Digitális technika 2.

2013.02.11.

1. előadás

Horváth Tamás
toa@iit.bme.hu
IB320

www.iit.bme.hu/digit2
f.: digit2
j.: vicia106

Tárgykövetelmény:

- gyakorlatokon jelenlét
- 4. hét hétig (12 ea hétig leadni)
  - hardverleírás
  - Assembly program
- vizsga
  - 20/12 - beugrón kell
  - 40 pontos 2. rész

- MSI áramkörök
- mikroprocesszorok
- Assembly programozás

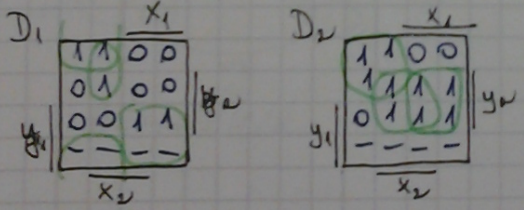
- Mikroproc. alkalmazási segédlet
2. részben a vizsgán használható

	00	01	11	10	B
00	A	C	C	A	A	0
01	B	B	C	D	B	0
11	C	A	B	C	C	1
B						

Moore-modell
Állapottábla

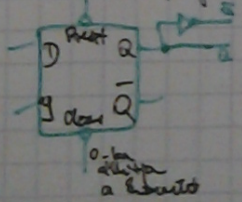
Kódolt állapottábla

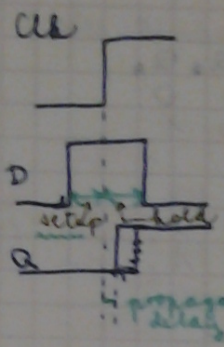
	00	01	11	10
00	11	11	00	00
01	01	11	01	01
11	00	01	11	11
10	-	-	-	-



$D_1 = y_1 x_1 + \bar{y}_2 \bar{x}_1 + \bar{y}_1 \bar{x}_1 x_2 \qquad D_2 = y_2 x_2 + \bar{y}_1 \bar{x}_1 + y_2 x_1$

Valódi flip-flopot használjunk

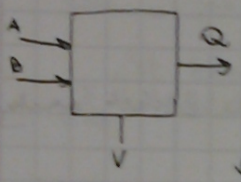




MSI áramkörök

-(SSI : kapuk, flipflopok)-

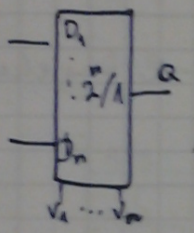
→ multiplexer

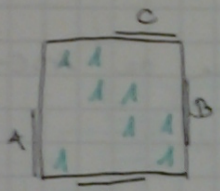


Q kimenet

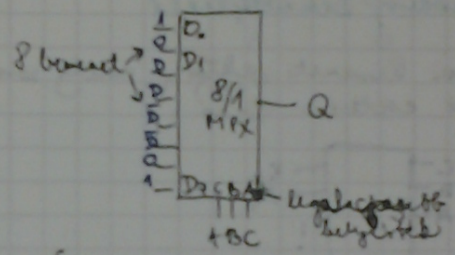
Q = A ha v=0
Q = B ha v=1

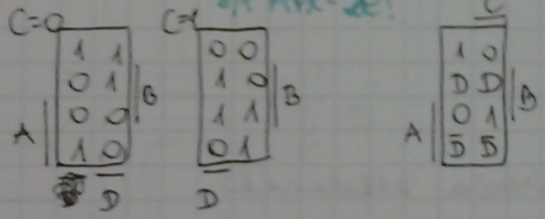
$$Q = A \cdot \bar{v} + B \cdot v$$





Valósítsuk meg 8/1 MPX-et!

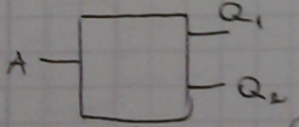




"a pontás tudja, hogy ég-e a terembe a lámpa"

$\overline{1}$ pontás visszajelzés, csengő, ha a lámpát elfelejtjük

→ demultiplexer



$Q_1 = A$ ha $v = 0$
$Q_2 = A$ ha $v = 1$

(jelkimenet alapértelmezésben 0.

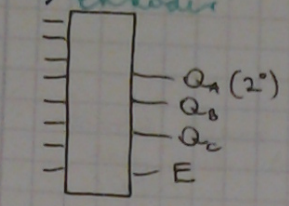
A V	Q_1 Q_2
0 0	0 0
0 1	0 0
1 0	1 0
1 1	0 1

$$Q_1 = A \cdot \bar{v}$$

$$Q_2 = A \cdot v$$

8 bemenet, 3 kimenet

→ Kódoló



D_7 D_6 ⋯ D_1 D_0	Q_C Q_B Q_A
0 0 ⋯ 0 1	0 0 0
0 0 ⋯ 1 0	0 0 1
⋮ ⋮	⋮
0 1 ⋯ 0 0	1 1 0
1 0 ⋯ 0 0	1 1 1

annak a jelnek a sorszáma jelenik meg a

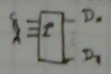
n-ből 1 kód 1/

több 1-es : prioritás ; ált. a legnagyobb sorszám jelenik meg a kimeneten

nincs 1-es :

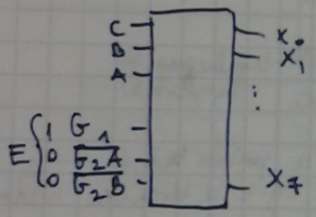
→ Dekóder

Az az áramkör, mely azon kimenetén ad 1-es értéket, amely sorszám szerepel a bemenetén



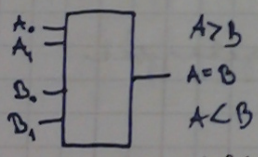
→ Dekóder / Demultiplexer

Azon kimenet aktív (0), amely sorszáma az ABC-ből megjelenik, más esetben 1-es.



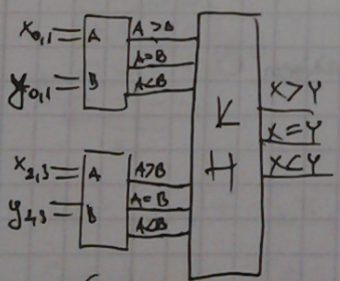
Komparátorok

Bináris számok összehasonlítására alkalmasak



$\overline{A_1 \oplus B_1} \cdot \overline{A_0 \oplus B_0}$

n-1 kódú kimenetek



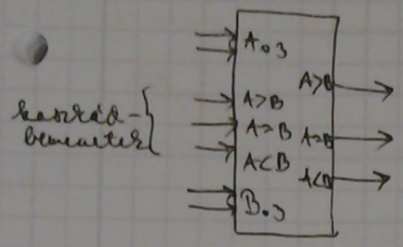
$A > B_1 + A = B_1 \cdot A > B_0$

$A = B_0 \cdot A = B_1$

$A < B_1 + A = B_1 \cdot A < B_0$

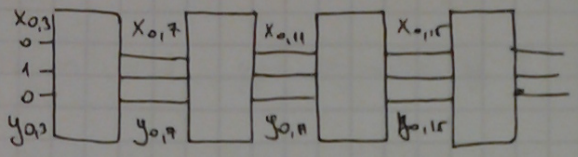
(Kaszkádosítás : ugyanolyan alkotóelemeket építünk össze)

4 bites kaszkádosítható komparátor



INPUT			OUTPUT		
A>B	A<B	A=B	A>B	A<B	A=B
1	0	0	1	0	0
0	1	0	0	1	0
0	0	0	1	1	0
1	1	0	0	0	0
x	x	1	0	0	1

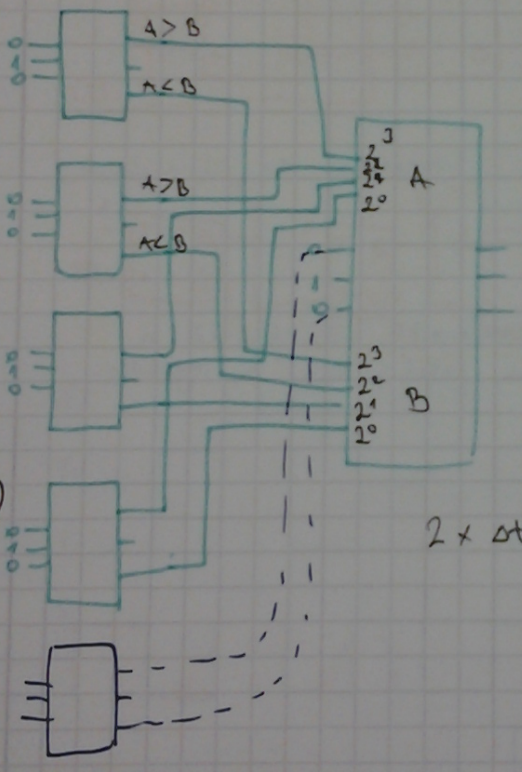
Kaszkádolás: pl. 16 bites komparátor kialakítása 4 bitesekből



Δt komp. → 4 bites k. kialakítása

4 × Δt komp. → 16 bites k. kész.

A	B	
A>B	A<B	
1	0	A>B
0	1	A<B
0	0	A=B



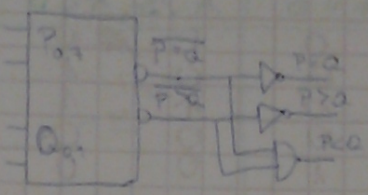
2 × Δt komp.

ü

2019. 02. 14.

2. előadás

8 bites komparátor

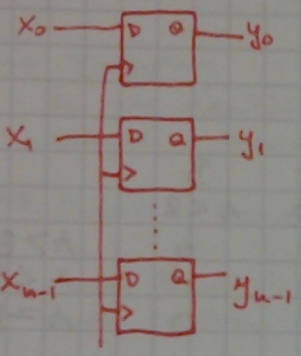


N-1 bites kimenő bus értékeit állítsa

Regiszter

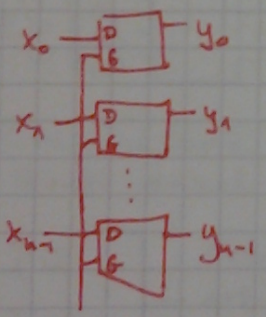
n bites reg. → n db D ff valósítja meg, közös órajel

adattárolás

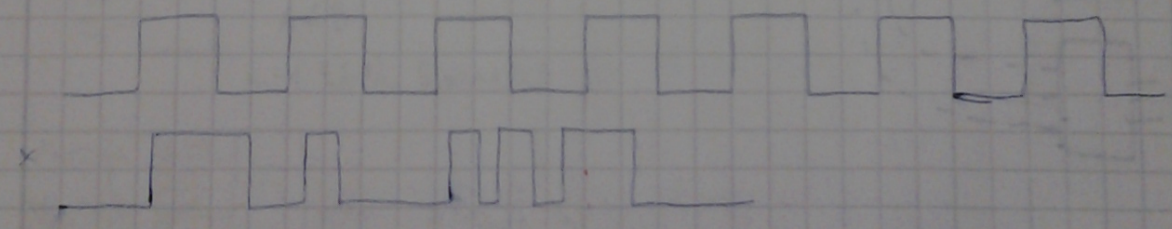


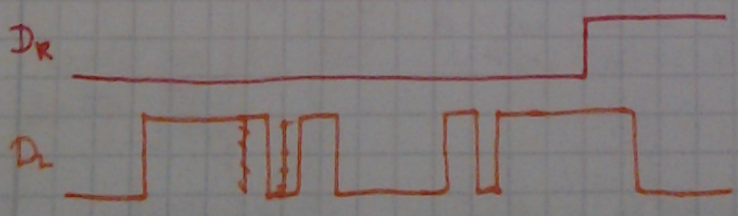
Latch

D ff-ből épül fel

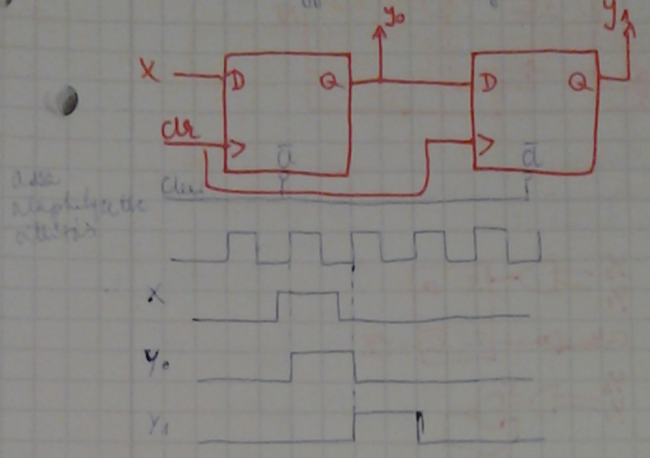


Akkor lemenő állítja elő a kimenő jelet, mint a regiszter





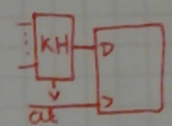
2 db D-ff, közös órajel

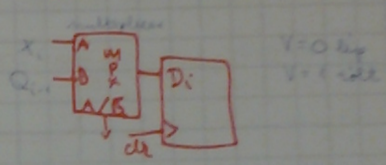


Léptető / shiftregiszter

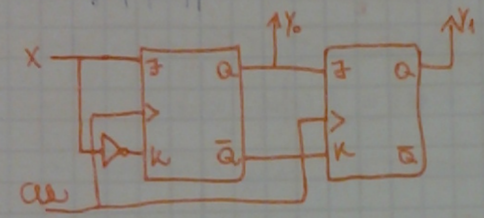
(2 bites)

1 bitcella

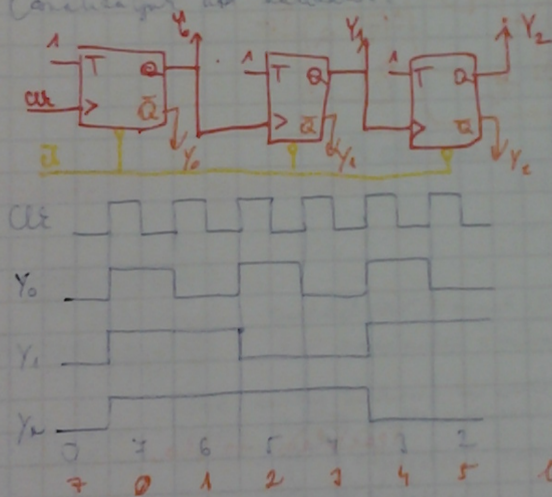




JK flipfloppal:



Analizáljuk a hálózatot!



Aszinkron számláló

Visszafelé számláló számláló

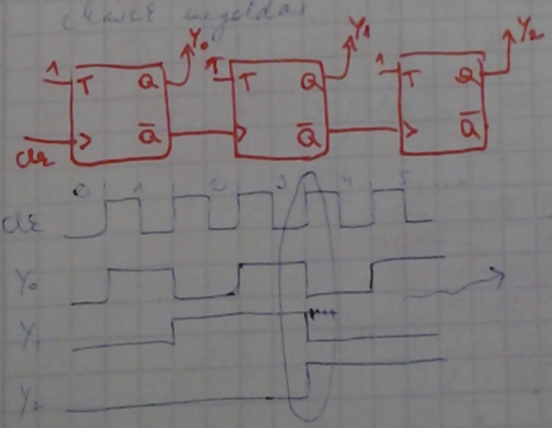
↓

Felfelé ha Q-t ... gyártani

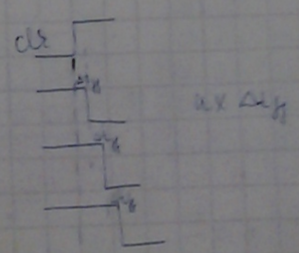
bitenként negáljuk

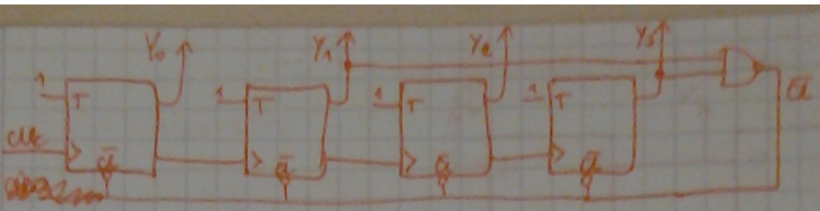
→ invertálók segítségével könnyen gyártható az ellenkező irányú számláló

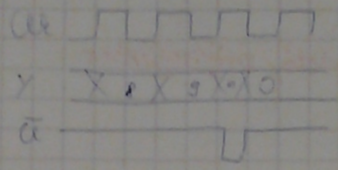
Másik megoldás



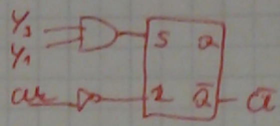
Δt_{ff} késleltetés

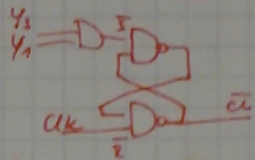






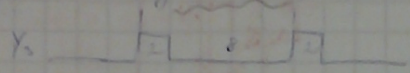
Impulzusformálásra van szükség, így nem biztos, hogy 1-től kezd számlálni





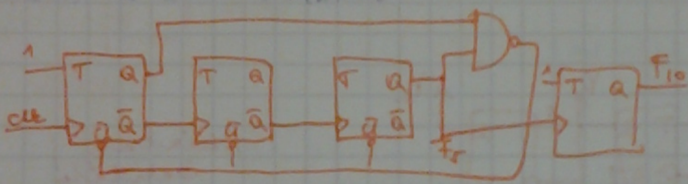
Ezt használjuk az órai számláló →

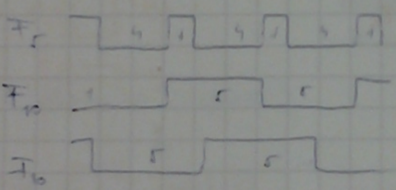
(Az eredeti frekvenciát 10-zel osztó áramkört készítünk)



20%-a kitöltési tényező

Kitöltési tényező: megmondja egy jelről, hogy hány százalékban 1.
50%-a k.t.-t akarunk



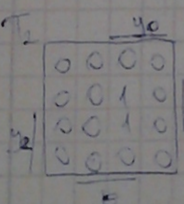


Szinkron számláló

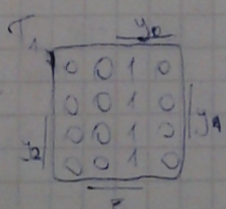
$y_2y_1y_0$	E = 0	E = 1
000	000	001
001	001	010
010	010	011
011	011	100
100	100	101
101	101	110
110	110	111
111	111	000

Vezérlési tábla

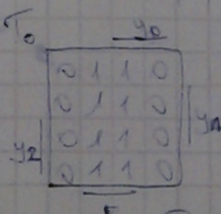
$y_2y_1y_0$	0 ($T_2T_1T_0$)	1
000	000	001
001	000	011
010	000	001
011	000	111
100	000	001
101	000	011
110	000	001
111	000	111



$T_2 = y_0 y_1 E$

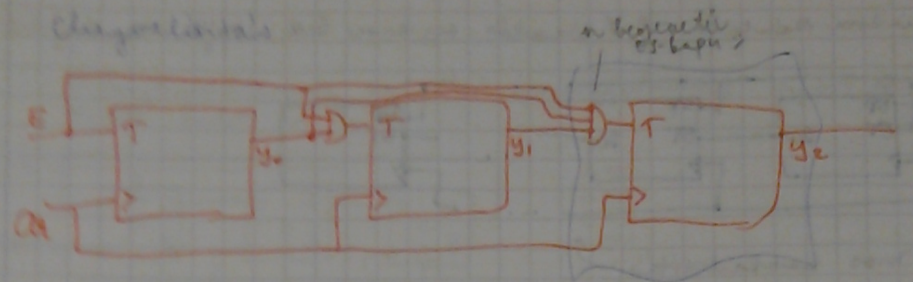


$T_1 = y_0 E$

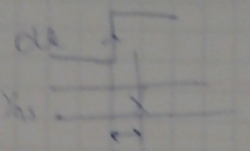


$T_0 = E$

Elágaztatás (?)

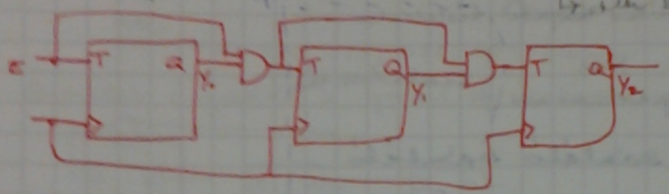


b-tábla   és-kapu + T ff



Δt_{ff} kimenet érvényes

$\Delta t_{ff} + \Delta t_{kapu}$ késleltetés után jöhet a köv. órajel
így nem számít a bitszám (n)

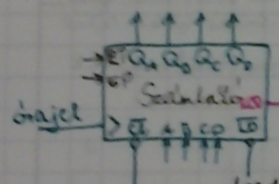


$\Delta t_{ff} + (n-1)\Delta t_{kapu}$
20 nsec   10 nsec   → 5,5 MHz soros esetben
(25 MHz párh. esetben)

3. előadás

2013. 02. 18.

Számlálók a gyakorlatban

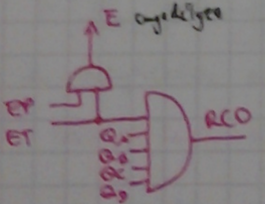


engedélyező bemenet (E)
RCO maximális értéknél 1-es (itt 15)

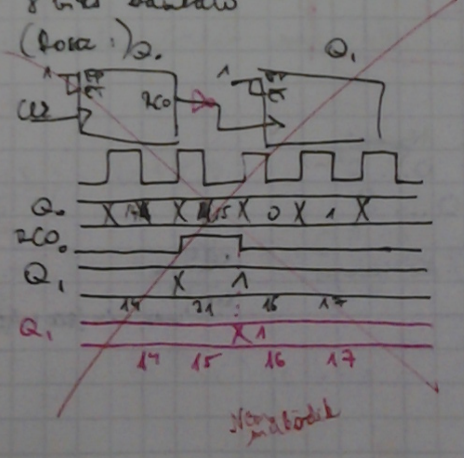
ET = 1 és
EP = 1 } → számol

Load (mindig szinkron) ha aktív, akkor ABCD-től folytatja a számlálást
aszinkron – amint töröljük (nullázó), azonnal végrehajtódó törlés
szinkron – kell az órajel is a törléshez, normál működés

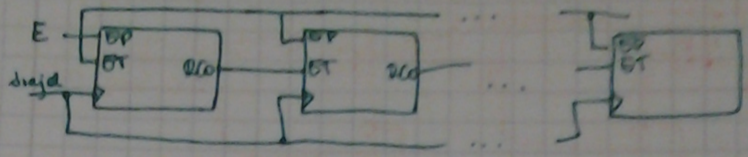
Clear v. load ? Ált. a clear prioritása a nagyobb (le van írva a katalógusban)



8 bites számláló
(Rossz:)

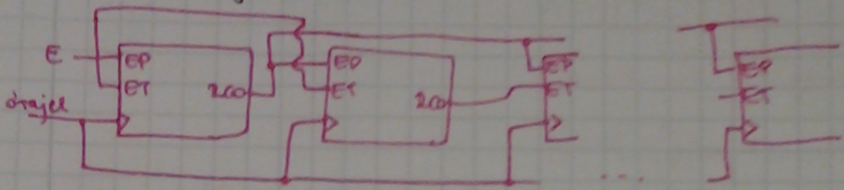


Szinkron számláló → ∀ tároló egyszerre kell kapja az órajelet



Soros kaszkádosítás ↗

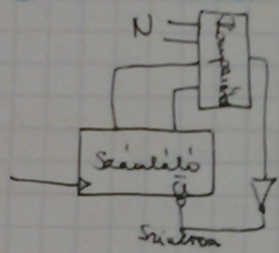
Más megoldás



16 órajelperiódus késleltetés

2. fokozattól soros
1. fokozat párhuzamos } kaszkádosítás

0 és N között számláló számláló



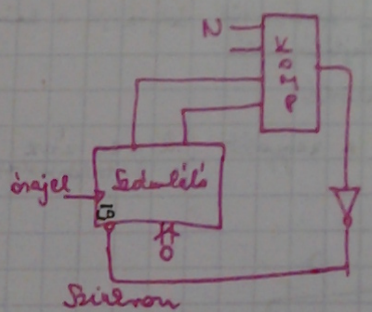
Szinkron törlésre van szükség (köv. órajelperiódusban hajtódik végre a 0-ba állítás)

asz. száml.: 011 → 100 ; ↳ 010 → 000 ↑

szin. száml.: 011 → 100 ; ↳ 000 ? 111] lehet eltolódik az az érték amit tárolni kell, az ezekben kritikus törlés lehet

↳ ezért kell szinkron törlőbemenet

Szinkron számláló + asz. törlés ~

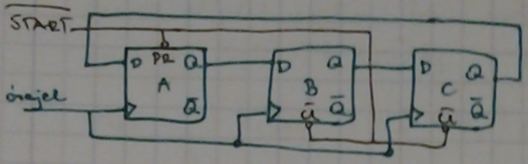


Számlálók típusai
- Bináris 0... 2^n-1
- BCD 0 ... 9
- Szin. / asz.
- Fel / Le / Kétirányú

011 ↓↑ 100 max/min (RCO kimenet)

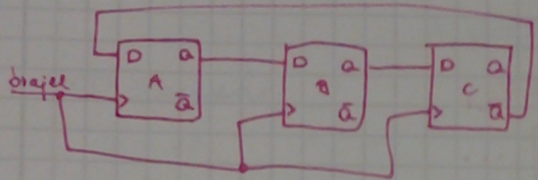
Gray számláló

∀ órajelperiódusban az előző órajelperiódushoz képest szomszédos kódot ad ki



ABC
000
010
001
100

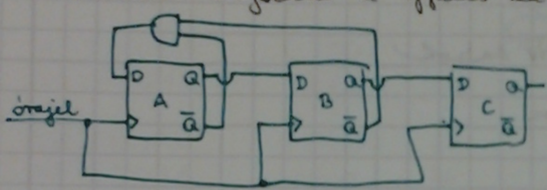
Gyűrűs számláló



Möbius / Johnson számláló

ABC
000, 100, 110, 111, 011, 001, 000 . . .

Önbeálló megoldás a gyűrűs számlálóra

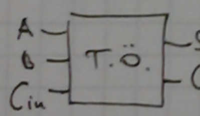


Aritmetikai műveletek

Összeadás

```
  1011
+ 1110
(1)1001  eredmény
  1110   átvitel
```

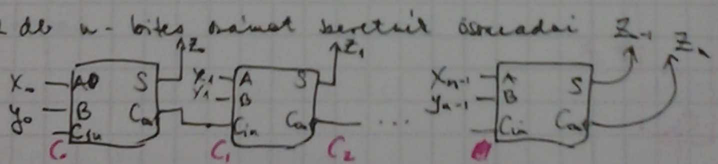
Teljes összeadó



$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + AC_{in} + BC_{in}$

2 db n-bites számot szeretnék összeadni



n. helyiértéken a hálózat késleltetés után áll rendelkezésünkre

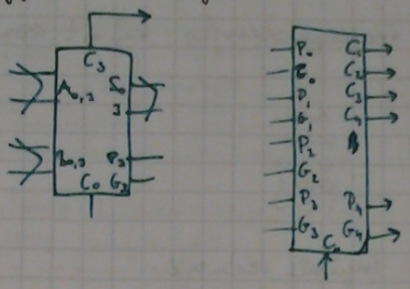
Carry-look ahead

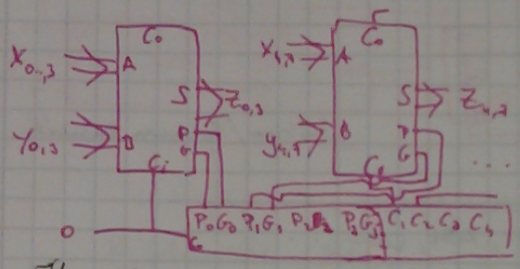
$$C_1 = \underbrace{A_0 B_0}_{G \text{ (generate)}} + C_0 \underbrace{(A_0 + B_0)}_{P \text{ (propagate)}} = G_0 + C_0 P_0$$

$$C_2 = \underbrace{A_1 B_1}_{G_1} + C_1 \underbrace{(A_1 + B_1)}_{P_1} = G_1 + P_1 (G_0 + C_0 P_0) P_1 = G_1 + G_0 P_1 + P_1 P_0 C_0$$

$$C_{i+1} = G_i + P_i G_{i-1} + \ldots + P_i P_{i-1} \cdots P_1 \cdot P_0 \cdot G_0$$

Gyors átvitelképzésre alkalmas kiléptet

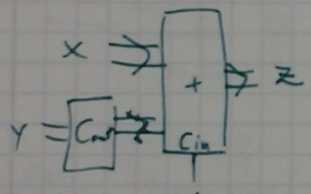




Kivonás

Kettes komplemens → negatív számok összeadással tudunk kivonni

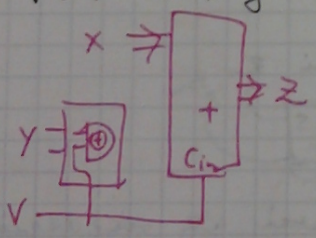
$$Z = X - Y = X + (-Y)$$



Áramkör, ami vezérlőjel függvényében összead, ill. kivon

$V = 0 \quad Z = X + Y$

$V = 1 \quad Z = X - Y$



4 bites kettes komplemens

$-8 \dots 7$

$$\begin{array}{rr} 0100 & 4 \\ +\ 0110 & 6 \\ \hline 1010 & \end{array}$$

→ negatív szám 4 bites 2-es kompl. alakban

$-6 \neq 10$ ⚡

$$\begin{array}{rr} 1100 & -4 \\ +\ 1010 & -6 \\ \hline 0110 & 6 \end{array}$$

túlcsordulás

$OUF = \overline{X_3 \oplus Y_3} \cdot X_3 \oplus S_3$ összeadás

~~Ellentét~~ előjelek esetén nincs túlcsordulás (se összeadás, se kivonás esetében nincs)
Ellentéte
Azonos előjelek esetén ha másik előjel jön ki → túlcsordulás

$OUF = X_3 \oplus Y_3 \cdot X_3 \oplus S_3$ kivonás

$OUF = \left(\overline{U}\ \overline{X_3 \oplus Y_3} + U X_3 \oplus Y_3\right) X_3 \oplus S_3$ összeadó/kivonó hálózat

TÚLCSORDULÁS ≠ ÁTVITEL !

4. előadás

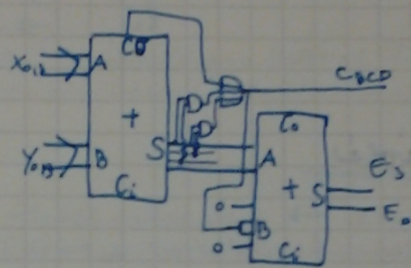
2013.02.25.

BCD összeadás

0 ... 18

	Bináris	BCD	
0	00000	00000	0
9	01001	01001	9
10	01010	10000	16
18	10010	11000	24
19	10011	11001	25

+6



$C_{BCD} = CO + S_3 S_1 + S_3 S_2$

>=16   10 ... 15

6os korrekció

ha ez a bit
10 1010 ért 1-es
11 1011 vagy a 2
1100 helyiértékű
bitpozíción
áll 1-es
15 1111

Szorzás

1jegyű 1jegyű , összeadni őket → decimális számok esetében

bináris számok esetében

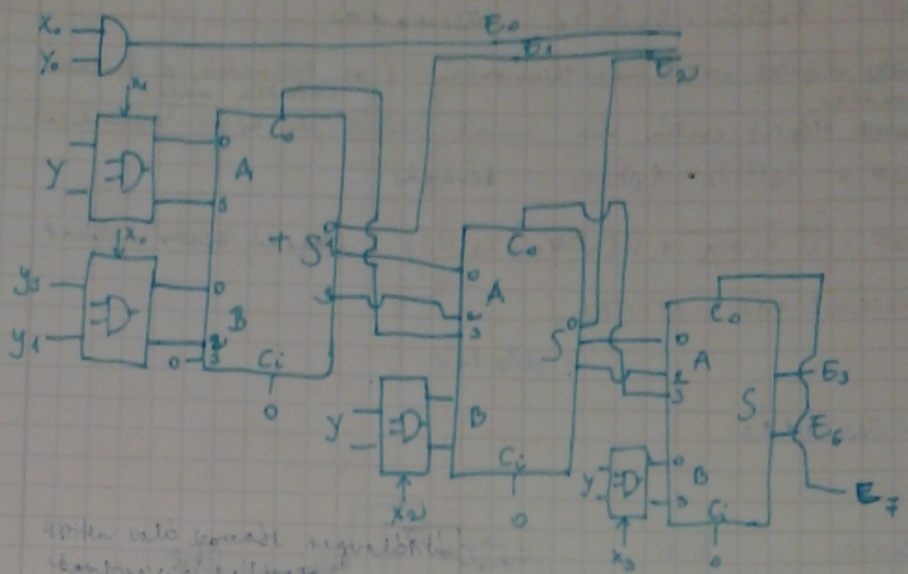
$Y_3\ Y_2\ Y_1\ Y_0 \cdot X_3\ X_2\ X_1\ X_0$

1101 · 1011

$1 \cdot 0 = 0$
$0 \cdot 1 = 0$
$0 \cdot 0 = 0$
$1 \cdot 1 = 1$
} ÉS kapuval megoldható

```
   1101        x0 ]
  1101         x1 ] 1
 0000          x2 ] 2
+1101          x3 ] 3
10001111
```

3 4 bites összeadó



4 biten való szorzást regisztereken kombinációs hálózattal

Adattárolás

regiszterek → regiszterdombok a memória (hány bit adat tárolására alkalmas a memória)

Kilo 2^{10} 1024

MEGA 2^{20} 1024 x 1024 (≈ 1 millió)

Giga 2^{30}

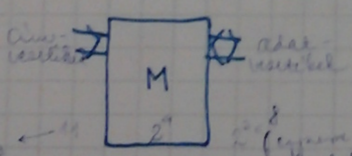
Tera 2^{40}

Hány címvezeték kell, hogy elérjük az adatokat

2^{10} → 10 vezeték

2^{20} → 20 -||-

2 k · 8 bit → 16 kbit



2^{k}

10-12 között

c k a

$2^c \cdot 2^a = 2^k$

$c + a = k$

$2^c \cdot 2^a = 2^k$

$c + a = k$

Memóriák típusai

ROM - read only memory
csak olvasható

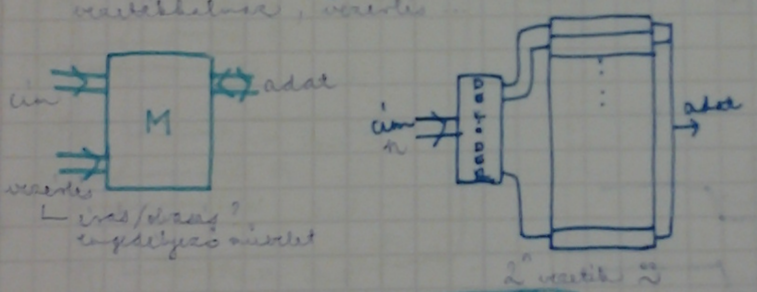
↳ **MPROM** - mask programmable
gyártás során eldöntik, hogy egy adott rekeszbe milyen adat kerüljön

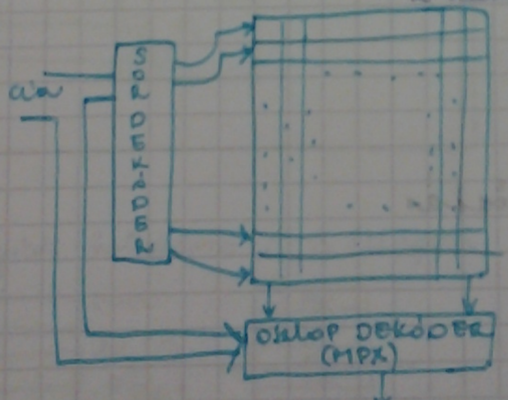
↳ **PROM** - programable
1-szer programozhatja a felhasználó

↳ **EPROM** - Erasable prog
- UV fénnyel törölhető, ha napra teszik a chipet, törlődik az addig tárolt információ **UVEPROM**
- **OTPEPROM** One time programable EPROM - beépítésnél
- **EEPROM** Electrically Erasable
(E²PROM) elektronikus úton ~~programozható~~ rendezhető a törlés blokkonként zajlik, nem iszonyatos
$t_{írás} >>> t_{olvasás}$

RAM - random access memory
tetszőleges hozzáférésű - írható - olvasható memória
indításkor nem definiált a memória tartalma
↳ **SRAM** - static
tárolás flip-flopokon
↳ **DRAM** - dynamic
tárolás kondenzátor (kisebb, mint a ff-)
baj: szivárgás előfordulhat → periodikusan frissíteni kell,
ki kell olvasni minden bitcellát, így tárolja újra

Mire van szükségünk?
vezérelhetőség, vezérlés





Vezérlések

CE chip Enable
működjön-e a halózat?

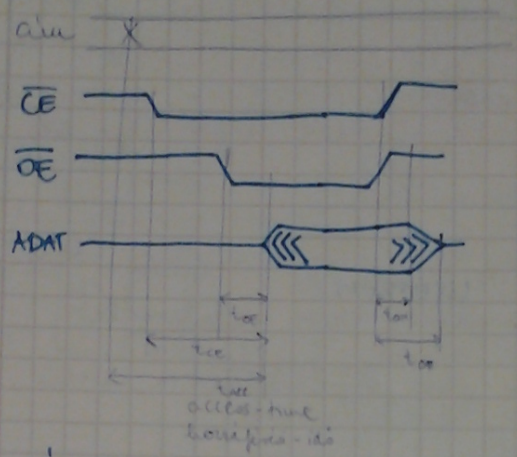
OE output Enable
érvényes legyen-e a kimeneti meghajtó

R/$\overline{W}$ read / $\overline{write}$
alapesetben olvas, de írhatjuk is

CAS Column } Address
RAS Row

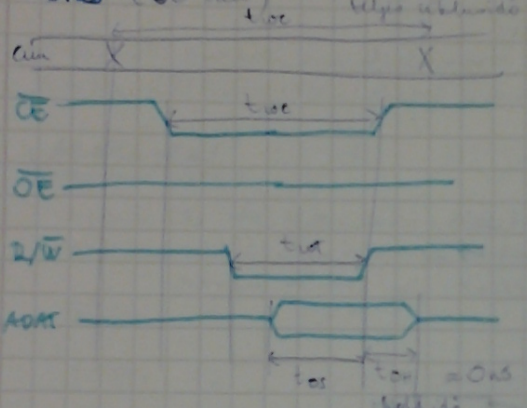
Sor vagy oszlopcím?

Olvasás

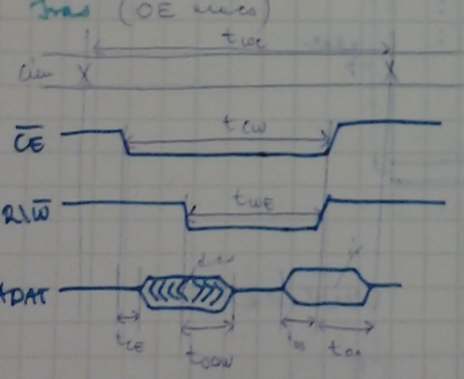


$t_{OE} < t_{CE} \leq t_{ACC}$

Írás (OE mindig)

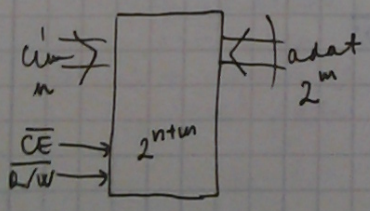


Írás (OE vezérli)



2013.02.28. 5. előadás

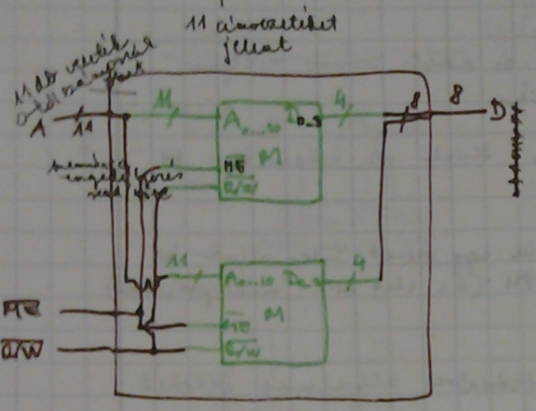
Memória interface:



Memória kapacitásának növelése

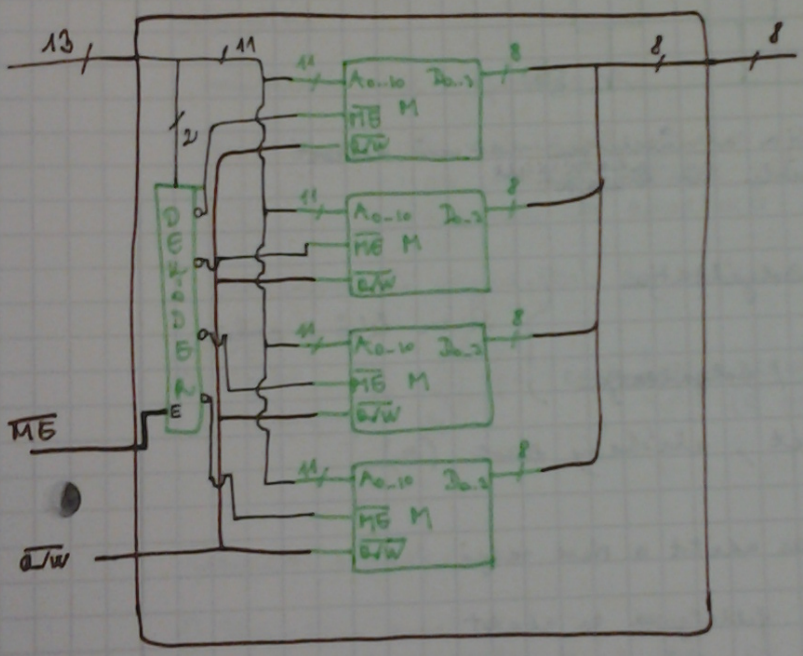
↳ horizontális - bitszélesség növelése

2k x 4 bit → 2k x 8 bit

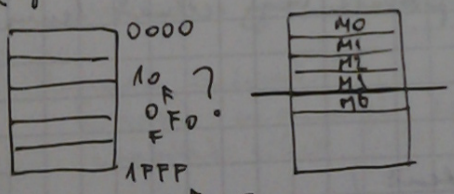


↳ vertikális - memória címezhetőségének növelése

2k x 8 bit → 8k x 8 bit



(nyitott kollektoros / 3 állapotú lehet, de inkább 3áp.)



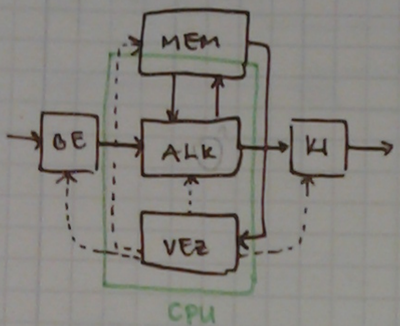
RAM extra célokra

ROM memória esetén nem mindegy, milyen módszert választunk

Vezérlőcsík kell, ami 4-ből 1 bitet szolgáltat a kimenetén → dekóder

Neumann architektúra

Aritmetikai logikai egység ALK
Operandusok tárolása a memóriában MEM
VEZ Vezérlőegység - a műveleteket megmondja a [illegible]

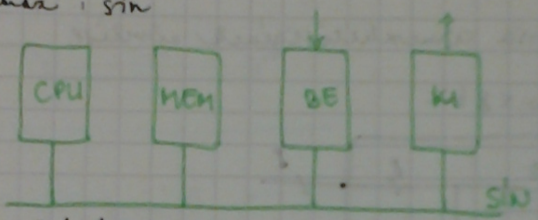


Utasítás és adat azonos helyen tárolódik

Periféria - kezeli az eredményt KI

A memória egy részét integrálják be
↳ CPU (ha elég kicsi: mikroprocesszor)

Adatcsere a cél → adatok továbbítására alkalmas vezeték-halmaz : sín



adat
cím - azonosítja a műveletben részt vevő eszközt
vezérlés - vezérli az adatátvitelt

MASTER
~~ADATOT~~, CÍMET, VEZÉRLÉST szolgáltatja
SLAVE
adatot értelmezi / szolgáltatja

} Sínen lévő 2 eszköz

! csak 1 master lehet, akárhány slave (n)

FORRÁSESZKÖZ
vezérlés hatására az adatot a sínre rakja
CÉL
vezérlés hatására feldolgozza az adatot
! 1 forrás, n cél

Nem szükséges a masternek forrásnak vagy célnak lenni
↓
DMA direct memory Access
közvetlen memória hozzáférés

(lehet több master → megküzdenek)

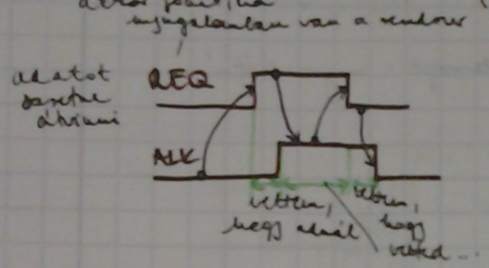
Többen fele lehetne a cím és vezérlés, de 3state-t használnak inkább. [illegible]

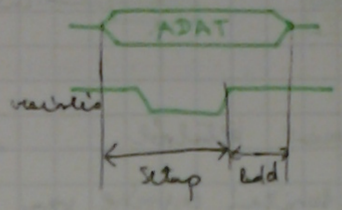
Szinkron

órajellel ütemezve történik az adatátvitel

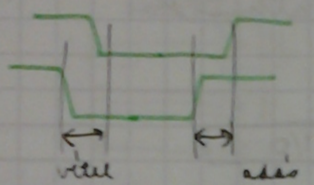
Aszinkron

Hand-shake jelekkel valósítja meg az adatátvitelt (jelpár)





(vezérlőjelek negált logikájúak, ha elvesztene egy vezetéket, aktív lenne, ha nem) negált lenne.



Utasítás

műveleti kód + operandusok

Szükség van: az operandusok címeire   op1   op2 } 4 című gép
: az eredmény címére
hol van a következő utasítás → PC

Szekvenciális végrehajtás a Neumann modellben

3 [op1, op2, eredmény] + 2 című [op1, op2 + eredmény] → akkumulátor (a processzor 1 regisztere) → [op1 → 1 című gép

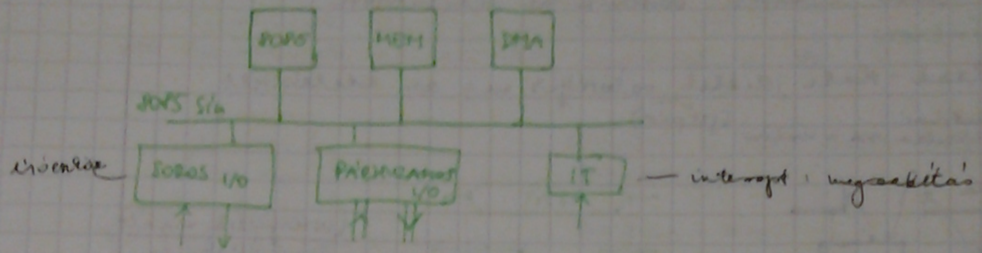
Címzési módok

1) Közvetlen adat
   operandus értéke benne van az utasításban ~ konstans érték

2) Indirekt
   utasítás az operandus címét (memória) tartalmazza

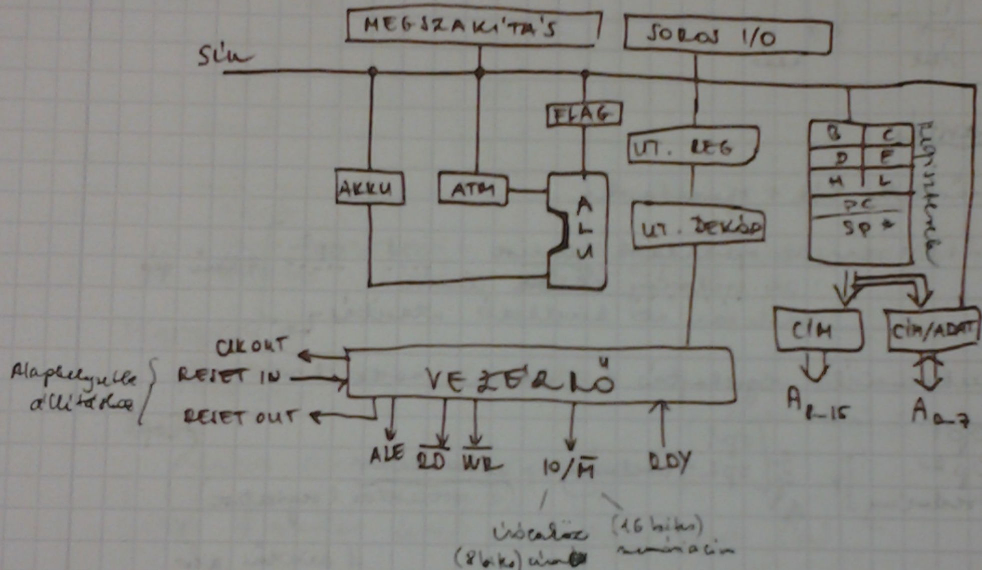
3) Regiszter indirekt
   utasítás megmondja, hogy melyik regiszter tartalmazza az utasítás címét

80■85-ös processzorok



Egy 8085-ös processzor belülről

8 bites aritm. logikai egysége van
egyik bemenetére csatlakozik az akkumulátor



Flagek
- aritm. logikai művelet 0 volt-e? → Z zero
- ~~Ez~~ p — " — poz. v. negatív-e? → S sign (előjel)
- átviteli flag, keletkezett-e átvitel a leg. ~~...~~ helyiérték felé? → CY carry (nem overflow!)
- páros v. ptlan 1-es bit van → P - parity
- BCD számoknál keletkezett-e a 7. és 5. bit között átvitel → AC - segédátvitel

Szinkron hálózat felelős az vezérlésért és a külvilággal való kapcsolattartásért

Stack adatok..., stack mutató * (LIFO - last in first out)

Időmultiplexelt a cím/adat vezetékek
idő egy részében, más részében

I/O eszközök 1-1 regiszteren keresztül programozhatók

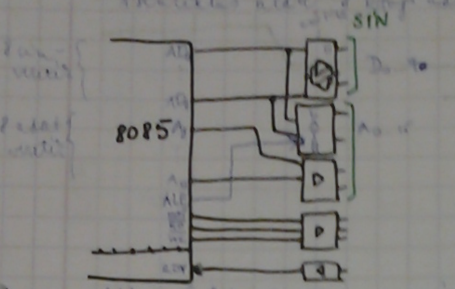
ALE address latch enable
adatként v. címként kell értelmezni - közli a külvilággal

Clk out
6.144 MHz
↓
3.072 MHz

6. előadás

2013.03.04.

[illegible]



Address Latch Enable - vezérlőjel ALE

Meghajtó áramkörök / bufferek / használata kell ezen - erősítő ▷

LATCH DG-ff

IO/$\overline{M}$: IO, vagy memóriaeléréss

Ready filozófia

↳ Ready mindig legyen (→ mikor) nagyon gyorsnak kell lenni r/w
↳ Alapértelmezésben RDY = 1, szólhat az eszköz, hogy megvárja a processzor → WAIT kérés
↓
Kimenetek összekapcsolását elviselő alkatrészek kellenek (pl. open collector)

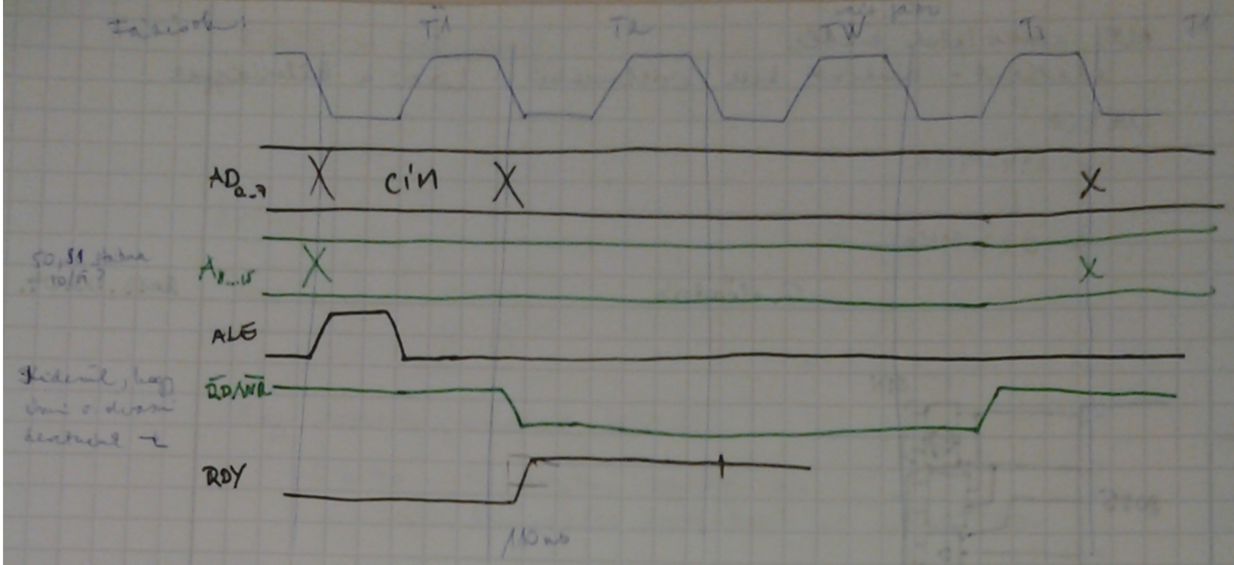
BAJ: nem veszi észre a rendszer, ha valamelyik eleme nem működik...

↳ Alapértelmezésben RDY = 0
↳ Ready adás
Hátrány: kell Ready-logika

$\overline{READY}$

WATCH DOG áramkör

→ Ha nem jön a ready jel, akkor a rendszer megáll. Kell egy olyan áramkör, ami biztosítja azt, hogy kb. 10 ms alatt minden eszköztől választ kapjunk



RDY a 2. es fázis végén mintavételez

1. fázis címkibocsájtás

2. fázis művelet kijelölése, időzítése
processzor a vezérlőjelet adja

3. fázis befejezi a µp a műveletet, elkezdődik előlről ...

---

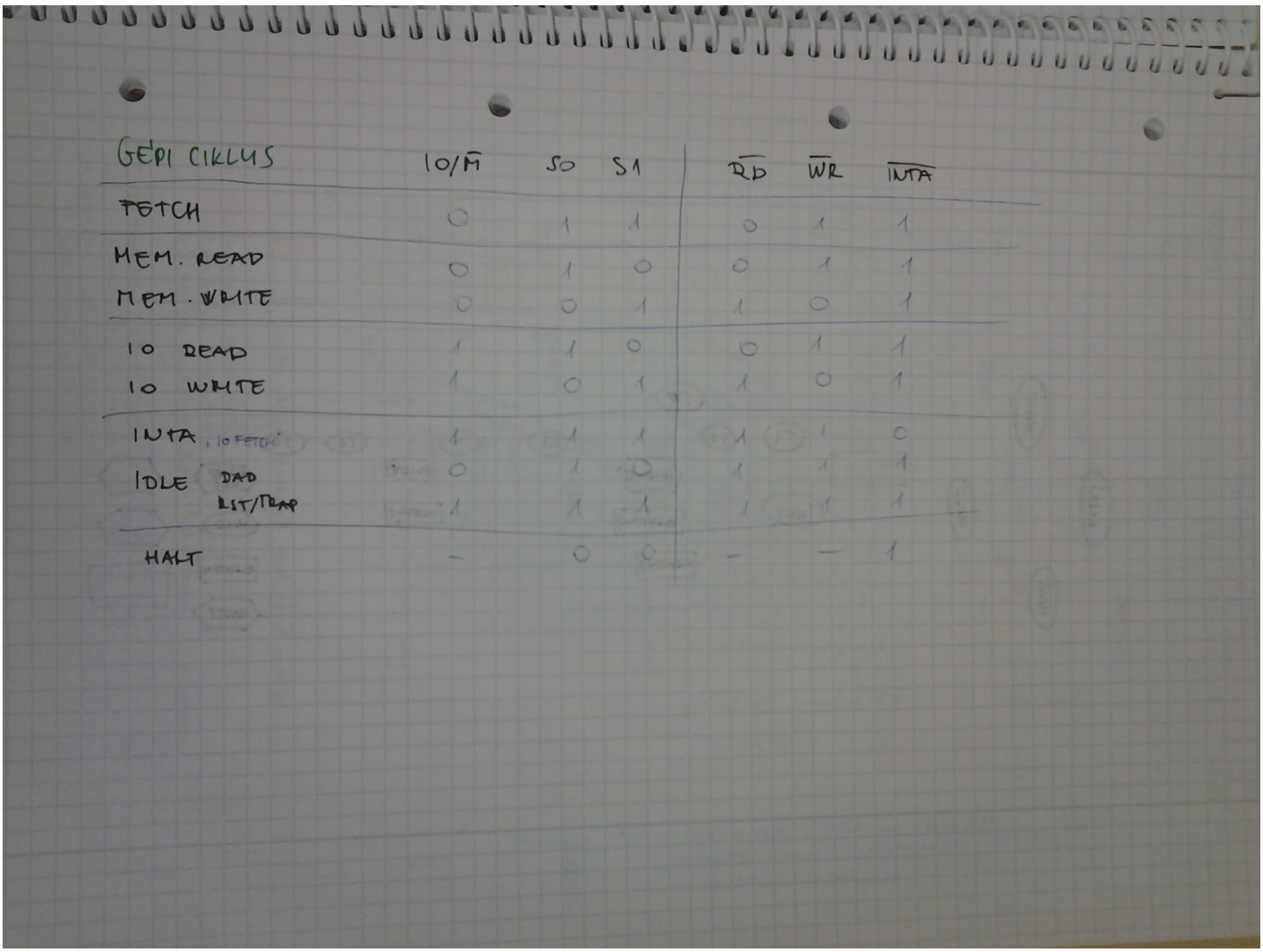
1 gépi ciklus 1 művelet a buszon

utasítás → gépi ciklus – fázis
1 — 1-5 — 3-6

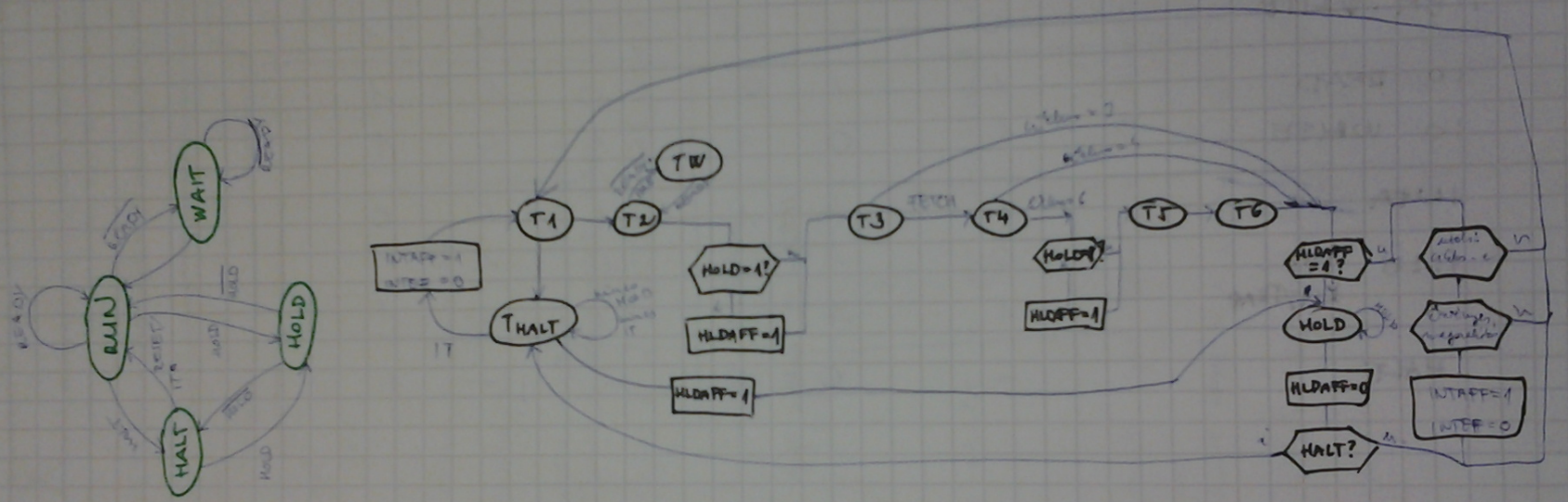
Gépi ciklusok

- FETCH
  a beolvasott adatot a proc utasításként értelmezi
- MEM READ
- MEM WRITE
- IO READ
- IO WRITE
- INTA – megszakít., spec. típus, 8080 kompatibilis
- IDLE < DAD / RST/TRAP
- HALT : processzort áll, nem csinál semmit

Minden adatművelet a legfeljebb 6 fázist vesz igénybe



GÉPI CIKLUS	IO/$\overline{M}$	S0	S1	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
FETCH	0	1	1	0	1	1
MEM. READ	0	1	0	0	1	1
MEM. WRITE	0	0	1	1	0	1
IO READ	1	1	0	0	1	1
IO WRITE	1	0	1	1	0	1
INTA, IO FETCH	1	1	1	1	1	0
IDLE DAD RST/TRAP	0	1	0	1	1	1
	1	1	1	1	1	1
HALT	–	0	0	–	–	1


WAIT
RUN
HOLD
HALT
T1
T2
TW
THALT
T3
T4
T5
T6
HOLD=1?
HLDAFF=1
HOLD
HLDAFF=0
HALT?
INTAFF=1
INTF=0

RESET

- PC ← 0 , INTE FF ← 0
- T1 fázis, FETCH
- SOD = 0
- RST MASK : 1 (letiltódás)
- RST 7.5 flipflop : 0

] Definit

Nem definit

- Regiszterek
- Flagek.

8085-ös program azzal kezd, hogy a stack mutatót beállítja, úgy, hogy a vége eggyel kisebb címen RAM legyen

UTASÍTÁS.

Legalább 1 byte, legfeljebb 3 byte hosszú

- 1. byte: mindig a művelet kód (opcode)
- 2. 3. byte: művelet operandusa; attól függ, milyen címzési módot használ

Ezeket támogatja a 8085-ös proc.:

- → közvetlen adat (operandus = adat) 8-16 bites
- → direkt adatcímzés (16 bites)
- → indirekt adatcímzés
  - ↳ regiszterpár tartalmazza az operandus címét
- → regiszter

7. előadás

2013.01.11.

Regiszter, Regiszterpár

A	PSW	
B	C	B
D	E	D
H	L	H

7 0 7 0 — 8 bit, 16 bit

- Z zero
- CY carry (átvitel)
- S sign (előjel)
- P parity
- AC (részátvitel) Auxiliary carry

15 0

PC
SP

SP: stack utoljára elhelyezett adat címét mutatja

- Adatmozgató utasítások
  - regiszterből regiszterbe r ↔ r
  - regiszter és memória közti adatcsere r ↔ mem.
- Aritmetikai utasítások
  - összeadás / kivonás
- Logikai alapműveletek
  - AND, OR, NOT, XOR

- Vezérlés átadó utasítás
  - Elágazás (JUMP)
  - Szubrutinhívás (meg kell jegyezni, hova akarok visszatérni) (CALL)
  - Szubrutin visszatérés (RETURN)
- Stack műveletek
  - Adattárolás (PUSH)
  - Adatvisszakeresés (POP)
- Egyéb
  - I/O, ~~NOP~~ IT

Címzési módok

ADAT
- Közvetlen adat
  ↳az utasítás tartalmazza az adatot
- Regiszter
  ↳adat /operandus/ egy regiszterben helyezkedik el
- Közvetlen adatcímzés
  ↳utasítás ad a címet tartalmazza, ahol az adat van
- Regiszter indirekt adatcímzés
  ↳regiszter tartalmazza az adat címét (ill egy regiszterpár)

UTASÍTÁS
- Közvetlen utasítás
  ↳~~[illegible]~~ az utasítás tartalmazza a kív. utasítás címét
- Regiszter indirekt utasítás
  ↳regiszterpár tartalmazza a kív utasítás címét

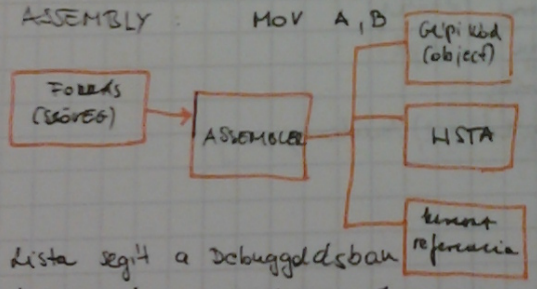
Gépi kód

Az az adat, amit utasításként értelmezve valamit csinál

B → A 78H

ASSEMBLER

gép utasításait szöveggel helyettesíti



lista segít a debuggolásban
Jelzi adatra a programon belül hol került utasítás

[címke:] op. kód [operandus] [; megjegyzés]
[MNEMONIC]

operandus:
- regiszter (A, B, C, D, E, H, L)
- regiszterpár (BD, H, SP)
- közvetlen adat
- Közvetlen cím

SZÁMJEGY KELL KEZDENI! (AFH → 0AFH)

Bináris: 01100100 B
oktális: 144 Q
Decimális: 100, 100D
Hexadec.: 64 H

Célszerű 0-ral feltölteni egy kód elejét, ha nem elég hosszú
1H → 01H

ASCII konstans 'A' 'AB'
Címke : START
Definiált konstans : DARAB
Kifejezés : DARAB+1

Assembler direktívák

utasítások a fordítónak

- Konstans definíció
  - DARAB EQU 5 — csak 1 lehet
  - SOK SET 1 — éppen az aktuális érvényes

    ⋮

    SOK SET 5
- Adatdefiníció
  - Byte : DB 10,0AH ..... 'A', 'Dalr'
  - Szó : DW 1234H, 12H, 'A', 'BC'

    LITTLE ENDIAN ábrázolási mód
    alacsonyabb címen alacsonyabb helyiértékű rész

34H	0
12H	1
12H	2
00H	3
41H	4
00H	5
43H	6
42H	7

- Helyfoglalás

  megadott konstans értéknek megfelelő helyet biztosít

  ADAT1 : DS 10 (! nem ismerjük a lefoglalt helyen !)

  ADAT2 : DS 10
- Feltételes fordítás
  - IF kifejezés
  - ELSE
  - ENDIF
- ASSEMBLER vége :
  - END

— o —

KÓD	ROM
KONSTANS	ROM
ADAT	RAM
STACK	RAM

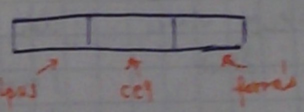
CSEG kódszegmens — CSEG [PAGE] 256-tal osztható címmel kezdődik
DSEG adatszegmens

(BSEG) ~ DSEG
(SSEG) stack-szegmens

ORG cím16
ASEG abszolút-sz., előre meghatározott címen elhelyezkedő adatokat határoz meg.

Hivatkozás.
PUBLIC DARAB
EXTERN DARAB

8085 ös utasítás max. 3 byte lehet
1. művelet kód
2.3. paraméter



↳ MOV r1, r2 = r1 ← r2 (8085 ösnél)
adatmozgatás művelete a flageket nem állítja
gépi ciklus = 1 byte átvitele a sínen

→ 1 Fetch ciklus (1. fázis cím kiadása, státusz
4 fázis 2. fázis R/W vezérlőjel
3. fázis WAIT?! (ready)
4. fázis adatátvitel elvégzése)

↳ MOV M, r [HL] ← r
MOV r, M r ← [HL]

→ 1 FETCH ciklus, 4 fázis
1 adatátvitel, 3 fázis 2/7

↳ MVI r, adat r ← adat
move immediate (közvetlen adat)

Ideális tart?: 2/7

↳ LXI rp, adat16 rp ← adat16
regiszterpár

LXI H, 1000H 5/10

21H
00
10

} LITTLE ENDIAN

→ 1 FETCH 4 fázis
2 adatátvitel 2x3
10

↳ LDA cím16 A ← [cím16] 1 FETCH 4
load accumulator 2 adatátvitel 2x3
↳ STA cím16 [cím16] ← A 1 adatátvitel 3
store acc. 16 bites 13

4/13

↳ LDAX rp A ← [rp]
16 bites regiszterpárokra utal
↳ STAX rp [rp] ← A

↳ rp B,D 1 FETCH
1 adatátvitel

2/7

↳ XCHG DE ↔ HL
regiszterpárok utasításait csökkenti meg

load direct 1/4
↳ LHLD cím16 HL ← [cím16] L ← cím16
↳ SHLD cím16 [cím16]←HL H ← cím16+1
store direct

1 FETCH 4
2 ADAT 2x3
2 adat 2x3
16 5/16

8085ös leghosszabb gépi ciklusa.

8. előadás

Aritmetikai utasítások

Összeadás

1. ADD r A ← A + r
   [illegible]
2. ADD M A ← A + [HL]
   [illegible]
3. ADI adat8 A ← A + adat8
   [illegible]

1/4

2. [illegible]

2/7

3. [illegible]

2/7

4. ADC r A ← A + r + CY
   [illegible]
5. ADC M
6. ACI adat8

Kivonás

1. SUB r A ← A − r
2. SUB M A ← A − [HL]
3. SUI adat8 A ← A − adat8
4. SBB r A ← A − r − CY
   [illegible]
5. SBB M
6. SBI adat8

INR r r ← r + 1
DCR r r ← r − 1

INR M [HL] ← [HL] + 1
DCR M [HL] ← [HL] − 1

BC, DE, HL, Stack mutató (SP) [illegible]

INX rp   rp ← rp+1
DCX rp   rp ← rp−1   } semmilyen flaget nem állít   1/6

DAD rp   HL ← HL+rp   3/10 csak a CY-t állítja
16 bites összeadó utasítás

BCD számokkal végzett műveletek

DAA

BCD értékké konvertál

1. Ha $A_{0..3} > 9$ (alsó 4 bit) vagy AC = 1 (segédátvitel): $A_{0..3} = A_{0..3} + 6$
2. Ha $A_{4..7} > 9$ (felső 4 bit) vagy CY = 1 (átvitel): $A_{4..7} = A_{4..7} + 6$

1/4

pl.

```
  29          0010 1001
+ 89        + 1000 1001
----        -----------
  B2  AC=1    1011 0010     AC=1
                  + 0110    CY=1
                    1000
            + 0110
            -----------
            1 0001 1000     CY=1

             (1)  1   8
              |
              CY
```

BCD: 29 + 89 = 118

pl.

```
BIN                                   BCD
  87        1000 0111                  87
+ 94      + 1001 0100                + 94
----      -----------                ----
 11B     (1)0001 1011                 181
                + 0110
          -----------
         (1)0010 0001
          + 0110
          -----------
         (1)1000 0001

            1  8  1
```

Logikai utasítások

ÉS:

ANA	r	A ← A AND r	1/4	állítja a flageket: CY (átvitel) törli, többit állítja az eredménynek megfelelően
ANA	M	A ← A AND [HL]	2/7	
ANI	adat₈	A ← A AND adat₈	2/7	

vagy:

ORA	r	A ← A OR r	1/4
ORA	M	A ← A OR [HL]	2/7
ORI	adat₈	A ← A OR adat₈	2/7

kizáró vagy:

XRA	r	A ← A ⊕ r	1/4	akkumulátorba 0-t tölthetünk a segítségével; CY, AC = 0, zéró flag = 1, előjel flag = 0, paritás flag = 1
XRA	M	A ← A ⊕ [HL]	2/7	
XRI	adat₈	A ← A ⊕ adat₈	2/7	

Compare utasítás

CMP r   FLAGEK ← (A − r)   1/4

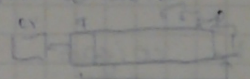
CMP M   2/7

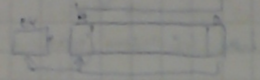
CMP adat₈   2/7

Z = 1   A = r   8 bites, azonos előjelű számokra használható

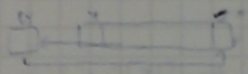
CY = 1   A < r

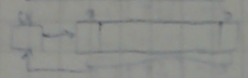
Rotate / Forgató utasítás

RLC (left) 

RRC (right) 

9 bites forgatás

RAL 

RAR 

CMA $A \leftarrow \bar{A}$
1-es komplemensét képezi az akkumulátornak (invertálás)

STC   CY = 1
carry értékét 1-et ír

CMC $CY \leftarrow \overline{CY}$
carryt 0-ból 1-be vagy 1-ből nullába állíthatom

1/4

·Programszámláló befolyásolása - [illegible]

Elágazások

JMP $cím_{16}$ PC ← $cím_{16}$ 3/10
programszámlálót tölti közvetlen adattal

J_{cc} $cím_{16}$ PC ← $cím_{16}$, ha cc igaz   3/10
feltételes elágazás

Z	Z = 1	zero
NZ	Z = 0	
C	C = 1	carry
NC	C = 0	
M	S = 1	előjel flag értéke alapján elágazás
P	S = 0	
PE	P = 1	paritás flag alapján
PO	P = 0	

ha cc nem igaz: 2/7

PCHL   PC ← HL   1/6
feltétel nélküli, indirekt elágazás
regiszterpárból megtöltjük a programszámlálót

Szubrutin műveletek

[illegible] stack műveletek [illegible]

CALL $cím_{16}$
$PC \leftarrow cím_{16}$
$[SP] \leftarrow PC$
$SP \leftarrow SP - 2$

5/18

PC-H SP-1
PC-L SP-2

1 FETCH (6)
2 db olvasási gépi ciklus (2×3)
2 írás (2×3)

18 [illegible]

C_{cc} $cím_{16}$ $PC \leftarrow cím_{16}$ ha cc igaz 5/18

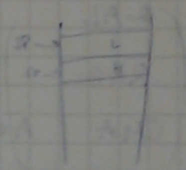
ha cc nem igaz 2/9

RETurn

szubrutinból visszatérés

RET $PC \leftarrow [SP]$ $PC\text{-}L \leftarrow [SP]$ 3/10

$SP \leftarrow SP+2$ $PC\text{-}H \leftarrow [SP+1]$



R_{cc} $PC \leftarrow [SP]$ ha cc igaz 3/12

feltételes visszatérés

cc nem igaz 1/6

RST_n $n = 0..7$

restart

3/12

[illegible] 2 írás (2×3)

Stack műveletek

PUSH rp stack ← rp
POP rp rp ← stack
} 3/12

16 bájtos [illegible] a stackbe, [illegible]

H - high
L - low

rp B, D, H
PSW

PUSH:
$[SP-1] \leftarrow rp\text{-}H$
$[SP-2] \leftarrow rp\text{-}L$
$SP \leftarrow SP-2$

POP:
$rp\text{-}L \leftarrow [SP]$
$rp\text{-}H \leftarrow [SP-1]$
$SP \leftarrow SP-2$

XTHL $[SP] \leftrightarrow HL$ 5/16

stack tetején lévő adatot kicseréli a HL regiszterben lévővel

FETCH 4
[SP] → W 3
[SP+1] → Z 3
[SP+1] ← H 3
[SP] ← L 3

9. előadás

2013. 03. 18.

8085-ös proc. egyéb utasításai

1. IN $cím_8$ 8 bites címet beolvas az akkumulátorba
2. OUT $cím_8$

1. IN 20H → 20 hexa cím
   2 + 1 gépi ciklus, 10 fázis   3/10
   (végleges adatátvitel)

1. HLT = MOV M M!
   halt állapotba kerül az utasítástól a processzor
   /ebből kilökni: másik buszmaster } képes /
   reset
   külvilág kérhet megszakítást

1. NOP = 0 byte =   1/4
   no operation, időhúzási célra alkalmas
   10 művelet után használatos, !

RIM EI
SIM DI   1/4 } processzorba integrált áramkör szabályozása
interrupt-ok

Írjuk ki, hogy HELLO!

```
DSEG
TEXT: DB 'HELLO', Ø

EXTRN STROUT
CSEG
   :
   LXI H, TEXT
   CALL STROUT
   :
```

Szubrutin: (STROUT PGV.:)

```
EXTRN CHROUT
PUBLIC STROUT
CSEG
STROUT: MOV A, M
        ORA A      ← [flagek beállítása vagy/és kapcsolatba önmagával]
        RZ
        CALL CHROUT
        INX H
        JMP STROUT
```

CHROUT PGV.:

```
PUBLIC CHROUT
TXRDY  EQU 00000001B
USARTD EQU ØOH
USARTC EQU USARTD+1

CSEG
CHROUT: PUSH PSW
TXWAIT: IN USARTC
        ANI TXRDY
        JZ TXWAIT
        POP PSW
        OUT USARTD
        RET
```

„80" adatregiszter
„81" kontrolregiszter
10 cím

Célszerű 1 helyen definiálni a konstansokat, egyszerűbb a módosítás

— o —

```
                ORG  1000H
1000 → LXI  SP, 8000H  → 31
1003 → MVI  A, 2       → 3E
1005 → STA  8000H      → 32
1008 → ORA  A          → B7
1009 → JZ   TOVA       → C2
100C → CALL SZUB       → CD
100F → TOVA : HLT      → 76
1010 →        NOP      → 00
1011 → SZUB : LXI H, 8000H → 21
1014 → CIKL : DCR M    → 35
1015 →        JZ       → C8
1016 →        CMA      → 2F
1017 →        JMP CIKL → C3
```

cím	adat	FORRÁS	RD/WR	REG, MEM, FLAG
1000	31	PC	RD	
1001	00	PC	RD	
1002	80	PC	RD	SP = 8000
1003	3E	PC	RD	
1004	02	PC	RD	A = 2
1005	32	PC	RD	
1006	00	PC	RD	
1007	80	PC	RD	
8000	02	cím$_{16}$	WR	[8000] = 2
1008	B7	PC	RD	CY = 0 Z = 0 S = 0
1009	C2	PC	RD	
100A	07	PC	RD	
100C	CD	PC	RD	
100D	11	PC	RD	
100E	10	PC	RD	PC = 1011
7FFF	10	SP	WR	
7FFE	0F	SP	WR	SP = 7FFE
1011	21	PC	RD	
1012	00	PC	RD	
1013	80	PC	RD	HL = 8000
1014	35	PC	RD	
8000	02	HL	RD	
8000	01	HL	WR	[8000] = 01
1015	C8	PC	RD	
1016	2F	PC	RD	A = FD ← 2 kiegészítő
1017	C3	PC	RD	
1018	14	PC	RD	
1019	10	PC	RD	PC = 1014

Folyt.:

cím	adat	FORRÁS	R/W	REG, MEM, FLAG
1014	35	PC	RD	
8000	01	HL	RD	
8000	00	HL	WR	[8000] = 00, Z = 1, S = 0
1015	C8	PC	RD	
7FFE	0F	SP	RD	
7FFF	10	SP	RD	SP = 8000, PC = 100F
100F	76	PC	RD	

(return zero utasítás → 1015)

HALT

Egybites I/O (be ill. kimenete)

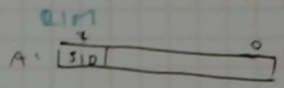
2 lába a PORT-ja pocu

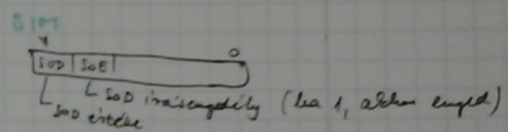
SID Serial Input Data

SOD Serial Output Data

vezérlés: RIM / SIM

read interrupt mask





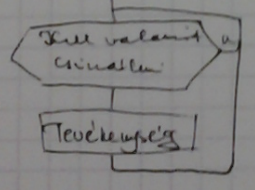
Program, ami átmásolja a SID bemenetet a SOD kimenetre (negálja)

```
SID MSK   EQU   80H
SOD SET   EQU   40H
   :
RIM
ANI   SIDMSK
ORI   SODSET
SIM
```

VAGY → ANI SIDMSK → XRI SIDMSK (a kizáró vagy művelet segítségével 1 bitet negálunk meg)

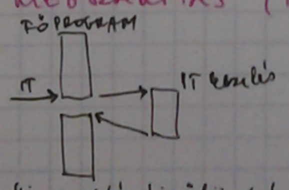
1. PROGRAMMAL ELLENŐRZÖTT KÉSZENLÉT

„Kopogással ne zavarja a munkát!"



☹:
- Túl sok lehetséges input esetén bonyolult lehet.
- Eseményig kezelése sokáig tart

2. MEGSZAKÍTÁS (INTERRUPT)



Hardverből előidézett szubrutin-hívás

„Kopogtatás módjai":
- ÉLVEZÉRELT megoldás ⇒ flipfloppal tároljuk
- SZINT Érzékeny interrupt (Folyamatosan csengő)
- SPEC. ÉRZ: ÉL + SZINT

! Aktuális utasítás fejeződjön be!

! Biztosítani kell, hogy az interruptot csak adott időben fogadhassa be → maszkolás

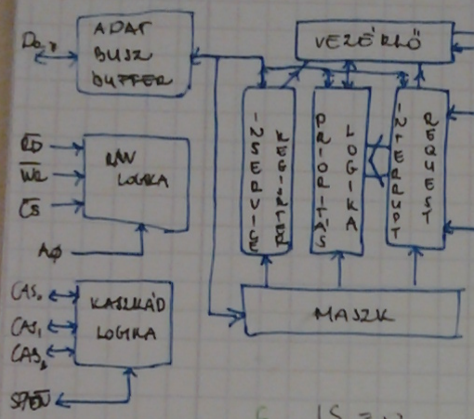
! Prioritás is fontos

! Nem maszkolható interrupt (fontos) nem tartható fel a normál IT mechanizmussal

2013.03.28.

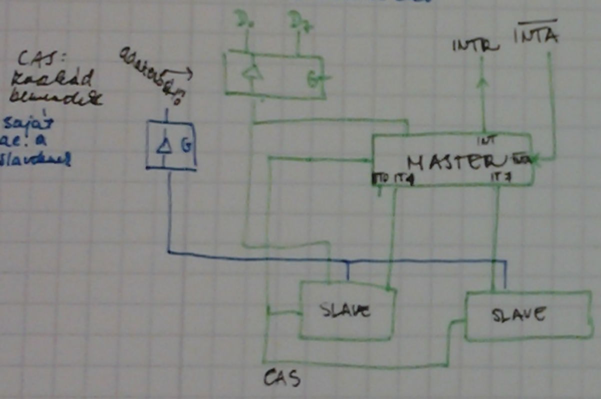
11. előadás

8259-es blokkvázlata



1. IT_a 0→1 IR_a → 1
2. Prioritás + MASZK → INT=1
3. Ha a CPU-n fennállnak a feltételek: $\overline{INTA}=0$
4. CALL kódja (ØCDH) a slave
   $IS_a=1$ $IR_a=0$
5. A következő 2 INTA ciklusban szubrutin cím alsó és felső byte
6. $IS_a=0$
   - ↳ AEOI Automatikus end of interrupt
   - ↳ Programból - felhasználói paraméter függvényében az alacsonyabb prioritású interrupt

Kaszkádosítás



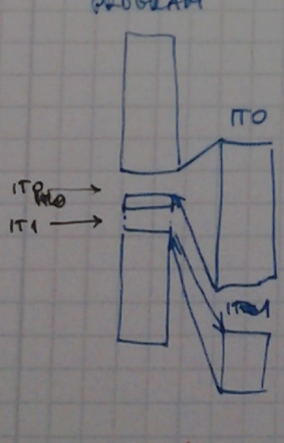
MASTER fő.: Call utasítást felhasználva a slave

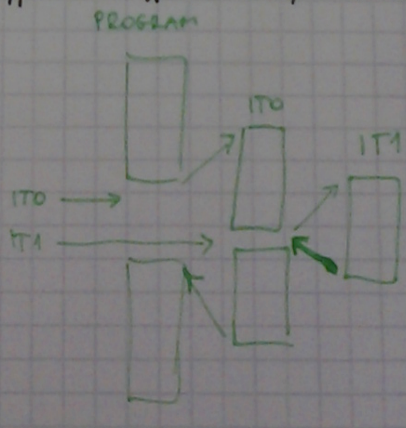
G = INTA

$G = SP/\overline{EN}$

Amennyiben max 8 db slave, az IT0-ra tilos slavet kötni!
Az interruptok egymás után kerülnek végrehajtásra.

← De olyan IT rendszerre lenne szükség, ahol egymásba ágyazott interruptot valósítunk meg

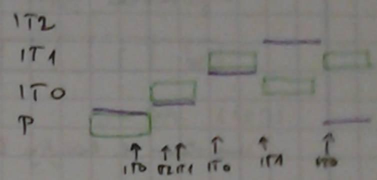




ITRUTO: PUSH
:
POP
STI
RET

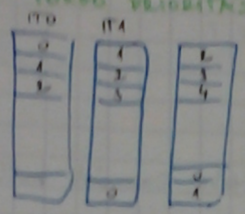
AEOI
használható ✓

ITRUTO: PUSH
:
CLI
:
POP
EOI
RET



Az alacsony prioritású IT nagyon későn jöhet
csak létre a fix prioritás esetén
↳ kiéheztetés jelensége

Megoldás:
FORGÓ PRIORITÁS

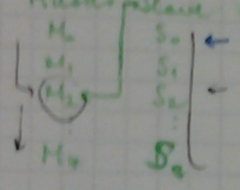


- 8259-es támogatja

Biztosítja, hogy előbb (utóbb) ∀ IT kezelésre tudjon jutni

A EOI oktató vizsgásunk, mert saját magukat is megszakíthatják
Csak ⅄ esetben jó az AEOI használata

Master - Slave prioritás: M0, M1, S0, S1, S2 .. S7, M3 ... M7



Speciális rögzített prioritás:
Master továbbítja mindig az IT-t a Slave-ről
Master EOI csak akkor következhet be, ha a Slave regiszterben nincs kiszolgálásra váró Slave
(inservice reg.-ben)

Kívülről 2 regisztert látunk

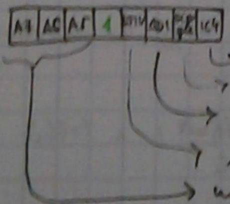
A0 = 0 ICW 1 vagy OCW 2/3

A0 = 1 ICW 2,3,4 vagy OCW 1

Inicializálás

- szubrutin tábla címe
- szint / él-re érzékeny legyen a bemenet
- MASTER SLAVE konfiguráció
- EOI
- Bufferelés
- μP (mikroprocesszoros rendszerben vagyunk? milyen)

ICW = 1 A0 = 0
D4 es bit 1-es



ICW2 A0=1

ugrótábla felső byte

| A15 | | | | | | | A8 |

ICW3 A0=1

master-sl. konfig-hoz:

→ MASTER / bitminta, hogy hol vannak slave-ek

→ SLAVE / melyik master bemenetre csatlakozik

ICW4 A0=1

| | | | | M/S | BUFF | AEOI | µP |

→ mikroprocesszor 85 v 86 ?
→ AEOI
→ Bufferelt üzemmód
→ MASTER / SLAVE
→ Spec. rögzített prioritású üzemmód beállítása

V.T.
11

8259: nincs RESET láb!

Maszkregisztert állító parancs: OCW1 A0=1

OCW2 A0=0

prioritást, EOI-t állítják vele

| | | | 0 | 0 | C2 | C1 | C0 |

D4 D3

PARANCS — sorszám, hányas számú IT-re vonatkozik a kiadott parancs

PARANCSOK:
- EOI - éppen a legmagasabb prioritásút fejeztem be
- Specifikus EOI - hányas sorszámú IT-t fejeztem be
- EOI és forgatás - akit befejeztem, az legyen a legalacsonyabb prioritású
- AEOI forgatás engedély / tiltás (marad a fix prioritás)
- specifikus EOI és forgatás
- Prioritás beállítás - a legalacsonyabb prioritású IT-t tudom beállítani

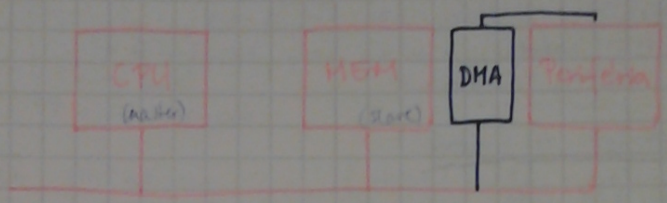
OCW3

IS / IR regiszter olvasás

POLL — ki tudjuk olvasni az IRR regiszterből, hogy ki válaszolná a legmagasabb prioritáson

Speciális maszkolás — akit maszkolunk → tiltjuk, a többit engedjük, a prioritástól függetlenül

DMA - Direct Memory Access



1 Master (CPU)
1 Slave (Memória)

Forrás: CPU / MEM
Cél: MEM / CPU

Periféria ↔ Memória
Ki legyen a master?

Blokkos mozgatásra pl

Másol:

```
CIKL:
    IN STAT          10
    ANI VAN ADAT      7
    JZ CIKL           7
    IN ADAT          10   (kiolvasom az adatot)
    MOV M,A           7
    INX H             4
    DCR C             4
    JNZ CIKL         10
```

Mennyi idig tart?

→ 45 fázis 1 byte mozgatás
↳ 25 μsec 1 byte átvitele

Más megoldás, IT-vel

```
    JMP ITRUT        10
ITRUT:
    PUSH PSW         12
    PUSH H           12
    LHLD BUFFCIM     16
    IN ADAT          10
    MOV M,A           7
    INX H             4
    SHLD BUFFCIM     12
    POP H            10
    POP PSW          10
    EI                4   (IT engedélyezés)
    RET              10
```

117 fázis 1 byte mozgatás
↳ 37 μsec

FORRÁS PERIFÉRIA
CÉL MEMÓRIA
MASTER 1 DMA vezérlő
SLAVE 2 : MEM, Periféria

(Gyors) Adatátvitel a sínen

CIM - MEMÓRIA CÍM
vezérlés : MEMÓRIA - WRITE parancsot kap } egyszerre
PERIFÉRIA - READ parancsot kap }

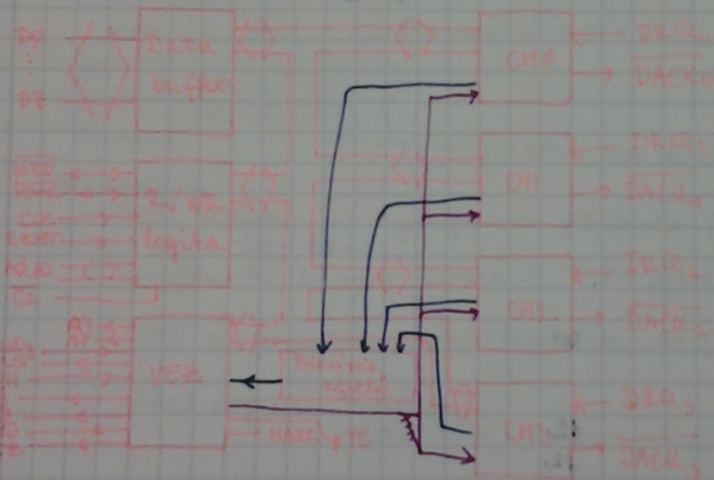
(elsőnél nem kell puffert építeni)
Szekvenciális R/W ciklusok gyik kellenek

Periféria kivezetésekor

DRQ - DMA igény jelzése
DACK - elfogadás

Adatátvitel időzítés $\overline{IORD}$, $\overline{IOWR}$

8257 Működése

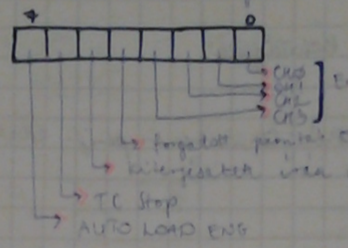


SW-modell

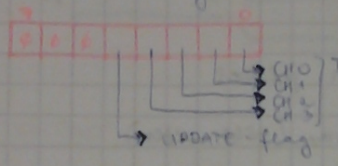
→ 1 db 16 bites CIM regiszter
→ 4 db 14+2 bites Terminal Count regiszter

TC15	TC14	
0	0	visszatati ciklus
0	1	MEM WRITE
1	0	MEM READ
1	1	Tiltott

→ 1 db MODE SET regiszter

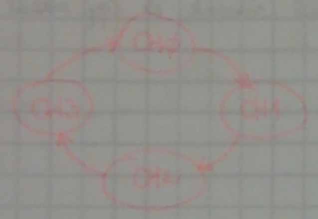


→ 1 db STATUS reg. (olvasás estén státuszinformációk)



Prioritás

forgatott prioritás
CHØ alapból a legmagasabb prioritású



Kiszolgált	CHØ	CH1	CH2	CH3
magas	CH1	CH2	CH3	CHØ
	CH2	CH3	CH0	CH1
	CH3	CHØ	CH1	CH2
alacsony	CHØ	CH1	CH2	CH3

DMA vezérlőbe first - Last flipflopot építettek be

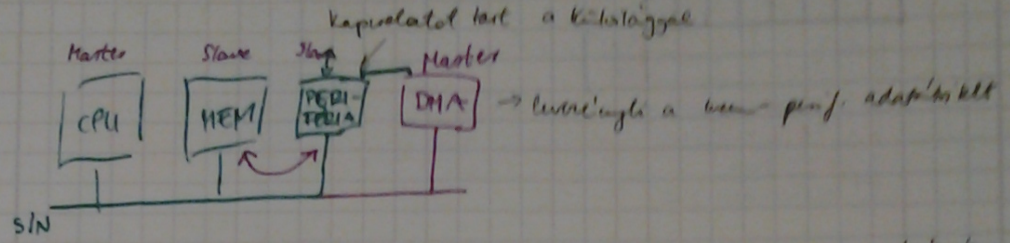
Belső regiszterek elérése

A3	A2	A1	AØ	F/L		
0	0	0	0	0 1		CHØ cím reg.
0	0	0	1	0 1		CHØ TC
0	0	1	0	0 1		CH1 cím reg.
0	0	1	1	0 1		CH1 TC
0	1	0	0	0 1		CH2 cím reg.
0	1	0	1	0 1		CH2 TC
0	1	1	0	0 1		CH3 cím reg.
0	1	1	1	0 1		CH3 TC
1	0	0	0	0	W	MODE SET
				–	R	STATUS

8237-es DMA vezérlő már képes 2 ciklusú átvitelre, memóriából memóriába is tud másolni, a ~~regiszterek~~ AUTO LOAD üzemmód kihasználhatók.

2013. 04. 11.

13. előadás



DMA cikl.: speciális, mert egyszerre kell a sínen egy írás és olvasó jelnek lennie

⇓

Szeparálják a mem. és IO vez. jeleket

Soros periféria → nagy távolságra (több méter)

1 adatátvitel - 1 bit

Párhuzamos periféria

1 adatátvitel - 8 bit

(nyomtatók régen párh., de ma már sorosak és így sokkal gyorsabbak)

Párhuzamos adatátvitel

Adatátvitel vezérlése

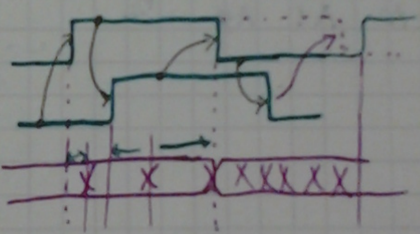
1. Nincs (kapcsoló, lámpa)
   - INPUT: pillanatnyi érték
   - OUTPUT: következő értékig megmarad LATCH

2. Handshake jelek
   ADÓ - VEVŐ

ADÓ: DATA_READY

VEVŐ: ACK

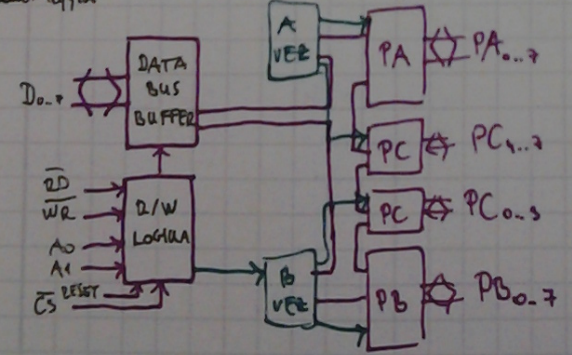
ADAT:



A 8085-öshöz kifejlesztett áramkör: 8255-ös

C port: 4 bitenként állítható, hogy be- v. kimenet legyen

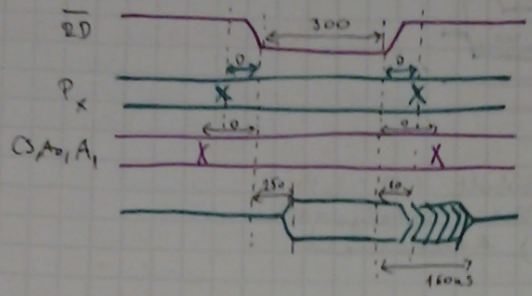
A és B port - be és kimenetként csoportosan programozható



8255-ös 3 üzemmódja:

- MODE 0 : Egyszerű I/O — 300 ns feletti read pulzus

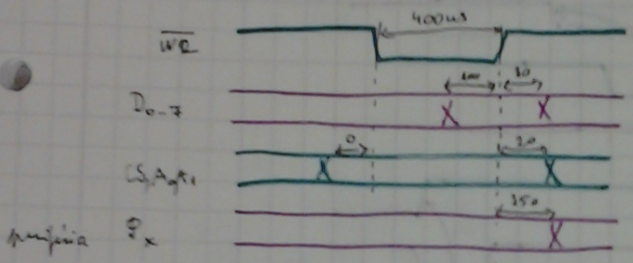
INPUT



OUTPUT

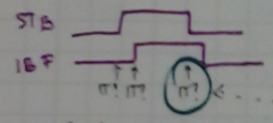
$T_{wp} = (15+N)T \cdot 80ns = 400ns$ (wait)

$T = 320 ns$



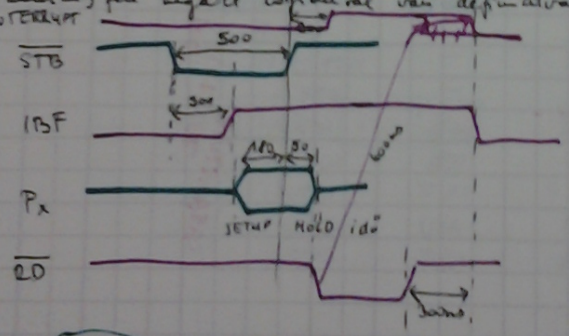
- MODE 1 Strobolt I/O

INPUT
ADÓ : Periféria — DATA-READY : STB (Strobe)
VEVŐ : 8255 — ACK : IBF (input buffer full)



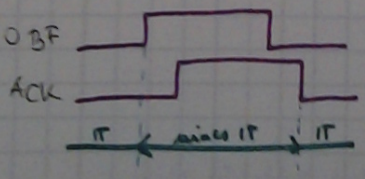
Kol keljen az interruptot?
Akkor utánt a strobe jel.

STB jel a kiolvasás felé' negatív logikával van definiálva.

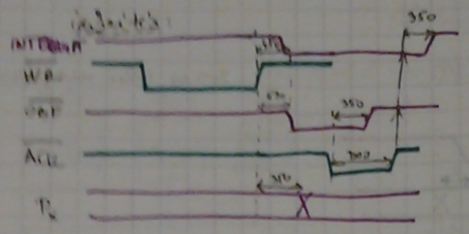


OUTPUT

ADÓ : 8255 — DATA_READY : OBF (output buffer full)
VEVŐ : PERIFÉRIA — ACK : ACK

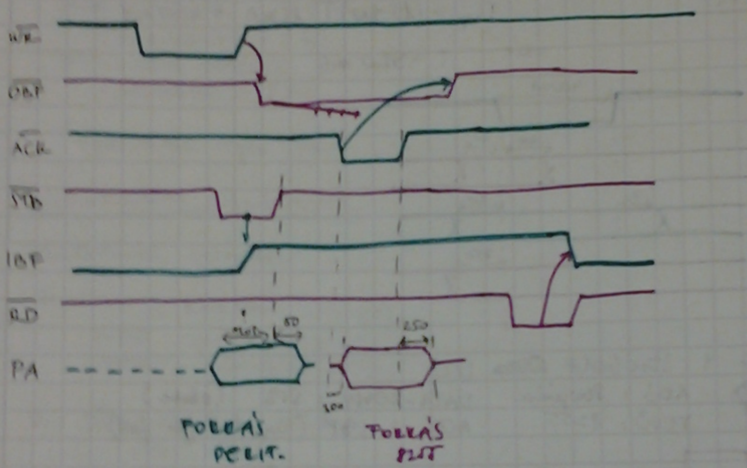


IT? → ha kész az adat fogadására a kiolvasó



Írás: felfutó élre

– MODE 2 – kétirányú strobolt I/O („5/4")
hardver handshake jelek szabályozzák
csak az A port (PA) használható



2013.04.15

14. előadás

PC	INPUT	OUTPUT	MODE2	STÁTUSZ(C)
0	INTR B	INTR B	–	INTR B
1	IBF B	OBF B	–	IBFB/OBFB
2	STB B	ACK B	–	INTE B
3	INTR A	INTR A	INTR	INTR A
4	STB A	–	STB	INTE A in
5	IBF A	–	IBF	IBF A
6	–	ACK A	ACK	INTE A out
7	–	OBF A	OBF	OBF A

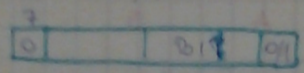
C PORT HARDVER BEVÖRÖSE

Gport alsó 3 bitje

PA
PB
PC
CONTROL NEM olvasható, csak írható.
→ üzemmód
→ Bit SET / RESET

A, B, C: irány (C portnál 4 bitenként adható meg)
üzemmód
A: 0, 1, 2 B: 0, 1

Bit SET/RESET



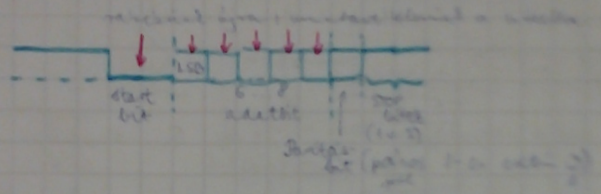
Soros adatátvitel

- 2 vezeték
- bonyolultabb hardver megoldás
- léptető/shiftregiszter alkalmas a soros - párh. átalakításra
- Előre megegyezett bitidő (vagy frekvencia) Baud Rate

tipikusan 75 bit

75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

- Aszinkron - karakteres kapcsolat
- Szinkron



9600, 8, N, 1

1200, 7, E, 2

Előny: 10 bit alatt ...

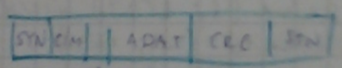
Hiba: Parity Error

Frame Error (...)

Break karakter (...)

Logikai átvitel szinkron módon

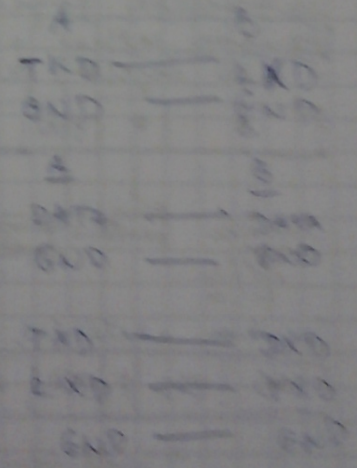
SDLC



7EH

01111110

Bit Stuffing

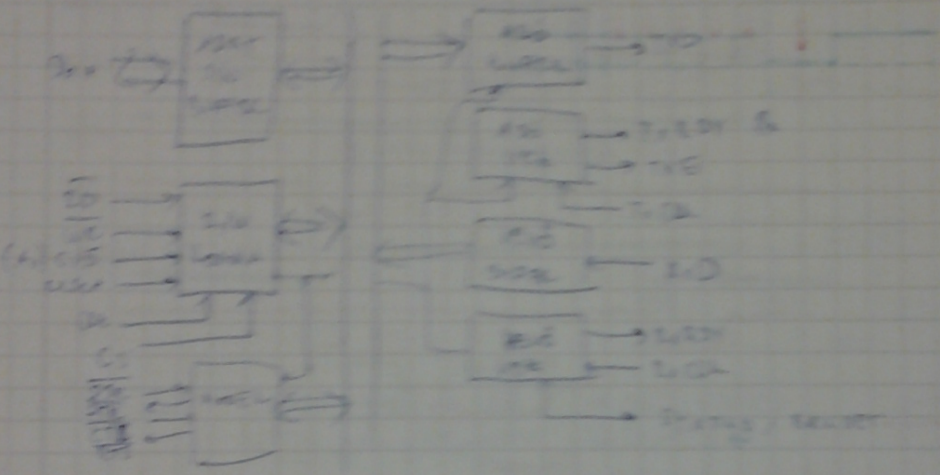


RS232 a szabvány

feszültségszinteket (±15V)

írja elő

8251



08.04.22.

1	0	PEN	1	0

SCS	ESD	0	PEN	1	0	0

Command instruction

EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN

- EH → baud üzemmód
- IR → belső reset (következő parancs mode instruction lesz)
- RTS → RTS
- ER → hibák törlése
- SBRK → break küldés
- RxE → vétel engedélyezés
- DTR → DTR kimenet 1 → DTR = 0
- TxEN → adás engedélyezés

Status

DSR	SYNDET	FE	OE	PE	TxE	RxRDY	TxRDY

- DSR → DSR bemenet értéke
- SYNDET → SYNDET kiolvasható értéke
- FE → keret hiba
- OE → túlfutási hiba
- PE → Parity error: paritás hiba
- TxE → az utolsó karakter is elküldve
- RxRDY → vételi puffer tele
- TxRDY → transmit ready: köv. karakter tölthető

8251-es inicializálása ...

```
USARTD   EQU  80H
USARTC   EQU  USARTD+1
                              (reset)
belső reset →    USARTR   EQU  01000000B
USARTM   EQU  01001110B ; 16, 8, N, 1
                 (adatok száma)   (16-tal osztva)
inicializálás    USARTI   EQU  00110111B ;
```

Aszinkron eset

```
XRA A
OUT USARTC
OUT USARTC
OUT USARTC
MVI A, USARTR
OUT USARTC ← command parancs/regiszter
MVI A, USARTM
OUT USARTC
MVI A, USARTI
OUT USARTC
```

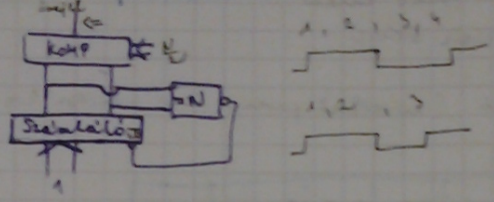
Szinkron üzemmód

```
USARTS   EQU   10001100B
SYNCHAR  EQU   7EH
USARTSI  EQU   10110111B
[
MVI A, USARTR
out USARTC
MVI A, USARTS
! out USARTC
! MVI A, SYNCHAR
OUT USARTC
MVI A, USARTSI
out USARTC
```

Processzor órajele:

3,072 MHz → osztóval → 320 × 9600
8251: 16
külső osztó: 20

Programozható frekvenciaosztó

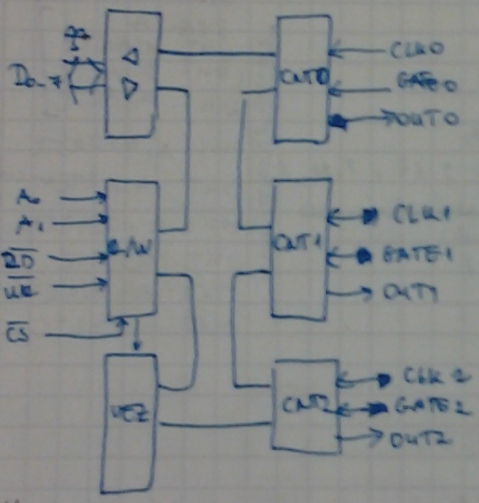


8253

3 db 16 bites számláló
bináris, ill. BCD számlálásra alkalmas
(16 bit) (4 dek)
Lefutóél vezérelt
Lefelé számlál

$T_{CLK} = 380\,ns$ (2,6 MHz) ⇒ a vezérlőjelet is ki kell osztani, azt választja a 8253-asra

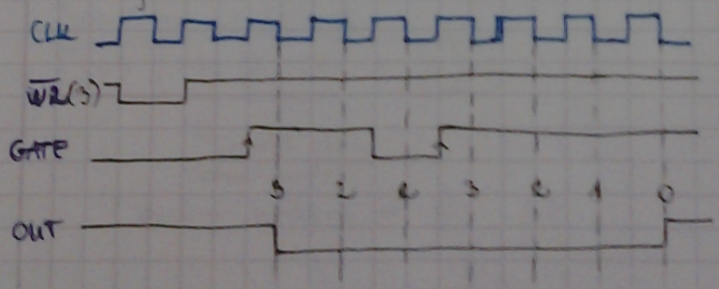
Blokkvázlat:



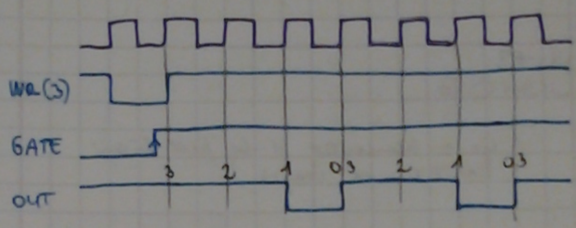
üzemmód

0 MODE 0
IT, ha a számláló lejárt

1 MODE 1
adott szélességű impulzus előállítására (monostabil)
újraindítható ←



2 N-ed osztó



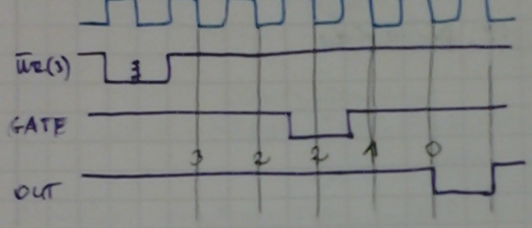
3. Négyszögjel generátor

Kitöltési tényező:

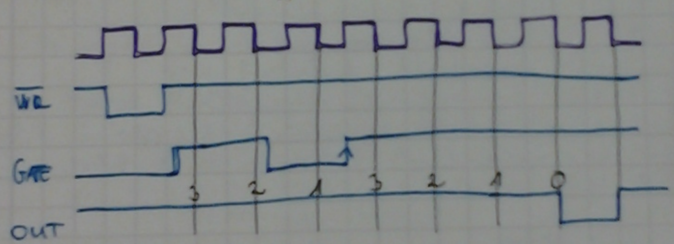
- N páros : 50%
- N páratlan (2k+1)

$\frac{k+1}{2k}$

4. jelzés adott idő múlva (szoftver SW triggerelt strobe)



5. Hardware triggerelt strobe



4 Regiszter

A_1	A_0	
0	0	CNT0
0	1	CNT1
1	0	CNT2
1	1	CONTROL

Egy átmeneti regiszteren keresztül tudjuk kiolvasni a számláló tényleges értékét

Vezérlő szó

S	C	R	L	.	.	.	BCD

- S C: kiválasztás számláló
- R L: Töltés/olvasás
  - 11: LSB, MSB
  - 10: LSB
  - 01: MSB
  - 00: Tárolás
- üzemmód:
  - 000: 0
  - 001: 1
  - 010: 2
  - 011: 3
  - 100: 4
  - 101: 5
- BCD: BCD / BIN mód állítás (BCD = 1 / BIN = 0)

9600 → 1536 MHz → 160

```
CNTRØ   EQU  90H
CNTRC   EQU  CNTRØ +3
CNTRCW  EQU  00100110B

MVI A, ~~CNTRC~~ CNTRCW     (Gate-bemenetet 1-be kell kötni
OUT CNTRC                    CNTRØ-ra órajel)
MVI A,10
OUT CNTRØ
```

2013.04.25.

16. előadás

Diasor

2013.05.06.

18. előadás

Memóriaáramkörök diasor

2013.05.13.

19. előadás

Gyakorlás

① 7 bites 2es kompl számokat kell összehasonlítani 4 bites 2 db komparátorral

Kettes kompl számok komp.

- Azonos előjel esetén a komparátor helyesen működik
- Különböző előjel esetén a poz szám a nagyobb, de a poz számok kódja kisebb, mint a negatív számé

Lehetőségek

- Komb hálózat | Ha különböző előjelek esetén a komparálási eredményt fordítjuk meg
- Negáljuk a számok legnagyobb helyiértékű bitjét és
- Cseréljük fel az operandusok legnagyobb helyiértékű bitjét

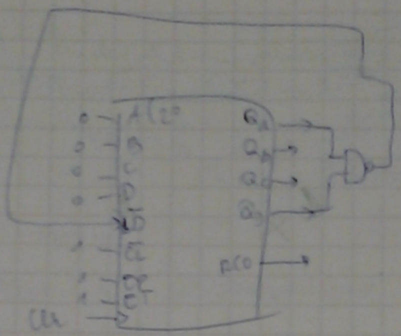
A komp. 8 bites, az operandusok 7 bitesek ⇒ 8. bites azonos értéket kell kötni

Kaszkádosítás

② 4 bites szinkron bináris számláló, aszinkron törlés (clear) és szinkron betöltés (load), csináljunk belőle BCD számlálót

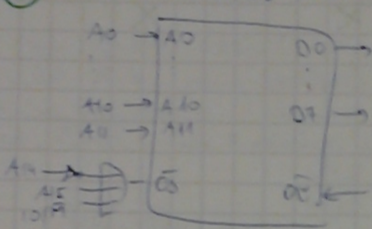
Feladat

- A számlálót módosítjuk a következő periódusban, ha az értéke 9.
- A számlálónk aszinkron clear funkcióval rendelkezik
- Használjuk a load bemenetet, 0-t töltünk a számlálóba
- Komparálás helyett használjunk NAND kaput $LD_9 = Q_D Q_A$



Ha 0-12-ig számláltunk volna, Q_2, Q_3-at kell összekötni
14 esetén 3 bemenetet kell összekötni

③ EPROM



Memóriaáramkör kapacitása?
Hány byte-ot tudok megcímezni 12 címvezetékkel
2^{12} = 4 kbyte

Chip select
A15 = 0
A14 = 1
A13 = A12 = x

Címtartomány
0100..0000 - 0111..1111
4000H → 7FFFH -ig tölt meg a memória
4000, 5000, 6000, 7000 címről elérhető

④ Assembly

Melyik memóriacímek milyen tartalmat kapnak a 8085-ös mikroprocesszorra?

```
ADAT 1   EQU  100
         ORG  1000H
ADAT 2 : DS   1
ADAT 3 : DB   ADAT 1
         ORG  5000H
ADAT 4 : DW   ADAT 1
```

Cím (hexa)	Adat (hexa)
1000	XX
1001	64 (decimális 100)
5000	64
5001	00

little endian: alacsonyabb helyiérték elöl

⑤ Teljes megszakítás, mikorra kerül sor?

	Engedélyezésnél	Tiltásnál
EI utasítás	X	
DI utasítás		X
TRAP megszakítás érvényre jutása		X
HOLD kérés elfogadása	—	—

HOLD: nincs rá hatása

⑥ Melyik jelzések / utasítások jönnek ki a 8085-ös proc. átmenetileg / véglegesen a HALT állapotból

	Átmenetileg	Véglegesen
HOLD kérés elfogadása	X	
RST 0 utasítás a program memóriában	—	—
RESET		X
TRAP megszakítás érvényre jutása		X

(7) 2 busz felép. rendszer

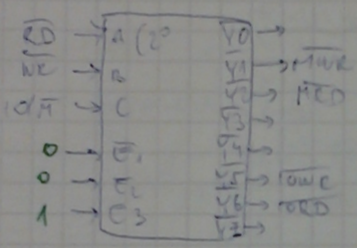
1. CLK, Reset, $\overline{RST5.5}$, $\overline{MRD}$, $\overline{MWR}$, $\overline{IORD}$, $\overline{IOWR}$, $\overline{ACK}$, $\overline{HOLD}$, HLDA

2. – " – $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$, READY, – " –

melyik buszon melyik műveletek láthatók kívül?

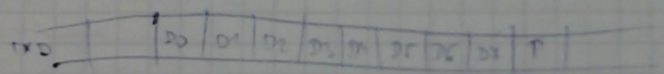
művelet	1 busz	2 busz
DMA átvitel	X	[illegible] jel
INTA gépciklus	nincs interrupt jel	nincs INTA ... jel
Memória írás vagy olvasás	X	X
I/O egység írás vagy olvasás	X	X

(8) DMA vezérlőt tartalmazó rendszerben 8085-ös proc $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$ jeleinek átalakítása szükséges a memória és az I/O portok illesztéséhez. Használja a dekódert, állítsa elő: $\overline{IORD}$, $\overline{IOWR}$, $\overline{MRD}$, $\overline{MWR}$



	IO/$\overline{M}$	$\overline{WR}$	$\overline{RD}$	
	C	B	A	
2	0	1	0	$\overline{MRD}$
1	0	0	1	$\overline{MWR}$
6	1	1	0	$\overline{IORD}$
5	1	0	1	$\overline{IOWR}$

(9) aszinkron üzemű UART kimenet (TxD) jelalak elsőként D0 bit továbbítása



11010100 = D4 H páros db száma 1-es (4 db)

0 → 1 → páratlanra egészíti ki ⇒ páratlan paritás

(6) Hány db és milyen bemenetű komb. hálózat alakítható ki egy Xilinx Spartan 3 FPGA áramkör egy CLB egy slice-ában (egy elemi cellájában)? Hány soros állapottáblát lehet megvalósítani a fenti keretben (elemet)?

1 SLICE
- 2db LUT (4 bemenet, 1 kimenet)
- 2db multiplexer
- 2db flip-flop

→ 2db 4 bemenetű k. h. (2 LUT)
  1db 5 bemenetű k. h. (2 LUT + 1 MPX)

→ 4 soros állapottábla (2 ff = 2 szek. változó = 4 áp.)

Bengió

① Aritmetikai áramkör, 4 bites kettes komplemensben ábrázolt számokkal (X, Y) a következő műveletet hajtja végre

$Z = 4^{*}X - Y$ ha $X \le Y$

$Z = 4^{*}Y - X$ ha $X > Y$

Ahol Z 8 bites kettes kompl. ábr. szám.

Használja ezeket: 2 db 4 bites teljes összeadó, 1 db 4 bites komparátor, 2 db 4 bites 2/1 multiplexer

- kettes kompl. komparálás
- 4-gyel szorzás
- kivonás

} tudni kell

← bitenként negált +1

blokkvázlat: funkcionális elemekből hogy hogyan épül fel

4 bites komp.

② 8085-ös proc. az alábbi ass

```
        ORG   800H
INDUL:  CALL  SRUT     0800
        INR   A        0803
        HLT            0804
SRUT:   INR   M        0805
        JZ    KILEP    0806
        DCR   A        0809
KILEP:  RET            080A
        HLT
        END
```

Cím (hexa)	Adat (hexa)	Forrás	RD/WR	megváltozott regiszter értékek
0800	CD	PC	RD	
0801	05	PC	RD	
0802	08	PC	RD	
5FFF	08	SP	WR	
5FFE	C3	SP	WR	SP = 5FFE
0805	34	PC	RD	
5FFE	03	HL	RD	
5FFE	04	HL	WR	5FFE = 04, Z = 0 (CY = ? előjelflag = 0)
0806	CA	PC	RD	
0807	0A	PC	RD	~~A = 00, Z = 1~~
0809	3D	PC	RD	A = 00, Z = 1
080A	C9	PC	RD	
57FE	04	SP	RD	
5FFF	08	SP	RD	SP = 6000
0804	76	PC	RD	HALT

Stack-be pakolás; 12 ütem; Z ≠ 0 → Ugrás nincs; RET

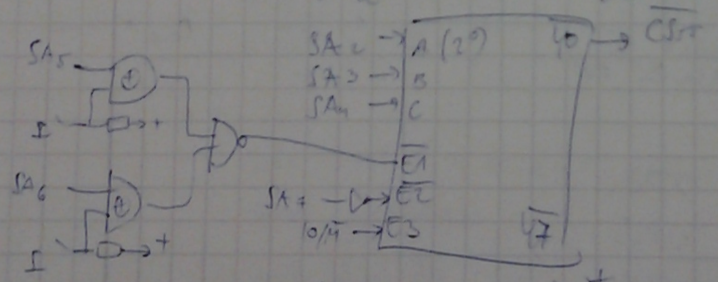
PC értéke : 0805

(± DAD utasítás ! Fontos fejből tudni, azokat utasításokat nem lévén az idő)

③ Perifériaillesztés

① Címdekóder

felső 128 16 cím ??-vel kezdik ... olvasható / írható legyen



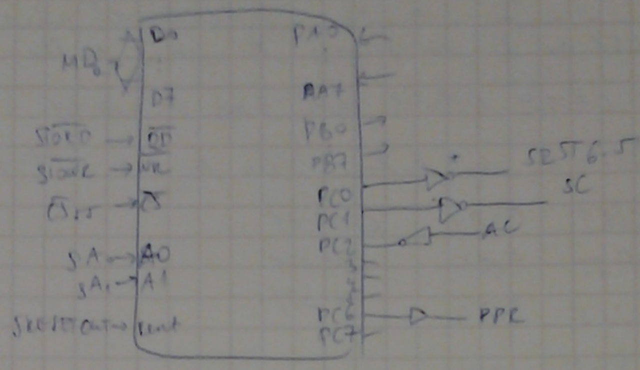
② Ready logika : 0 wait => READY = CS

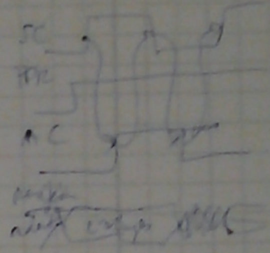
$\overline{CS_{55}}$ ——▷* —— $\overline{READY}$

① Bekötés

B port:
- $SC = \overline{OBF_B} = PC1$
- $AC = \overline{ACK_B} = PC2$
- $INTR_B = PC_0$

A port:
- $IBF_A = PC5$
- $\overline{STB_A} = PC4$
- $INTRA_A = PC3$





→ PPR = PC6 / PC7

④ inicializáló szubrutin (INIT55) beállítja a fentieknek megfelelő üzemmódba az 0E0H báziscímen található 8255-öt, jelzi a külső perifériának a működésképes állapotot (PPR), engedélyezi a B port megszakítását a 8255-ben

8255 üzemmód beállítás

A port	bemenet	1-es üzem	
B	bemenet	1-es üzem	
C	felső 4 bit	bemenet	
C	alsó 4 bit	– (bemenet)	

10110100 = 0B4H

PPR (PC6) bit set

00001101 = 0DH

INTEB (PC2) bit set

00000101 = 05H

```
CW55   EQU   0E3H   (E0 báziscím + értelmezés alapján)
M55    EQU   0B4H
PPR    EQU   00DH
INTE   EQU   005H

MVI    A, M55
OUT    CW55
MVI    A, PPR
OUT    CW55
MVI    A, INTE
OUT    CW55
```

(3) RST 6.5 megszakítási rutinban (ITRUT), KARKI nevű memóriarekeszben található adat kiírása a 8255 B portjára. A rutin a STAT nevű változóba írja 0AAH értéket jelezve, hogy az adatot kiküldte, a RST-6.5 maszkolását tiltsa le az ... itkezelést

```
        DISIT  EQU   004H
        BF55   EQU   0E1H
        CW55   EQU   0E3H

ITRUT:  PUSH   PSW
        LDA    KARKI
        OUT    BF55
        MVI    A, 0AAH
        STA    STAT
        MVI    A, DISIT
        OUT    CW55
        POP    PSW
        EI
        RET
```

(4) Memóriaillesztés, ahogy a házíban