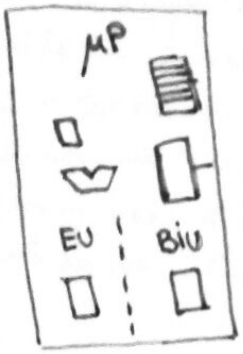
PM - CURS 3

BIU = Bus Interface Unit

EU = Execution Unit



Avem nevoie de un mecanism a.î. dispozitivele din jurul μP să lucreze în același timp cu el (să aibă aceeași frecv.) pt. a se putea realiza comunicarea dintre ele.

* NICIODATĂ N-A FOST SĂ NU FIE CUMVA.

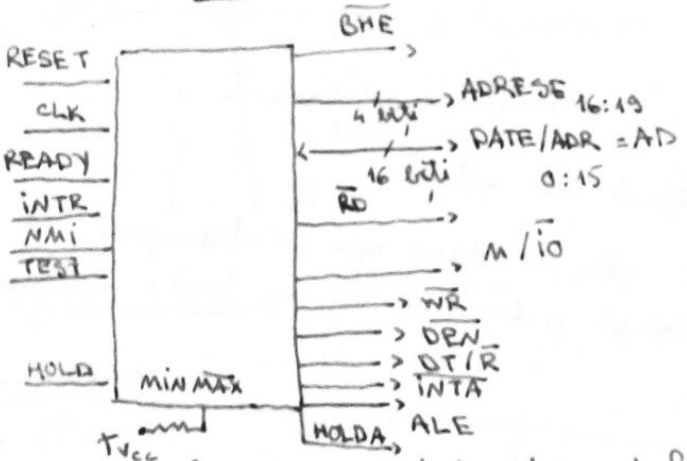
Componentele cu care dialoghează μP în afară:

- adrese
- date
- magistrala de comenzi (scriere / citire în/din mem.)
- stare (starea echipamentului resp. - gata (ready) sau nu)

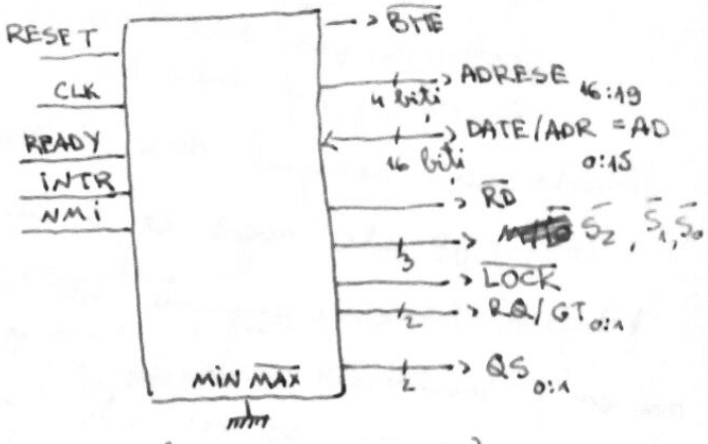
Procesorul are 2 moduri de funcț.:

- minim - scoate toate semnalele de bază necesare conectării cu exteriorul
- maxim - se poate adăuga un coprocesor matematic sau alte coprocesoare pt. a împărți task-urile

MIN



MAX



- Procesorul este anunțat de modul în care va lucra (min/max) cu ajutorul unei rezistențe sau masa (1/0).
- Pentru că nr. de pini nu era suficient (erau 40 de pini, iar proc. are nevoie de mai mulți), s-a folosit același mag. pt. date și adr. (la un moment dat se transm. date, apoi adr.)
- Pt. citire, trebuie ca procesorul să aibă un semnal prin care specifică de unde vrea să citească (mem. / std. input output)
- Avem nevoie de o leg. cu întreruperile a.î. atunci când primește înt. din ext. să le proceseze, apoi să își continue treaba (INTR

• Întreruperile INTR = **întreruperi mascabile**. Adică procesorul poate refuza să accepte întreruperile prin disable interrupt. De asemenea, acest refuz poate fi selectiv (anumite întreruperi sunt ignorate) prin enable interrupt (accept tot sau unele).

Întreruperile nemascabile sunt întreruperile ce nu pot fi refuzate. Acest lucru este necesar deoarece ∃ situații de urgență (căderea tensiunii de alimentare => sarcina de alim. va fi preluată de UPS; dacă căderea durează mai mult decât poate duce UPS-ul, pică tot => se pierd toate datele) în care sunt generate întreruperi speciale (nemascabile) de către UPS, iar procesorul trimite semnale ptr. salvarea tuturor datelor (se comportă ca și la shutdown). => semnalul NMI (non masca - - -)

• $\overline{DEN}$ = semnalul care ne spune că datele sunt stabile și pot fi scrise

• Trebuie să îl anunț pe cel din ext. că vreau să transmit/preiau date => semnalele DT sau $\overline{R}$

• Când avem întrerupere + enable interrupt în timpul unei instrucțiuni, termin întâi instr. curentă apoi se uită μP la întrerupere. Când s-a terminat instr. curentă, se transmite semnalul $\overline{INTA}$ = Interrupt Acknowledge

• Dacă DMA (= Data Memory Access???) și procesorul vor să acceseze în același timp mem., se folosește un semnal HOLD și HOLDA ptr. a întârzia procesorul.

• $\overline{TEST}$ = μP este rugat de cineva din afară să aștepte

! Negarea la unele semnale este necesară ptr. că dacă apar întreruperi ele nu vor fi considerate instrucțiuni. S-a stabilit prin convenție ca toate semnalele de comandă să fie active pe 0.

MAX

• $\overline{S_0}, \overline{S_1}, \overline{S_2}$ = se va codifica ciclul mașină curent

• Dacă vrem să lucrez cu încă un procesor, trebuie implementat un semafor ptr. a dirija accesul la memorie => $\overline{LOCK}$

• Avem nevoie de RQ (request), GT (0:1) prin care putem face sincronizarea la nivel de stare cu un coprocesor matematic sau un coproc. IO.

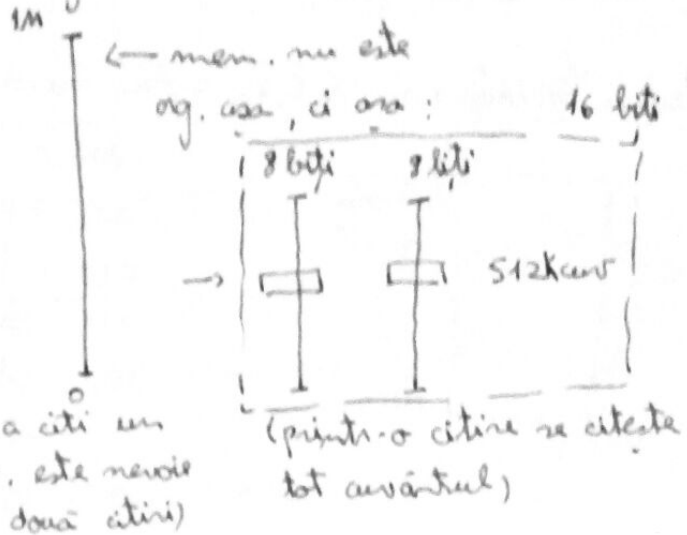
Coprocesor = unitate de sine stătătoare care execută instrucțiuni pe cont propriu (își ia singur operanzii, nu îi așteaptă de la proc.) -> => are interpretor de instrucțiuni

• $QS_{1:0}$ = semnal prin care se anunță ?starea stivei?

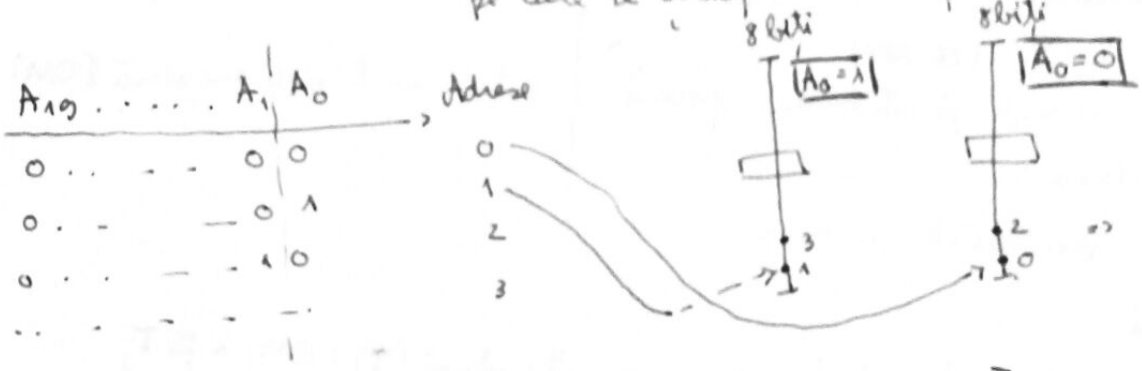
Organizarea spațiului:

adr. { M / I/E

Organizarea memoriei:



• $\overline{BHE}$ (Byte High Enable) = spune procesorul celor din ext. dim. datelor pe care se așteaptă să le primească



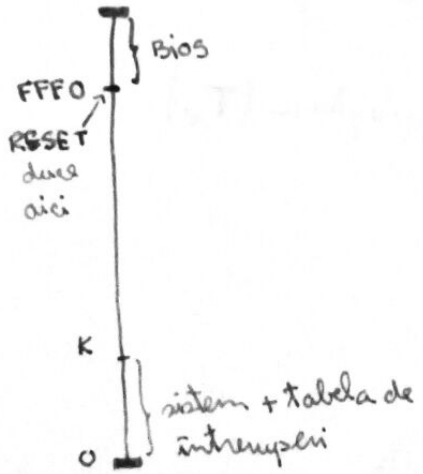
$\overline{BHE}$	A_0	
0	0	16 biți (se lucrează pe 16 biți : $D_{8:15}$, $D_{0:7}$)
0	1	8 biți ($D_{8:15}$)
1	0	8 biți ($D_{0:7}$)
1	1	X

8 biți 8 biți

NECESAR LA PROIECTAREA MEMORIEI

În memorie vor avea niște **celule rezervate**:

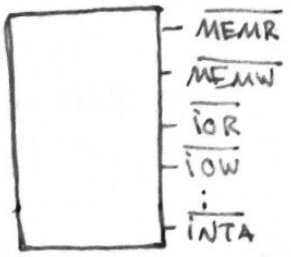
- primul K va fi rezervat ptr. sistemul de operare + tabela de întreruperi
- ultimele 16 celule rezervate ptr. Bios ~~[illegible]~~



Sistemul de intrare/ieșire:

• porturile cu adr. pară trebuie să pună $D_{0:7}$, iar cele cu adr. impară $D_{8:15}$

Pentru decodificarea $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ avem nevoie de un mecanism:



000 - Interrupt Acknowledge
001 - citire port I/E => $\overline{IOR}$
010 - $\overline{IOW}$ - scriere port I/E
011 - HALT
100 - citire instr. (fetch)
101 - citire din mem. => $\overline{MEMR}$
110 - scriere mem => $\overline{MEMW}$
111 - nu este alocat (folosit pt. a arăta că s-a terminat ciclul mașină curent)

Ciclu instrucțiune = citire, interpretare, execuție instrucțiune (**CI**)

1. citire instr. (ex. ADD)
2. citesc primul și al doilea operand
3. fac adunarea
4. pun rezultatul în mem

} astfel sunt **cicli mașină (CM)**

$$CI = \sum_i CM_i$$

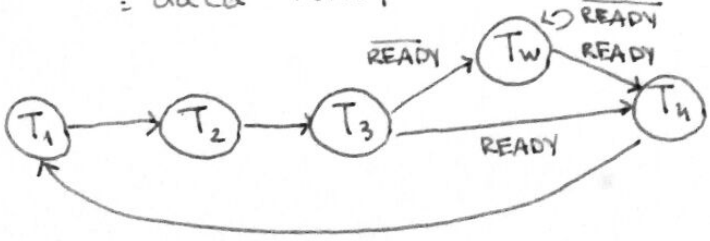
Ciclul mașină este format din mai multe **stări (T)**: $CM_i = \sum_j T_j$

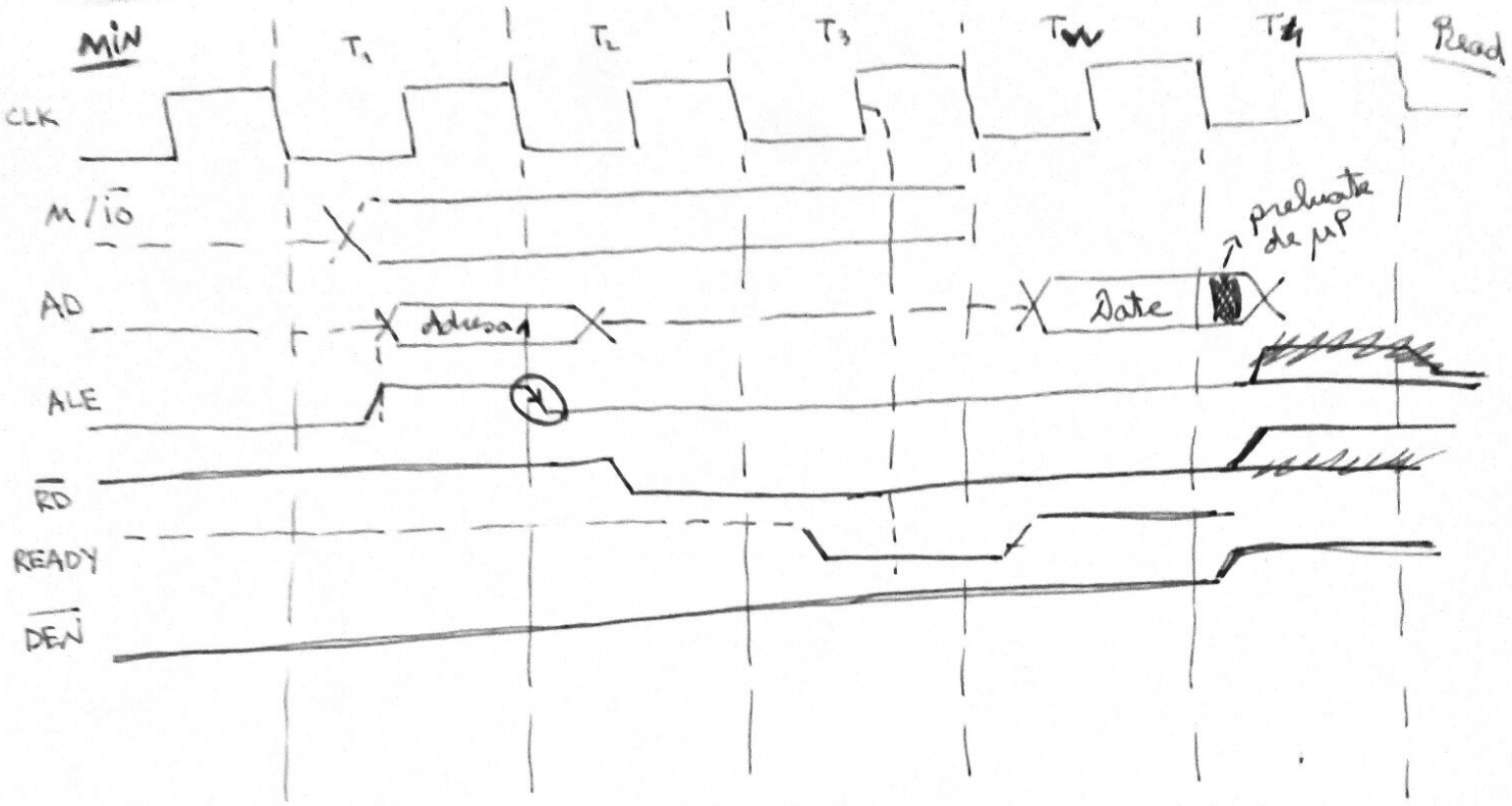
T_j = stare = perioadă de ceas a echipamentului.

În: • T_1 = are loc op. de început de ciclu în care se comunică adresa și stări pe magistrala AD

• T_2 = încerc să comut magistrala AD ? pt. că o dată am dat adresa, iar acum vreau să fac read?

• T_3 = ar trebui luate datele, însă trebuie să mă asigur că sunt gata => verific READY (0 = nu sunt gata, 1 = sunt gata)
: dacă READY = 1 => trec în T_4 (adică cel din afară a putut răsp. la comanda read/write)
: dacă READY = 0 => intru într-o stare de așteptare (T_W)




MIN
T1
T2
T3
Tw
T4
Read
CLK
M/IO
AD
Adresse
Date
prelevate de µP
ALE
RD
READY
DEN