MG

INFORMATIKA

I

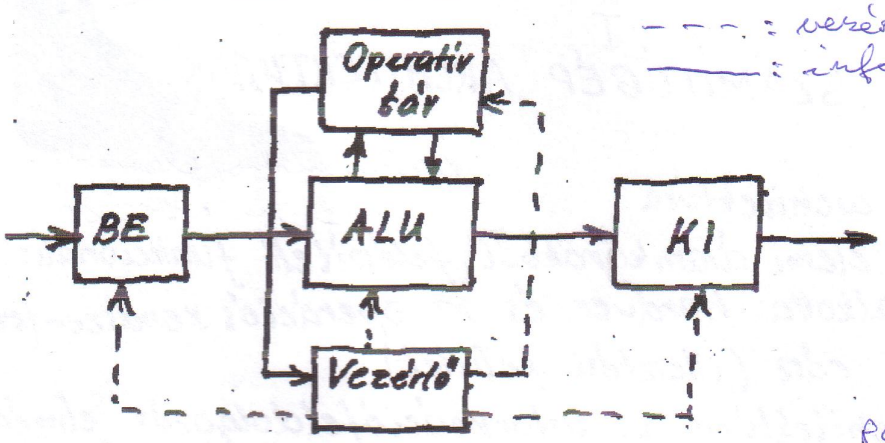
SZÁMITÓGÉP ARCHITEKTURÁK

funkcionális elemek összeillesztése ⟺ hatékonyság

— Sz.g architektúra
az elemi áramkörökből felépített funkcionális egységek alkotta hardver és az operációs rendszer (szoftver) közötti rész (illesztési felület)
Az architektúra az információfeldolgozás elméleti modelljeinek konkrét megvalósulása funkcionális elemekkel

— Technikatörténeti áttekintés
Blaise Pascal (1623-1662) mechanikus 6 digit ± — összeadó/kivonó gép
adószámítás
Charles Babbage (1792-1871)
Difference Engine mat. táblázatok automatikus számolása
Analytical Engine // elnevezésbeli különlegességek
ada-programnyelv
Malom CPU
tár Mem
nyomtató lyukkártyás bemenet
funkcionális elemek kb. u. azok
a költő lánya, másolatja
terv jó, de nem tudja megcsinálni
Ada Byron példaprogram — papíron írt példaprogram a csak tervben létező gépre ⟹ első programozó: NŐ.
Herman Hollerith (1860-1929)
elektromos, lyukkártya alapú gép
amerikai népszámláláshoz készült
Howard Aiken (IBM 1943)
MARK-I telefonreléken alapul
Neumann János
1946 Neumann-Goldstine tanulmány
EDVAC 1947-52 gyakorlati megvalósítás
Maurice Wilkes EDSAC, (Anglia) 1949
Electronic Delay Storage Automatic Calculator
1951 mikroprogramozás elve
statikus tár
indexregiszter MADM
mágnesszalagos tár UNIVAC

Neumann modell

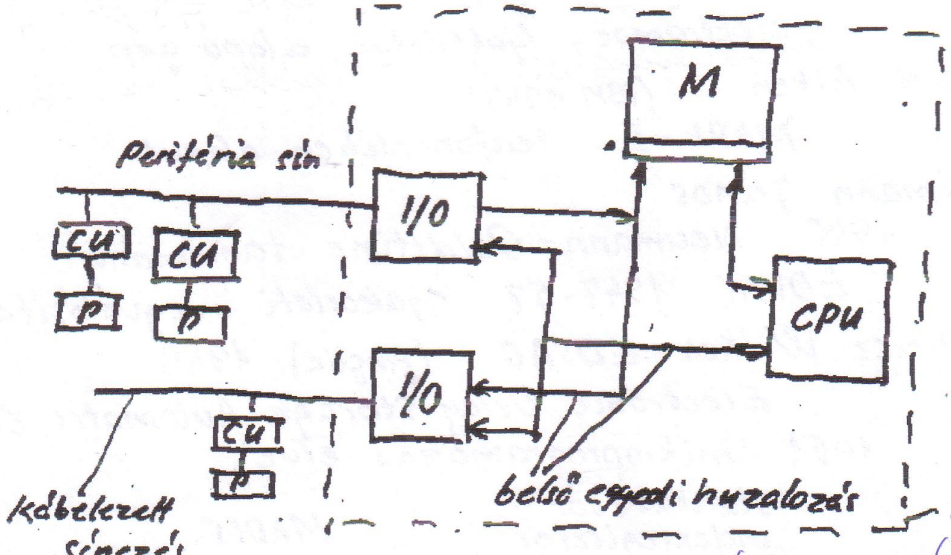


- - - : vezérlés útja
——— : információ útja

PC regiszter

alapelvek:
- belső programtárolás, illetve program vezérlés
- utasítás és adat azonos közegen és formában tárolva
  (értelmezés algoritmus illetve PC szerint)
  → utasítások programmal módosíthatók
  → az adattípusok műveletekhez rendeltek
- szekvenciális utasítás végrehajtás
- egydimenziós, lineáris címzésű memória
- bináris ábrázolás

Fejlődés:
- ALU + vez. egység ⇒ processzor (CPU)
- Külső kapcsolatok, perifériák leválása, KI-BE összevonása
- tárcentrikus felépítés
- I/O csatornák kialakítása



70-es évek: MSI - áramkörökből épített számítógép

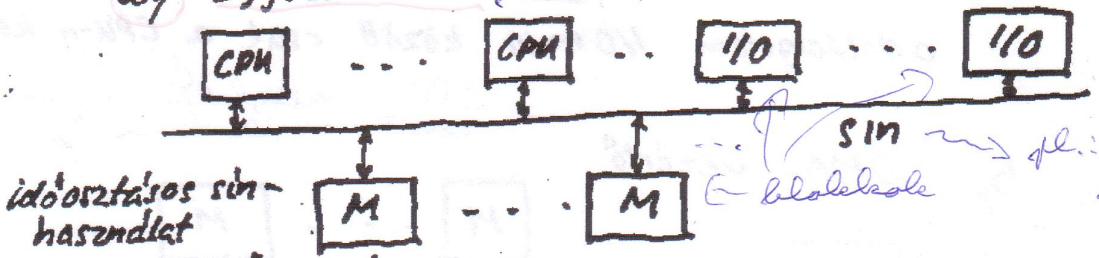
Ker első processzorokból nem építettek számítógépet, programozható feladatvégrehajtásra használták elsősorban

386-tól ~~is~~ van kb. ált. célú számítógép

– Modularizáció
tipikus egységek: CPU Mem. I/O
hogyan kapcsolhatók össze egy rendszerbe

szélső megoldások:

a) egyszintű sín: erre fűződik fel mindent



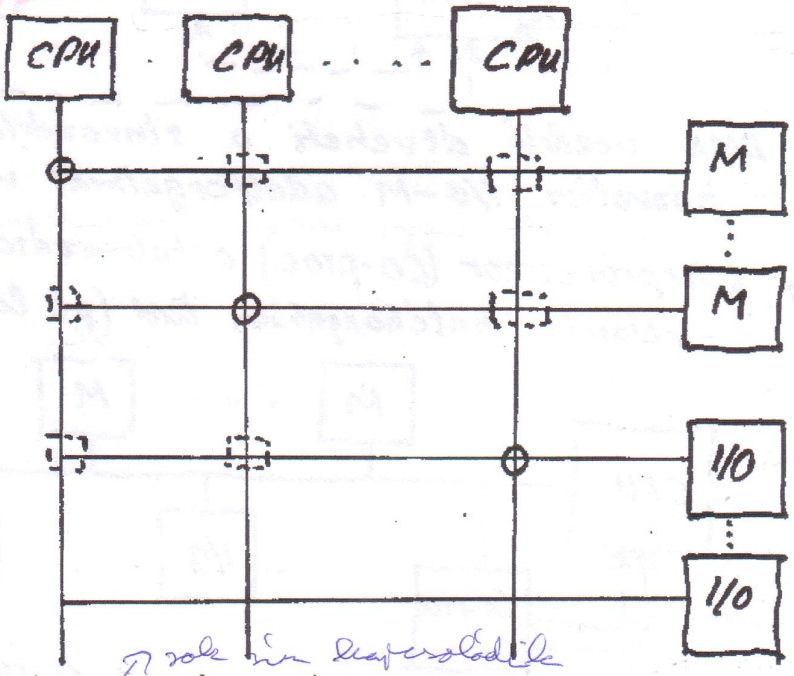
→ blokkolás

pl.: PDP minicomputer

időosztásos sín-használat

c: egyszerű, olcsó
h: szűk keresztmetszet ⇒ a sín a szűk keresztmetszet, ha a sín telítődik, a CPU-k minusenek kihasználva várakoznak

b) Crosbár



sok sín kapcsolódik

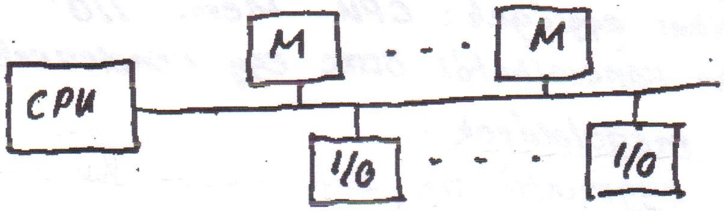
Kétdimenziós rács, csomópontokban kapcsolók
→ bonyolult drága

'közbülső' kompromisszumos megoldások
pl dual-port memória
hierarchikus sínrendszer ⇒ hierarchiában több fajta sín van; magasabb a adatcsomagolás prioritásnál!
⇒ egyszerűbb kialakítás

↓ memóriatömb szétválasztásából hozzáférhető, 2 sín vezet hozzá pl.: egyszerre két feladat, folyamat használhatja a memóriát

µP-s architektúrák fejlődése

a)

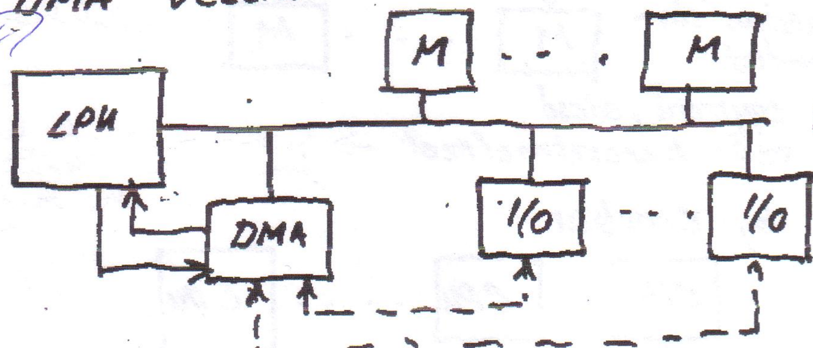


adatforgalom I/O ⇔ M között csak a CPU-n keresztül

CPU-t lefoglalja a nagy adattömbök mozgatása

b) DMA vezérlő

már a régi elektroncsöves gépekben is volt; most a µP-s gépekről van szó!



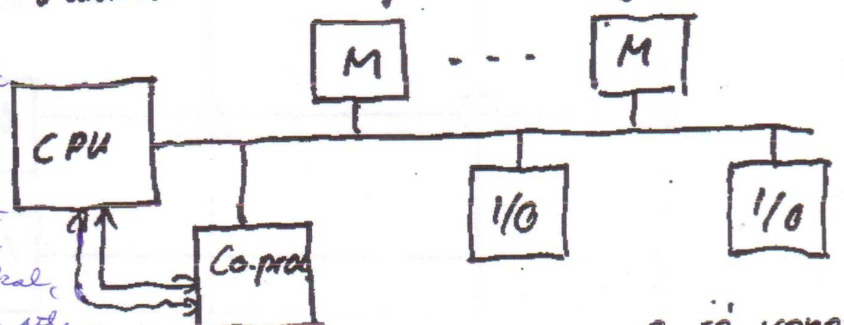
DMA vezérlő átveheti a sínvezérlést és közvetlen I/O–M adatforgalmat vezérel

adatmozgatás nem a CPU-t terheli

gyorsabb: ugyan abban a ciklusban olvas az egyikből és ír a másikba egyidejűleg

c) társprocesszor (Co-proc.) alkalmazása
- valamit hatékonyabban tud (pl leb. pontos aritm.)

pl.: lebegőpontos regiszterek csak az aritmetikai társprociban voltak;
co-proci: párhuzamosan volt kötve a másikkal, felismerte, h az utasítás az övé, neki kell végrehajtani



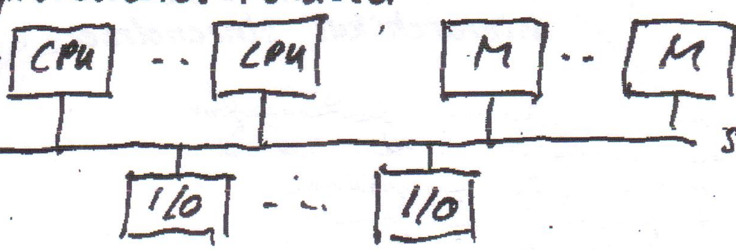
/pl aritmetikai grafikus ... stb

a rá vonatkozó utasításokat ez hajtja végre

386/486-nál is volt társ aritmetikai proci

drágább kivitelben benne volt; olcsóbban külön IC-n volt az aritmetikai proci

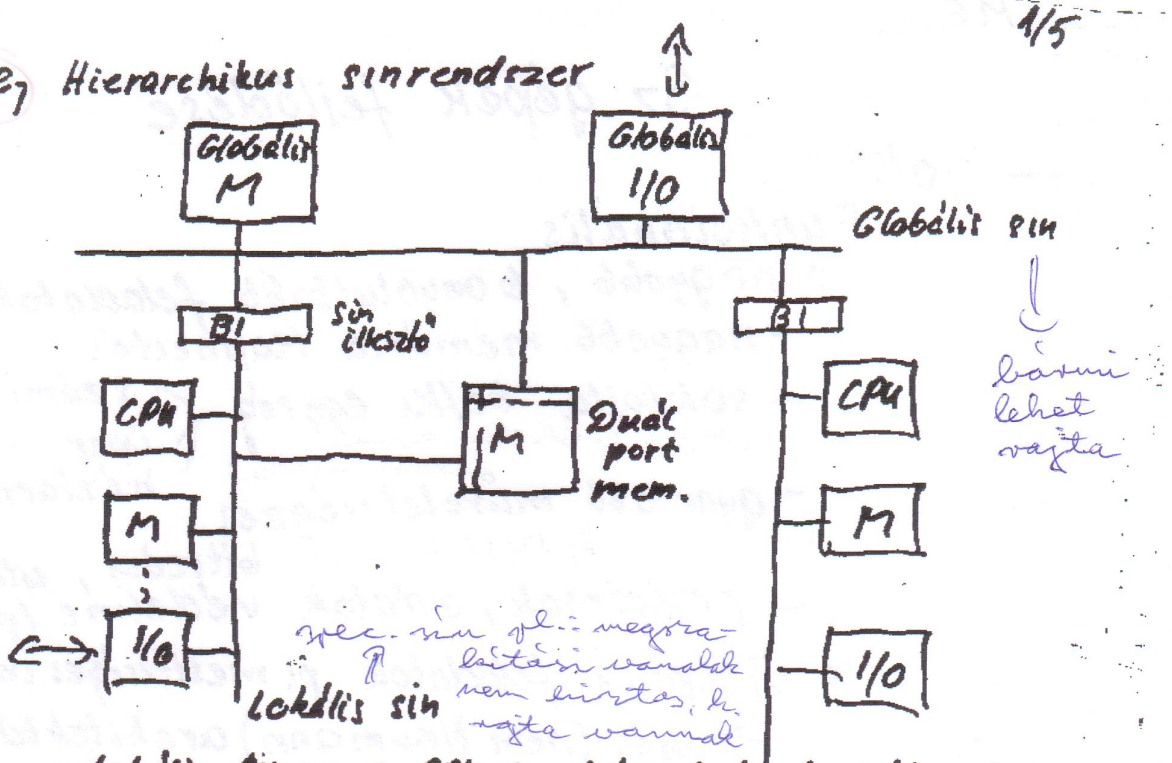
d) többprocesszoros rendszer
- buszvezérlésért verseny
- szűk keresztmetszet



MG

1/5

e) Hierarchikus sínrendszer



- lokális sínen a CPU-k dolgozhatnak akkor is ha a globális sín foglalt ⇒ gyorsulás; munkamegosztás

Sz. gépek fejlődése

okok:

Funkcionális

- o nagyobb, bonyolultabb feladatok megold.
  - nagyobb memória kapacitás
  - sokfajta be/ki egység (egyre több területen alkalmazzák) ← számítástechnika, ipar, háztartás, stb...
  - gyorsabb műveletvégzés
  - bitszám, utasítás (8; 12; 16; 32; 64)
  - programok, adatok védelme (pl bank)
- o új típusú feladatok pl mesterséges intelligencia

⇒ más (nem Neumann) architektúra

Technológia:

- több elem egy IC-ben ⇒ új IC-k (nő az integráltsági fok & csökken a vonalvastagság pl.: 90 nm, 65 nm)
- szoftver fejlesztésre szisztematikus technológia

⇓

Hardver, szoftver fejlődés egy határon túl ⇒ új architektúra

⇒ több képesség ⇒ több új igény

CISC: Complex Instruction Set Computer

RISC: Reduced Instruction Set Computer

Sz. gépek teljesítőképessége függ:

- alkalmazott áramkörök sebességétől
- sz. gép szervezésétől
  - funkcionális egységek belső felépítésétől (CPU, MEM, Be/Ki)
  - funkcionális egységek külső kapcsolódása
  - utasítás készlet, stb
  - kiegészítések (pl DMA; társprocesszor)

pl Utasítás készlet

CISC		RISC
sok, komplex	utasítás	kevés, egyszerű
sok	címzési mód	kevés

(sok bonyolultabb címzési módok)

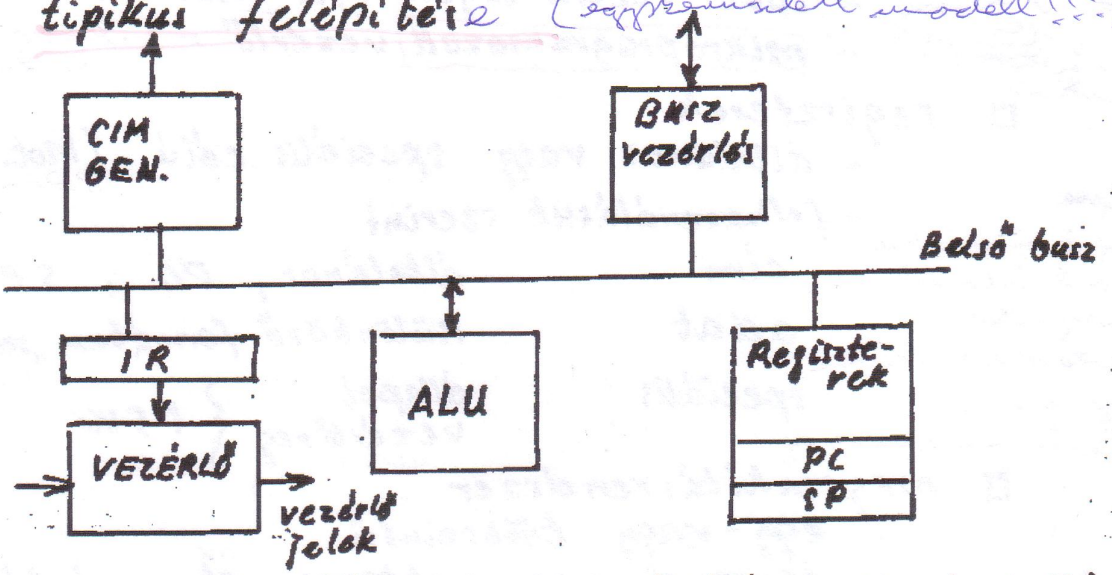
(primitívebbek az utasítások, de ezeket sokkal gyorsabban hajtja végre; pl.: memóriaelérés: csak LOAD és STORE utasítás van, egyéb mozgató utasítások nincsenek)

MG

2/1

CPU-K

tipikus felépítése (egyszerűsített modell !!!)



CPU feladata: tárolt program utasításainak végrehajtása;
e.g. részegységei közötti adatforgalom vezérlése

funkció	modul
• mem. – CPU forgalom utasítás elővétel operandusok írása, olvasása	címgen., busz vezérlő
• utasítás végrehajtás	ut. dekoder, ALU, Regiszt.
• periféria – CPU forgalom nincs, ha memóriába ágyazott be/ki	periféria sín vezérlés
• megszakítások kezelésére	megszakítás logika

Egyéb
- busz arbitráció (pl DMA)
- társprocesszor vezérlés
- virtuális tárkezelés
- gyorsítótárak

I/O címen érünk el memóriát

MG

2/2

- utasításdekóder (vezérlő egység)
  - spec. vezérlő logika (fix huzalozású) => gyorsabb
  - mikroprogramozott vezérlő => könnyen bővíthető, de lassabb működés

* => ha csak spec gépek nem mikroprogramozottak

- regiszterek
  - általános vagy speciális célú (Mot. – Intel) Motorola
  - felhasználásuk szerint

cím	általános, PC, SP
adat	különböző formátum „szélesség" (szélesség)
speciális	állapot, vezérlőreg (vezérlőreg) } PSW

SP = pre auto inkremens, post auto dekremens regiszter

A Motorola ∀ regisztere tudja ezt, pl.: 8 címreg 8 adatreg

- megszakításrendszer
  - egy- vagy többszintű
  - egyszerű vagy vektoros ok meghatározás
  - megszakítás érvényre jutása
    - ut. ciklus végén
    - CPU-n engedélyezve van
    - legnagyobb prioritású

Utasításrendszer

„SW architektúra" ← az utasításkészlet

CISC		RISC ← utasításkészlet tervezési elvek
sok, komplex	– utasítás –	kevés, egyszerű
sok	– címzési mód –	kevés

CISC

Az utasításrendszer tervezési szempontjai

általánosabb utasítások

- Kódsűrűség növelése: a lehető legkevesebb biten legyenek megoldva az utasítások
- ortogonalitás: az utasítástípus és operandus címzési módja független
- szisztematikus kódolás => a belső architektúra egyszerűbb
- Kompatibilitás meglévő utasításrendszerre
- magasszintű funkciók támogatása, operandusok támogatása

Motorola jobb e szempontból

hibakeresés könnyebb

összes x86-os-on futnak a régi kódok, kompatibilitás

bármilyen utasítás bármilyen utasítással lehet együtt

* utasítások felbontása elemi lépésekre => ezek a mikroutasítások

pl.: a belső sínre kijusson a egyik belső regiszter tartalmát

⇓

lesz az utasítás fázisai egy-egy mikro utasítás

mikroprogram
- egyszintű: ∀ bitje egy vezérlő jel a rendszerben => assembly utasítás: mikroprogram kezdőcíme
- kétszintű: az első szint csak címez egy második vezérlőtárat

Utasításrendszer jellemzése

- utasítás részei
  - műveleti kód (mindig van)
  - operandusok
  - köv. ut. címe (elmaradhat!)
  - operandusok + köv. ut. címe → címek (4 db) / mikroprogramoknál 4. cím

- 4 címes: 2 operandus (1. & 2.), eredmény (3.), köv. ut. címe (4.) (csak mikroprogramozott vezérlőben)

PC-t használnak ill. vezérlésátadó utasítást pl.: JMP → PC, JMP

3 címes: 2 op. (1. & 2.), eredmény (3.) (ritka) [illegible]

↓ eredmény az egyik op. helyére

legegyszerűbb → 2 címes: 2 op. (1. & 2.) // az eredmény valamelyik operandus helyére kerül pl.: 8085

↓ 1 op kijelölt pl AC

1 címes: 1 op, AC // egyik operandus fix helyen van pl.: AKKU

↓ másik regiszter

1.5 címes: 1 op, Reg // 1. egyik cím nem fix 2. valamely regiszterben van

ritka → 0 címes: operandus fix helyen pl. (verem tetején) → mindig fix hely

- Funkciók szerinti csoportosítás
  - Adatok kezelése
    - adatmozgatás (move, exchange) MOV; MVI
    - műveletek adatokon — pl.: Motorola: kivitelezni is ont(?) szoroz
      - aritmetikai (+, −, * .. különböző adatformátum: fix pontos, lebegőpontos, BCD, string) (milyen adattípuson végezhető)
        - 486-ban már lebegőpontos regiszterek
      - logikai (AND, OR, NOT ...
      - konverziók (különböző adatábrázolási formák között) — 1 db gépi utasítás ami konvertál
      - bitműveletek (léptetés, forgatás ...
  - Vezérlésátadás (feltételes – felt. nélküli)
    - ugrás (JMP); relatív ugrások; indirekt ugrás → PC tartalom cseréje valamely másik reg. tartalmával
    - szubrutinhívás
    - iteráció (hurok); ciklusutasítás: regiszter tartalmát változtatja és összehasonlítja egy másik tartalmával
  - Egyéb
    - halt
    - interrupt tiltás/engedélyezés

M6 — 2/4

Címzési módok

> a cím számítását egy helyről vett komponenssel végezte

- 1 komponensű
  - Közvetlen
    - Közvetlen memóriacím — abszolút adat (abszolút címzés LHLD; STA) / közvetlen adat (immediate) — rögtön az adat is ott van az utasítás után MVI, LXI stb.
    - Közvetlen regiszterkijelölés — regisztercímet adok meg
  - indirekt (közvetett)
    - indirekt mem. cím — ut. címrésze → memóriarekeszre ahol a cím van
    - indirek regiszter címzés
    - {pre/post} auto {inkremens/dekremens} címzési mód (pl.: 8085-ben SP működik így! Ha tárolásnál ∨ címregiszter)
    - magába foglalt (implicit) pl veremműveletnél SP — SP kezelés, nem kell külön megadnom a címet

1. az operandus címe egy regiszterben van
2. ahova a cím mutat (memóriában) ott az operandus címe van

⇓ párhuzamos stack-kezelés hardveres & softwares stack

- 2 komponensű címzés — csak index!!
  1. indexelt (utasítás fix (bázis) címet tartalmaz + index regiszter tartalma)
     ef. cím = b + (in) * méret // effektív cím számítása
     megadható skálafaktor 1, (2, 4, 8) ← fix van / állítható
     tömbkezeléshez előnyös
  2. bázisrelatív — az ut.-ban lévő címhez + (bázisreg) — teljes címzési tartományt lefedi
     relokációnál — új helyre való fordításnál csak a bázisreg tartalmát változtatom
     ↑ a reg. tart. állandó, mindig ehhez viszonyítok
  3. programszámláló-relatív — az ut.-ban lévő címrész + (PC)

Az a cím ami a tényleges cím lesz, több helyről vett adatokon végzett műveletek eredménye lesz. Ez az effektív cím.

- több komponensű — az előzők kombinációi

Magasszintű nyelvek támogatása

magasszintű ⟷ (ELLENTÉT) alacsonyszintű

problémaközeli, "emberi gondolkodás" közeli | gépi

(szakadék) rés — szemantikus rés áthidalása (áthidalása)

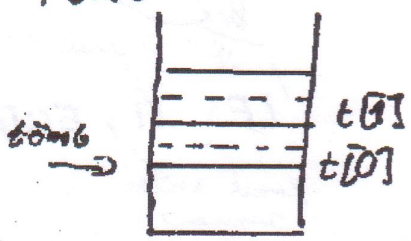
- fordító, értelmező } program — melyik a legjobb? Melyik nyelv?
  A processzorban van egy fordító, így magasszintű utasításokat is megért. De melyik nyelvet értse?
- gépi ut.-ok szintjének növelése — szintjének — egyre összetettebb utasítások
  ⇓ Ha sok utasítás van, ezeket úgysem használom ki.

MG

Követelmény Magasszintű nyelvek támogatásához 2/5

- speciális adattípusok támogatása
  (BCD, lebegőpontos, karakterlánc, stb)
  utasítással való támogatása
- összetett adatszerkezetek kezelése ; támogatás utasításokkal
  - tömbök — struktúrája — indexelt címzés
  - rekordok — indirekt —"—
  - listák, láncolt listák — spec. utasítások mutatók kezelésére

Tömb



i : integer;
tömb: array [0..max] of integer

tömb[i] := 3

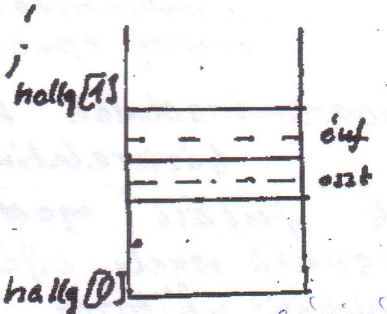
cím ← tömb + i × méret

move tömb(i, méret) ← 3

rekord

pl

hallgato = rekord
  név : array [1..10] of char;
  osz : integer i
  evf : integer;
End;



Struct hallgató
{
char név[10]
int oszt;
int evf;
}

hallgosztály

név
↑ 10 byte

hallg oszt := 5

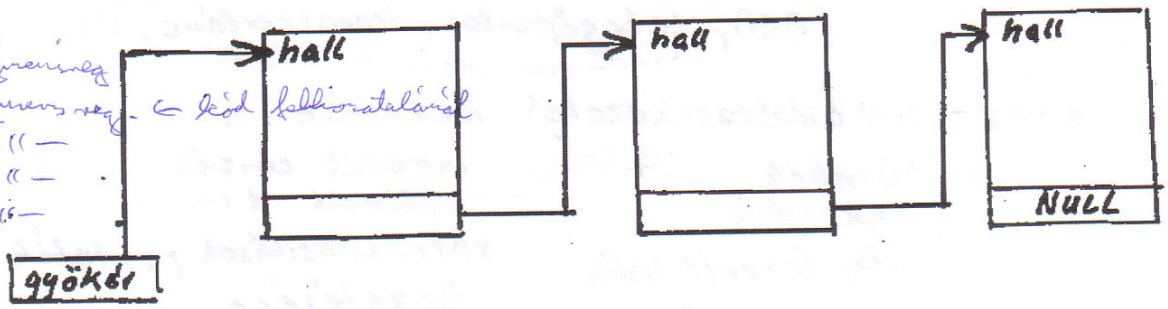
cím = hallgato + oszt offszet

move (offset, hallgató) ← 5

vagy cím = tömbkezdő + i × méret + offset

(11)

láncolt lista



spec ut pl Intell

LES ,SI gyökér

hp → ES : SI + 14 → SI

ES : SI + 16 → ES

LES SI, ES:[SI+14]

- operációs rendszer funkciók támogatása
  - rendszerhívások
  - védelmi rendszer
  - párhuzamos futtatás támogatása
- programszerkezetek támogatása
  - eljárások, függvények szubrutinhívás + verem használat esetleg spec utasítások (pl ENTER LEAVE)
  - programmodulok kezelése ⇒ moduláris programozás
  - bázisrelatív címzés ⇒ memóriablokk elkülönítése a dolgozat, eltérő programozás

Eljárások futási modellje magasszintű nyelv eljárásainak megvalósítása

- globális változók — fix tárcím; láthatóan látható
- paraméterek (paraméterátadás)
  - fix tárc. ⇒ nem lehet rek.
  - reg-ben ⇒ kevés van
  - paraméterlista ⇒ (mutatók láncolt lista átadására)
  - verembe ⇒ hatékonyság (!)
- lokális változók ~ paraméterek
- visszatérési érték — regiszter(-ek)-ben (mert kevés van) → stacken távolítjuk

verem keret (stack frame) => fix ofsetet használok

eljáráshívás:

1 paraméterek veremre írása } → eljáráshívás előtt
2. szubrutin hívás } Hívó program végzi (állapotmentés)
3. implicit paraméterek mentése a verembe → base pointer register
 – hívó keretének címe (BP) → keret regiszter: 2 részre bontja a stack-et
 (– esetleg regiszterek) pl SP
4 hívott keretének beállítása
5 helyfoglalás lokális változóknak → bemenő par. / lokális változó a stacken => arrébb állítom az SP-t helyet hagyok ki
6, eljáráson belül hivatkozás lok. változókra
7, visszatérési érték beállítása
8., lok. vált.-k helyének felszabadítása => SP állítása vissza
9. hívó keretének visszaállítása
10 visszatérés a hívóhoz } egy lépés pl PASCAL
11 paraméterek lebontása } (→ bemenő paraméterek)

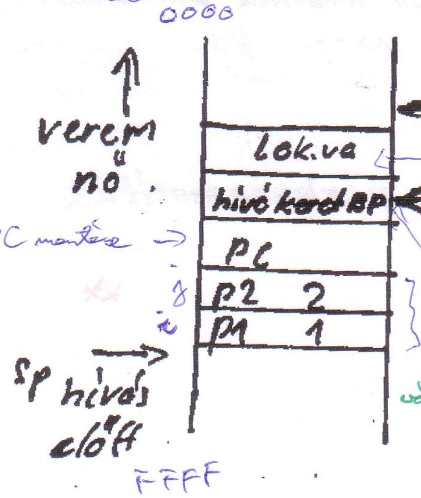
példa PASCAL

```
FUNCTION f(i,j : INTEGER) : INTEGER ;      "C"
VAR l : INTEGER ;                          int f(int i, int j)
BEGIN ... END;                             { int l ;
```

(16 bites int; // l egy int lokális változó)

Hívás f(1,2)	80186	8086	
	PUSH 1	MOV AX, 1	
		PUSH AX	→ 1
	PUSH 2	MOV AX, 2	
		PUSH AX	
	CALL f	CALL f	→ 2 ADD SP, 4
	E4:	PUSH BP	→ 3
	ENTER 2,0	MOV BP, SP	→ 4
		SUB SP, 2	→ 5
		MOV AX, WORD PTR [BP-2]	→ 6
		MOV AX, ...	→ 7
	LEAVE	MOV SP, BP	→ 8
		POP BP	→ 9
	RET 4	RET 4	→ 10, 11

(push-nál már van immediate!)



lokális változónak helyfoglalás – eljárás

hívó base pointer mentése; bemenő paraméterek

hány bytenyi lok. változónak foglaljon helyet

Most 0. a mélység ha vagyok pl. rekurziónál hányadik szinten

exception-nél az oprendszer tudja RET-ig meddig kell visszacsévélni

kilépéskor lebontást végez

4-gyel növeli meg az SP értékét

Pascal fgv.: mindig annyi paramétert foglalok neki, ahánnyal deklaráltam

C fgv.: kevesebb paraméterrel is hívhatom mint ahánnyal deklaráltam – csak a hibakezelést garantálja

(... PC alatt ...) => fordított sorrendben ... a

Szg Teljesítmény növelése

Az utasítás végrehajtás fázisai

- Utasítás felhozatal — olvasás → gépi ciklusok (fázisok)
- értelmezés
- paraméter felhozatal — olv. → Külső MEM, I/O
- belső: művelet
- eredmény kivitele — írás

fázisidő-csök. ⇑ frekvencia növ.

Sebesség fokozása
- → ut. végrehajtás gyorsítása (1. mód)
- → ut. szám csökkentése: komplexebb utasítások
- → több utasítás párhuzamos végrehajt. ⇓ algoritmusidő csökkentése

Utasítás végrehajtás gyorsítása

- fázisok rövidítése → frekv. nő (határ?!)
- szükséges gépi ciklusok számának csökkentése
  - → regiszterek alkalmazása → nem kell felhozni a paramétereket
  - → adatszélesség növelése 4/8/16/32 bit (64) ⇓ nem kell több darabban felolvasni az adatot

  ezzel nő az utasítások száma, mivel nő az adattípusok/hosszak száma
- gépi ciklushossz rövidítése
  - memóriához fordulás idejének csökkentése → adatokat
  - hierarchikus mem. felépítés → CACHE
  - kapacitásnöv. → háttértárak & virtuális tár

  ⇓ memóriahozzáférés is más: mennyi adatot ér el egyszerre?
- műveletvégzés gyorsítása
  pl spec. ALU (pl kombinációs hálózat) — huzalozott logika ⇓ gyorsabb, de nem mikroprogramozott
- memóriához fordulás és belső munka párhuzamosítása pl ut. Queue — proci belső egységeinek optimális kihasználása

  felolvasott parancsokat FIFO-ban tárolja: utasítás Queue

  mikorra a köv. ut. végrehajtására ért, az már a regiszterben volt a. akkor DE feleslegesen is olvasott fel adatot
- Utasítás felhozatal, értelmezés, végrehajtás párhuzamosítása — párhuzamosan végrehajthatók

  ⇓

  PIPE LINE

C fgv.: ① fordított sorrendben mennek le a bemenő paraméterek

↑ 13. old.

② CSAK RET van, ui. nem tudom meddig cserélek vissza

— ∘ —

neve: keret: stack frame

Pascal: f(i, j, k)

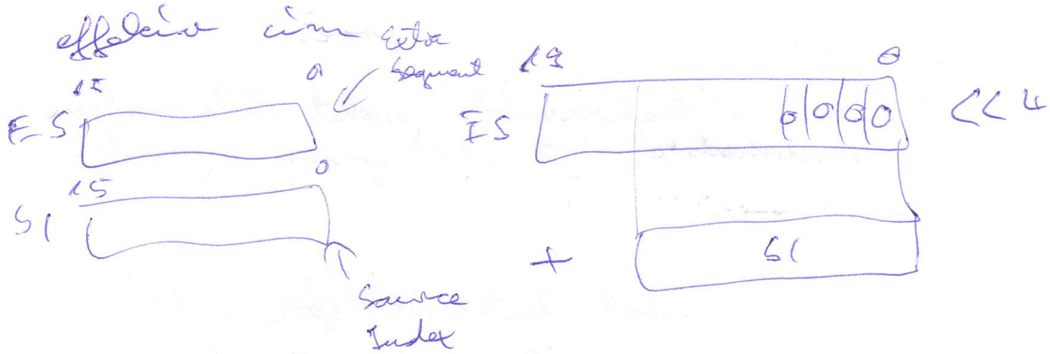
function f(i, j, k : integer)
  var m, n : integer
  begin ... keret bázis → BP
  end;

SP → v. hajtás után
3
1
5
IP
BP
m
n
SP → v. hajtás alatt

} memória 20 bites címek

— ∘ —

effektív cím



— ∘ —

X86-nál privilegizált utasítások, üzemmódok vannak

— ∘ —

Németh Gábor – Informatika 1.
355 021

RISC: pipeline

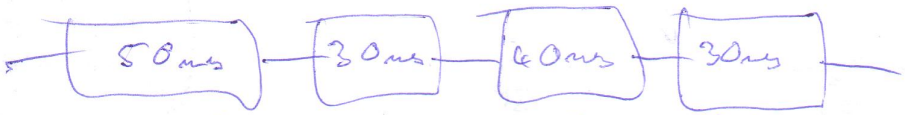
lappangási idő: 1 utasítás végrehajtásának ideje nem csökken

DE

több utasítás végrehajtása folyik párhuzamosan

— o —

pipe



$t = 150$ ns pipeline nélkül

pipeline-nal: 200 ns $= 4 \cdot 50$ ns : leglassabbikon időzítünk

DE hosszabb időre nézve a throughput $\frac{1}{50}$ ns $\Rightarrow$ olyan, mintha 4x gyorsabb lenne a proci

— o —

pipeline-ban: csak a leglassabb puffert ütemében léptethetők, DE átlagosan gyorsabb lesz a proci

- { ez csak kutelnél gáz CISC:
  RISC procikban $\forall$ utasítás egyforma "hosszú": azonos időig tart.
  $\Rightarrow$ OTT optimális a pipe sebessége

6

Utasítások számának csökkentése

Utasítások komplexitásának fokozása
(funkciók, címzési módok, adattípusok) ⇒ CISC
⇒ sebesség növelésével ellentmondó követelmény

RISC

$P = I \cdot C \cdot T$

I = ut száma
C = órajelciki. átlagos száma
T = órajelciklus hossza

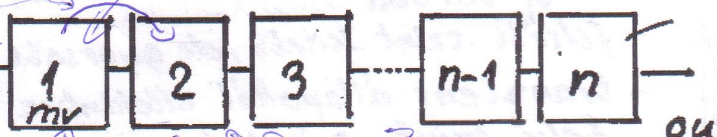
az egyszerűbb utasítások száma nagyobb ⇒ I = nő DE C & T csökken ⇒ P = nő } RISC

RISC elvek:

- egyetlen ciklus alatt végrehajtható, egyforma hosszú ut-ok
- tárolóhoz csak írási-olvasási utasítás, az összes többi regiszteren dolgozik → Reg tömb nagy
- huzalozott vezérlő egység → gyors
- software végzi a bonyolult funkciókat kód optimalizálást elősegítő fordító kell

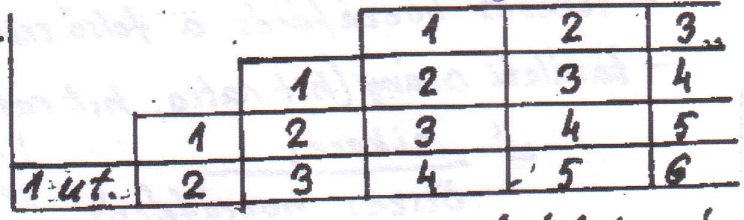
A bárium RISC szj megvalósításához kevesebb áramkör kell → gyors

PIPE LINE alkalmazása:



n. művelet végző

elv: az utasítást szekvenciálisan végrehajtandó műveletekre bonthatjuk.



feldolgozása

pl 1 ut lehívó
2 ut dek
3 ut végrehajt.

hosszabb bárra n egyszerű n-ed részére

csökkenti a feldolgozási időt

problémák! sorrend, utasítás egymásra hatás

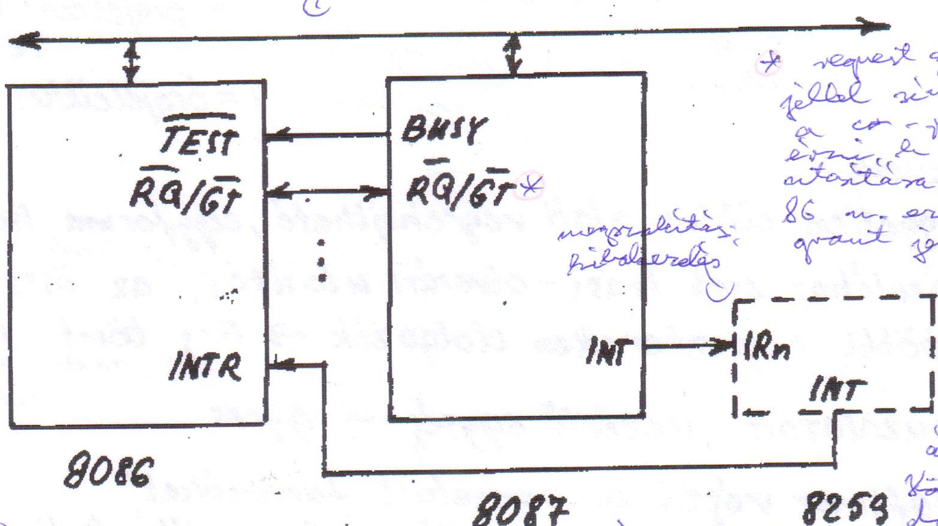
(147)

processzor tehermentesítése

– funkcionális elemmel

– társprocesszorokkal (ez optimalizálható a plusz feladatra)

proc és co-proc -ra szinte minden jel párhuzamosan van kötve



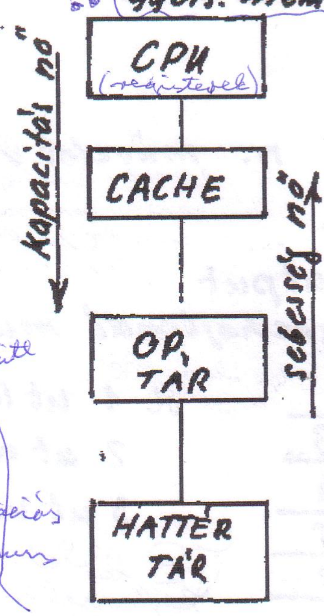
* request grant jellel vezérlést kér a co-proc, ha érzi, h az ő utasítása van a sínen, 86 ezen a sínen grant jellel válaszol. Ilyenkor a co-proc. dolgozik, busy jelét foglaltra állítja. Közben a 86 dolgozhat. Ha a 87 által végzett művelet eredményére szükség lesz, a 86 oda beteszi egy wait utasítást. Itt max-szal addig vár üresben, míg a 87 kész nincs az eredménnyel.

286 ⇓ már 24 bites címkezelés és virtuális tárkezelés

486-ban már benne volt a co-proc

Memória hierarchikus szervezése

!! Nagy, olcsó mem lassú; gyors mem kis kap. drága



lokalitási elv

a, időbeli lokalitás → ha használok egy infót valószínű h a későbbiekben is fogom

b, térbeli lokalitás – h a közvetlen közelében lévő utasításokat is fogom

– felsőbb szint kisebb, de gyorsabb

– tranziens állapottól eltekintve a felső szint részhalmaza az alsó-nak

– kommunikáció a szintek között blokkos

– sikeres hozzáférés a felső szinthez HIT

– találati arány (hit ratio, hit rate)

$$= \frac{\text{sikeres}}{\text{összes hozzáférés}}$$

// mennyi a sikeres hozzáférés aránya %-osan

a teljes tárterületet fogom fel, mint virtuális címtartományt, mert az operatív tár mindenhol más méretű ⇓ címtranszformációs mechanizmus ⇓ **kapacitás növelés**

Ha egy infóra szükségem van, akkor a nagyobb és lassabb tárból áttöltöm azt és környezetét, mint meghat. méretű blokkot a kisebb és gyorsabb tárba: CACHE-be. Később innen használom. ⇒ **sebességnövelés**

áttöltés: fix méretű blokkokban

MC

CACHE
(gyorsítótár)

Hová tesszük be a blokkokat (melyik adatot, a környezetnek mely részét?)

szervezési módok (cache memória leképezés)
- Közvetlen leképzés (direct mapping)
- Asszociatív leképzés (associative mapping)
- Részben asszociatív leképzés (Set associative map.)

elv
- memória blokknak cache blokkot feleltetünk meg
- memória címét
- mem.-ben egy adatelem (byte, blokk, stb) címét a mem.-ben elfoglalt helye egyértelműen meghatározza

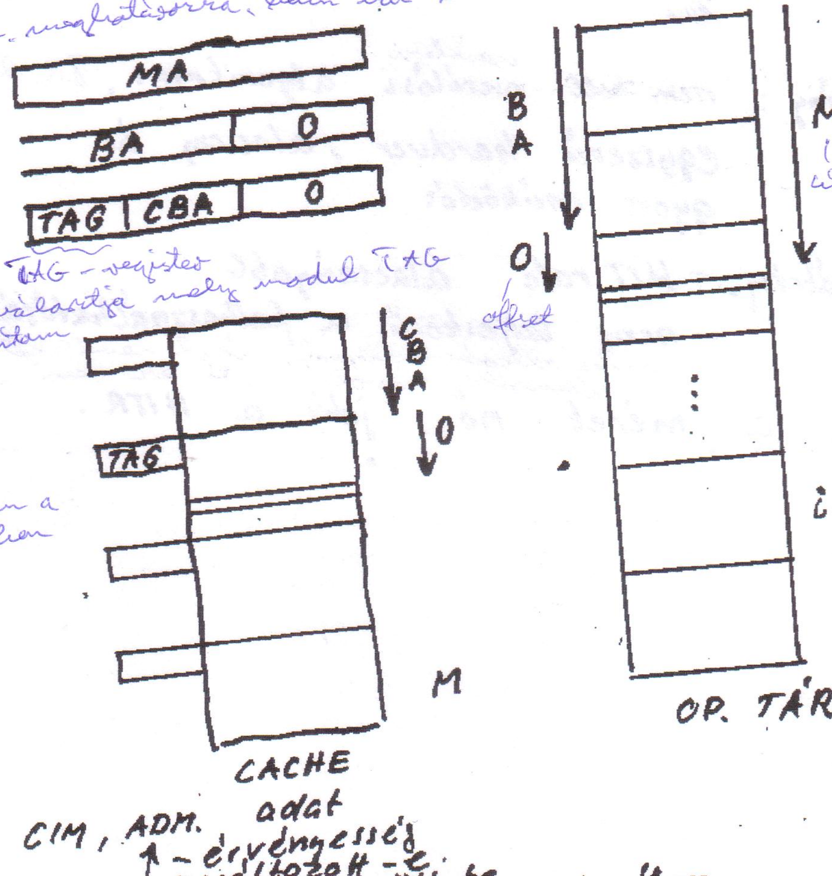
Cache-ben adat mellett még cím és egyéb információ

Ha 4 blokkos a cache, osszuk az op. tárat is 4 blokkos részekre ⇒ megfeleltetés

① Direkt leképzés

a cím egyért. meghatározza, hogy van-e az adat a cache-ben

adat memóriában való elhelyezkedése egyértelműen meghatározza hová kerül a címlet cache-be



TAG – regiszter
CBA kiválasztja mely modul TAG regiszterét hasonlítom össze a cím TAG-részével
ha „=" ⇓ benn van a cache-ben

offset

i

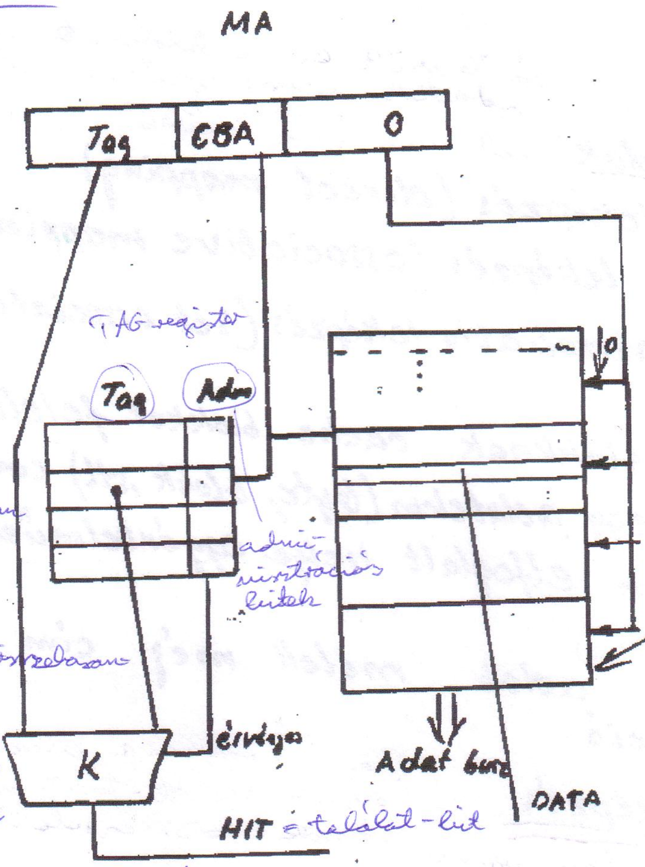
Cache M blokk

$\frac{i}{M}$ → maradék ↓ hely

adat
CÍM, ADM.
↑
- érvényesség
- változott-e
- mikor került be
- mikor jött be a mortandban
- használták-e

M6 Chache: 3/5

Egyszerre indul a chache-hez és az op.tárhoz való hozzáférés. Ha chache-ben találat van abortálja az elkezdett még kezdeti fázisban levő op.tár hozzáférést



OP. TAR-nál nagyságrenddel gyorsabb memória

TAG memória
írás: beíródik az adatmezőben levő adatot
olvasás: szintén bejön az adat a sínen komparátorban összehasonlítás

komparátor
csak 1 helyen kell megnézni a chache mnódját, direkt leképezés miatt csak ott lehet az adat

előny: nem kell cserélési algoritmus, fix helyek vannak már előre; nincs keresés
egyszerű hardver, alacsony ár
gyors működés

hátrány: - HIT rate alacsonyabb
- nem teljeskörű a felhasználhatóság (más alkalmazásnál, ahol nem utasítást & adatot akarok tárolni, ott ez nem előnyös)

ha a méret nő jobb a HITR.

M6

② Asszociatív leképzés : nincs szabály arra, h melyik blokkokat hová tegyük

— egy adott blokk a Cache bármely helyére beírható → nincs CBA ⇒ komplikáltabb

Lényeg minden TAG-ot egyszerre komparáljuk

offset után minden bit a TAG része

③ Részben asszociatív (set associative)

egy komparátor már nem elég, annyi kellene az optimálishoz, ahány blokk van

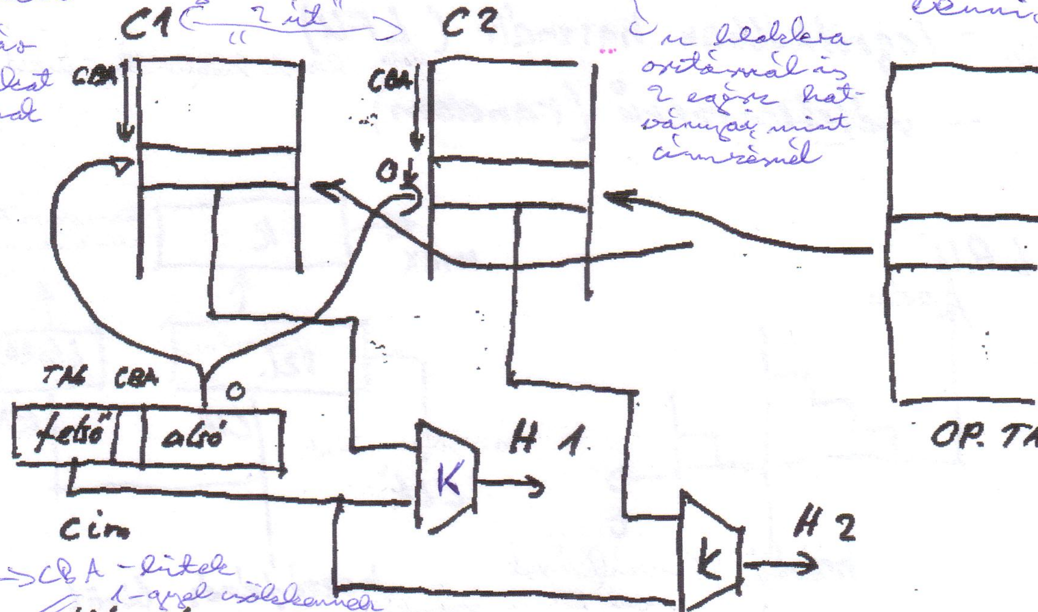
— egy blokk több helyre (n utas direkt leképzés) n db komparátor — cache

amit nem szokott lenni, nagy és drága

Kevert módszer, mindkettő előnye benne van

Ma már 4 utasokat használnak

n blokkra osztásnál is 2 egész hatványa mint címzésnél



2 út → CBA-bitek 1-gyel csökkennek

1-gyel több bit a TAG memóriában

Kérdések

1, Mikor hozunk be egy blokkot (Fetch policy)?

- — közvetlen igény esetén (MISS-nél) ritkán hivatkozunk rá és nincs találat
- — előrelátással i-nél i+1 is — i-edik blokk behozatalakor az i+1-ediket is behozza
- — szelektíven pl Data, Instruction

stratégiák

2, Hogyan írunk a Cachbe - illetve Op mem-be.*

— írás C-be (cache) és Op. membe (write-through)

- • write-through - with write - allocate ← írás MISS-nél betöltés
- • write -||- -||- no write allocate

— írás csak a cache-ben (write back) — D bit kell! ← háttértárnál ez a gyakori!

Név szerint választás, h mit hozzuk be

pl.: 1 csak utasítás cache

2 különös intervallumokat letiltok a cache-elhető tartományból

tranziens ifónál kisebb halmaz részhalmaza a nagyobbnak

* Update policy

— Adat konzisztencia!

Motorolák és i486 tudott előtte

— nem volt I/O utasítása

memcím szerinti szelektív ned két master / két proci esetén nem mindegy hol van a cache (pl.: DMA)

Ha nincs bent behozza és mindkettőt átírja

csak kint írja át

• write back : csak a cache-ben írom át,

háttértárnál mindig write back ha bent kint csak később → ettől gond a konzisztenciánál

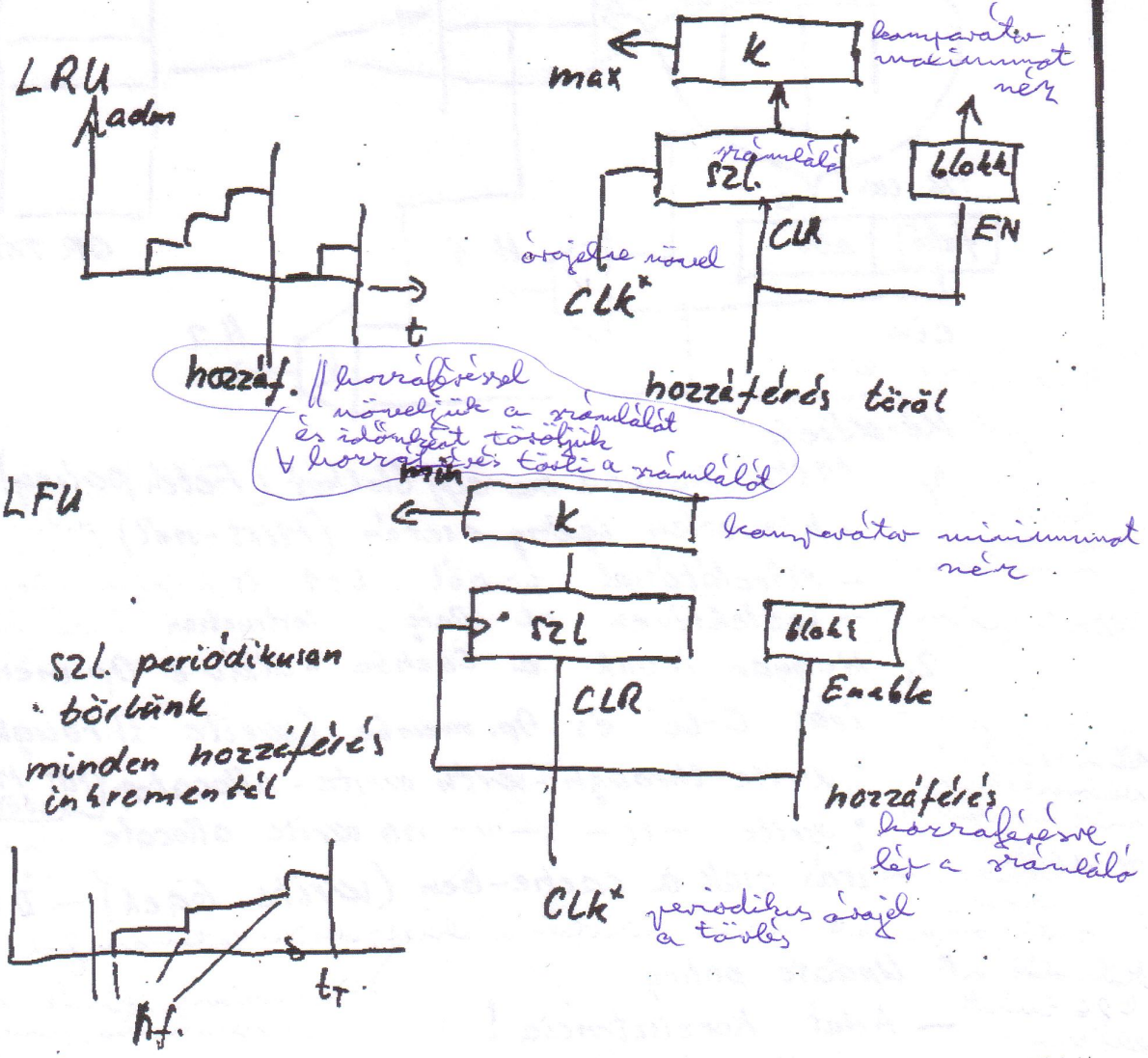
3. Új blokk betöltésekor melyik blokk helyére viszünk be (block replacement policy)

- direktnél ez nem kérdés
- ha van üres (hely) oda visszük

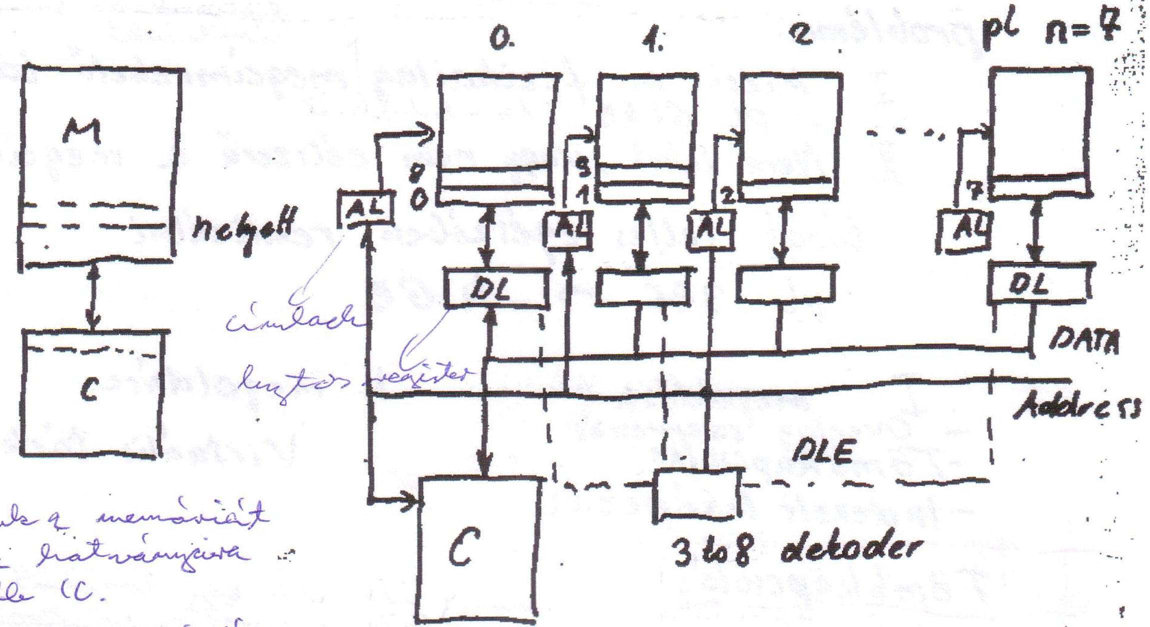
Ha nincs üres hely

- legrégebben használt (least recently used LRU)
- legrégebben betöltött (FIFO)
- legritkábban használt (LFU) (least frequently used)
- véletlenszerű (random)

stratégiák



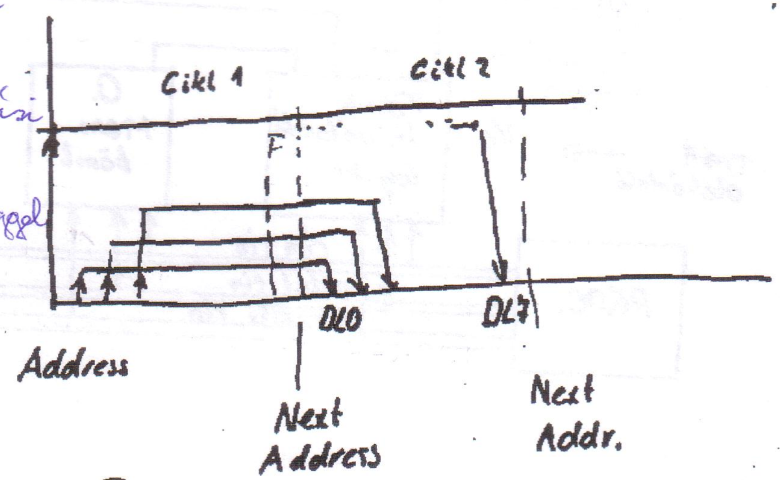
Probléma: lassú a memória hozzáférés



- felosztjuk a memóriát 2 egész hatványaira => 8 db (C.
- cache-ben értéket kiírom az adatregbe, kiírom a címet a latch-be, letelt az operatív tár várakozási ideje, beírom i egy gyorsítóba

  cache 8x-os sebességgel működik
- így tűnik a 8 chip átlapolása miatt, mintha 1/8 x-os lenne a hozzáférési idő

  1/8-a



$$T_h \cong \frac{T_c}{n}$$

Memória interleaving (beékelés, mem. átlapolás) (memória átlapolás)

- Burst adatátvitel (csomagolt ... átvitel)
  először sor & oszlop címet is adok, utána az oszlopcímet automatikusan inkrementálja a vezérlőjel ütemére

Tároló kezelési módszerek

0 - 640k ram M/I lapokat → driver és tömbkiválasztó

felette más nem driverekneк fenntartott címterület

probléma:

I Kicsi a fizikailag megcímezhető tárkapac.
pl 64kB // 16-bitesnél

II Nem lehet, vagy nem célszerű a megcímezhető tárat teljes egészében realizálni
pl 386 → 4 GB

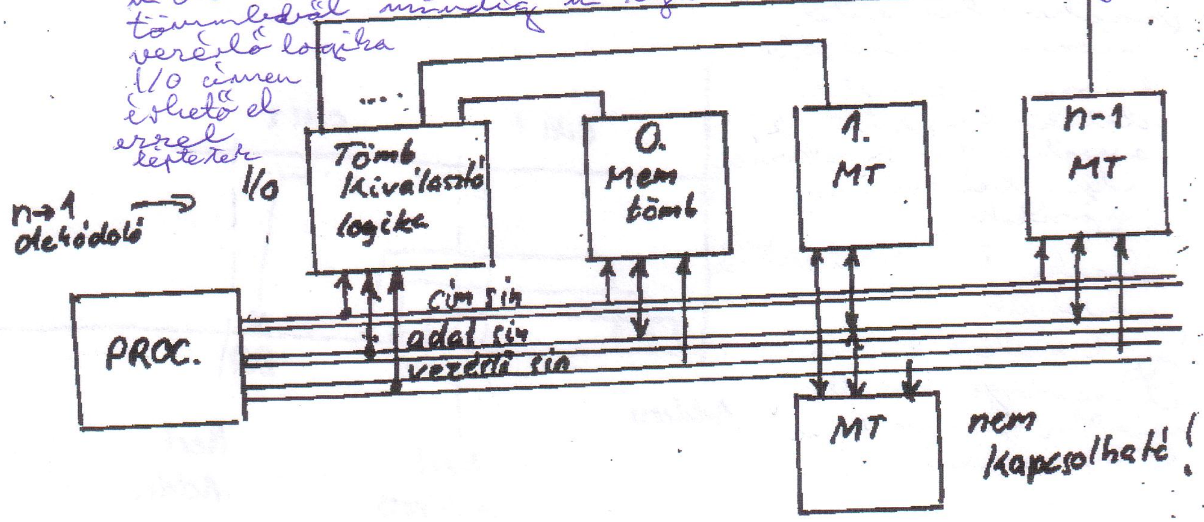
I. megoldása
- Overlay szervezés
- Tömbkapcsolás
- Indexelt leképezés

II megoldása
Virtuális tárkezelés

I. Tömbkapcsolás

n db tömbből (mem.) illesztek egy logikát, h a tömmbből mindig n byte memória látszódjon

vezérlő logika I/O címen érhető el ezzel lépteted



előny: nagyon egyszerű, gyors
• tömb átkapcsolás OUT ut.-al

hátrány: durva és merev memóriafelosztás
közösen használt rész mindig aktív
átkapcsolást végző programnak & megszakítási rutinoknak mindig látható mem. területen kell lenni

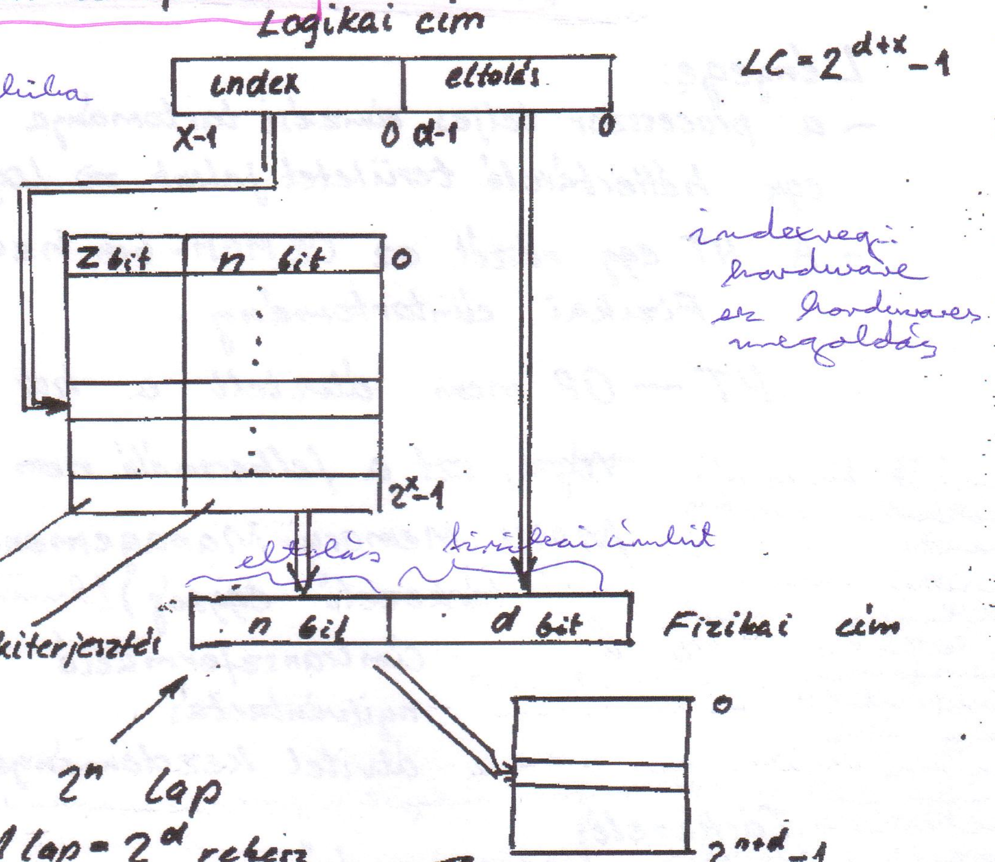
MG

4/2

II. Indexelt leképezés

$LC = 2^{d+x} - 1$

pagefault = laphiba



indexregiszteres hardware az hardwares megoldás

2^n lap

1 lap = 2^d rekesz

ebből egyszerre $2^{d+x} - 1$ rekeszt

ha n > x akkor nagyobb címterülete is lesz tudás címezni, mint amit a proci egyébként tud

előny: egyszerű

rugalmas

Log → Fiz átalakítás gyors (mert regiszteres hardwares I/O címen írható)

programok a tárolóban szabadon mozgathatók (mert logikailag bárhová tehetem)

hátrány: – viszonylag költséges

– a közösen használt részeknek (pl I/O, IT-k) mindig aktívnak kell lenniük

– nem: adott méretű részekre van osztva

– NEM tömbleképezés logika van; az indexelt címtranszformáció a logikai cím dd bitje mondja meg melyik lap kell (x) (dd az eltolás) közvetlen fizikai címe nagy.

a cím felső része kiválaszt a 2^x regiszter közül 1 db indexreget. ezek I/O címen érhetők el.

16 címreg ⟹ 16 × 4 kbyte

Az aktív területet címreg átírásával jelölöm ki; többi bitje a regnek: adminisztráció pl.: user/system; írható/read only terület stb.

A címterületen nem öf. 4k-s blokkok logikailag összefüggőnek tűnnek.

VIRTUÁLIS TÁRKEZELÉS

Lényege:

- a processzor teljes címzési tartománya tulajdonképpen egy háttértároló területet jelent ⇒ Logikai cím
- A HT egy részét az OP.MEM-ben helyezzük el
  Fizikai címtartomány

HT — OP mem átvitelt a hw és sw (operációs rendszer) végzi, ezt a felhasználó nem látja

MMU Memory Management Unit (tárkezelő egység) (hardware)

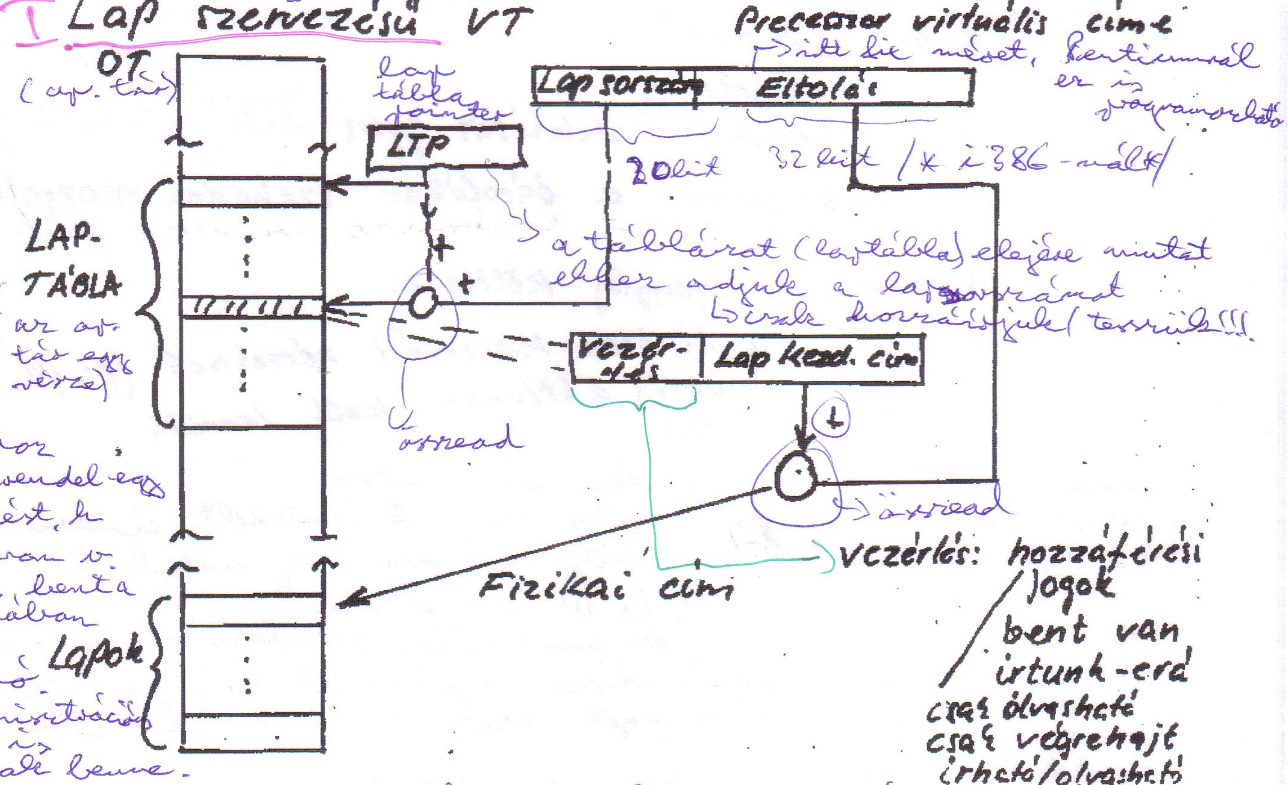
- címtranszformáció
- nyilvántartás
- átvitel kezdeményezése

- mem.-at fix méretű blokkokra, lapokra bontom mind a fizikai mind a virtuális címteret; a kettő között egy transzformációs táblázat teremt kapcsolatot; ez az op. mem. egy része
- ha egy utasítás egy operandusa nincs bent az op. tárban, megáll a proci, az oprendszer betölti azt a háttértárról, kitölti a transzformációs táblát, folytatja v. újrakezdi az utasítás végrehajtását

Tárkezelés

- Lapszervezésű
- szegmens szervezésű

I Lap szervezésű VT



- lapokhoz hozzárendel egy bejegyzést, h bent van v. nincs bent a memóriában. Ez a laptábla. Adminisztrációs bitek is vannak benne.

- 1 byte kiolvasásához 2× kell memóriához fordulnom
  1: laptábla olvasás
  2: kiolvasom / írom a byteot

Megoldandó problémák

- sebesség: ne kelljen mindig többször az OP-tárhoz fordulni → TLB (translation look aside buffer) ↑ ne kelljen 2x olvasni
- Több lépcsős Laptábla → Laptábla könyvtár → Laptábla
- Behozatali / Cserélési / Írási algoritmusok (ez az egyedüli dolga) — egy lépcsős: nagy lesz a laptáblám a sok lap miatt; többlépcsősnél a laptábla részeit is úgy hozom be mint a lapot magát
- Mi van Lap nincs bent jelzésnél

II. Szegmens szervezés

- a logikai objektumokhoz (pl program, adat, stb) változó méretű memória részeket (szegmenseket) használ
- Címfordítás hasonló mint lapozásnál, de egy szegmens összefüggő (folytonos) fizikai tárterületet igényel
  a szegmens hosszát is meg kell adni

előny: - jobban illeszkedik a „programozói szemlélethez"
- hatásosabb védelem

hátrány: - az eltérő méretek miatt un. memória fragmentation (szétesés) jöhet létre
- bonyolultabb algoritmusok kellenek

- lap: fix méret
- szegmens: nem fix a méret, dinamikus → adat / kód / stack } objektum
  nem fix területekre, hanem logikai objektumokra bontjuk az algoritmust és ehhez rendelünk majd tárterületet
- meg kell adni a leírásban hol kezdődik a szegmens
- vigyázni kell, h ne címezzem túl a szegmenst, hosszát is meg kell adni ⇒ védelem /* lapot nem lehetett túlcímezni */
- dinamikus méretű összefüggő területek vannak
  van-e akkora összefüggő területem ahh amekkora kell ???
  op.rendszer szereli !!! /* lapnál ez nem volt gond */
  Részek vannak, apró üres részek keletkeznek a memóriában amik használhatatlanok : mem. szétesés / fragmentáció

- 386: lapszervezés
  szegmentált lapszervezés

Holtidőben, mikor a CPU nem dolgozik, futtasson más feladatot → inkább software megoldás

Multitasking

ehhez ütemezés kell

pl.: time scheduling időosztásos ütemezés

A modern szg-ek számos részegységet-erőforrást tartalmaznak. Egy felhasználó által írt program egy időszakban csak ezek egy részét használja ← CPU kihasználtsága növelhető jobb programokkal

→ Op. rendszer gondoskodjon, hogy olyan másik feladatot is futtasson a szg., amely egy éppen szabad erőforrást igényel → a rendszer átbocsátóképessége nőhet

de → Op rendszer is igényel erőforrást → váltás: mentés új betöltése

– **Task** → algoritmust & adatállományt tartalmaznak

tevékenység, amely a szg.-en más tevékenységgel elvben párhuzamosan folyhat

A task utasításokból és kiinduló adatokból álló programot futtat

tipikus task-ok: file szerkesztés, forrás file fordítás, stb..

ugyanazt a programot több task is futtathatja

pl 10 ms-ig A, majd B, stb forrását fordítja

– **Virtuális processzor** /* op.rendszer „állítja elő" */

minden task-nak (felhasználónak) úgy kell érezni, mintha saját külön processzora lenne

elméletileg egy task teljesen párhuzamosan fut a többi task-al (kivétel: a másik adatára vár)

Valóságban: ugyanazon a fizikai processzoron osztoznak

Az Op. rendszer mindig hozzárendeli a fiz. proc-ot (pl időosztásos /time-sharing/ módszerrel)

→ minden taskhoz adatszerkezet (pl proc. regiszterek + Op. rendszer specifikus)

⇓

betöltés
mentés

Kontextus váltás /kontext switching/ Opr. hívás (leösszegezi a változást)

⇓ hw → hw regiszterei
sw → Opr. táblák, file leírók, stb

– a proci teljes állapotát mentenie kell
– másik taskhoz tartozó állapot betöltése

⇓

hardware támogatás kell a gyorsításához a többit a rendszer valósítja

+ taskok memóriaterei ne fedjék egymást

Ha mind a betöltésben megy, a felhasználók nem tudnak egymással.

M6

– Védelem

– felhasználói task-on belül fellépő bizonyos hibák ellen
– egyik task megvédése a másik hibáitól (pl.: szegfűszál)

→ észlelni kell a védelem megsértését
– hiba hatását a task belsejére korlátozni

Egyszerű megoldás:

tároló terület védelme
rögzített vagy változó méretű részekre bontják a tárolót. Ezekhez védelmi attribútumokat rendelnek
pl csak olvasható
írható/olvasható

tartomány ellenőrzés
⇓ prioritás hozzárendelés

A fentiekkel tárolóterületet védünk, pedig a feladat algoritmusa és adatai – logikai egységei – védendők

→ Megoldás: szegmentálás

tároló területek : csak kód
stack
adat
extra

(címzés, állam védelem)

relatív címzés (szegmens reg + offset)
szegmens hossza ismert → ellenőrzés (+védelem)

fentiek csökkentik a rugalmasságot

⇓ megoldás:
– szegmenseket a tárolóban tetszőleges helyre helyezhetjük (ez úgy is kell az eltérő konfigurációk miatt
– speciális utasítás (pl prefix) -al eltérhetünk a hozzárendeléstől (vigyázat hiba esetén baj van)

Védelem – magasabb – logikai szinttel → lsd. 38

DE itt is vannak közös használatú területek pl.: egy adatbázist több task is elérjen
ha.: pl.: arany os területre definiáljuk a regisztereket

M6 — az első 32 bites ? az x86 családban

80386

32 bites μP 4Gbyte fizikai 64Terabyte virtuális címtartomány, beépített lapszervezésű MMU, 3-4millió ut/sec felülről kompatibilis 8086, 80286 μp-al

fizikai címtartomány is 32 bites ; logikai címtér 4 Gbyte
fizikai —"— 4 Gbyte
virtuális —"— 64 Tbyte

386 címzési rendszere

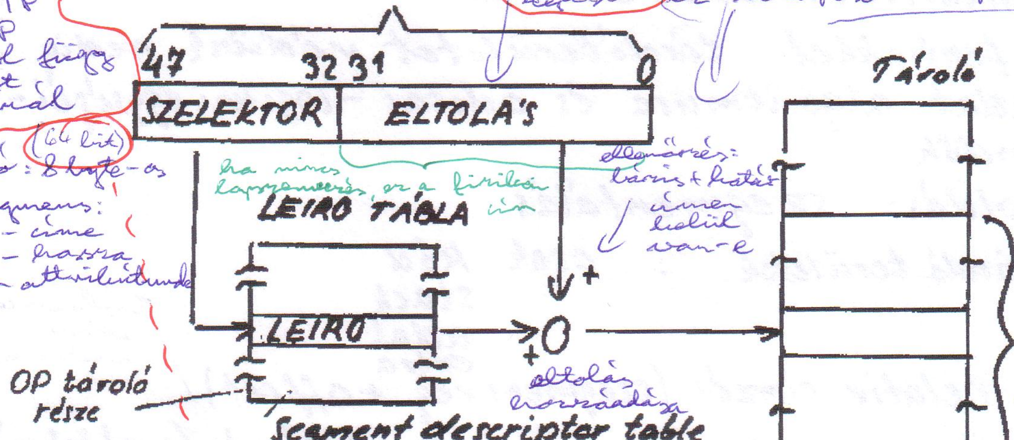
- **Lineáris címtartomány** — nincs logikai objektum ellenőrzés!!
  4Gb (Gbyte) egybefüggő, strukturálatlan tárolóterület

2 üzemmód → real → induláskor: gyors 8086 - az proci először
→ protected → a szegmens szervezést csak itt tudja; itt szűnik megvalósul

- **Szegmensek** — szegmensméret
  logikai egységek 1 byte — 4Gb (4 Gb) közötti méret (EIP-t) (a programstruktúrának megfelelően)
  Logikai cím — instruction pointerben kapják — az leikapcsolt lapszervezésnél

az eltolás műveletnél, objektum típus függő
EIP
SP
attól függ mit csinál

leíró: 8 byte-os (64 bit)
szegmens:
- címe
- hossza
- attribútumok



286: 16 bites
DE szegmens szervezés már van
átmeneti lépés 8086 & 80386 között

80386: 6 szegmensregiszter
8086: 4 szegmensreg.
16 bitesek
a többi reg 32 bites

Szegmens szelektor/segment selector 16bit

Szegmensen belüli eltolás/OFFSET/32 bit

A szegmensleíró tartalmát az op. rendszer kezeli

8 byte : 1 Op. tárbeli kezdőcím
2 Sz. hossz
3 Attribútum-ok (védelem)

16 bites szegmensreg

64 bites "árnyékreg." automatikusan betöltődik a leíróval együtt

real mode-ban ∀ reg. 16 bites módban

→ lassú működés elkerülésére belső szegm.regiszterek

15 Szegmens reg. ↑ RPL 0	63 (64 bit) Leíró 0		
Szelektor	Kezd.cím Hossz attr.	CS	kód
	ALELSŐ IDE KERÜL	SS	stack
Felhasználói	pr-ból nem hozzáférhető!	DS	data
		ES	extra
Ha nincs bent	a Proc hw úton kiolvassa	FS	"
a leírót és a 2. mem ciklusban hozzáfér		GS	

utasítással nem "érhető el"
↓
a leírót nem a memóriában lévő indextáblából veszi, hanem innen

módváltásnál fel kell építeni az indextáblát leírótáblákat stb.

CS-ben van az aktuális privilégiumszint + a leírónak is van egy privilégium szintje: DPL
descriptor privilégium

Az operandusnak is van privilégium szintje, amiket megvizsgálhat...

Az árnyékregiszterben az indextábla egy része van MÉGHOZZÁ AZ AKTUÁLIS LEÍRÓ
h. ne kelljen a memóriában lévő indextáblához nyúlni

globális szegmensleíró tábla — mindenkinek látszik

lokális — " —

— ∀ taszkhoz rendelhető külön tábla, ezt csak az adott taszkból látszik

Az árnyékregisztert külön nem tudom utasítással elérni!
A leíró kimásolása automatikus hw. mechanizmus

ELTOLÁS: – utasítás felhozásnál: EIP
– stack műveletnél: SP
– más utasításokhoz defaultból rendelve, h a DATA v. EXTRA szegmens legyen az eltolás, máshol én adhatom meg mi legyen

— o —

GDTR } helye, h honnan kezdődik: egy spec
LDTR } pointer mutatja

— o —

4 Gbyte max szegmensméret + 16 bit szegmensreg $2^{14} \cdot 4$ Gbyte ⇒ ⇒ 64 Tbyte
↓
virtuális címtér

— o —

∀ ~~árnyékregiszter~~ szegmensregiszterhez egy db saját árnyékreg. tartozik
Ameddig nem írjuk át a leírót, addig az árnyékregiszterből használja

— o —

Flat Model

Ha ∀ leíróban u. az a cím van, méghozzá 000...H kezdőcím ⇒ védett módban így láthatok 4 Gbyte összefüggő címterületet KIVÉVE stack-nél, mert az fordítva működik
SS reg tartalma ↓
itt a kezdőcím a stack alját jelenti, és „felfelé" nő.

Protected Flat Model

(1) ∀ code segment-et más külön kezeli

(2) ∀ ~~diszkussz~~ szegmensleíró más, külön leíró van, (mutathat 2 db 1 helyre is) ; ez a kiépített teljes védelem.

---

intel (x86) laphibánál mindig újrakezdi az utasítás végrehajtását; Motorola tudja folytatni.

∀ taszknak lehet (**) saját laptáblakönyvtára

/* Figure 3-28. */

directory → 1024 table (1024 laptábla) → 1024 lap (1024 lap)

1024 × 1024 lap

∀ tábla & ∀ könyvtár 4 kbyte
Ezeket pontosan laphatárra rakják be a memóriába

(1) "csak" lapok vannak, szegmensek nincsenek (2) a szegmenseket darabolja fix méretű lapokra.

M6 – **Lapok**

akár lineáris
akár logikai strukturált } cím esetleg kisebb fiz. tároló méret

Virtuális tárkezelés: lapszervezés /kikapcsolható!!/

MMU 4 K-s lapok kétszintű lapkezelés

Előzőek alapján 386 címképzési lehetőségei (címtér-modellek)

– strukturálatlan (lineáris) címzés nincs virtuális tároló (→ lin. összefüggő; 0 kezd, 4Gb hosszú)

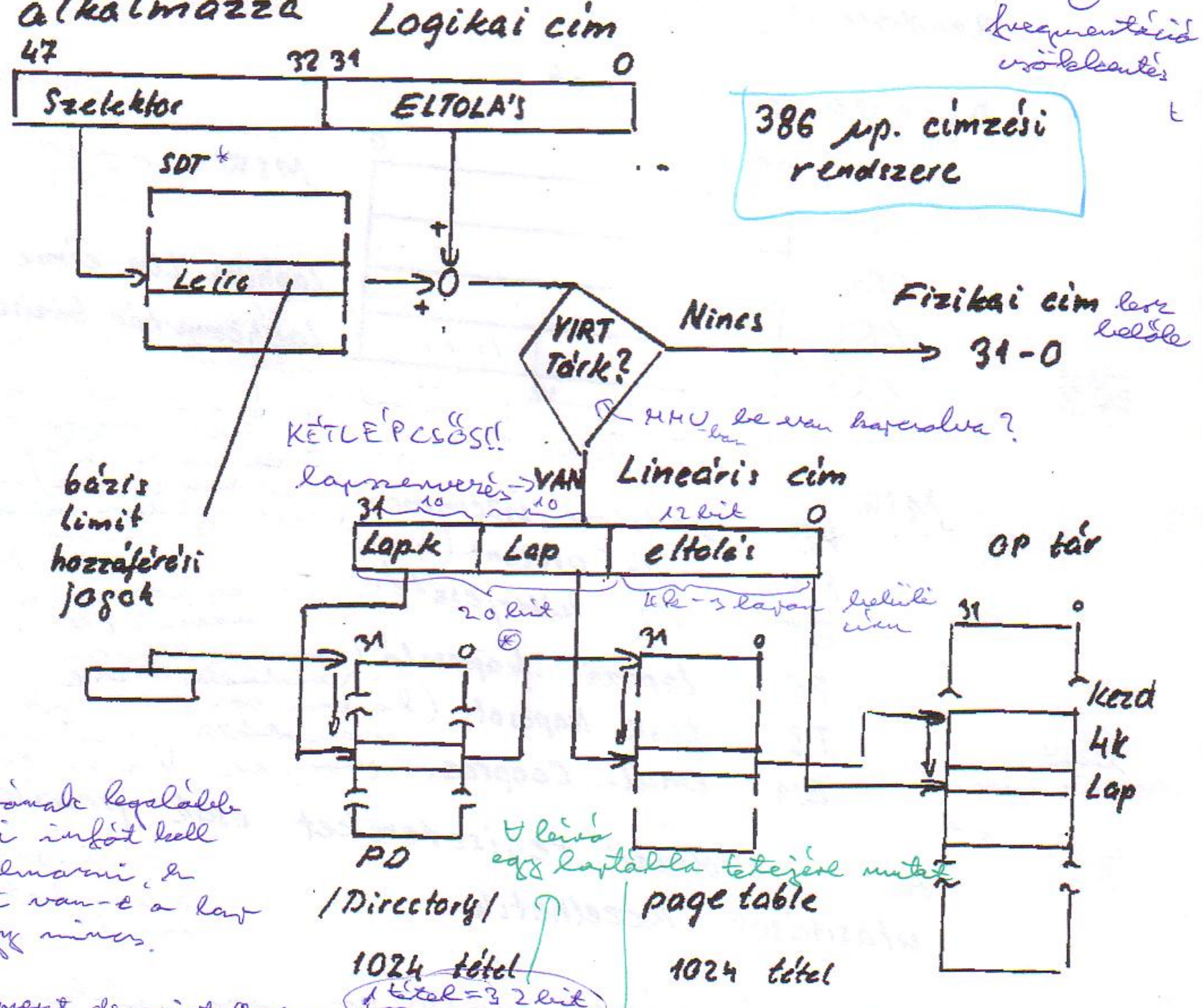
– szegmentált címzés – " –

– külön bekapcsolhatom mindkettőhöz a lapszervezést, mint virtuális tárkezelést

(1) – strukturálatlan címzés virtuális tárkezelés

(2) – **szegmentált** címzés virtuális tárkezelés (→ szegmentált lapszervezés: legmodernebb)

Az Op. rendszer határozza meg hogy melyiket alkalmazza (szegmentációs választás)



– 1 leírónak legalább annyi infót kell tartalmazni, h bent van-e a lap vagy nincs.

* segment descriptor table

⊛ felső 10 bit → laptábla könyvtárból választ leírót, ez mindig bent van

(**) alsó 10 bit → a kiválasztott laptáblából kiválasztja a tényleges leírót

46

Rendszer regiszterek

(Az Op. rendszer használja) 8 byte-os leírókból csak 16 bitet választ

rendszer indulásakor 1-szer íródik be

táblázat címe, hossza

Rendszer cím reg-ek	47 Bázis (32 bit)	15 Limit (16 bit) 0
GDTR	Bázis	Limit
IDTR	Bázis	Limit

GDTR /Global descriptor table/ globális leíró tábla — a globális leírótábla címe és hossza

IDTR /Interrupt desc. table/ megszakítás leíró tábla — ez is globális leírótábla, de spec.

Ezek csak a legmagasabb privilégiumszinten írhatók ⇒ csak az oprendszer írhatja

task register: ez egy 16 bites selector

	15	0	63	0
TR	selector		Báz. Lim. attrib	
LDTR			LDTR által mutatott leíró	

egy TSS-re mutat

környezeg lokális leíró tábla

IDE TÖLTŐDIK A SELECTOR ÁLTAL MUTATOTT LEÍRÓ

futó task-hoz /állapot szegmens/ Lokális leíró tábla / vagyis a TSS

a selector egy task leíróra mutat

taskhoz tartozik egy lokális leíró tábla, ezt is mentjük context switching-nél, ez a

GDT egy | LDT sok lehet | átírás védhető

TSS-ből tölti fel

Rendszer szegm. reg-ek

Rendszer vezérlő reg-ek

	31	12	0
CR0			
CR1			
CR2			
CR3		////	

MSW ← machine status word → állapotbitek

CR2: laphiba lin címe

CR3: laphönyvtár báziscíme (laptáblakönyvtár kezdőcíme)

20 bit ez értékes; laphezdethez ez nem kell

intelnél tárolni kell mert pagefault-nál újrakezdi az utasítás végrehajtását (adresse mode / protected mode), miután behozta az új lapot

MSW

0. PE védelmi üzemmód
1. MP math present (ha van math co-proc: ezt bebillenti)
2. ET 387 kiterjesztés (387-es co-proc van nem 287-es)
3. PG lapozás kapcsoló (van-e lapozás / virtuális tárkezelés)
4. TS task kapcsolt (ha van co-proc, task váltásnál van virtuális math proc.-is. Ha ez bebillen, akkor a math proc állapotát is menti)
5. EM emul. Coproc.

math co-proc-hoz

A rendszer regisztereket csak privilegizált utasítások kezelhetik

Ha nincs math-co-proc, akkor ha ez bebillen, emulál, nincs fizikai co-proc, de a program algoritmus hajtja végre a számítást a „sima" procin. Ez persze lassabb.

Multitasking

megoldandó:

a) lehetnek olyan prgr és/vagy adat részek amelyekhez több task kíván hozzáférni, illetve olyanok amelyekhez csak egy. *Ezeket el kell különíteni, de a közös objektumhoz közös hozzáférés kell*

b.) Task váltáskor a task-ot leíró adatszerkezetet cserélni kell

Táblák a leíróknak → globális vagy lokális logikai címtartomány

a.) – GDT globális leíró tábla /GDTR/ mutatja az elejét
egész rendszerre kiterjedő csak egy van

– LDT az egyes taskoknak (opcionálisan) egy-egy saját leíró tábla /LDTR/

– IDT megszakítás leíró tábla /IDTR/

– GDT és LDT között TI bit /szelektorban/ választ (* 34.o)

b. Task állapot szegmens TSS /speciális memória szegmens/

31 — ez egy TSS modellje: … 4 kbyte (????) — 0

TSS
I/O Bit Map (* 37.o)
op. rendszer függő
LDT
Szegmens reg-ek
Állapot-Flag-reg.
Ut. szlá (EIP)
Laptábla könyvtár kezdete
Privilegizált SP-k
Back Link

TSS egy task virtuális processzorának állapotát tartalmazza *(op.rendszer menti)*

ez automatikus, mindig fix

TSS deszkriptor csak GDT-ben lehet

→ ∀ belső reget menteni kell

*Ha egy taskban hibát követek el, azt ne vihessem át (1) másik taskra (2) másik privilégiumszintre.
Alacsonyabb szintről magasabban lévő fgv-t hívok, a paramétereket stack-en keresztül adom át. Ha itt hibát követek el, az magasabb szintre jut
⇓ védelem
∀ szinthez rögzített stack pointer érték van. Ezt az oprendszer adja.*

Új task létrehozásakor az op. rendszer létrehoz egy TSS-t és beállítja a kezdeti értékeket

/TR reg: futó task szelektor és deszkriptor/

*⇓ ∀ taskhoz 3 SP érték van; a 4. szinthez nincs.
Felsőbb szintnek külön stack-en adok át paramétert.
∀ taskhoz külön laptáblakönyvtára tartozik, ennek kezdőcímét tárolja*

M6

TSS-ben egy bit jelzi a leíró tartalmát CSAK tárolja TSS bit, ha foglalt

5/7

Új task futtatására való áthíváskor az op. rendszer

pl: JUMP TSS$_i$ ut-t ad ki (operandus az új task szelektora)
Call, interrupt, exception, Task gate-n keresztül

A processzor:
- ellenőrzi a hozzáférés jogosultságát (DPL, CPL, RPL) és az új Taskleíró
- elmenti a reg-ek értékét a TSS-be (régi) Busy bitjét
- beírja az új task szelektorát és leírót TR-be
  (TR tartalma mutat TSS-re)
- beírja a foglaltságot jelentő Busy bitet a leíróban
- Az új TSS címe alapján feltölti a regisztereit TSS tartalmával (processzor regiszterei LDTR, CR3)

Valamennyi task használhatja a GDT-szegmenseit

Több tasknak lehet közös LDT-je

Különböző LDT-kben lehet ugyanarra a szegmensre mutató leíró

A task nem lehet re-entrant (csak egy TSS$_i$ van!)

Védelem

1, Különböző tas-ok illetve task-on belül (pl kód, adat, stb) szétválasztás szegmensre

2, Szegmensekre és lapokra védelmi hierarchia

3, Hibák hatását task-on belül tartja

Deszkriptorok (védelem szempontjából)

a rendszer objektumait írják le részletesen
- memóriaszegmensek (adat/kód)
- rendszer objektumok (pl táblák)
- speciális (pl TSS)
- hozzáférés vezérlő /Gates (kapuk)/

↓ indirekt ill. hozzáférés

(36)

2 bitnyi adat: IOPL
Figure 2-4.

I/O Bit map

Egy állapotvektorban I/O privilégium szint

Egy program csak akkor hajthat végre I/O műveletet, ha az ő szintje magasabb, mint az I/O műveleté

A task I/O privilégium szintje az EFLAGs regiszterben van: 2 bitnyi: ez egy szám, privilégiumszint. Csak azokat az I/O eszközöket használhatja a task, amelyiknek a privilégiumszintje kisebb a FLAG-ben meg lévő értéknél

---

386-os 286-ra kialakított szegmensekkel is tud dolgozni.

286 még nem tudott I/O bitmap-et

pl.: státuszok kiolvasásához ~~286~~ is oprendszer kellett a 286-nál. Ez nehézkes, ne kelljen mindenhez oprendszer: rendszerhívás

∀ lehetséges I/O címhez hozzá lehet rendelni 1 bitet ⇒ I/O Bit map: 64 kbit

Ha az I/O bit map adott ~~I/O~~ I/O-ra I/O-címre vonatkozó értéke: Ø ⇒ hozzáférhet más is

---

Figure 6-2.
stack pointer =
= stack szegmens tartalom + stack pointer tartalom

MINDENHEZ CSAK LEÍRÓN KERESZTÜL FÉRHETEK HOZZÁ

már a leírónak is van privilégium szintje

---

task állapotleíró: kapcsolat a fizikai & virtuális proci között

1 taskhoz 1 TSS tartozhat

Figure 6-3.
báziscím: 32 bit → kezdet } memóriában 2db szó
limit
A limit és a báziscím nem összefüggő a 286-tal való kompatibilitás miatt
286: báziscím: 24 bit

Figure 6-3. folyt.
Ha a szegmensméret < 1 Mbyte => byteonként változtathatom a határt

|| > 1 Mbyte => 4 kbyteonként

Ez a G-bit : granularity

P-bit : present : bent van-e a leíró által mutatott lap a memben. Ha nincs => pagefault

Type : gate típusa

Figure 4-8.

4-10. → ellenőrzi, h. azonos szintről jött-e a kérés
ZMP-vel nem lehet lefelé lépni privilégium szinten

Call-gate-en való áthaladás:

Call-gate-leíró : attribútumok

↳ báris és limit (helyett) selector és offset

új címre mutat átengedi, de ő mondja meg, h. hová engedi

ez NEM az általam megadott: a kapu offsetjét használja

Figure 4-11.
újra ellenőrzi gate: milyen privilégiumszintről milyen szintre visz

Figure 4-A.

1. ábra } memória típusú
2.

3. ábra - rendszer típusú

leíró (e 2 féle van)

---

lapleíró - 32 bites
szegmensleíró - 64

∀ virtuális szegmenshez hozzárendelődik egy leíró

38.

6

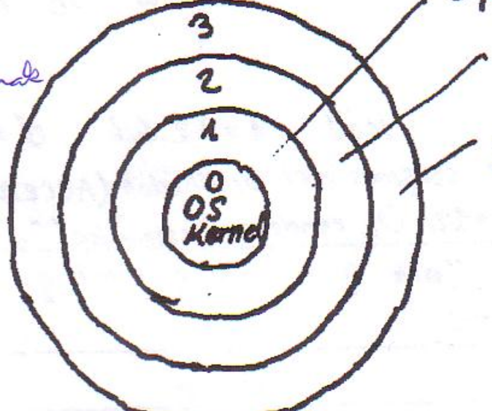
Privilégiumok

- 80386 4 pr szintet kezel |0,1,2,3| (0 a legmagasabb)

CS reg-ben levő szelektor privilégium szint azonosítója

- CS reg-ben levő szelektor privilégium szint azonosítója

az éppen futó rutin szintje: CPL → ehhez képest már hivatkozásnál, ki...



Ezen szintek között a 386-nak kötött elérési szabályai vannak

hozzáférhet-e a task az adott hivatkozott leíróhoz, vagy nem.

pl.: MATLAB

hozzáférés a szintekhez

Δ data
□ kód

→ legális
⇢ illegális } hozzáférés

→ azonos szintű v. alacsonyabb szintű adatállományokhoz férhet

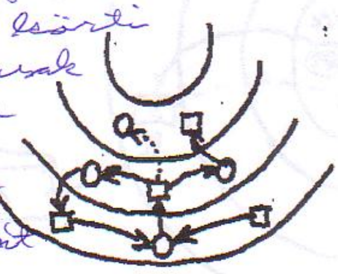
kód → kód másik kódszegmensre csak a saját szintjén hivatkozhat

Kivétel: Ha az. szinten egy kódot KONFORM-nak nyilvánítanak, ehhez alacsonyabb szintről is hozzáférnek

* kivétel ha a kod un. kon...

Ez egy jelzőbit a leíróban

Szintek közötti kommunikáció



O kapu
□ kód

Új: Nem hozzáférés vezérlő objektum: ez a kapu (gate)

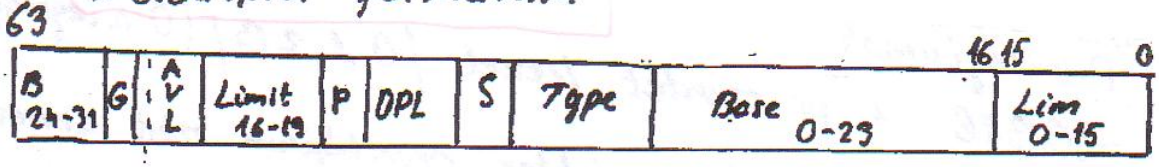
Különböző szintek közti program átjárás csak az indító szintjén lévő kapun át lehetséges. Kapukat az op.rendszer biztosít

Ez a védelmet szolgálja.
A kapu határozza meg, hova mehetek, nem én.
„K-épület: lépcsőház effektus"

az alsóbb szinteknek stack szegmenst kell b... sítani a magasabb sz... / hogy ne legyen h...

illetve a stack egy részét ...já a magasabb szint st...

Deszkriptor formátum



S=0 rendszer, gate
S=1 mem

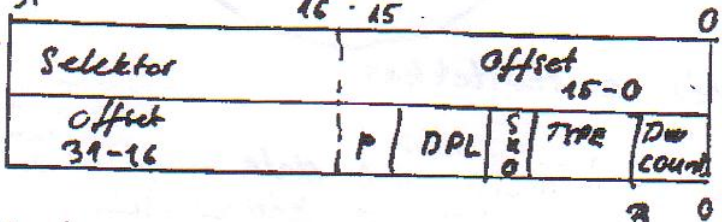
S 1 E Ex/C W/R A

pl LDT, TSS, Gates
p present
DPL deszk privil
Adat/kód
E=0 → adat
Ex Expansion dir.
W írható
C Conform
R olv / nem olv

D kód 1=32 bit 0=16 bit
A=1 szegmes szel. betöltődött (Accessed)
AVL=az Op rendszer használhatja (Available)

Gates Task g, Call g Int g. trap g, TSS busy
S=0

Gate deszkriptor



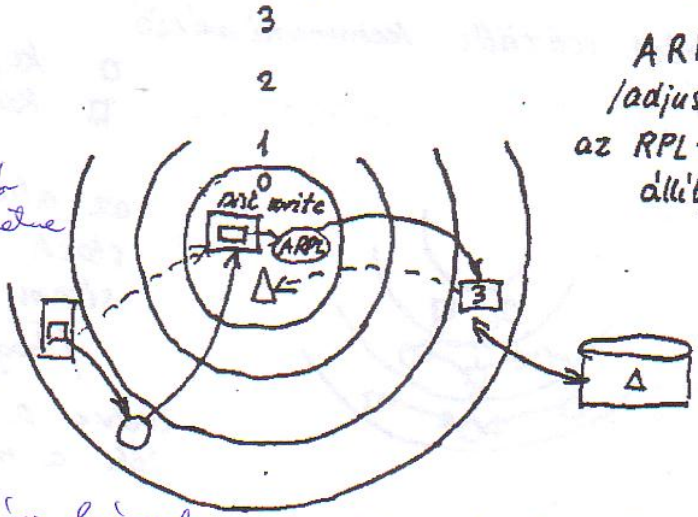
0 Task gate-ban csak selektor van /a GDT-ben levő TSS leíróra mutat/ s ez mutat a TSS-re

Virtuális Offset nem használt

Szelektor mutat egy táblára (pl Call gate)
TSS-re (Task gate)
A táblából veszi elő a tényleges adatokat

RPL használata (Requester privilege level)

0 gate



ARPL utasítás /adjust requested priv. Level/ az RPL-t a hívó CPL-jéhez állítja

A 3. szinten lévő felhasználó disk írást vagy más műveletet végeltatni: paramétereket megadja a file nevet" "rendszerhívás" 0-ás szintű fogl. indul. De ha a fájlnál bármely privilégiumnál van, amelyik átírható 0-ás fogl-nél ARPL = ajust privilegium level = utasítás: ez alacsonyabb szintet állítja be (0 és 3 közül) → már csak 3. szinthez fér hozzá a fogl.

Figure 4-1.
adat seg. 0: ez adatseg. ezzel jelzi
1. ábra A: access-bit hozzáférhető-e
W: írható-e
E: extended bit
lefelé v. felfelé bővülő adatsegmentről
van-e szó
határellenőrzéshez kell

code seg.
2. ábra 1: ez code seg. ezzel jelzi
R: olvasható v. csak végrehajtható
C: konform kód
/* szintátlépés */

AVL: oprendszer ide bármit írhat,
az architektúra nem
használja
B: big → 64 bites kód
D: 16-32 bit közti választás

Reserved: itt ez nem értelmezett,
de NEM is szabadon
felhasználható
- foglalt későbbi típusokhoz

— 6 —
Figure 6-7.
1 taskhoz 1 db állapotseg.!
"megívásakor busy bit foglaltba billen
Hogy 1 task több helyről is hívható
legyen, több táblában helyezzük el
task gate-et; a gate-re hivatkozzuk, ha a
task-tól akarunk vmit.
Task-gate: lecseréli a selectort
itt csak selector van benne.
Ez a selector választja ki a
globális táblából a task-leírót (41.)

44.0

(*1) privilégiumszint

lapszervezés privilégiumszint — lineáris cím

0, 1, 2 —> system
kommentálás
3 —> user

táblázat: mikor melyik szint írhat, olvashat

TLB: lineáris címtérben működő 4 utas set associative cache (*2)
↗ cache-ben keresés és
↘ laptárban
} egyszerre indul a keresés

kiválasztáskor ide tölti a lapot, gyorsít nagyon sokat

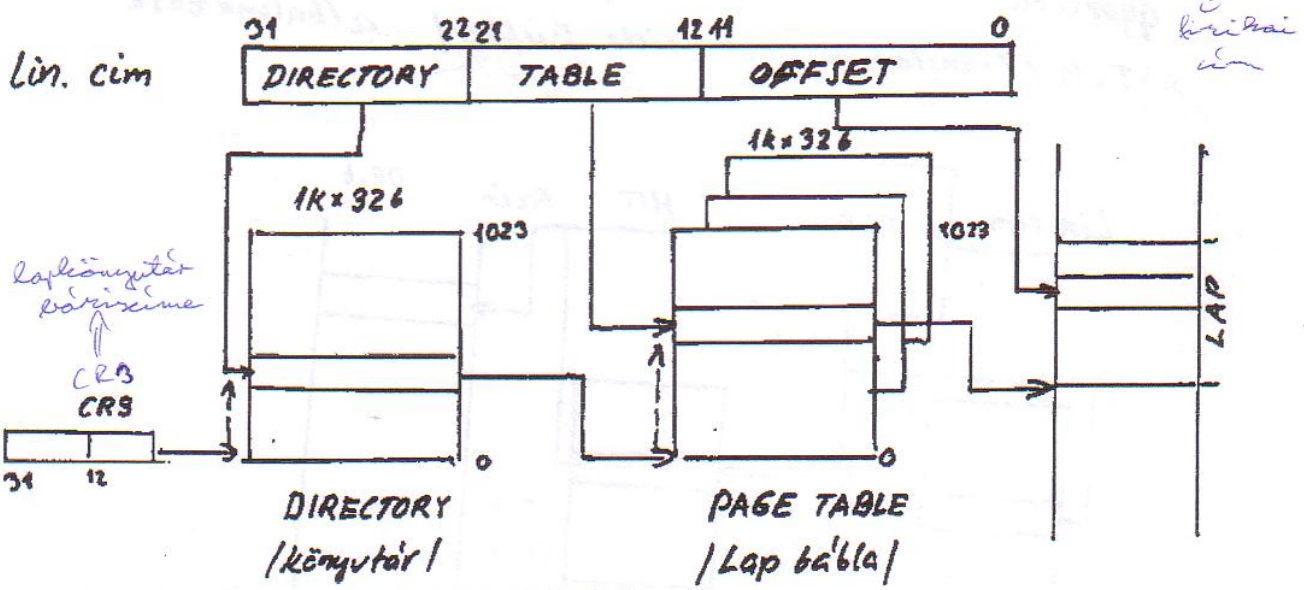
(*3) taszkhoz új laptáblakönyvtár tartozik => CR3 -at át kell írni
⇓
ekkor érvényteleníteni kell a CR3-t, tartalma elavult

386 lapozás (MMU)

- virtuális tárkezelés
  előny:
  - a fix lapméret jól „illeszthető" a háttértár szektor méretéhez
  - elkerülhető a memória „szétesés" (fragmentation)
  - nagy objektumoknál (pl mem szegmens) a fizikai tárban nem kell folytonos tárterület
  - láthatatlan a felhasználó számára

  probléma
  - ha csak lapozás van csökken a védelem lehetősége (szegmens jobban védhető)
  - internal fragmentation (pl lapm. 4096b kell 5000b)
  - nagyobb overhead kell

386 szegmentált lapozás: segment:offset → linc. → fizcím

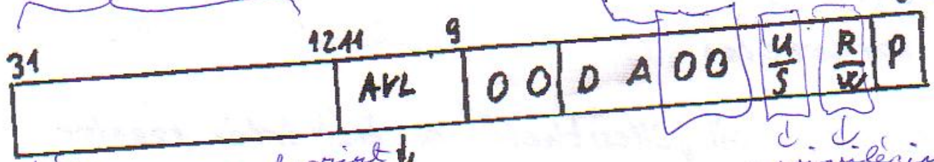


1k×1k = 1M 1M×4k = 4G /2^{32}/

kétszintű lapozás előnye
/nem kell az egész laptáblát bent tartani/
pl /szegmensenként külön laptábla

csak 486-nál : elérhető-e a lap vagy sem pl.: ha I/O illesztőnél nem cache-elhető

csak ez a 20 bit a különbség



könyvtár PT address — 1. szint — OS.-nek

vagy PF address — 2. szint

laptábla

privilégiumszint leíró kezelése: 0, 1, 2 → system; a 3 → user

U/S	R/W	3-nak	0,1,2
0	0	nem	R/W
0	1	nem	R/W
1	0	R only	R/W
1	1	R/W	R/W

P present =1 (bent van-e)

A accessed (ha betöltöttük & hozzáférték =>1 : 1 bites gyakorisági számláló)

D Dirty (írták-e a behozott lapra)

U/S User 3 szint; system 0,1,2 szint

R/W

cache írás stratégiák miatt

ha minden bent: az operendszer rendelkezik egyéb bittel (az első 20-on kívül)

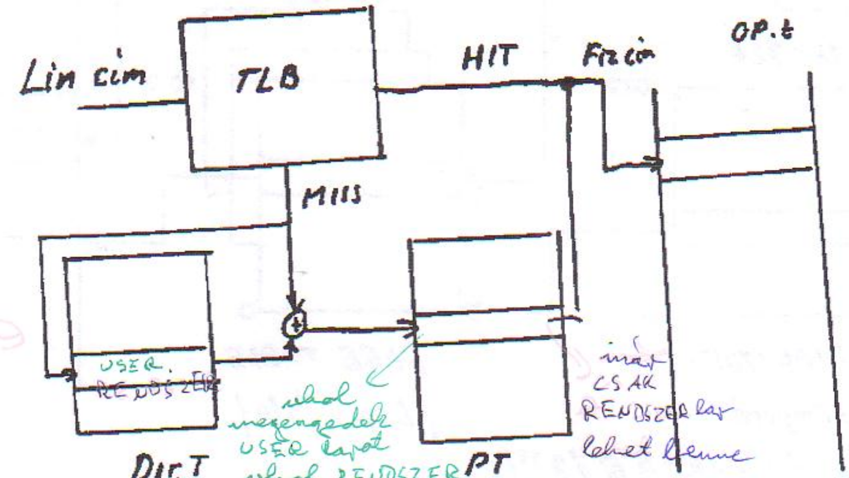
Könyvtár és lap védelmi lehetőség

pl .10 01 => 01

megengedőbb felől megyek a szigorúbb felé

gyorsítás

TLB / Translation Lookaside Buffer / alkalmazása memória hozzáférés gyorsítására



4 utas set associativ cash a legutoljára hivatkozott 32 lap adatait tárolja (128 kB)

HT ≈ 98%

érvényteleníteni kell !! pl CR3 írásakor TSS váltáskor /CR3/

Előzőek alapján

Védelmi ellenőrzések

vl.: lásd ← (1) 1 Típus ellenőrzés /pl Cs, Ds/ Csak írható lehet SS
csak kódszegmensből olvasható fel ...; nem írható egy Cs
adat, stackolvasásnál is; írható/nem írható

BUTA! :)

(3) 2 Limit ellenőrzés ellenőrzés: báziscím + limit ≤ limit
báziscím + limit kell legyen

(2) 3 Priv szintek → adatok elérésének korlátozása CPL, DPL, RPL
ez hamarabb van!!!

(4) 4 Vezérlés átadás korlátozása → kapuk → védett átjárók

(5) 5, Utasításkészlet korlátozása

priv. utasítások pl L GDT, L TR — 0. szintnél használhatók

I/O utasítások → I/OPL I/O bit map
egy adott priv. szintű lásd csak akkor hajthat végre I/O műveletet, ha a CPL-nél magasabb szintű a FLAG-ben lévő 2 bites van I/O priv. szintnél ≤

(6) 6, Lapozás szintű védelem

386 üzmódok

real mód → 8086 címzési mód

protected mód

FLAG-be olvasás pop FL stack-ről olvassa fel; ez csak 0-s szintről lehető

VIRTUAL 86 üzemmód

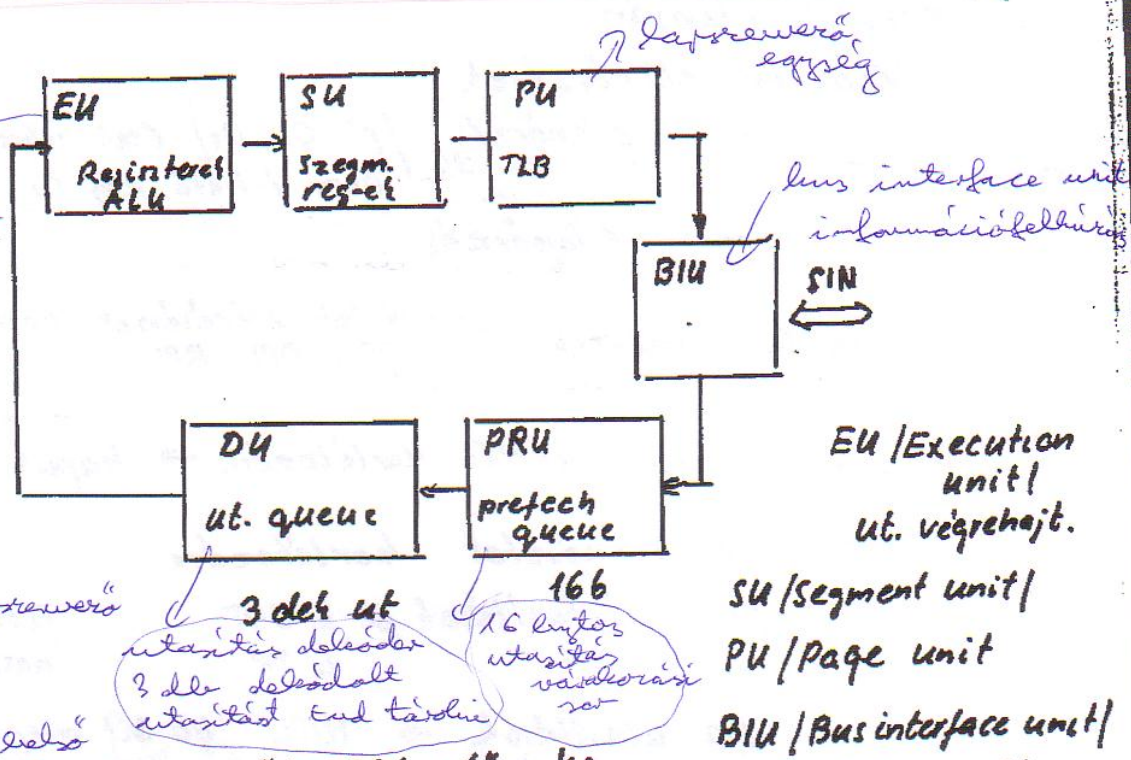
Flag reg bitje VM86 ← TSS tartalmazza meg

286-ra írt programok futtathatók

ez egy virtuális mód: 86-os utasításait hajtja végre DE itt van védelem, nem úgy mint real módban

ebben a módban a szegmensek 1 Mbyte-ra vannak korlátozva ⇒ ezt jelzi a flag reg VM86 bitje

6 386 utasításfeldolgozó pipeline-struktúrája



Pentiumnál már van belső cache ez az L1 cache

L2 külső cache volt, aztán ez is a lapkára került

Pentium Pro-nál már L1a proci magban van

A Pipeline működést előlegzi

- utasítás várakozási sor
  a sin nem érhető el mindig periódikusan
- ut. dek. várakozási sor
  az utasítások végrehajtása nem egyforma időt igényel

probléma: utasítás egymásra hatás pl feltételes ugrás
→ új utasítás lehívását felfüggeszti
a feltétel kiértékelődéséig

címzési rendszer miatt módosítások
- a műv. során kell/nem kell op. lehívás/kivitel
- címzési módtól függő címszámítás
- Van/nincs virtuális tárkezelés

(6.)

M6 7/1

I/O kezelés, perifériák

Ismétlés: a szg. alapvető részegységei: CPU, MEM, I/O

I/O – kapcsolat a külvilág és a CPU, illetve mem. között

Funkciói:
- Ember – gép kapcsolat
- Adathordozók – szg. kapcsolat pl hálértárak
- Adatátvitel / gép-gép kapcsolat/
- Speciális /pl technológiai folyamatok jelei, de lehet a szg belső működését befolyásoló egység is - pl mem. tömbkapcsoló/

A fenti funkciók megvalósításához alkalmazott eszközök a <u>perifériák</u>

Szerteágazó heterogén eszközcsoport, illesztése függ a periféria <u>tulajdonságától</u>, illetve a szg I/O kezelési módjától

pl <u>sebesség</u>
- Nagyon lassú
  Operátori beavatkozás, sec. nagyságú szünetek pl klaviatúra : max 5-10 kar/sec
- Közepes
  mechanikus perifériák, msec. nagyságrendű szünetek pl nyomtató max 1-2 Kb/sec
- gyors
  folyamatos „elektronikus" sebességű adatáramlás pl mágneslemez 1-2 Mb/sec

<u>Vezérlési lehetőségek</u>

(1)
- aszinkron működtethető (megállítható pl nyomtató)
- szinkron /saját ütemében működik, nem megállítható pl mágneslemez/

(2)
- karakteres
- blokkos

átvitelre alkalmas-e?

<u>Vezérlési felületek egységesedése</u>

pl soros /RS 232, RS 485 ... stb
párhuzamos/ BSI, Centronics stb
folyamat /0-20, -4-20 mA, stb/

<u>szabványok:</u>
- jelek
- jelszintek
- mechanikai paraméterek
- protokoll paraméterek

(48)

100-200 Kb/sec vmnyó

→ megállítás csak a sebességgel befolyásolja, az információátvitelt nem.

Bitlink standard interface

fizikai jelek és szabványos jelek - szabályozás technika

I/O kezelés

ismétlés

Utasításkészlet (1) IN, OUT — Külön I/O címtartomány
(2) Mem. ref. ut. — Memóriába ágyazott címtart.

előny ↔ hátrány

I/O-kezelés: logikai szintű eszköz, illetve logikai szintű I/O kezelés

I. Eszközszintű I/O kezelés

direkt írás, olvasás feltétel vizsgálata nélkül
pl kapcsoló állás beolvasása
lámpameghajtó latch írása

feltételes írás, olvasás
jelző bit, vagy bitek használata
pl periféria kész, átvitel nyugtázás → kézfogásos (handshake) adatátvitel

programozott lekezelés
státusz szó – vagy bit lekérdezéssel

megszakításos lekezelés
írás, olvasás igénye esetén IT kérés
az IT-t kiszolgáló rutin végzi az írás/olvasást

fentiekre példa 8255 üzemmódjai

Processzor tehermentesítése

(1) — DMA vezérlő alkalmazása
nagytömegű és/vagy nagysebességű adatátvitelnél

(2) — Nagyintegráltságú, programozható funkcionális elem használata /pl floppy, diszk vezérlő/

pl.: 8085 – kör csak DMA vezérlővel lehet floppy meghajtót illeszteni a sebesség miatt

I. 46 **Logikai kezelés**

- A perifériák sokfélesége miatt a szg I/O rendszere ne kötődjék meghatározott eszközök használatához,
- Helyette általánosított I/O eljárásokat és illesztési felületeket biztosítanak — egységesített I/O felület.
- Nem használnak közvetlen processzor irányítást

Az I/O folyamatok kezelésére
- vezérlő
- periféria-lekérdező
- információátviteli

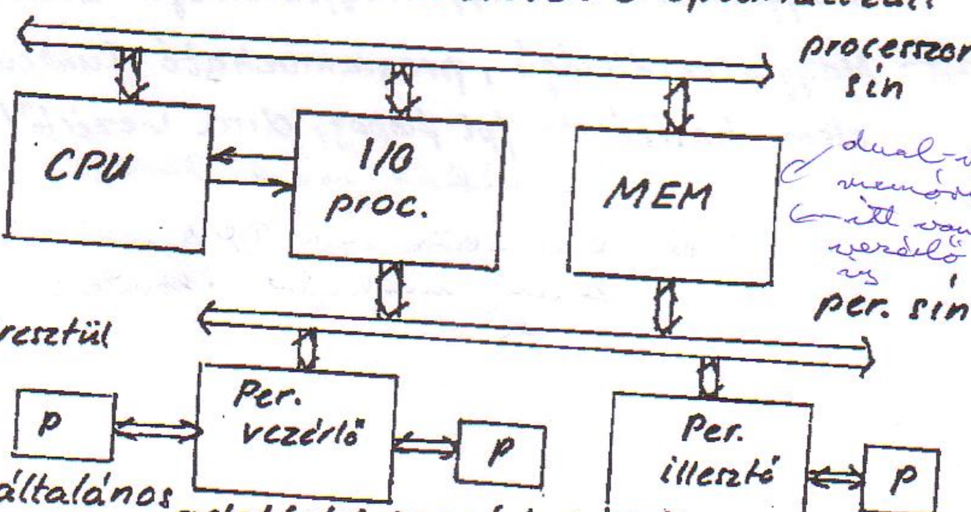
utasítások szolgálnak.

elv: az Op. rendszer „takarja" el az I/O hardver részleteit és az I/O műveletet megfelelően paraméterezett op. rendszer funkcióhívás végezze(ltehet) (LEHET) szoftver IT is / — erre az I/O védelem miatt egyébként is szükség van. Privilegizált IN-OUT lsd 386

előny: a perifériától függő részektől független lesz a felhasználói program

elv: A periféria kezelést végezze különálló, intelligens funkcionális modul / I/O processzor, I/O társprocesszor (társ proc másik neve), I/O csatorna /, amelynek utasításkészlete I/O műveletekre optimalizált

I/O program, paraméter állapot információ a memórián keresztül



dual-port memória — itt van az I/O vezérlő program

I/O proc: általános adatfeldolgozást is ~~végezhet~~ végezhet

I/O proc utasításai pl.:
- START I/O + paraméterek
- folytonosan olvas és újraszerkeszti pl.: szavas adatként továbbítja

46 I/O csatorna (BR vezette be) ⊂ már a CO-processzorok előtt is voltak I/O vezérlő processzorok

Az I/O műveletek ellátó külön processzor egyszerűsített változata → csak az I/O műveletek autonóm irányítására szolgáló utasítások

A csatorna programot az op. rendszer állítja össze

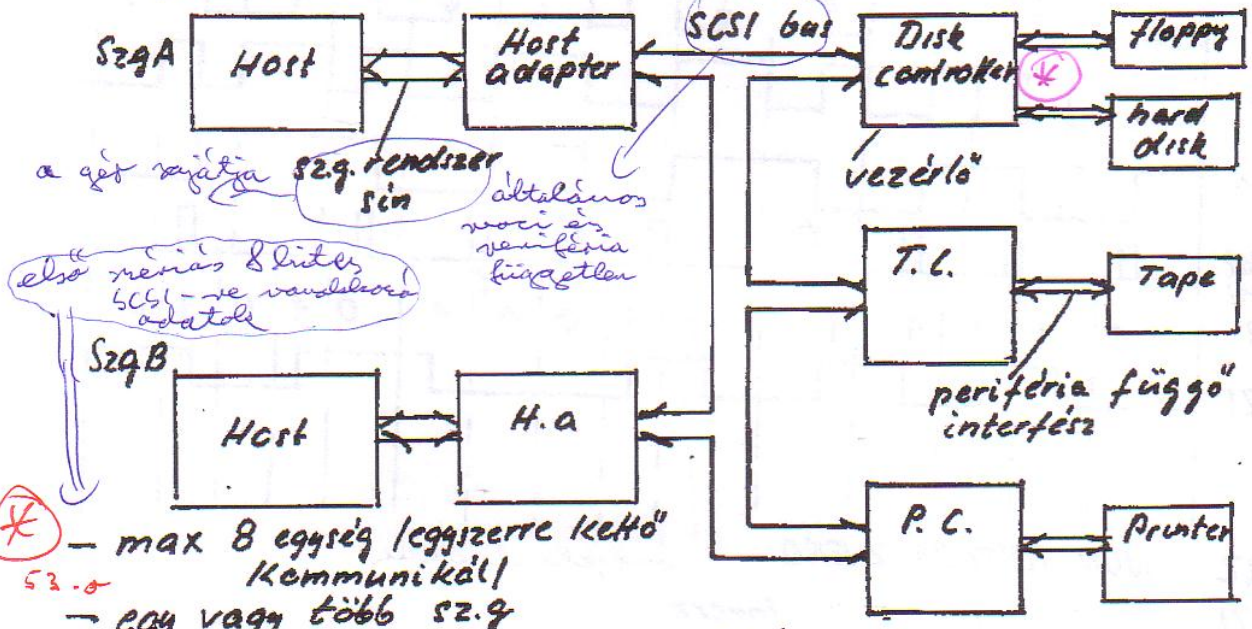
→ start$_{i,cim}$ i = csatornaszám
cim = program kezdőcíme

szelektor csatorna
nagy sebességű perifériákhoz / a csatornaprogram végrehajtása során egy perifériát kezel/

multiplexor csatorna
megosztva több lassú periféria
/a byte-os adatokból adatblokkot - pl szó - állít össze/

SCSI /Small Computer System Interface/

Standard interfész a sz.g. és a perif. vezérlő között



(*) 53.o
- max 8 egység /egyszerre kettő kommunikál/
- egy vagy több sz.g
- SCSI bus vezérlési jog prioritásos alapon
- parancs, állapot, paraméterek, információ
- szinkron vagy aszinkron adatátvitel 1,5 - 4 MB/s / byte-os
- szimetrikus vagy aszimetrikus (*)2 vonalmeghajtás max 15m
- átlapolt periféria működés (*)3
- közvetlen adatátvitel két periféria között

lehetett kettős funkciójú egység is: hol host hol target

SCSI - master = host } információcsere
\ slave = target } adapter

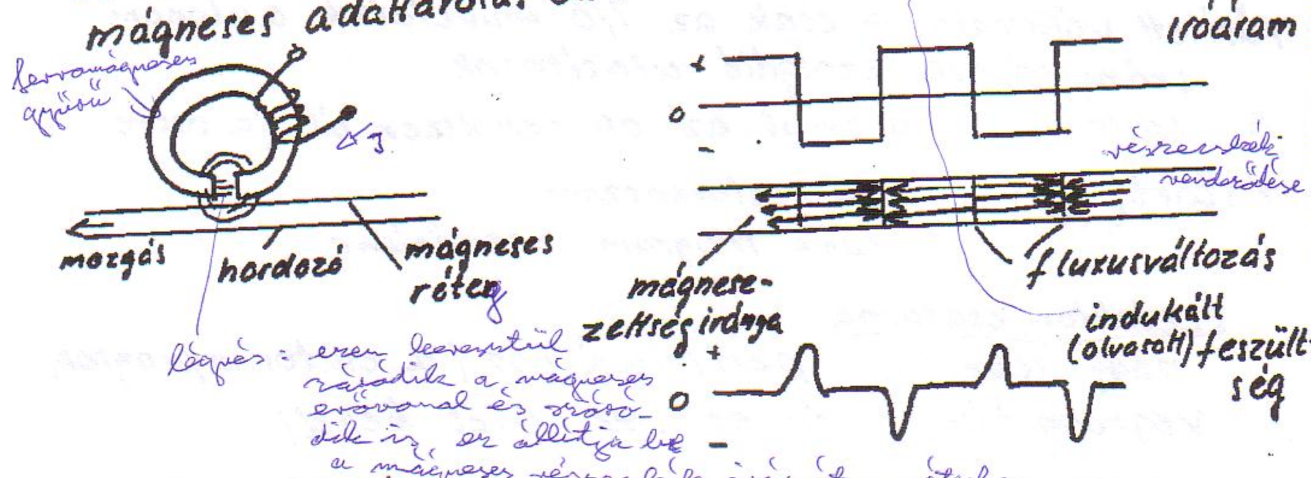
több host & target is lehet

(*)1 erősítások

Perifériák

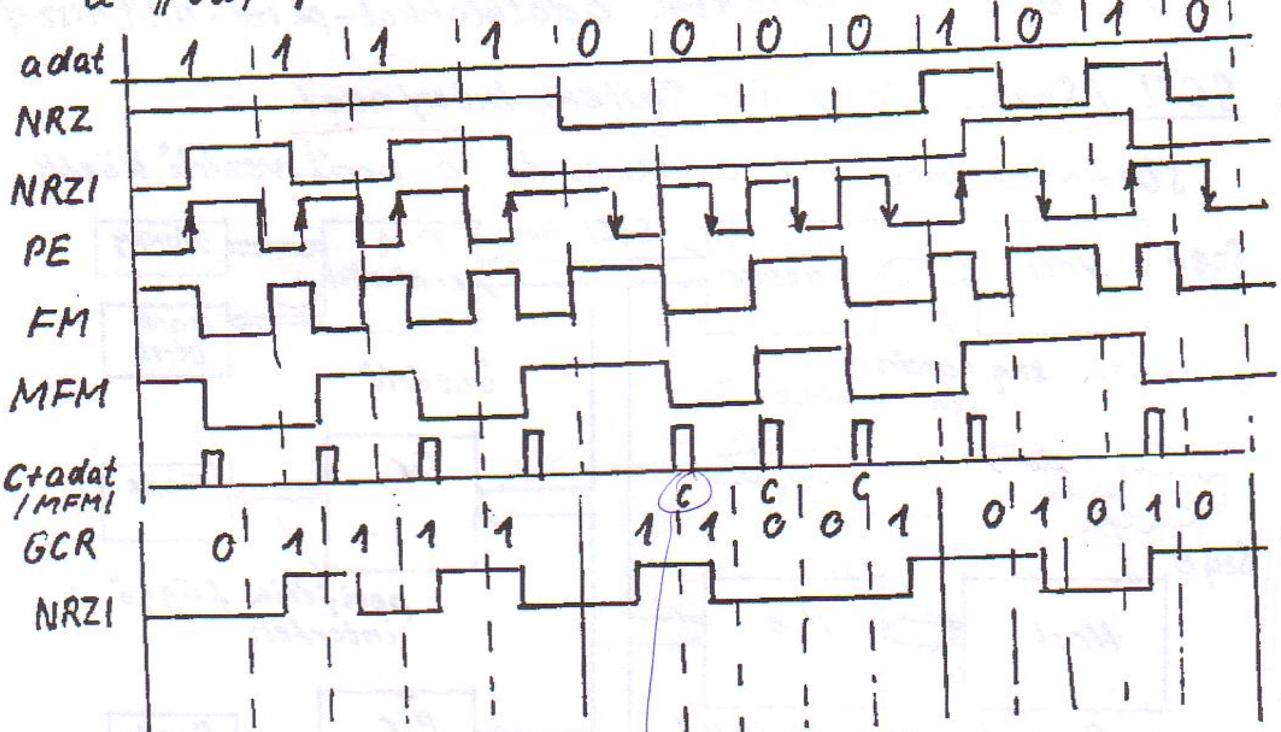
Háttértárak

mágneses adattárolás elve



Adatkódolás

- bináris értéket fluxus változássá kell alakítani
- a „bitfolyam”-ból az adat egyértelműen visszaállítható



NRZ NON RETURN ZERO

NRZI — " — Inverz

PE Phase encoding

FM Frequency modulation

MFM Modified FM

GCR Group coded recording

/RLL Run lenght limited pl RLL 0,2

min hány és max hány 0 lehet

nincs 0 a min és max 2 nulla lehet egymás után

IBM RLL 5,7-et használ

(*) Master kezdeményez, megcímzi a targetet, az felismeri, logikai kapcsolat jön létre

szelekciós fázis → felismerési fázis
arbitrációs fázis
⇓
Több master verseny ⇒ prioritás

(*)2 vonal meghajtás
- aszimm.: egy jel 1 sínen megy, a földhöz viszonyítunk ⇒ zavarérzékeny
  # kisebb seb. rövidebb táv
- szimm.: 2 be & 2 kimenet
  be/ki szint, a két bemenet különbsége a jel
  ⇓
  külső zavarás mindkét jelet azonos irányba viszi: különbség állandó
  ⇓
  kevésbé zavarérzékeny,
  ⇓
  gyorsabb működési seb.
  hosszabb vezeték

— ° —

adat átviteli mód
- aszinkron: szelekciós fázis utáni információcsere handshaking jelekkel zajlik; addig tart, amedig, csak válaszjelnél lépek tovább
  ⇒ lassú DE adatbiztos
- szinkron: adott ütemmel adom az információt és kész
  ⇒ gyors, DE nincs visszajelzés
  ⇓
  adatvesztés

indulásnál a két eszköz „letárgyalja" mehet-e biztonságosan az aszinkron működés

— ° —

Host adapter: „konvertál" a proci sín és az SCSI sín között; a gép a maga ütemében dolgozik, nem kell törődnie az I/O egységekkel

— ° —

(*)3 átlapolt periféria működés: ha nem állnak készen az eszközök, ők kezdeményezhetik a kapcsolat lebontását; ~~...~~ nem foglalja a sínt, átadja másnak. Mikor kész az adatátvitelre kezdeményezi a kapcsolat létrehozását.

Az egy kontrollerre kapcsolódó egységek képesek önmagukban is adatot cserélni a kontrolleren keresztül az SCSI bus foglalása nélkül.
pl.: (*) 51. o

---

52. a floppy / winyó

ön órajelező jel : kemenelve mindig bitsorosan írok mágnes lemezre nem feltétlenül

NRZ = non return zero kódolás || bitcellák
1 : + áram
ϕ : − áramirány
hibia: ha 1k ϕ száma nem korlátozott: csak akkor tudom mennyi bit volt, ha van órajel. DE mi van ha változik a fordulatszám ?
Ez NEM órajelező. Rossz.

meghatározott max bit db szám után mindenképp kéne egy váltás, ekkor lehet szinkronizálni mind órajelet mind fordulatszámot

NRZI : non return zero inverz
cellatartalom 1 ⟹ cella közepén váltson áramirányt
||
olvasásnál itt fluxusváltás
−||− ϕ ⟹ semmi nem lesz
Rossz. de másodlagos kódolással kiegé: egymás után max adott számú ϕ jöhet : így már jó.

PE : fáziskódolt rögzítés
cella közepén mindenképp vált irányt, valahogy
$\bar{0} \to \overset{+}{1}$ 1 jel
$\overset{+}{1} \to \bar{0}$ 0 jel } a bitérték
∀ cellában min. 1 db váltás helyezkedik, ez már jó órajelező kódolás
DE túl sok is a váltás
főleg szalagoknál használták

FM: frekvencia modulált
- ∀ cella elején ∃ változás
- cella közepén csak akkor ∃ változás, ha a cellatartalom 1.

jó önszinkronizáló

régen floppiknál

DE felesleges a cella elején & közepén is váltortatni

MFM: módosított –"–
- cella közepén akkor változik, ha 1 a tartalom
- –"– elején –"– , ha ∅ a tartalom
de ott is csak akkor, ha az előző cella tartalma is ∅ volt

— o —

Írófej: 1-be kötött T flip-flop-ra adom a jelzést, akkor vált irányt az áram.

~~Adjál~~

— o —

GCR: csoportkódolás /⇒ RLL a másik neve/
pl.: - ∀ 4 bithez hozzárendelek egy 5 bites kódot ⇒ nem lesz olyan adatfolyamat, hogy adott időn belül ne legyen jelváltozás. Biztos lesz.
Ez a kieg az NRZI-hez.
Így már jó, önszinkronizáló

— o —

MFM-~~nél~~mel 2x annyi adat rögzíthető mint FM-mel;
kiegészített NRZI-vel még 50%-os javulás

— o —

(*) Az a jó tárolási szervezés, ha a fejet minél kevesebbet kell mozgatni, a cilinderen írjak végig, fejmozgás nélkül.

14 oldalak voltak az első HDD-k.

(*²) Minél közelebb vagyok a lemezhez, annál nyugodtabban tudom valami a sínt.

⇓

nem léptető motor van

szervómechanizmus /* vége: alsó távcsán csak a szervó fej jelei voltak, vezérlőjelek */

+ gond: A tracken legfelsőtől a legalsóig n. annyi adat: ma már ez is meg van oldva.

új olvasási mód:

nem indukció

vezetőképesség!! ← magneto rezisztív

⇓ olvasófej – mivel olvasom

⇓

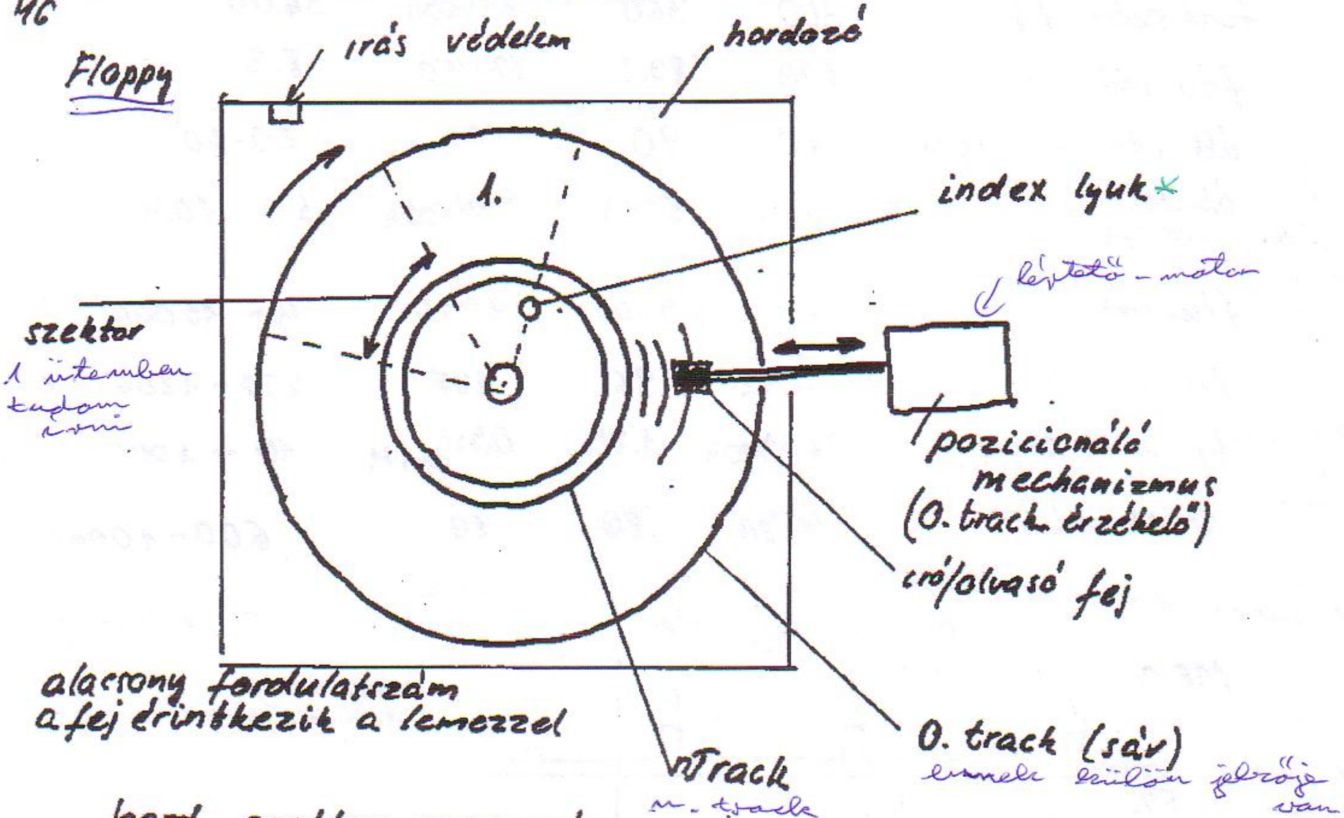
magnetic adatsűrűség

Ez már nem csak vízszintes, hanem függőleges tűrés – irányváltás is van.

Információ tárolás szervezése mágneslemezes tárolón

46

Floppy



alacsony fordulatszám
a fej érintkezik a lemezzel

hard szektor szervezés

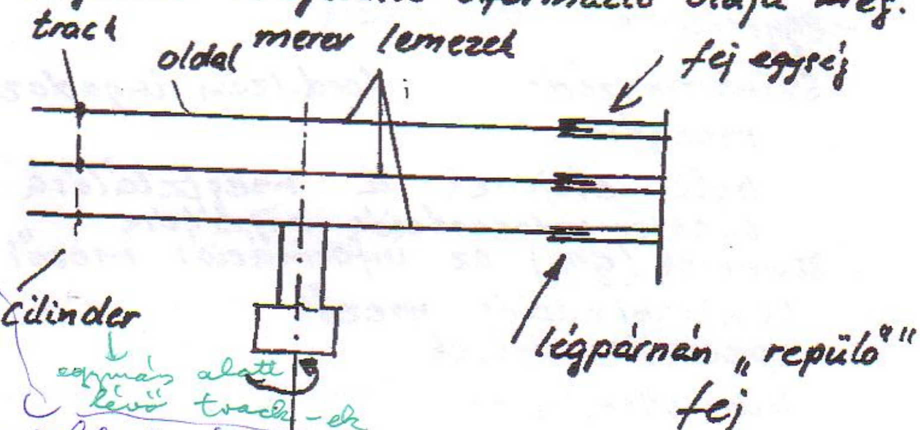
* minden szektor kezdetét egy lyuk jelzi
+ 1. szektor kezdetén: 2 lyuk egymás után

szoft szektor

→ csak 1 db lyuk van, ez az 1. szektor kezdetét jelzi

csak index lyuk van, a szektor szervezést a lemezre rögzített kiegészítő információ oldja meg.

Hard disk



(*) 56.a

a szektorok eleji azonosítók is mágneses jelek, nem fizikai lyukak

A fej írás/olvasáskor fizikailag is a lemezhez ér flopinál

sokkal végebbi, mint a floppy

Az elején üveg tárcsára vitték fel a mágneses anyagot

hermetikusan lezárt egység
nagy fordulatszám
fej és lemez nem érintkezik (10-20μ távolság)
(parkolópálya)

(*)[2] ma: 10 nm

szek.: léptetési idő + fordulási idő

HDD-nél 10x gyorsabb fordulás, mint a floppynál
A pörgő lemez légpárnát alkot, a fej lebeg a tárcsa felett; parkolópályára küldés: később automatikus

(57)

MG	5¼	5¼ (high density → Hd)	3½	HDD 5¼	„ma”
ford. szám f/p	300	360	300/600	3600	7200–15000
fátl. idő ms	100	83,3	100/50	8.3	
átl idő ms (seek)	90	90	100	20-40	4 ms
átvitel Mbit/s MFM	250k	500k	250/500k	5M - 10M	
flux välts fci	~6000	9600	9600	9 - 15000	1 Mbit
tracks tpi	48/96	96	135	250 - 1200	10^5
tip. form. kapac. (*)	360/720k	1.2M	0.72/1.44M	10 - 400M	100Gb
track szám	40/80	80	80	600-10000	

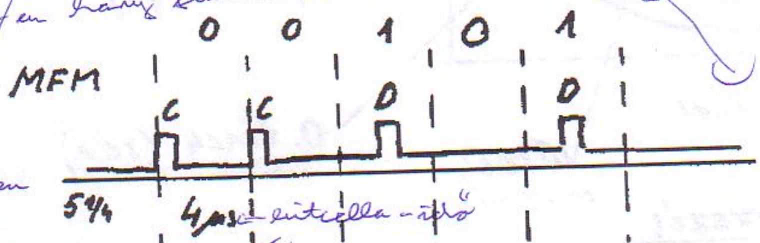
szélekben az elérési idő

fluxusváltozás per inch

1 inch-en hány fluxusváltozás a biztonságos távolsághoz

(*) formázott kapacitás

mindkét oldalra tettek track-et → kétoldalas floppy: 2 lyuk van, eltolva, nem szimmetrikus



5¼ 4µs cella-idő

HI 5¼ 2µs

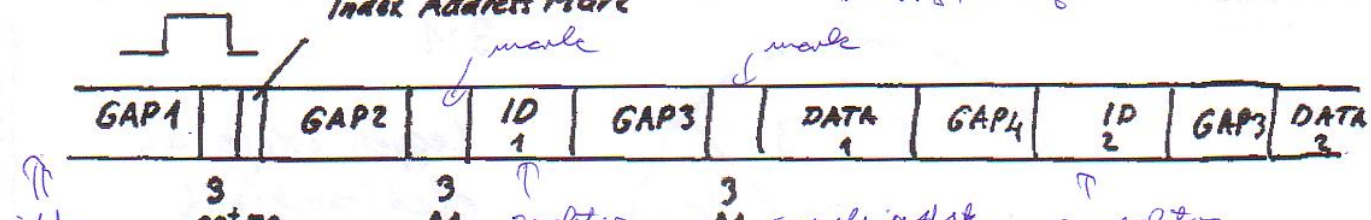
HD 5¼ 200ns

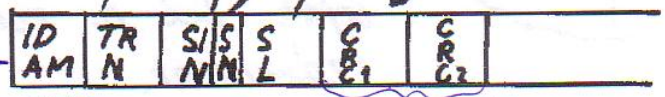
1 inchen sugár irányban hány szektort tudnak elhelyezni

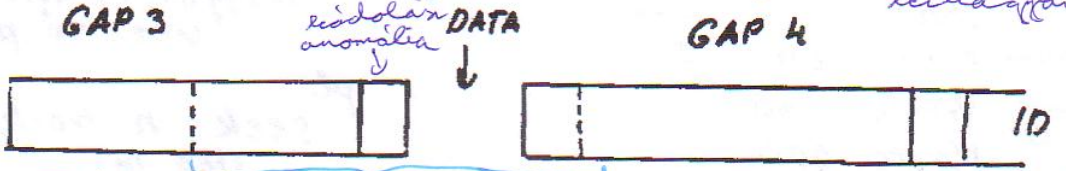
Megoldandó feladatok

- az órajel (C) és adatbit (D) „folyamból” az adatok egyértelmű visszaállítása
- szinkronozódás (pl ford. szám ingadozás) mechanikai mozgásra
  byte elejének a megtalálása
  szektor információk rögzítése
- szünetek (gap) az információs mezők közé
  szinkronizációs mezők → ütemezés, mire adathoz érek, már szinkronizálva legyen
  speciális jelzések → mark-ok a lemezen
  hibaellenőrzés

Szoft szektor formátum (IBM SYSTEM 34, MFM)



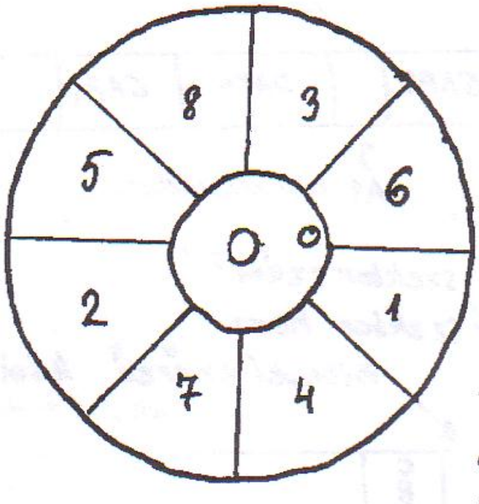




'6 _ _ (→ ne szaladjak túl a lemezen, legyen időm írni)

Szektor interleave

3:1



↑ legyen ideje az elektronikának (pl hiba ellenőrzés státusz, új parancs, stb..)

formattáláskor így írják fel a szektor számokat.

pl floppy meghajtó interfész

(← jön a drive-ba; → megy a drive-ból)

- Index →
- motor on ← (motor ki/be)
- Write data ← (bitfolyam ← T-flip-flop vezérlés)
- Write gate ←
- Write protect → (biztonsági jel)
- Read data →
- Side select ← (fej kiválasztása)
- STEP ← (ahány impulzust kap, annyit lép az adott irányba)
- Direction ← (váltás track-re; melyik irányba lépjen a motor)
- Track 0 → (itt a 0-n vagyok, ezt jelzem)
- Drive select 1-4 ← (szalagon 4 db drive van, sorba kötve; ezzel választok, melyiket használom)
- +5V
- GND
- +12V

(ford. szám váltás)

Nagyintegráltságú vezérlő parancsok

pl
- seek n. track
  - step in
  - step out
- read n. tr. m. szektor
- write -"-
- format track
- stb

Hard disk
- ST 506
- ST 412

} interface IC-ben régen

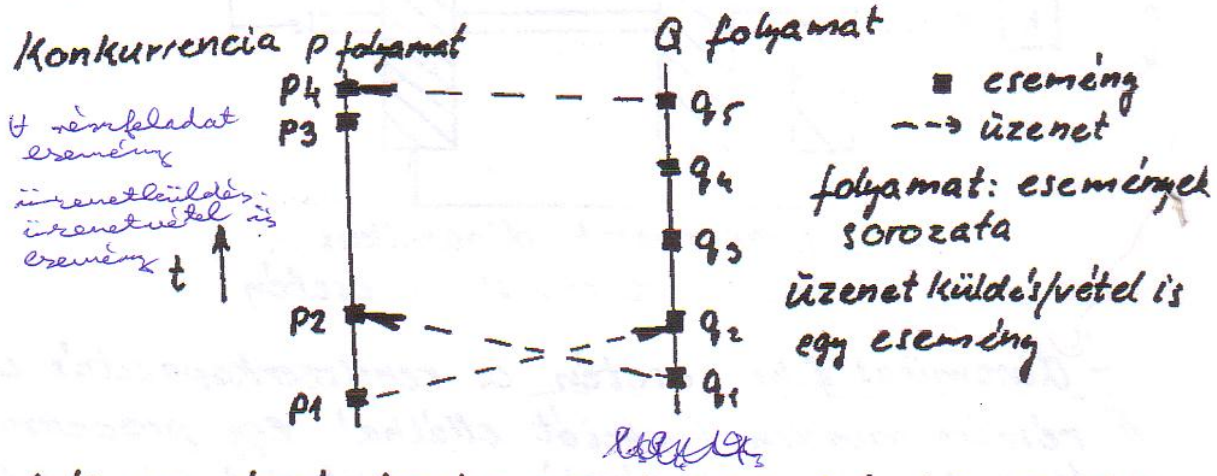
Vezérlő beépítése a meghajtóba → PATA / SATA

(pl AT-busz interfész, SCSI interfész)

Intelligens vezérlők — ma a lemezen belül tök más van, mint amit az interface mutat

Multiprocesszoros rendszerek alapfogalmai

- a µp-s technika (és technológia) lehetővé teszi, hogy egy nagyobb feladat kisebb elkülöníthető részeinek végrehajtását egy-egy külön processzorra bízzuk
  → kell egy mechanizmus, amelynek segítségével az egyes proc.-ok működésüket koordinálhatják a közös cél elérése érdekében
- Multiprocesszoros rendszer: olyan struktúra, amelyben ugyanazon rendszeralgoritmus egymástól független feladatainak konkurens végrehajtása folyik
- Konkurrencia



Két eseményt konkurrensnek nevezünk, ha egyik sem tudja okszerűen befolyásolni a másikat
/pl.: p_3 konkurrens q_3, q_4 eseménnyel/

A multiprocesszoros rendszereket osztályozhatjuk a feladathozzárendelés módja és a processzorközi kapcsolat jellege szerint

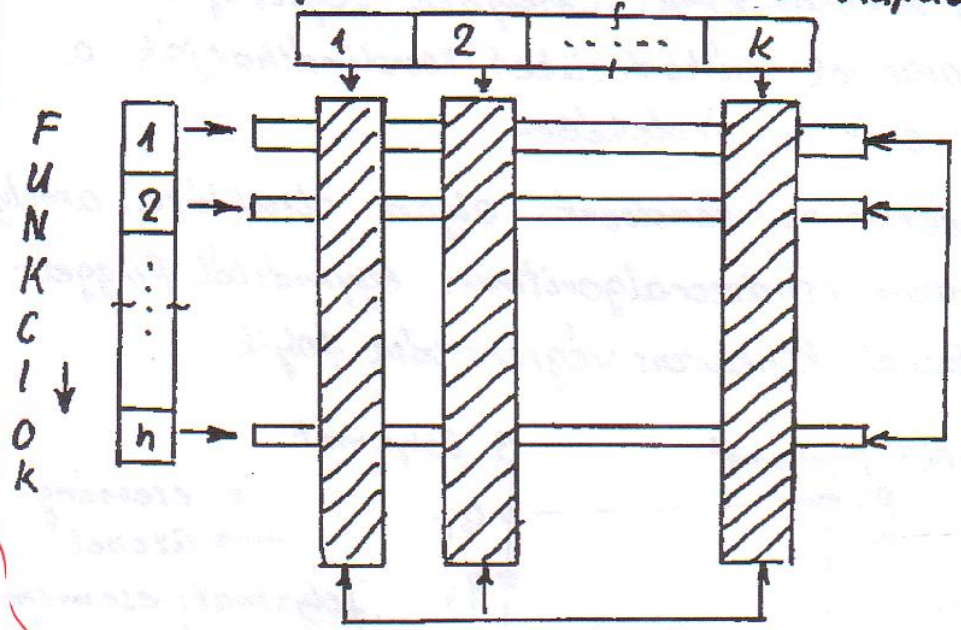
Feladathozzárendelés

I – dinamikus feladathozzárendelés → a leosztáskor rendeljünk egy procit általános célú processzor futás közbeni elosztás, még nehézkes

II – statikus feladathozzárendelés

I. Dinamikus f. hozzárendelés (egy funkcióhoz rendelünk egy processzort)

A multiprocesszoros rendszernek bizonyos funkciókat /szolgáltatásokat/ kell biztosítania a felhasználó számára. Továbbá lehetővé kell tennie minnél több felhasználó kiszolgálását /kapacitás/ → kapacitás



processzorok statikus feladathozzárendelés esetén

processzorok dinamikus feladathozzárendelés esetén

- dinamikus f.hr. esetén a rendszerkapacitás egy részére minden funkciót elláthat egy processzor /terhelési viszonyok alapján egy feladat bármelyik processzornak odaadható (logikailag egyenértékűek a proc-ok)/

előny: a) egy proc meghibásodása a rendszernek csak egy részére terjed ki (megfelelő hibaterjedés védelem kell)

b) a proc-ok teljesítőképessége jól kihasználható (megfelelő terhelésosztó algoritmus kell)

hátrány: a) a proc teljesítőképessége korlátozza a megvalósítható funkciókat/szolgáltatásokat (a proci általános, DE sokszor kratéskompatibilis célproc.)

b) bonyolult szoftvert igényel (a processzorok egyenrangúak)

affine fordítás előtt orvtom el a feladatot (van)

online futás közbeni feladatkiosztás (még nehézkes)

11. Statikus feladathozzárendelés esetén egy processzor egy (előre) meghatározott funkciót/szolgáltatást valósít meg az egész rendszer számára (pl 1/0 proc.)
célprocesszorok vannak, melyek nem egyenrangúak

előny:
- egyszerű szoftver mert minimális a kölcsönhatás az egyes proc-ok feladatai között (minden célproci a saját dolgát végzi => egyszerű ütemező) elmarad a taskváltással összefüggő adminisztráció egyszerűbb az op. rendszer ütemezése
- a proc felépítése a funkcióhoz optimalizálható

hátrány:
- az egyes proc.-ok pillanatnyi terhelése jelentősen eltérhet
- érzékenyebb a meghibásodásokra / feladatát általában csak ugyanolyan típusú proc. veheti át / 1 proci hibás => 1 funkció kiesik

Az együttműködés megvalósításához mind hardver mind szoftver szinten kapcsolatot kell kialakítani az egyes processzorok között => 2 mód kialakítás szerint

1. - Lazán csatolt: a processzorközi kapcsolat üzenetorientált / lassú és kevés üzenet /. Az egyes processzorokon különálló operációs rendszerek vannak / lehetnek ez relatív Mbit / Gbit ... ??? de általánosan igaz

2. - Szorosan csatolt: a processzorközi kapcsolat közös erőforráson keresztül van megvalósítva. közös az operációs rendszer

pl.: Windows-zel összekötve egy linux kiszolgálóhoz. Ez lazán csatolt rendszer.

közös erőforrás: memória, ez a kapcsolódási pont

/* grid : sok gép hálózatban => számítási kapacitás növekedése */

(1.) **Lazán csatolt rendszerek**

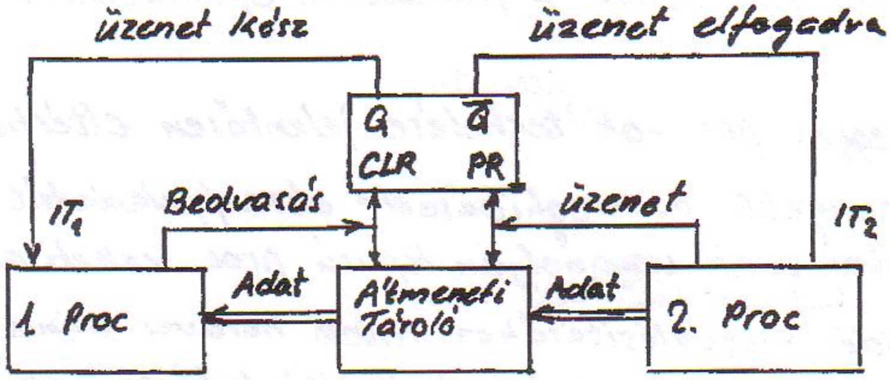
'16 a processzorokat kommunikációs alrendszerekből megvalósított csatornák kötik össze. → üzenetváltás / átadás

A logikai és fizikai közeg függ a rendszer nagyságától, térbeli elhelyezkedésétől.

Lényeg: az üzenet átvitelének idejére a forrás processzor a fogadót perifériájának tekinti

kis rendszernél, kevés információ átvitelére

pl hardverre alapozott szemafor



üzenetküldés átmenetet szemafor jelzi szemafor az együttműködést koordinálja

szemafor → flip-flop

pont-pont közötti feltételes bevitel

előny: egyszerű ↑ pont-pont közti átvitel ∀ proci között

hátrány: lassú, merev /DMA segíthet/

nagyobb rendszernél jelentős válhat

az op. rendszerek együttműködésének biztosítása

ha egyik proci pillanatnyi leterheltsége az egész rendszert leterheli.

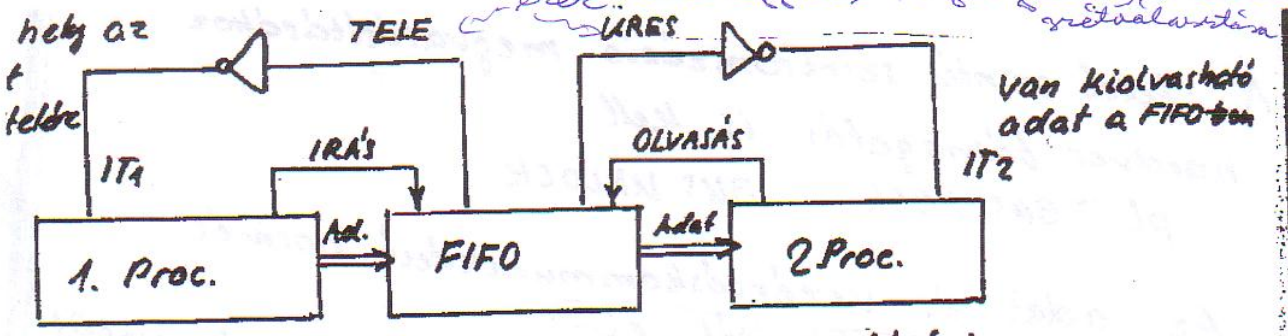
⇓ átmeneti tár kell a procik közé

⇓ szétválasztom a szemafor két irányát

⇓ FIFO - szervezés

probléma: a forrás és a nyelő proc.-ok között nagymértékben ingadozik az adatátvitel sebessége (ez függ az egyéb feladataiktól)

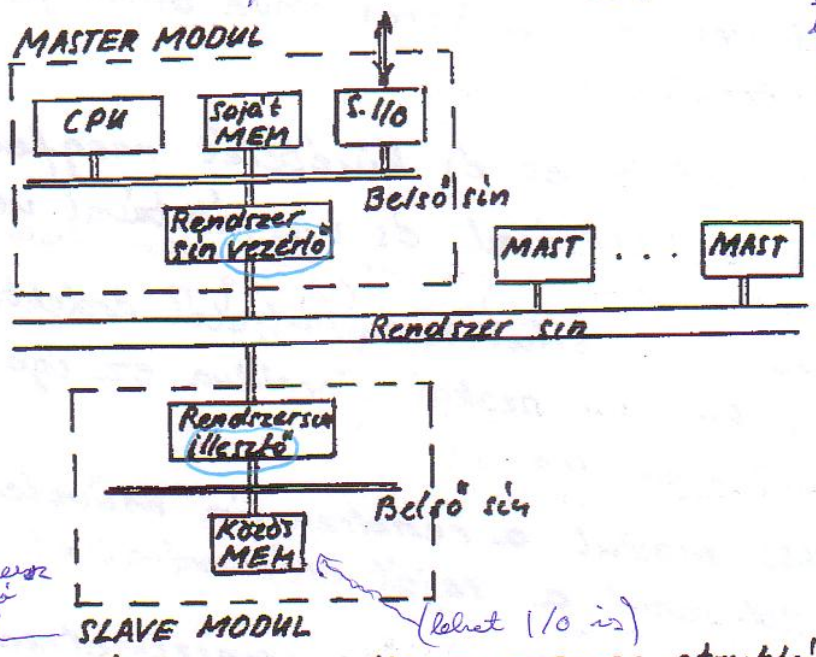
megoldás: mindkét oldal felé külön szemafor és FIFO szervezésű átmeneti tároló



laza csatolás FIFO segítségével /írás/

2. **Szorosan csatolt rendszerek**

kapcsolat közös erőforráson keresztül



Kétsínes multiprocesszoros struktúra

A processzorközi kommunikációnak két szintje van

Felső (logikai) szinten a folyamatok közötti adat- és vezérléskommunikációt kell megoldani (postafiók elv)

Alsó (fizikai) szinten a master modulnak a rendszersínhez (közös erőforrás) való hozzáférését kell koordinálni. Ez lehet központosított vagy elosztott illetve soros vagy párhuzamos megoldású

(ld. később ARBITER modul)

MG

A felső szintű szinkronizáció megvalósításához hardver támogatás is kell
pl BUS LOCK, BUS UNLOCK

Az adat és vezérléskommunikáció normál formája a postafiók elv
a küldő master modul a rendszersínen keresztül adatokat visz át a közös slave modulnak a feladathoz rendelt tároló területére
A fogadó master modul a rendszersínen keresztül adatokat vesz ki a közös slave tároló feladathoz rendelt területéről

Az adatok betöltését és kivételét megfelelő szemaforok beállításával és vizsgálatával végezzük
↳ szemafor: jelzőbit, annak ellenőrzéséhez kell hardware ez egy flag (*) 67

előny: a fogadó modul szempontjából érdekteln ki rakta be azokat. Fordítva ez igaz a küldő modulra is
A többi modul a rendszersín művelet alatt dolgozhat a saját erőforrásaival

hátrány: a rendszersín szűk keresztmetszetté válhat

A postafiók elvnél biztosítani kell a kölcsönös kizárás megvalósítását
amikor master; a postafiókhoz (közös erőforrás) fordul, akkor ezalatt ki kell zárni master$_k$ mod ennek használatából

Szemaforhoz RMW típusú utasítás
LOCK (automatikusan)
s.hw + n.hw megoldás (P és V primitív)

(*)

Egy és oszthatatlan művelet

A szemafor állapot lekérdezése és visszaírása közötti időben más master nem ~~tudja~~ kaphat vezérlési jogot, mert ha mindkét proci egyszerre szabadnak találja a szemafort, mindkettő írni kezd ugyan arra a slave memória területre.

⇓

test-and-set utasítás

DE 8086-nál ez még nem volt, e helyett

bus lock, bus unlock utasításpár

⇓

bármilyen magas prioritással kérem is a sínt, ha valaki lock-ra állította, nem kapja meg

/* utasítás elé bus lock prefix írható */ ~~kötési~~

? stb

(*) 74.o

∀ modulban latch bemenet

∀ kártyahelyen ehhez kötjük a megfelelő számú adat-cím vezetékkel

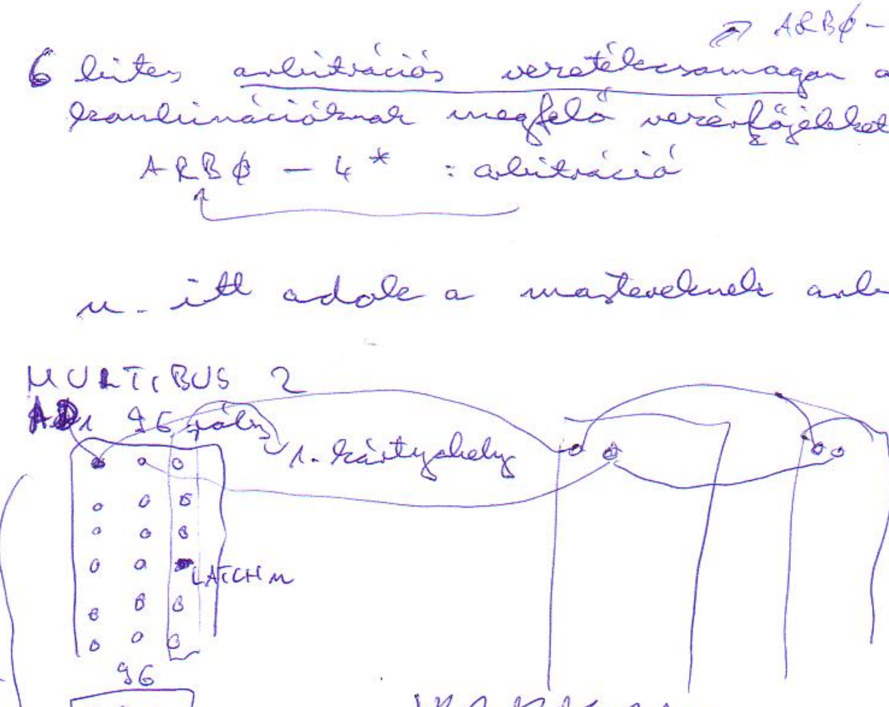
↗ ARB0 – ARB5

6 bites arbitrációs vezetékcsomagon adja ki a kombinációknak megfelelő vezérjeleket

ARB0 – 4* : arbitráció

u. itt adok a mastereknek arbitrációs kódot.

MULTIBUS 2



~~∀ adatvonalon~~

∀ kártya regiszterébe betöltöm a kártyahely értékét, később ezzel hivatkozott

⊛ A megvalósítás a protokollban üzenetként megy át, ahol egy oldalán pedig tényleges megvalósítás lesz.

(*) Maga az előkészítés válasz nélküli üzenetekkel keresztül történik. Az előkészített üzenet már maga a 32 byte-os adatcsomag ill. csomagok.

Sínrendszerek

- Technológiai fejlődés → „építőelemek" kialakulása
- Hagyományos egyedi feladatra tervezett berendezések funkcionális elemeinek tipizálása → egységes illesztési felület → sín /bus/ kialakítása → modularizáció

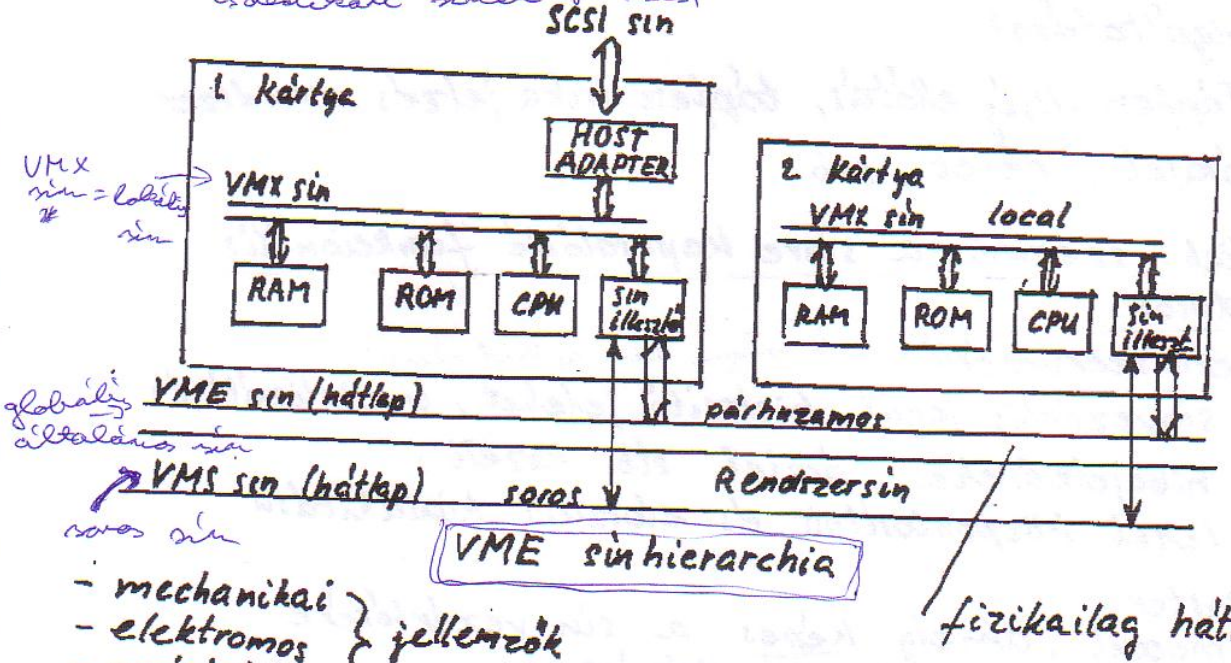
Sín vezetékek csoportja (→ lehet: adatsín, stb. csoportok), melyeken digitális rendszer elemeket kapcsolhatunk össze. A rendszer elemei a sínen kommunikálnak egymással egy definiált protokol szerint

Az összeköttetés típusától függően a sín lehet párhuzamos vagy soros

Korszerű rendszerekben többféle sínt használunk hierarchikus felépítésben

- Alkatrész szint
- Kártya szint /local bus/ (pl.: egy mátrixnak saját belső
- Hátlapszint (NYÁK a sín, hole sínje is van
- Interfész -szint kártyákat fogadok)*

(↳ dedikált sínek pl.: SCSI)



- mechanikai
- elektromos
- protokol

} jellemzők

fizikailag hátlap *

Rendszersín

rendszersín funkcionálisan az alábbi lehetőségeket biztosítja

- Adatátvitel
  a sínre csatlakozó modulok között információcsere → egy MASTER
  lebonyolítása a sín felügyeletét ellátó modul irányítá-
  sa alatt. → információ egy ütemben az adatsín szélességével

- Programmegszakítás
  a sínre csatlakozó modulok kérhetik valamelyik
  modul normál működésének megszakítását
  → prioritási szisztéma → érvényre jutás → kiszolgálás
  ↳ megszakítás kérő / nyugtázó vonal
  megszakítási rutinok címének vonala

- Arbitrálás / sínvezérlési jog kiosztás /
  Olyan rendszerekben, ahol több modul is képes az adatátvitel vezérlésére (pl multiprocesszoros rendszer, DMA) kell egy mechanizmus, amellyel a versengő igények közül egy adott időben csak egy nyeri el a sín vezérlési jogát.

- Szolgáltatások
  Tápfeszültség ellátás, tápfesz. hiba jelzés, rendszer órajel, reset, stb..

Fentiek szerint a sínre kapcsolódó funkcionális modulok =

- Rendszervezérlő ← és végzi az arbitrációt
  sínvezérlés jogát biztosító jelelket, inicializáló és hibajelzéseket, órajel stb kezeli
  lehet központosított és elosztott kialakítású

- Master
  modul, amely képes a sín vezérlésére és adatátvitel irányítására

- Slave

  nem képes a sín vezérlésére, az információcserében egy master irányítása (cím, parancsjelek) alatt vesz részt pl Memória, I/O

- Megszakításkérő

  megszak. kérő jel kiadásával kiszolgálást kér

  (Ez egy jel a sínen amiből egy feldolgozón tömegesen megszakítás lesz.)

- Megszakítás kezelő

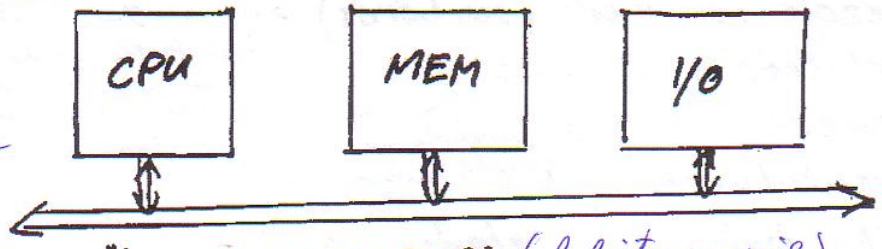
  képes a megszakításkérések érzékelésére és a kiszolgálási folyamat kezdeményezésére

A modulok konkrét kialakítása függ a rendszer specifikációjától.

A sín lehet egy funkcióhoz rendelt (pl I/O sín) illetve osztott használatú (közös) sín

A sín függ az alkalmazott kártyaméretről is

I. - erőforrás particionált sín (kiskártyás rendszerek)



végén: 2 fő csoportja volt a sínnek

jellemzők: - egyprocesszoros (8 bites vonal)
- tipikusan CPU-mem információcsere
- egyszerű adatátvitel
- rövid busz ciklus idő (CPU-wait)!
- szeparált cím, adat, vezérlő jelek
- egyszerű sínprotokol (CPU jelek!)
- nincs hibavédelem

egy-egy funkcionális modul szerint osztjuk fel a kártyát

1 modul 1 funkció

pl.: MEM, CPU, I/O } modulok

II.º – Feladat particionált sínrendszer (Nagykártyás)
lsd VME, MULTIBUS II. stb

kis feladatokhoz: egy lapra vágyaltak mindent ami akkor kellett. 1 kártya 1 feladat

egy kártya több funkcionális modult tartalmaz, így a feladat egyrészét vagy egészét önállóan is elláthatja (SBC elv!)

jellemzők: → „egy kártyás számítógép"

- multiprocesszor orientált (kártyán belül lokális busz + van globális sín)
- üzenet orientált / a kommunikáció intelligens egységek között tipikusan üzenet / ...
- blokkos adatátvitel lehetséges
- nagy áteresztő képesség
  üzenetek sorba állíthatók, buffereltek
- bonyolult sín protokol (nem CPU jelek) → spec. sínvezérlő jelek
- lehet multiplexelt cím/adat, kódolt vezérlő-vonalak → sínvezérlő / protokoll vezérlés nagyintegráltságú

Adatátviteli folyamat:

- Kezdeményezés (sínkérés) — egy master sínt kér, neki kell eldönteni h belső v. küső sínt kér, ki(?)vel akar kapcsolatot teremteni
- arbitráció ← eldönti, h a kérés mikor kap ott vezérlési jogot (külső sínnél globális sín)
- címzés
- adatátvitel ← kivülről címképzés
- hiba észlelés- és jelzés

↓
a cím dönti el, h pl.: egy adott memória a globális v. a lokális címen van-e.

több slave kiválasztása
Címzés: egy, vagy több slave kiválasztása

I. logikai címzés (pl kártyacím felső cím bitek
Kártyán belül alsó cím bitek)
/* az alsó címbitek már a kártyán belüli fizikai címbitek */

| pl VME egyszerre használható 16, 24, 32 bites cím ⇒ AM0-AM5 címmodosító
↳ adott kombinációval jelölöm ki, ezen a 6 vonalon, hogy milyen "szélesség" címet használok ill. milyen adat szélességet.
pl.: byte-os szavas adat

többféle proci miatt

pl címsín szélesség, adat/program, user/supervisor, egyedi/blokkos, I/O stb |

Ezek is AM0-5 -ből jönnek ki →

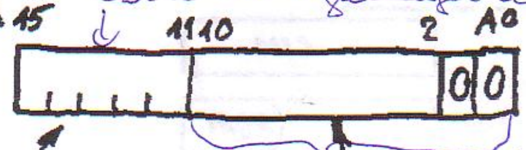
II. helyfüggő címzés (geographical addressing)
pl MB-II. (Intel)

Logikai: a kiadott logikai cím választja meg, hogy milyen ill melyik kártyát választom ki.
A cím a modul /kártya/ rendszerbeli elhelyezkedésétől /fizikai helyétől pl 1. kártyahely, 2. kh. stb/ függ

∀ kártyán címdekódoló, ő érzi, h kiválasztják
A rendszer a kártyát megcímezheti annak ismerete nélkül /Mem, I/O, kezdőcím stb/

Helyfüggő címzés: kártyahely megcímzése. Ezzel a kártyahelyet választom ki.
MB-II -nél a rendszervezérlő inicializáláskor vagy a rendszer újrakonfigurálásakor beírja a kártyahelyek számát /geographic address/ egy a kártyán kiépített regiszterbe, s a továbbiakban ezt használja mint címazonosítót a helyfüggő címzési üzemmódban

Több címtartomány van erre a kártyahelyet címzi → Multibus II. : 20 kártyahely

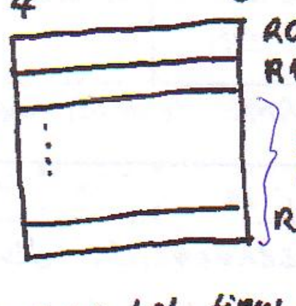


(max) 512 byte-os regisztertömb egy byte-ját jelöli ki
Ebben infók vannak a kártyáról

Kártyahely azonosító
/Cardslot ID/
0-19

Regiszter szám

csak olvasható illetve írható/olvasható, csak írható regiszterek a kártyán



gyártó azonosító
modulhoz tartozó kártya-specifikus adatok

∀ kártyán van egy regiszter, amiben az van h ő hányadik kártyahelyen van. Ebből ismeri fel, hogy őt címezték meg.

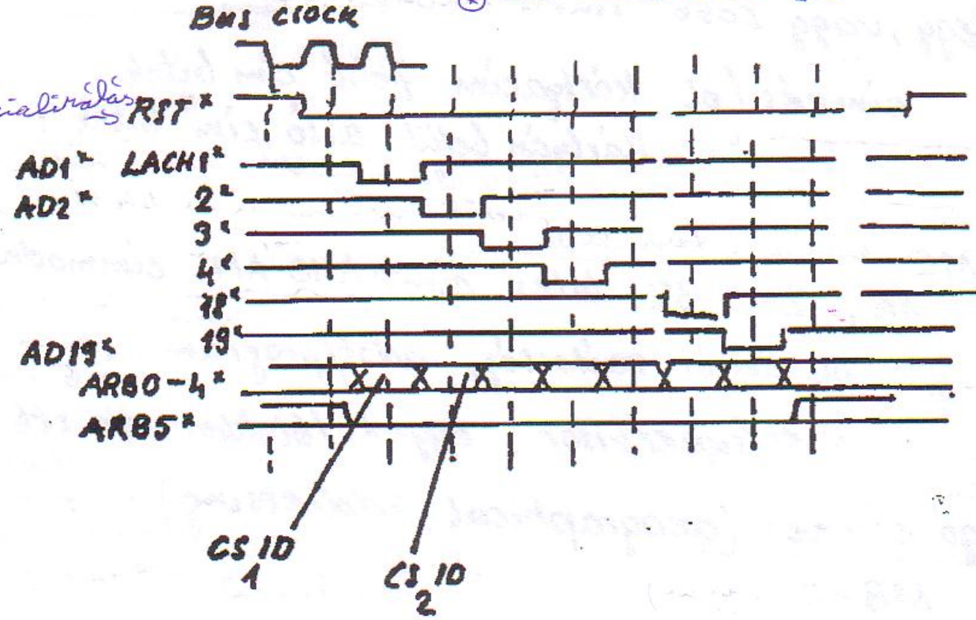
R0, R1 gyártó azonosító
R2-R511 kártyaspecifikus adatok / pl típus azonosító: működéshez szükséges adatok stb.

Ezt a regtartalmat hasonlítja össze a sínen levő címmel

önmagát konfiguráló eszközök, konfiguráció a regiszterekben levő adatokból ⇒ PLUG AND PLAY eszközök

M6 J10

⊛ → mullalstin jel



inicializálás → RST*

ΦΦΦΦ kártyahelyen mindig a rendszervezérlő modul van

6Z.o

akkor ARBΦ-4 a kód kiadása
ARB5 - ez a kód arbitrációs kód-e vagy kiosztás

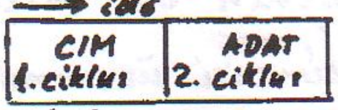
MB-II. helyfüggő címkiírás

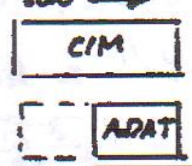
ha ARB5* = H akkor arbitrációs ID.

- legtöbb adatátvitelnél egy slave modul kerül kiválasztásra /megcímzésre/
→ speciális esetben szükség lehet un. broadcall, broadcast /általános olvasás, ált. írás/ műveletre
speciális címmel vagy külön vezetékkel választjuk ki

MULTIPLEXELT CÍM/ADAT — DEDIKÁLT CÍM/ADAT

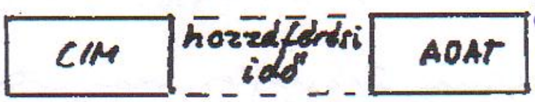
Adatátvitel
→ idő — idő →

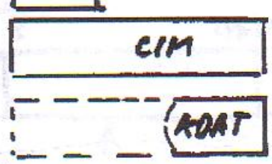




írás szeparált cím-és adat vezetékek esetén

- írás (multiplexált)





közben más nem veheti át a sínt

- olvasás → 1 gépi ciklus

szemafor kezelés

pl megszakíthatatlan, szemafor kezelésére

- RMW /olvasás-módosítás-írás/ read-modify write.

most u. erre a címre írok

- szekvenciális átvitel /blokk vagy burst/ ← ha a cím fázist csak 1-szer adom ki, mint kezdőcímet, utána csak ütemezve adom

- Olvasás-írás-után/read-after-write/ pl írás helyességének ellenőrzésére

az adatot. Ehhez az kell, hogy az az eszköz, akinek adatot adok, az önmagán belül léptesse a címet!

SCSI nem ilyen

Az előző példáknál az átvitel teljes ideje alatt fennmarad a forrás-nyelő kapcsolat /áramkörkapcsolt üzemmód/. Nagy hozzáférési idők esetén ez hátrány

→ logikailag felbontják a kapcsolatot
Üzenetkapcsolt üzemmód /pl MB-II. Message passing protocoll támogatására MP társprocesszor/ → ment bonyolult a protokoll
multibusz II.

Master kéri az átvitelt egy üzenet kiadásával /tartalmazza a cél és a forrás címét, üzenet típusát, és max 28 byte adatot/, ezután elengedi a sínt.

4 byte-os header
1 byte címek
—"— cím
—"— típus
—"— egyéb felhasználás
+28 byte adat

Amikor a slave képes az átvitelre masterként kezdeményezi az adatátvitelt

pl

1, CPU_n adat blokkot kér j. I/O vezérlőtől egy megszakítás üzenet kiadásával, amely tartalmazza az adatblokk azonosításához szükséges paramétereket
(j.-edik; ez a típus; (*) 68. o.)

2. I/O_j összeállítja az adatblokkot és válaszol egy memória puffer kérés üzenettel CPU_n-nek
(ez is váratlan üzenet)
(*) 2 üzenetcsoport — váratlan → ezért van elején, h ne utánra vissza az üzenetet a másik egység; — előkészített; csak 28 byte adat az

3. CPU_n kiadja az üzenetet, hogy a puffer rendelkezésre áll a local memóriájában

4 I/O_j 32-byte-s üzenet csomagokban küldi az adatokat // 2 csomag között holtidő, ekkor más használja a sínt

Az üzenetek között más egységek használhatják a sínt

Adatátvitelnél figyelembe kell venni bizonyos időzítési szabályokat a forrás-nyelő együttműködésének biztosításához

Sinidőzítési protokollok

2 fő { (1) - Szinkron → sínen levő órajelhez szinkronizáltan viszik át adatot
{ (2) - Aszinkron → nincs órajel

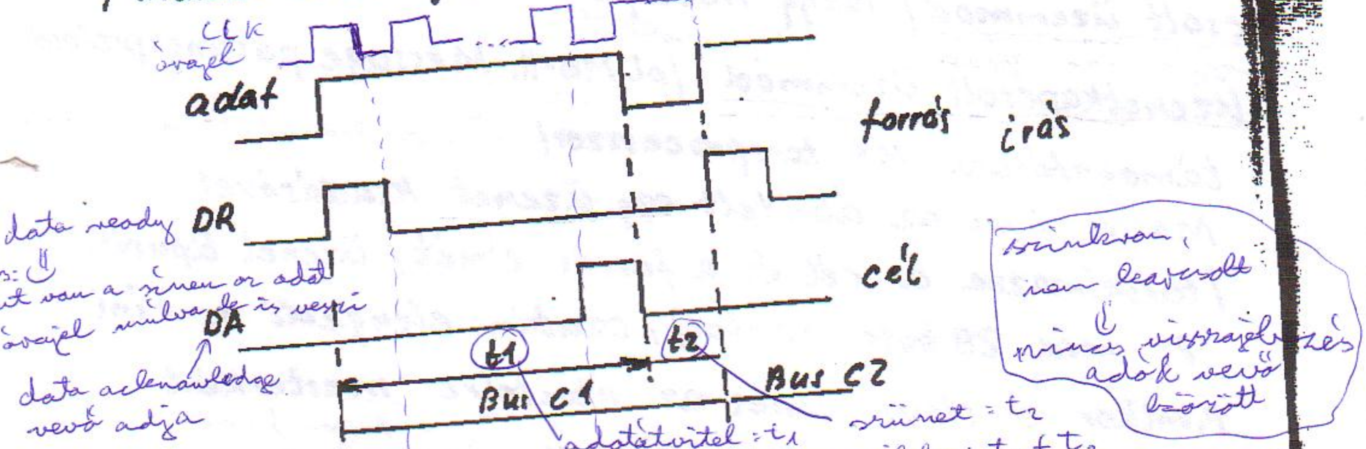
— kapcsolt / reteszelt = interlocked /
— félig kapcsolt, / implicit események/ ... számától függően

vegyes (3) - Semi szinkron — nem kapcsolt

ez NEM CPU órajel!!...?

① Szinkron időzítés

/ minden esemény fix időpontban történik (órajel) /

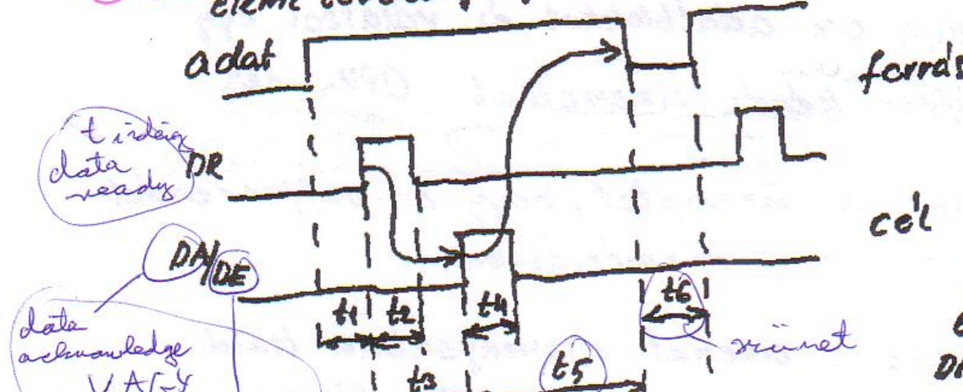


elméletileg a leggyorsabb
de leglassúbb elem szabja meg a sebességet
nincs visszajelzés az adó-vevő között

igény szerint nincs időzítés ⇑ tervezéskor a leggyorsabbhoz szabok, de az újításokat nem tudom követni

② Aszinkron időzítés / nem fix időponthoz kötődnek elemi tevékenységek / események /

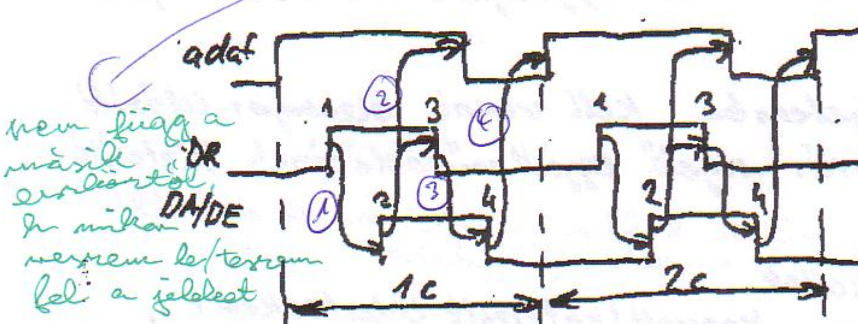
t_3 nyelőtől
t_5 forrástól függ



probléma: t_2, t_4 fix
pl gyors forrás (kis t_5)
elindul a következő adat DA vége előtt. → megoldás

nem reteszelt / kapcsolt, n. interlocked /

t_5 idő múlva a forrás leveszi a szintet



nem függ a másik eszköztől, ha mikor veszem le/teszem fel a jelet

⇓

retesztelt aszinkron adatátvitel

forrás	nyelő
1 adat ki	2 adat vétel
3 jelzés ①	4. felismerés adat elvétel
6, felismerés	5, elvétel nyugtázása ②
7, inf levétele	
8 jelzés, hogy levéve ③	9 felismerés
	10. befejezés nyugtázás ④

NYUGTÁZÁS

t_2: túl hosszú, olyan mintha új adat lenne ("új" tevékenység)
t_4: –‖– , azt hiszi ...
túl gyors forrás / nyelő esetén hiba!

} hibalehetőségek időzítési elcsúszások

④ 37.

(4) hibátlan állapot:
t4 csak akkor mehet le, ha már DR lement

félig kapcsolt protokoll: bináris jelenél kapcsolat van a 2 eszköz közötti időzítésben
az időzítés a busz óvajel egész számú többszöröse

teljesen vezetett adatátvitel:
minden jel között kapcsolat van

⌜ ⌟ ⇒ kiválasztott számok:
közöttük kapcsolat van, hogy ne legyen időzítés miatti hiba/ elakadás

forrás – nyelő kapcsolat

akció – reakció
csak akkor léphetek tovább, ha ok.
A tevékenység kapcsolódik a megfelelő másik oldali tevékenységhez

ábra: kiválasztott élek: nyugtázás
együttműködés

vitlán használt, csak spec. esetekben

A sinvezérlési jog eldöntésének mechanizmusa

I – **Statikus** Előre időzített módon, a kérésektől függetlenül adjuk oda a mastereknek a sínt. Időszeletek, a maga időszeletében akkor is megkapja a master, ha nem igényli

Valamennyi lehetséges master egy előre meghatározott úton adja át a vezérlést

pl M1 ... M4 master. T1-ben M1, M2-T2-ben M3-T3, T4-ben, M4-T5-T6-ban vezérli a buszt, függetlenül az igényektől. => szinkron protokollhoz használható

(Nem igények szerint osztok DE pontosan tudom melyik master mikor kapja meg a sínt ⇓ időkritikus rendszereknél ez jó)

– Túl merev, még leginkább szinkron protocollnál lehet használni

– egyszerű hardver, garantált busz áteresztő kapacitás mindegyik master számára

II – **Dinamikus** A jelentkező igények & milyen algoritmus alapján dönt a sínvezérlésről

lehetővé teszi a busz vezérlési jogának igények szerinti átadását

Az arbitrációs mechanizmus busz megszerzési és busz elengedési stratégiákat használ

I. – **Busz megszerzési stratégiák** /Bus allocation policies/

1. – prioritásos /priority based/
   – fix prioritás szerint dönt az egyidejű kérésekről /pl I/O alrendszer/
   (egyszerre kérők közül a legmagasabb prioritású kapja meg a sínt)
2. – Egyenlő esélyű /Fairness/
   a masterek prioritása egyenlő, ez garantálja hogy minden master igénye kiszolgálásra kerül mielőtt a jelentkezők (az összes vele együtt jelentkezett) közül valaki másodszor is vezérlési jogot kap. ⇝ pl.: forgó prioritásos megvalósítás kezelés
   Ez megelőzi a kiéheztetést.
   (dinamikus feladat-hozzárendelésű procik-nál használják, ahol V proci egyforma)
3. – Kombinált: az előzőek együttes használata
   /pl I/O prior. több proc. fairness/

II – Busz elengedési stratégiák /Bus deallocation policy/
dealloc*a*tion policy

(1) – elengedés kéréskor /release on request/ *(upon)*

A busz master mindaddig magánál tartja a vezérlést amíg más nem kéri /pl CPU-DMA/
– újabb igénynél gyors lehet

) nincs automatikus elengedés. Jó, ha a masterek között prioritás van pl.: CPU – DMA-vez.

(2) – elengedés ha kész /release when done/

ha befejezi az átvitelt elengedi a vezérlést
– mindig újra kell kérni
(ez gond lehet, mr. az elengedés is időbe telik)

a CPU csak akkor adja át a vezérlést, ha a DMA vez. kéri

(3) – befejezés előtti elengedés /Pre emption/

magasabb prioritású kérés esetén elengedi a vezérlést mielőtt az összes adatot átvitte volna
/nagy adatblokkok átvitelénél lehet szükség rá/
pl.: burst adatátvitelnél ez jó

a masternek tudni kell hol hagyta abba az átvitelt, majd újra vezérlési jogot kér

Busz arbitráció hardver mechanizmusai

(I) – Centralizált /központosított/ → *van egy kártya, ez az arbiter*

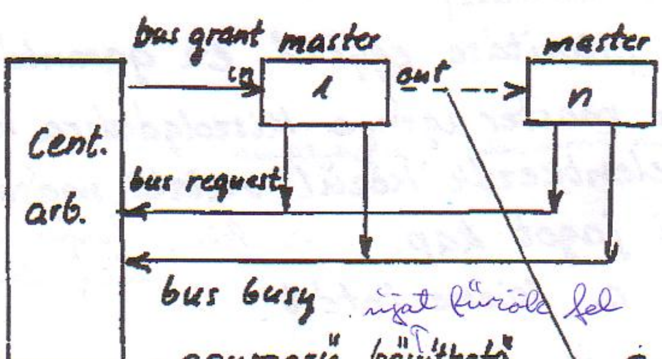
(II) – decenralizált /elosztott/ → *valamennyi master tartalmaz egy áramkörészt, ami a többivel együtt (többi masterrel) képezi az arbitert*

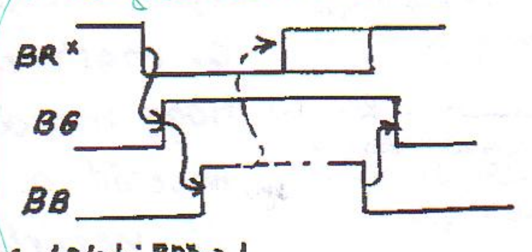
I. Centralizált

arb. hardver egy helyen
a masterek kérik a buszvezérlés jogát, az arbiter dönt és nyugtáz./visszajelzi a jog odaítélését

(A) közös kérés – felfűzött válasz

Ha 1 master kiesik, megszakad a lánc





pl 1,2 kéri BR* → L
CA → BG → 1 → BB → H vége BB → L
→ BR* H out H

– egyszerű, bővíthető
– prioritás merev, döntés lassú lehet. meghibásodásra

felfűzött válasz

1 db kérő vonal, open collectoros, ∀ master ehhez csatlakozik + 1 db foglalt vonal = közös kérés

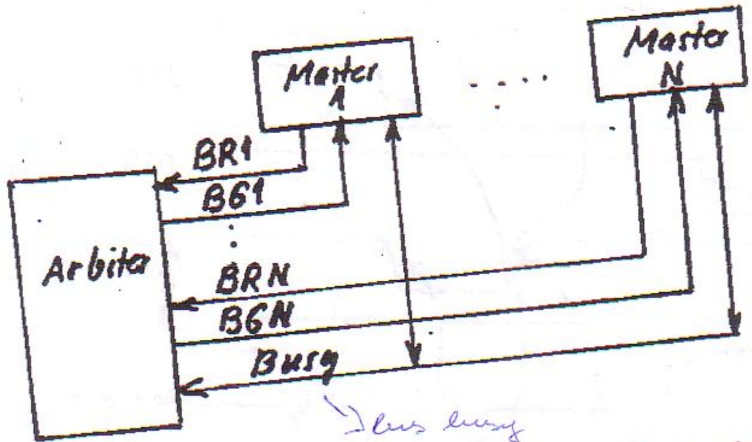
(B) felfűzött válasz: mindenkihez van acknowledge 1 db vonalon, az arbiter a legközelebbinek adja. Ha ennek a masternek kell nem adja tovább, ha nem, a kimeneten továbbadja. Prioritás = sorrend: merev, kötött

2 db előző módszert kombinálom
"Ⓐ + Ⓑ => Ⓒ"

M6

Ⓑ Független kérés – független válasz

- centralizált arbiter
- ∀ masternek külön kérő vonal
- ∀ masternek külön grant jel
- busy vonal közös marad



bus busy

Valamennyi master rendelkezik külön kérés- és válasz vonallal a sín használatát a Bus Busy jel jelzi

előny: - alkalmazható bármilyen megszerzési stratégia (előző merev, kötött volt)
- gyors működés → nincs lánc, terjedő jel
- hibára kevésbbé érzékeny

hátrány: - sok vonal kell a megvalósításhoz
- nem bővíthető

(hibás master kiesik és kész)

Ⓒ Kombinált – mechanizmus

az "előző" két módszer kombinációja /pl. VME rendszer/

Ⓐ = φ akció!!!

allocation:

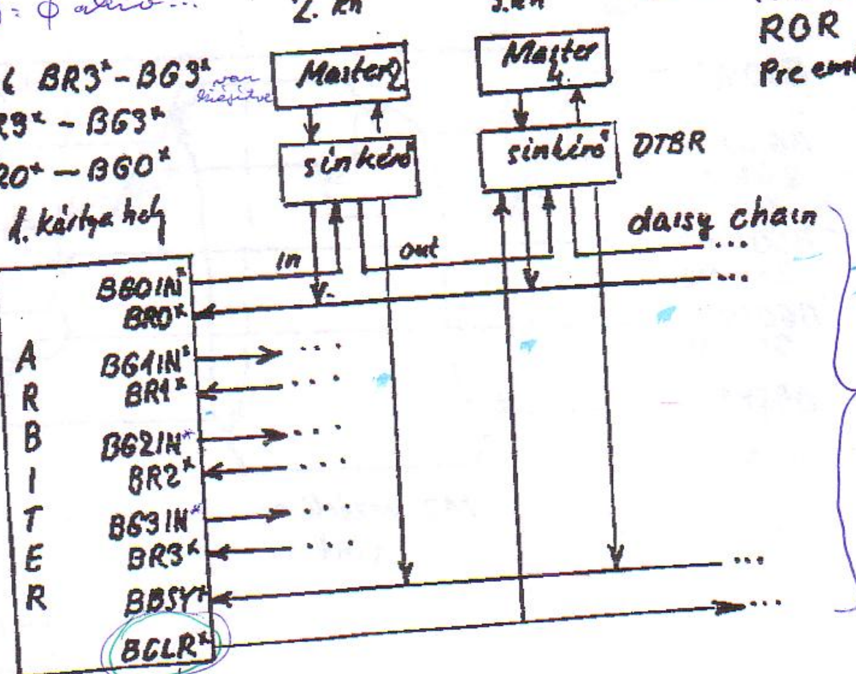
egyszintű: csak BR3* – BG3* van kiépítve

prioritásos: BR3* – BG3*
BR0* – BG0*

Round-robin 1. kártya hely

forgó prioritás

deallocation:
RWD
ROR — BCLR*
Preemtion / kell



∀ szinthez külön kérő és elengedő mechanizmus van

ha kihúzok egy kártyát, át kell kötni az in / out jeleket ne szakadjon meg a sín, lánc: üres kártyahelyre átkötést, "áthidaló" kártyát kell tenni

1 szinten belül : Ⓐ } megoldás
2 szint között : Ⓑ }

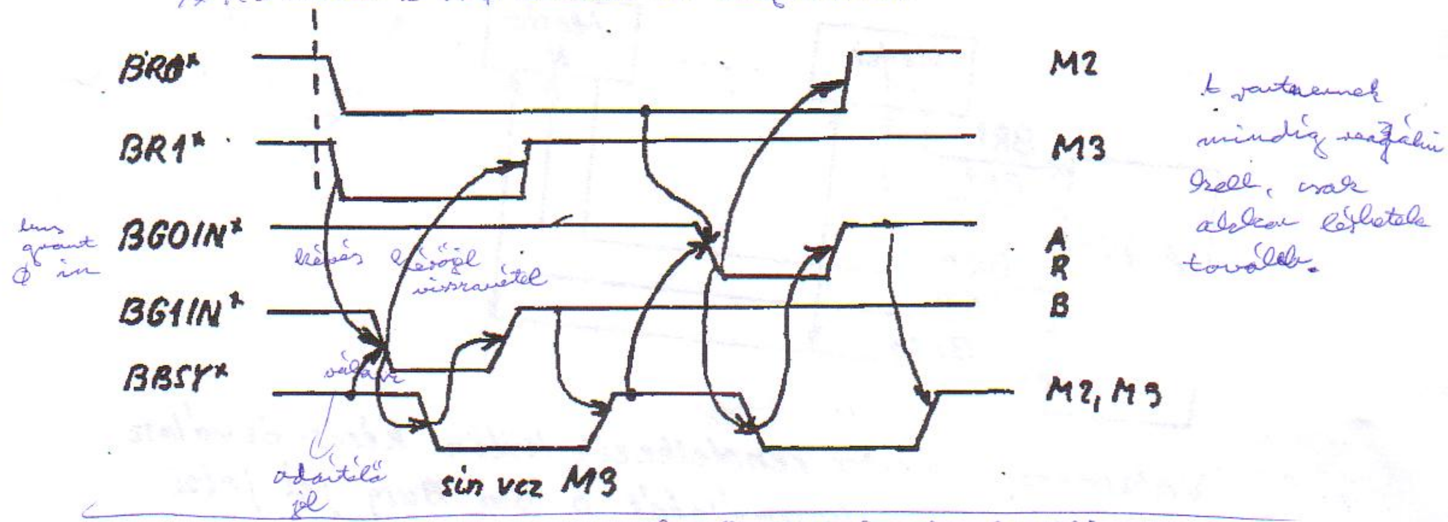
csak erdekbe jöttelenő bekötni

bus clear jel, release on request elengedő stratégiánál számtam van-e kérés másik szinten, ezek erre a jelre elengedik a jogot

M6

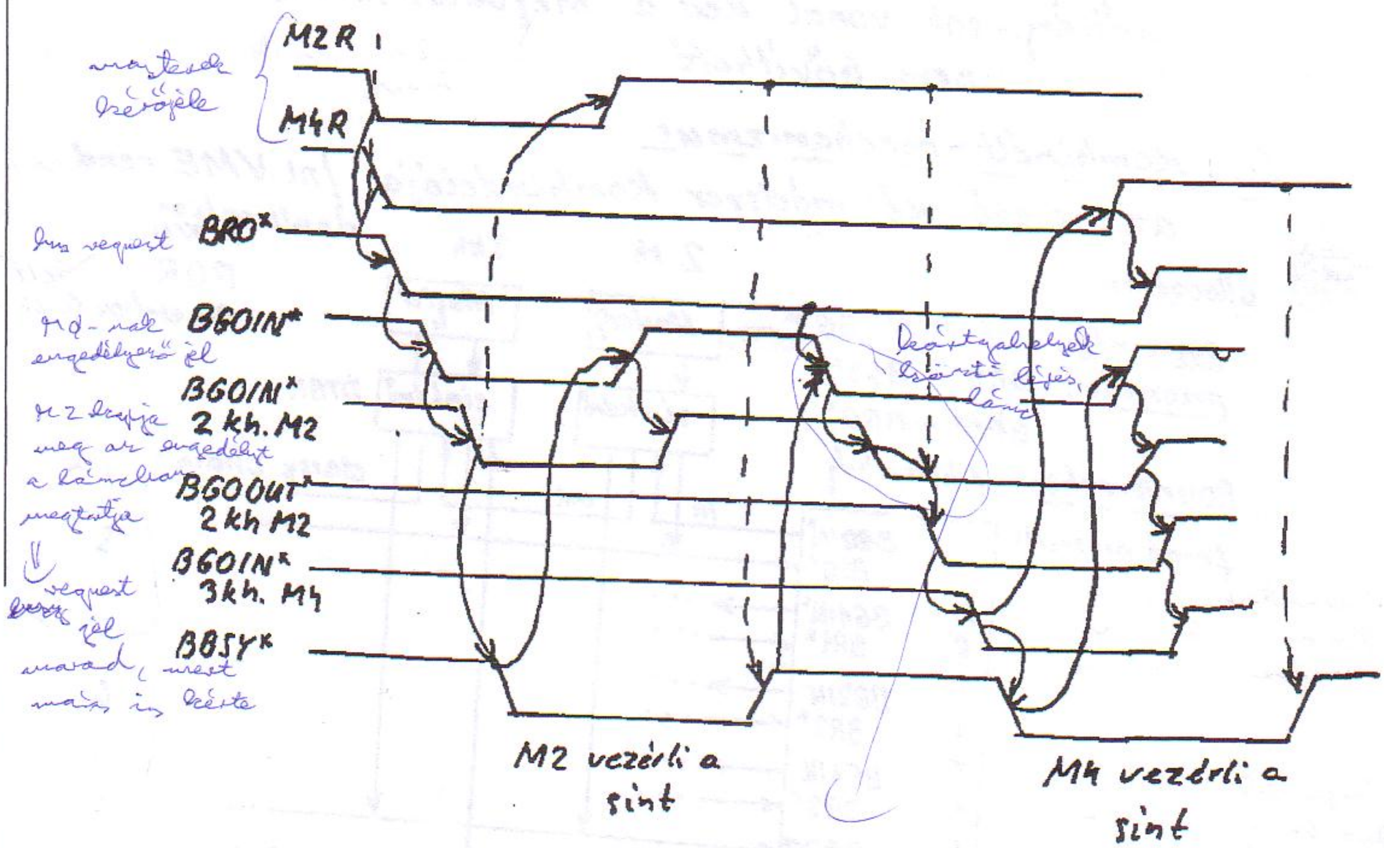
két különböző szinten kérik a vezérlési jogot 10/4

x M2 /BRO*/ M3 /BR1*/ kéri a sinvezérlés jogát
/+ M2 master a BRφ vonalon kér megszakítást/



egy szinten belül, felfűzött elemek közti verseny

pl M2 /BRO*/ M4 /BRO/ kéri a sinvezérlés jogát
2 kh. 3 kh.

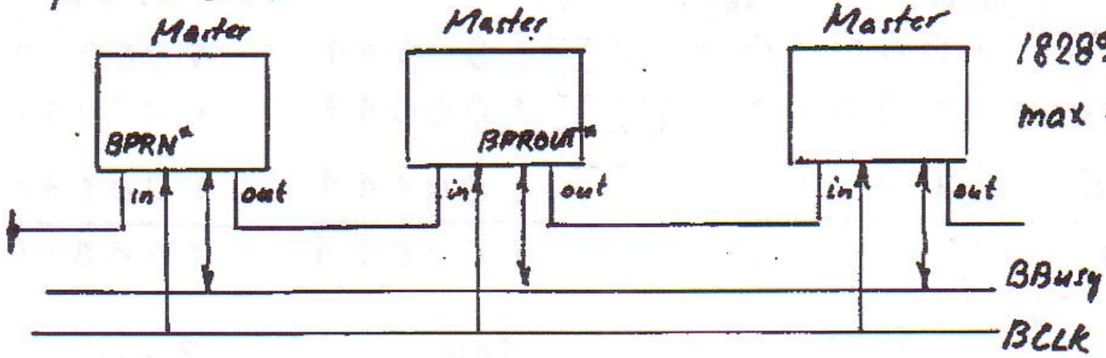


in-en kapja
out-on továbbadja az engedély jelet

II. Decentralizált arbiterek

∀ master tartalmaz egy részarbitert, ezek együtt alkotják az arbitert

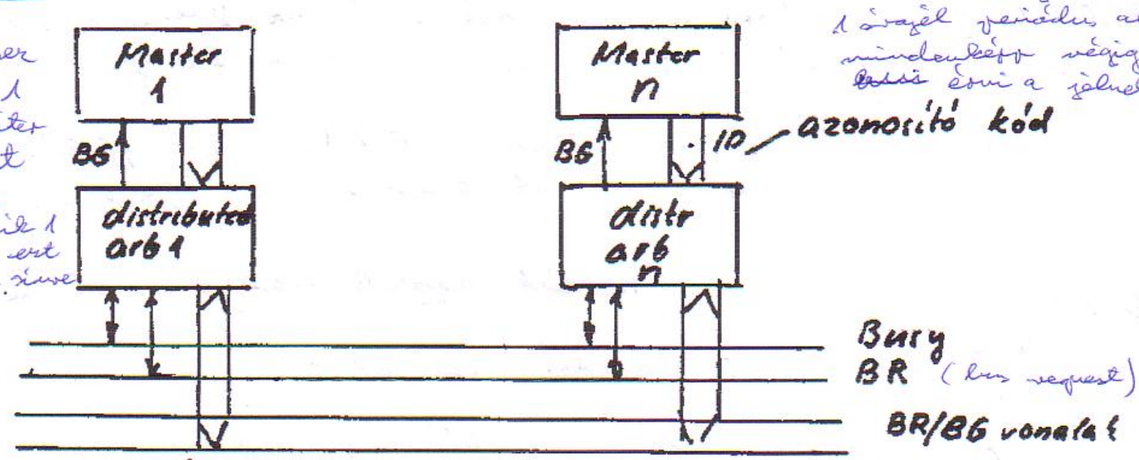
(1) SOROS
pl MULTIBUS-I → max 4 master fűzhető fel, h. ne legyen órajel-elcsúszás



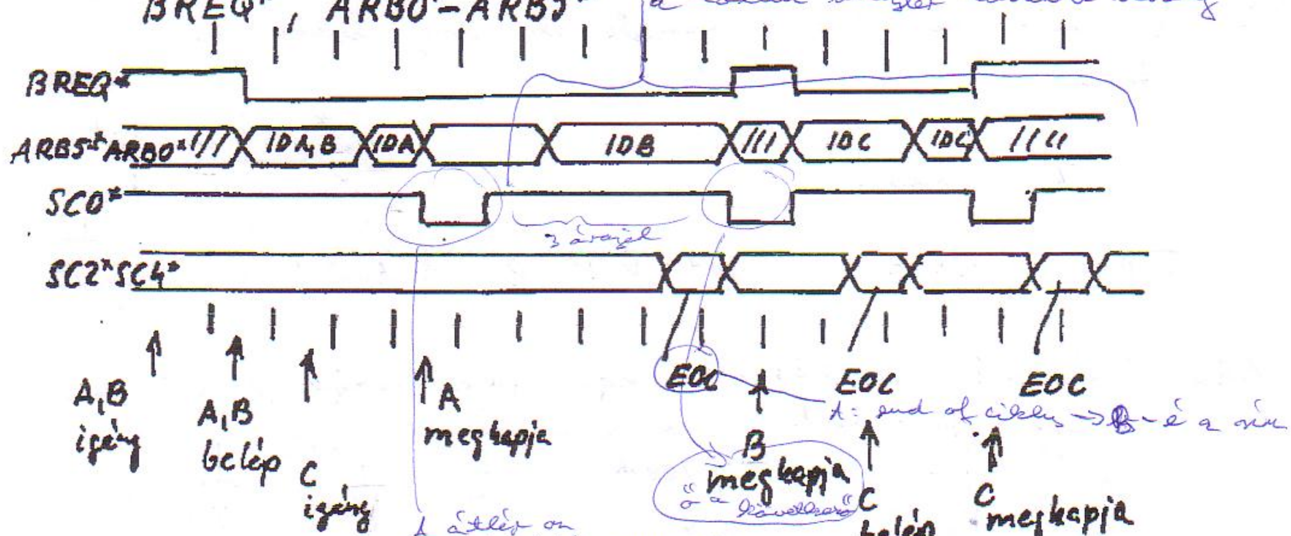
központi órajele adok ki jelet
1 órajel periódus alatt mindenképp végig kell érni a jelnek

(2) párhuzamos

∀ masterhez tartozik 1 rész arbiter hálózat
∀ master rendelkezik 1 kóddal, ezt adják a sínre



pl Multibus II (párhuzamos versenyeztetés) (* max 20 db master*)
BREQ*, ARB0*-ARB5* a többi master tovább verseng



ARB5* magas prioritású kérés

ciklusok: arbitrációs ciklus: itt döntik el a masterek, h. kinél lesz a sín.
↳ csak ekkor léphet be ide új master, ha már nincs kiszolgálatlan kérés

Egyenlő esély módszer

döntési fázis: 3 órajel-ciklus: itt dől el az igény

kivétel ezalól: különleges prioritású kérések, ARB5*-en jelez (84. o)

∀ master csak a saját kódját és a sínen lévőt látja

3 master vár versenyzési jogra: A, B, C

L = 1 " itt
H = 0 "

párhuzamos versenyeztetés módszere

arb ID	1. órajel (ARB5* … ARB0*)	2. órajel: ezzel hasonlítok, az első eltéréstől mindent L szintre teszek a sínen	ez már negált, most ezt hasonlítják a saját kódjukkal ©
A	100010	100011	100010
B	100011	100011	100011
C	100100	100111	100111
sín	100000	100011	100010
	ez lesz az első kód	új eredő kód	új eredő kód
	Start	1 per	2 per, 3 órajel

A master kódja: ennek negáltja

(*) ha valamely master → ARB5*-on jelez
különleges prioritási kérések beléphetnek a folyó arbitrációba, a következőnél (kb 3 órajel után) ő kapja a vezérlést mindenképp

ARB0–5 : 32 kombináció lehetséges, DE ebből →
csak 20-at használnak: Intel által kiosztott

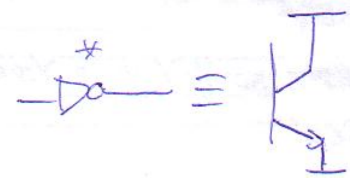
© melyikük által kiírt kód egyezik meg a sínen lévővel

A –
B – megegyezik, eredeti kódját teszi a sínre

C – nem egyezik, a 2. fázisban kialakított kódot teszi a sínre

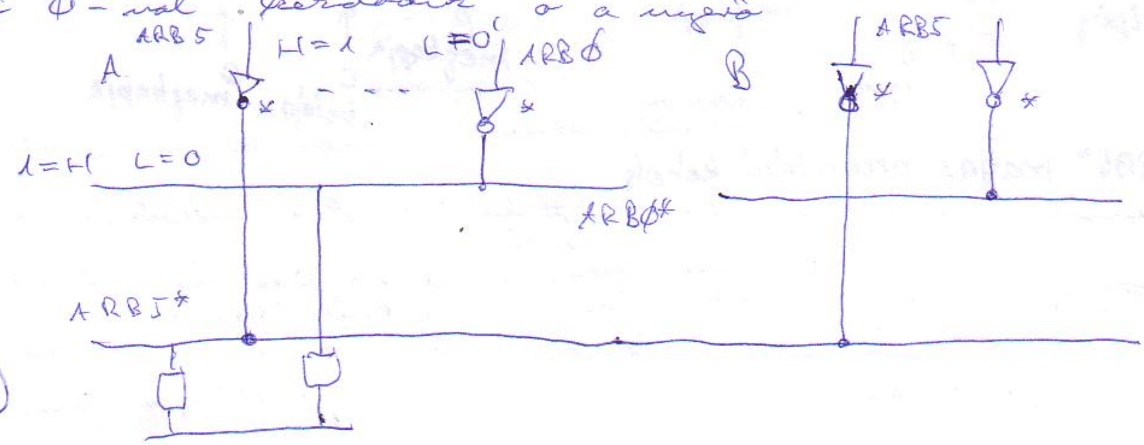
3. ütemre 1 db masternek fog megegyezni a kialakuló kóddal, ő kapja a sínt

A a nyerő



ZH-n ezt a 3 lépést kell bemutatni + a nyerőt

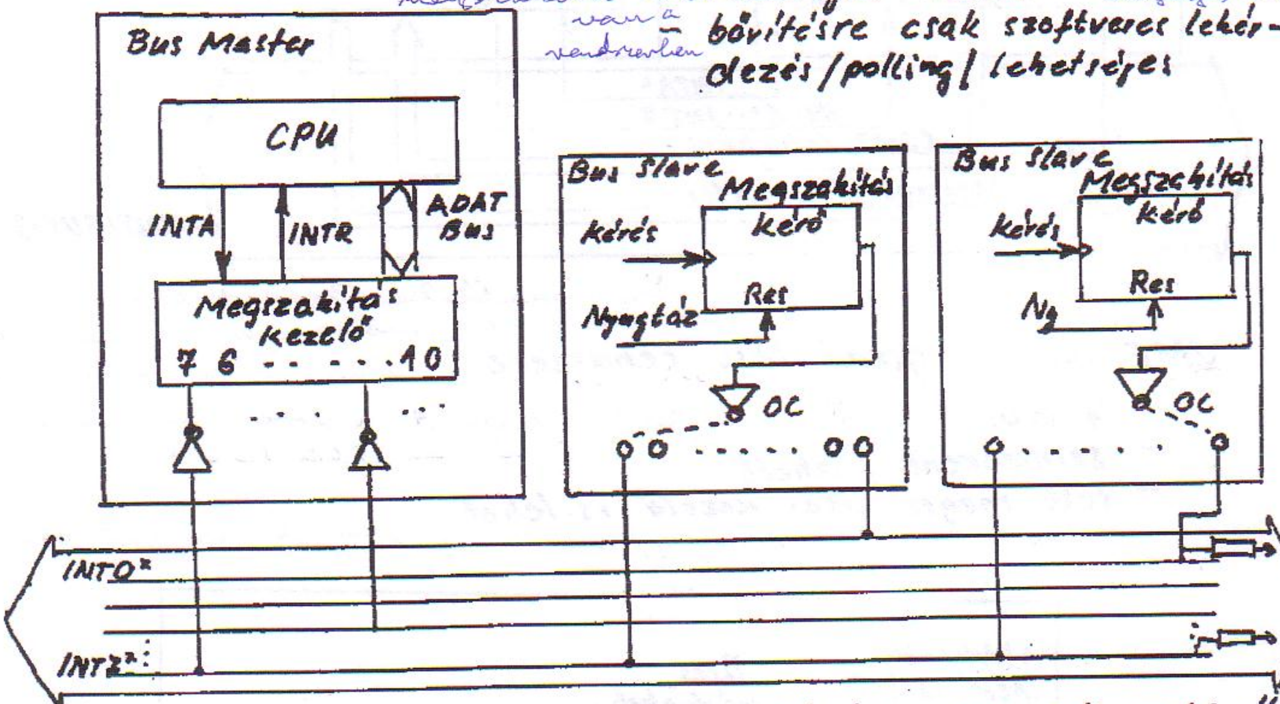
Ha kívül jelentkezik egy spec órajel ARB5-ön, ő Φ-val kezdődik, ő a nyerő



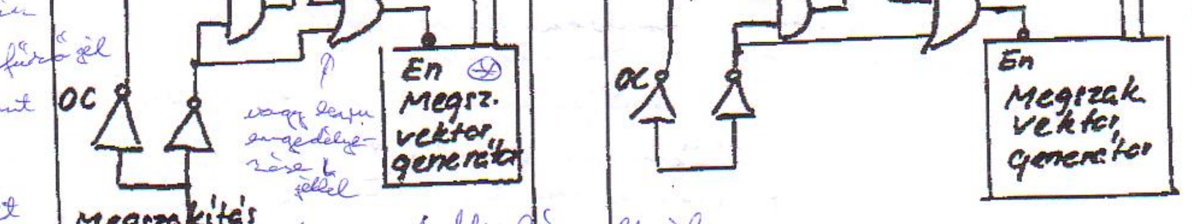
Megszakítás kezelés

- A sinnek tartalmaznia kell a megszakítás Kérés-kezelés lebonyolításához szükséges jeleket
  + kell egy protokoll → 8259 (8085, 8086) - tal 2 mód
- A sin csak megszakításkérő jelet biztosít (Multibus I) /nem bus vektoros/
  vektoros: nem a rutin kezdőcímét adó címet adja fel, hanem egy 8 bites vektort, ezen a címen van a rutin tényleges kezdőcíme
  csak 1 db megszakítás kezelő van a rendszerben
  - Kezelő csak a Master kártyán lehet
  - bővítésre csak szoftveres lekérdezés /polling/ lehetséges

pl



- A sinen történik a megszakításhoz rendelt „vektor" továbbítása is /Bus vektoros megszakítási séma/
- láncolásos /daisy chain/ bővítés
  ahova nem teszek kártyát, lezáró kártya kell a lánc miatt (interrupt request)



IACK*: interrupt acknowledge

PCI in, PCO out: felfűző jel

vagy lehet engedélyezése 1. jellel

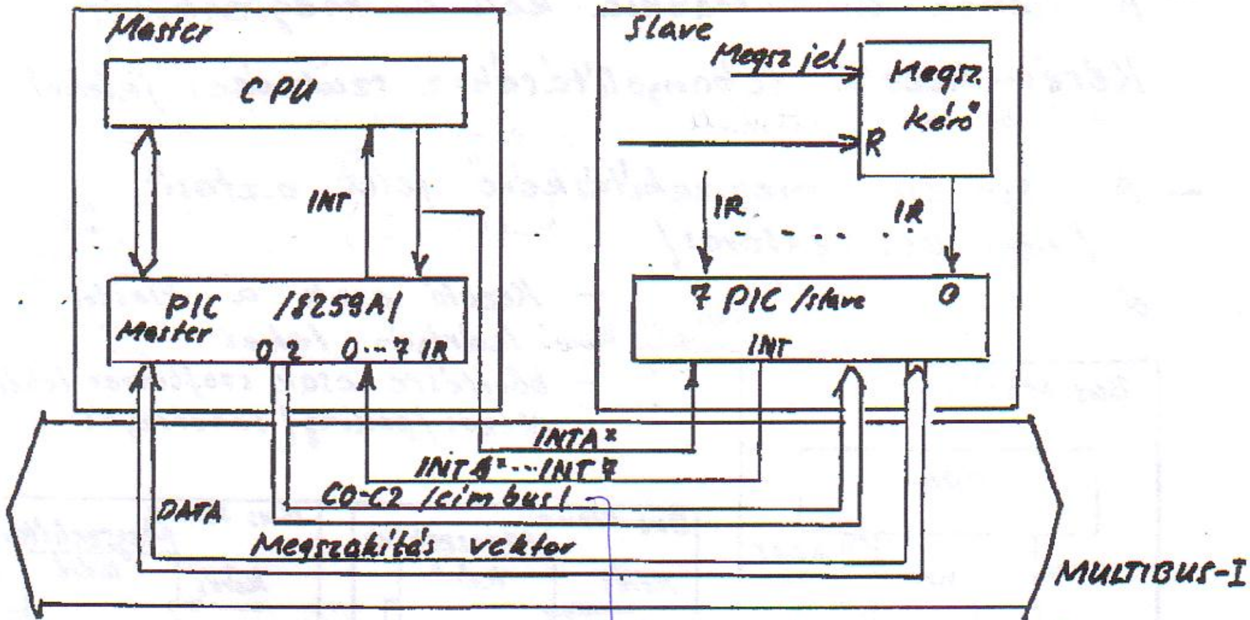
ha saját megszak. kérésem van, nem adom tovább a PCI-n érkező engedélyező jelet; ha nem ő kért PCO-n továbbadta a jelet

van 1 db láncolt jel
2-féle felhasználás: (1) kérés felfűzött válasz; (2) prioritásmeghatározás

⊕ 8 db tri-state meghajtó fix értékekre állítva ⇒ címfeladás

pl **MULTIBUS-I bővítés**

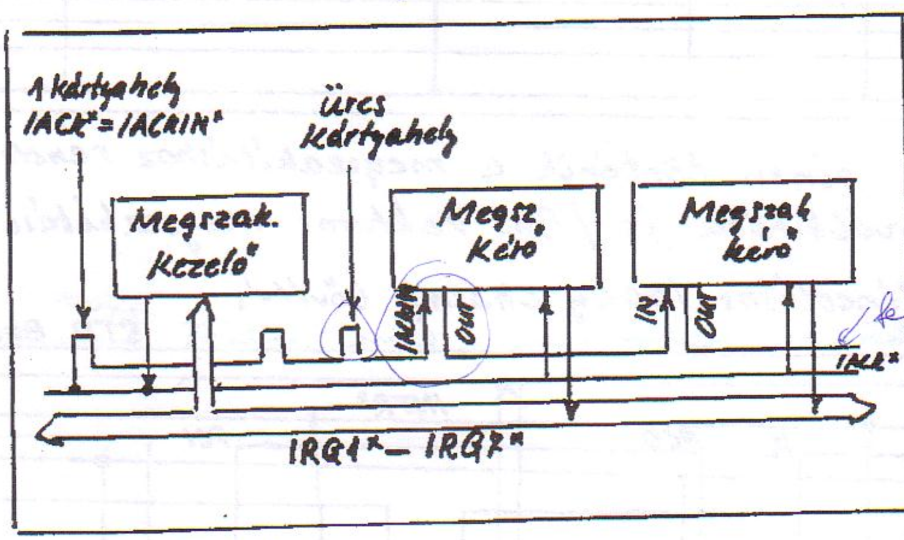
15



(címvezetékeken kapták a korábbi vonalak jeleit => nem kellett plusz jeleket a sínre tenni)

VME bus megszakítási rendszere

- 7 szint (független szintek) /* 8. szint a processzor szintje, itt nem lehet kérés */
- szintenként fűzhető
- több megszakítás kezelő is lehet
  (1 megszak. kezelő egyszerre csak 1 masterhez tartozhat)

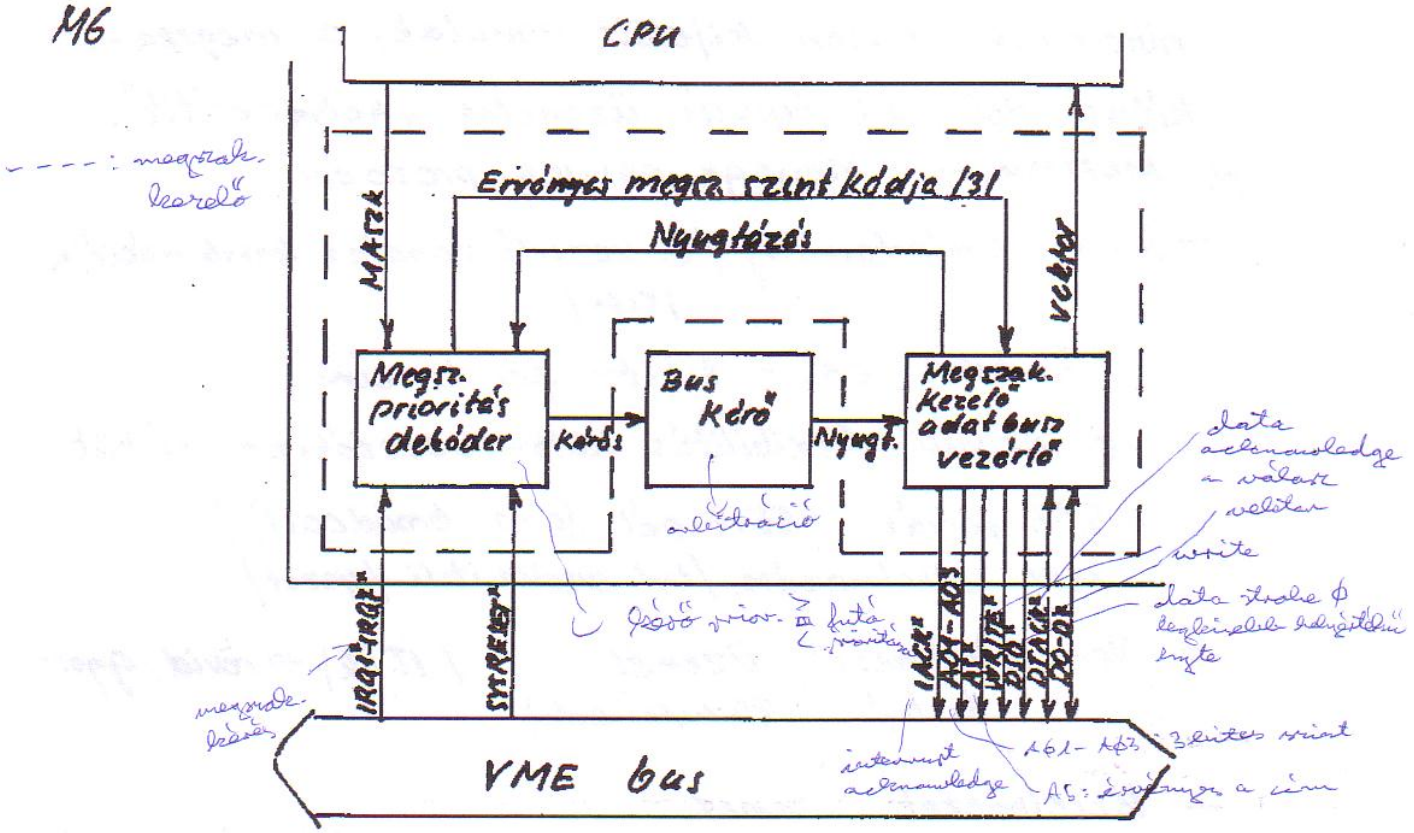


összesen 1 felfűző jel van
csak az azonos sorban lévő kérők játszanak szerepet a megszakítás-kérésben.
A címbusz alsó 3 bitjén van a szint kódja

1 kértem az engedélyt
2 ezen a szinten kértem
3. IACK* aktív
} enyém a válasz, a megszakítás vektort én adhatom fel.
Egyébként nem.

pl VME megszakítás kezelő modul

M6



Csak az a megszakítás kérő adhatja "fel" a vektorát amelyre teljesül:

- a megszakító IT kérést generált
- az elfogadott IT szint megegyezik a generált IT szinttel
- a megszakító nyugtázást kapott a láncolaton keresztül /IACKIN"/

A három feltétel teljesülésekor a megszakító DS0 hatására a D0-D7 vonalra hajtja a vektorát és DTACK" jelet generál.

- Több megszakítás kezelő esetén az érvényre jutás sorrendje függ a megszakítás kezelők arbitrációs prioritásától is / a kezelőnek a nyugtázáshoz meg kell kapnia a buszvezérlés jogát is.

– Virtuális megszakítás kezelés

nincsenek külön kijelölt vonalak, a megszakítást definiált típusú üzenetek „helyettesítik".

pl. MULTIBUS-II Message passing protocol

– Külön címtartomány /a vezérlő vonalak kombinációja jelzi/

8 bites cél - 8 bites forrás cím

rendkívüli flexibilitás külön vezetékezés nélkül

/ 256 forrás 255+1 cél ,(a+1 broadcast)/

MPC alkalmazása /lsd adatátviteli fejezet/

– Nem kérelmezett üzenet /IT-k/ ⇒ rövid, gyors

4 byte (+28 byte ada)

– Kérelmezett üzenet

nemkérelmezett üzenetekkel „felépített" adatátviteli kapcsolat nagyobb tömegű adatátvitelre.

VME rendszer (1980---

– 8, 16, 32 bites processzorok alkalmazására

– dupla méretű un. eurocard, 2×96p tüs csatlakozó

– bővítés ⇒ VMX, VMS, I/O alrendszer, /VME eredeti rendszerbus/

VME

– adatátviteli /DTB/

– prioritásos megszakítási vezeték csoportok

– arbitrációs

– utility

funkcionális modulok

– Master

– Slave

– Lokáció monitor

– Busz timer

– Megszakító

M6

Speciális sz.g. architektúrák

12/1

- feldolgozási sebesség növelése
  - áramkörök működési sebességének növelése
  - sz.g. szervezésének módosítása
    /pl cache, stb/
    egy határon túl → már nem Naumann architektúra
  - párhuzamos működési megvalósítása
    nagylépték̈ű /több processzor működik konkurensen/
    Kislépték̈ű /a processzor egyes részei működnek konkurensen/

1, Harvard-architektúra

- „oldja" az egyetlen tároló – sín szűk keresztmetszetét
- ha alkalmazzuk, akkor két cache és két virtuális tárkezelő kell.

2, Gépi utasításon belüli párhuzamosítás

- alapvetően megtartja a SISD szervezést, de mielőtt egy ut. végrehajtása befejeződne elkezdi a soron következő(k) végrehajtását is ⇒ pipeline

pl

I: utasítás lehívó
E: utasítás végrehajtó

I: - lehívja OT-ból (esetleg cache, Queue, -ból) a soron következő utasítást
- részben dekódolja
- elvégzi az operandus címszámítását
- kezdeményezi az oper. beolvasását
- átadja az utasításra vonatkozó részletes információt E-nek

E: végrehajtja az előírt műveletet

Közben I a következő ut. lehívását végzi

E és I eltérő sebessége miatt várakozósort szokás alkalmazni

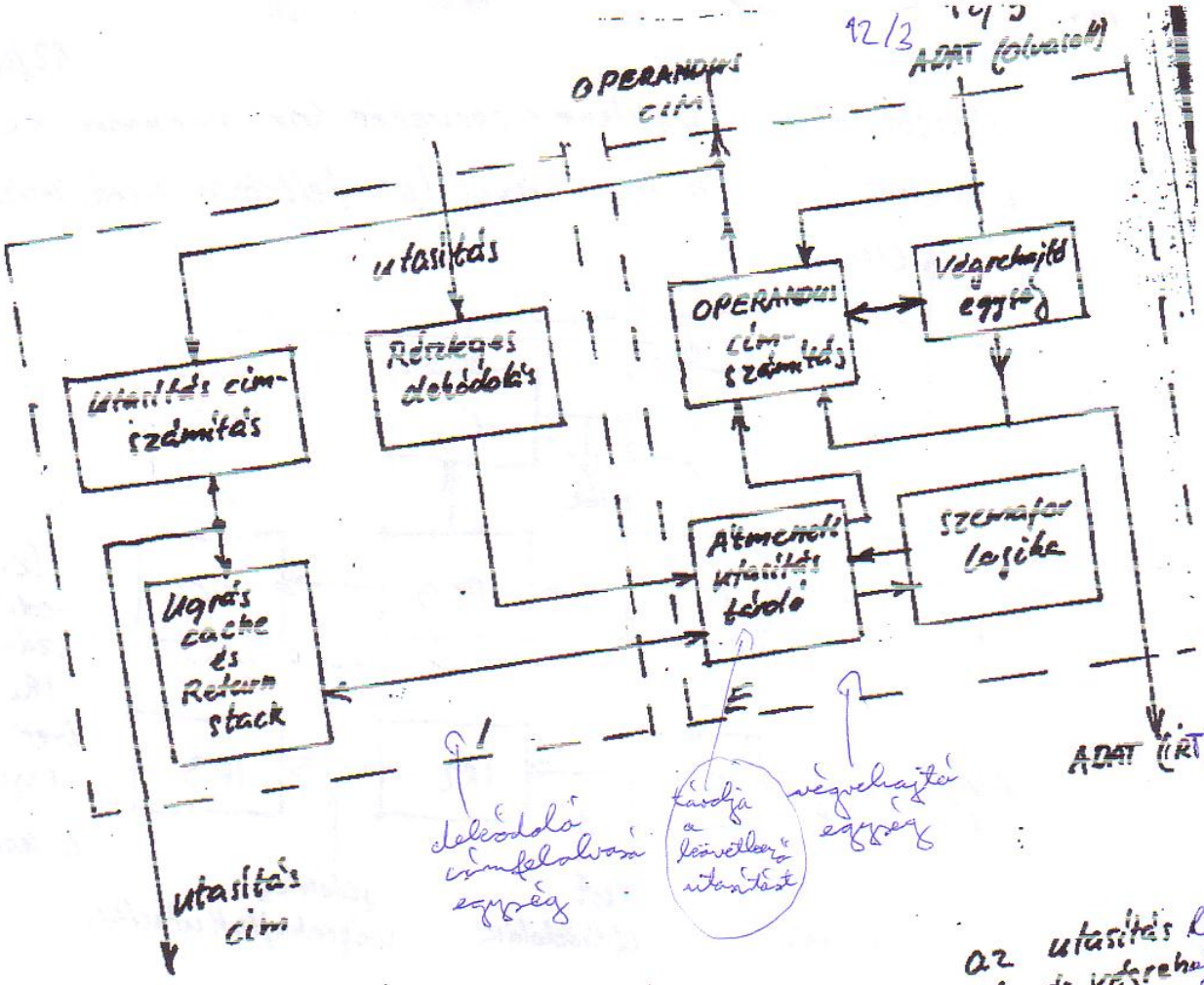
Utasítás egymásra hatás problémája

- feldolgozási (két utasítás ugyanazt a feldolgozó erőforrást igényli) — IT-kezelés!
- procedurális (pl feltételes ugró utasítás) — Szemafor!
- adat (utasítás egyik operandusa az előző ut. eredménye)

⇒ a végrehajtandó utasítás környezetében nem szabad utasítás módosítást programozni

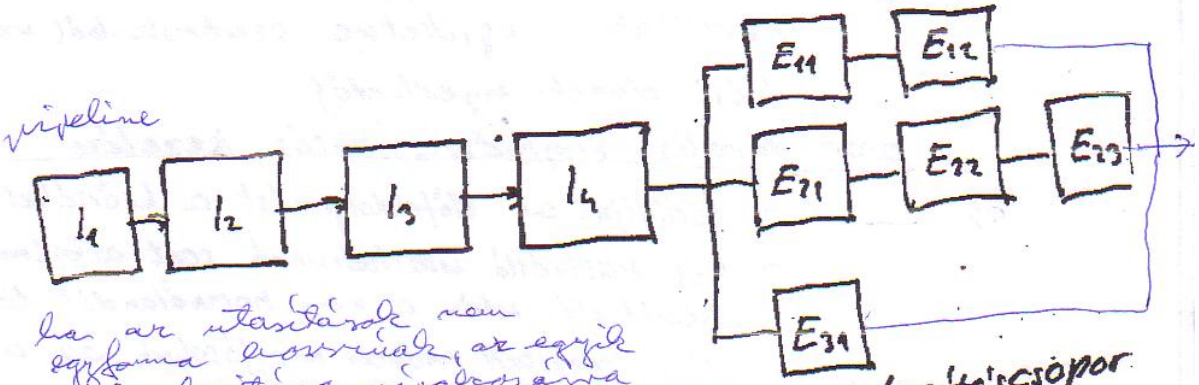
Egy műveletvégzés alatt levő utasítás eredményétől függ, mit kell a következő utasításnak végrehajtania

Az utasítás lehívása az előző utasítás eredményét



pipeline

az utasítás lehívás és végrehajtás teljes folyamatát kisebb autonóm funkcionális egységek hajtják végre



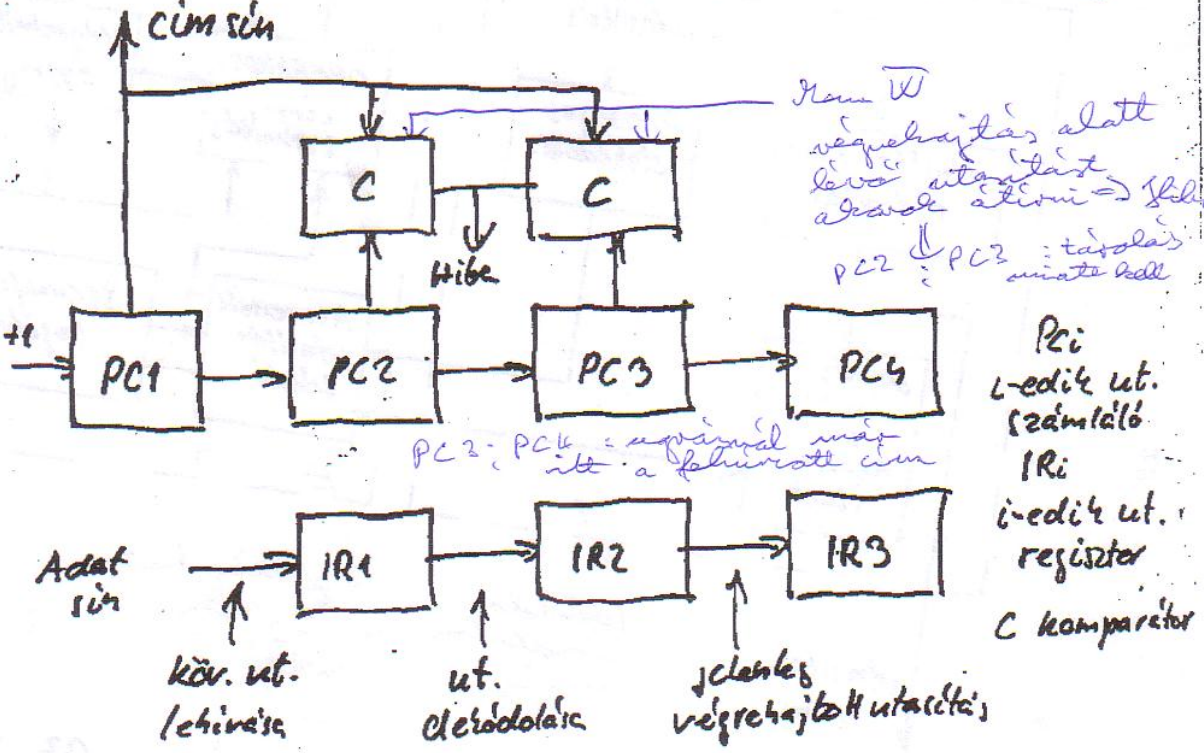
vipeline

ha az utasítások nem egyforma hosszúak, az egyik végrehajtására várakozásra lesz szükség ⇒ adagáram

a végrehajtási fázis lépései különbözőek ⇒ több E lánc (különböző hosszúságú)
bizonyos E tagok átlépése

RISC-nél lesz szabályos az azonos hosszú utasítások ⇒ ott optimálisan kihasználható a pipe. CISC-nél nem

Utasításlehívás pipeline szervezésekor nyomon kell követni az ut-ok célját (pl feltételes ugrás miatt)



PC2, PC3 a már lehívott utasítás átírásának megakadályozására

PC3, PC4 tartalmát felhasználva relatív ugrások valósíthatók meg, illetve szubrutinból való visszatérési címek nyerhetők

procedurális egymásra hatás kezelése

386 → — leállítjuk az előfeldolgozást a kiértékelésig
— két különálló utasításvevő sort alkalmazunk kiértékelés után a nem használandót töröljük
— előre rögzített módon megjósoljuk az utat (pl feltétel nem teljesül)

U86 → Dinamikus a rögzített utat folytatja a proc. de megjelöli az ugrás utáni utasításokat és végrehajtásukat (E) felfüggeszti a kiértékelésig

feltételes elágazást megjelöli, de a NEM ágat tölti be a proc, a pipeline csak az egyik ágat tölti be

Pentium: szintén a NEM ágon megy, de saját maga „becsült" statisztikát, mely alapján a gyakoriabb