szokásos” címzési módok a direkt regiszter vagy direkt memória címzés, az indirekt memória címzés, az indirekt regiszter címzés (ennek normál és pre/post auto in/dekremens változataival), a bázisregiszteres címzés és a közvetlen címzés.



5.2. ábra. Direkt regiszter címzés

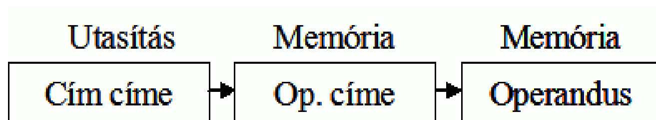
Direkt címzés

- *Direkt regiszter címzés*: címrészben regiszter címe található, a hivatkozott regiszterben pedig az operandus (5.2. ábra). Rövid operációkódot ereményez, egyszerű, gyors az istrukció elemzés, dekódolás.



5.3.ábra. Direkt rekesz címzés

- *Direkt rekesz címzés*: címrészben memória rekesz címe található. A memória cellában az operandus (5.3. ábra). Az instrukció hosszú, dekódolása egyszerű és természetes. Valós címzésű memóriamenedzselés esetén em relokálható a kód.



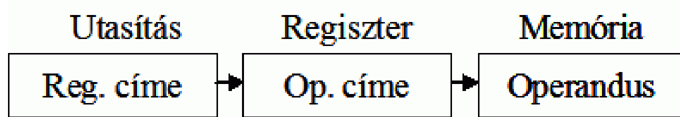
5.4. ábra. Indirekt memória címzés

Indirekt címzés

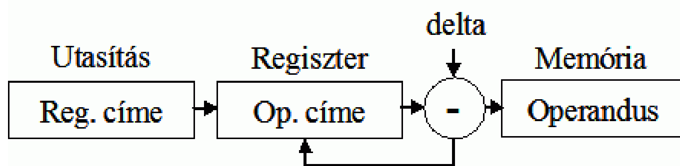
- *Indirekt memória címzés*: címrészben memória rekesz címe, ebben rekeszben az operandus

címe van (5.4. ábra). Hosszú kód az eredmény, valós címzés esetén nem relokálható a kód. Összetett adatstruktúrák kezelésére jó ez a címzésfajta.

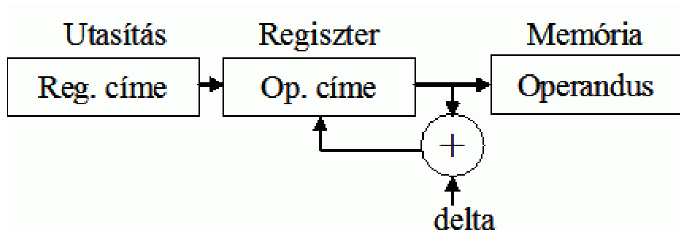
- *Indirekt regiszter címzés*: a címrészben regiszterhivatkozás, a regiszterben az operandus címe. Néha maga az instrukció kód (push, pop) implicite hivatkozik a veremmutató regiszterre (SP-re). Alesetei:
  - - *normál* (5.5. ábra).
  - *pre-auto dekremens címzés*: tipikusan a veremhasználatnál a *push* instrukciónál (5.6.ábra).
  - *post-auto inkremens címzés*: tipikusan a veremhasználat pop instrukciója ilyen (5.7. ábra).



5.5. ábra. Indirekt regiszter címzés



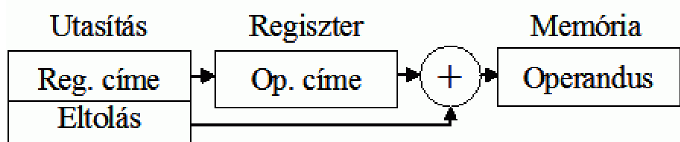
5.6.ábra. Pre-auto dekremens címzés



5.7.ábra. Post-auto inkremens címzés

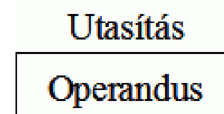
Relatív (bázisregiszteres és közvetlen) címzés

- *Bázisregiszteres címzés*: az instrukcióban regisztercím és eltolás érték van kódolva. A regiszter tartalma egy memória rekesz cím, melyhez adva az eltolás értéket kapjuk az operandus címét (5.8. ábra). Hosszú a kódot eredményez, de a program relokálható (áthelyezhető), hiszen a bázisregiszter változtatásával más kiindulóponttól címezhetünk.



5.8. ábra. Bázistregiszteres címzés

- *Közvetlen* (immediate) *címzés*: tulajdonképpen PC/IP relatív címzés eltolás nélkül, az operandus magában az instrukcióban van, vagyis az instrukció kód után közvetlenül (5.9. ábra).



5.9. ábra. Közvetlen címzés

5.8. Instrukciókészletek, instrukciók csoportjai

Sokféle processzor van, különböző instrukciókészletekkel. Reménytelen lenne mindet felsorolni, megtanulni.

De vannak alapvető fontosságú instrukciócsoportok. Nézzük ezeket, úgy, hogy néhány példát is felsorolunk az egyes csoportokban. A példákban az instrukciók egy elképzelt assembly mnemonikjait adjuk meg, mert ez olvashatóbb, érthetőbb (tehát nem egy elképzelt gépi kódot).

1. Adatmozgató instrukciók

```
LOAD, STORE, LB, LW, SB, SW, ...
MOVE
IN, OUT
```

2. Aritmetikai és logikai instrukciók

```
ADD, SUB
MUL, DIV
AND, OR, XOR, NOT
NEG, COMPL csoport (Komplemensképző)
TEST
COMPARE csoport
```

3. Ugró instrukciók (Jump és Branch)

- feltétel nélküli

```
JUMP, BRANCH
```

- feltételes

```
J(feltétel): JZ, JS, ...
```

4. Bitléptetések, bitforgatás, inkrementáció, dekrmentáció, jelzőbeállítások

```
SHIFT, SLL, SRL, SRA, ...
RCL, RCR        (Rotate L/R carryn át)
ROL, ROR        (Rotate L/R)
INC, DEC
SET csoport
CLEAR csoport
```

5. Eljárás/függgvényhívás, IT hívások, visszatérések instrukciói (CISC)

```
CALL, RET, (LEAVE)
IT, IRET
SYSCALL (RISC)
BREAK, HALT
WAIT
NOP
```

6. Ciklusszervező instrukciók (CISC)

```
LOOP
REP (Repeat stringműveletekre)
```

7. Veremkezelő instrukciók

```
PUSH, PUSHA, ...
POP, POPA, ...
```

8. Társprocesszor instrukciók

```
FINIT   (Társprocesszor inicializálás)
FLD             (Töltés a veremre)
FST             (Leemelés veremről)
FADD, FSUB, FMUL, .. (Aritmetikai instr.)
FWAIT   (Szinkronizációhoz)
```

5.9. Processzorok működési módjai

A korszerű operációs rendszerek működéséhez elvárjuk a korszerű processzoroktól, hogy legyen legalább két - egymástól jól megkülönböztethető - működési módjuk. Ezek szokásos nevei:

- normál mód (v. felhasználói mód),
- védett mód (v. kernel mód).

(Egyes processzoroknak több (egyre privilegizáltabb) üzemmódja is lehet.) Az üzemmódot - mint működési állapotot - a CPU nyilvántartja.

A védett v. kernel mód beállítása mély operációs rendszerbeli feladat, szokásosan ezt a "trap" (csapda) konstrukción keresztül végeztetjük.

Az üzemmódok közötti lényegbeli különbségek:

- védett (vagy több módnál egyre privilegizáltabb) módban szélesebb az instrukciókészlet: azaz bizonyos instrukciókat a CPU csak privilegizáltabb módban tud végrehajtani.
- védett (vagy privilegizáltabb) módban szélesebb címtartományt képes a CPU kezelni: azaz normál (kevésbé privilegizált) módban bizonyos címeket nem "lát" a processzor.

Az üzemmód váltás egyszerű felhasználói programokból nem lehetséges. Ezt csak az operációs rendszer magjának (kernel) hívásával, a trap konstrukción át érhetik el az alkalmazások.

A. függelék: A verem (stack) adattípus formális specifikációja

A formális specifikációhoz meg kell adni:

- a típusokat, amikből az új adattípus épül (konstruálódik);
- az operációkat (operátorokat) definíciószerűen;
- az axiómákat és
- a peremfeltételeket.

Emlékezzünk! Mire kellett egy compilernek a típus?

- Milyen a helyfoglalása és milyen az implementációja (ez együtt megadja az értékkészlete).
- Milyen operátorok vannak a típuson, és ezek hogy hatnak. (Ez a szintaktikus ellenőrzéshez is kellett:)

Típusok a verem adatszerkezethez:

```
Type    X:    akármilyen ismert típus;
        B:    boolean;
        S[X]: X-ek verme;
```

Funkciók:

```
new(x: X)                  -> S[X]  ! létrehozás
empty(s: S[X])   S[X]   -> B     ! üresség vizsgálat
full(s:S[X})     S[X]   -> B     ! teliség vizsgálat
push(x: X)       x,S[X] -> S[X]  ! x-et verem tetejére
pop(s: S[X])     S[X]   -> X     ! tetejéről levétel
grow(s: S[X])    S[X]   -> S[X]  ! verem növelése
delete(s: S[X])  S[X]   ->       ! verem megszüntetése
```

Axiómák: ∀(x: X, s: S[X])

```
    empty(new(x))
not empty(push(x))
    pop(push(x))
```

Peremfeltételek (preconditions) ∀(x: X, s: S[X])

```
pre pop(s)      not empty ( s )
pre push(x)     not full ( s )
pre delete(s)   empty ( s )
pre new(x)      van hely a heap-ban
pre grow(s)     van hely a heap-ban
```

6. Processzor teljesítmény növelés

6.1. A processzorok ciklusideje

Minden mikrolapka - legyen az mikroprocesszor vagy különleges funkciókat ellátó integrált áramkör - tulajdonképpen hatalmas mennyiségű tranzisztor olyan együttese, amelyeket a feladatok elvégzéséhez különbözőképpen összekapcsolnak. Manapság több millió tranzisztor helyezhető el egy-egy lapkába.

A bináris információk előállítása tulajdonképpen a lapka bizonyos pontjai magas ill. alacsony feszültségszintjeinek előállításából (vagy a feszültségszintek változásának előállításából) áll. A mikrolapka áramköreiben a jelek átalakítása, lefutása viszont időigényes.

A mikrolapka - köztük a CPU - ún. ciklusokban működik, valamilyen frekvenciával. A ciklusok közben tranziens állapotok vannak, a ciklusok végén állnak be a 0, 1 bit információt hordozó feszültségszintek vagy szintváltozások.

A mikroprocesszor valamilyen óraeszközről leosztva kapja a meghajtó frekvenciát, amivel működik. Így pl. beszélhetünk 25 MHz-es Intel 386-os mikroprocesszorról, 1,6 GHz-es Pentium 4-ről stb. Minél magasabb a frekvencia, annál „gyorsabb” a mikroprocesszor.

Egy-egy gépi instrukció végrehajtására egy, két, néhány tíz, vagy akár néhány száz ciklusra is szükség lehet.

6.2. Processzor teljesítmények (Processor performance)

Egy CPU teljesítménye mérhető azzal az idővel, ami egy meghatározott feladat (program, algoritmus, benchmark test) teljesítéséhez szükséges.

idő-per-feladat = C * T * I

ahol:

- C: utasításra eső ciklusok száma,
- T: ciklus ideje (óra sebesség),
- I: feladatra eső utasítások száma.

Bármelyik tényező csökkentése növeli a teljesítményt.

6.3. Teljesítmény értékelési módok

6.3.1. MIPS: Million Instruction per Second

A legegyszerűbb processzor teljesítmény jellemző az az érték, hogy egy másodperc alatt hány millió gépi instrukciót hajt végre a processzor. A mérték a $MIPS_i$: az i. utasítást 1 sec alatt hány milliószor tudja végrehajtani a processzor.

$MIPS_i = 1 / (T * C_i)$

Egy példa: 32 MHz 80386 processzornál a T = 31.25 nsec, és ezzel

i	C	MIPSi
Register to Register instrukció	2	16 MIPS
Register to Memory instrukció	7	4.5 MIPS
Protected Mode Task Switch i	270 - 300	0.1 MIPS

Miután az egyes instrukciók végrehajtásának ciklusszáma nem egyforma, belátható, elég nagy az eltérés az egyes utasítások MIPS értékei között. Amikor egy processzort jellemezni akarunk, gondot jelent, melyik instrukciójának MIPS értékét adjuk meg. A gyártók (propaganda célokból) a legkedvezőbb értékeiket közlik, esetleg „átlag" MIPS-et adnak meg (ha igazán becsületesek, valahogy meghatározzák az utasítások valószínűségét, és ezzel súlyozott átlagot adnak). Így. pl. a gyártója 4.5 MIPS-et állít erről a CPU-ról.

A MIPS egyszerű és népszerű sebesség jelző. Miután a processzor specifikációkban adottak lehetnek az instrukciókra eső ciklusszámok, továbbá adottak a processzorok működési frekvenciái, nem is kell mérni a MIPS értéket, az akár számítható is.

Gondot jelent, hogy a felhasználói programok instrukció-statisztikái (a „workload") ritkán egyezik a gyártók utasítás-gyakoriságaival! Ráadásul más az instrukció-gyakoriság tudományos számításokra, más adatbázis kezelésre s.í.t. További gond, hogy a MIPS jellemzővel az architektúrák nem vethetők össze igazán. (Pl. az egyiknél van „cache" használat, a másiknál nincs, egyiknél van „pipe-line", másiknál nincs).

6.3.2. Korszerűbb sebesség-meghatározások

Adott típusú (integrális aritmetikai, lebegőpontos aritmetikai, tranzakciós stb.) feladatosztályokhoz *szabványos* (standard) terhelés-osztályú *mérőprogramokat* (ún. benchmarkokat) alakítanak ki, és ezeket futtatják, ezek futási idejét mérik, ebből állapítanak meg „sebességértékeket" (instrukció per sec értékeket). Ugyanazzal a mérőprogrammal előállított teljesítményértékek már összehasonlíthatók, ezzel a különböző gépek is összevethetők. Néhány elfogadott benchmarkot ismertetünk.

6.3.3. Whetstone Benchmark

Kidolgozója a National Physical Laboratory in Whetstone, Great Britain, 1970-76.

Kisméretű, mérnöki és tudományos programokat reprezentáló benchmarkot állítottak elő single- és double precision FORTRAN programként. Az eredményeket KWIPS-ben (thousands of Whetstone instruction per second) mérik.

6.3.4. Livermore Loops Benchmark

Kidolgozó a Lawrence National Laboratory in Livermore, CA. (1980-86)

Nagy, szuper-számítógép rendszerek benchmark-ja. Úgy is ismert, mint Livermore FORTRAN Kernel. Alkalmas széleskörű tudományos feladatokhoz, beleértve az I/O-t, grafikát és a memória menedzselési feladatokat.

24 rutinja van, az eredményeket MFLOPS-okban adják meg (Millions Floating-Point Operation per Second).

6.3.5. Dhrystone Benchmark

Kidolgozója Dr. Reinold Weicker, Siemens (1984), volt, ADA programként írta meg. Rick Richarson (1986) C nyelvre átírta.

A benchmark a processzor és a fordító (compiler) hatékonyságát is méri. Rendszerprogramozási környezetet reprezentál. Mértékegysége a Dhrystone instruction per second.

Megkülönböztetünk Dhrystone V1 és V2 változatokat. A V1 tartalmaz „soha nem futó" utasításokat (dead code) is, melyeket a fordító (optimizing compiler) kiszűrhet. A V2-ben minden utasítást fordítanak, futtatnak: egy jó fordító magasabb Dhrystone rátát ad a V1-re, mint a V2-re.

6.3.6. Linpack 100 *100 és 1000 * 1000 Benchmark

A Linpack benchmark FORTRAN-ban írt lineáris egyenletrendszer megoldó program. Nagy mátrixok lebegőpontos összeadásait és szorzásait tartalmazza (1981). Az eredményeket MFLOPS-ban adja.

6.3.7. TPC Benchmark A

Kidolgozója a Transaction Processing Performance Council (1988).

Kereskedelmi tranzakció feldolgozó rendszerekhez kidolgozott teljesítmény mérő rendszer. Olyan komponenseket tartalmaz, amelyek on-line tranzakció feldolgozási környezet feladatainak mérését biztosítják, hangsúlyt helyezve az adatbázis szolgáltatásokra (ezen belül is az intenzív update-ra). Az on-line tranzakció feldolgozási környezetek (On-Line Transaction Processing: OLTP environments) a következőkkel jellemezhetők:

- többszörös terminál ülések (multiple on-line terminal sessions) léteznek,
- jelentős a diszk I/O,
- mérsékelt a rendszer és alkalmazás végrehajtási idő,
- a tranzakció integritás biztosítandó.

A TPC-A mérőszámai a tpsA, illetve a $/tpsA. A TPC Benchmark A specifikáció előírja, hogy az eredmények mellett közölni kell a mért rendszer részletes HW és SW leírását is (Pl. DEC 3000/800S AXP C/S hardveren DEC OSF/1 AXP V13A-4 operációs rendszer mellett ORACLE V7.0.13 adatbázis kezelővel a tpsA=186.02, ill. a $/tpsA=6.503).

6.3.8. Dongarra teszteredmények

Dongarra jelentése számítógép teljesítményeket hasonlít össze lineáris egyenletrendszereket megoldó programokkal. Az összehasonlításba mintegy száz gépet vett be, a CRAY Y-MP szuperszámítógéptől kezdve, az Apollo és Sun munkaállomásokon át egészen az IBM személyi számítógépekig.

Az összehasonlításhoz használja a

- LINPACK 100*100 Benchmark-ot (eredmények Mflops/s-ban),

- a TPP-t (A LINPACK 1000*1000 egy módosított változatát) (eredmények Mflops/s-ban) és az
- "elméleti csúcs teljesítmény" (theoretical peak performance) értéket (Mflops/s).

Ezekből az elméleti csúcs teljesítményt kell magyarázni. Ez azon lebegőpontos összeadás és szorzás instrukciók száma, melyeket 1 sec alatt képes a gép elvégezni. Ezt az értéket a számítógépgyártók gyakran emlegetik, és ez tulajdonképpen a teljesítmény felső határát reprezentálja: garantált, hogy ezt az értéket nem haladhatják meg a valós programok (ha úgy tetszik, ez az adott gép "fény sebessége").

Például, a CRAY Y-MP/8 ciklusideje 6 nsec. Egy ciklus alatt 2 összeadást, ill. 2 szorzást képes egy processzor elvégezni. Ezzel

egy_proc_elm_csúcs_telj = (2 operáció/1 ciklus) * (1 cikus/6 nsec) = 333 Mflops/s.

Mivel a CRAY Y-MP/8 8 processzort tartalmaz, az elméleti csúcs teljesítménye: 2667 Mflops/s.

6.3.9. SPEC Benchmark Suites (SPEC készletek)

SPEC: Standard Performance Evaulation Corporation, 1989-ban a HP, a MIPS, Sun és az Apollo alapította. 1990-ben az IBM, Fujitsu, Siemens, DEC, Intel, Motorola, AT&T, Unisyc csatlakozott.

A testület módszere az, hogy alkalmazás orientált, objektív teszteket (workload-okat) határoztak meg, melyek futtatási idejét egy közös referencia gépen való futási időkhöz viszonyíthatunk. A módszer alkalmas különböző rendszerek teljesítményeinek összevetésére, megmondja ugyanis, hogy egy-egy teszt hányszor gyorsabban fut le egy-egy vizsgált gépen, mint a viszonyító gépen. Kezdetben 10 tesztet (real workload-ot) állítottak össze, melyek elég nagyok (nem lehet az egészet a gyorsítótárból (cache) kiszolgálni!).

Egy-egy teszt minősítő mértéke az ún. SPEC_ratio volt:

SPEC_ratio = futási_idő/DEC_VAX-11-780_futási_idő

azaz, referencia gépnek a DEC VAX-11-780-as gépét választották. (Kérdés merülhet fel, miért ezt a gépet választották? Nos. ez egy nagyon elterjedt gép volt, és ez volt az első ún. 1 MIPS-es gép!)

Amikor egy gépet jellemezni akartak, meg kellett adni a 10 SPEC_ratio értéket, illetve ezek geometriai átlagát, ami az ún. SPEC_mark érték.

Egy példa: IBM RISC/6000-930
Min. SPEC_ratio = 17.5
Max.SPEC_ratio = 76.1
SPEC_mark = 28.9

1992 januárjában a SPEC bejelentett két új benchmark készletet: a CINT92-t és a CFP92-t. A SPEC CINT92 jól jellemezte a kereskedelmi, üzleti számítástechnikai rendszereket, a SPEC

CFP92 pedig a mérnöki, tudományos alkalmazásokhoz alkalmas rendszerek teszt-készlete volt (intenzív lebegőpontos számítások).

A SPEC CINT92 készletbe 6 teszt tartozott, ezek geometriai átlaga a SPECint92 szám. A készlet elemei:

008. espresso-Circuit theory

022. li-LISP Interpreter

023. eqntott-Logic design

026. compress-Data compression

072. sc-UNIX spreadsheet

085. gcc-GNU C compiler

A SPEC CFP92 készletbe 14 valós alkalmazáshoz hasonló teszt tartozik. Kettőt C-ben, a további 12-t FORTRAN-ban írták. A 14 teszt ratio geometriai átlaga a SPECfp92. A készlet elemei:

013. spice2g6-Circuit design

015. doduc-Monte Carlo simulation

034. mdljdp2-Quantum chemistry

039. wave5-Maxwell equations

047. tomcatv-Coordinate translation

048. ora-Optics ray tracing

052. alvinn-Robotics, neural nets

056. ear-Human ear modeling

077. mdljsp2-Single precision version of 034.mdljdp2

078. swm256-Shallow water model

089. su2cor-Quantum physics

090. hydro2d-Astro physics

093. nasa7-NASA math kernels

094. fpppp-Quantum chemistry

1995-ben - a kor igényeinek megfelelően - megújították a SPEC összehasonlítási rendszert. A SPECint95, vagy egyszerűen CINT95 a CINT92 továbbfejlesztése, és természetesen megújult a SPECfp (CFP95) is. A viszonyító gép (amire „normalizálnak") a Sun SPARCstation 10840 gép lett (40MHz-es SuperSparc processzorral, L2 Cache nélkül).

A SPECint95-höz, SPECfp95-höz tartozik egy *alap* (baseline) minősítő szám is, ennek nevei rendre SPECint_base95, SPECfp_base95. Az SPECxxx_base95 minősítés és a SPECxxx95 (baseline) minősítés közötti különbség tulajdonképpen az, hogy *baseline* esetén a teszteket "konzervatív" optimáló fordítóval fordítják, míg az egyszerűnél agresszív optimáló compilerrel.

A CINT95 tesztjei a következők:

099. go - AI, go játékprogram

124. m88ksim - Moto88K chip szimulátor

126. gcc -GCC egy verziója

129. compress - (memóriában) fájl tömörítés/kitömörítés

130. li - LISP interpreter

132. ijpeg - grafikus kompresszió/dekompresszió

134. perl - szöveg manipulálás (anagrammák, prímszámok)

147. vortex - egy AB program

A SPECint95 a fenti 8 teszt normalizált eredményének (alapgéphez viszonyított hányad) geometriai átlaga, agresszív fordítás mellett. SPECint_base95: a 8 teszt eredményének (a viszonyszámnak) geometriai átlaga, konzervatív (normális, nem különleges) optimálás mellett.

A SPEC95-ben 10 lebegőpontos tesztet határoztak meg. Ezek:

101. tomcat - hálógeneráló

102. swim - 1024*1024 griden hullámzó víz modell

103. su2cor - Monte-Carlo szimuláció quantumfizikában

104. hydro2d - hidrodinamikai egyenletek megoldása

107. mgrid - 3D feszültségmező számítások

110. applu - parciális diff. egyenletek megoldása

125. urb3d - csőben turbulencia szimulálása

141. apsi - meteorológiai szimuláció (hő, szél, pollenszennyezés)

145. fppp - quantum-kémiai problémamegoldás

146. wave5 - elektromágneses tér szimuláció (plazmafizika)

Ezek után a SPECfp95: a 10 teszt normalizált eredményének (alapgéphez viszonyított hányad) geometriai átlaga, agresszív fordítás mellett. A SPECfp_base95: a 10 teszt eredményének (a viszonyszámnak) geometriai átlaga, konzervatív (normális, nem különleges) optimálás mellett.

Megjegyezzük, hogy a SPECxxx95 mérőszámok ún. *sebesség* mérőszámok: azt mondják meg, hogy a célgép milyen gyorsan old meg feladatotokat, vagy hányszor gyorsabban oldja meg a feladatokat az alapgéphez viszonyítva. Ezek a mérőszámok alapvetően egy processzoros gépek jellemzésére, összehasonlítására alkalmasak csak! Nem alkalmasak több processzoros gépek összevetésére.

6.3.10. Többprocesszoros gépek összevetése

A SPEC 1992-ben bejelentette, 1995-ben továbbfejlesztette a többprocesszoros gépek összehasonlítására is alkalmas mérési rendszerét. A SPECxxx_ratexx mérőszámok nem sebességet, hanem munkavégzési képességet mérnek (viszonyítva persze az alapgéphez). A módszer a teszt-program szövetre alapozva azt méri, hogy *adott idő alatt mennyi munkát tud a rendszer elvégezni*. Az új mérték a SPECrate, ez kapacitás mérték, nem azt adja, milyen gyorsan tud elvégezni a rendszer valamilyen feladatot, hanem, hogy mennyit tud elvégezni egy feladatból egy adott idő alatt. Egyprocesszoros rendszeren a „sebesség” és a „munkavégző képesség” mérőszámok persze ugyanazok lesznek. Többprocesszoros rendszerek összehasonlítására a munkavégzési képesség (a rate) mérőszámok a helyesek.

Egy teszt SPECrate-jának kiszámítási formulája:

SPECrate = #CopiesRun * ReferenceFactor * UnitTime / Elapsed ExecutionTime

A SPECint_rate92 a CINT92 6 benchmark SPECrate-jának geometriai átlaga, míg a SPECfp_rate92 a CFP92 készlet 22 teszt SPECrate-jának geometriai átlaga. A mostani mérőszámok pedig megvannak a konzervatív és az agresszív optimálással is, így ismerünk SPECint_rate95-öt, SPECint_rate_base95-öt, SPECfp_rate95-öt, végül SPECfp_rate_base95 mérőszámot is.

6.3.11. A SPEC teljesítménymérés továbbfejlesztése

2000-ben ismét továbbfejlesztették a SPEC teljesítménymérést. A CINT2000 szerint 12 terhelésosztály létezik, ezek geometriai átlagával jellemeznek, 4 metrikában:

SPECint2000 (csúcs) sebesség, erős optimáló fordítással

SPECint_base2000 sebesség, konzervatív optimáló fordítóval

SPECint_rate2000 munkavégzési képesség (throughput), erős optimáló fordítóval

SPECint_rate_base2000 munkavégzési képesség, konzervatív fordítással

A CFP2000 szerint 14 lebegőpontos terhelésosztály van, itt is 4 metrika használható:

SPECfp2000 (csúcs) sebesség, agresszív optimálás a fordításban

SPECfp_base2000 sebesség, konzervatív optimálás

SPECfp_rate2000 munkavégzési képesség (throughput), erős optimáló fordítóval

SPECfp_rate_base2000 munkavégzési képesség, konzervatív fordítással

Néhány - különböző - teljesítmény összehasonlítási lehetőséget találnak az alábbi címen:

The Benchmark Gateway: http://www.ideasinternational.com/benchmark/bench.html

6.4. A CPU teljesítmény növelés módszerei

Nem strukturális- ezzel a legegyszerűbbnek tűnő – módszer az órajel frekvencia növelése (T csökkentése). Ma már kaphatók 1,6-2 GHz-es processzorok (lásd: Néhány 64 bites processzor jellemzői c. táblázat). A frekvencia növelésének a különböző technológiákban fizikai határai vannak.

Nem strukturális módszer az instrukciók számának csökkentése a feladathoz (I csökkentés): az optimált fordítás (compilálás). Természetesen használják ezt a módszert. Természetes az is, hogy programokban a elemzik a nagyon gyakran futó kódrészleteket, és elemzik azokat, vajon lehetne-e hatékonyabban megírni ezeket.

Már a strukturális módszerekhez tartozik az utasításokra eső ciklusok számának csökkentése (C csökkentés). A processzorok fejlesztésében törekednek az utasításokra eső ciklusok számának csökkentésére. Ezt leggyakrabban valamilyen párhuzamosítási technikával segíthetik elő.

A ciklusszám csökkentése, a frekvencia növelése versenyben a CISC és a RISC architektúrájú processzorok a mai napig vetélkednek.

6.5. A CISC és a RISC architektúrák.

6.5.1. CISC: Complex Instruction Set Computer

Történelmileg a számítógépiparban a CISC architektúrájú gépek domináltak. A piac nyomására, hogy megőrizzék a kompatibilitást, megtartva a régi utasításkészletet, egyre bonyolultabb gépi instrukciókat vezettek be a CPU családokon belül, ahol is a sokféle instrukcióval kényelmes gépi kódú programozás lehetséges, és megfelelő hatékonyságú kódot lehet generálni a magas szintű nyelveken írt programokhoz is. A nagy instrukciókészlet viszont nagy belső mikroprogram tárat igényel. A CISC jellemzői:

- Sok, akár néhány száz, közöttük több összetett instrukció: sokminden a mikrokódban, amit a VLSI technika lehetővé is tesz. A CISC gondolat támogatói azzal érvelnek, hogy ha minél többet bízzunk a hardverre, annál nagyobb lesz a teljesítmény.

- Bonyolultabb címzési módok lehetségesek, emiatt viszont
- változó hosszúságúak az instrukciók. Ez mint látni fogjuk, nehezíti a futószalag (pipeline) feldolgozást.
- A gépi instrukciók változó ciklusidőt, a komplexebbek meglehetősen nagy ciklusszámot igényelnek. Ez is nehezíti az átlapolt feldolgozást.
- Az assembly programozás feltétlenül egyszerűbb: a komplex instrukciók valóban komplex feladatokat oldanak meg. A compilerek írása is könnyű.
- Viszonylag kevés regiszter van.

Neves CISC processzorok az Intel 286/386/486 és a Pentiumok, a Motorola 68000 család processzorai, a DEC VAX processzora stb.

6.5.2. RISC: Reduced Instruction Set Computer

Újabb felfogás szerint a teljesítmény növelhető *redukált instrukció-készletű processzorokkal*, ahol viszont a hardver, a firmware és szoftver között sokkal kifinomultabb és igényesebb együttműködés lehetséges. A koncepció statisztikai felmérések alapján merült fel. Azt vizsgálták, hogy a szoftverek hogyan használják a processzor erőforrásait. Kiderült, hogy az egyszerűbb instrukciók túlnyomórészt dominálnak még a CISC architektúrákban is. Hiába implementálták a komplex instrukciókat, azokat ritkán használják. Egy csökkentett instrukció-készletű processzor, ami tipikusan 50-80 instrukciót jelent, és amelynél szemben a CISC felépítéssel az instrukciók dekódolására fix logikát alkalmaznak, nagyságrenddel nagyobb ütemezési sebességgel tud dolgozni. Az amúgy is domináló egyszerű instrukciók mellet a felmerülő komplexebb feladatok - kicsivel több kóddal , de optimált fordítással segítve - azért elvégezhetőek maradnak. Valószínűleg hosszabb lesz a kód, de a gyorsító-tár (cache) ezen is segíthet, a háttértároló kapacitás pedig egyre kevesebb gond a fejlődés során. A RISC fejlődést teszi lehetővé az a tény is, hogy a gyorsító memóriák (cache) is fejlődnek, a processzor mikrokód helyettesíthető az egyre gyorsabb cache memóriákkal.

A RISC architektúrák jellemzői:

- Csak a legalapvetőbb instrukciók léteznek gépi szinten, viszont
- meglehetősen sok regiszter van. Ennek előnye nyilvánvaló, kevesebb a tárművelet, sok a regiszterművelet..
- fix a kódhosszúság, egyszerűek a címzési módok: biztosított az ortogonalitás és a szisztematikus kódolás és ezek miatt
- egyszerű a dekódolás és gyors is. Az utóbbi három pont eredménye, hogy a C kicsi!
- A RISC processzorokat eleve az operációs rendszerekhez és a compilerekhez igazítottan tervezik. Ezért az I kicsi.
- Az egyszerű instrukciók nemcsak egyforma hosszúságúak, hanem egyforma ciklusidőt igényelnek, ezért az ún. futószalag (pipe-line) feldolgozás könnyű. Ez azt jelenti, hogy kicsi lehet a T.
- Egyetlen hátrány látszik: a bonyolultabb feladatokat instrukció-szekvenciákkal kell megoldani, ez a programok méretét, - hosszát - növelheti.

Híres RISC processzorok

A DEC cég 21064 Alpha processzora 200 MHz-es órajellel 400 MIPS-re, 200 MFLPOS-ra képes.

A MIPS cég (a céget utóbb felvásárolta az SGI) R3000 RISC processzora működött a tanszék Silicon Graphics Indigo munkaállomásaiban. Az SGI Power Series (az ős zeus) 4 db. R3000 processzort tartalmazott. Az Indi2 processzora az R4000, az O2 gépeké az R5000. Már R12000-es processzorok is vannak.

A Hewlet Packard híres RISC processzorai a PA-RISC processzorcsalád tagjai. A HP-9000 munkaállomások mind PA-RISC architektúrájúak.

A SUN SPARCstation munkaállomások, SPARCserverek processzorai az Ultra SPARC RISC processzorok.

Az IBM is előrukkolt; ismerjük a RISC/6000 POWER processzorcsaládját (gold, silver), a PowerPC processzorát.

A RISC processzorok részesedése a csaknem 30 milliárd $-os piacon:

	Mrd $			%	
	'93	'94		'93	'94
PA RISC	7.5	9.6		33.7	32.7
Sparc	5.0	5.8		22.5	19.7
PowerPC	2.4	5.4		10.8	18.4
MIPS	4.5	5.2		20.2	17.7
Alpha	0.3	1.5		1.3	5.1
egyéb	2.6	1.8		12.8	6.1

Néhány 64 bites RISC processzor jellemzői

Mikroprocesszor	Alpha 21164	MIPS 10000	PowerPC 620	UltraSparc
Gyártó	DEC	MIPS	IBM&Motorola	SUN
A lapkán lévő adat/instr. cache, KB	8/8 primary96 secondary	32/32	32/32	16/16
Címtartomány	2^{40}	2^{40}	2^{40}	2^{41}
A független egységek száma (int, fp stb)	4	5	6	9
Max instr./ciklus	4	4	4	4
Instr. átrendezés	Nem	Igen	Igen	Igen
Előrehozott elágazás figyelés	Nem	Igen	Igen	Igen
Óra sebesség, MHz	300	200	133	167
Memória busz szélesség	128	64	128	128
Becsült SPECint92	330	300	225	275
Becsült SPECfp92	500	600	300	305
Tranzisztorok száma, mill	9.3	6.4	7	3.8

Teljesítményfelvétel, W	50	30	30	30

6.6. Párhuzamos architektúrák, átlapolás

A CPU feldolgozási sebesség egyik növelési módszere a párhuzamosítás. A párhuzamosítási lehetőségeknek legalább két szintje van: párhuzamosítás a processzoron belül; és párhuzamosítás azon kívül.

6.6.1. Párhuzamosítás a CPU-n belül: futószalag (csővezeték, pipe-line) feldolgozás

A C (szükséges ciklusok száma) csökkentésének egyik módja az a felismerés, hogy átlapolhatók az instrukciók feldolgozási fázisai, és az átlapolt fázisok párhuzamosan végrehajthatók.

A gépi utasítás végrehajtása több fázisból áll (ezt már a korábbiakban láttuk). Mondjuk, egy elképzelt processzoron egy utasítás három fázisban hajtódik végre (ilyen pl. az M68000):

- utasítás felhozatal (fetch: F) fázis az első,
- dekódolás (decode: D) fázis a második, végül
- az ALU végrehajtja az instrukciót az harmadik (execute: E) fázisban.

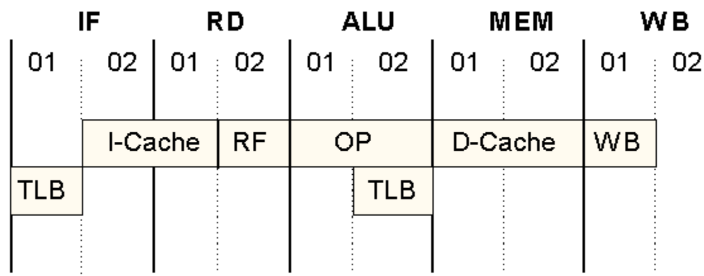
Miután az egyes fázisokat a CPU különböző részegységei hajtják végre, az egyes fázisok egymással párhuzamosan hajthatók végre: az első instrukció E fázisa alatt a második instrukció D fázisa és a harmadik instrukció F fázisa:

Instr.\Cikl.	i-2	i-1	i	i+1	i+2
1.	F1	D1	E1		
2.		F2	D2	E2	
3.			F3	D3	E3

Ezzel, bár egy instrukció végrehajtására több ciklus kellhet, a ciklusok számát csökkenthettük az *előfelhozással, elődekódolással*.

A futószalag (szupercső, super pipe-line) alapgondolata tehát *párhuzamosítás*, annak kihasználása, hogy a feldolgozás különböző fázisait autonóm és párhuzamosan működő alrendszerek végezhetik.

Az R3000 processzor futószalagja



6.1. ábra. Egy instrukció végrehajtása

A MIPS cég processzora az instrukciók végrehajtását 5 fokozatra (stage) osztja, minden fokozatot további 2 fázisra. Egy fokozat végrehajtása egy ciklus alatt történik. A fokozatok:

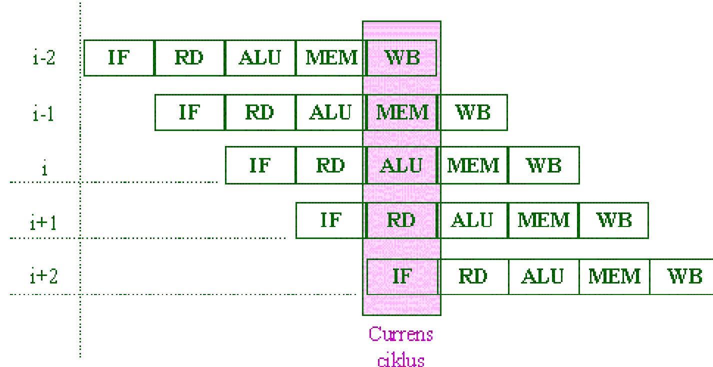
IF: instrukció felhozatal;
RD: olvasások;
ALU: ALU operációk;
MEM: adatmemória elérés;
WB: regiszter visszaírás.

A 6.1. ábrán bemutatjuk egy instrukció végrehajtásának menetét.

A fokozatokban, ezek fázisaiban a tevékenységek:

IF	01	TLB-t használva virtuális címet fizikaira képez
IF	02	Leképzett címet küldi az I-Cache-nak
RD	01	Felhoz az I-Cache-ből, ellenőriz
RD	02	Olvassa a regiszter-fájlt, vagy ugráshoz számítja az ugrás címet
ALU	01+02	Aritmetikai instrukciónál végrehajt, vagy
ALU	01	Ugrásnál dönt, lehet-e ugrani, vagy LD/ST-nél számítja az adat virtuális címét
ALU	02	LD/ST-nél a TLB-t használva leképez fizikai címre
MEM	01	LD/ST-nél küldi a fizikai címet a D-Cache-nek
MEM	02	LD/ST-nél mozgatja az adatot, ellenőriz.
WB	01	Írja a regiszter-fájlt

Még egy ábrát bemutatunk (6.2. ábra), ez az R3000-es processzor 5 mélységű csövét mutatja.



6.2. ábra. Az R3000 5 mélységű csöve

Milyen problémák merülhetnek fel a szupercsövezéseknél?

Az időzítési hazard: mikor egy instrukció az előző instrukció eredményét használná, de az még késik. Megoldása az instrukciók ügyes átrendezésével lehetséges (assembly programozói vagy fordítói feladat, vagy a processzor képes "instrukció átrendezésre"!).

Az ugró utasítások problémája: ilyenkor a "csövet" ki kell

üríteni, mert az instrukciófolyam egészen máshol folytatódik, mint amit a csőbe felhoztunk. A futószalag újratöltése időveszteség. Megoldások:

- *jump* instrukciók után *nop* utasítások elhelyezése (az időveszteség így persze még megmarad).
- *késleltetett ugrás:* minden ugró utasítást az eredeti helyéhez képest kisebb című helyekre teszünk, (ahány fokozata van a futószalagnak). (Ember legyen a talpán, aki ezt követni tudja majd!)
- *lookahead pipe line vezérlővel (előrehozott elágazás figyelés)*: ez intelligens hardver komponens, ami szoftver támogatást nem kíván. Figyeli az ugró instrukciókat és előre beolvassa a lehetséges elágazások mentén az instrukciókat, majd arra az ágra kapcsol, mely szükséges. (legjobb, de nem minden processzorban van ilyen.)

A különböző ciklusidejű instrukciók megzavarják az egyes fázisok szinkronizációját. Ez főleg a CISC-nél fordul elő! Megoldások:

- A RISC processzorokat szinte erre tervezték!
- CISC processzoroknál *nop*-ok beiktatása (időveszteség!)
- "reteszek" beiktatása a futószalag fokozatok közé: egy instrukció csak akkor léphet a következő fokozatba, ha az ahhoz tartozó retesz nyitva áll, de a retesz zárva marad, amíg a fokozatban utasítás tartózkodik.

A megszakítások és kivételes állapotok kezelése is probléma: a futószalag helyzetét is menteni kell!

6.6.2. Szuperskalaritás

Processzoron belül nemcsak azért lehet párhuzamosítani, mert az instrukciók feldolgozási fokozatait különböző egységek végzik, hanem mert egyes egységeket többszörözni is lehet! Lehet több dekódoló, több ALU (akár lebegőpontos is, akár külön ALU az eltolásokhoz stb.), külön címgeneráló az instrukcióknak, az adatoknak stb. A belső buszok a különböző egységekhez párhuzamos csatornákat biztosítanak ekkor. A többszözözött egységek párhuzamosan, egyszerre több instrukcióval dolgozhatnak.

Lesznek persze ekkor is gondok. A legjellegzetesebb, hogy az instrukció sorrendiség felborulhat. Ezt figyelni kell és helyre kell hozni a helyes sorrendiséget, hogy az eredmények helyesek legyenek. Ugyanekkor nyilvánvaló a szupercsatornázás is, ez bonyolítja a dolgot.

Természetes, hogy a RISC processzorok között vannak szuperskalár, szupercsatornás processzorok. De vannak ilyen CISC processzorok is! Pl. a Pentium kétutas szuperskalár, szupercsatornás CPU. A Pentium Pro háromutas szuperskalár processzor.

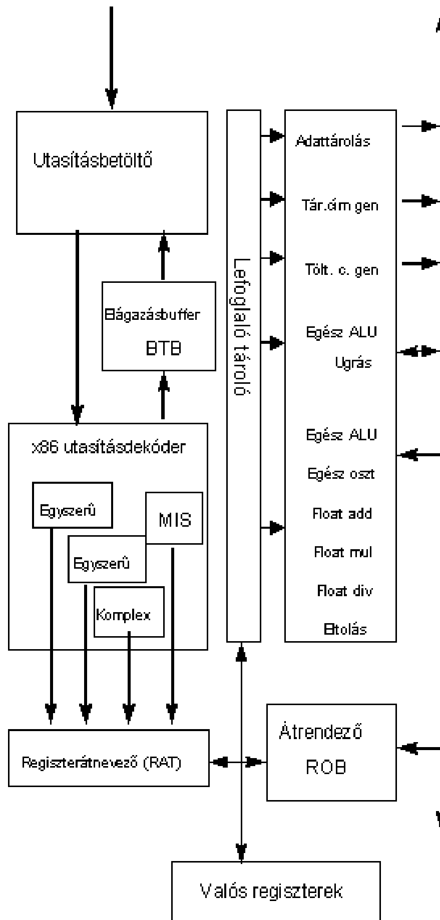
A Pentium Pro

Ennél a processzornál egy tokban két lapka van: a CPU és az L2 gyorsítótár (cache). Már 180 MHz-en is működhet. Egy instrukció feldolgozásánál 14 fokozat (3 szakaszban 8+3+3) van. Képes soron kívüli végrehajtásra, utasítás átrendezésre, elágazás előjelzésre, spekulatív végrehajtásra. Funkcionális ábráját alább láthatjuk (6.3. ábra).

Az egyes fokozatok tevékenysége a következőkben foglalható össze:

Az 1. szakasz 1-6 fokozatában az *elágazás buffer* (BTB) irányításával x86-os utasításokat hoz be az L1 gyorsítótárból. Ezeket átadják a 3 (párhuzamos) *dekódernek.*

A dekóderek mikroművelettté alakítják az x86-os utasításokat: a 2 egyszerű dekóder egy x86-ost egy mikroműveletre, a komplex dekóder egy x86-ost négy mikroműveletre. Lehetnek olyan bonyolult x86-os utasítások, amiket a komplex dekóder sem tud átalakítani, ezeket a *mikroutasítás sorvezérlő* (MIS) több mikroműveletre alakítja. Elvileg így 6, gyakorlatilag átlagosan 3 mikroművelet lesz az eredmény. A mikroműveleteket a regiszterátnevezőbe (RAT) küldik, a hamis függőségek kiszűrésére. Legtöbbször a függőség oka az, hogy az x86-os utasítások ugyanarra a regiszterre hivatkoznak (hiszen kevés az x86-os regiszter). Kiszűrhetők ezek a hamis függőségek úgy, hogy a 40 fizikai regiszterre leképzik az X86-os regisztereket (Persze, ezt nyomon kell követni!). Ezek után csak az igazi függőségek miatt "dugulnak" el a csatornák!



6.3.ábra. A Pentium Pro processzor

A 2. szakasz 3 fokozatában a következők történnek: a mikroműveletek átmennek az *átrendező tárolóba* (ROB), itt sorbaállnak a *lefoglaló tárolóhoz.* A lefoglaló tároló mintegy kiegyenlítő tartályként viselkedik: dekódolt utasításokat tartalmaz (max. 20-at). Ezeket küldi a 11 *végrehajtó egységnek.* A végrehajtó egységek a mikroműveleteket párhuzamosan hajtják végre (max. 8-at, átlag 3-at párhuzamosan). Ha soronkívüliségek fordulnak elő, akkor az eredményeket tárolják.

Végül a 3. szakasz 3 fokozatában a sorrendiség visszaállítása történik, hogy az eredmények a regiszterekben, a memóriába helyes sorba íródjanak. Ha hibás előrejelzések voltak, azok visszavonódnak, és ha az eredmények beíródtak a regiszterbe, memóriába, a megfelelő mikroműveletek is visszavonásra kerülnek.

Különleges és érdekes az a tény, hogy a Pentium Pro, ami alapvetően CISC processzor, a mikroműveletekre való átalakítással belül RISC jelleggel működik. Ez teszi lehetővé az "egyszerűsítést", az pedig a frekvencia növelést, a nagy teljesítményeket.

6.6.3. Párhuzamosítás a processzoron kívül (Multiprocesszoros rendszerek)

Két alapvető típusuk van:

- Fix feladat-szétosztású és a
- Változó feladat-szétosztású multiprocesszoros rendszerek.

Fix feladat-szétosztású többprocesszoros rendszerek

Jellegzetesen ilyenek a különböző *társprocesszoros* (coprocessor) rendszerek. Fix feladatot oldanak meg

- az aritmetikai társprocesszorok (lebegőpontos aritmetikát),
- grafikus társprocesszorok (megjelenítést),
- képfeldolgozó társprocesszorok (megjelenítést),
- hangfeldolgozó társprocesszorok (hangképzést) stb.

Változó feladat-szétosztású, általános célú többprocesszoros rendszerek, multiprocesszálás

Több processzor párhozamosan dolgozik, nemcsak meghatározott feladatokat végeznek.

Alapfogalmak:

SIMD párhuzamosság (Single Instruction Stream, Multiple Data Stream): ebben az összes processzor ugyanazt az instrukciót hajtja végre különböző adatokon. Ilyenek a *vektorprocesszorok*. (Pl. CONVEX gépcsalád processzorai, a DEC Alpha processzora stb.) A vektorprocesszorokon a mátrix műveletek gyorsan végezhetők: párhuzamosan a tömbökön. Az a jó persze, ha ez *transzparens*: a compiler döntse el, mikor érdemes a vektorprocesszálás, mikor nem!

MIMD párhuzamosság (Multiple Instruction Stream, Multiple Data Steram): az egyes processzorok különböző instrukciófolyamokat hajtanak végre a különböző adatokon. Ilyen többprocesszoros rendszer a Silicon Graphics Power Series gépe; a *zeus* (4 db R3000-es processzorral), a CONVEX gépek újabb tagjai (2, 4, 8 vektorprocesszor még MIMD párhuzamosításban is!), egyes DEC Alpha gépek stb.

Alapstruktúrák a MIMD-n belül:

Közös erőforrású struktúrák (minden erőforrás: memória, perifériák stb.) közös, ezen osztoznak a processzorok;

Részben vagy telesen *saját erőforrású* MIMD (pl. saját memóriája van minden processzornak, esetleg saját perifériái, de lehetnek részben közös perifériák is.

Az elsőben az erőforrás kiosztás és védelem vet fel megoldandó problémákat, a másodikban a kommunikáció intenzív, a szinkronizálás jelentős.

A MIMD-n belül beszélhetünk a párhozamosság fokáról is. Ez lehet:

- durva szemcsés: az egyes processzorok nagyobb feladatokat oldanak meg, amelyeknél a kommunikáció mértéke nem túlzott;
- finom szemcsés: az egyes processzorok kisebb, egyszerű feladatokat oldanak meg, ezeknél viszont a kommunikáció biztos, hogy intenzív. (Transzputerek!)

7. A sínek

Angolul bus (buses), magyarul sín a neve a számítógépek egyes komponensei közötti adatforgalmat lebonyolító, a rendszer vezérléséhez szükséges áramköröknek. A sín jelút, melyhez a rendszer komponensei kapcsolódhatnak. A kapcsolat kialakítása során meg kell határozni, hogy mely részegységek kapcsolódjanak össze (erre szolgálhat a címzés), el kell dönteni az adatmozgatás irányát a kapcsolatban résztvevő komponensek között (melyik lesz az adó és melyik a vevő), és természetesen, össze kell hangolni a komponensek működését (ez a folyamat a szinkronizáció).

A továbbított információtól függően megkülönböztethetünk

- *adatsínt* (data bus), ami adatokat továbbít az egyes komponensek között. Jellemzője a szélessége, azaz, hogy párhuzamosan hány bitet tud továbbítani a komponensek között. A nagyobb szélesség nyilvánvalóan nagyobb átviteli teljesítményt eredményez.
- *címsínt* (address bus), ami címeket továbbít, ezzel a kommunikációban résztvevő komponens (annak pontos eleme) kiválasztásában segít). Ennek is jellemzője a szélessége, azaz, hogy hány biten hordozza a címeket. A nagyobb sínszélesség nagyobb címtartomány használatára ad lehetőséget. Megkülönböztethetünk még
- *vezérlő sínt* (control bus), ami vezérlő, engedélyező és állapotjeleket továbbít. Ilyen jelek az adatátvitelt vezérlő jelek (az adatátvitel helyét, az adatátvitel irányát, az átvitt adatmennyiség méretét meghatározó jelek), a cím és az adatok stabil állapotát jelző jelek, a „kész” (ready) jelzés, amivel az adatátvitel végét vagy egy eszköz rendelkezésre állását jelzik), a megszakítást vezérlő jelek (megszakítás kéréssel és elfogadással kapcsolatosak), a sínvezérlő jelek (sín igényléssel, visszaigazolásával kapcsolatos jelek), a szinkronizációs jelek és egyéb (pl. a tápvezetékhez tartozó) jelek.

Sőt, ha úgy tetszik megkülönböztethetünk még tápfeszültség továbbítására szolgáló vezetékeket is, ezek is egyfajta sínek (bár nekünk most nem érdekesek).

A sínek „hatáskörének” kiterjedése szerint lehetnek

- *helyi sínek* (local bus), melyek rendszerint egyedi kialakításúak (nincs rájuk szabvány), általában egy nyomtatott kártyán vagy egy lapkán belül helyezkednek el, belső jeleket továbbítanak, melyeknek a kártyán (lapkán) kívül nem használhatók.
- A *rendszer sínek* (system bus), amelyek fontos rendszer komponensek összeköttetésére szolgálnak. Rendszerint szabványosak. A szabvány magába foglalja az elektromos és a mechanikai specifikációkat is (az adat- és címvonalak számát, vezérlővonalak típusait és funkcióit, jelek feszültségszintjeit, csatlakozási lehetőségeket, beleértve a csatlakozók típusát, bekötését, a kártyák méretét, meghajtó-képességet, terhelhetőség adatait stb.), valamint az információcsere szabályait is (protokollok, időzítési viszonyok, sínhozzáférési algoritmusok stb.).
- Lehetnek végül *rendszerközti sínek* (intersystem bus), mely számítógéprendszereket köt össze. Az összekötött rendszerek számítógép hálózatokat alkot, az áthidalt távolságtól és a kialakítástól függően LAN, MAN stb. konfigurációban.

Bár az eddigi rajzainkon a síneket egy-egy vonallal jeleztük, jegyezzük meg, hogy a sínek áramkörök, aktív és passzív elektronikai elemekből épülnek, nem biztos, hogy egyszerű vezetékek. A sínek áramköreiben is időigényes a jelek lefutása, ezek is ún. ciklusokban dolgoz-

hatnak, és a tranziensek lefutása után, a ciklusok végén jelennek meg a szükséges jelszintek (vagy jelszint változások), amik az információkat hordozzák.

A síneken a kommunikációs módszerek lehetnek:

- *szinkron* jellegűek. Megadott sebességgel történik az adás és a vétel, meghatározott vezérlőjelek időzítésével. Az adó ilyenkor nem vár választ, a rendszer helyes működésével a kommunikáció garantáltan hibátlan.
- *Aszinkron* átvitel során az adó és a vevő nem jár szinkronban. A kommunikációhoz kapcsolatfelvétel és gyakran a vétel visszaigazolása szükséges (handshake).

A síneken az átviteli kapacitás függ:

- az *átvitel sebességétől*, azaz, a sín működés ciklusidejétől, a sín órajelétől.
- Adat és címsíneknél függ a *sín bitszélességétől* (bitszámától), azaz párhuzamosan hány biten megy az információtovábbítás.
- Függ az átviteli kapacitás az *átviteli protokolltól* is, és végül
- a sínen elhelyezkedő *vezérlők számától* is. Több vezérlő esetén ui. versenyhelyzet állhat elő, mikor is el kell dönteni, melyik jogosult a busz használatára (arbitráció), és az az algoritmus, ami eldönti a versenyhelyzetet időigényes.

Különböző vezérlési módszerek vannak, melyeket korszerű síneken akár menet közben is válthatnak. A teljesség igénye nélkül:

- Dinamikus adatsín szélesség változtatás lehetséges (pl. 32 bites adat olvasása 8 bites portról 4 részletben, automatikusan stb.).
- Protokoll váltás lehetséges, vagy „párhuzamosan” több protokoll működhet stb.
- Blokkos átvitellel (burts mode) adatblokk mozgatható egyetlen sínműveletben. Gyorsít!

Az átvitelben két entitás (egység) vesz részt: a forrás és a cél entitás. Lehet pl. forrás a memória, cél a CPU (regisztere), vagy fordítva, de lehet forrás egy periféria vezérlő puffere, ugyanakkor cél a memória, vagy ezek fordítva. Elvileg bármelyik entitás lehet a kezdeményező, és ekkor az ő szempontjából (pontosabban az átvitel irányától függően) beszélhetünk olvasásról vagy írásról.

Még egy dolgot szükséges megértenünk, a címzést, hiszen egy sínre több entitás csatlakozik, tudni kell, honnan, vagy hova kell az adatokat küldeni. Írásról beszélünk, ha a kezdeményező a forrás (pl. CPU regiszteréből memóriába ír). Ekkor a buszon előbb a cél címét, majd az adatot küldi. Olvasásról beszélünk, ha a kezdeményező a cél (pl. CPU memóriából regiszterébe olvas). Ekkor a kezdeményező a buszon címet küld, majd másik irányú forgalomban jön az adat.

Most már magyarázhatjuk a blokkos átvitelt, annak teljesítménynövelő lehetőségét. Hagyományos átvitel során íráskor minden adatot cím előz (valóban előznie kell, hogy a cél felkészülhessen az adat fogadására). Burst módban egyetlen cím után (ez a cím a célnál a blokk kezdő címe) több adatot, blokkot küldhetnek, azaz megspórolhatják a címek átvitelét: az adatokat a kezdő cím utáni címekre helyezik el. Hasonló a helyzet az olvasás esetén is: hagyományos átvitelnél a kezdeményező előbb egyenként címet küld, majd várja az adatot, megint címet küld és vár s.í.t. Blokkos átvitelű olvasásnál is a címek átvitelével takarékoskodhatunk:

egy kezdő cím után jöhetnek az adatok (a forrásnál egymás utáni címekről, a célnál egymás utáni címekre helyezve a blokk adatait).

Nézzünk ezután sínrendszereket! Számos jogilag védett, széles körben használt sínrendszer létezik, léteznek ezek között nyílt szabványok is.

7.1. Az IBM PC-k sínrendszerei

1981-ben vezették be PC XT rendszer sínt. Ez volt az IBM XT (Extended Teechnology) személyi számítógép rendszer-síne. Ez a sín 8 bites adat és 20 bites címszélességgel rendelkezik. Utóbbiból következik, hogy maximun 1 Mbájt memória kezelésére volt alkalmas. Az XT sínt csak a processzor vagy az alaplapon lévő DMA vezérlő (lásd később!) vezérelheti. Nyílt szabvány, viszonylag egyszerű szinkron sín, azonban az IBM a jelek időzítési adatairól keveset árult el, így nehéz volt teljesen kompatibilis eszközöket gyártani a sínhez.

Az AT (Advanced Technology, 1984) rendszersín a később ISA (Industry Standard Architecture) néven elnevezett és szabványosított sín. Ennek 16 bites az adatsíne, de a 8 bites kártyák is csatlakoztathatók a sínre. Címsíne 24 bites, ami 16 Mbájt címtartomány kezelhet. Az AT sínt külső egység is vezérelheti. A sín vezérlési jogát lehet igényelni. Ez is nyílt szabvány, de mivel az IBM nem adott ki időzítéseket is tartalmazó specifikációkat, később, az EISA kialakításánál rögzítették szabványát, és ekkor kapta ISA nevét is. Eredetileg 6 MHz-es, később 8,33 MHz-es órajelet továbbított az adaptereknek, de a sínen folyó adatátvitelnek nem kell ehhez az órajelhez igazodnia, vagyis az ISA aszinkron sín. Átviteli sebességének elvi maximuma 8MB/sec, a gyakorlatban 4-6 MB/sec érhető el rajta.

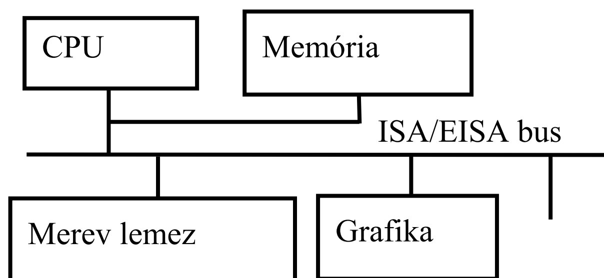
Kilenc nagyobb IBM PC kompatibilis számítógépet gyártó cég egy, a régebbi változatokkal kompatibilis sínrendszer kidolgozásával a piaci részesedését megtartani, illetve növelni kívánta, ezért kidolgozták az EISA (Extended Industry Standard Application) sín specifikációját. Az EISA 32 (16, 8) bites adatszélességű, rajta 33 Mbájt/sec sebesség érhető el. A régebbi fejlesztésű 8, 16 bites kártyákkal is használható. A címsín 32 bites, ezzel 4 Gbájt a címtartomány. Specifikációja, csatlakozó-kiosztása, időzítési adatai mindenki számára hozzáférhetek. Az EISA már szinkron sín, az átvitelét az órajele ütemezi. Az ISA kompatibilitás miatt megtartották a 8,33 MHz-et (így jön ki a 33 MBájt/sec: 4 bájt * 8,33 MHz). Ára és nehézkes konfigurálása miatt mérsékelten terjedt.

Az IBM PS/2 személy számítógépeihez kidolgozott sínrendszer az MC (Micro Channel) sín.

Az MC 32 bites adat és címszélességű. Nem kompatibilis az ISA-val illetve EISA-val. Nem nyílt szabvány, azaz alkalmazásához jogdíjat kell fizetni. Műszaki szempontból egész jó megoldás, de a jogdíjak miatt nem terjedt el, végül az IBM is lemondott róla.

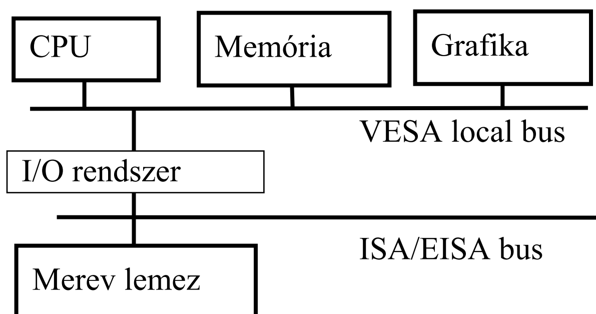
7.2. A PCI sín

A "hagyományos" PC sínek, beleértve az ún. "local bus"-t is, nem tudnak lépést követni a processzor-sebességekkel, ráadásul nem processzorfüggetlenek. Ma már csak az az előnyük, hogy sok-sok olcsó csatolókártyát használhatunk ezeken a síneken. Grafikus felhasználói felületet (Windows), CAD programokat, képfeldolgozást, hálózatot használó felhasználóknak szükségük volt egy gyorsabb sínrendszerre. Eleinte segített az EISA, az MC, később a "local bus" (bár ezekhez a bővítőkártyák drágábbak). A "hagyományos" PC-k sínrendszerét a 7.1. ábrán, a VESA helyi sínes architektúrát a 7.2. ábrán láthatjuk.



7.1. ábra. PC sínrendszer

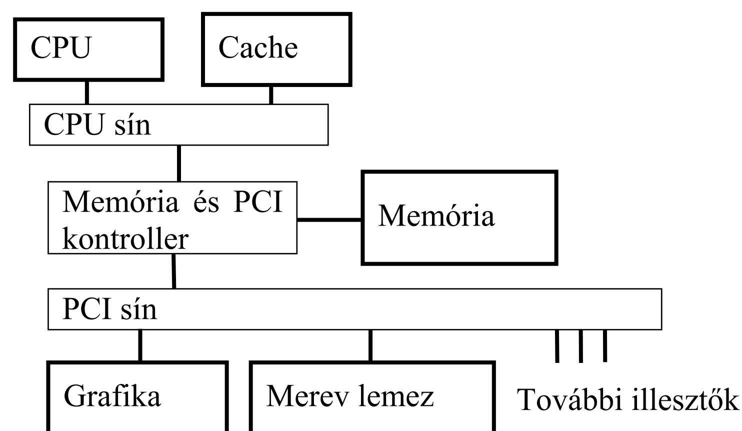
A „local busz” az alaplapra épített, gyors sín, elsősorban a grafika, másodsorban gyors merevlemezek kiszolgálására. Erősen processzorfüggő, közeli kapcsolatban van a mikroprocesszorral. A leginkább elterjedt „helyi sín” a VESA local busz. Igen-igen processzor-függő.



7.2. ábra. „Local bus”-os PC

Az Intel többszáz számítógépgyártó céggel összefogva új, iránymutató koncepciót dolgozott ki. Ez a PCI (Peripheral Component Interconnect) sín, amit 1992-ben mutattak be (7.3. ábra).

A PCI sín illesztő-helyeire merev lemezvezérlő, hálózati kártya, grafikus kártya, multi-média bővítő stb. csatlakozhat. Az összes jel- és tűkiosztás szabványos. A mikroprocesszor ugyanolyan sebességgel tudja kezelni az eszközöket, mint a memóriát. Átfogó megoldást kínál, szemben a VESA local bus-szal, ami elsősorban a grafikát, másodsorban gyors merevlemezeket támogat. Transzparens marad akkor is, ha 64 bitesre bővítik a cím- és adatsínt: a sínszélesség ilyenkor megduplázódik, de megmarad a kompatibilitás a 32 bites perifériákkal.



7.3. ábra. PCI sínrendszer

Mivel a PCI sínhez csatlakozó bővítőkártyák a mikroprocesszor nélkül is tudnak egymással kommunikálni, a CPU tehermentesíthető.

Egyszerűbben kezelhető, mint a hagyományos PC sínek, egy-egy bővítőkártya beépítése során nem kell "jumperelni", "setup"-olni: minden bővítőkártyához rendeltek egy ún. konfigurációs regisztert, amit rendszer-indításkor (boot) a mikroprocesszor kiolvas, és felismeri, milyen kártya van az illesztőhelyen, azonnal le is futtat installáló, inicializáló programot.

A PCI sín nem processzorfüggő, nemcsak Intel processzorokkal használható, hanem akár RISC processzorokkal is. Ezért nemcsak az "asztali" gépeknél, hanem munkaállomásoknál, szervereknél is jól alkalmazható. Mivel olyan áramköröket is képes kezelni, melyek a szokásos 5 volt helyett csak 3,3 voltot vesznek fel, a hordozható gépeknél is ígéretes megoldást kínál.

A PCI szabványban meghatároztak egy osztott illesztő-helyet (shared slot), amit ISA, EISA vagy MC sínnel együtt lehet használni. Erre az illesztő-helyre hagyományos bővítőkártyákat vagy olyan PCI kártyákat lehet tenni, melyek az említett sínrendszerek egyikét is használják (a gyártóknak tehát egyetlen kártyát kell fejleszteniük!).

A PCI sínes rendszerek egyik első képviselője a müncheni Tuncer cég PCI-1 gépe: 66 MHz-es 486DX2processzorral, 8MB memóriával, 520MB merevlemezzel (utóbbinak 32 bites SCSI-2 kontrollere van, beépítve az alaplapba). Három PCI illesztő-helye van, egyikben a grafika. (Kb. 5900 német márka volt az ára 1994-ben).

Az 1992-ben kiadott első PCI szabványt 1993-ban követte a PCI 2.0, és 1995-ben jelent meg a PCI 2.1. A PCI szinkron sín. Első definíciója 32 bit szélességű volt, 33 MHz-es órajellel (4*33=132 MB/sec). Második változata már 64 bit szélességet is megenged, igazodva az akkor megjelenő Pentiumhoz (264 MB/sec). A 2.1-s változat már 66 MHz-re növelte az órajelet (524 MB/sec). Ezek az sebességadatok természetesen csak elméleti maximumok, a gyakorlatban ezek 50-80 százaléka érhető csak el. Ennek oka: a PCI sínnek nincsenek külön cím és adatvezetékei, ugyanazt a vonalakat használja cím és adatsínként egyaránt. Hagyományos (nem blokkos) adatátvitelnél a "veszteség" elég nagy, ezért hacsak lehet, a PCI "burst módban" dolgozik, de még így is kellenek ciklusok pl., várakozásokra, maguknak a címeknek átküldésére stb. Mindenesetre manapság a személyi számítógépek sínrendszere a PCI sínrendszer.

7.3. További híres sínek

A maga idejében nagy jelentőségű sín volt a DEC cég PDP 11 gépcsaládnál alkalmazott UNIBUS. A CPU mellett a periféria kontrollerek is vezérelték. A DEC a mikroVAX rendszereinél a Q bus rendszert alkalmazta, később a Turbo Channel-t.

Európai nyílt szabvány a VME sínrendszer. Sok-sok ipari vezérlőt építettek erre, de számítógépeknél is alkalmazható.

Széles körben használt szabvány az IEEE 488 vagy GPIB (General -Purpose Interface Bus), amit az ANSI/IEEE 488-1978 szabvány definiál. A szabvány létrejöttét a Hewlett-Packard cég kezdeményezte, a HP-IB nevű - szabadalmi jogok a HP-nél - sínjére is alapozva.

Az SGI munkaállomások rendszersíne a GI032 nevű sín. Nem nyílt szabvány, és ha rápillantunk egy SGI munkaállomás funkcionális blokkdiagramjára, láthatjuk, hogy vannak ott még egyéb sínek is.

Az Intel saját síne a Multibus.

A SUN munkaállomások rendszersíne az SBus (32 bites adat, 25 bites cím-szélesség), multiprocesszoros implementációiban az MBus-t használják. Nyilvánosak. Gyakori, hogy van VME csatlakozási lehetőség is.

8. A memória

A memória vagy tár programok (kód) és adatok tárolására szolgáló címezhető cellák készlete. A processzor instrukcióival közvetlenül megcímezhető tárat sokszor *központi memóriának* (main memory, central memory) nevezzük. Ennek szoros kapcsolata van a CPU-val, ez a kapcsolat a rendszerbuszon, vagy akár valamilyen nagyobb átviteli kapacitású *memória buszon* realizált. A CPU instrukcióinak címrésze vonatkozhat a központi tár rekeszeire, bájtjaira.

Látni fogjuk, hogy az egyes periféria vezérlők is tartalmazhatnak *memóriát*, adat és vezérlő *regisztereke*t (amiket jól megkülönböztetünk a CPU regiszterektől), átmeneti tárolásra *buffereket*. A perifériavezérlők regisztereinek, esetleges saját memória-mezőinek címkészlete része lehet a CPU központi memóriát is átfedő címtartományának, de különbözhet is attól. Most szenteljük figyelmünket a központi memóriára.

A célja világos: adatok és instrukciók tárolása.

Manapság a kialakításuk: *magasintegráltságú félvezető lapkák*. Régebben szokásos volt a ferrit-gyűrűs [3]központi tár, ami az információkat a mágneses tulajdonságok megváltoztatásával rögzítette, az információk visszanyerését a mágnesezettség irányának „kiolvasásával” biztosította. A mai viszonyok között szokatlanul nagy méreteket igényelt, adat visszanyerési sebessége jó volt, az információtartalmát a tápfeszültség megszűnése esetén sem vesztette el.

Két követelmény fogalmazható meg, két trend alakult ki a félvezető memóriákkal kapcsolatosan:

- növekszik az igény a kapacitás növelésére[4](egyben a címtartomány növelésére), méghozzá az egységnyi költségre eső kapacitás növelésére;
- még jelentősebben növekszik az igény az kiszolgálási idő[5] csökkentésére. A mai processzorok általában sokkal gyorsabbak, mint a kommerciálisan beszerezhető memóriák.

A két követelményből az elsőt sikeresen teljesítik, a második azonban az elsőhöz képest elmarad.

---

[3] A ferrit gyűrűs tárak 4096 bitet 6400mm^2-en tároltak (ez ~0.64 bits/mm^2, ~0.52 bits/mm^3).

[4] 1 Mbits DRAM 1048576 bitet tárol, kb. 50 mm2-en. Ebbõl: ~21000 bits/mm2, ~42000 bits/mm3 jön ki. Ez 32000-szer nagyobb a mm^2-re, 81000-szer nagyobb a mm^3-re. Megjegyezzük, a DNA molekula 45*10^{12}-szer nagyobb információ tároló, mint a meglapka.

[5] A kiszolgálási időnek két összetevője van:
Egyik az elérési idő (access time), ami a memória modulhoz beérkező kérelemtől az adat rendelkezésre állásáig eltelt idő (másként a memória modul válaszideje). Ez nagyságrendileg 80-50-ns szokott lenni;
A másik a memóriasinbeli (chipset-beli) idő. Ez átlagosan 125 ns.
Mindezekkel az átlagos elérési idő 195 ns. Összevethetjük ezt egy átlagos L2 gyorsítótár (ez a CPU-n kívüli, ún. external cache) kiszolgálási idővel, nos ez 45 ns. Ezekkel szemben egy 1000MHz-es CPU (ma már elavult) ciklusideje 1 ns.

8.1. A félvezető tárolók

A lapkákba tokozott félvezető tárolók megnevezéseinél különböző betűszavakat (RAM, DRAM, ROM, PROM stb.) használunk. Nos, a betűszavak közül több osztálynév, mely memória osztályba aztán további, esetleg más betűszavas memóriafajta is tartozik. Nézzük ezeket a betűszavakat!

- RAM (Random Access Memory) az első betűszó. Szó szerint „véletlen elérésű memória" volna a jelentés. Olyan memóriát nevezünk RAM memóriának, melynél egy cella (rekesz) elérése nem függ a többitől, azaz akár „véletlenszerű" sorrendben is elérhetjük a cellákat. Az elnevezés szembeállítja a RAM-ot a soros elérésű memóriával, ahol egy rekesz eléréséhez „át kell jutni" az előtte lévő rekeszeken.
  RAM lapkákat többféle technológiával is megvalósíthatnak. Közös jellemzőjük azonban az, hogy sorokból és oszlopokból álló háló csomópontjai, elemei a cellák. Sor- és oszlopcímekkel kiválaszthatók a cellák. Egy-egy cella információtárolási módja, megvalósítása azonban különböző lehet a különböző RAM-oknál.
- DRAM (Dynamic Random Access Memory) lapkák a RAM családba tartoznak. Ezeknél egy-egy cellát egy tranzisztor – kondenzátor (kapacitor) pár valósít meg, egy bit tárolására. A cella kondenzátora „tárolja" az 1 bitet, ha fel van töltve. A cella tranzisztora a feltöltéshez, ill. a kiolvasáshoz szükséges. Beláthatjuk, hogy a cella információt tároló kondenzátor töltésszintje magára hagyva „lecsenghet", azaz az információ „elveszhet". Ezért a cella tranzisztorokat időről időre „frissíteni" kell. (A frissítést vezérelheti a memória kontroller, de vannak már „önfrissítő" lapkák.) A frissítés azaz a jelszintek megemelése, illetve a beírás – kiolvasás időigényes, ciklusokban történik: mindezekért nevezik dinamikusnak az ilyen lapkákat. A feszültség kimaradása esetén ezek a memórialapkák „felejtenek".
  A DRAM lapkákban további speciális áramkörök segítik a sor- és oszlopkiválasztást, a cellákból kiolvasott jelek átmeneti tárolását, a frissítést.
- ROM (Read Only Memory) típusú félvezető tároló lapkák nem vesztik el adataikat a gép kikapcsolásakor sem. Ezekbe a tartalmat a gyártásuk során töltik be („Beégetik", szoktuk mondani). Elérésük szintén „random"" jellegű, azaz rekeszeik, bájtjaik véletlenszerűen címezhetők és kiolvashatók, viszont nem írhatók. Címezhetőségük, címtartományuk megegyezhet a DRAM-ok címezhetőségével, címtartományával.
  A ROM lapkák is sor és oszlop tömbbe rendezett cellák hálózata. A cellákban diódák vannak, az 1 bithez összeköttetést adnak, a 0 bithez nem adnak összeköttetést.
  A sor- és oszlopkiválasztás hasonló a DRAM lapkákéhoz.
- PROM-oknak (Programable Read Only Memory) nevezzük azokat a lapkákat, melyeknél a felhasználó is elvégezheti a „beégetést" (de csak egyszer). Az ismert sor-oszlop háló celláit itt „olvadó biztosítékot" (fuse) képezik. „Beégetéskor" a kiválasztott cellák biztosítékait a munkafeszültségnél nagyobb feszültséggel terhelve ténylegesen kiégetik. A beégetés természetesen erre a célra készült berendezésen végezhető.
- EPROM-oknak (Erasable PROM) nevezzük azokat a lapkákat, melyeket ultraviola fénynyel lehet törölni, majd újra lehet programozni (természetesen külön készülékben, nem normál működés közben).
  A cellákban két tranzisztor, két kapu található ezeknél. Az egyik kapu azt szabályozza, hogy a ráadott „normális szintű" töltés átjusson-e, vagy sem a kapun, 1-et vagy 0-át adjon-e a cella. „Kissé nagyobb" töltést adva viszont ez a kapu elektronágyúként viselkedik: elektronok átjutva a két kapu közötti oxidrétegen „csapdába esnek". Később ez „zárást" jelent a cellán, a „normális szintű" töltés nem tud majd átjutni.

- Az EEPROM (Electronical Erasable PROM) nagyon hasonló az EPROM-hoz, két tranzisztoros, két kapus a cella implementációjuk. Három töltési szint jelenhet meg egy-egy cellán: a törlési, az írási és az olvasási szint, azaz nem ultraviola fénnyel törölnek itt. Általában a törlés (a két tranzisztor közötti oxidrétegen átjutott, „bennrekedt” elektronok kiszivárogtatása) lassú, vagyis az ilyen lapka kiszolgálási ideje lassú. Az ún. FLASH memória valójában EEPROM, azzal a kiegészítéssel, hogy egyszerre blokknyi információt (512 bájtot) lehet újraírni.
  A ROM-ok, PROM-ok. EPROM-ok a „nem felejtő memóriák”.
- Az SRAM-ok külön említést érdemelnek. Ezek is "random elérésűek" és írható olvasható memóriák, kiolvasási idejük azonban hallatlanul gyors, szinte nulla idejű. Viszont relatíve drágák és jelentős az energiaigényük. Utóbbiak miatt nagy kapacitások kialakítása nem egyszerű SRAM-okból, nemcsak drága, hanem különleges hűtési viszonyokat is igényel már a nem is nagy kapacitású SRAM tár. Mindenesetre ilyenekből szokás kialakítani az ún. cache tárakat, a gyorsítótárakat.
  Egy-egy cellájuk 4 – 6 tranzisztorból álló flip-flop áramkör, azaz nincs bennük kondenzátor. Megvalósításuk a CPU regiszter megvalósításokhoz hasonlít.

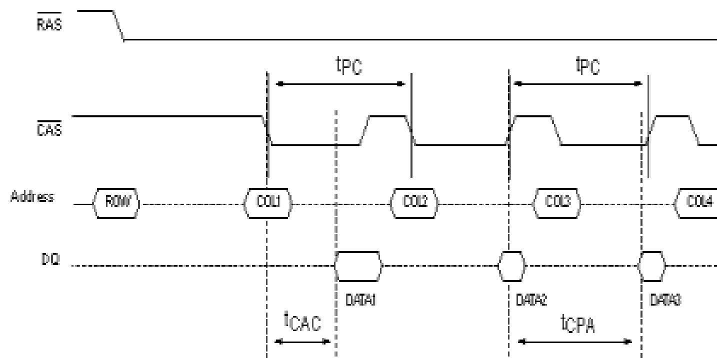
8.2. Az alapvető DRAM operációk

A DRAM memória memóriacellák táblázatának képzelhető el. A cellák valójában „kapacitások” (cells are comprised capacitors), és „képesek” biteket tárolni. A cellák sor- és oszlopdekóderekkel címezhetők, a dekóderek pedig a RAS és CAS óragenerátoroktól kapják a jeleket. A sor és oszlopcímeket sor- és oszlop cím-bufferekben fogják össze. Pl. 11 címvonal esetén 11 sor és 11 oszlop cím-buffer van. „Sense amps”-nak nevezett elérési tranzisztorok kapcsolódnak minden oszlophoz, hogy az olvasási és olvasási operációkat segítsék. Miután a cellák „kapacitások”, minden olvasással „lecsengenek”", az elérési tranzisztoroknak minden ciklusban vissza kell állítani (restore) az értékeket a cellákban. A frissítő vezérlő (refresh controller) határozza meg a frissítési időintervallumot, gondoskodik a teljes tömb - minden sor - frissítéséről. Megjegyezzük, hogy ezzel valahány ciklus a frissítésre „fordítódik”", teljesítményvesztést okozva.

Egy tipikus memória elérés a következő mozzanatokból áll: először a sorcím bitek megjelennek a cím lábakon. Amikor a RAS jel „leesik”, beíródik a sorcím a sorcím bufferbe és aktiválódnak az elérési tranzisztorok. Ezután az oszlopcím adódik a címlábakra, és a CAS leesésére az oszlop cím az oszlop cím bufferbe kerül, ugyanekkor aktiválódik az output buffer. Ezután az elérési tranzisztorok a címzett cella tartalmat az output bufferbe írják.

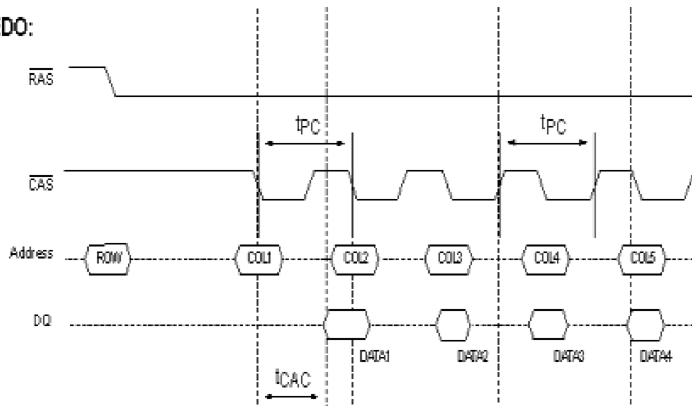
8.3. A Page Mode, Fast Page Mode, Hyper Page Mode elérések

A DRAM elérések „gyorsítására” (belső operációk elhagyásával) találták ki a Page Mode eléérést, majd ennek továbbfejlesztett változatait. Ezeknél a RAS jel végig aktív, és ezzel „egész lap” (tulajdonképpen sor) elérhető az elérési tranzisztorokkal: ciklikus CAS jelzésekkel a különböző oszlopcímek „elcsíphetők”. Egyetlen sorcímzésre több oszlopcímzés jut. (Persze, ez csak bizonyos esetekben hoz tényleges gyorsítást). A „puszta” PM nem használatos, ennek változata a Fast Page Mode viszont igen. Itt a CAS „leesésre” induló oszlopcímzés sikere után indulhat az output buffer töltés. Az output buffer aktiválódása is a CAS leeséskor történik, az output buffer „kikapcsolása”" pedig a CAS jelszint emelkedésekor (lásd a 8.1. ábrát!).



8.1.ábra. Fast Page Mode

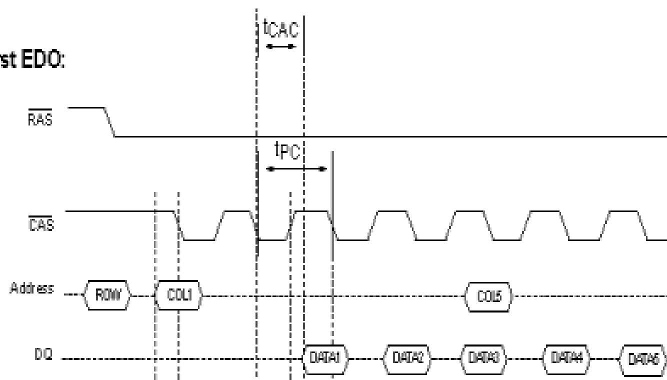
A Hyper Page Mode (más néven Extenden Data Out: EDO) tovább növelheti az elérési teljesítményt Az új ebben, hogy az output buffer „kikapcsolás” nem történik meg az emelkedő CAS élnél, emiatt a CAS emelés hamarabb megtörténhet (ebből következően a CAS leesés is hamarabb jöhet): egy cella címzés (precharge time) egyidőben lehet az output buffer (korábbi cellatartalommal) aktív állapotával (lásd 8.2. ábra).



8.2.ábra. EDO RAM

A Burst EDO (BEDO) gyorsítás alapja az, hogy egy cím bevitele után a következő 3 cellacím „belsőleg” generálódik (megszűnik 3 oszlopcímzés ezzel). (Üzletpolitikai okok miatt hamar elhalt).

Mindezek (Fast Page Mode RAM, EDO, BEDO RAM) a teljesítménynövelést a blokkosítással (bursting, több adat egy „löketben”) érték el. Ma azonban már elavultak. Az újabb technológiák és ezekkel az újabb betűszavak az SDRAM, ESDRAM és a RDRAM.



8.3.ábra. BEDO RAM

8.4. Az SDRAM-ok (Synhronous DRAM Operation)

Itt a lapka megkapva az információkat a processzortól (címek, vezérlő jelek olvasáskor, továbbá adatok íráskor) ezeket rögzíti és ezután a rendszeróra kontrollja alatt működik: a processzor ezalatt más feladatokat kezelhet. Bizonyos ciklus után az adatok beíródnak a memóriacellákba (íráskor) illetve rendelkezésre állnak és a processzor olvashatja azokat a kimeneti vonalakon. Előnye a megoldásnak, hogy a rendszeróra adja az időzítéseket, ezzel a processzornak nem kell törődni. Nagyobb buszsebességeknél (100 MHz és e fölött) az előző megoldások már nem kielégítőek (A BEDO még működne, de nem alkalmazzák).

Az ESDRAM (Enhhanced SDRAM) lapkákban a szokásos SDRAM áramkörökön túl kisebb SRAM típusú gyorsítótár (cache) is van. A lokalitás elvet kihasználva mehet a sínfrekvencia akár 200 MHz fölé is.

Az RDRAM (Rambus DRAM) a Rambus cég speciális, nagysebességű adatsínes technológiája. Igen erős „párhuzamosítással" nagy frekvenciát (800 MHz) érnek el. Viszont a Rambus csatorna csak 16 bites. A RDRAM lapkákhoz speciális RIMM (Rambus Inline Memory Modul) modult kell használni.

A DDR SDRAM a Double Data Rate SDRAM rövidítése. Az idea a technológia mögött a következő: az output operációk a lapkán mind az eső, mind az emelkedő órajel élekkel aktiválódnak. 100 MHz-es sínnél az effektív sebesség így 200 MHz.

Már létezik Dual Channel Rambus (DCR) is. Ennél az RDRAM modulokat párban alkalmazzák, elérve ezzel a 3,2 GBps átviteli sebességet.

A Protocol Based SDRAM technológiában ugyanazokat a sínvonalakat használják mind a vezérléshez, mind az adatokhoz. Az eddigi szeparáltság ugyanis technológiai okokból már sebesség korlátokat jelentettek. Kétféle figyelemre méltó osztály van ezen belül: a Synlink DRAM (SLDRAM) és a Direct Rambus DRAM (DRDRAM). Az SLDRAM nyílt szabvány, ahol is az output ráta 400 MHz, sőt, a Double Rata lehetőség miatt akár 800 MHz is. A DRDRAM – mint ahogy minden Rambus technológia licenszes.

8.5. A memória modulokról

A DRAM lapkák tokozásairól az előadáson szólunk. Mindenesetre megjegyezhetjük, hogy manapság a TSOP (Thin Small Outline Package) és CSP (Chip Scale Package) a népszerű. Mindkettő felületszerelt technológiájú, lapos. A TSOP kredit kártyákban, notebook-okban is használatos, a CSP az új Rambus lapkák tokozása.

A tokozott memória lapkákat szabványos memória modulok hordozzák. A SIMM (Single In-Line Memory Modul) a „közönséges", a DIMM (Dual In-Line MM) a 64 bites processzorokhoz való, a SODIMM (Small Outline DIMM) a hordozható gépekhez, notebook-okhoz való memória modul. A Rambus lapkákat a RIMM és a SORIMM modulok hordozzák.

8.6. Hogyan csökkenthető a hozzáférési idő?

1. A buszműveletek minimalizálásával:

- párhuzamosítással (több vezetékkel),
- soros módszerekkel (burst üzemmóddal, blokkátvitellel).

2. Autonóm buszvezérlők és az utasítás pufferek alkalmazásával, ún. prefetch queue-val.

Az alapgondolat az, hogy a tár igénybevételt időben "elosztják": a viszonylag lassú tár tartalmát egyenletes, a tár számára maximális sebességgel, autonóm módon, a CPU támogatása nélkül beolvassuk egy gyors FIFO tárba. Ez a FIFO a prefetch queue (előbehozó sor). A CPU az instrukciókat a sokkal gyorsabb hozzáférésű, viszonylag azonban kis kapacitású prefetch queue-ból olvassa. Különösen jó ez a CISC architektúráknál, ahol láttuk, hogy a klasszikus

CPU-n belüli párhuzamosítás a feldolgozási fázisok ciklus-idő különbségei miatt nehézségekbe ütközne.

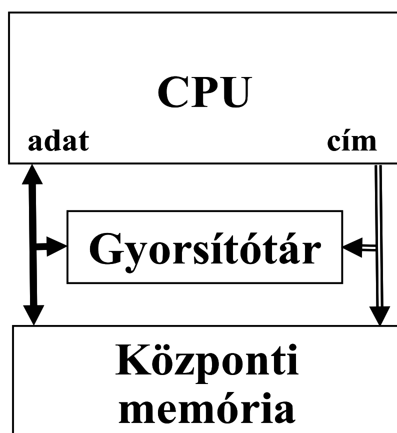
3. Az effektív hozzáférési idő csökkentése.

Az alapgondolat az, hogy bár lassú memóriát alkalmazunk, de olyan megoldást keresünk, hogy egy fizikailag létező memória lapkához viszonylag ritkán kelljen fordulni.

- Közéillesztés (interleaving) módszere: az egymás utáni címeken felváltva más-más tármodul „szólal" meg. Alkalmasan kialakított címdekóder kell, de megvalósítható.
- Gyorsítótárak (cache memory) alkalmazása.
- Asszociatív tárak alkalmazásával.

8.7. A gyorsító tárak (cache)

A programok „lokalitása"[6] megengedi, hogy az éppen végrehajtás alatt lévő program és adatrész környezetét átmenetileg egy kisebb kapacitású, de nagyon gyors elérésű memóriában is tároljuk. Ez a CPU-hoz „közeli" memória a gyorsítótár (cache). A cache készülhet különleges technológiával és fajlagos ára lehet magas, mindez a kis méret miatt nem jelent túl magas költséget. Tulajdonképpen a központi memória és a CPU közé illesztjük a gyorsító memóriát (8.4. ábra).



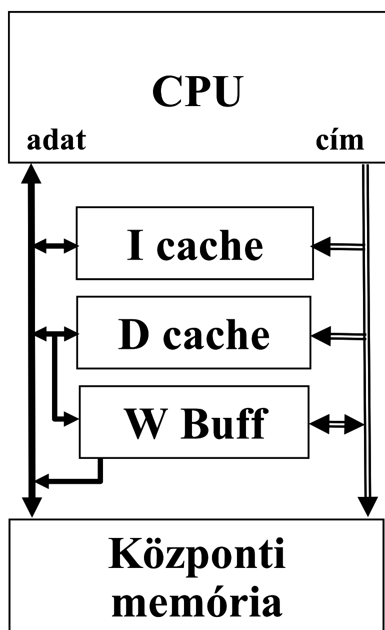
8.4.ábra. Egyszerű gyorsítótár

Amikor a processzor olvasásra megcímez egy központi memória cellát (akár instrukciót, akár adatot), a megcímződik a gyorsítótár is, és ha a cella tartalma a gyorsítótárban is megvan, a cellatartalom mozgatás a gyorsítótárból történik – sokkal nagyobb sebességgel. Ha a cellatartalom nincs a gyorsítótárban („cache miss"), akkor a központ tárból történik a beolvasás, ugyanakkor a gyorsítótár frissítődik, a lokalitás elve miatt ui. van esély, hogy a közeljövőben erre a cellatartalomra szükség lesz. A CPU-ból való írás – ez gyakorlatban csak adat írás lehet – sajnos mindenképp lassú folyamat: bár gyorsan be lehet írni a tartalmat a gyorsítótár cellába, az adatkonzisztencia fenntartása miatt a központ tárba is be kell írni, ami bizony lassú.

---

[6] Principle of Locality
Processzek sztatisztikailag megfigyelhető tulajdonsága, hogy egy időitervallumban címtartományuk egy szűk részét használják.
Időbeli lokalitás: hivatkozott címeiket újra használják.
Térbeli lokalitás: közeli címeiket használják.
Az elv érvényesülése miatt van értelme kisebb, de gyorsabb átmeneti tárolók használatának …
(Gyorsítótár; munkakészlet; TLB; diszk buffer cache …)

Éppen azért, mert az instrukciókat a processzor sohasem írja, van értelme elkülönítve adat gyorsítótár (D cache) és instrukció gyorsítótár (I cache) kialakítására. Az I cache és a vele kapcsolatos sínek megvalósítása más, egyszerűbb, mert az I cache-t sohasem kell a CPU-ból írni. További gyorsítást érhetünk el az ún. átmeneti tárolós íróegység (write buffer) alkalmazásával (lásd 8.5. ábra). A *write buffer* autonóm egység, a processzortól függetlenül, vele párhuzamosan is tud dolgozni. Adat írás esetén a processzor csak az adat gyorsítótárba (D cache) ír. Ezzel párhuzamosan a write buffer egység is működésbe lép, és a CPU-tól függetlenül, vele párhuzamosan dolgozva, a D cahe-ből tölti a központi memória cellát. Kissé lassabban, de megteremtődik a konzisztencia.



8.5.ábra. Gyorsítótár write buffer-rel

A mai korszerű rendszerek – további teljesítménynövelés elérése miatt – többszintű gyorsítótárakat alkalmaznak. Az ún. első szintű gyorsítótár (L1 cache) rendszerint a CPU lapkában implementálják, 10 ns átlagos elérési idővel rendelkezik, 4 – 16 KB körüli a nagysága. Az L2 gyorsítótár – a CPU lapkán kívüli a megvalósítás, rendszerint SRAM implementáció – átlagos elérési ideje 20 – 30 ns, maximum 45 ns, nagysága mostanában 128 – 512 KB. Emlékeztetőül, a valamilyen SDRAM technológiájú központ memória átlagos elérési ideje manapság 60 – 195 ns tartományban van, szokásos kapacitása 128 – 512 MB a kliens gépeknél.

8.8. Az asszociatív tárak (CAM Content Addressable Memory)

Nevezik tartalom szerint címezhető táraknak, illetve ***TLB***-nek (***Translation Lookaside Buffer***-nek) is. (Lásd pl. a MIPS cég R3000-es processzorát, a CP0 System Control Coprocessor részeként).

Az alap itt is a programok lokalitása: kisméretű memóriát alakítunk ki - az MMU (Memory Management Modul) részeként -, ebből az adatnyerés viszont nagyon gyors.

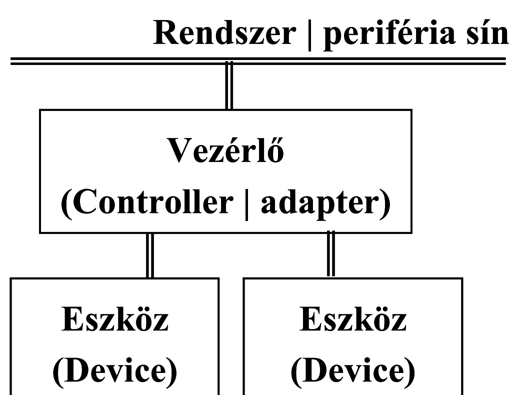
A TLB kevés számú bejegyzéssel rendelkező memória tábla (pl. az R3000-nél 48 bejegyzésű). Mikor egy címen lévő adatra szüksége van a CPU-nak, az átadva az MMU (harver!) először ellenőrzi, hogy az jelen van-e az asszociatív memóriában. Mindezt úgy teszi, hogy párhuzamosan összeveti a TLB bejegyzéseket (egyszerre minden bejegyzést) a kívánt adattal. Ha egyezést talál (és nem sért védelmi előírásokat), akkor a cím máris megvan. Ha nem talál egyezést, akkor persze szükséges a "frissítés", ami történhet blokkos átvitellel is.

Az asszociatív memóriahasználat a mai kommerciális számítógépekben elsősorban a virtuális-valós címleképzést segíti, nem a közvetlen adat/instrukcióelérést. Az Operációs rendszerek c. tantárgyban bővebben lesz szó erről a témáról.

9.Perifériák, eszközök

A külvilággal való kapcsolattartáshoz elengedhetetlenül fontos eszközök a perifériák. Szerepüket az alábbiakban soroljuk fel:

- Kapcsolattartás biztosítása a felhasználókkal (alfanumerikus/grafikus terminálok).
- Kapcsolattartás gépek között (hálózatvezérlők, soros/párhuzamos portok stb.)
- Kapcsolatbiztosítás nyomtatókhoz, rajzgépekhez stb. (soros/párhuzamos portok).
- Kapcsolat információforrásokkal (A/D jelátalakítók, érzékelők stb.)
- Másodlagos adattárak: (diszkek: struktúrált v. blokkorientált eszközök)
  - fájlrendszer biztosítása;
  - lapozási/kisöprési terület biztosítása.
- Harmadlagos tárak archiváláshoz, biztonsági mentésekhez (szalagok, kazetták, CD-k stb.)
- Különleges célú eszközök (pl: óra eszköz).



9.1. ábra. A perifériák architektúrája

Legáltalánosabb architektúrájuk a 9.1. ábrán figyelhetjük meg. Az ábrán kivehető, hogy az eszközök valamilyen vezérlő áramkörrel (vezérlő, controller, adapter) kapcsolódnak a számítógéphez, annai is valamelyik sínjéhez.

Az eszközök valamilyen módon az input-output-tal kapcsolatosak, azaz adatáramlás van az eszköz-vezérlője-sín-memória(-processzor) között, ami rendszerint kétirányú adatforgalom.

9.1. A vezérlők (controllerek, adapterek)

Az eszköz vezérlők aktív és passzív elektronikai elemekből épülő nyomtatott áramkörök. Implementálhatók az alaplapon, de lehetnek a rendszersín, vagy valamilyen I/O sín csatlakozójába helyezett kártyán is. Egy vezérlő több eszközt is vezérelhet, akár különböző fajtájúakat is (multicontroller). A vezérlő áramkörök lehetnek bonyolultak és „intelligensek” is: saját processzoruk saját mikroprogramja szerint működhet, meglehetős autonómiával, a CPU-tól függetlenül is. Ekkor persze a kontroller és a CPU működésében szinkronizálási feladatok merülnek fel. Rendszerint van saját memóriájuk, vannak saját regisztereik (adat regiszterek a parancsaik paramétereinek tárolására, vezérlő regisztereik a parancsok számára), van átmeneti tárolójuk (buffer) a mozgatott adatok számára

9.1.1. A vezérlők feladatai (általánosan)

- Kapcsolódó felületet (interface) biztosítanak a rendszer vagy periféria sínhez, ezzel a gép további részéhez.
  - Néha képesek a rendszer vagy periféria sín vezérlésére, azaz a saját pufferükből a központi memóriába, vagy a központi memóriából a saját pufferükbe adatok mozgatását menedzselni a CPU felügyelete nélkül is (Direct Memory Access, DMA vezérlés).

- Saját átmeneti tárolójuk és a központi memória közötti ellenőrzött adatforgalmat hajtanak végre, vagy az adatforgalomban résztvesznek.
    - A szinkronizálást oldanak meg megszakítás (interrupt) generálásával. Ezzel jelzést küldenek a processzornak, azt jelezve, hogy bizonyos részfeladatok bizonyos eredményeséggel befejeződtek.
- Átmenetileg tárolják az átviendő adatokat.
- Kapcsolódó felületet (interface) biztosítanak az eszközök felé.
    - Ezen belül kiadják az eszközöket vezérlő, szabályozó jeleket (pl. az eszköz szerelvények mozgatásához szükséges jeleket, diszk fordulatszám szabályozás jeleit, író-olvasó fejek pozícionáláshoz jeleket stb.).
    - Ellenőrzött adatforgalmat valósítanak meg a saját bufferük és az eszköz között.
    - Hibakezelést valósítanak meg. (Pl. szükség esetén bizonyos számú ismételt adatátvitelt kezdeményeznek az eszközel.)
- Sokszor ipari szabvány a kapcsolódó felület (interface) a vezérlő és az eszköz között. (Pl. híres szabvány az SCSI [Small Computer System Interface]), de vannak más szabványok, szokványok is.

9.1.2. Egyszerűsített forgatókönyv (scenario) diszkes adatátvitelre

Egy elképzelt diszkvezérlő esetére bemutatjuk azt a forgatókönyvet, ami egy diszk blokk behozatalakor (olvasásakor) játszódik le. Tanulmányozva a forgatókönyvet, következtethetük a disz blokk írás lépéseire is, megérthetjük egy vezérlő működését és működtetését.

A feladat tehát adott diszk blokkot behozni a memória adott című területére. A feladathoz ismernünk kell a diszk blokk (logikai) címét (mit hozzunk be) és a memóriában a cél cime(ek)et (hova tegyük), továbbá azt a tényt, hogy behozatalról van szó és nem kiírásról (adatforgalom iránya). Tegyük fel, hogy egy processz ezeket az információkat már előállította (önállóan vagy rendszerprogramok segítségével). Az előállított adatokat valamilyen módon közölni kell a vezérlővel, ami ezek után már meglehetős önálósággal tud feladatok végezni.

Képzeljünk el a mi diszkvezérlőnket. Ez egy nyomtatott áramköri kártya, melynek van legalább 2 adatregisztere, 1 vezérlő regisztere és legalább 1 blokknyi saját puffere. Tegyük fel, ez az elképzelt vezérlő képes autonóm módon működni (függetlenül a CPU-tól), és képes vezérelni a buszt (Direct Memory Access, DMA vezérlő).

Konvenciónk (és a vezérlőt gyártók szándéka) szerint első adatregiszter fogadja a diszk blokk (logikai) címét, azt, hogy melyik diszk (partíció, kötet) melyik blokkját kell beolvasni. A második adatregiszter fogadja központi memóriában a cél kezdőcímét. A diszk blokk e kezdőcímtől kezdve növekvő címek szerinti egymásutáni rekeszekbe kerül majd, a blokk hosszúságának megfelelő végcímig. Miután a blokk hossza konvencionálisan kötött, ezt nem kell „közölnünk” a vezérlővel. Végül a vezérlő regiszterbe kell majd írni az adatforgalmi irányt is megadó indítási parancsot. Vagyis, ha a vezérlő regiszterbe bekerül az „olvass be” parancs kódja, vezérlő „önállóan” kezd működni és a lentebb látható forgatókönyv bizonyos pontjai szerint tevékenykedik.

A fentiekből látható, hogy a vezérlőnket a regisztereibe való beírással lehet „programozni”, megadni neki, mit is csináljon. A kérdés ezek után, vajon mi módon tudunk a regiszterekbe írni? A vezérlő regisztereinek mi a címe? Milyen CPU instrukciókkal tudjuk azokat kezelni? Vajon a CPU által ismert, a CPU által a sínre kiadható címtartományba esnek a regiszterek címei?

Válaszunk legtöbbször az igen. Általában a vezérlők regiszterei a CPU által ismert címtartományba esnek. A vezérlők címtartományainak kezelése - architektúrától függően - kétféle lehet:

- A vezérlők címtartománya egybeesik a központi memória címtartományával: ekkor a szokásos MOVE instrukciók érik el a vezérlők rekeszeit is. (pl: VAX). A példánkban ezt választjuk.
- Az eszközvezérlők saját memóriájának elkülönült címtartománya van. Más "mozgató" instrukciók kezelik a főtárat mint a vezérlő regisztereit (pl: IN/OUT instrukciók az Intel processzoroknál).

Amennyiben a vezérlők címtartománya elemeire a CPU közvetlenül nem tud hivatkozni (olyan sínen van a vezérlő melyen a memóriái nem szólíthatók meg), a vezérlőket „üzenetekkel" programozzák. A periféria sínen át „vezérlő karaktersorozatokat", „vezérlő szekvenciákat" küldenek. Ezt a vezérlő értelmezi és saját maga állítja be regisztereiben a paramétereket. Mint említettem, a mostani példánkban nem ez az esetet választottam.

Ekkor a forgatókönyv lépései a következők:

1. lépés.: CPU instrukciók „felkészítik" a vezérlőt az adatátvitelre. A CPU végrhajtja a következő kis kódsorozatot:

```
MOVE  blokk_cim  választott_vezérlő_adatregiszter1-be
MOVE  hova        választott_vezérlő_adatregiszter2-be
MOVE  be         választott_vezérlő_controlregiszter-be
```

2. lépés. A CPU ezek után végrehajthat valamilyan más hasznos feladatot. Ezzel párhuzamosan a vezérlő „dolgozni kezd" a 3. lépés szerint.

3. lépés. A vezérlő előállítja a diszk mozgatáshoz szükséges jeleket, küldi ezeket a diszknek. „Felpörgeti", a diszket, az olvasófejet mozgattatja, leolvastatja a szektort, behozza a saját pufferébe.

4. lépés. Még mindíg párhuzamosan dolgozik diszkvezérlő a CPU-val: saját pufferben ellenőrző összeget (check-sum) képez, ellenőrzi, vajon sikeres volt-e a blokk olvasása és a pufferébe való behozatala. Hiba esetén adott számú ismétlést végez a 3. lépéssel együtt. Végleges hiba esetén megszakítást (interrupt) a CPU számára, jelezve a hibát.

5. lépés. Az előző lépés sikere esetén a vezérlést átveszi a rendszer-sín fölött, és elküldi a blokkot a saját pufferéből az adott memória címre. Az előző lépés végzetes hibája esetén ez és a következő lépés kimarad.

6. Ha elkészült az előző lépéssel, akkor megszakítást (interrupt) generál a CPU számára. Jelzi, hogy sikeresen megtörtént a diszk blokk behozatal.

7. A CPU lekezeli a megszakítást. A kezelés a megszakítás típusától függő.

9.2. Megszakítás (interrupt, IT)

A megszakítás jelzés a CPU-nak. Váratlan és asszinkron esemény bekövetkezésére utal. Az aktuális utasításfolyam végrehajtása ideiglenesen felfüggesztődik (megszakad), és az megszakítás típusától függő utasítássorozat hajtódik végre. Ez az instrukció folyam az IT kezelő (IT handler). A lekezelő instrukciósorozatot szokásosan egy „megszakításból való visszatérés” intsrukcó zárja a kezelés után folyatódik az eredeti instrukciófolyam végrehajtása.

Az IT-ket kiváltó események csoportjai

- Perifériális eszközökkel kapcsolatos események (I/O interrupt). Ezeket az eszközök vezérlői generálják.
- Folyamatokon belül előidézett szándékos esemény (pl: CPU mód váltás: user-mód - trap - kernel mód).
- Folyamaton belüli nem szándékos esemény (pl: hiba: error).
- Óraeszköz megszakítás (idő/dátum mezők módosítása stb.)
- Másik folyamat által keltett esemény (inter-process communication, szinkronizálások stb.)
- stb.

A megszakítás kezelő (interrupt handler)

Az operációs rendszer magjához tartozó kódszegmens, rutin, amire a vezérlés átadódik, ha IT következett be. A kezelők meghívása eltér a szokásos függvény- eljárás hívásoktól. Utóbbiak hívása ún. „call jellegű”: a hívás argumentumai és a visszatérési cím a veremtárba kerül, utána feltétel nélküli ugrás instrukcióval ugrik a vezérlés menete a függvény, eljárás kezdő címére. A megszakítások meghívása rendszerint egy „ugrási tábla” indexeléssel kiválasztott címére való ugrással történik. A mai hardver architektúrák rendszerint „vektoros” megszakítási rendszerűek.

A lehetséges megszakítások mindegyike kap egy egyedi sorszámot. Az egyes megszakítások kezelőinek kezdőcímeit egy cimeket tartalmazó vektor elemei tartalmazzák. Ez az ugrási vektor a megszakítás sorszámával indexelhető. A CPU-nak elég annyit jelezni, hogy hányas sorszámú megszakítás következett be, a kezelőjénel kezdőcímét a vektor indexelésével könnye és gyorsan ki tudja válastani.

IT bekövetkezésekor a CPU az állapotát (a processz kontextus dinamikus részét, a regisztereket) le kell menteni egy veremtárba, hogy a kiszolgálás után „felvéve” a regiszter kontextust, ott folytatódjon a munka, ahol megszakították. Általában a programszámláló regisztert (PC vagy IP) és az állapot regisztert (PSW) az *interrupt hardver* menti le. A többi regisztert az IT kezelő (szoftveresen) menti le.

Összefoglalva: megszakítás bekövetkezésekor hardveresen lementődik egy veremtárba a programszámláló regiszter és az állapotregiszter. Utána a megszakítás sorszámával indexelve az ugró vektor elemeit, a vezérlés a megszakítás kezelőjére ugrik. Itt az első instrukciók a további, az általános regisztereket mentik a verembe, majd ténylegesen elindul a megszakítás kezelése.

A kezelés végén először egy assemly rutin az általános regisztereket veszi vissza a veremről. Majd egy „visszatérés megszakításból” instrukció a státus szót és a programszámlálót emeli

fel. Utóbbi felvétele egyben a normális, aktuális instrukciófolyam végrehajtásához való visszatérés.

Felmerülhet bennünk a kérdés, vajon egy megszakítás kiszolgálását más megszakítás megszakíthatja? Hiszen a megszakítások aszinkron jellegűek, azaz nem lehet őket valamilyen isnstrukcióhoz kötni, bármelyik instrukció végrehajtása közben, bármelyik két instrukció végrehajtása között bekövetkezhetnek.

A mai – tulajdonképpen Neumann elvű - számítógéprendszerekben a megszakítások prioritási (fontossági) osztályokba vannak sorolva. A kevésbé fontos IT kiszolgálását megszakíthatja egy fontosabb megszakítás. Az ugyanolyan vagy magasabb fontossági osztályba sorolt megszakítás kezelését az újabban keletkezett megszakítás nem szakíthatja meg. Úgy is szoktuk mondani, hogy egy adott szintű megszakítás kezelése „lemaszkolja”, letiltja az ugyanolyan vagy alacsonyabb szintű megszakításokat. Sőt, rendszerprogramokban szoftveresen is „lemaszkolható”, letiltható adott szintű, végső esetben minden megszakítás. A maszkolás, a blokkolás megszűnik, ha az adott szintű megszakítás kezelése befejeződött, illetve, ha a szoftveres maszkolást szintén szoftveresen megszüntetjük, engedélyezzük a megszakításosztály kezelését.

A kérdésünk folytatódhat, mi történik, ha „letiltott” megszakítás következik be? Ez most nem szolgálható ki, vajon a megszakítás ekkor elvész?

A válaszunk nem. Nem vesznek el lemaszkolt, blokkolt megszakítások. Az megszakításokat kezelő, azokat a CPU felé közvetítő hardver képes „sorbaállítani” az egymásra következő, de lemaszkolt IT-ket. A bekövetkezett és kezelésre váró, hardveres sorokon tárolt megszakításokat *függő megszakításoknak* (pending IT) szoktuk nevezni. A megszakításokkal jelentkező igények nem vesznek el, előbb-utóbb sorra kerülve lekezelődnek. Mindez a prioritásoktól és a sorokban elfoglalt helyektől függően történik.

9.3. Az eszközök osztályai

Tulajdonképpen három nagyobb csoportba sorolhatók az eszközök.

Struktúrált (blokkorientált) eszközök az egyik osztály. Ide tartoznak a mágneses elvű és az optikai diszkek, a mágnes szalagok, kazetták. Általános jellemzőjük, hogy blokknyi (szektornyi) adatátvitel történik az eszköz és a vezérlő buffer, illetve a buffer és a memória között. Nem lehet a blokknál kisebb adatmennyiséget átvinni. A blokkorientált eszközöknél a blokk (szektor) méret kötött érték, azt nem a programozó választja meg. Az osztály nevében a blokkorientáció ezek után érthető. A strukturált eszköz elnevezést azért kapták eszek az eszközök, mert az eszközön tárolt minden blokknak (szektornak, klaszternek) egyedi címe van, az eszköz blokkstruktúrát ad nekünk.

Karakterorientált eszközök képezik a következő osztályt. Ebbe a csoportba tartoznak a terminálok, nyomtatók és rajzgépek, az I/O portok, a hálózati vezérlők, az érzékelő- és beavatkozó eszközök stb. A karakterorientált eszközöknél a legkisebb átvihető adatmennyiség a karakter, vagy bájt lehet. Természetesen, lehetséges ennél nagyobb egységekben is az adatátvitel és bizonyos esetben a programozó szabhatja meg az egy tranzakcióval átvihető adatmennyiséget. A karakterorientáltság mellet - kizárással - mondjuk, hogy ezek az eszközök nem strukturáltak. A valóság az, hogy bizonyos strukturáltság itt is megvalósítható. Pl. terminál eszköznél a sorvég karakter konvencionális értelmezéssel sorokra tördelhetjük a különben szer-

kezet nélküli adatfolyamot. Ettől függetlenül, nem szoktuk ezeket az eszközöket strukturáltaknak nevezni.

Végül a harmadik eszközcsoport a *különleges (speciális) eszközök* csoportja. Legjelentősebb tagjuk az óraeszköz. Az óra nem is igazi I/O eszköz, senem forrás, se nem nyelő. Programozható (beállítható) időközönként ún. óramegszakítást képes előállítani az óraeszköz. Az óra megszakítás (clock IT) kezelője képes a napi idő és dátum mezők „karbantartására”, képes eltelt idők nyilvntartására, adott időtartam letelte esetén riasztásra, ekkor esetleg ütemező szoftvetrek beindítására. Részletesebben az Operációs rendszerek c. tárgyban szólunk az óraeszközről.

9.4. Mágneslemezes tárolók

Ebbe a csoportba tartoznak a szokásos merev lemezek (hard disk), cserélhető csomagok, a floppy lemezek stb.

A mágneslemezes a szokásos *másodlagos tárolók*. Több fontos használati céljuk lehet: fájlrendszereket valósíthatunk meg rajtuk, használhatjuk rendszerindításhoz betöltési (boot) területnek, a központ memória „kiterjesztéseként” kisöprési-kilapozási területnek, néha mentési, illetve rendszerek közötti adatátviteli médiumnak.

Jellemzőik:

- A mágneslemezes tárolóknak nagyobb a kapacitása, mint a központ (fő) memóriának,
- az ár/bit jellemzőjük sokkal kedvezőbb, de lassabb az elérésük.
- Mágnesezettség megváltoztatásán alapul a tárolás: kikapcsolva sem felejtenek.

A mágneses jelrögzítés két, kapcsolatban álló fizikai törvényen alapul: az egyik szerint az elektromos áram mágneses mezőt hoz létre, és ezzel mágnesezhető anyagokban a mágnesezettséget megváltoztathatja, a másik szerint változó mágneses térben lévő vezetőben áram indukálódik. Az első a jelrögzítés, a második a jelkiolvasás alapja.

A mágneslemezes tárolók alapfelépítése a következő:

Forgó, nem mágneses korongokra vékony rétegben felhordott mágnesezhető anyag fölött *karok* mozognak, a karokon *író-olvasó* fejek a felületen „repülnek” (0.3-0.5 um), vagy „csúsznak”. Ahány mágnesezett *oldal* van a lemezes tárolón, annyi író-olvasó fej is van a tárolóban. A karok sugárirányban el tudnak mozdulni, és különböző átmérő (sugár) értékeknél pozícionálni tudnak az író-olvasó fejek. Egy-egy fejpozíció alatt elforduló mágneses sáv fontos fogalom, a neve *sáv* (track). Az *írófej* átmágnesezi a felületet, az *olvasófejben* az átmágnesezett felület fölötti mozgás közben áram indukálódik. A korongokat forgatómotor szabályozottan, állandó szögsebességgel forgatja. A karok együttes, sugárirányú mozgatása történhet lineáris mozdítással (sínek mentén), vagy nagyobb sugarú köríves mozgatással is.

Jegyezzük meg a következő alapfogalmakat, elnevezéseket:

Beszélhetünk *lemezoldalakról.* Egy meghajtón (mágneslemezes eszközben) több *oldal* is lehet, és ahány oldal, annyi fej szükséges az oldalakhoz. Az oldalaknak, ezzel együtt a fejeknek van címe: az *oldalcím* (fejcím). A szokásos oldalszám 2 és tízegynéhány között változik.

A *sáv* (track) egy koncentrikus kör egy lemezoldalon. A sávok sűrűn helyezkednek el a lemezoldalakon, és minden olyan koncentrikus kör, ahol a fej pozícionálni tud, egy-egy sáv. A sávoknak is van címe az oldalon. Ennek neve: *sávcím*. A sávok száma szokásosan néhány száz és közel ezer közötti. A sávok fizikai ívhossza nem állandó, a nagyobb pozícionálási átmérőhöz tartozó sávok fizikailag hosszabbak. Az állandó szögsebesség miatt a fejek kerületirányú sebessége a sávátmérő növekedésével arányosan növekszik, azaz ez sem állandó.

Az adattárolás a sávokon belül ún. *szektorokon* történik. A szektor (blokk) egy sávon belüli körcikk. Tíz és húsz között van a szokáso szektorszám egy-egy sávon. A szektorok között *hézagok* (gap) vannak. A szektorok ívhossza sem állandó, a sávjuk ívhosszától, annak átmérőjétől függ. A szektorok a fizikai méretüktől függetlenül azonos mennyiségű adatot tartalmaznak. A sávokon belül a *szektoroknak* is van *címük*.

Régebben a sávokra jutó szektor szám állandó volt, a nagyobb átmérőjű sávokon is ugyanannyi szektort alakítottak ki. Ezek a külső szektorok jóval nagyobb ivhosszúak voltak így. Az adattárolási kapacitásuk mégis ugyanannyi, mint a belső, kisebb ivhosszú szektoroknak. Újabban a külső sávokon a szektorszámot növelik. A sávokat 10-20 zónára (notches) osztják, az átmérő-különbségek szerint. Egy-egy zónában azonos a szektorszám, de a nagyobb átlagos átmérőjű zónákban a szektorszám növekszik.

Fontos fogalom a *cilinder*. A lemezoldalak egymás fölötti sávjait, melyek egy fejállással írhatók-olvashatók nevezzük cilindernek (cylinder). Ugyanazon a cilinderen tárolt adatok (szektorok, blokkok) sorozatos írása-olvasása gyorsabb lesz, mert fejmozgatás nélkül történhet a tranzakció. A *cilinderek címe* a sávcímekkel egyezik.

A mágneslemezes tárolókon az adatátvite legkisebb egysége a szektor (blokk). Az átviteli tranzakciót (szektor írást-olvasást) kérelmező szofvtvernek (a BIOS-nak, az operációs rendszerek magjához tartozó diszk-driver-eknek) valamilyen módon meg kell címeznia a starnzakcióban szerplő szektort. Az eddig elmondottak szerint ez történhet úgy, hogy megmondjuk, melyik oldal melyik sávjának melyik szektoráról van szó: címezhetünk *oldal-sáv-szektorcím hármassal*. A régebbi BIOS-ok valóban így is címeztek. Ha a lemezoldalak adott sorrendben, konvencionálisan számozzuk, egy-egy oldalon a sávokat is konvencionális (megegyezés szerinti) sorrendben sorszámozzuk, a sávon belül a szektorokat is, akkor a szektorok egy lineáris logikai sorrendje (lineáris logikai címe) előállítható az oldal-sáv-szektor címhármasból (és fordítva). Logikus megegyezés, ha az első oldal első sávjának a szektorait a második oldal első sávjának szektorai követik, utána a következő oldal első sávjának szektorai s.í.t. A cilinder utolsó oldal első sávja szektorait a következő cilinder első oldali sávjainak szektorai követhetik. Ezzel a konvencióval elérhettük, hogy egy cikinder szektorainak logikai címei sorozatot alkotnak. Ha a címhármasból a lineáris címekre történő (és vissza történő) cím-átalakítást a diszkvezérlő (újabban már magába diszkegységbe épített elektronika) megvalósítja, a mágneses diszk kívülről „úgy látszik”, mintha szektorok (blokkok) sorozata lenne, ahol a szektorok (blokkok) címe 0-tól n-ig tart. A BIOS, vagy az operációs rendszer magja csak annyit kér: igénylem az i-edik blokkot. A vezérlő (vagy maga a diszk) kiszámítja, hogy ez melyik oldal melyik sávjának hanyadik szektora.

A mágneslemezes tárolókon egy csatornán (a *sávon*) tudnak adatokat rögzíteni (szemben a szalagos tárolókkal, ahol műanyag alapú szalagra felhordott mágnesezhető rétegre egyszerre több sávon, több csatornán, párhuzamosaan történik a rögzítés/olvasás.)

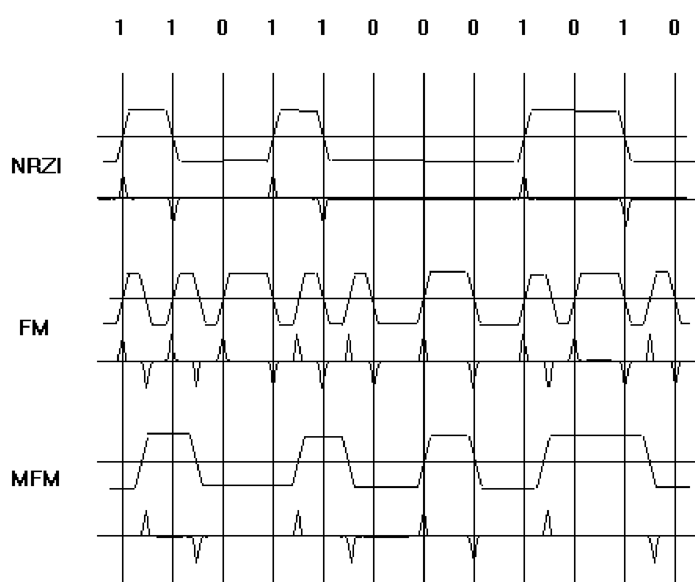
Adott hosszúságú sávban a tárolható információ mennyiséget az *írássűrűség* szabja meg, egységnyi hosszon elhelyezhető bitek számával jellemezhető ez. az írásűrűséget a mágneses anyag mágnesezhetősége és a fluxussűrűség határozza meg.

Nézzük az írást! Van valamilyen íróáram, ha abban változás van, akkor feszültség indukálódik, ez pedig fluxusváltozást hoz létre, ez változtatja a mágnesezettséget.

Kérdés, hogyan rögzítsünk egy bitsorozatot?

Szalagos tárolóknál megfelel a NRZI (Non Return to Zero Inverting) kódolás, ahol csak az 1 biteknél van fluxusváltás.

Mágneslemezeknél az FM (frekvencia moduláció), MFM (módosított FM), vagy RLL kódolás szokásos (9.2. ábra).



9.2. ábra. Kódolások mágneses rögzítéshez

Az FM kódoláskor rendszeresen érkező szinkronjeleket írnak fel, adott frekvenciával. A szinkron jelek között ha 1 bitet akarunk írni, a beíróáram frekvenciája kétszerese a 0 bitek beírási frekvenciájának. Pl. 1 bitet 2 pulzussal, 0 bitet 1 pulzussal kódolunk, az átlag így 1.5 pulzus/bit.

Az MFM kódolásnál az aktuális bitet megelőző bit értékét is felhasználjuk. Így 0.75 pulzus/bit átlag lehet az eredmény. Ekkor, az 1 bit kódolásához az aktuális és a következő szinkronjel között mindig van áramszintváltás. A 0-ás bit kódolása a megelőző bit értéktől függ:

- ha előtte 1 bit volt, a szinkronjel pillanatában nincs váltás;
- ha előtte 0 bit jött, a szinkronjel pillanatában van váltás.

A diszk-blokk tarnzakció (írást-olvasás) teljesítményét (sebességét) három tényező befolyásolja. A tranzakció ideje három időegységből tevődik össze. Ezek

- a fej megfelelő sávra (cilinderre) való mozgatási ideje (seek time),
- az az idő, míg a megfelelő szektor elfordul a fej alatt (rotation latancy);
- magának az ellenőrzött adatátvitelnek az ideje (data transfer time).

A három időből a fej mozgatási idő a legnagyobb, az adatátviteli idő a legkisebb. Érdemes a keresési időt (seek time) optimalizálni! Az optimalizálásra a diszkegységek egyre nagyobb intelligenciája ad lehetőséget.

A diszkekhez a tranzakciós kérelmek „felülről” (a BIOS-tól, az operációs rendszer magjától) jönnek sorozatban. A kérelmeket az eszköz ideiglenesen tárolhatja, sorbaállíthatja, optimális

sorrendben szolgálhatja ki, azaz nem feltétlenül a kérelmek sorrendjében. A kérelmek sorrendjének optimalizálását ún. *diszk ütemező algoritmusok* (disk scheduling algorithms) végezhetik. A befutó kérelmekben ugyan a blokkok lineáris logikai címe szerepel, abból az optimalizás szempontjából fontos cilinder (sáv) cím könnyen kiszámítható (amúgy is kell majd ez a cím a fejpozícionáláshoz). A befutó kérelmeket a cilinder-címek szerinti valamilyen sorrendbe rendezik és tárolják (általában láncolt listákon), és e sorrend szerint történik a kiszolgálásuk.

A legegyszerűbb „rendezési" algoritmus az *először az elsőt* (FCFS, First Come First Served) rendezés. Amilyen sorrendben jöttek a kérelmek, olyan sorrendű a kiszolgálás. Ez az algoritmus nem is optimalizál.

Optimalizáló diszk ütemező algoritmus a *legrövidebb mozgatási idejűt előbb* (SSF, Shortest Seek First) algoritmus. Tapasztalat szerint a fejmozgatási idő annál kisebb, minél közelebb vannak egymáshoz a sávok, amik a mozgatásban résztvesznek. Az algoritmus azt „mondja", rendezzük a kérelmeket úgy, hogy egy-egy fejálláshoz legközelebbi fejállás legyen a sorban a következő.

Optimalizáló algoritmus az ún. *lift algoritmus*. Nevét az egyirányban gyűjtő felvonók működési módjához való hasonlóság miatt kapta. Alapja az a tapasztalat, hogy az író-olvasó fejek mozdításaiban a mozgási irányváltás időigényes. Az azonos irányba mozduló fej mozgási idejei esetleg távolabbi sávokra való mozdulásoknál is kisebbek lehetnek, mintha irányt változtatva kisebb távolságra mozdulva történne a keresés. Érdemes a kérelmeket úgy rendezni, hogy a valamelyik irányba elmozduló fej „gyűjtse be" az irányba eső kérelmeket, s csak a végén „forduljon vissza", a maradék kérelmek kiszolgálására.

A mai, korszerű diszkegységek teljesítményét a *diszk-gyorsítótárak* (*disk cache*) egységekbe történt integrálása is segíti. A diszkek átmenetileg tárolják az utóbbi időben kért tranzakcióik eredményeit, és ha a lokalitás elvének érvényesüléséből újabb igény merül fel a gyorsítótárban is meglévő blokkra, akkor onnan a kiszolgálás gyorsan megtörténik. Figyeljünk fel, hogy ez a gyorsítótár más, mint a klasszikus „cache" memória, sőt, más mint az Operációs rendszerek tantárgyban majd emlitendő „buffer cache" (bár ez is a diszk blokk elérész gyorsítja). És persze, más, mint az ESDRAM memórai-lapkán található cache … Persze, mindegyik gyorsító mechanizmusban (cache-elésben) a lokalitás elve a lehetőség!

A diszkek általános tárgyalása során szólnunk kell a winchester diszekekről. Ezek mágneslemezes tárolók, ahol is a mechanika és az elekronika zárt dobozba foglalva – ezzel szennyeződéstől védve- szerelt. Szabványos méretek, egyre nagyobb tároló kapacitások adottak a diszk egységeket vásárlók számára.

Ugyancsak szólnunk kell, hogy a mai személyi számítógépekhez három interfészen át kapcsolhatunk diszkeket.

A mai gépek alaplapján megvalósított EIDE vezérlő adja a legolcsóbb megoldást. A vezérlő két csatornával (primary and secondary channels) rendelkezik, mindkét csatornája 2-2 eszközt képes vezérelni. Az eszköz lehet mágneses diszk, vagy CD olvasó is. Ügyelnünk kell arra, hogy egy csatornára kapcsolt CD ronthatj az ugyanarra a csatornára kapcsolt W diszk teljesítményét.

Másik lehetőség a SCSI vezérlős diszk vásárlás. Külön vezérlő kártyát kell vásárolnunk, és speciális SCSI diszk egységeket. Drágább megoldás, de a többletköltséget kompenzálhatja a nagyobb teljesítmény (a gyorsaság) és a megnövelt megbízhatóság a tartósság.

Harmadik lehetőségként vásárolhatunk USB interfészű diszkeket is. A teljesítmény valamivel gyengébb, de a mcsatlakoztatás – akár külső diszkként - nagyon egszerű.

9.5. CD (Compact Disk) lemezek

A CD technológia zene rögzítésére alakult ki a 80-as évek elején. Hamarosan elterjedt a számítástechnikában is ez a rögzítési mód. Az iparág vezetői 1985-re hozták létre a technológia szabványait.

A működési elvéhez a lézer technológia (LASER: Light Amplification by Stimulated Emission of Radiation) tartozik (fényerősítés a sugárzásnak gerjesztett emissziója révén). A lézer berendezés segítségével nagy energiasűrűségű és szigorú párhuzamosságú (monokromatikus: egyszínű) fénynyaláb állítható elő. A mai CD-k többsége a színspektrum kisebb frekvenciákhoz tartozó tartományát (pl. vörös, sárga) használják. Lézerfény-nyalábot vetítenek egy felületre, és a fény visszaverődésének, vagy vissza nem verődésének érzékelésével kódolják a bináris 0-ákat és 1-eket. A lézer szigorú párhuzamosságából következően nagy sűrűségű lehet a rögzítés: igen közel lehetnek egymáshoz a visszaverő és a nem visszaverő felület-elemek. Természetesen, gondoskodni kell az lézerfényt kibocsátó és a visszavert fényt érzékelő olvasófej megfelelő „mozgatásáról” a rögzítő felülethez képest.

Az adattároláshoz a CD lemezeken spirál alakú sávot használnak (nem koncentrikus körök a tároló entitások). A spirálon – meglehetősen sűrűn – fény visszaverő ép felület-elemek és nem visszaverő, roncsolt felületelemek vannak (lands & bumps). A bájtok kódolás ún. EFM (Eight to Fourteen Mode) kódolás, a bájt 8 bitjét 14 bitté konvertálják. A spirális sávon 2048 bájt adatot tároló szektorokat képeznek. Minden szektor eleje egy 12 bájtos szinkronmezőt és egy 4 bájtos fej mezőt tartalmaz. A fejléc a szektor címét rögzíti, méghozzá lejátszási perc-másodperc-századmásodperc formában (látszik a zenei rögzítési örökség), továbbá az ún. kódolási módus kódját (ami 1 vagy 2 lehet).

Az 1-es módusú kódolásnál a szektorok végén 288 bájtos hibadetektáló (EDC, Error Detection Codes) és hibajavító (ECC, Error Correction Codes) mező is van. Így az 1-es módusnál a valódi szektorhossz 12 + 4 + 2048 + 288 = 2352 bájt. Ez használatos adattároláshoz. A 2-es módusú szektornál nincs ez a mező, a szektorhossz itt 12 + 4 + 2048 = 2064 bájt. Hangrögzítésre felel meg ez a kódolás. A CD teljes kapacitását a szektorhossz és a spirális sávon lévő szektorszám (szokásosan 270 000 körül) határozza meg.

Szólni kell még a sebességekről is. A CD lemez fogatását és az olvasó fej mozgatását vezérlő elektronika állandó kerületi sebességet (CLV, Constant Linear Velocity) biztosít a spirális sávon az olvasófej számára. Ezzel a szektor ívhosszak fizikailag is állandók. Az alap kerületi sebesség kb. 75 szektor/sec., ehhez a CD lemez kb. 200 – 530 fordulat/perc közötti szögsebességgel forgatandó. A CD meghajtók fejlődésével az említett alap sebességet többszörözik, így beszélhetünk 2x, 4x stb. többszörös sebességű CD meghajtókról. A hagyományos zenei felvételeket az alap sebességgel kell lejátszani, és a 6X, vagy az a fölötti sebesség lehetővé teszi a videofilmek életszerű lejátszását.

A CD lemezek 120 mm átmérőjűek. (Az ún. Photo CD kisebb átmérőjű.) Az acryl korongra alumínium lemezt préselnek, ezen kialakítva a fény visszaverő, vissza nem verő elemeket, majd polikarbonát fényáteresztő lakkal védik az alumínium réteget.

9.6. Terminálok

Manapság CRT (Cathode Ray Tube) eszközök, esetleg folyadékkristályos kijelzők az outputra, billentyűzet és mutató eszköz az inputra.

A CRT működése: memóriában tárolt képpont információkat (vagy karaktereket) használnak a CRT elektronsugár modulálására. A hagyományos TV technikához hasonló, a képcső elektronsugara a képmű videojele alapján, a bal felső saroktól indulva balról jobbra, felülről lefelé halad (Először a páratlan sorokat vetíti ki, majd a párosokat.) Az elektronsugár "erősségét" (a képpontok fénysűrűségét) a videó memóriában tárolt adatok határozzák meg.

Színes monitoroknál három különböző (RGB) foszforréteg van, melyeket három együttfutó, de különböző intenzitású elektronsugár gerjeszt. A színek színkeveréssel állíthatók elő.

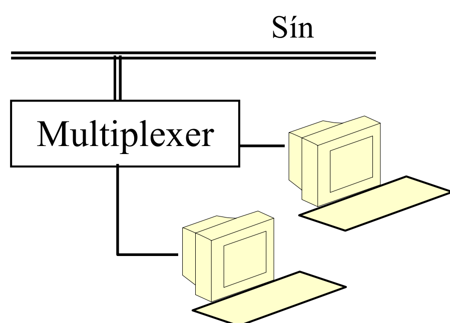
A folyadékkristályos (LCD) képmegjelenítés alapja az, hogy bizonyos kristályok elektromos tér hatására fénytörésüket megváltoztatják. A kristályok: rúd alakú molekulák, folyékony halmazállapotukban dipol jellegűek: elektromos tér hatására kristálysíkonként kicsit elfordulnak, ezzel szűrőként viselkednek. A szűrőn csak megfelelő polarizációjú fény tud áthaladni, visszaverődni.

A kristályokat pontonként ki-bekapcsolható rasztertáblába szervezhetjük, az megfelelő megvilágítással (esetleg saját háttérvilágítással) már elég jó kontrasztos képet biztosít. Sorokra és oszlopokra bontottak a képpontok: egy pontot gerjeszthetünk.

A kérdés: hogy jutnak az információk a CRT vagy LCD "saját memóriájába" (képmemóriába, videó memóriába)?

A klasszikus terminál (buta terminál)

Soros vonali meghajtón (vezérlőn), vagy multipexeren (nyalábolón), vagy "terminál szerveren", duplex, half-duplex, simplex vonalon (porton) van a terminál. Saját memóriájának semmi köze a gazdagép memóriájához!



9.5. ábra. Klasszikus terminálok

A "vonalon" karakter (bájt) sorozatok átvitele történik. A bájtok lehetnek

- megjeleníthető karakterek, képpontok;
- vezérlő szekvenciák.

A terminálok szabványosak lehetnek, pl. ANSI, VT 100, VT 200 stb.

A szabvány megmondja:

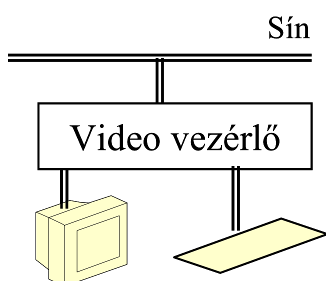
- milyen karakter kódolást használnak (pl. ASCII),
- Milyen koordináta rendszerük van,

- mik a vezérlő szekvenciák és hogy kell ezekre reagálni...

Ha úgy tetszik: a szabványos terminálokat "programozhatjuk"; pl. *cursor* mozogjon adott koordináta pozíciókra, legyen inverz, villogó stb. a kijelzés stb.

Ismerjük meg az ANSI és a VT 100 terminál szabványt!

A memória-leképzett (memory mapped) terminálok



9.6. ábra. Memória leképzett terminál

A CRT-nek nincs saját memóriája, az a vezérlőn van, és a CPU címtartományának része (9.6. ábra). Kicsit leegyszerűsítve: a

```
MOVE karakter CRT-memory CPU
```

instrukció megváltoztatja a megjelenítést.

A vezérlőn itt is vannak adat és vezérlő regiszterek, továbbá nagy memória a képpontok(pixel)/karakterek megjelenítéséhez.

Több szabvány van: CGA, Herkules, VGA, SVGA stb.

- Több szintű lehet a kezelés:
- közvetlen memóriába írás MOVE ill. IN/OUT instrukciókkal. Ez a legalacsonyabb szint. Veszélyes, mert széteshet a vezérlés.
- IBM PC-ken a BIOS rutinok használatával (SW interrupt-okon át). Alacsony szint ez is.
- IBM PC-ken a DOS rutinokon, SW IT-ken) át. Ez is alacsony szint.
- Ún. "driver" programok segítségével. Betölthető pl. az ANSI driver: ezzel a memórialeképzett terminál is a szabványos ANSI terminálként viselkedik.

További perifériák, működésük

Nyomtatók lehetnek

- Ütő típusúak
  - Folyamatos jelűek
    - Íróрudas, láncos;
    - Betűkerekes, gömbfejes
  - Pontmátrix
    - Tűmátrix
- Nem ütő típusúak
  - Pontmátrixos
    - Hő;
    - Tintasugaras;
    - Mágneses;
    - Elektrosztatikus;
      - Lézeres,
      - Ionsugaras.

A tintasugaras nyomtatók

Porlasztókból finom tintacseppeket juttatunk a papírra. 0,025mm a csepp átmérője, sebessége 700km/óra körüli. Becsapódásakor 0,16mm-es pont az eredmény. 4000 csepp/sec érheti a papírt.

Az igazán érdekes a porlasztás! Három mód szokásos:

- Piezzoelektromos kristály - alakját változtatja feszültség impulzusra - nyomást okoz a fúvókában, kilő a csepp (tintacseppes nyomtató).
- A festékből kiváló gőzbuborék növeli a nyomást (fűtőelem felizzik, ez gázkiválást okoz).
- Folyamatos sugarú nyomtató. Hasonló a CRT-hez, de itt a cseppeket elektrosztatikusan feltöltik, majd elektrosztatikus térben gyorsítják, vezérlik a sugár irányát, letiltják (és visszavezetik a tartályba), ha nem kell festeni.

Még egy színű nyomtatónál is több porlasztó van.

Elektrosztatikus nyomtatók

Közös elv: van egy homogén töltésű dobjuk. Ezen fénnyel, vagy ionnyalábbal töltésmintát alakítanak ki. Ez az elektrosztatikusan feltöltött festéket magához vonzza. Utána a dobról a papírra viszik a festéket (nyomóhenger segítségével, vagy ellentétes tér hatással). Ezután rögzítik a festéket a papíron (pl. hő segítségével). Végül a dobot "tisztítják", a felesleges festéket (ami nem ment át a papírra) egy késsel leszedik.

A lézernyomtató gyenge lézernyalábbal tölti a dobot. Mozgó tükör fut a dob alkotóján, a fény hatására a dob alkotó pontjai kisülnek (vagy sem). Fényérzékeny szelén henger a dob.

Az ionsugaras nyomtatónak különleges bevonatú alumínium hengere van. Ezt vezérelt ionnyalábbal töltik. A festéket vonzzák egyes töltött pontok. Nyomóhengerrel viszik át a festéket a papírra.

Billentyűzetek

Benyomást - felengedést kell érzékelni. A mechanikus érintkezés nem jó: elfárad, kopik az anyag. Manapság

- optikai (a billentyű lenyomás fénysugarat szakít meg, melyet optoelektronikai elemek érzékelnek);
- Hall effektuson alapuló (billentyűn pici állandó mágnes van. Lenyomáskor egy speciális kristályhoz közelít, ebben változik a mágneses tér, ez változó elektromos teret hoz létre: ez már érzékelhető);
- Reed-elemes (kis mágnes közelít tokba zárt elektródákhoz, melyek a mágneses tér hatására érintkeznek) billentyűzetek a gyakoriak.

Egerek

Egyik típus a golyós: a golyó gördülését két tengely körüli forgásra bontják. A két tengelyen mérőtárcsák elfordulása adja a jeleket. Nézzék a gyakorlaton!

Rajzgépek

Nem raszteres, hanem vektoros a grafika. Két tengelyen mozgás, parancsokkal, koordináta értékek megadásával mozog a rajzoló toll.

Dob plotterek: egyik tengelyen a papír mozog, másikon, egy szánon a toll. Nagyon érzékeny a papírminőségre.

Sík plotterek: mindkét vonalon a toll mozog. Kevésbé érzékeny a papírminőségre.

Vezérlő nyelvek közül híres: HPGL nyelv. Parancsaival lehet

- színt, tollat kiválasztani,
- pozícionálni,
- tollat letenni, felemelni,
- koordináta rendszert transzformálni stb.

Ma már kisebb méretekben nem érdekesek, helyettük ott vannak az elektrosztatikus nyomtatók. Nagyobb méretű rajzok készítésben viszont van szerepük.