412 - 456

pluszs @ freemail.hu

PC HARDVER (2000)

A SZOFTVER

Mikroszámítógép rendszerek

mikroszámítógépek

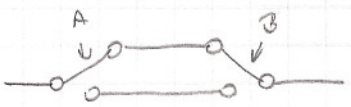
A	B	$\bar{A}$	$\bar{B}$	N VAGY	N ÉS	XOR
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	1	0	0	1	1	0

2 változós: 16 kapcsolat

n változó: nagyon sok, ezért egyszerűsítünk ⇒ konjunktív és diszjunktív alak ⇔ egyszerűsítés

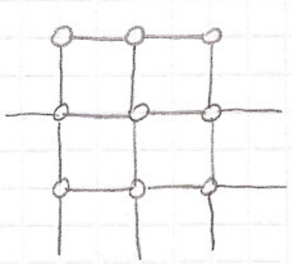
és: villamos ajtók a működése
vagy: autó ajtó és lámpa

XOR:



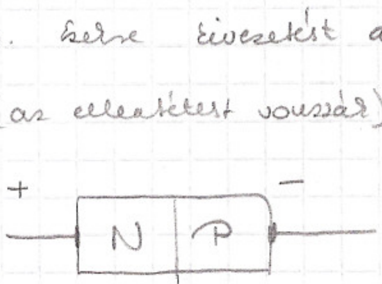
váltókapcsoló

alapanyag: Si
1 kristályos állapotban nem létrehozni
Nagy hő hatására szeresdik
3 vegyértér p típusú
5 vegyértér n típusú ⇒ 1 szabadon van

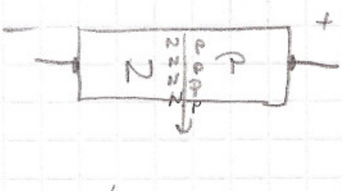


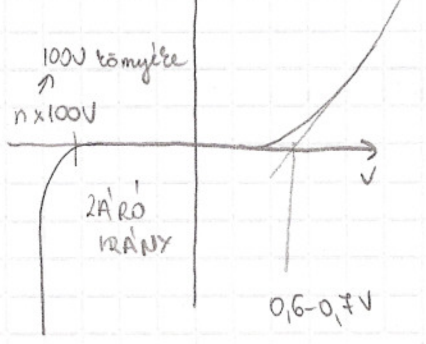
3 vegyértékű szennyezéssel szennyezik, így e- hiány jön létre.
A pólus erejét használják ki.
(5 vegyértékkel is, az akceptor nem fogad el több e⁻-t, egy e- szabadon mozog.)

Hoznak olyan félvezetőt létre, amelynek egyik oldalát n a másikat p típusúval szennyezzük. Ezek kiegészítik egymást, semleges lesz az anyag, és a középen már nincs kiegyenlítődés. Erre kivezetést adva töltésmegosztás alakul ki. (az ellentétest vonzzák)



Az e⁻-ban szegény határréteg ki fog szélesedni.
Ha megcseréljük a polaritást, akkor nem taszítani, hanem vonzani fogja az egymásét, és a határréteg felé fog elmozdulni.





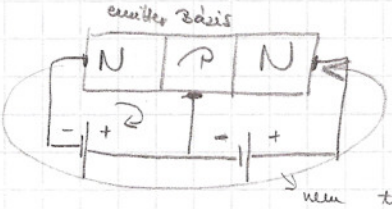
Egy ideig ha feszültséget kapcsolunk rá, nem alakul ki.

a diódákat a záróirány érdekében helyezzük.

- Nyitóirányba rajzos ábra alakul ki.
- Akkor, h. a dióda kinyisson, szükséges alapfeszültség, 0,6-0,7 V alatt nem szokott kinyitni, nem vezet!

- Alakítsunk ki 3 réteget 2 félvezetőből ⇒ TRANZISZTOR

gyakorlatilag nem szimmetrikus, csak elméletileg



Mellékhatás lép fel:

A kollektoron folyó áram csak az emitteren jelenik meg, a bázison nem.

$$I_B + I_C = I_C$$

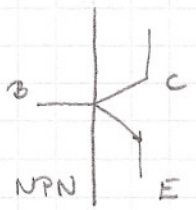
Rájöttek, h a bázisáram arányos a kollektorral

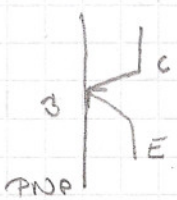
$$I_B \sim I_C$$

$\beta \Rightarrow$ arányossági tényező

$$I_B \cdot \beta = I_C$$

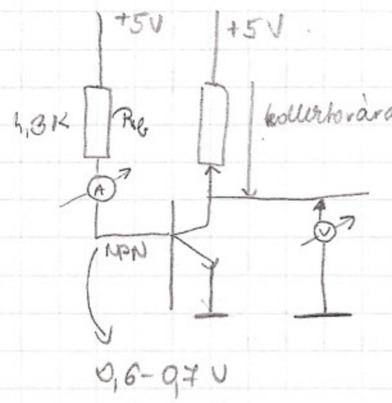
Tranzisztor elektronos kapcsolása





Minden tulajdonság igaz az NPN-re, az a PNP-re is, csak másfele polaritást kell adni

Ohm törvénye $R = \frac{U}{I}$



Méri a bázisáram kialakult értékét.

Az NPN bázisa nyitóirányban van.

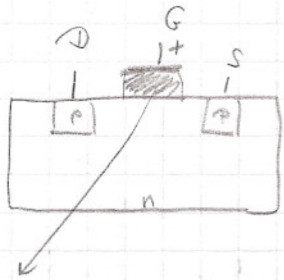
$$I_B = \frac{4{,}3}{R_{be}}$$

$$I_C = \beta \cdot I_B = \frac{100 \cdot 4{,}3}{R_{be}} = 0{,}1\,A$$ (szorzási érték)

100-as β-nál 100 mA fog folyni

Hiába tennénk a bázisba olyan áramot, hogy a kollektoron több áram is folyhatna, Ohm törvénye érvényben marad.

β: lehetőség, hogyha tudna folyni a kollektorban nagyobb áram, folyna, de több nem (kevesebb lehet)



Egy negatív töltésű szennyeződésben 2 pozitív töltésű szigetet hozunk létre, de ez szigetel, nem engedi az e^--ok mozgását.

szigetelőanyag: SiO_2

A SiO_2 tetején van kivezetés gőzölögtetve = GATE (kapu)

DRAIN
SOURCE } ha ..... tennénk közéjük, nem folyik áram.
GATE

+ feszültséget kapcsolunk a GATE-re, akkor magához vonza a + töltéshordozókat, itt csatorna jön létre ⇒ áramot tud vezetni

Hardver II.

A n és a p csatornás egymást komplementerének nevezik

FET n és p csatornás lehet

MOSFET

CMOS Komplementer mos elemeket tartalmaz

nem kell energia, h. fennmaradjon az áram

közök: • Makó Imre: PC Hardver

• PC-t konfigurálása és installálása (Hardver)

• A szoftver — " —

Tankönyv:

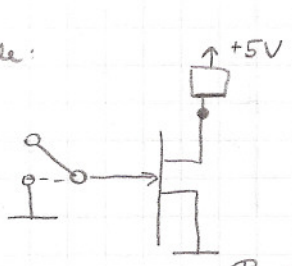
Csernyi László: Mikroszámítógépek
Mikroszámítógép-rendszerek } együtt

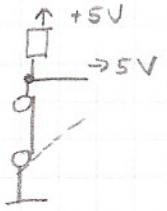
Budai Attila: A mikroszámítógépek rendszertechnikai alapjai

2. előadás

IX.23.

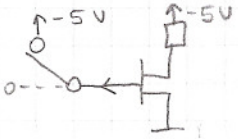
FAT rajzjele:

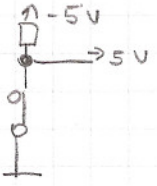




P csatornás FAT rajza

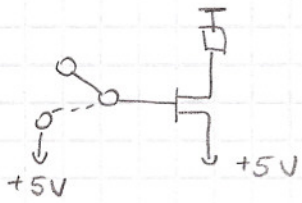
P 5V z(árt)
0V Ny(itott)

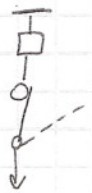


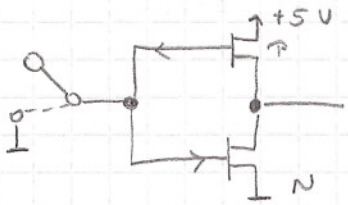


N csatornás

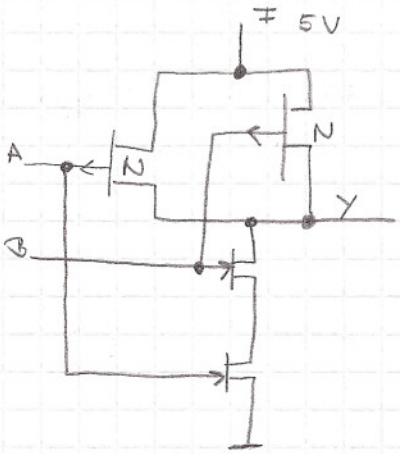
N -5 V Z 0
0 V Ny +5 V

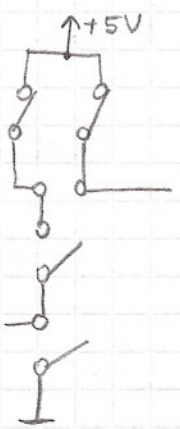






P +5 V Z
0 V Ny



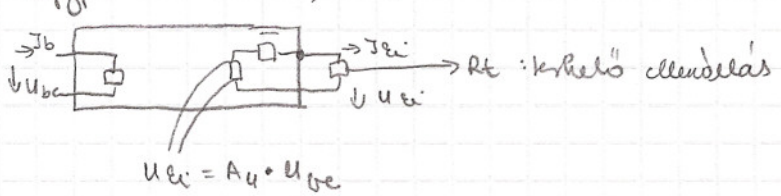


A	B	V
0 V	0 V	5 V
5 V	0 V	5 V
0 V	5 V	5 V
5 V	5 V	0 V

Nand kapu

Analóg kchnika

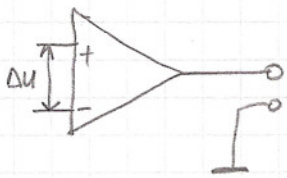
négypólusú (2 be, 2 ki)



$U_{ki} = A_u \cdot U_{be}$

Műveleti erősítő: a feszültséget erősíti fel

$A_u = \frac{U_{ki}}{U_{be}}$ a bemeneti és kimeneti feszültség hányadosa



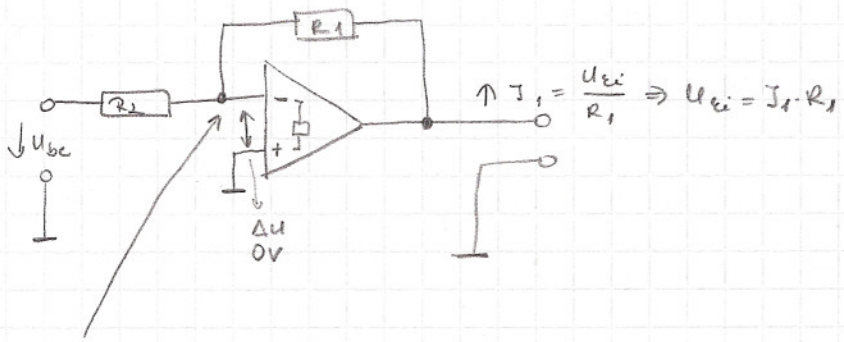
Műveleti erősítők:

Fázisfordító erősítő:

$R_b \approx \infty$

$A_u = \infty$

$U_{be} = I_2 \cdot R_2 \Leftarrow I_2 = \frac{U_{be}}{R_2}$



virtuális föld

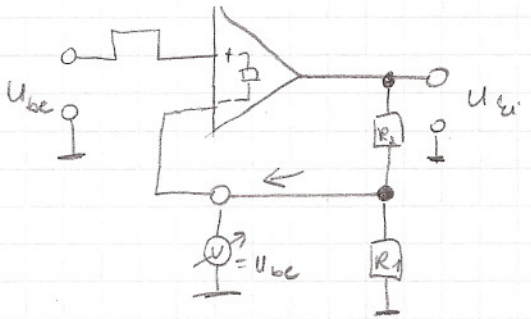
$A = \frac{U_{ki}}{U_{be}} = \frac{I_1 \cdot R_1}{I_2 \cdot R_2} = -\frac{R_1}{R_2}$

Passzív áramkör

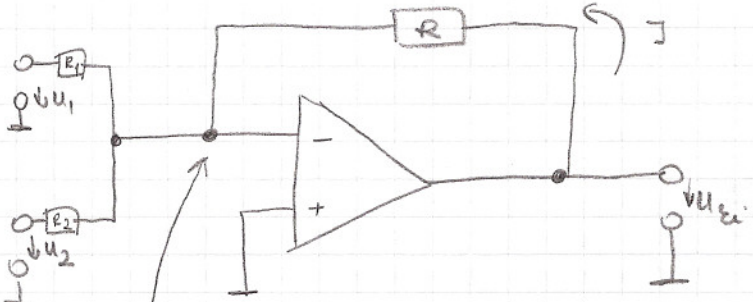
Az áramkörön fázis folyik

Fázisfordító erősítő: csörrenő feszültségre növekvővel fog válaszolni

Fázist nem fordító erősítő



$$A_u = \frac{U_{ki} = J \cdot (R_2 + R_1)}{U_{be} = J \cdot R_1} = \frac{R_2 + R_1}{R_1} = 1 + \frac{R_2}{R_1}$$



Virtuális Föld pont

$$\frac{J_1}{R_1} = \frac{U_1}{R_1}$$

$$J = \frac{U_{ki}}{R} \qquad U_{ki} = J \cdot R$$

$$J_2 = \frac{U_2}{R_2} \Rightarrow U_2 = J_2 \cdot R_2$$

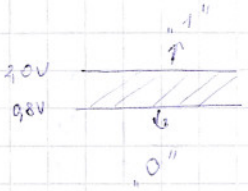
$$J = -(J_1 + J_2)$$

$$U_{ki} = -(J_1 + J_2) \cdot R = -\left(\frac{U_1}{R_1} + \frac{U_2}{R_2}\right) \cdot R$$

$$\downarrow$$

$$-U_1 + U_2 \cdot \frac{R}{R_1}$$

3. előadás



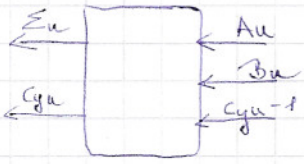
10-es szr. 10^0 ... 10^n

2-es szr. 2^n ... $2^2\ 2^1\ 2^0, 2^{-1}\ 2^{-2}$

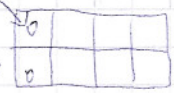
0-255 (1byte) → ezt a számtartományt képes ábrázolni

```
0 0 0 0 1 1 0 1 1 1 0
0 0 0 1 0 0 0 0 0 1 0
0 0 0 1 0 1 1 1 1 1 1
0 0 0 0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 1 0 0
```

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

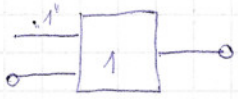


A	B	C_{u-1}	Eu	Cyu (átvitel)
0	0	1	1	0
0	0	0	0	0
0	1	1	0	1
0	1	0	1	0
1	0	1	0	1
1	0	0	1	0
1	1	1	1	1
1	1	0	0	1



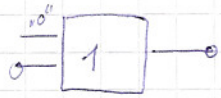
függ

ha standard 0, akkor a Eimenet és a bemenet között nincs különbség.



standard 1 a Eimenet

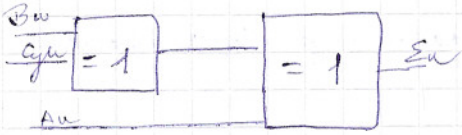
ami a bemenet, az a kimenet

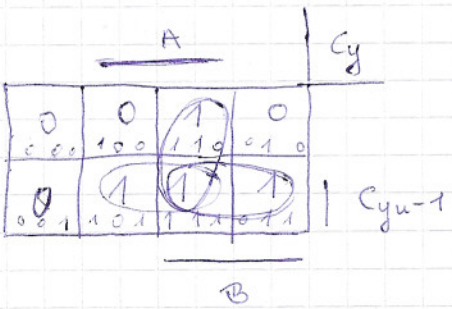


a kimenet a bemenet

0	0
1	1
0	1
1	0

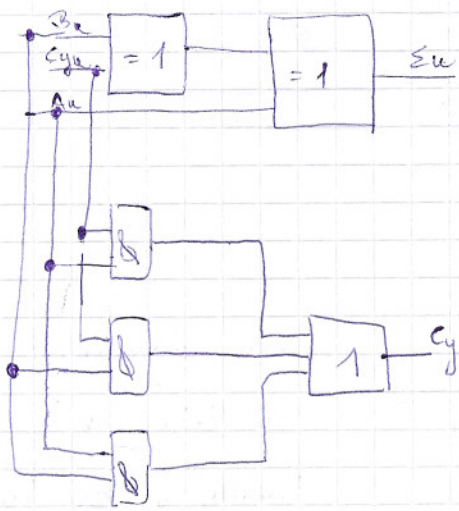
a bemenethez a kimenet a negált lesz



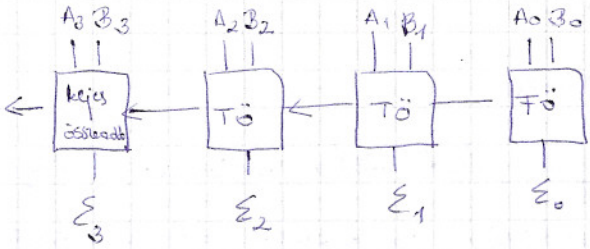


$$\Sigma_n = A_n \oplus B_n \oplus C_{yn-1}$$

$$C_{y\,n} = (A_n \cdot B_n) + (A_n \cdot C_{yn-1}) + (B_1 \cdot C_{yn-1})$$



1 bites teljes összeadó



A_n	B_n	E_n	C_{gy_n}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
		XOR	É.S

- Így működik a CPU-n belüli összeadás logikája és elve.

(0-255)

```
  0 0 0 7
+ mínusz 7 = 9993
1|0 0 0 0  -> ilyen nincs
- 0 0 7 2
```

$A + (-A) = 0$

```
  9 9 9 9
+ 0 0 7 2
  9 9 2 7  <- 9-es komplemens
  0 0 0 1
  9 9 2 8  -> -72
```

10-es komplemens (kiegészítő érték)

```
  1 1 1 1 1 1 1 1
- 0 0 0 1 0 0 0 0
  1 1 1 0 1 1 1 1   -> 1-es komplemens
+ 0 0 0 0 0 0 0 1
  1 1 1 1 0 0 0 0
```

0 -> 127 (128 db)
-1 -> -128 (128 db)
} 256 db

Ez egy nibble

0 0 0 0 0 0 0 0 -> 0

0 1 1 1 1 1 1 1 -> 127

1 1 1 1 1 1 1 1 → -1

1 0 0 0 0 0 0 0 → -128

Olyan az első helyen álló száma, mintha előjelbit lenne. De ha fordítjuk az előjelet, megfordítjuk minden bitjét és hozzáadunk 1-et.

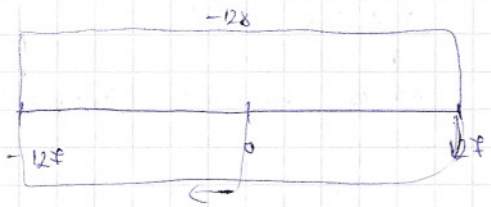
1955 ⇒ $1.955 \cdot 10^3$

1 1 1 1 0 1 0 0 0 1 1

± 1, 1110100011 · $10^{\pm 1010}$ (2^{10})

32	normál pontos
64	dupla pontos
80	KITERJESZTETT pontos

- Lehet ábrázolni úgy is, hogy az abszolút értéket ábrázoljuk és feltartjuk 1 előjelbitet.

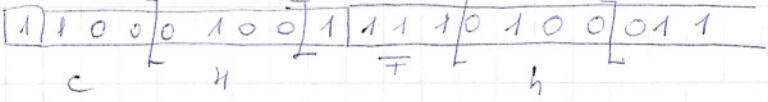


Nincs előjel, hanem mindig a számból 127-tel többet tárolunk (ábrázolunk.)

Hiányzik a -128, de helyette egy értéket (a 0-t speciálisan tudjuk kezelni. Ha a kitevő bitjében 0-val, az speciális érték.

Nem kell ábrázolni, de mindig 1.

955



4. előadás

45,23

$101101,\overline{0011101011100001001}$

↓ végtelen szakaszos tört

Osztjuk 2^5-nel.

$1,01101\ldots \cdot \underbrace{10}_{2}^{\overbrace{101}^{5}} \longrightarrow (+127)$

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

előjel

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

$132 = 127 + 5$

1-es, ami rejtett (mindig 1-es)

42 3A EB 85

16 biten ábrázolás → $2^{16}-1$ a maximum

8 biten ″ $2^8 - 1$ ″

VISSZA:

CAF8B080

1	1	0	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
C		A		F		8		B		0		8		0																	

$\begin{array}{r} 1\,1\,9 \\ -\,1\,2\,7 \\ \hline 2\,2 \end{array}$

Tfh.: ez a szám

11111000, 1011 000010

-248, 6875

Csak akkor tudunk összeadni két lebegőpontos számot, ha azonos a hatványkitevő. (azonosat az alapot)

1,0 1 0, 1 0 0 0 0
0,1 0 0, 0 1 1 1

1,0 1 0 1 $\cdot 2^3$
1,0 0 0 1 1 1 $\cdot 2^2$
↳ 0,1 0 0 0 1 1 1 $\cdot 2^3$

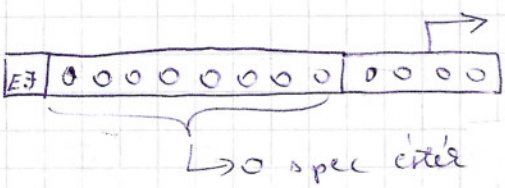
összeadva: 1, 1 1 0 1 1 1 1 $\cdot 2^3$

Ezt az átvitelt kompenzálni azzal, hogy:

1,1 1 0 1 1 1 1 $\cdot 2^4$ → normalizáljuk

1, 1 1 0 1 1 0 0 0
− 1, 1 1 0 1 0 1 1 1
0, 0 0 0 0 0 1 1 1 → 1,1 $\cdot 2^{-6}$

127 többletes ábrázolásban a 0-t nem ábrázolják.



Megsínálták a koprocesszort, ami átalakítja, nem kell számolni.

Neumann János

Helv.

- címezhetőség: minden tárolóhelyre legyen száma (azonosítás kettes számrendszerben)

Ha 256 ilyen cella van, a cím:

$$\log_2 256 = 8$$

↳ 8 bitre volt szükség

Memóriacellák számát nem 8 bitben, hanem 16-ban ábrázolták: $2^{16} = 65536$.

↓

$2^{10} = 1024$

$2^6 \cdot 2^{10}$

64 K → Commodore 64

XT: ha első nem 16, hanem 20 címbitből rendelkezett.

$2^{20} = 2^{10} \cdot 2^{10}$

K · K

M

2^{40} tera

2^{50} penta

32 címbittel 2^{32}-en cella címezhető meg.

386-os óra óta a gépek 4G RAM-ot képesek lennének megcímezni.

286-os 24 címvonal 16 MB RAM

A címzés 0-val kezdődik.

címzés: CPU irányította

a cella bitcsoport címét tárolta

byte

16 bit esetén 2-től mozgat

0, 2, 4

vagy 3-astól címzi 16 bitet

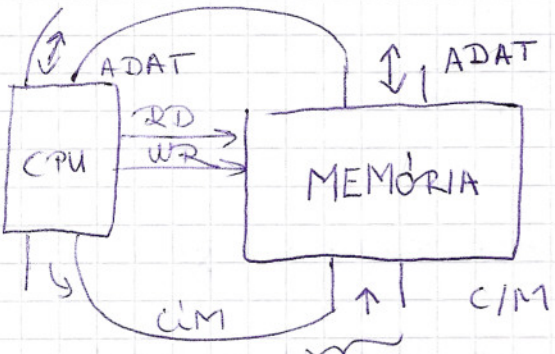
3-as és 4-es

Régen páratlan címet nem lehet adni.

Ha páros-ról veszünk címet, a páros és a páratlan tartalmát együtt kezeli

⇒ egyik irányból megcímezzük a memóriát, a másikról magát címet.

Funkciói: memória olvasás
memória írás



20 bit szélés 1024×1024 címből áll

adat: 8 bit széles

hardver II. EA

Egy csatornán „dolgozik mindenki", így mindenki tudja, mi volt a kérdés, de csak az arra vonatkozónak szól, és az is válaszol rá.

Nagyon speciális hardverigényekkel lehet > 1 memóriacím 1 adat tárolására.

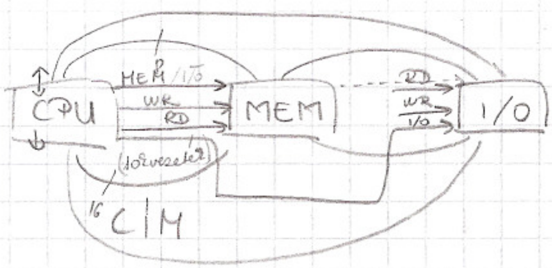
Az, hogy ? bitesen érem el a memóriát, annyit jelent, hogy mennyi az együtt mozgó bitek száma.

A perifériák is megcímezhetők.

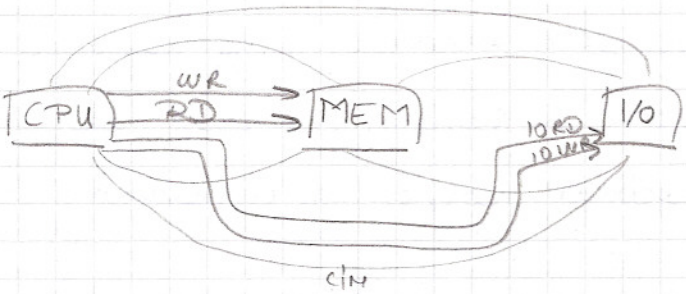
5. előadás

X. 13.

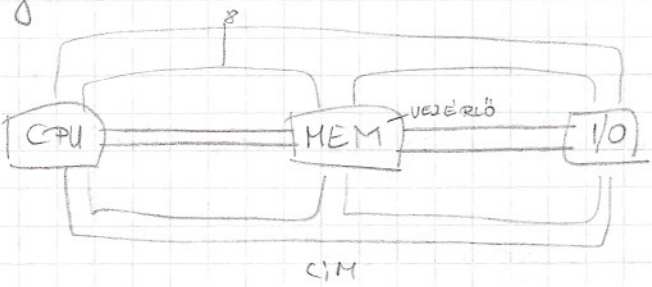
• Neumann elvű CPU perifériás műveleti hasonlóak, mint a memóriák.



• A CPU-ból az adat kifelé és befelé is mozog
• A memóriának megadhatjuk, hogy írjuk (WR), vagy olvassuk (RD)
• Létezhet olyan verzió, hogy a perifériát is lehet írni v. olvasni, neki is van RD és WR jele. Ilyenkor egy külön jellel jelöljük, hogy MEM v. I/O.
• Van amikor 1 vezetékkel oldják meg, MEM / $\overline{IO}$.



- Ebben az esetben külön vezérlő szolgál a megkülönböztetésre.



- A lényeg az, hogy meg legyen különböztetve, az mindegy, hogy hogyan.
- Egyenes ábránál nem különböztetjük meg.
- Van olyan vezérlő, amit csak az I/O, csak a MEM használ
- perifériakártyák ősei az ISA típusú kártyák → itt nem lehetett eldönteni, milyen, arról dőlt el, ha használták.
- A lényeg, hogy logikailag meg lehessen mondani, hogy melyik melyikhez tartozik.

Memória:

- kétféle típusú: a, csak olvasható (ROM)
  b, közvetlen elérésű (RAM)

- Az írható, olv.-ható memóriáknál mágnesezem logikailag az adatot
- Fenti memória: a lemezen lévő vezérlő találkozási pontjaiban vagyok a fémtárcsán, ezért lehetett átmágnesezni. ⇒ írható

1., • FLIPFLOP memóriával lehet a RAM-ot elképzelni. Ezek kikapcsolás után elvesznek az adatok. Bekapcsolás után véletlenszerű értéket kap

• A gép mindig ugyanazzal ébred, de azt nem őrzi meg, milyennel lett kikapcsolva

• ROM legértékesebb tulajdonsága, h. megőrzi a tartalmat kikapcsolás után is, az, h. csak olvasható, már mellékes

• A PC a ROM-ban lévő programok segítségével indul.

Tárolás másik eszköze: a mágnesszalag

• Nem lehet összevissza lejátszani, csak ha sorosan átlépünk v. mennyit

• a tetszőleges (itt: NEM SOROS) hozzáférés a RAM-nak köszönhető.

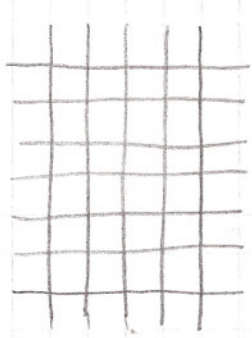
RAM

DRAM SRAM

→ • hagyományos értelemben vett memória

• nagyon rövid ideig őrzi meg az adatokat, de mégis ezt használják a legtöbb gépben

• megmarad abban az állapotban, ha állandóan kérdezgetjük.



Megkérdezés: I. oszlop, 3. sor.

Ha megkérdezzük az I. oszlopot, mindenkinek (1 sornak) frissül a címe.

- Körben frissít hátul (a háttérben) valamilyen hardver, így a dinamikus memória nem felejt el semmit
- A dinamikus memóriához 1 FET, 1 statikus memória cellához 5 FET szükséges.

(· a válaszokat a ~~CHAS~~ (cas)-ben tárolják)

- Kiolvasásnál 1 sort kér, aztán mindig csak az oszlopokat kéri be!

pl.:

	A ↓	B
1	▨	▨
2	▨	▨

A 1, 2 B 1, 2

- Sorral gyorsabban beolvassuk így a módszerrel, mintha egyenként tettem volna.

ROM gyártása:

- félvezető gyárnak nálunk · MEM kell
- gyártáskor kiállítják a megfelelő biteket ⇒
- kitalálták, az üzembe beépítenek 1 gyenge pontot, ennek kiírva hamarabb történjen
- Létrehoztak egy homogén memóriát.
- a gyenge pontokat megszüntetik ⇒ feladta

Ez a memória (PROM) = programozható, de csak olvasható ⇒ beégetjük a ROM-ot

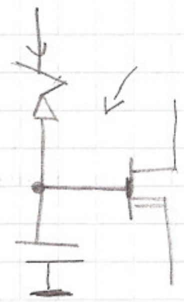
mindegyik homogén (v. Ø v. 1).

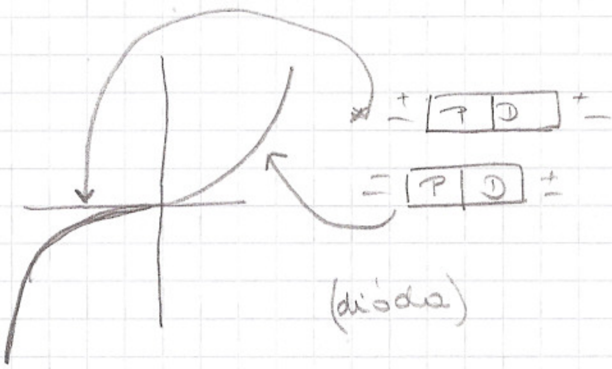
később tönkretették a tartalom a másik állapotba billent

2.,

· Észrevették a töltésvesztést elektron/cső

· Rájöttek, hogy nem kell tönkretenni az alkatrészeket





(dióda)

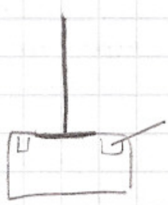
· Eleinte a diódán nem folyik áram.

· Ha 30-nál nagyobb feszt kapunk: ↓

∘ csak nagyobb töltés kellett, hogy az kondenzátor töltött legyen.

· hogy lehet kondenzátorból töltést kivenni

<u>kondenzátor:</u>



ha a vezető vezet → rövidre zárja a kondenzátort.

EPROM később UV EPROM

ha a chipet UV-vel megvilágítjuk, elfelejti a tartalmát. Törölhetjük (EPROM),

programozhatjuk (PROM).

EEPROM: ha elektronikusan meg tudjuk változtatni a tartalmát, akkor nem kell kivenni.

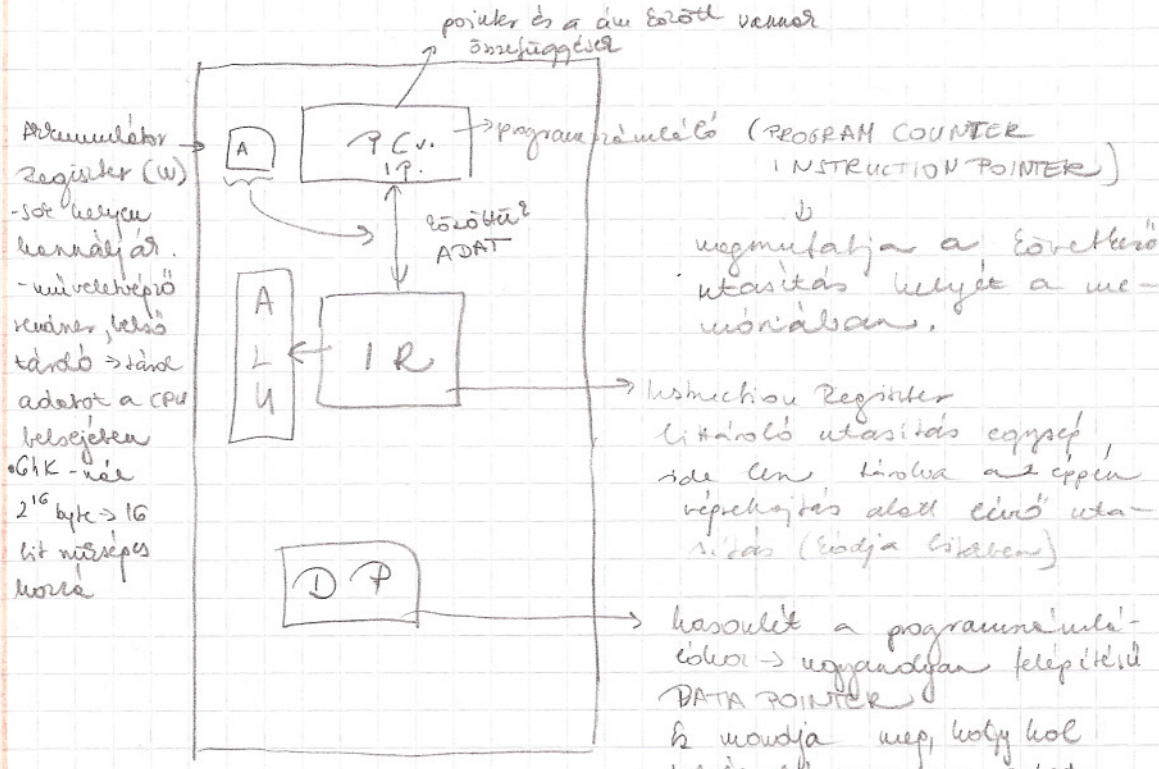
→ elektronikusan törölhető, programozható olvasható memória

FLASH memória EEPROM-ot tartalmaz, csak továbbfejlődött.

Miért nem váltja fel a ~~ROM-ot~~ RAM-ot a ROM?

- a sebesség miatt. (a ROM nem olyan gyors)
- egy EEPROM-ban a megváltoztatás korlátozott. 100 000 és 1 000 000 között

Memória és I/O eszköz



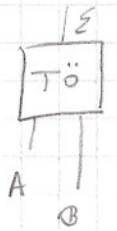
<A akkumulátor olyan nélkül, amilyen nélkül a .... >

JUMP (ugrás)

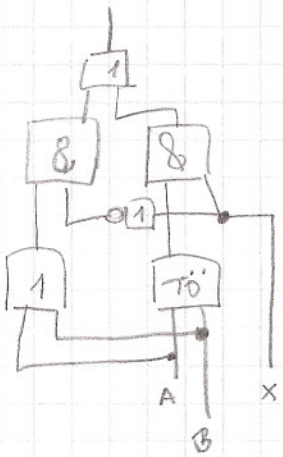
JMP ⇒ ezzel bárhol lehet folytatni

ALU: Aritmetikai - Logikai Egység

feldolgozza az IR.



teljes összeadás



$x = 1 \Rightarrow \Sigma$

A teljes összeadás összege Σ-ből ki, ha $x = 1$.

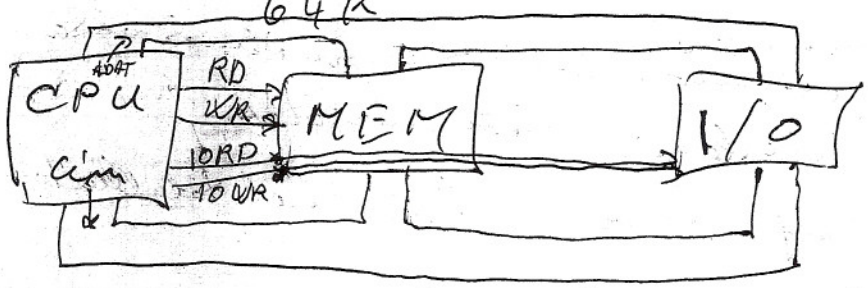
Ha $x = 0$: a kimenet A (VAGY) B. másolata Σ-ből ki.

2003. 11. 04. | Hardver II. | ea

$2^{10} = K$

$2^{16} = 2^6 \cdot 2^{10}$

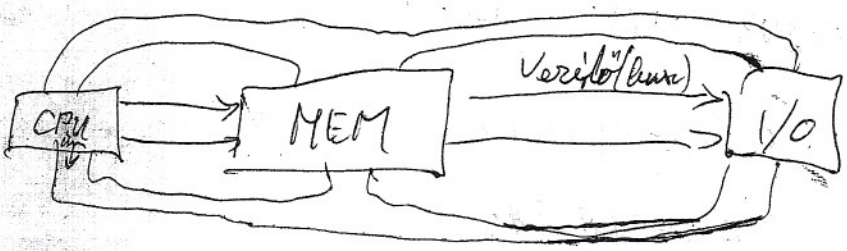
64K



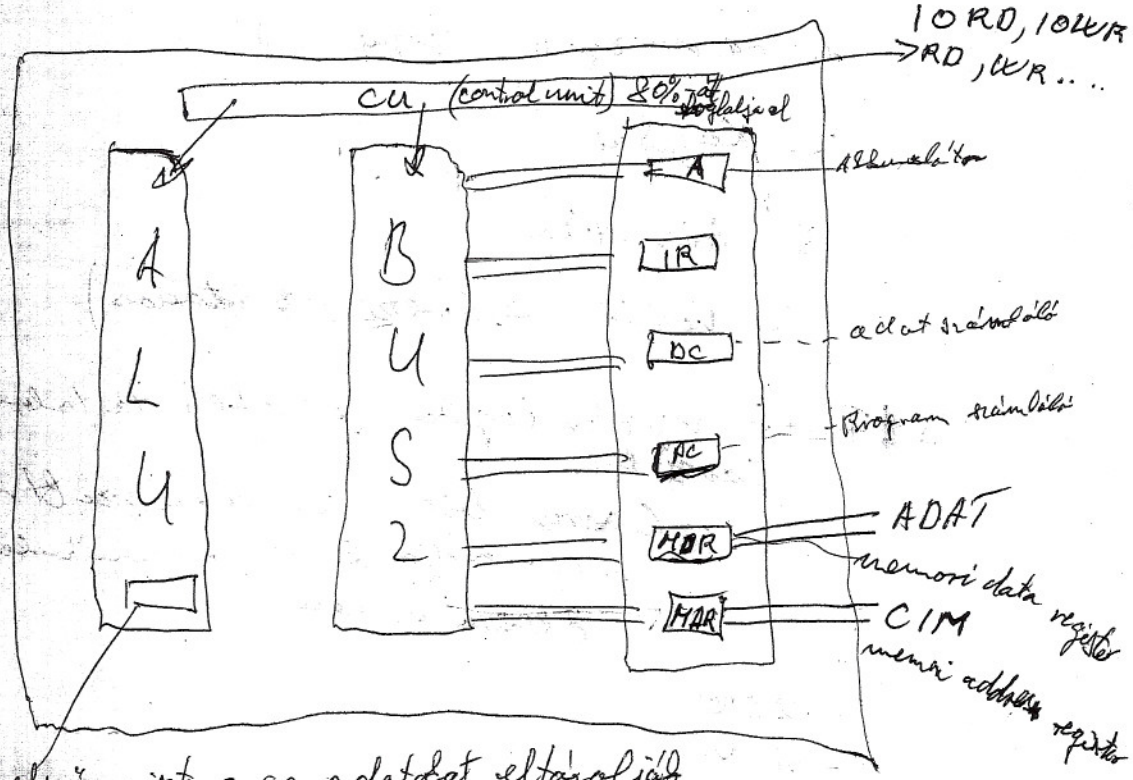
CPU adhat adatot a mem-nek az i/o-nak, de olvashat is.

címbusz:

Adatbusz:



Neumann féle CPU:



belső regiszter: az adatokat eltárolják

FLAG

CPU utasítás (ez a legáltalánosabb):

MK	1. adat	2. adat	Eredmény	Köv. ut.
	cím	cím	cím	cím

Négycímes utasítás
Ha programszámlálót használunk akkor felesleges a köv. utasítás.

Megmutatja, hogy az adat hol található

Az akkumulátort adjuk össze a memóriában lévő értékkel és itt is tárolja az eredményt.
Így alakult ki az egycímes utasítás.

1. PC → MAR (kiemásolódik memóriaregiszterbe)
2. MDR → IR (a memória data regiszterből az IR-be kerül)
3. PC ⇐ PC+1
4. VÉGREHAJTÁS

1. Adatmozgató (copy - zap)
2. Logikai
3. Aritmetikai
4. Vezérlő → (vezérlés átadás)

a FLAG regiszter logikai jelzőket tartalmaz, pl. a Carry bit.
- Carry bit az elvégzett műveletnél megmutatja hogy a művelet elfért-e 8 biten v. nem.

- ZERO
- SIGN
- OVERFLOW

Jump típusú változás a végrehajtások helyét változtatják meg maradandóan
↓ lehet
feltétel feltétlen

CALL hívó jellegű utasítások csak ideiglenes

RET (return) térj vissza ez a CALL párja
Csak akkor tud visszaugrani ha előtt megjegyezték azt a helyet ahová vissza kell menni.

A STACK az egy LIFO típusú

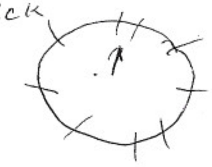
CASCADE STACK ez a CPU része elkülönített

pl. így
STACK POINTER
(1.) PC → [SP]
(2.) SP ← SP-1
(3.) PC ← új cím

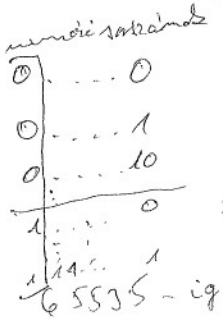
CALL
(1.) SP ← SP+1
(2.) PC ← [SP]

RET

PASCAL STACK



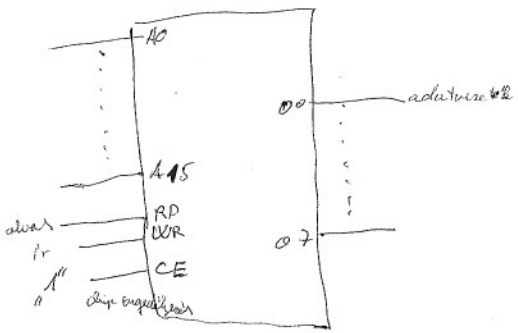
Memória címzés:

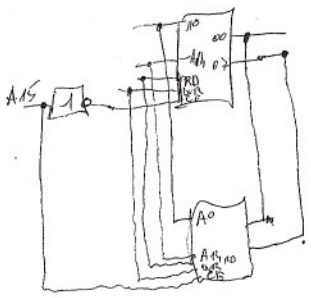


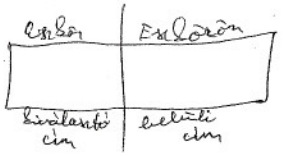
32768 (2^{15})

$A_{15} \ldots A_1 A_0$

16 címvonal



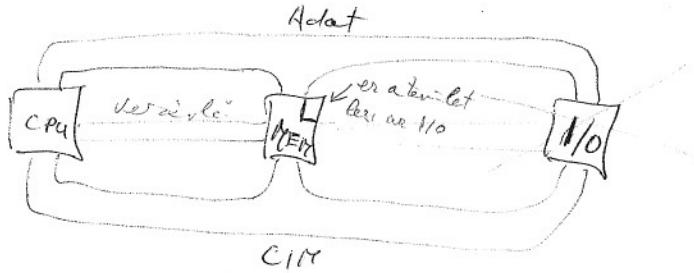




2003. 11. 11. Hardver II. 7.

Perifériák

Hogyan találhatók perifériák mikroprocesszoros környezetben?



IN
OUT

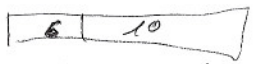
- Memóriába képzett I/O (PDP típusú gépek dolgoztak így)
- Dedikált I/O

A megcímezhető I/O területek száma 64 K. (16 bit)

3 hely, 3 I/O címtartomány a párhuzamos portnak.

1 K (10 bit) } egyszerre csak egyet figyelünk

16 címvonalból 6-nak az értékét önkényesen választjuk.



378- as cím nem figyel csak 10 bitet

$378 = 778 = 378 = 778$

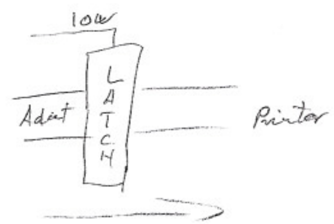
000 000 1101111.000 → 378

0000011101111000 → x78

Egy összefüggő cím csoportot vizsgálunk.

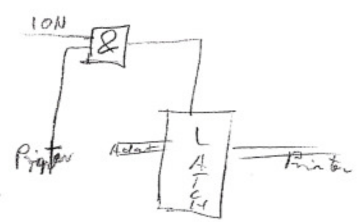
Pl.: Joystick
egyetlen porton van kiolvasható
200 - 207-ig

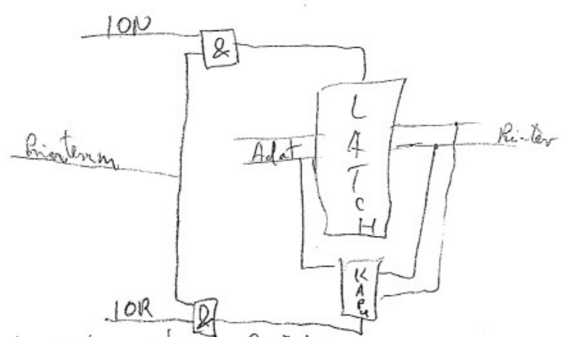
Valahol az adatbuszon van egy tároló



Helyezzük azt ... hogy az IOW legyen (I/O write)

~~printer~~





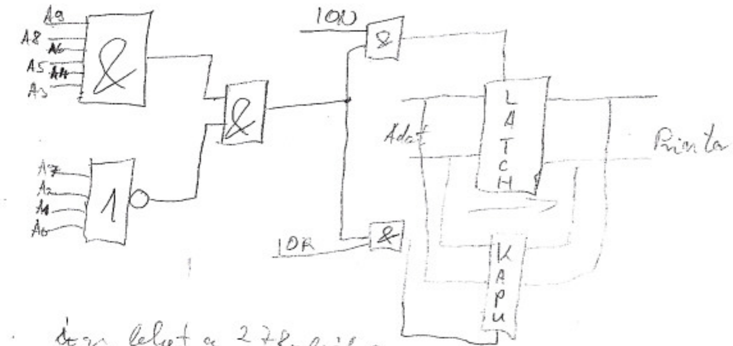
A printer címe 378.

01

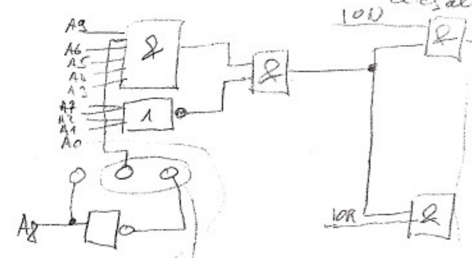
A_{15} 0000 | 0011 | 0111 1000 378 ($A_2 A_1 A_0$)

000 | 0010 | 0111 100 278

itt kell megoldani



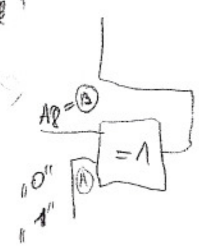
Így lehet a 278-ból kicsalni (jumper segítségével)



rövidrezárjuk
így az A8 megjelenik

Plug & play címzés
a periféria intelligenciájával lehet szelektálni.
A plug & play technika kapukkal tudja beállítani a címeket

Így egyszerűen változik



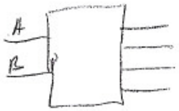
(XOR)

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

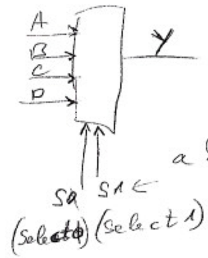
02

ALU (Aritmetical Logical Unit)

Dekóder: bemenetként n db jelet és a kimenetén 2^n jel jelenik meg



Multiplexer v. adatszelektor:



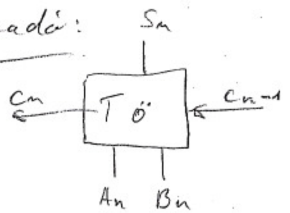
az Y melyik bemenetet másolja

a kimenet melyik szerint állítsa be a bemenetet.

$S0 \quad S1$

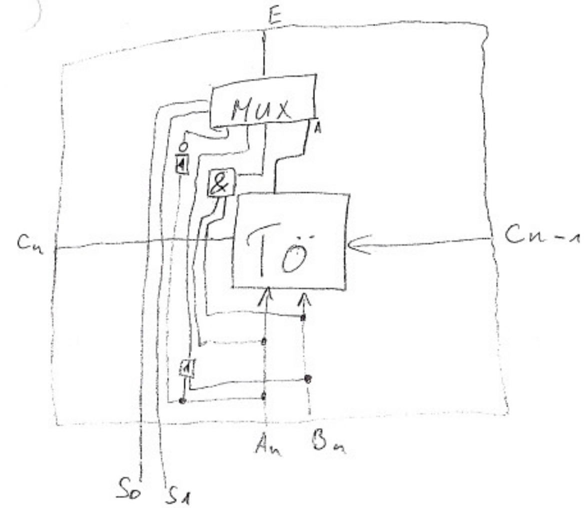
$0 \quad 0 \overset{\text{esetén}}{\Longrightarrow} Y = A$

Teljes összeadó:



An	Bn	Cn-1	Sn	Cn
0	0	0	0	
1	0	0	1	
0	1	0	1	
1	1	0	0	
0	0	1	1	
1	0	1		
0	1	1		
1	1	1		

1 Bites ALU



S0	S1	Eredmény E
0	0	A + B
0	1	A és B
1	0	A vagy B
1	1	$\overline{A}$

Ilyenből lesz a n bites ALU

Aritmetikai
Logikai
Komplementálási
Léptetési
} műveletek

A 1011 0111
B 1111 0000
ÉS 1011 0000

Maszkolásnak az „és" műveletet nevezik

A 1011 0111
B 1111 0000
OR 1111 0111

ott ahol ∅ volt nem változott

Bitmanipuláció

Karakter
ÉS OF
SZÁM.

30	'0'	00110000
31	'1'	00110001
32	2	
:	:	
39	'9'	

35

Hardver (2003. 11. 18.)

Fegyvermegajánlás a ZH-ból

2 vizsgaidőpont

Jan 3. 24 nem jó
Sorolja fel a CISC és RISC processzor utasításait megy.
Milyen adatcímzési eljárásoknál
Írja fel az OR kapu igazságtáblázatát.
Miért hívjuk kapuzásnál a kapuzást?

A processzor utasítás típusai: Adatmozgató ut.

- Adatmozgató utasítások
- Aritmetikai
- Logikai
- Léptető —//—
- utasítás sorrendet változtató —//—
- Processzor vezérlő —//—

Adatmozgató utasítások:

Memória } mozgató
I/O

Stack
Adatot mozgatnak a processzor és a memória között esetleg memória-memória között.

36

Aritmetikai utasítások:

Add, Sub → kivonás: a forrásból kivonja a célbit.

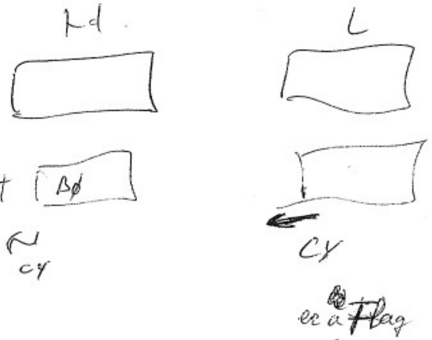
SBB → SBC (együttes átvitellel)

→ ADC: 3 mennyiséget ad össze: a carry bitet is hozzáadja

Képezi az eredményét egy összeggel

8 bit + 8 bit = 9 bit

(2) ADC (1) Add



(n) SBB ...... (2) SBB (1) SUB

ez az eljárás n-darabnál is érvényes

MUL: szorzás

összeszorzunk 2 3 jegyet akkor 6 jegyű lesz.

IMUL: előjeles számok szorzása!

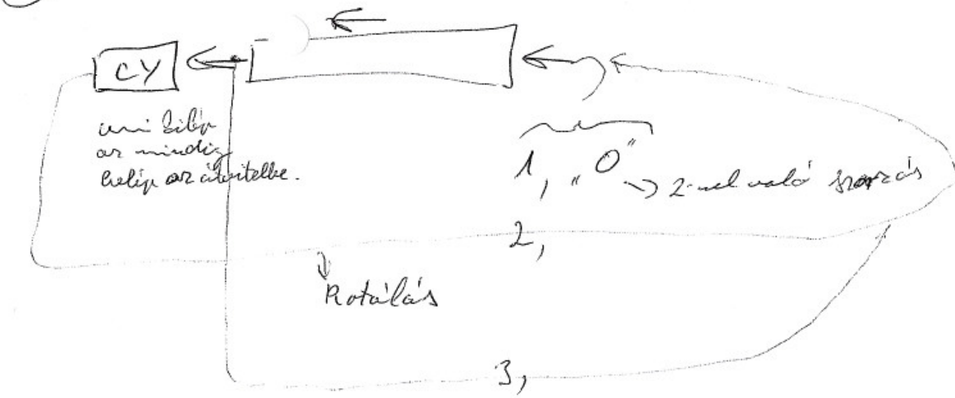
DIV: osztás

Ha nem fér osztó: 32 N (osztandó) : 16 M (osztó) = 16 N, M (maradék) — osztási túlcsordulás (tárolási bit számok)

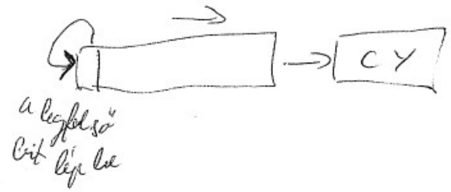
Léptető utasítás: egy bit értékkel jobbra vagy balra mozgat

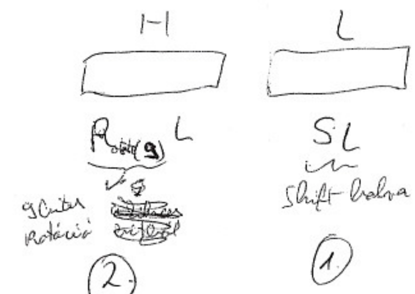
Jobbra mozgatás: 2-vel való szorzás

(2.)



8 bites Rotáció





(1.) SR ilyenkor kilép a carryben R(9) R

Processzornál 3 alapvető művelet: Logikai utasítások

a megfelelő helyiértékekig

és

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

maradás

vagy

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

eredeti Kovácsról Javítva!

37

```
   01011100
XOR 11110000
   10101100
```

Bitmanipulációs utasítások:

- XOR
- AND
- OR

Utasítássorrendet változtató utasítások:

SKIP: valamilyen feltételtől független utasítás egyetlen utasítás kihagyására alkalmas ettől függ hogy 0 v. 1.

JMP: ugrás (valamitől függő ugrás) megadja az utasítás megmondja, hogy hol folytasd.

→ Verziói:

→ ezeket összekapcsolhatod: legyen az ugrás feltételes

CALL: hívás jellegű utasítás

lelő folytatnia ha befejezte a dolgát

Ret: visszatérés

IT: megszakítás

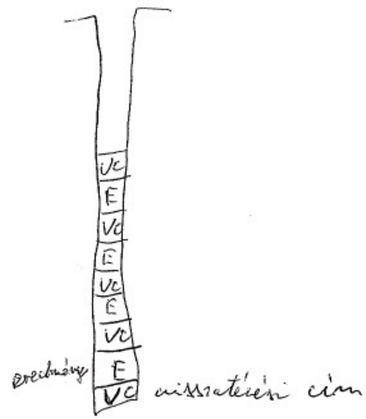
RETI

IRET

Processzor vezérlő utasítás:

- Clear Carry
- STC Setcarry
- Halt

Rekurzió:



Címzések

- direkt
- indirekt

DIREKT: MOV EAX [4B3F12H] ↗ memóriahivatkozás

MOV EAX 4B3F12H ↗ literális cím ⇒ ez nem memóriaművelet, hanem értékadás ⇒ literális álcímzés

Pl.: IQ-knt 8 embere ⇒ módosítani 9-re.

DARAB EQU 8

MOV EBX DARAB

LACI [ ]
DEZSO [ ] 〉 memóriaterületek, amiket szimbolikus névvel használunk

⇓

tehát nem kell tudni a tár fizikai címét

⇓

MOV EAX DEZSO

Innen nem látni, h. az egy konstans vagy nem, mert elhagyható a [ ].

↘ szintaktikailag helyes, szemantikailag viszont nem

INDIREKT:

nem az értéket reprezentáljuk, hanem azt, ami megmondja az értéket.

MOV EAX [EBX] a címet nem tudjuk változtatni ⇒ bele van építve a programba.

- A program futása során is meg tudja változtatni a saját kódját.
- Eddig a programozás során ezt használtár, ma már nem.
- relatív a tartalma
- relatív címzés: az éppen pc aktuális értékéhez képest
- A program futásához képesti hely ⇒ lehetővé tette, h. több program fusson.

BÁZISREGISZTER: megmondja azt a címet, amelyhez képest az adatok elérhetőek voltak. (olyan, mint a pc)

- A kezdethez képest azonos indexeken voltak, mennyivel volt az eltolás ⇒ INDEX (ELTOLÁS)
REGISZTER → viszonyít az alapcímhez

A távolságot BYTE-ban adtár meg = hány bytenyira van az alapcímtől

MOV A, [BP + (IX)] → hány byte a távolság és alapcímhez viszonyít.

MOV EAX, [EBX + ESI]

a memória konkrét értékét a bázis és az index-regiszter jelenti

- Mindegy, h. melyiE, melyiE, mert 6+3=3+6. Csak a neve alapján különböztetjük meg.

Virtuális címzés

Nincs baj, amíg egy program fut, de ha több program tud elindulni egyszerre ⇒ a program azt hiszi, hogy nincs több memóriája, pedig sok van.

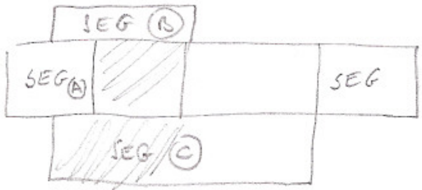
A pc-t kezelhetjük úgy, mintha 4G memóriánk lenne.

Az nem fontos, h. fizikailag hol van a memória ⇒ figyeli a címet, amit a cpu használni akar, oda „leteszi".

pl.: cölöp

lapokra osztották a memóriát ⇒ elfogyott.

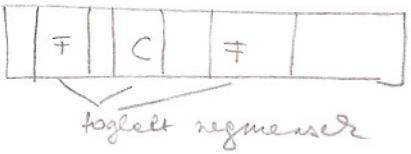
LRU : a legutoljára használt eszköz
⇓
kell tárolni a lapoknál, h. mikor használták utoljára



Változó hosszúságú szegmensekkel dolgozik, ezek egymásba lóghatnak.

Szegmenseknél a közös részre (átlapolódó rész) közös tulajdonságokat lehet definiálni.

- Paraméterátadáskor van szerepe. A közös részbe tesszük a beolvasandókat és a végeredményeket



Először üres területet keresünk.

BEST FIT → végig azt a memóriaterületet, ahová az adatbetétel a legkisebb veszteséggel jár.
Lövege a memória gazdaságos kihasználása

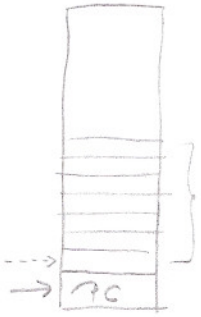
FIRST FIT → az első üres helyet kihasználja függetlenül mindentől. (GYORS)

LEGROSSZABB → a legnagyobb helyre teszi magát, biztos, hogy belefér

(EL A LEGINKÁBB BŐVÍTHETŐ ELJÁRÁS)

Rekurzív rutinok:

(Stack duzzadás kapcsán használjuk)



- rutinhívásra a stackpointer módosul
- ha van módosítás → utasításból

ezt nem használjuk semmi tárolásra, de most a használata lehetővé válik. *

- Ha a rutinnak ideiglenes tárterülete van szüksége?

ADD SP,8 → mintha 1 POP utasítást hajtottunk volna végre.

* MOV BP,SP

BASE POINTER → azt mondja meg, hol a saját változó kezdődik

Használata:

MOV 3 REG

```
MOV [BP],3
MOV [BP,2],4
```

Megjegyzések, hogyan kell használni

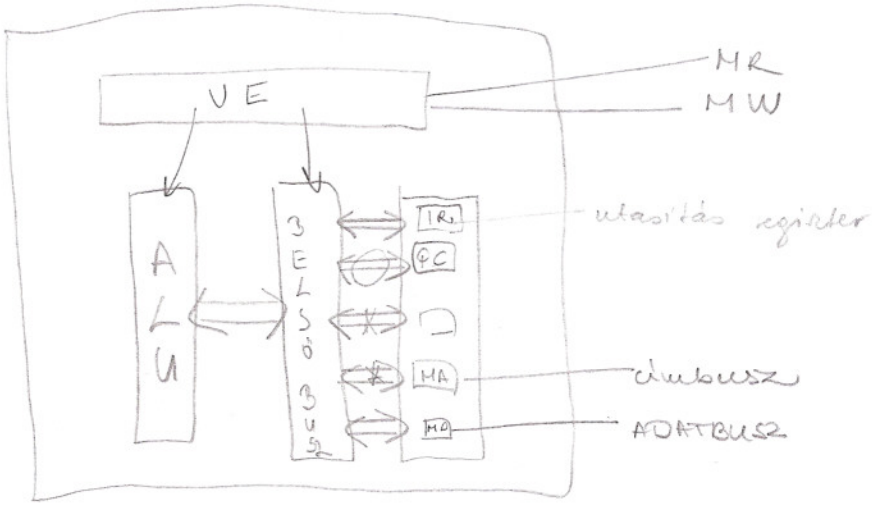
CALL ⋮ a végén

```
MOV SP,BP
```

RET → visszaállítjuk.

↳ elmentés → újra ugyanúgy kezdeni

Mikroprogram



Belső busz → alu → komplementálás (akkumulátor) busz

1; nyissuk ki az utat úgy, hogy a PC ráírhassa a buszra

* nyissuk ki a (*) utat

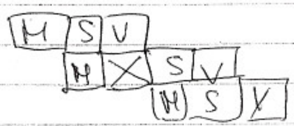
MA ← PC

2., MR ← AKTÍV (NEM: JUT) → MD

aktiváljuk a memóriaolvasást → memóriacím, a címbusz jelenik meg.

3; IR ← MD

- R4-be teszek egy flaget, módosítani fogom.
- Ha vége, módosult az R4, ⇒ flaget vissza-sorom
- A művelet végrehajtás előtt megvizsgálom, h. az általam használni kívánt regiszterek könyvelve vannak-e?
  - Ha igen ⇒ leállítjuk a pipeline feldolgozást



Az egész pipeline elcsúszott 1 ütemmel, de nem kellett hozzá másik fordítóprogram.

Másik megoldás:
- adatelőre engedés : (data forwarding)

regiszterek könyvelése: SCORE BOARDING
- vagy az utasítássorrendet változtatjuk
- -||- nop-ot használjuk
- vagy fokozatok — jump feltétel - feltételes ugrás

Valamilyen döntés alapján elkezdjük olvasni
- ha nem jó ⇒ töröljük az egész részfeldolgozást
- Nem mindig elég a pipeline-t törölni, ⇒ a már elvégzett utasítás hatásait kell törölni ⇒ lassul a progi

Ugrási előjelzés
- a processzornak jelent
- A processzor ennek a feltételezésnek megfelelően dolgozik
- Egy kimenetelű ugrás előrejelzés
- Ha mindig rossz ugrás -||- esetén ⇒ lassul a végrehajtás

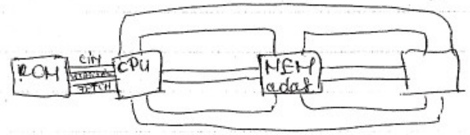
Mennyi valószínű. je, h. ez ugrani fog?

ADD AX, 0FFFFH
Jnc

Hurok: hurok végrehajtásával egy utasítást kell végrehajtani

Ez az ún. 2 kimenetelű ugráselőrejelzés
- a processzor bizonyos feltételekkel megbecsüli az elágazás várható feltételeit
- és azt a processzor jelzi

HARVARD architektúra



Kettős célra használjuk a memóriát.
előnye: és mint program saját magának is tudom módosítani

! külön busz utasításkódok továbbítására; ut. olvasására és külön a címe a fokozatokra
- ez a CPU külön módban éri el az utasításokat és van 1 külön busz amin a utasításkódok vannak

- fetch: utasítás lehívás

- külön utasítás és külön adatmemória található benne.
- Ha az külön van akkor

- Sokkal többször számolok rosszul, mint jól. Számoljuk +, h. hányszor gondolkozott jól a proci ⇒ tároljuk azt (ugrás történeti tárat vezetünk ⇒ előre jelzi pl 738 ugrásból 40 volt rossz ⇒ ezt az outputot általánosítjuk továbbra is
az ugrás várható kimenetele tárolva van.

Ha ⊖ értéket talál ⇒ ellenkezés } ugrást
ha ⊕ -||- ⟶ jelzett } hajt végre

- egyetlen 1 integrált áramköri topológia tartalmazza a hardvert
  - külvilág felé csak perifériákhoz csatlakozói felülettel rendelkezik
  - pipeline utasítások jól felhasználását 3 részre kell osztani ⇒ a fázisok azonos idő alatti végrehajtása a cél
    - minden utasításra azonos körülmények között történik

2 utasítás:

STC (carry bitet legyen 1-egyes)
MOV [BX+SI], DX
- azt a memóriát töltjük + DX értékével
- beolvassa BX-et
- —"— SI-it
- összeadja őket
- eredményt a memória regiszterbe teszi

utasítás hossza:
1–11, 17 byte

hagyjuk el a komplex utasításokat, csináljunk másik processzort!

CISK	RISK
- változó utasításhossz	- fix utasításhossz

- minden utasítás egy szónak felel meg, egy hosszú

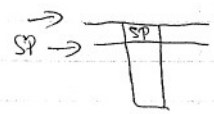
- változó utasítás végrehajtási idő	- fix utasítás végrehajtási idő

- minden utasítás 1 ciklus alatt hajtódik végre

mai proci: Kifelé cisk jellegű architektúrát mutat, belül risk-ként működik, a kettő között bee, ott fordító progi van.

CISK	RISK
- bonyolult mikroprogram	- egyszerű mikroprogram ⇓ huzalozott felépítés - csak kapuáramkörök vannak benne!
- összetett memória utasítások	- csak load, store - beírni v. kiolvasni tudok, semmi többet
- kisméretű regisztertár	- nagy méretű regisztertár - itt tartja a munkaváltozókat
- gyenge pipeline	- erőteljes pipeline
- egyszerű fordító progi	- bonyolult fordító program - elemeznie kell az ugrást, annak várható kimenetelét.

PUSH SP
- tegye a stackbe az sp-t.



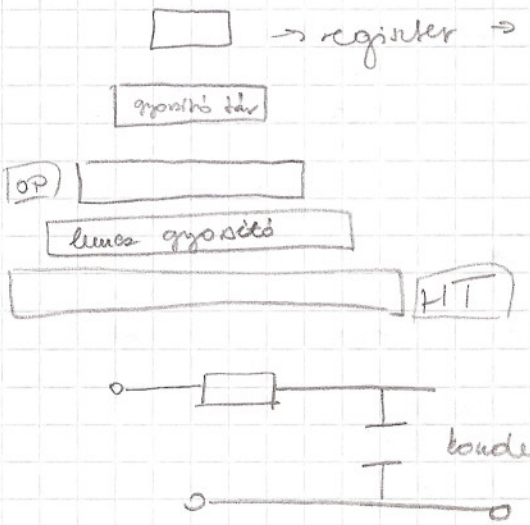
- fizikailag ugyanúgy kapcsolódnak a külvilághoz
- belülről más

jan 10 8⁰⁰
16. 13³⁰ } Kandver

11. előadás XII. 9.

- Tároló hierarchia

- A processor adatokat milyen mennyiségben, milyen gyorsasággal képes tárolni.

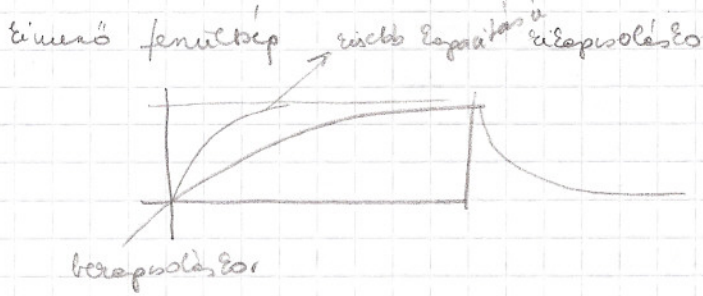


Coulomb mondta más bővel
$C = \frac{Q}{U}$-val

Coulomb: $C = \frac{Q}{U}$

$Q = CU$

$U = \frac{Q}{C} = \frac{I \cdot t}{C}$ → időfgv.



OP: operatív tár RAM

- cache-ben

- nem képes olyan sebességre, mint a processor belső regiszterei ⇒ több van belőle

JD

- drágább

HT: kattintható

- a regiszterek a legjobb és sok a hely, de lassabb az elérése

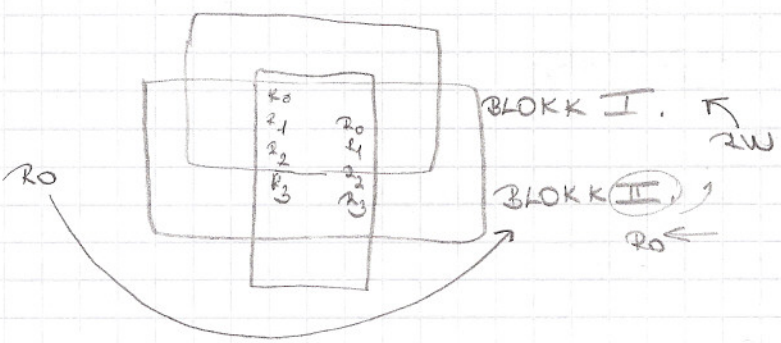
Multitask ⇒ több program használja a CPU regisztereit

a regiszterről egy másolatot kell készíteni az első és a második program számára.

⇒ alkitektúra

A felhasználó a regisztereket csak 1 lényeges részét látja

Nem látható a többiek adata ⇒ akkor és regiszterkezelésével sem ⇒ ALKALMAZZUNK egy másik eljárást.



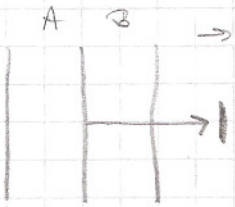
Egyik írható-olvasható, a másik csak olvasható.

Pl.:

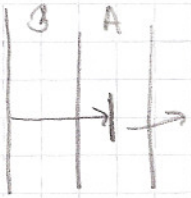
R_0-ba betérünk egy számot, és a program hol, k. van 1 szám, akkor $\sqrt{}$-ot kell vonni. 2_2-be vonatkozik $\Rightarrow \sqrt{R_2} = R_2$

Így paraméterátadás vált lehetővé.

A operációsnak védenie kellett magát a felhasználótól.



→ operációs tárban való helyfoglalás

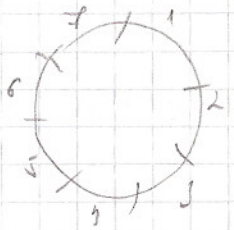


→ betöltötte a programba

A címleosztó nem volt jó → forítva nem működött.

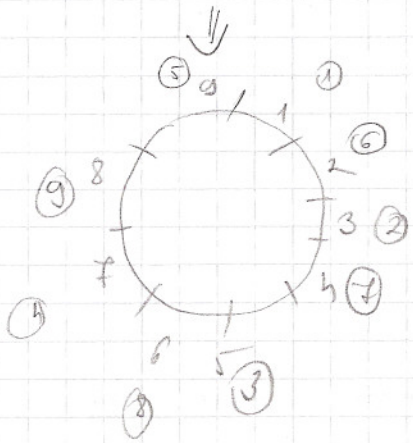
- lemez gyorsító tár:

régi lemez:



→ nektorok

Tejes tört kellett beolvasni → mert tovább forgott a betöltés után → nem a kellő részt érte.



eljárás neve:
INTERLEAVE

a programozó éri az 1-es nektort → de mindent eltárol a gép, mégis nem is érte a felhasználó

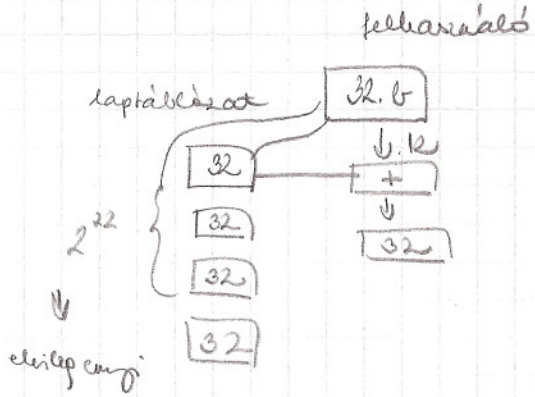
Így egy eljárással megvan minden adat 1 sávból

Sorrend az operáció memória részének látjuk →
a lemez írásos részét betartja a memóriában
Nem kell plusz mozgás végezni a 2. beolvasásnál,
csak átmásolja. (Addig a memóriában van)

Gyorsító: előbb gyűjti → utána írja fel ⇒ IGY van némi
veszélye

1 : 1	1 823772
1 : 2	1823773
1 : 3	1823771
1 : 4	1823772
1 : 5	

Memóriagyorsító tár.
286-os gépeknél jelent meg először a MULTIUSER,
MULTITASK



TLB — átlépésekhez
Fordítási előrejövő tár
↓
jó lenne a processorban tárolni, úgy, h. a memória
része

MÁSOLAT tár
↳ a gyorsító tárban másolatot tárol
CASH tár első felhasználási iránya