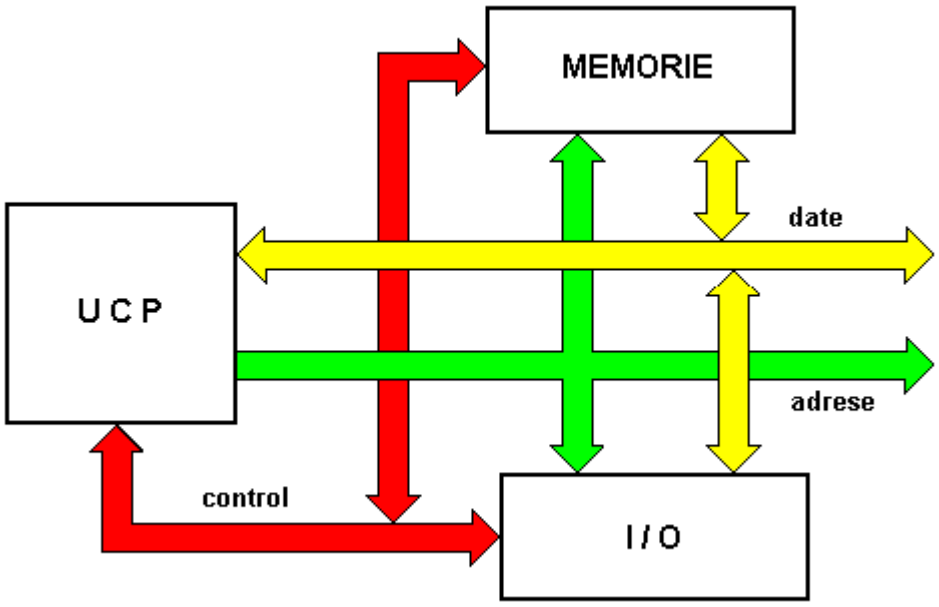
. . . ::AMP 1::. . .

I.Structura unui microcalculator.Definitii

…:::1.1 Componentele multifunctionale ala unui microcalculator:::…

Microcalculatorul, structurat ca o maşină **"VON NEUMANN"**, este un sistem programabil de prelucrarea informaţiei care are două componente inseparabile şi definitorii: **hardware şi software.**

schema bloc functionala.



A. Componenta **hardware**; blocurile funcţionale sunt:

1. UNITATEA CENTRALĂ DE PRELUCRARE (UCP); două funcţii esenţiale:
   - prelucrarea datelor;
   - controlul activităţii întregului microcalculator.

   O *Unitate centrală de prelucrarea informaţiei*, având funcţiile enunţate mai sus, care coordonează un sistem structurat funcţional ca în figură şi care, fizic, se prezintă sub forma unui singur cip se numeşte **MICROPROCESOR (µP).**

2. MEMORIA este o secvenţă de locaţii pentru stocarea informaţiei. Fiecare locaţie este definită prin două entităţi informaţionale:

• ***Conţinutul***, reprezentat de o înşiruire de cifre binare 0 sau 1 ("biţi");
- numere
- coduri etc.

Numărul de cifre binare conţinute într-o locaţie depinde de modul in care microprocesorul organizează informaţia în memorie; mărimea unei locaţii va fi denumită formatul memoriei, exprimat în număr de biţi (de regulă 8, 16, 32 sau 64 biţi).

• ***Adresa***, reprezentând numărul de ordine al locaţiei, care permite identificarea sa în cadrul secvenţei de locaţii (există o corespondenţă biunivocă între fiecare locaţie de memorie şi adresa sa).

Noţiuni aferente:

- **"Harta memoriei"**: totalitatea locaţiilor de memorie pe care le poate adresa un microprocesor.
- **"Pagini"** şi/sau **"segmente"**: subdiviziuni logice ale hărţii memoriei, ale căror dimensiuni, fixe sau dinamice, sunt specifice modului în care un microprocesor anume organizează memoria.

Structura fizică a memoriei unui microcalculator este formată din unul sau mai multe cipuri, cu capacităţi diverse; capacitatea totală de stocare a informaţiei pe care o realizează fizic cipurile de memorie într-un microcalculator este definită ca "memorie internă". Aceasta nu acoperă, în mod necesar, harta memoriei aferentă microprocesorului respective.

Semnificaţia conţinutului memoriei microcalculatorului → două zone:

→Memoria de date conţine operanzi şi/sau rezultate; fizic, această porţiune de memorie este de tip RAM (cu scriere/citire).

→Memoria de program care conţine instrucţiuni; de regulă, (dar nu obligatoriu) această zonă este o memorie de tip ROM (memorie din care se poate doar citi).

Instrucţiunea:

→informaţia codificată (binar) prin care se impune microprocesorului desfăşurarea unei acţiuni specifice.

Observaţii:

• Fiecare instrucţiune este asociată în mod biunivoc cu un şir de cifre binare; deoarece acestea "codifică" instrucţiunile, vor fi denumite **coduri.**

• O instrucţiune reprezintă cea mai simplă acţiune, cu rezultat bine precizat, din activitatea unui microcalculator a cărui *unitate centrală de prelucrare* a informaţiei este un microprocesor anume.

• Un microprocesor concret poate "recunoaşte" şi executa numai codurile corespunzătoare instrucţiunilor pentru care a fost construit; totalitatea instrucţiunilor pe care un microprocesor le poate recunoaşte şi executa alcătuieşte setul de instrucţiuni al microprocesorului respectiv.

• Înşiruirea instrucţiunilor în memoria de program nu este haotică ci sub formă de **programe**, noţiune definită ca fiind o secvenţă de coduri de instrucţiuni organizate în mod logic şi coerent după un anumit algoritm, astfel încât întregul microcalculator să execute o sarcină prestabilită. Noţiunea de **"sarcină" (task)** nu trebuie confundată cu cea de program: sarcina unui microcalculator corespunde unei alocări dinamice a resurselor hardware şi software; există sarcini pentru a căror îndeplinire sunt necesare mai multe programe.

Concluzii:

Semnificaţia conţinutului locaţiilor de memorie este conferită de programator în concordanţă cu funcţiile specifice realizate de microprocesor:

→ *numere binare* atunci când ne referim la date (operanzi/rezultate);

→ *coduri* când ne referim la instrucţiuni.

În schema bloc funcţională propusă, memoria nu are nici un control asupra semnificaţiei informaţiei pe care o conţine.

---

3. DISPOZITIVELE DE INTRARE/IEŞIRE (I/O): circuitele prin care se realizează legătura între microcalculator şi lumea exterioară. O unitate elementară de conversaţie cu exteriorul poartă numele de **"port de intrare/ieşire"**.

Între porturi şi locaţiile din Memorie există nişte similitudini:

→ Porturile sunt în esenţă tot locaţii de memorare a informaţiei, adresabile; informaţia care se foloseşte uzual aici este alcătuită din operanzi/rezultate (date).

→ Există o **"hartă a porturilor"** care poate sau nu să facă parte din harta memoriei.

Singura deosebire esenţială faţă de locaţiile de memorie este *legătura fizică pe care porturile o asigură cu exteriorul*; pentru microprocesor, de multe ori, această legătură fizică este transparentă şi nesemnificativă.

"Magistrală": un set de conexiuni fizice între blocuri prin care informaţia care circulă are o semnificaţie prestabilită. Sistemele la care ne referim au o magistrală unică, ce le caracterizează; din punct de vedere funcţional, există trei componente ale acestei magistrale:

1. *Magistrala de date*, bidirecţională, permite circulaţia datelor (operanzi/rezultate), a instrucţiunilor şi chiar a adreselor. *

2. *Magistrala de adrese*, unidirecţională, permite microprocesorului să localizeze informaţia în Memorie sau în Dispozitivele de intrare/ieşire; deci pe această magistrală circulă numai adrese.

3. *Magistrala de control* permite circulaţia, bidirecţională, a semnalelor de comandă şi control de la/la microprocesor, în calitatea sa de Unitate centrală.

Magistrale
→unidirectionale
→bidirectionale

---

*Magistrala de date este o caracteristica fundamentala VON NEUMANN-permite sa treaca si numere(date) si instructiuni.

B. Componenta software: o serie de programe organizate în moduri specifice.

Două categorii de software:

1. Sistemul de operare: totalitatea programelor care permit utilizatorului accesul complet la resursele sistemului (exemple: MS-DOS, UNIX etc.). Poate fi: rezident

(permanent în memoria internă) sau încărcabil dintr-o memorie externă (operaţie denumită "bootstrap").

2. Software-ul utilizatorului, alcătuit din totalitatea programelor folosite pentru îndeplinirea unor sarcini specifice.

Caracteristicile arhitecturii "Von Neumann":

→ Microprocesorul constituie Unitatea centrală de prelucrare a unui sistem având schema bloc funcţională din figură. El concentrează:

→funcţia de prelucrare
→functie de comandă.

→ Toate celelalte componente ale sistemului nu au putere de decizie. Memoria nu controlează şi nici nu e necesar să controleze semnificaţia informaţiei pe care o deţine şi modul în care este organizată logic.

→ Legătura dintre blocuri este asigurată de o magistrală unică cu trei componente funcţionale; pe magistrala de date circulă toate tipurile de informaţii.

→ Funcţionarea sistemului se face pe baza unor programe alcătuite din secvenţe de instrucţiuni. Acestea sunt citite din memorie de către microprocesor, recunoscute şi apoi executate.

Arhitectură:

→totalitatea atributelor sistemului (în cazul de faţă, microprocesorul) care sunt disponibile ("vizibile") utilizatorului (ca, de pildă: registrele, modurile de adresare, tipurile de transferuri de date, modul de organizare logică a memoriei, tehnicile de intrare/ieşire, setul de instrucţiuni etc)

...:::1.2 Definitii, microprocesoare CISC si RISC:::…

→Microprocesor, microcalculator, minicalculator.

Asemănarea: caracteristicile globale ale atributelor de arhitectură.

Deosebirile între ultimele două: resurse (memorie internă şi externă, echipamente periferice) şi performanţe (viteză de prelucrare, cost, număr de componente, gabarit).

Definiţia microprocesorului ca Unitate centrală de prelucrare; am presupus implicit că sistemul din care face parte este un micro(mini)calculator, deci *un sistem de calcul*. Putem extinde însă noţiunea şi asupra **sistemelor de comandă şi control** (de tip **"controler"**), ceea ce măreşte aria de cuprindere a noţiunii de microprocesor.

Noţiunea de *"logică programată"*. Sistemele cu logică programată nu înseamnă, în mod automat, sisteme cu microprocesor. Microprocesorul poate constitui una dintre modalităţile de proiectare a sistemelor cu logică programată. Nu se va face confuzia "sistem cu logică programată cu microprocesor" ≠ "sistem microprogramat".

Clasificări ale noţiunii de microprocesor:

a) După lăţimea magistralei de date: microprocesoare pe 8, 16, 32 sau pe 64 de biţi.

b) După tipul de sarcini eficient realizabile:

→ *microprocesoare de uz general (µPUG), nespecializate;*

→ *microprocesoare specializate, ca de pildă:*

- procesoare de intrare/ieşire, pentru conversaţii complexe între microcalculator şi lumea exterioară; exemplu: Intel 8089;
- coprocesoare aritmetice, specializate pentru funcţii aritmetice de utilitate generală (exponenţiale, trigonometrice etc); exemplu: Intel 80387;

- procesoare digitale de semnal, specializate pentru algoritmii specifici prelucrării semnalelor (FFT, produse de corelaţie, filtre digitale, calcul matriceal etc.); exemplu: Texas Instruments TMS 320.

c) După principiile de bază ale arhitecturii care guvernează funcţionarea:

→ procesoare cu set complex de instrucţiuni (CISC) numite microprocesoare "standard" sau simplu "microprocesoare";

→ procesoare cu set redus de instrucţiuni (RISC).

…:::1.3 Reprezentarea informatiei in sistemele digitale:::…

- **bit** (prescurtat b) pentru o cifră binară 0 sau 1;
- **nibble** (prescurtat n) pentru o înşiruire de 4 biţi;
- **byte** sau **octet** (prescurtat B) pentru o înşiruire de 8 biţi;
- **cuvânt** sau **word** (prescurtat w) pentru o înşiruire de 2 octeţi;
- **cuvânt dublu** sau **double word** (prescurtat dw) pentru o înşiruire de 4 octeţi;
- **prefixele**:

  • k pentru $2^{10} \approx 10^3$;
  • M pentru $2^{20} \approx 10^6$;
  • G pentru $2^{30} \approx 10^9$;
  • T pentru $2^{40} \approx 10^{12}$.

1.3.1 Reprezentarea interna

a) Reprezentarea programelor

Fiecare instrucţiune este reprezentată în memorie de un *cod binar*. **Formatul instrucţiunilor**, adică totalitatea cifrelor binare necesare pentru codificare, are, de regulă, drept cuantă de informaţie, octetul. Pentru fiecare instrucţiune există un număr prestabilit de octeţi cu care e codificată (de pildă, pentru Intel 8086, este între 1 şi 6 octeţi).

b) Reprezentarea numerelor

1) Reprezentarea întregilor fără semn în "binar natural": este reprezentarea uzuală, "naturală" a numerelor binare.

$$\text{număr binar cu 8 cifre} = \sum_{i=0}^{7} b_i 2^i \quad \text{cu } b_i \in \{0, 1\}$$

$$\text{număr binar cu 8 cifre} = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

b_0 : lsb

b_7 : msb

2) Reprezentarea întregilor cu semn în "binar natural":

Semnul numărului este reprezentat de msb cu următoarea convenţie:

msb = 0 semnifică număr **pozitiv**;

msb = 1 semnifică număr **negativ**.

Pentru un număr fără semn cu 8 biţi, plaja numerelor reprezentabile acoperă 256 de poziţii, între 0 şi 255, în zecimal. Pentru un număr cu semn, plaja numerelor reprezentabile acoperă tot 256 de poziţii, dar în intervalul -128 ÷ +127, presupunând 0 număr pozitiv.

Convenţii de reprezentare:

Tipul reprezentării	+ 5	- 5
"mărime şi semn"	00000101	10000101
"complement faţă de 1"	00000101	11111010
"complement faţă de 2"	00000101	11111011

Regulile de reprezentare în aceste trei convenţii:

→ Numerele pozitive se reprezintă identic.

→ În "mărime şi semn", numerele negative diferă de cele pozitive numai prin bitul de semn.

→ În "complement faţă de 1", mărimea numărului negativ se obţine din reprezentarea precedentă prin complementare bit cu bit; convenţia pentru bitul de semn se păstrează.

→ În "complement faţă de 2", mărimea numărului negativ se obţine din reprezentarea precedentă prin adunarea unei cifre binare 1 la lsb.

Fanioane

→un bit de informatie in interiorul microprocesorului care isi indica evenimente speciale aparute in functionarea microprocesoarelor

→**"Transportul"** care apare între rangul unui număr binar şi cel imediat superior în operaţiile aritmetice (la scădere, îl vom numi "împrumut"): **C** (de la "carry").

→**"Depăşirea"**: **O** (de la "overflow"). După cum numărul are semn sau nu, se poate scrie că:

$$O = C_{msb} \nearrow \quad SAU \quad C_{msb-1} \nearrow msb$$

Extinderea numerelor cu semn reprezentate în complement faţă de 2, de la 8 la 16 biţi.

	Reprezentare cu 8 biți	Reprezentare cu 16 biți
+ 1	00000001	0000000000000001
- 1	11111111	1111111111111111

Regulile de "extindere a numerelor cu semn, în complement faţă de 2":

→ Bitul de semn rămâne pe poziţia cea mai semnificativă.

→ Partea care reprezintă mărimea numărului va ocupa poziţiile cele mai puţin semnifica-tive ale numărului extins.

→ Restul poziţiilor din numărul extins se completează cu cifre binare identice cu cea care reprezintă semnul (0 pentru numere pozitive şi 1 pentru numere negative).

---

3) Reprezentarea întregilor în "zecimal codificat binar" (ZCB): se reprezintă fiecare cifră zecimală separat, în binar natural, cu un nibble.

Microprocesoarele folosesc două tipuri de reprezentări ZCB:

→ Reprezentarea **"ZCB împachetat"** în care fiecare octet din memorie cuprinde câte două cifre zecimale, una pe nibble-ul mai puţin semnificativ şi cealaltă pe nibble-ul superior. Plaja de numere zecimale acoperită de o reprezentare cu 8 biţi se micşorează de la 256 la 100 de numere: 0 ÷ 99.

→ Reprezentarea **"ZCB neîmpachetat"** în care fiecare octet cuprinde o singură cifră zecimală pe nibble-ul mai puţin semnificativ. Restul cifrelor binare se completează cu 0.

---

4) Reprezentarea numerelor cu zecimale "cu virgulă fixă": se foloseşte principiul de a aloca un număr fix, prestabilit, de cifre binare pentru a reprezenta partea întreagă şi respectiv partea zecimală a unui număr.

Se poate folosi fie reprezentarea în binar natural fie în ZCB. Pentru partea întreagă se foloseşte regula de reprezentare a numerelor întregi cu semn, iar pentru partea zecimală regula de reprezentare a întregilor fără semn. (Apar: "trunchierea" sau "rotunjirea" numărului).

Modul de reprezentare foloseşte următoarele convenţii:

→ Se rezervă un şir de biţi cu care se exprimă numărul total de cifre ale numărului care urmează să fie reprezentat.

→ Se rezervă, apoi, un şir de biţi în care se înscrie numărul de zecimale cu care se va reprezenta numărul.

→ Urmează reprezentarea propriu-zisă a numărului înşiruind reprezentările pentru partea întreagă şi cea zecimală fără o altă delimitare explicită între ele.

Un exemplu în ZCB împachetat:

0000 0100 0010 0000 1001 0110 0001 0101 ....

numărul reprezentat este **+ 96.15** .

---

5) Reprezentarea numerelor cu zecimale "cu virgulă mobilă"; reprezentare normalizată.

Două entităţi informaţionale: **"mantisa" M** şi **"exponentul" EXP**:

$$\text{număr binar} = M * 2^{EXP}$$

$$2^{-1} \leq M < 2^{0} .$$

Un exemplu:

b_{31}.....$b_{24}b_{23}$.....b_0 ,

în care: - **b_{31} ÷ b_{24}** reprezintă exponentul, având semnul în poziţia **b_{31}**.

- **b_{23} ÷ b_0** reprezintă mantisa cu semnul la **b_{23}**.

Plaja numerelor reprezentabile în acest fel: $M * 2^{\pm 128}$

---

c) Reprezentarea datelor alfanumerice

Vom înţelege prin **"date alfanumeric**e" sau **"caractere"** oricare dintre semnele care pot fi tipărite de la tastatura unui calculator.

"Seturi de caractere": grupuri minime de simboluri considerate suficiente pentru a asigura o editare cât mai completă a unui text.

Pentru fiecare caracter se va folosi o reprezentare binară, un cod, cu care caracterul (dintr-un set prestabilit) este în relaţie biunivocă.

Standardul **"ASCII"** (de la "**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange") cu care se codifică următorul set de caractere:

- 26 de litere mari ale alfabetului latin;
- 26 de litere minuscule corespunzătoare;
- 10 simboluri numerice: 0 ÷ 9;
- 20 de simboluri speciale adiţionale: +, -, (,), [,], {, }, * , # , $ etc.

Codificare cu 7 biţi; **msb** : "*bit de paritate*". Convenţia folosită este următoarea:

msb = 0 dacă codul are un număr *par* de cifre binare **1**;
msb = 1 dacă codul are un număr *impar* de cifre binare **1**;

De exemplu: **A** = 01000001;
B = 01000010;
C = 11000011 etc.

Bitul de paritate → fanion dedicat (**P**).

1.3.2 Reprezentarea externa

Reprezentarea externă se referă la modul în care informaţia prelucrată de un microcalculator apare utilizatorului (programatorului).

a) Pentru codurile instrucţiunilor se vor folosi abrevierile sugestive pe care, de regulă, fabricantul le impune şi pe care limbajul de asamblare le foloseşte ca atare: "mnemonice".

b) Pentru numere se utilizează mai multe tipuri de reprezentări:

→ Reprezentarea binară, imagine fidelă a conţinutului locaţiilor de stocare a informaţiilor.

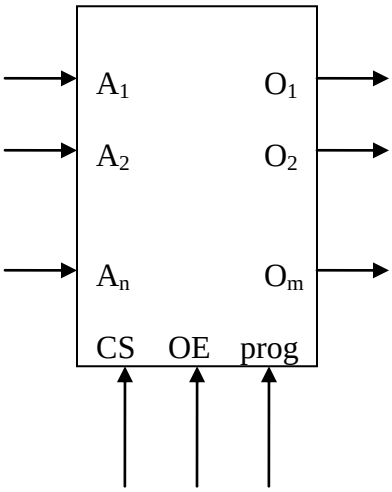
→ Reprezentarea octală, care transformă numerele binare în baza de numeraţie 8.

→ Reprezentarea hexazecimală : un simbol reprezentând o cifră în baza de numeraţie 16 înlocuieşte 4 cifre binare. Caracterele folosite sunt cifrele zecimale 0 ÷ 9 şi literele A ÷ F. Vom folosi convenţia de a utiliza litera **H** ca sufix pentru numerele reprezentate în hexazecimal (de pildă 1199H).

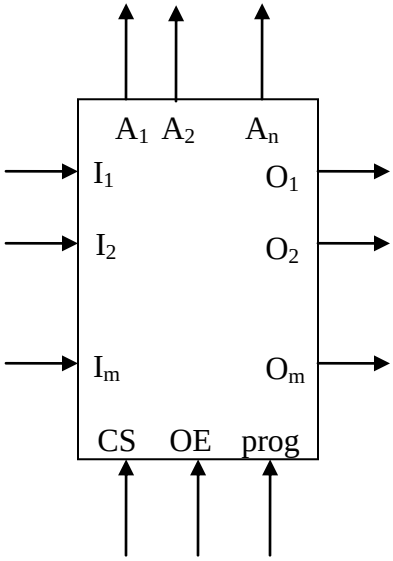
c) Pentru caractere se vor folosi chiar simbolurile cu care ele sunt individualizate. Programele utilitare folosite pentru examinarea conţinutului locaţiilor de stocare a informaţiilor fac conversia **ASCII → simbol caracter** dacă programatorul stabileşte că semnificaţia informaţiei vizate impune această conversie.

…:::1.4 Elemente de structura digitala. Memorii.:::…

ROM



RAM



A_1……..A_n →intrari adresare
→folosesc pentru adresarea fizica in acest camp

O_1……..O_m →iesiri de date
→intr-o locatie am m biti

CS →chip select
→mai multe cipuri; mai intai selectez chipul si apoi ce am nevoie din el

I_1……..I_m →intrari de date

A_1……..A_n → magistrala de adresare

O_1……..O_m
Si
I_1……..I_m → magistrala de date

CS - chip select → magistrala de adresare

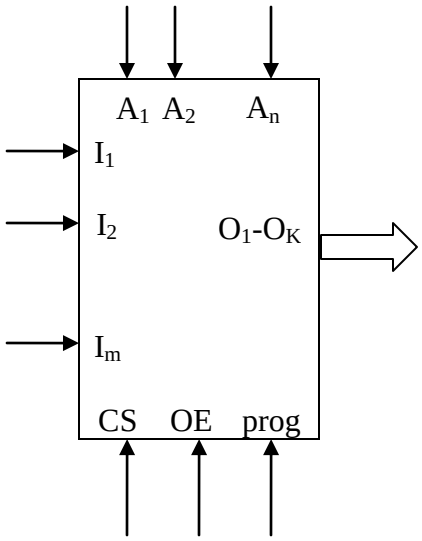
R/W → read/write (magistrala de control)
→ ii spune memoriei acuma scriem acuma citim.

Circuite combinationale

A. PORTI

SI , SAU , NICI, NUMAI , SAU_EXCLUSIV

B. MULTIPLEXORUL

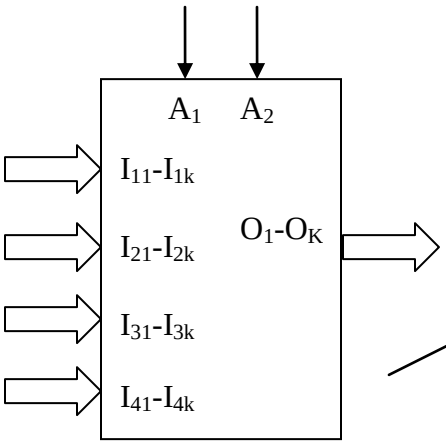


I_1……..I_m →intrari de date

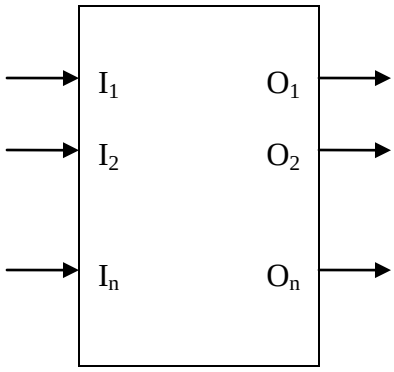
A_1……..A_n →intrari adresare

O – iesirea

Multiplexor cu n=3



Multiplexeaza pe o magistrala patru magistrale.



$$O_1=f_1(I_1,I_2,\ldots..,I_N)$$
$$O_2=f_2(I_1,I_2,\ldots..,I_N)$$
$$\ldots\ldots\ldots$$
$$O_n=f_n(I_1,I_2,\ldots..,I_N)$$

$$0<=O_k,f_k<=1$$

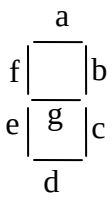
functiile $f_1,f_2,\ldots\ldots f_n$ sunt independente

C. DEMULTIPLEXORUL

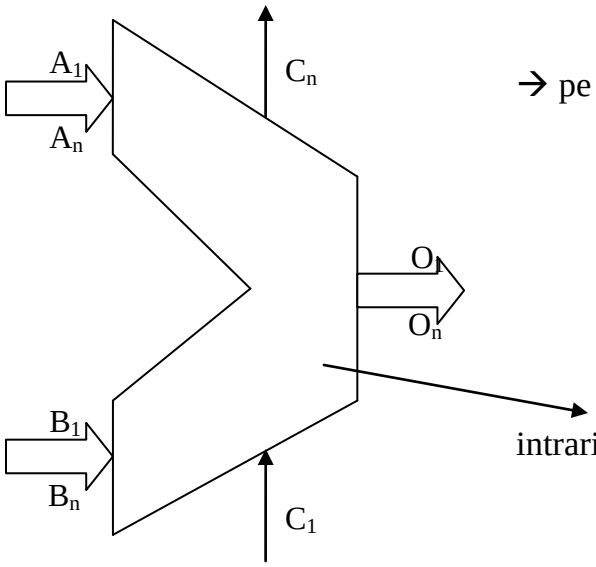
I_4	I_3	I_2	I_1	O_1	O_2	.....	O_{16}
0	0	0	0	0	0	.....	**0**
0	0	0	1	0	1	.....	**0**
.....	.....	.....	.....	.....	.....	.....	**.....**
1	**0**	**0**	**1**	**0**	**0**	**.....**	**1**

-recunoaste codul--- vine o intrare o anumita iesire ridica mana

ZCB- 7 segmente (display electronic)



D. UNITATEA ARTIMETICO-LOGICA (UAL sau ALU)



→ pe n biti in parallel

intrari de control care selecteaza functia pe care o folosim

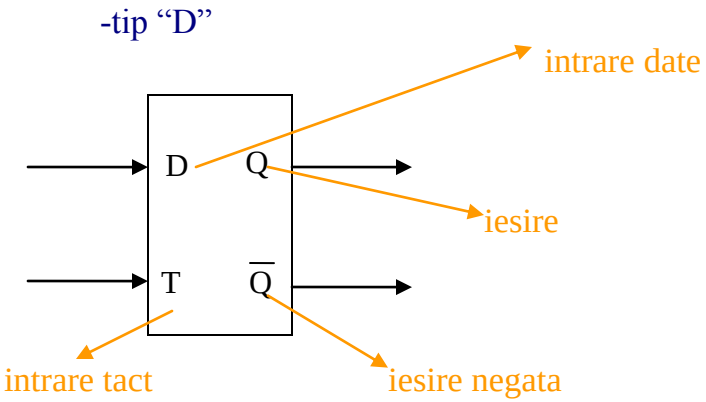
-1,2 operanzi.

Circuite secventiale

→ circuite logice ale caror iesiri depind si de intrari dar si de starea anterioara a circuitului.

circuite secventiale sincrone→trecerea de la o stare la alta se face cu ajutorul unor **impulsuri de tact.**
(frecventa cea mai mare din circuit)

a) bistabili



D_n	Q_{n+1}
0	0
1	1

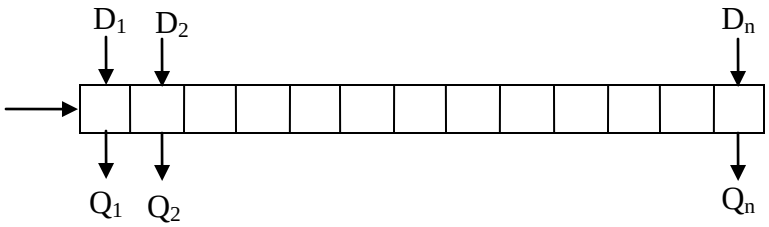
→ce se intampla la trecerea de la o stare la alta

→daca intra 1 <u>stocheaza</u>

-cat timp nu se schimba stocheaza

→fanioane pe bit (folosim bistabili)

b) registre



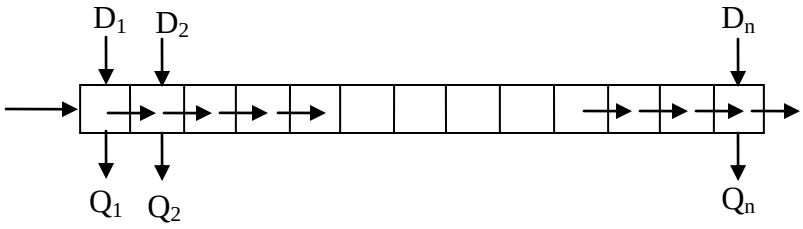
-**concatenare (alaturare) de bistabili sincronizati cu accelasi impuls de tact.**

-stocheaza n biti pentru n bistabili

⇩

locatie de memorie

registru de deplasare

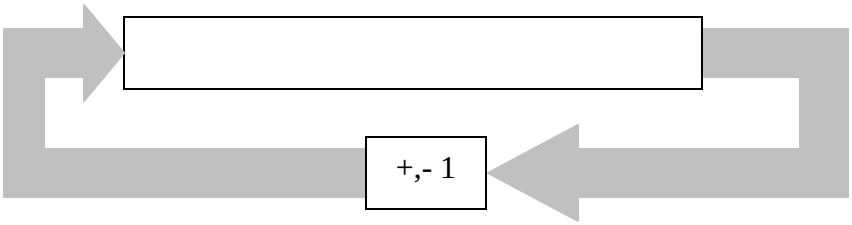


-deplasarea la dreapta :2
-deplasarea la stanga *2

registru numarator

-registru care daca are un numar incarcat atunci cand primeste un impuls de tact incrementeaza cu 1 sau decrementeaza (circuitul numara inainte sau inapoi)

-numaratoarea este prestabilita.



Observatii:

→registrul deplasare (*2,:2)
registrul numerator (incrementez,decrementez)

→ nu are rost ca la intrare decoder sa vina altceva decat cod
nu are rost sa pun la UAL un cod
nu are rost sa pun intr-un registru de deplasare sau intr-un registru numarator un cod .

…:::1.5 Conventii pentru notatii:::…

⇩ descriere formala a semanticii μP

1. neterminali

→entitati informationale neprecizate
→folosesc indice ca sa pot face diferenta intre ele (r_i,r_j)
→cifra dupa arata dimensiunea (r8)

r	**un registru oarecare**
r8	**un registru de 8 biţi**
r16	**un registru de 16 biţi**
r_i , r_j	**registre individualizate, diferite**
mem	**o locaţie de memorie oarecare (sau mai multe locaţii succesive)**
mem8	**o locaţie de memorie de un octet**
mem16	**o locaţie de memorie de 16 biţi** (pot fi două locaţii succesive dacă formatul este octetul)
mem32	**o locaţie de memorie de 32 de biţi** (pot fi patru locaţii succesive dacă formatul este octetul)
mem_i	**o locaţie de memorie individualizată** (în scopul de a o deosebi de alte locaţii de memorie)
adr	**o adresă oarecare**
adr8	**o adresă pe 8 biţi**
adr16	**o adresă pe 16 biţi**
adr24	**o adresă pe 24 de biţi**
adr_i	**o adresă individualizată** (în scopul de a o deosebi de alte adrese)

( ) → continut.

(( )) → adresare indirecta
≠registru
≠continut
→continutul unei locatii de memorie a carei adresa se afla intr-un registru.

(r)	**conţinutul unui registru oarecare**
(r_i, r_j)	**conţinutul a două registre concatenate**
$(r)_l$	**conţinutul jumătăţii inferioare** (mai puţin semnificativă) **a unui registru**
$(r)_h$	**conţinutul jumătăţii superioare** (mai semnificativă) **a unui registru**
((r))	**conţinutul unei locaţii de memorie a cărei adresă se află într-un registru (adresare indirectă)**
(mem)	**conţinutul unei locaţii de memorie oarecare**
adr_l	**jumătatea inferioară a unei adrese** (low)
adr_h	**jumătatea superioară a unei adrese** (high)
data	**un operand oarecare**
data8	**un operand de 8 biţi**
data16	**un operand de 16 biţi**
disp8	**un deplasament pe 8 biţi**
disp16	**un deplasament pe 16 biţi**
port	**un port de intrare/ieşire oarecare**

2. terminali

→entitati informationale precizate
→litere mari

R1, R2, A, AX, BP, A6, Dn, An	**nume de registre**
(R1)	**conţinutul registrului R1**
(R1, R2)	**conţinutul perechii de registre R1 şi R2**
((R1))	**conţinutul locaţiei de memorie a cărei adresă se află în registrul R1**
MEM, MEM_1	**nume de locaţii de memorie**
ADR, ADR_n	**nume de adrese**

3. operatori

←	atribuire;
↑	concatenare;
not	complementare (negație);
∀	operația logică SAU;
&	operația logică }I;
⊕	operația logică SAU EXCLUSIV;
+	adunare;
-	scădere;
*	înmulțire;
DIV	împărțire între numere întregi;
MOD	restul împărțirii între numere întregi;

4. alte simboluri

[ ]	încadrează elemente de sintaxă opționale;
|	delimitează elemente de sintaxă alternative.

II.Structura unui nucleu de μP de uz general, CISC

nucleu → elementele μP fundamentale valabile pentru oricare μP CISC de uz general

principii VON NEUMANN

1. microprocesorul este unitatea centrala de prelucrare (2 functii)

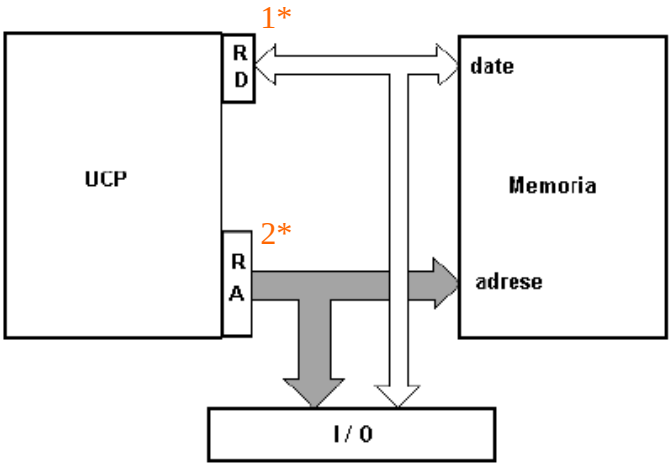
2. avem magistrala de date (pe care circula informatii)

3. intreaga functie a sistemului se bazeaza pe existenta unui program memorat alcatuit dintr-o secventa de instructiuni

→indiferent de instructiuni μP parcurge trei etape fundamentale:

1.indentifica si adreseaza in memorie codul unei instructiuni
2.decodifica acest cod (recunoaste semantica acestei instructiuni)
3.executa instructiunea.

…:::2.1 Pasul I de detaliere: registru de date si de adrese:::..



1* RD -registru de date
-nu e atribut de arhitectura
-bidirectional
-dimensiune data de magistrala de datea µP

-oricare informatie pe care µP o trimite spre restul microcalculatorului este inscrisa in RD.

-oricare informatie pe care o aducem in informatie pe care o aducem in µP o inscriem in RD

-RD asigura sincronizarea activitatii pe magistrala de date a microcalculatorului

2* RA -registrul de adrese
-foloseste pentru adresarea memoriei si a porturilor
-in el se scrie adresa fizica a unei locatii de memorie si sau a unui port
-unidirectional
-nu e atribut de arhitectura
-dimensiune data de dimensiunea adresei fizice care la randul ei e data de dimensiunea hartii memoriei

…:::2.2 Pasul II de detaliere: registre generale:::…

rgistre generale→

-stocare temporara a informatiilor in interiorul µP

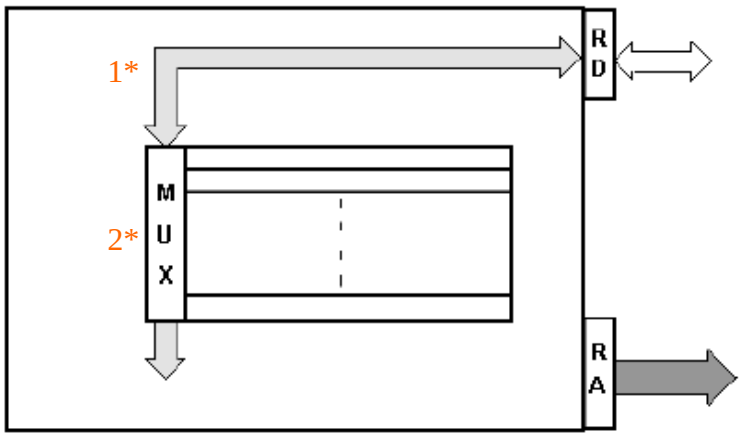
-stocheaza mai ales operanzi si/sau rezultate

-sunt attribute de arhitectura

-fac parte din structura interna a microprocesorului, cea mai rapida entitate de stocare a informatiei

-dimensiunea si nr de locatii sunt criterii de performanta (dimensiunea data de dimensunea operanzilor de obicei

(RISC fata de CISC mai multe locatii interne)



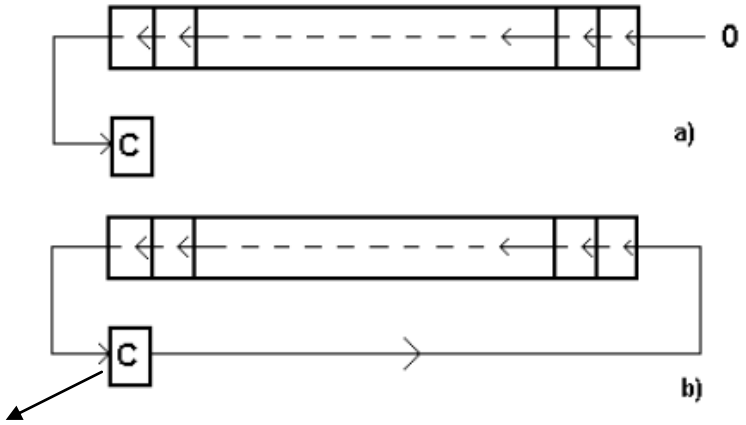
1* magistrala interna de date a microprocesorului

-este continuarea magistralei externe in interiorul µP

-dimensiunea (criteriu de performanta) – nu este neaparat egala cu dimensiunea magistralei externe

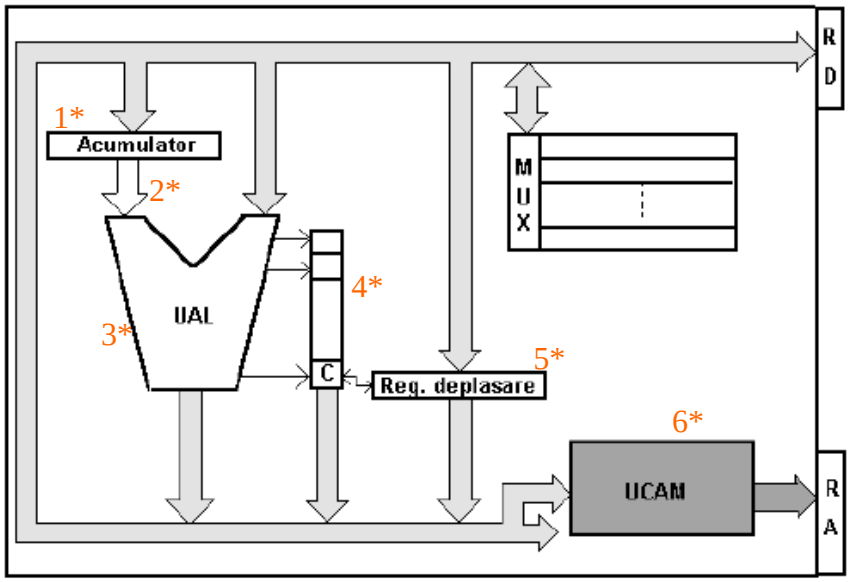
2* circuit care permite accesul la registru.

doua deplasari tipice:



fanion de transport

…::2.3 Pasul III de detaliere: unitatea aritmetica de procesare::..



1* -registru dedicate
-functii prestabilite
-stocheaza un operand pentru process si dupa operatie rezultatul
-atribut arhitectura
-dimensiunea este egala cu dimensiunea registrelor generale

2* exista anumite portiuni care nu fac parte din magistrala interna de date.

3* circuit combinational cu un set de functii aritmetico-logice
-dimensiunea este egala cu dimensiunea operanzilor de lucru

functii tipice
+, -, *, :
SI, SAU, SAU, XOR, complement fata de 1, complement fata de 2

4* -set de bistabili
-C=carry transport si este un fanion

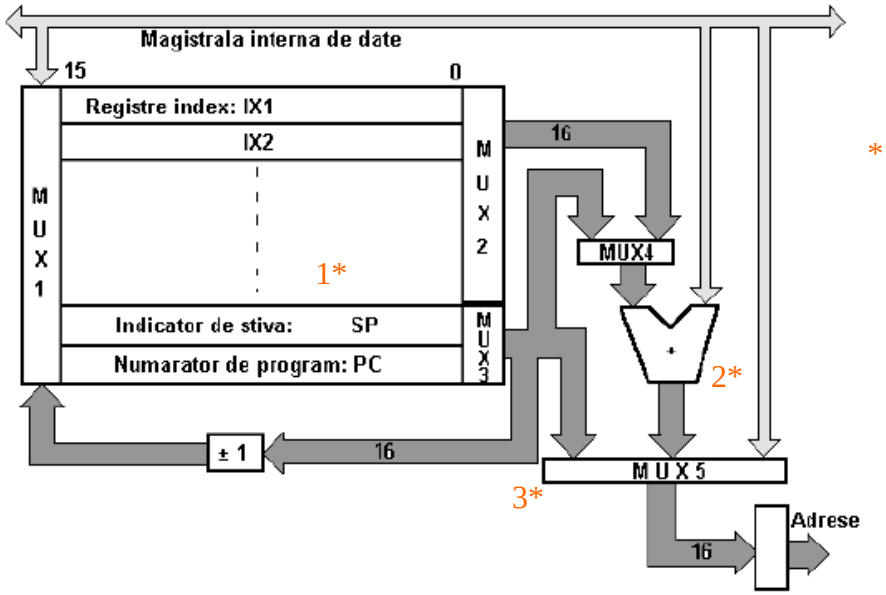
fanioane=stocheaza informatiile evenimentelor deosebite
→sunt stocate intr-un registru virtual(logic)
-atribut architectural

5* -inmulteste sau imparte cu baza de numeratie un operand
-stocheaza un opearand si dupa deplasare rezultatul

-nu e atribut arhitectural

6* unitatea de control al adresarii memoriei

…:::2.4 Pasul IV de detaliere: unitatea de control al adresarii memoriei:::..



1* registre numarator

2* un UAL (este un sumator) suplimentar

3* multiplexor(MUX)-multiplexeaza mai multe magistrale pe una singura

UCAM-uniatate de control al adresarii memoriei

-**functie principala**-de a furniza µcalculatorului adresa fizica in memorie si/sau port (fabrica adresa fizica)
-are informatii de pe magistrala interna
-intrarea lui este spre RA

UCAM furnizeaza:
a)adresa instructiunii urmatoare
b)adresa unor date

a) → a1 → furnizeaza adresa intr-o secvanta de instructiuni
→ a2 → furnizeaza adresa in cazul uni salt (adresa mai complicate)

b) → b1 → date elementare (oriunde in memoria de date sau porturi)

b2 → date structura organizata temporar **(stiva)**

b3 → date organizate spatial **(tablou=tabel)**

aceasta schema presupune un mod de acces in memorie numit **organizare liniara** care presupune ca atat µP cat si utilizatorul folosesc direct adresele fizice ale informatiei accesata liniar adresa dupa adresa

informatia este accesata liniar de la locatia cu numarul de ordine 0 pana la cea cu ordinal 2^n -1 (unde n=nr de linii ale magistralei de adrese)

a1 → realizat de PC - registru care apeleaza adresa fizica a instructiunii curente
- numarator, el este incrementat furnizand succesiv adresele intr-o secventa de instructiuni din memoria unui program
- cea mai simpla modalitate de a a accsesa informatii dintr-o secventa de instructiuni

a2 → realiazat de PC
-inscrierea in numaratorul de program (PC) al adresei de salt utilizata intr-o intr-o anumita maniera pe magistrala interna de date

*numaratorul de program (PC)
-registru dedicate
-nu e atribut de arhitectura
-dimensiunea lui in sistem cu organizare liniara a memoriei este data de dimensiunea hartii memoriei

b1 → furnizeaza adresa unui operand/rezultat prin magistrala interna de date
- nu mai afectez numaratorul de program(cum e la a-uri)

b2 → **stiva** - structura temporala de date
- posibilitatea UCAM de a aloca in memorie o stiva
- LIFO(last in first out)
- numerator special – indicator de stiva SP, cel care este automat actualizat cu adresa varfului stivei curente
- orice scriere in stiva se face cu decrementarea automata a lui SP
- orice citire in stiva se face cu incrementarea automata a SP

" stiva creste in jos " –stiva creste spre adrese inferioare

Observatii:

1| organizarea de tip stiva se numeste stiva virtuala (stiva software)

stiva hardware → set de registre organizare sub forma de stiva in interiorul µP

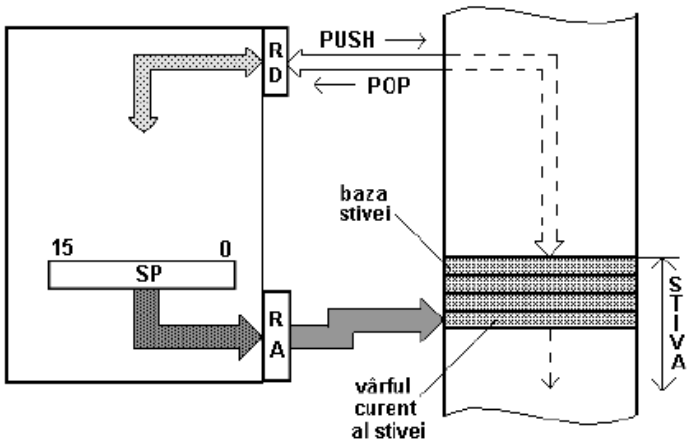
2| stiva µP este utilizata implicit in cateva actiuni importante ale µP
-transfer de date

3| SP-atribut arhiectural

4| SP- are dimensiunea dictate de dimensiunea adresei fizice daca folosim organizarea liniara a memoriei

Push=actiune de scriere

Pop=actiune de citire



adresa n

nu trebuie sa cunosc adresa trimiterea facandu-se la stiva

adresa 0

b3 → **tablou** - structura spatiala de date
- incarca registrul IX a adresei fizice a bazei tabloului → se face un tablou virtual
- orice acces in tablou se face invocand o adresa relativa =deplasament
- UCAM calculeaza adresa absoluta a unui element de tablou

Observatii:

1| ca si la stiva nu trebuie sa cunoastem adresa absoluta a unui element de tablou

2| spre deosebire de stiva registrul index (IX) nu se actualizeaza

3| IX-atribut arhiectural

4| dimensiunea registrelor IX depinde de dimensiunea adresei fizice adica dimensiunea hartii memoriei, atunci cand folosim organizarea liniara a memoriei

5| numarul de registre indez constituie un criteriu de performanta (cate tablouri potentiale trebuie sa am in memorie)

6| deplasamentul imi da dimesiunea virtuala a tabloului (8 biti= 256 locatii de memorie)

→Organizare liniara a memoriei

(PC) ← AF instrucțiune curentă **a1 si a2**

(SP) ← AF bază stivă **b2-stiva**

(SP) ← AF a vârfului stivei curente

(IX) ← AF bază tablou **b3-tablou**

AF element din tablou ← (IX) + disp

Exercitiu:

Fie un µP CISc cu schemele functionale din 2.1→2.4:
- µP are organizarea memoriei liniara
- adresa fizica pe 16 biti
- formatul memoriei un octet
- magistrala de date are 8 biti
- numaratorul program are 16 biti (=adresa fizica)
- un indicator de stiva are 16 biti (=adresa fizica)
- set de registre R1,R2,R3,R4 au 8 biti fiecare concatenate doua cate doua
-are un aumulator A pe 8 biti (operandul si rezultatul se afla in acumulator)

$$(A) \leftarrow ((R1)\uparrow(R2)) + ((R3)\uparrow(R4))$$

adresare indirecta

-continutul locatiei de memorie a acrei adresa se afla in (R1)↑(R2) se aduna cu continutul adresei (R3) ↑(R4) iar rezultatul ramane in acumulator

ciclu	stari		
1.	1.1	(RA) ← (PC) READ	-este adresata in memorie codul instructiunii curente
	1.2	(PC) ← (PC) + 1	-numaratorul este incrementat
	1.3	(RD) ← ((RA))	-codul instructiunii curente aduse in µP
	1.4	(RI) ← (RD)	-cod mutat registru special
	1.5	decodificare	-cod recunoscut
2.	2.1	(RA) ← ((R1)↑(R2)) READ	-se adreseaza primu operand
	2.2	(RD) ← ((RA))	-operand adus in µP
	2.3	(A) ← (RD)	- operand adus in acumulator
3.	3.1	(RA) ← ((R3) ↑(R4))	-se adreseaza al doilea operand
	3.2	(RD) ← ((RA))	-operand adus in µP
	3.3	(A) ← (RD) + (A)	-fac adunarea

Concluzii:

1| 3 cicluri de masina cu un numar de stari

2| 3 tipuri de cicluri masina

1. citire a codului instructiunii curente
2. citire din memorie
3. citire memorie si oretaie aritmetica

3| fiecare stare inseamna o actiune elementara

4| fizic fiecare actiune elementara reprezinta activarea,validarea unor registre,multiplexoare, decodoare, UAL

5| actiunile elementare sunt valabile pe schemele Block (2.1→2.4)
alte scheme block alte actiuni elementare

6| semantica fiecarei actiuni elementare este fundamentale
1.3≠2.2 din pct de vedere al semanticii

7| nu toate starile de pe ecran sunt realizabile; e posibil sa am actiuni care sa dureze mai mult de o stare.
2.1 si 3.1 mai intai aduc 8 biti si apoi inca 8.

…:::2.5 Pasul IV de detaliere: unitatea de control a µP:::…

doua functii

a. coordonarea aducerii din memorie a tuturor informatiilor necesare executiei corecte a unei instructiuni
(desfasurarea in spatiu a instructiunilor)

b. coordonarea activitatii tuturor componentelor µP a.i. oricare instructiuni sa se desfasoare corect
(desfasurarea in timp a instructiunilor)

a. formatul instructiunilor → toate informatiilor din memoria de program aferente executiei corecte a unei instructiuni

-prima informatie este codul=da semantica instructiunii

-cod format din →campuri
→operanzi
→adresa
→deplasament

b. desfasurarea in timp

-exista niste unitati de masura a actiunilor desfasurate in timp

asfel sunt doua masuri:
1. cicluri masina
2. **stare** = unitate fizica de timp (perioada de tact a µP) = 1ns

- intr-o stare µP realizeaza actiune elementara

cicluri masina = mai multe stari, µP realizeaza actiune bine definite in cadrul unei instructiuni

cicluri masina:
- scriere memorie
- citire memorie
- scriere stiva
- citire stiva

- scriere porturi
- citire porturi
- executie operatie aritmetico-logica…..

- un µP are de regula cateva cicluri masina care combinate in diverse feluri duc la executia oricarei instructiuni din setul de instructiuni al µP

Tipic µP CISC de a avea:

a) formatul instructiunilor diferit

b) desfasurarea in timp a instructiunilor difera in functie de complexitatea acestora

deosebire fundamentala RISC:

a) format riguros identic
b) desfasurare in timp identical

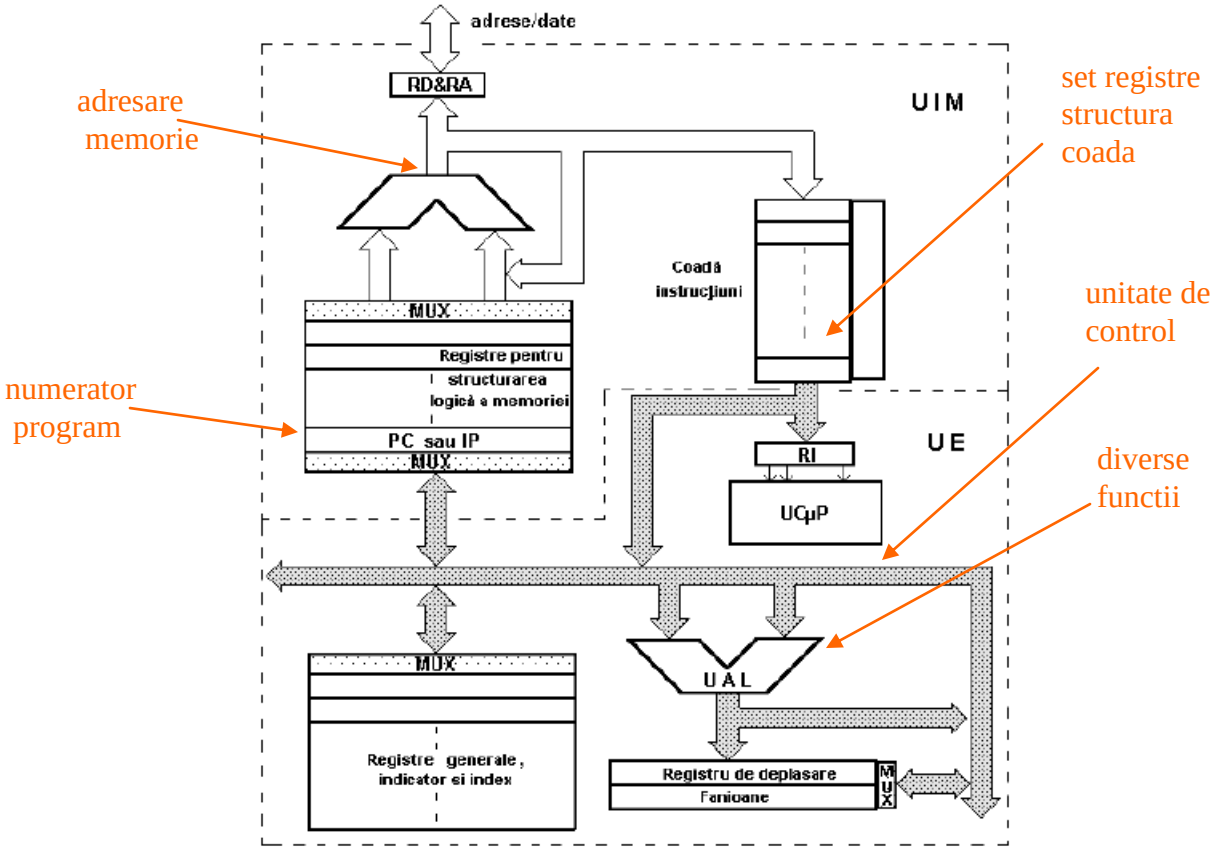
Intel x86 CISC:

-formatul instructiunilor : 1-6 octeti pentru 16 biti
1-15 octeti pentru 32 biti

- de la 4-100 stari pentru diferite instructiuni

III. Dezvoltarea functionala a unui µP pe 16 biti de uz general

…:::3.1 Organizarea functionala a unui µP 16 UG tipic:::…



Functii:

UE(unitate de executie)

- prelucreaza operanzii;
- contine registrele generale si ALU (Unitate aritmetica logica);
- accepta informatiile aferente instructiunilor deja aduse de catre UIM;
- primeste operanzii de la UIM;
- trimite rezultatele la UIM;
- contine unitatea de control al microprocesorului.

UIM (unitate interfata cu magistrala)

- furnizeaza (calculeaza) adrese pentru instructiuni si date;
- aduce instructiuni din memoria de program si le stocheaza într-o coada; (deosebire de a le astepta din memorie→viteza de executie mai mare)
- întrerupe umplerea cozii de instructiuni când se impune aducerea operanzilor sau trimiterea rezultatelor;
- realizeaza structurarea logica a memoriei (de pilda: organizarea segmentata a memoriei).

→ Unitatea Interfata cu Magistrala este o generalizare a UCAM

→operanzii primiti de catre UE de la UIM si apoi rezultatele sunt trimise catre UIM

CONCLUZIE:

-aceste unitati au functii distincte, lucreaza in parallel si duc la cresterea vitezei de executie

-generatia a treia are mai multe unitati

Noile Atribute:

1. *mai multe procesoare* care lucreaza in paralel, cu functii dinstincte.

2. *coada de instructiuni* (o unitate se ocupa cu realizarea unei cozi de instructiuni, aducerea in µP a unor instructiuni astfel incat dupa executia unei instructiuni µP nu mai asteapta aducerea instructiunii urmatoare)

→in momentul unui salt se goleste stiva si se continua de acolo

3. *versatilitatea functiilor registrelor : " register multifunctionale"*
   - multifunctional este opus dedicat
   - dedicat = functie a registrului stabilita de fabricant care nu poate Modificat(accumulator,stiva…)

   -registrele multifunctionale au functii
   -implicite
   -alternativa

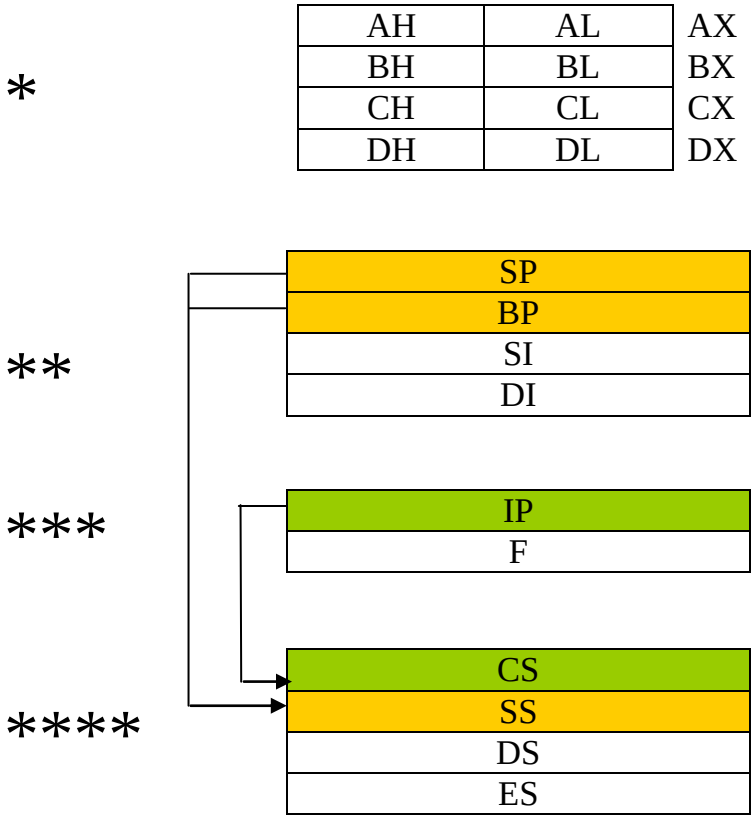
4. *unitate de calcul a adreselor*
   accesul in memorie se face calculand adrese(pointeri)

5. *structurarea logica a memoriei*
   optional sunt µP care realizeaza structuri logice de memorie

Se realizeaza alternative la organizarea liniara a memoriei ➔ organizare segmentata.

…:::3.2 Structura registrelor:::…

3.2.1 Microprocesoare Intel



---

* register generale=stocare temporala operanzi si sau rezultate
  - toate attribute de arhitectura
  - dimensiunile lor tine de tipul de µP
  - utilizabile si pe jumatati (cele pe 32pe sfert)
  - multifunctionale

AX (AH si AL)

-acumulatorul implicit pe 16 biti este de 16
pe 8 biti –AL
pe 32 biri – EAX

-alternative AX,AL,AEX pot fi register normale de date

BX (BH si BL)

-stocheaza o adresa registru de tip index, contine adresa de baza

-alternativ BX – acumulator
BL,BH – acumulatoare
BX – registru obisnuit de date

CX (CH si CL)

-functie de numerator → numara elementele
-contine implicit numarul dintr-un set de date
-este implicit contor intr-un ciclu cu contor

-alternativ ECX – acumulator
CL,CH – acumulatoare sau register obisnuite de date

DX (DH si DL)

-implicit este registru de date

-alternativ DX,DL,DH – acumulatoare

** register indicatoare si index

- sunt atribute de arhitectura
- sunt toate multifunctionale
- toate pe 16 sau 32
- nici unul accesibil pe bucati

SP

-indicator de stiva

-alternativ – acumulator si registru de date

BP

-indicator de stiva (stive secundare , intrare secundara stiva primara)

-alternativ – acumulator si registru de date

SI,DI

-registre index
-implicit – contin adrese doua categorii de siruri, sursa destinatie→adrese efective

-alternativ – acumulatoare si registru de date

*** register dedicate

IP

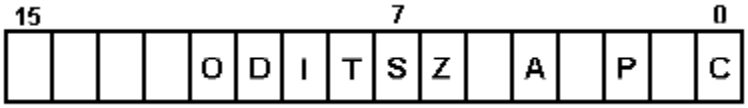
-indicator de instructiuni (numarator de program numai ca el nu contine adresa efectiva nu adresa fizica a instructiunii curente)

- nu e atribut de arhitectura

-are 16 biti pentru procesoare de 16 si 32 pentru procesoarele de 32

F

-registru de fanioane



-registru virtual
→ care e alcatuit din concatenare de bistabili si cu cellule care contin informatii

- e atribut de arhitectura

- numele registrului pe 16 biti F sip e 32 biti EF

fanioane logico - aritmetice

C – fanion transport

P – fanion de paritate

A – fanion de transport auxiliar (1 si 2 nibble→aritmetica ZCB)

Z – fanion de 0 (setat 1 de fiecare data cand rezultatul unei operantii e 1)

S – fanion de semn, preia semnul rezultatului unei operatii aritmetice intre 2 intregi cu semn

O – fanion de depasire

fanioane generale – stocheaza fenomene oarecare din µP

D – fanion de directie si permite parcurgerea unui sir de date, numere in ordine crescatoare sau descrescatoare a adreselor elementelor sirului

I – fanion de validare a unor intreruperi

T – fanion de capcana si foloseste pentru a propune procesorului functionarea pas cu pas (instructiune cu instructiune)

**** registre segment= dedicate, ele contin niste entitati numite adrese segment

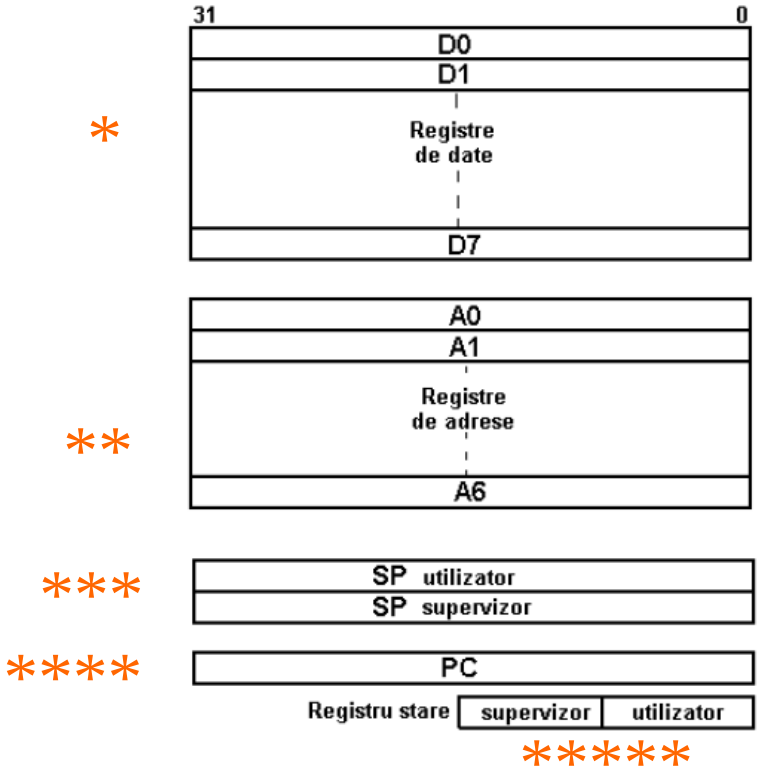
- ele folosesc pentru realizarea in memoria µP a unor diviziuni logice numite segmente

-aceasta segmentare este o alternative la organizarea liniara

-atribute de arhiectura toate

-16 biti toate

3.2.1 Microprocesoare Motorola



---

* registre de date (8) = stocare temporala a informatiilor in sesnsul 2.2
  - toate atribute de arhitectura
  - toate multifunctionale
  - toate pe 32 biti
  - accesibile si pe portiuni

-functie implicita – registre generale de date

-functie alternativa – a) acumulatoare
b) registre index (adrese de tabele) →8 tabele in meorie

** registre de adrese (7) = functie de baza de a stoca adrese

- toate atribute de arhitectura
- toate multifunctionale
- toate pe 32 biti
- sunt accesibile si pe portiuni

-functie implicita – indicatoare de stiva alternativa ➔ 7 stive

-functie alternativa – a) registre index → inca 7 tabele
b) acumulatoare
c) registre generale de date

*** registre dedicate

uSP – SP al utilizatorului
sSP – SP al supervizorului

-2 indicative de stiva primara,contine adresele fizice ale celor doua stive

2 categorii de utilizatori:
1. obisnuit→drepturi limitate resurse sisteme
2. supervizorul → utilizator preferential si drepturi nelimitate

- aceste doua stive constituie *mecanism de protectie*

**** PC – numarator de program (adresa fizica a instructiunii curente)
- nu e atribut de arhitectura

***** registru de stare = registre de fanioane

pe 16 biti-supervizor
pe 8 biti – utilizator

GLOBAL. Caracteristicile seturilor de registre CISC

1. *numar registre*

   µP CISC –numar mic de registre (8 registre Intel, 15 Motorola)

   este clar ca Motorola are mai multe registre decat Intel

2. *dimensiunea registrelor*

   -dimensiunea operanzilor de lucru

   16 biti pe µP de 16
   32 biti pe µP de 16

   Motorola are dimensiunea registrelor mai mare decat la Intel

3. *majoritatea registrelor sunt multifunctionale*

   functii implicite/alternative:

   a) acumulatoare
   b) registre de date
   c) registre index
   d) indicator de stiva
   e) numarator

   -si din acest punct de vedere motorola superior

4. *registrele folosesc pentru*

   i. realizarea mai multor stive
   ii. realizarea mai multor tablouri

   -si din acest punct de vedere Motorola superior (motorola 15 potentiale, Intel 3 potentiale)

5. *registrele folosesc pentru realizarea unor mecanisme superioare*

   a. la Intel structurarea segmentata a memoriei (registre segment)
   b. la Motorola mecanism de protectie cu 2 nivele (uSP si sSP)

…:::3.3 Organizarea memoriei µcalculatorului:::…

1. dimensiunile hartii memoriei → ce memorie poate sa acceseze memoria fizica

2. formatul memoriei → memoria trebuie sa asigure formatul

3. conventiile de stocare in memorie a datelor care ocupa mai mult de o locatie

4. structura logica a memoriei → organizarea propriu-zisa

3.3.1 Microprocesoare Intel in modul real

1. harta memoriei – data de dimensiunea adresei fizice, 20 biti adica 1 megalocatii.

2. formatul memoriei – octetul

3. informatii stocate pe octeti succesivi care se acceseaza conform conventiei micului indian (→informatia cea mai putin semnificativa este pe adresa cea mai mica)

4. imi ofera alternativa
→*organizarea segmentata a memoriei*

Intel x86 → realizarea in memoria fizica a niste subdiviziuni logice numite segmente

organizare liniara
-adresa fizica : AF

organizare segmentata
-adresa logica AL = adr32 compusa din:
adresa segment AS = adr16
adresa efectiva AE = adr16

-adresa logica este atribut de arhitectura

-ori de cate ori faci referinta utilizatorul indica adresa efectiva si adresa segment; pentru adresare se foloseste adresa fizica care este calculata pornind de la adresa logica → translatarea adresei logice

-adresa fizica a bazei unui segment se face inmultimn cu 16 baza AS iar apoin in interiorul segmentului pozitionarea se face adunand AE la el

$$AF_{bazei\ segmentului} = AS \uparrow 0000$$

$$AF = AS \uparrow 0000 + AE$$

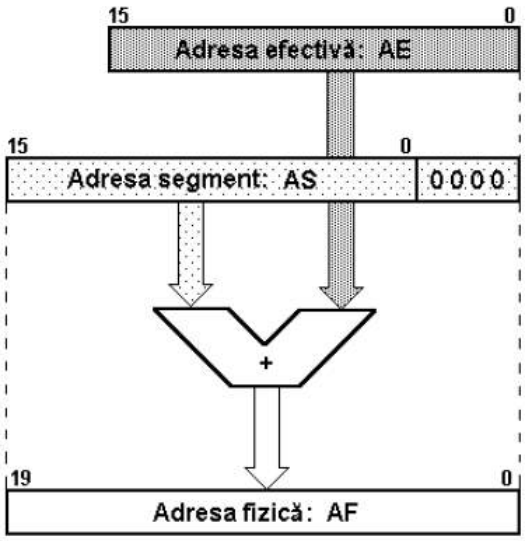
$$AF = AS \uparrow 0H + AE$$

-ultima e formula de translatare din adresa logica in cea fizica

Observatii:

1. prima deosebire in organizarea liniara si cea segmentata este ca in cazul organizarii liniara adresa fizica este atribut de arhitectura in schimb in cea segmentata nu este atribut de arhitectura.

2. adresarea segmentata. → pozitionarea segmentului in memorie si apoi adresare liniara in segment.



-se pierd 12 biti, dar ei se regasesc in organizarea virtuala in memorie

AS = (CS) | (SS) | (DS) | (ES) [| (FS) | (GS)]

-adresa segment se afla in registru segment

AE = (SP) | (BP) | (SI) |(DI) | (IP) | (BX) | adr

-adresa efectiva (nu fizica !)

AS impune:

1. numarul se segmente realizabile simultan

cate registre segment am atatea segmente pot sa realizez

2. tipurile de segmente

Intel imi impune ca in functie de registrul segment sa folosesc in memorie segmente cu o functie stabilita in memorie

-segment curent de program (CS)
-segment curent stiva principala (SS)
-segment de date (DS)
-segment suplimentar de date (ES,FS,GS)

3. contribuie la stabilirea pozitiei absolute a segmentelor

utilizata pentru stabilirea pozitiei absolute in memoria fizica a segmentului (adrese fizice multiple de 16)

4. pozitia relativa a segmentelor

pozitiile relative ale segmentelor sunt la latitudinea utilizatorului
segmentele pot si suprapuse

a) total
b) partial
c) deloc

AE impune:

-dimensiunea segmentelor

…::::Redefinirea utilizarii unui segment :::…

→redirectionare

Destinația	AF implicită	AF modificată
program	(CS) ↑ 0H + (IP)	NU
stiva primară	(SS) ↑ 0H + (SP)	NU
stiva alternativă	(SS) ↑ 0H + (BP)	(CS) ↑ 0H + (BP) (DS) ↑ 0H + (BP) (ES) ↑ 0H + (BP)
date	(DS) ↑ 0H + AE	(CS) ↑ 0H + AE (SS) ↑ 0H + AE (ES) ↑ 0H + AE
	în care: AE = (BX) | (SI) | (DI) | adr	

Observatii:

1. *redirectionarea segmentelor* → utilizarea unor segmente pentru alte informatii stabilite implicit de fabricant

2. redirecionarea inseamna ca segmentele in memorie pot avea functii implicite si alternative

3. redirectionarea segmentelor este restrictionata

4. redirectionarea se face in mod explicit in program cu un prefix de redirectionare (un mnemonic in assembler)

-segmentul de program poate fi dedicat sau multifunctional

→suprapunerea segmentelor

-utilizarea fara restrictii

programe relocabile dinamic

→ care folosesc toate informatiile intr-un singur segment, nu depasesc granitele segmentului.

3.3.2 Microprocesoare Motorola in modul real

1. harta memoriei – data de adresa fizica pe 24 biti

$$AF \equiv adr24$$

-16 megalocatii

2. locatia de memorie – 2 octeti

3. conventia de utilizarea a mai multor locatii, conventia micului indian

4. organizarea memoriei – mod liniar

-direct adresa fizica care este atribut arhitectura

$$AF \equiv AE = (PC) \mid (Ai)_{i=0..6} \mid (USP) \mid (SSP) \mid (Di)_{i=0..7} \mid adr24$$

GLOBAL. Concluzii organizare memorie ale µP CISC

1. dimensiunea hartii memoriei din ce in ce mai mare

2. formatul memoriei → 1 octet sau doi octeti

3.micul/marele indian – conventii utilizate la accesarea mai multor locatii succesive

4. unele au organizare liniara si altele au organizari alternative

Organizare liniara a memoriei versus **Organizare segmentata a memoriei**

-timp de access → pentru segmentare timp indelungat de access dar odata inauntru timpul de accesare este rapid

-ocuparea hartii memoriei → org. liniara mai eficace, org. segment risipeste memoria

-modularizare → in favoarea organizarii segmentate

µP are org. segmentata cu 2 tipuri: -intrasegment
-intersegment

IV. Principii de baza ale arhitecturii CISC

…:::4.1 Transferuri de date :::…

Transferurile de date: deplasarile de operanzi si/sau rezultate în interiorul microprocesorului, între microprocesor si celelalte componente ale microcalculatorului, sau între microcalculator si lumea exterioara.

Din punctul de vedere al complexitatii transferurilor de date, se poate face urmatoarea clasificare a microprocesoarelor de uz general:

- procesoare categoria A → acumulator dedicat care ia parte la orice transfer de date

$$(r_i) \leftrightarrow (A)$$
$$(mem) \leftrightarrow (A)$$
$$(port) \leftrightarrow (A)$$

- procesoare categoria B → poate transfera direct cu alte registre
  → au toate caracteristicile lui A

$$(r_i) \leftrightarrow (r_j)$$
$$(mem) \leftrightarrow (r_i)$$
$$(port) \leftrightarrow (r_i)$$

- procesoare categoria C → transfera direct din locatie de memorie
  → au toate caracteristicile lui A si B

$$(mem_i) \leftrightarrow (mem_j)$$

- procesoare categoria D → pot transfera bloc de memorie dintr-o parte in alta
  → au toate caracteristicile lui A si B si C

```
for i = 1 to n do
    begin
        (mem_i) ↔ (mem_j)
        j ← j + 1
    end
```

registrele sunt atribute de arhitectura→ daca iau parte la transfer de date

…:::4.2 Tehnici de adresare :::…

Tehnica (mod) de adresare: modalitatea de specificare, în formatul unei instructiuni, a locatiei (adresei) unui operand, rezultat sau a codului altei instructiuni.

1. Adresare implicita (în registru):

Un registru (pereche de registre) este specificat (specificata) într-unul dintre cîmpurile codului instructiunii.

cod semantică	cod reg. destinație	cod reg. sursă

→modul sau tehnica adresarii care presupune ca in chiar codul instructiunii curente se specifica registrul sau registrele de arhitectura unde se gaseste informatia vizata

2. Adresare imediata :

→modul sau tehnica adresarii care presupune ca in format, imediat dupa cod se gaseste informatia vizata (adica informatia se afla in memoria de program)

→ informatia vizata o data, date doar operanzi nu rezultate(de regula constante)

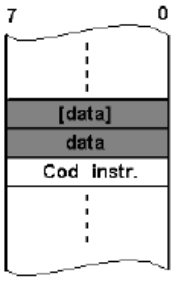
→2 octeti minim ca format

!fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti :

data8 = ((PC) + 1)

sau

data16 = ((PC) + 2) ↑ ((PC) + 1)



3. Adresare absoluta (extinsa, directa) :

→modul sau tehnica adresarii care presupune ca in format, imediat dupa codul instructiunii se gaseste adresa completa a informatiei vizate

→ informatie vizata si la date si la instructiuni (adresare absoluta directa sau extinsa)

→formatul este mai mare, de 3 octeti

adresa completa – totalul informatiei necesare pentru localizarea in memorie sau porturi a datelor sau cand e cazul instructiunii

a) daca organizarea memoriei este liniara

adresa completa = adresa fizica

b) daca organizarea memoriei este segmentata

b1) accesul este intrasegment

adresa efectiva = adresa completa

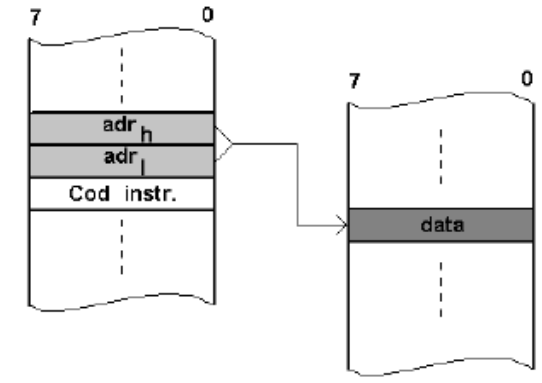
b2) accesul este intersegment

adresa logica=adresa completa

! fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti :

$data = (adr_h \uparrow adr_l)$

unde $adr_l = ((PC) + 1)$ şi $adr_h = ((PC) +2)$



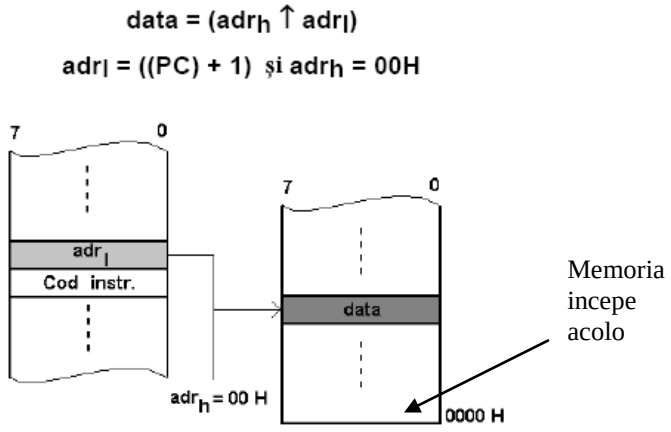
4. Adresare scurta :

→modul sau tehnica adresarii care presupune ca in format, imediat dupa codul instructiunii se gaseste o parte din adresa completa a informatiei vizate,cealalta parte fiind presupusa implicit

→se refera atat la date cat si la instructiuni, deriva din adresa completa

→ format de minim 2 octeti(1 octet cod+1/2 octet la adresa completa)

! fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti (1/2 inferior → formatul instructiunii si ½ superior implicit 0):



5. Adresare relativa :

→modul sau tehnica adresarii care presupune ca in format, imediat dupa codul instructiunii am pozitia relativa a informatiei vizate, relativa la dresa instructiunii curente

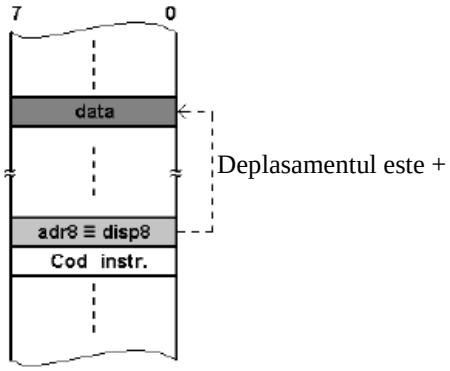
pozitie relativa data de deplasament care e un numar cu semn (ma pot duce inainte sau dupa)

→informatia vizata poate fi data sau instructiune (de obicei aceasta modalitate de adresare e folosita la cicluri cu contor)

→ format minim 2 octeti minim (1 octet de cod si deplasamentul minim 1 octet)

! fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti:

data = ((PC) ± disp8 | disp16)



+,- 128 de pozitii poate ataca deplasamentul

6a. Adresare indirecta prin registru :

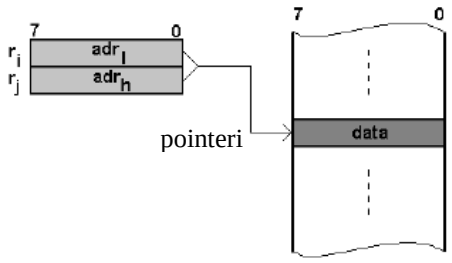
→modul sau tehnica adresarii care presupune ca in format, in chiar codul instructiunii se indica un registru sau o pereche de registre in care se gaseste adresa completa a informatiei vizate

→mod de adresare folosit pentru date si instructiuni

-este un criteriu de performanta Intel procesor puternic

→ formatul este de minim 1 octet

! fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti, o pereche registre (atrib.arhitectura) de 8 biti fiecare :



data = ((r_i) ↑ (r_j)) | ((r16))

6b. Adresare indirecta cu memoria :

→modul sau tehnica adresarii care presupune ca in format, dupa codul instructiunii exista adresa adresei vizate

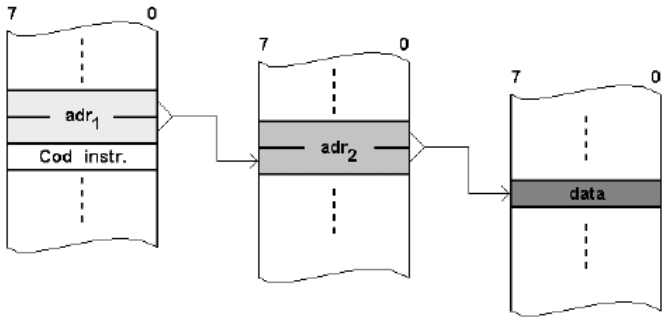
Observatie: toatea adresele implicate sunt complete

→ mod de adresare de regula pentru date dar si pentru instructiuni

→ format extins de cel putin 3 octeti

! fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti, conventia micului indian:

modul de adresare imi ofera relativ independenta intre program si datele vizate.



data = (adr2) = ((adr1))

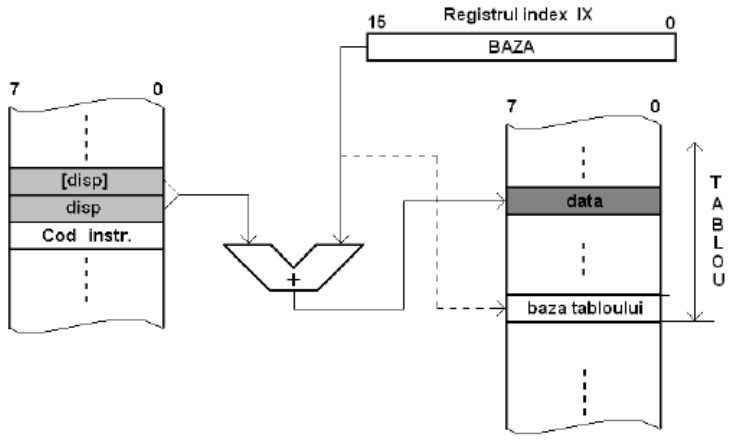
7a. Adresare cu preindexare :

→modul sau tehnica adresarii care presupune ca in formatul instructiunii curente exista pozitia relativa a informatiei vizate intr-un tablou de date, pozitia relativa faza de baza tabloului = deplasament (care aici e +)

→modul de adresare se refera la date

→formatul mai compact decat in cazul adresarii absolute, minim 1 octet

!fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti, cel putin un registru index de 16 biti, deplasament de 8,16 biti:



data = ((IX) +disp8) = ('BAZA' +disp8)

sau

data = ((IX) +disp16) = ('BAZA' +disp16)

Observatie:

- nr. regsitre index –criteriu de performanta
- marimea deplasamentului imi da marimea tabloului (2^{disp})

7b. Adresare cu postindexare :

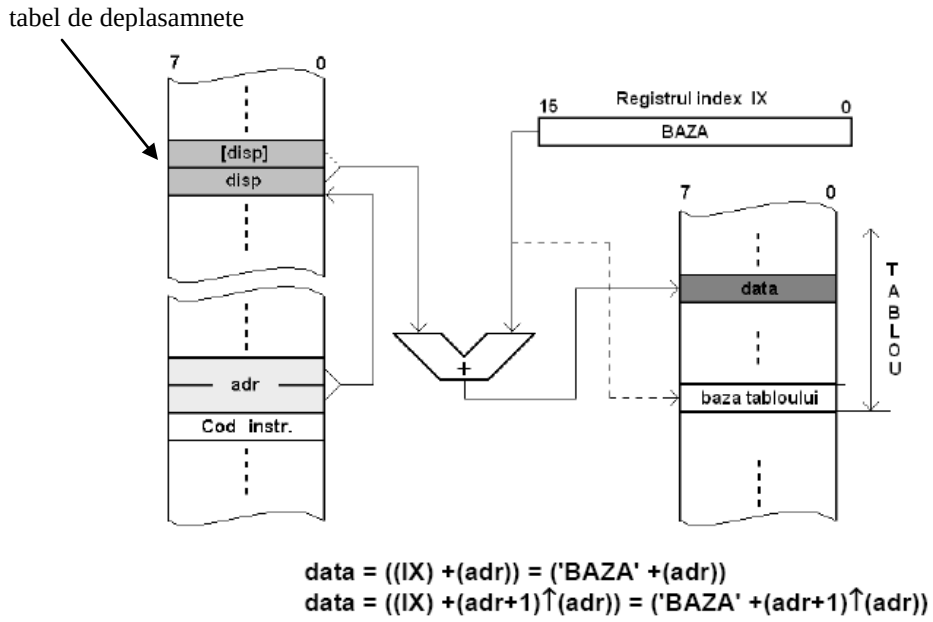
→modul sau tehnica adresarii care presupune ca in formatul instructiunii curente am adresa pozitiei relative a informatiei vizate uintr-un tablou de date (combinatie a lui 7a cu 6b)

→modul de adresare se refera la date

→formatul minim este de cel putin 3 octeti

AVANTAJ : imi ofera relativa independenta program si pozitie tabel ale datelor vizate

! fie un µP care organizeaza liniar memoria, fie formatul un octet si o adresa fizica de 2 octeti, un tabel de deplasamente:



data = ((IX) +(adr)) = ('BAZA' +(adr))
data = ((IX) +(adr+1)↑(adr)) = ('BAZA' +(adr+1)↑(adr))

Concluzii:

-toate sunt moduri de adresare simple

-mai exista moduri de adresare compuse care rezulta din combinarea mai multor moduri de adresare

…:::4.3 Tehnici de adresare tipice µP pe 16 biti :::…

4.3.1 Tehnici de adresare pentru Intel 8086

a) Adresarea memoriei de program:

1. **Adresarea relativă**:
   (IP) ← (IP) + disp8|disp16

2. **Adresarea directă** ("salturile intersegment"):
   (IP) ← adr32|
   (CS) ← adr32$_h$.

b) Adresarea memoriei de date (operanzi/rezultate):

1. **Adresarea imediată**:
   AF = [(CS)↑0H+(IP)+2 ↑] (CS)↑0H+(IP)+1

2. **Adresarea directă**:
   AF = (DS)↑0H + adr8|adr16 ,
   sau:
   AF = (DS)↑0H + [((CS)↑0H+(IP)+2) ↑](CS)↑0H+(IP)+1)

3. **Adresarea indexată**:
   AF = (DS)↑0H + (SI)|(DI) + disp8|disp16
   adică:
   AF = (DS)↑0H + (SI)|(DI) + [((CS)↑0H+(IP)+2) ↑](CS)↑0H+(IP)+1)

4. **Adresarea indirectă-implicită**:
   AF = (DS)↑0H +(SI)|(DI)

e posibila redirectionarea segmentelor

5. **Adresarea relativă la bază**:
   5.1. **Adresare relativă la bază directă**:
   AF = (DS)↑0H + (BX) + adr8|adr16 .
   cu: **adr8 = ((CS)↑0H+(IP)+1) ,**
   sau: **adr16 = ((CS)↑0H+(IP)+2) ↑ ((CS)↑0H+(IP)+1)**

Adresare indirecta + Adresare directa prin registru

   5.2. **Adresare relativă la bază indexată**:
   AF = (DS)↑0H + (BX) + (SI)|(DI) + disp8|disp16 ;
   cu: **disp8 = ((CS)↑0H+(IP)+1) ,**
   sau: **disp16 = ((CS)↑0H+(IP)+2) ↑ ((CS)↑0H+(IP)+1) .**

Adresare indirecta prin registre + Adresare cu preindexare

5.3. **Adresare relativă la bază implicită**:
AF = (DS)↑0H + (BX) + (SI)|(DI) 2 adresari indirecte prin registre

6. **Adresarea în stivă**:

6.1. **Adresare în stivă directă**:
AF = (SS)↑0H + (BP) + adr8|adr16 ,
cu: **adr8 = ((CS)↑0H+(IP)+1)** ,
sau: **adr16 = ((CS)↑0H+(IP)+2) ↑ ((CS)↑0H+(IP)+1)** .

6.2. **Adresare în stivă indexată**:
AF = (SS)↑0H + (BP) + (SI)|(DI) + disp8|disp16 ;
cu: **disp8 = ((CS)↑0H+(IP)+1)** ,
sau: **disp16 = ((CS)↑0H+(IP)+2) ↑ ((CS)↑0H+(IP)+1)** .

6.3. **Adresare în stivă implicită**:
AF = (SS)↑0H + (BP) + (SI)|(DI)

7. **Adresarea în registru**:
pe 8 biți: **AF = AL | AH | BL | BH | CL | CH | DL | DH** ,
iar pe 16 biți: **AF = AX | BX | CX | DX | SP | BP | SI | DI** .

4.3.1 Tehnici de adresare pentru Motorola 68000

■ **Adresarea în registru** („directă în registru”):

1. **În registre de date**:
AF = Dn, cu **n = 0,...,7**
data = (Dn)

2. **În registre de adrese**:
AF = An cu **n = 0,...,6**,
data = (An).

■ **Adresarea absolută**:

3. **Adresare absolută scurtă**:
AF ≡ adr16 = ((PC) + 1).

4. **Adresare absolută lungă**:
AF ≡ adr32 = ((PC) + 2) ↑ ((PC) + 1).

■ **Adresarea relativă**:

5. **Adresare relativă propriu-zisă** („cu deplasament"):

AF = (PC) + disp16,

în care: **disp16 = ((PC) + 1)**.

6. **Adresare relativă cu index şi deplasament**:

AF = (PC) + (Xn) + disp8,

în care **Xn = An|Dn** şi **disp8 = ((PC) + 1)|**.

■ **Adresarea indirectă prin registru** are şi ea mai multe variante:

7. **Adresare indirectă prin registru „simplă"**:

AF = (An),

adică: **data = ((An))**.

8. **Adresare indirectă prin registru cu deplasament**:

AF = (An) + disp16.

adică: **AF = (An) + ((PC) +1)**.

9. **Adresare indirectă prin registru cu deplasament şi bază**:

AF = (Xn) + (An) +disp8.

adică: **data = ((Dn)|(An) + (An) + ((PC) + 1)|)**,

10. **Adresare indirectă prin registru cu post-incrementare**:

AF = (An), (An) ← (An) + N,

11. **Adresare indirectă prin registru cu pre-decrementare**:

(An) ← (An) - N, AF = (An),

■ **Adresarea imediată**:

12. **Adresare imediată propriu-zisă**:

data8 = ((PC) + 1)|,

data16 = ((PC) + 1)

şi: **data32 = ((PC) + 2) ↑ ((PC) + 1)**.

13. **Adresare imediată „rapidă"**.

■ **Adresarea implicită în registre speciale**:

14. **AF = SR**,

AF = USP,

AF = SSP,

sau **AF = PC**.

…:::4.4 Tipuri de instructiuni:::…

-din puct de vedere semantic µP imparte setul de instructiuni in 5 subseturi:

1) transfer de date
2) prelucrari date
3) instructiuni de control al programului
4) instructiuni in/out
5)instructiuni de control al µcalculatorului

4.4.1 Transfer de date

(d) ← (s)

-sursa e copiata la destinatie iar dupa transfer sursa e neschimbata

1. un µP realizeaza acele tipuri de transferuri conform clasei din care face parte asa cum am facut clasificarea in 4.1

2. sursa si destinatia pot fi registre atribute de arhitectura si/sau locatii de memorie

3. dimensiunea sursei si destinatiei sunt identice

4. identificarea corecta a sursei si a destinatiei se face utilizand un mod de adresare specific tipului procesorului respectiv, cu restrictii

5. criteriul de performanta pentru aceste instructiuni de transfer este un factor de merit intre semantica si formatul instructiunii

exercitiu 1
-fie µP cu organizarea memoriei liniara, memorie formata in octeti, adrese fizice pe 16 biti, R1 si R2 registre pe 8 biti fiecare.

PUSH R1R2; (SP) ←(SP)-1 -actualizare varf stiva
((SP))← (R2) -trimit cate 8
(SP) ← (SP)-1 -actualizare varf stiva
((SP)) ← (R1) -iar trimit cate 8
POP R1R2 (R1) ← ((SP))
(SP) ← (SP)+1
(R2) ← ((SP))
(SP) ← (SP)+1

exercitiu 2
-fie µP cu organizarea memoriei liniara, memorie formata in octeti, adrese fizice pe 16 biti, R1,R2,R3,R4,R5,R6 registre pe 8 biti fiecare, concatenate doua cate doua.

-perechea R3R4 este preincarcata cu adresa fizica a primului element a unui sir numit sursa

-perechea R1R2 este preincarcata cu adresa fizica a primului element a unui sir numit destinatie

-perechea R5R6 este preincarcata cu un numar reprezentand numarul de elemente din sirul sursa

Repeat
((R1,R2)) ← ((R3,R4)) - adresarea indirecta a sursei si a destinatiei
(R1,R2) ← (R1,R2) + 1 – autoincrementarea adreselor
(R3,R4) ← (R3,R4) + 1 – autoincrementarea adreselor
(R5,R6) ← (R5,R6) + 1 – autoincrementarea contorului
Until (R5,R6) = 0H – ciclu cu test final

-tipic CISC

Transfer

MOV [1064H],AX ((DS)↕0H+1064H) ← (AL)
((DS)↕0H+1065H) ← (AH)

MOV [2000H],DS ((DS)↕0H+2001H)↕ ((DS)↕0H+2000H) ← (DS)

MOV [BX],491FH ((DS)↕0H+(BX)) ← 1FH
((DS)↕0H+(BX)+1) ← 49H

DS nu e utilizabil pe jumatati

segmentul de stiva nu e redirectionabil, dar cel de date este
(PUSH → intai actualizez si apoi efectuez transferal,POP – efectuez transferal, apoi actualizez)

XLAT (translate,instructiune pe 8 biti)
(Al) ← ((DS)↕0H+(BX)+(BL))

MOVSB / MOVSW (move stream byte, move stream word)
-primitive de transfer sir
-aici trebuie initializat DS cu adresa segment unde se afla sirul sursa, ES va fi destinatia segmentului,DF-ne spune cum trebuie parcurs sirul,intr-un sens sau in celalalt iar CX contine numarul de elemente din sirul sursa

daca destinatia sirului se suprapune cu sirul s pe un nr de elemente atunci vom realize o redirectionare, adica vom parcurge sirul dintr-o directie si apoi din cealalta

SS- nu se redirectioneaza
DS – se redirectioneaza

4.4.2 Instructiuni prelucrari de date

- operatii monadice (un singur operand)
  - cu acumulator (d) ← ⊗(d)
  - fara accumulator (d) ← ⊗(s)
- operatii diadice (doi operanzi)
  - cu acumulator (d) ← (d)⊗(s)
  - fara acumulator (d) ← (s1)⊗(s2)

⊗ → operatie logica aritmetica oarecare dintre care le vom enumera imediat.

Operatii accumulator → tipic CISC
Operatii fara accumulator → tipic RISC

CARACTERISTICI

1.tipuri de opeartii uzuale sunt:
a) logice : SI,SAU,XOR, complement fata de 1 si de 2 (CF1 si CF2)

b) aritmetice: +,-, . , :,inc,dec
(. si : sunt tipice CISC)
c) deplasari si rotatii, . , : la 2

2.dimensiunea rezultatului si a operanzilor este aceeasi (cu exceptia . , :)

3.identificarea operanzilor si a rezultatului se face folosind modurile de adresare ale µP respective,cu restrictii.

Operanzii si/sau rezultatul in registre pot fi atribute de arhitectura sau/si in memorie.

4.in semantica acestor instructiuni intra si modul in care sunt afectate fanioanele aritmetico-logice.

4 moduri de afectare a fanioanelor

a) fanioane neafectate indifferent de operatii (nu pot fi folosite in a lua decizii ulterioare)
b) fanioane afectate ulterior
c) fanioane care sunt fie setate fie resetate dupa o operatie indifferent de rezultat
d) fanioane care in functie de tipul operanzilor, in functie de valoarea operanzilor sunt setate sau resetate

5. Criteriul de performanta pentru aceste instructiuni este un factor de merit intre semantica operatiei si timpul de executie

!!! CISC - timpul de executie depinde de complexitatea operatiei
(ex: . are 5 stari,: 150 stari)

exercitiu 4
-fie µP cu registrele R1, R2 pe 8 biti, cu fanion de transport optional

(R1,R2) ← (R1,R2)+(R3,R4) [+(C)]

instructiuni ADD si ADC (add+carry)

exercitiu 5
-fie µP cu acumulator dedicat A, si (s) un alt registru
-semantica unei comparatii care nu se stocheaza in acumulator ci ma uit in fanioane
(A) – (s) if (A)=(s) then (Z) ← 1

if (A)>(s) then (C) ← 0
else (C) ← 1
ADC AX,[BX]
- (AX)←(AX)+((DS)↕0H+(BX)+1)↕(AX)←(AX)+((DS)↕0H+(BX))+(CF) (add+carry)
-acumulator AX
-operanzi → implicit si relative la baza directa
-segmentul de date poate fi redirectionat

SUB DH,[BP+4] (DH) ← (DH)-((SS)↕0H+(BP)+4)
-al doilea operand in stiva directa
-DH acumulator , 8 biti
-pot sa redirectionez pentru ca exista BP (SS+SP) - nu am voie
(SS+BP) – am voie

inca o primitive:
CMPSB/CMPSW = comparatii intre doua siruri
-trebuie sa initializez ES,DS,SI,DI,DF

4.4.3 Instructiuni de control al programului

-test
-salturi (decizii si cicluri)

TESTE:
A.testeaza un fanion
B. testeaza combinatii logice fanioane
C. testeaza un bit oarecare dintr-un operand oarecare
D. testeaza linii de intrare “serie”

SALTURI:

-dupa cum urmeaza sau nu unor teste:
- neconditionate
-conditionate

-dupa modul de adresare(mult mai retrictiv)
CISC foloseste patru salturi:
-adresare directa
-adresare scurta
-adresare relativa
-adresare indirecta prin registre

-dupa cum memoreaza sau nu valoarea curenta a numaratorului de program
-salturi propriu-zise
-apeluri de subprograme

-dupa complexitate
A.salturi 2 posibilitati
B. salturi 3 posibilitati
C. cicluri

criteriu de performanta: factor de merit: tipul de salt si tipul de test

exercitiu 6
-fie µP cu organizarea memoriei liniara CISC, memorie formata in octeti, adrese fizice pe 16 biti, numerator program PC si indicator de stiva SP de 16 biti

Moduri de adresare pentru salturile propriu-zise

- adresare absoluta

(PC) ← adr
in care **adr** face parte din formatul instructiunii

- adresare relativa

(PC) ← (PC) + disp8 | disp16
in care **disp8** si **disp16** fac parte din formatul instructiunii

- adresare indirecta prin registru

(PC) ← ((ri,rj)+1) ↑ ((ri,rj))

- adresare scurta (in "pagina 0")

(PC)l ← adr8
(PC)h ← 0H

Moduri de adresare pentru apelurile de subprograme

CALL adr; (SP) ← (SP) – 1
((SP)) ← (PC)h
(SP) ← (SP) – 1
((SP)) ← (PC)l
(PC) ← adr -salt adresare directa

Obs1. folosire implicita a stivei primare (PUSH,POP,CALL,RET)

RET (PC)l ← ((SP))
(SP) ← (SP) +1
(PC)h ← ((SP))
(SP) ← (SP)+1

Obs2. nu stii exact locatia din stiva, asta inseamna ca nun e scuteste de obligatia de a tine cont de numarul de accesari a stivei

CAL,PUSH,RET – GRESIT!!!
CAL,PUSH,POP,RET -corect

Obs3. in limbaj de programare nu exista nici un mod implicit de a transfera parametrii cand se face apelarea de program

4.4.3 Instructiuni intrare/iesire

-transferuri de date la si de la porturi

→ aceste instructiuni presupun ca μP face o harta a porturilor distincta de harta memoriei

IN d,port; (d) ← (port)
OUT port,s; (port) ← (s)

-dimensiunea sursei si destinatiei identice

Caracteristici speciale:

- **d** sau **s** acumulator implicit si dedicat

- **harta porturilor este mult mai mica decat harta memoriei**

- moduri de adresare foarte restrictive pentru porturi **: directa** si **indirecta prin registru**

Criteriu de performanta → acces rapid la porturi

exemplu.

Intel x8086 → IN
→OUT

- mod real, d si s sunt acumulatorii impliciti AL,AX,EAX
- porturi organizate pe octet ca memoria

- harta porturi – 64 kiloporturi/ octet
- adresare directa : 1 octet → 256 porturi / octet
- adresare indirecta prin registru: DX → 64 kiloporturi/octet

4.4.5 Instructiuni de control al µP

- instructiuni care sincronizeaza µP cu anumite stari ale lui
  - stare de asteptare
  - functionare pas cu pas
  - acces direct la memorie
  - partajarea resurselor sistemului cu alte procesoare
- accesul la anumite fanioane (setare,resetare), in special fanioanele care semnaleaza actiuni specifice (ca de pilda validarea cererilor de intrerupere)

Concluzii set de instructiuni ale µP CISC

1. un procesor CISC are un set de instructiuni bogat

2. o parte din instructiuni sunt complexe
   → operatii cu siruri, cicluri cu contor(loop), operatii aritmetice complicate

3. setul de instructiuni puternic influentat de alte atribute de arhitectura: utilizarea registrelor, tipuri de transfer de date, organizarea memoriei, tehnici de adresare

4. instructiuni CISC → formate diferite (moduri de adresare , organizare memorie) si au timpii de executie foarte diferiti

V. Principii de baza ale unei arhitecturi RISC

RISC → procesoare cu set redus de instructiuni

la procesoarele RISC → structura si arhitectura sunt simplificate

Concluzii:

- 80% timp de procesor CISC deserveste 20% instructiuni

- exista instructiuni complexe inlocuite in unele cazuri , deci timp de prelucrare mai mic

- transferurile intre registre, intre memorie si registre, instructiuni aritmetico logice, instructiuni de control al programului , salturile = 30% din program

OBSERVATII:

1. Procesoarele RISC se bazeaza pe arhitectura Von Neumann
2. Aceste procesoare au aceleasi tipuri de arhitectura

CARACTERISTICI definitorii RISC:

1. Unitatea de control este realizata prin "logica de tip cablat"
2. Instructiunile se desfasoara intr-o singura stare → durate timp identice
3. Numar instructiuni este mic (sub 128)
4. Dimensiunea este fixa pentru tot formatul instructiunilor
5. Instructiunile au format uniform. (nu mai mult de 4 tipuri de formate)
6. Numarul modurilor de adresare este mic (sub 4) → nu neaparat simple
7. Exista un numar mare de registre generale, atribute de arhitectura (cel putin 32)
8. Accesul memoriei se face numai cu instructiuni de transferuri simple de date intre registre si memorie (instructiuni tip "LOAD" si STORE"), operatiile de prelucrare ale datelor folosesc numai registrele μP

…:::5.1 Setul de registre:::…

AVANTAJ – set mare de registre generale:

1. marirea vitezei de procesare prin minimalizarea accesului in memorie a operanzilor si/sau rezultatelor
2. realizarea structurilor de date de tip stiva sau coada "hard" (in interiorul μP)
3. trasferul parametrilor intre programele apelante si apelate se face direct in interiorul μP
4. Deservirea cererilor de intrerupere si multiprocesarea pot fi realizate direct in interiorul μP
5. Marirea "factorului de uniformitate" a cipului

REGISTRELE:

1. procesor RISC – multe registre (cel putin 32……1000)
2. dimensiunea registrelor = dimensiunea operanzilor de lucru (32,64 biti)
3. registrele sunt multifunctionale nu au functii implicite
4. daca am multe registre generale pot defini subdiviziuni logice in setul de registre → setul de registre formeaza registre logice iar eu pot sa definesc registre logice.

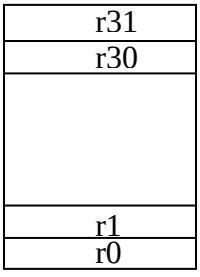
subset de registre logice folosite in program = **set de lucru**

fiecare program va avea afectat un set de lucru.
μP trebuie sa treaca de la registrul logic la legistrul fizic → translatarea registrelor logice in registre fizice
-translatarea se face automat si transparent pentru utilizator.

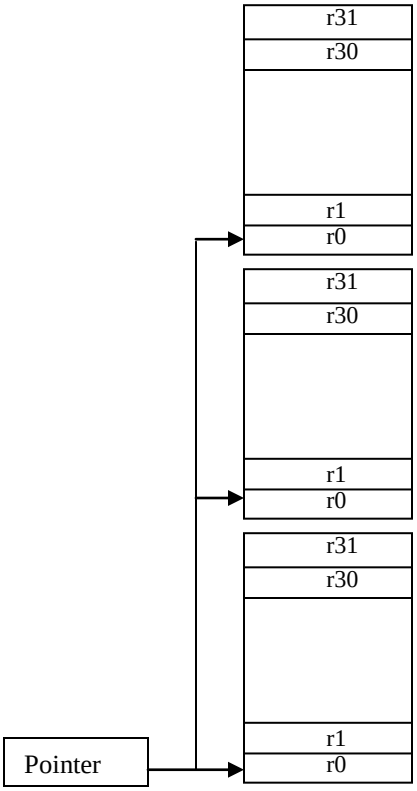
Tipuri organizari:

a) set de registre fizic simplu

-accesul la registre se face prin identificarea numarului sau de ordine r0,r1…r31 la fel cum se face la organizarea liniara a memoriei



b) mai multe seturi de registre logice(unic set de registre fizice)



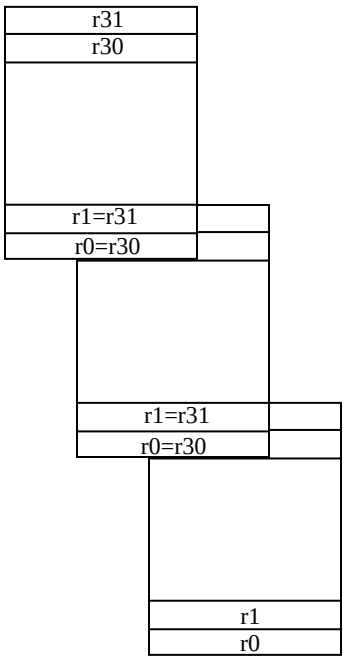
-fiecare subset are cel putin 32 registre ca sa fie set de lucru

-trec de la un subset la altul incrementand pointerul care devine un nou set de lucru
➔furnizeaza modularizarea programelor

-seamana cu paginarea memoriei la CISC
➔impartirea hartii memoriei in entitati de dimensiune fixa si riguros concatenate

-in aceasta situatie exista o corespondenta biunivoca intre registre fizice si logice

c) seturi de registre logice partial suprapuse (unic set de registre fizice)



-suprapunere = **ferestre de registre –** permite comunicarea intre seturile de lucru

-organizarea in care nu mai am corespondenta biunivoca intre registrele fizice si logice

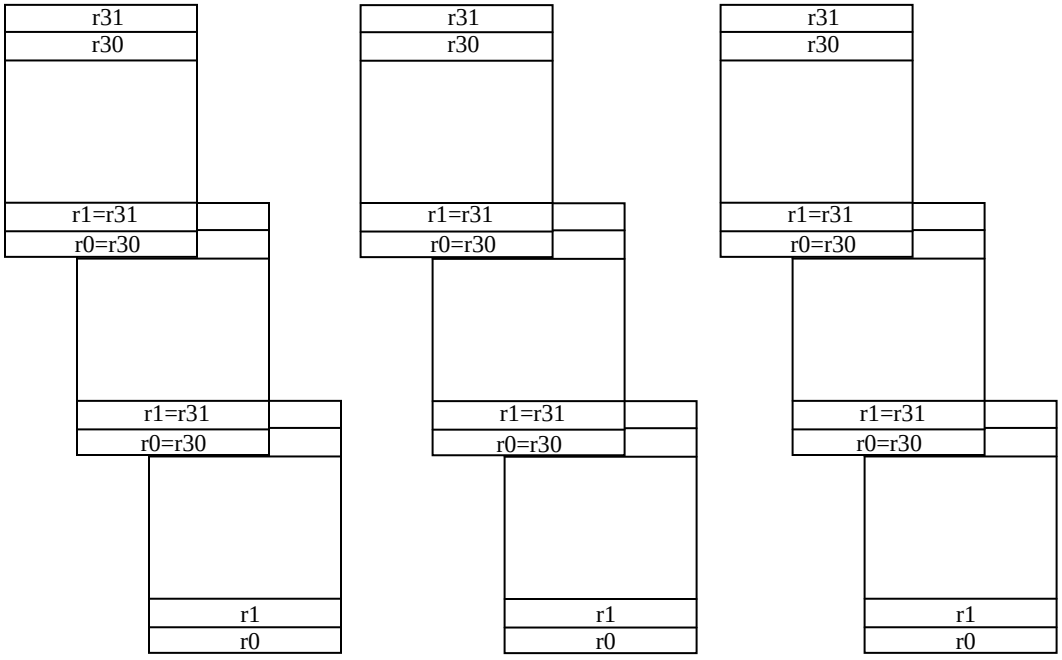
-fiecare set de registre logice=**set de lucru**.

-permite transferal de parametric intre programul apelant si cel apelat

-daca ultimul set de lucru se suprapune cu primul avem **ferestre circulare** = incurajeaja recursivitatea

-exista o analogie cu segmentele partial suprapuse:suprapunere dictate de fabricant .

d) mai multe seturi de registre logice pentru multiprocesare (mai multe seturi de registre fizice)

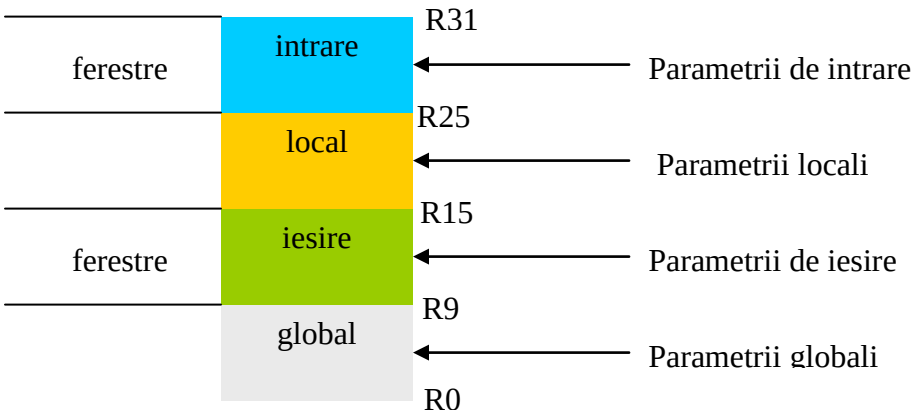


Multiprocesare – set fizic pentru fiecare process

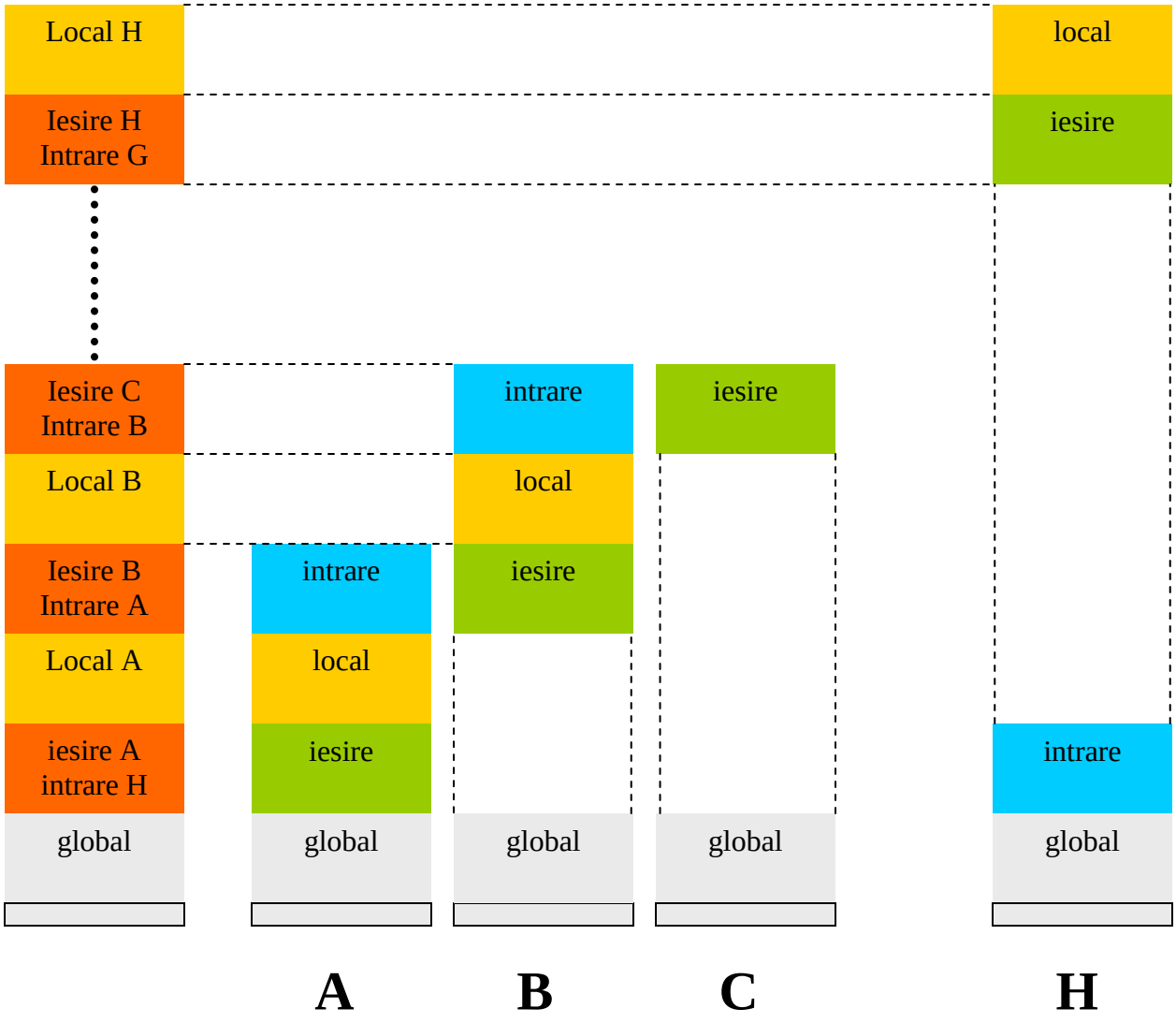
- trecerea de la un process la altul se face cu ajutorul unui pointer.

Registrele µP “Berkley RISC I si II”.

a) registrele disponibile pentru o anumita procedura



b) registrele fizice si registrele logice



8 seturi registre lucru ➔ 8 proceduri

10 registre commune, 10 registre dedicate fiecarui parametru de lucru

138 numar de registre fizice (suprapuse cate 6)

Registrul R0=0 stochez constanta 0

-consum de registre, abordare complet diferita fata de CISC

Translatarea intre registrele logice si cele fizice Berkley RISC:

R0=B0=C0=……..=H0

R10=A10=H26

R15=A15=H31

R31=A31=B15

…::: 5.2 Setul de instructiuni si tehnici de adresare :::…

- **setul de instructiuni mult simplificat (cel mult 128 instructiuni)**
- **instructiunile sunt simple**

instructiuni pentru acces in memorie: LOAD/STORE

LOAD r,mem; (r) ← (mem)
STORE mem,r; (mem) ← (r)

-singurele accesari ale memoriei

r – registrul din setul de lucru

instructiuni aritmetico-logice

(d) ← (s1)⊗(s2)

- fara accumulator
- d,s1,s2 registre din setul de lucru
- registrele de 32 si 64 → operatiile pe 32 si pe 64

- tipul de operatii:
  -operatii logice: SI,SAU,XOR,CF1,CF2

  -operatii aritmetice:+,-, . , : ,inc,dec

  . , : → nu sunt considerate operatii complexe

  -deplasari sau rotatii , nu distrug operandul, un nr variabli de cellule

instructiuni de control

-salturi propriu-zise si apeluri de subprograme si teste

- apelurile de subprograme nu folosesc neaparat stiva,transferal se face cu ajutorul ferestrei, nu am nevoie de numerator de subprogram
- daca exista stiva de multe ori ea este “hard” nu “soft”

Moduri de adresare pentru µP RISC

* in registru:

AF = r_n → logic sau fizic

* directa (absoluta) de regula intr-o portiune a hartii memoriei (poate fi considerate scurta)

AF = adr

* indirecta prin registru

AF = (r_n)

Moduri de adresari complexe:

* relative la baza,directa

AF = (r_n)+adr

* relative la baza cu registru index

AF = (r_i) + (r_j)

* relativa (la PC)

AF = (PC) + disp

Oricare registru poate fi baza si oricare registru poate fi index.

Exemple pentru Intel i860/960

→procesor RISC pe 32 biti

s1,s2,d → registre generale

- o adunare pentru intregi cu semn

adds s1,s2,d; (d) ← (s1) + (s2)

-nu am acumulator

- o referinta in memorie cu adrese in doua registre generale

ldl.l s1(s2),d; (d) ← ((s2)+(s1))

- nu fac operatii cu memoria

- o referinta in memorie folosind o constanta

st.s s1,#const(s2); ((s2)+const) ← (s1)

- o deplasare stanga cu trei operanzi

shl s1,s2,d; (d) ← (s2)* $2^{(s1)}$

-s1 contine numarul de celule cu care se face deplasarea

…::: 5.3 Unitatea de control al microprocesorului :::…

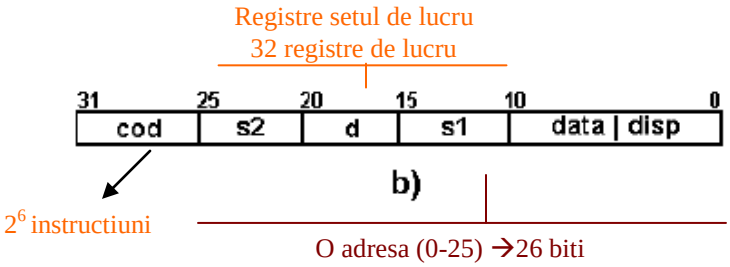
* organizate concepte VON NEUMANN

* aceleasi doua functii → desfasurarea in spatiu
→ desfasurarea in timp

DESFASURARE SPATIU

a) formatul unei instructiuni pentru µP Intel i386 (intre 1-15 octeti)

b) formatul unei instructiuni pentru µP Intel i860 RISC (4 octeti indeferent de instructiune)



procesoarele RISC au formatul instructiunilor identic pentru toate instructiunile
= de regula operanzii de lucru uzuali

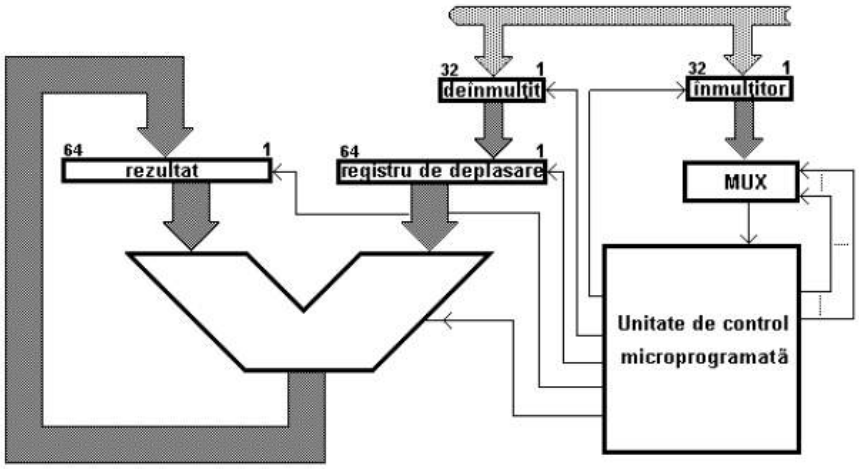
-tipurile de informatie din format sunt putine la numar
-registrele utilizate sunt cele din setul de lucru
-pot sa am un deplasament, o instanta, o adresa.

Din punct de vedere al desfasurarii in apatiu Unitatea Centrala a unui µP RISC este simplificata fata de cea a unui µP CISC.

DESFASURARE TIMP

a) pentru un µP CISC (inmultire pe 32biti)

se folosesc 2 acumulatori DX,AX (la CISC)




```
rezultat ← 0
for i = 1 to 32 do
     if inmultitor(i) = 1
          rezultat  ← rezultat + deinmultit
     end_if
     deinmultit ← deinmultit * 2
end_for
```

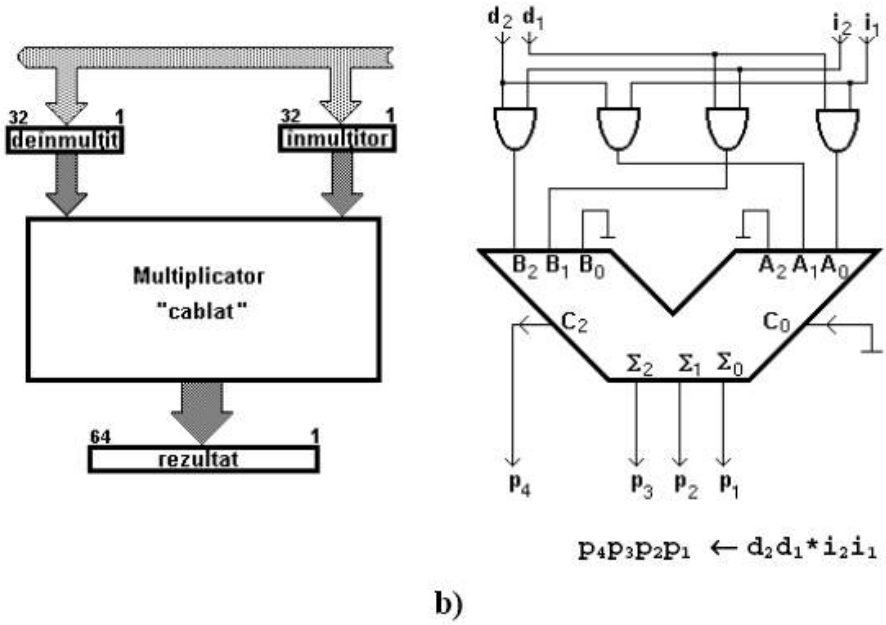
CONCLUZII:

1. pentru diverse operatii de prelucrari de date se folosesc aceleasi componente ale schemelor Block prezentate deja pentru un µP CISC
2. operatiile aritmetice complexe dureaza un numar mare de stari, variabil, depinzand de operanzi
3. succesiunea de stari este stocata intr-o memorie de µprogram in care fiecare instructiune e µprogramata

acumulator + registru de deplasare

→ realizare prin concatenare de 2 registri de 32 biti

b) pentru un µP RISC



b)

CONCLUZII:

1. operatiile de prelucrare de date se realizeaza cu circuite dedicate, combinationale de regula
2. operatiile se realizeaza cablat, exista premise ca aceste operatii de prelucrari de date, chiar cele complexes a se realizeze intr-o stare
3. Unitatea Control simplificata si din acest punct de vedere, nu e nevoie de un automat complicat care sa genereze succesiunile de stari

UC = nu este µprogramata este cablata.

TEHNICA PIPELINE

- fie un µP RISC care are orice instructiune cu acelasi numar de stari. Oricare instructiune dureaza 5 stari.

s1 -se identifica si se adduce din memorie codul instructiunii
s2 -se decodifica codul instructiunii
s3 -citesc operanzii
s4 -execut operatia
s5 -scriu rezultatul

(pasii s3,s4,s5 pot fi LOAD,STORE sau un salt)

1. procesoarele RISC sunt caracterizate de durata uniforma, egala practice pentru toate instructiunile intr-un numar mic de stari (deosebiri de CISC)

2. aceasta desfasurare uniforma in timp este posibila datorita catorva premise:
   a. realizare cablata operatii, prelucrari de date
   b. folosesc registre interne pentru operatii
   c. instructiunile au format identic si putine tipuri de informatii (aducerea si codificarea dureaza putin)
   d. putine instructiuni (cod scurt)
   e. putine moduri de adresare – dureaza putin
   f. pot avea coada de instructiuni
3. daca instructiunile se desfasoara uniform, putine stari , ele pot fi suprapuse asa fel incat in fiecare stare pot avea mai multe instructiuni aflate in diverse etape ale desfasurarii ➔ tehnica benzii rulante (**pipeline**) si poate duce la obtinerea unui rezultat in fiecare stare
4. daca toate instructiunile dureaza n stari ele pot fi suprapuse de amniera incat in fiecare stare sa am n instructiuni in diverse etape de desfasurare. Sa zicem ca executia de tip pipeline are n etaje.(n=5 PENTIUM)
5. daca suprapunereaeste perfecta ca in exemplu de mai sus obtinerea unui rezultat pentru fiecare stare este semnificativ, pentru fiecare stare CPI

CPI- clock per instruction

- in mod evident CISC are CPI>1 si variabil
- pentru procesoare RISC CPI=1 dar exista si exceptii
- exista procesoare cu CPI<1 ➔ procesoare superscalare (PENTIUM)

CONCLUZII:

- UC µP RISC este mult mai simplificata decat cea de CISC , este cablata si nu semiprogramata
- desfasurarea in timp si in spatiu a instructiunilor este uniforma
- exista multe caracteristici de arhitectura si stuctura care permit desfasurarea uniforma in timp
- desfasurarea uniforma in timp, prin aparitia tehnicii benzii rulante CPI=1

…::: 5.4 Caracteristici RISC la nivel software :::…

- impune restrictii desfasurarii programelor

- cel care proiecteaza procesorul poate sa optimizeze secventa de instructiuni cu care sunt emulate instructiuni de nivel inalt

- optimizari ulterioare ale codurilor sunt imposibile

utilizarea registrelor → registrele logice trebuiesc translatate registre fizice

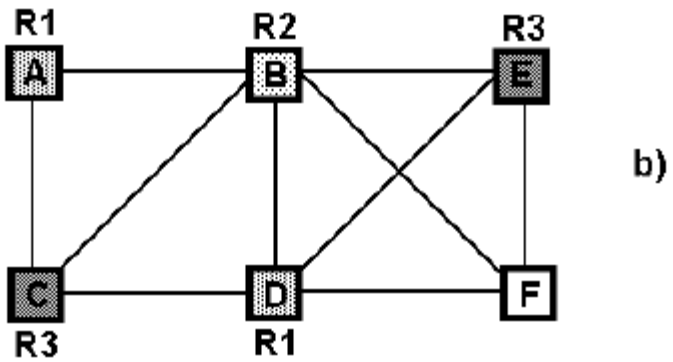
optimizarea RISC:

µP RISC set de lucru 6 regisre logice A,B,C,D,E,F

- presupunem secventa de instructiuni, registrele logice organizate intr-un anume fel . orice moment translatarea registrelor logice in trei registre fizice R1,R2,R3

a) diagrama timpilor de utilizare a regsitrelor logice A,B,C,D,E,F

b) graful de interferenta a utilizarii registrelor; alocarea registrelor fizice **R1**, **R2**, **R3**.



avand in vedere premisele exemplului propunem folosirea unui graf care sa permita translatarea registrelor logice in registre fizice. Nodurile grafului sunt registre logice. Laturile grafului semnifica cel putin utilizarea simultana a celor doua registre logice.

Alocarea registrelor logice se face evitand sa am acelasi registru fizic in doua noduri unite de o latura.

-daca alocarea nu e posibila pentru toate nodurile inseamna ca μP se blocheaza

Acest caz poate fi evitat in mai multe feluri:

a) mai multe registre fizice din constructie, nu optimizeaza utilizarea registre fizice
b) varianta software → voi avea grija ca secventele de instructiuni sa permita intotdeauna translatarea pentru un minim de registre fizice disponibile

→ aceste optimizari se fac in faza de proiectare a procesorului. In mod normal μP odata proiectat e insotit de realizarea compilatoarelor de nivel inalt cu secventele de instructiuni optimizate conform prevederilor, constrangerilor prezentate anterior.

Orice optimizare ulterioara este practica imposibila → optimizarea translatiei registrelor logice.

Avantajele RISC

1. Realizarea fizica în structuri VLSI:

- minimizarea ariei cipului dedicata Unitatii de control al microprocesorului (sub 10%);
- marirea ariei cipului disponibila pentru registre generale;
- marirea "factorului de uniformitate" a cipului (numar total de circuite / numar de tipuri de circuite: registre, UAL, numaratoare etc.);
- posibilitatea utilizarii altor tehnologii decât cea a siliciului (ex. GaAs).

2. Marirea vitezei de procesare:

- prin realizarea Unitatii de control cu logica de tip cablat;
- prin utilizarea unui numar mare de registre interne se reduce traficul cu memoria;
- prin suprapunerea executiei instructiunilor;
- prin utilizarea tehnicii "întârzierii salturilor" se previne golirea cozii de instructiuni.

3. Scaderea costului si marirea fiabilitatii:

- timp mai mic pentru proiectarea Unitatii de control;
- timpul global de proiectare si punere în fabricatie este considerabil mai mic decât pentru CISC;
- probabilitate mai mica de a avea erori de proiectare si usurinta de corectare;
- lungimea standard a formatului instructiunilor elimina riscul depasirii limitelor paginilor de catre o instructiune. Gestionarea paginilor devine mai usoara.

4. Suport pentru limbajele de nivel înalt:

- realizarea compilatoarelor este mai simpla (numar mic de optiuni în alegerea instructiunilor);
- cresterea eficientei prin utilizarea extensiva a operatiilor de prelucrare în interiorul microprocesorului;
- tehnica "ferestrelor de registre" usureaza implementarea apelarii subrutinelor (procedurilor).

Dezavantajele RISC

1. Numarul redus de instructiuni;
rezulta ca programele RISC sunt mai lungi decâtcele CISC (în medie cu 30%).

2. Numarul mare de registre interne:

- timp de acces mai mare;
- utilizarea registrelor “pointer” pentru ferestre complica selectia unui registru la decodare;
- spatiu mare pe cip;
- tehnici complicate de gestionare a ferestrelor;
- compilatoarele avansate folosesc mai eficient seturi reduse de registre;
- salvarea registrelor în contextul multiprocesarii (la trecerea de la un proces la altul) presupune timp mai îndelungat pentru stocare si recuperare în/din memorie.

3. Unitatea de control a microprocesorului realizata “cablat” este mai putin flexibila si mult mai greu de modificat.

4. Formatul redus al instructiunilor face imposibila adresarea directa a unei harti dememorie mare (de pilda, adrese fizice sau logice de 32 biti).

VI. Strategii de intrare/iesire

…:::6.1 Spatiul dispozitivelor de intrare/iesire :::…

2 modaliati de organizare a porturilor:

a) Ca porturi propriu-zise

- cicluri masina specifice
- semnale distincte pe magistrala de control: **IOR** si **IOW** (input output read si input output write)

IN d, port
OUT port, d

* instructiuni dedicate, speciale de transfer la si de la porturi
* transferul de date unde un membru este portul (sursa port pentru IN, destinatia port pentru OUT)
* d,s atribute de arhitectura
* **d sau s** → acumulatorul implicit sau dedicat
* harta porturilor este mult mai mica decât harta memoriei
* moduri de adresare foarte restrictive pentru porturi:
  → directa
  → indirecta prin registru

avantaj
* Acces rapid la porturi

dezavantaj
* Consuma coduri de instructiuni
* Consuma terminale de pe magistrala de control

-Intel x86 compatibil de astfel de organizari ale dispozitivelor intrare/iesire
-Intel x86 – instructiuni dedicate cu cicluri masina speciale prin care se valideaza
* acumulator util implicit AL,AX,EAX
* maximum harta porturilor 64kiloporturi/octet
  32 pentru 2 octeti
  16 pentru 4 octeti
* adresari
  -adresare directa → 8 biti
  -adresare indirecta prin DX obligatoriu (fara EDX)

b) Ca locatii de memorie

- cicluri masina de acces în memorie
- semnale pe magistrala de control folosite pentru accesul în memorie:**MEMR** si **MEMW** (memory read, memory write)

* mod de organizare tipic CISC – in general tot felul de transferuri de CISC

avantaje

* Extind toate caracteristicile transferurilor de date asupra instructiunilor I/O
* Extind utilizarea porturilor si în cadrul instructiunilor de prelucrari de date

→ Intel Pentium pot sa am portul pe post de acumulator

dezavantaje

* Consuma spatiu din harta memoriei
* Timp mare de acces (formatul instructiunilor complicat, modurile de Adresare sunt multe)

-oricare operatii in se foloseste memoria se aplica si la porturi
-oricare processor care poate folosi aceasta caracteristica b) poate folosi si caracteritica a)
-daca procesorul nu are a) atunci are b)

…:::6.2 Tehnici de intrare/iesire uzuale :::…

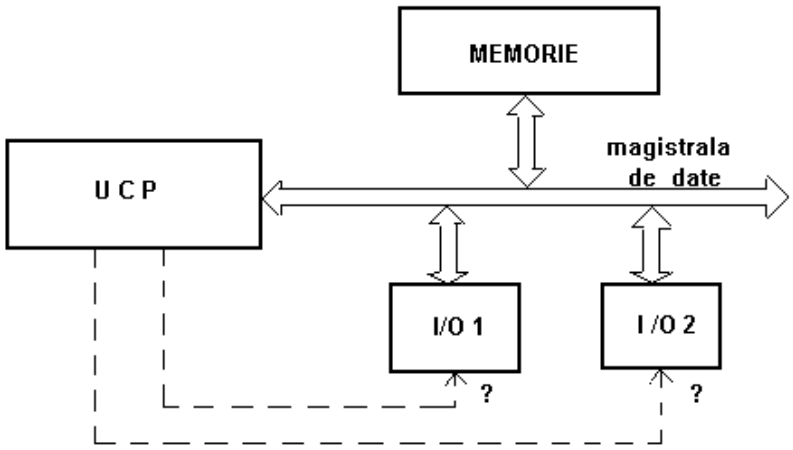
-sunt tehnici de sincronizare a microcalculatorului cu echipamentele periferice conectate prin intermediul porturilor

Tehnici:

A. Sincrone cu programul curent

B. Asincrone cu programul în curs de desfasurare

A. Interogarea continua (tehnica “polling”)



- tehnica interogarii presupune ca fiecare port poate furniza un cuvant de stare(unul sau mai multi octeti prin care comunica daca este sau nu disponibil, daca transferal de date de la sau spre un periferic etc.)

- interogarea consta in citirea de catre µP a cuvantului de stare de la fiecare port. Interogarea este continua in sensul ca procesorul citeste periodic un astfel de cuvant de stare pana cand portul este disponibil pentru transfer, de la µP sau catre Mp

EXEMPLU
Intel x86 compatibil. Doua porturi accesate **port1** si **port2**, fiecare poate furniza un octet de stare in care MSB indica disponibilitatea portului (1 → port disponibil).

μP interogheaza porturile sic and gaseste unul disponibil ii trimite 16 biti.

```
Start1:  IN   AL, PORT1 →citesc cuvant stare
         SHL  AL, 1 →msb trece in fanionul de transport
         JNC  Start1 →testez fanionul de transport
         OUT  PORT1, AX →trec mai departe AX=16biti
Start2:  IN   AL, PORT2
         SHL  Al, 1
         JNC  Start2
         OUT  PORT2, AX
```

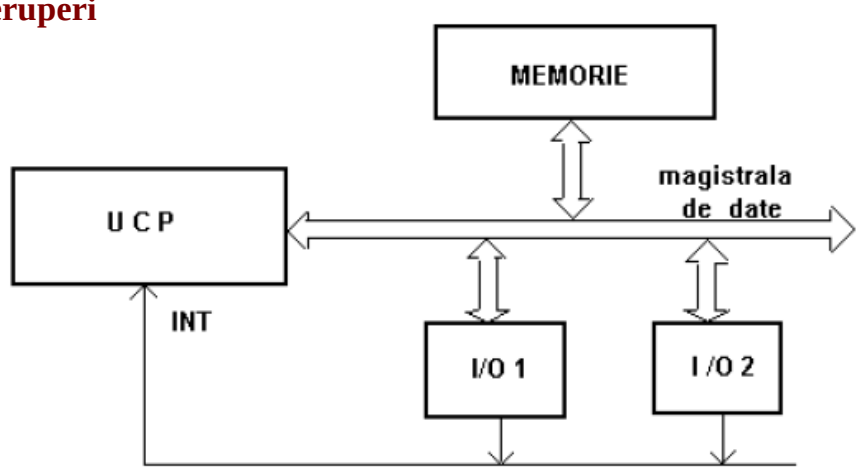
-interogarea se face cu intructiunea din program

JNC → jump if not carry
Daca carry=1 trec mai departe

avantaje:
* nu implica hardware suplimentar
* comunicarea cu perifericele se face sincron cu programul curent

dezavantaje:
* interogarea consuma timp
* se pierd posibile cereri de comunicare cu perifericele

B. Întreruperi



- presupune o legatura fizica speciala intre porturi si µP, porturile pot trimite semnal ca sunt disponibile, ca doresc transfer de la un terminal specializat al µP iar µP raspunde intrerupandu-si programul intr-o maniera specifica cererii unui port si revenind apoi la programul cuerent

- avantajele de la punctul A sunt dezavantaje si dezavantajele de la A sunt avantaje

Cerere de întrerupere: semnal trimis unui terminal dedicat al microprocesorului prin care un periferic (prin intermediul unui port) cere acces la resursele sistemului.

Raspuns la o cerere de întrerupere: o secventa de actiuni pe care microprocesorul o declanseaza parasind programul normal de functionare

Rutina de deservire a unei întreruperi: un program prestabilit, aflat la o adresa prestabilita, prin care microprocesorul raspunde la o anumita cerere de întrerupere formulata de un anumit periferic

EXEMPLU:

Fie un processor CISC, cu organizare liniara a memoriei, formatul memoriei octet si adresele fizice de 2 octeti, procesorul arenumaratorul e program de 16 biti numit PC, indicator de stiva pe 16 biti numit SP, are registrele generale r16, presupunem un registru de fanioane F pe 16 biti.

- fanionul IF valideaza anume tipuri de cereri de intrerupere

1. **(SP) ← (SP) - 2**
((SP) + 1) ↑ ((SP)) ← (F)

2. **(SP) ← (SP) - 2**
((SP) + 1) ↑ ((SP)) ← (PC)

3. **for i = 1 to n do**
(SP) ← (SP) - 2
((SP) + 1) ↑ ((SP)) ← (r16i),
În care **n** este numarul total de registre implicate.
4. **IF ← 0**

5. **(PC) ← adr**

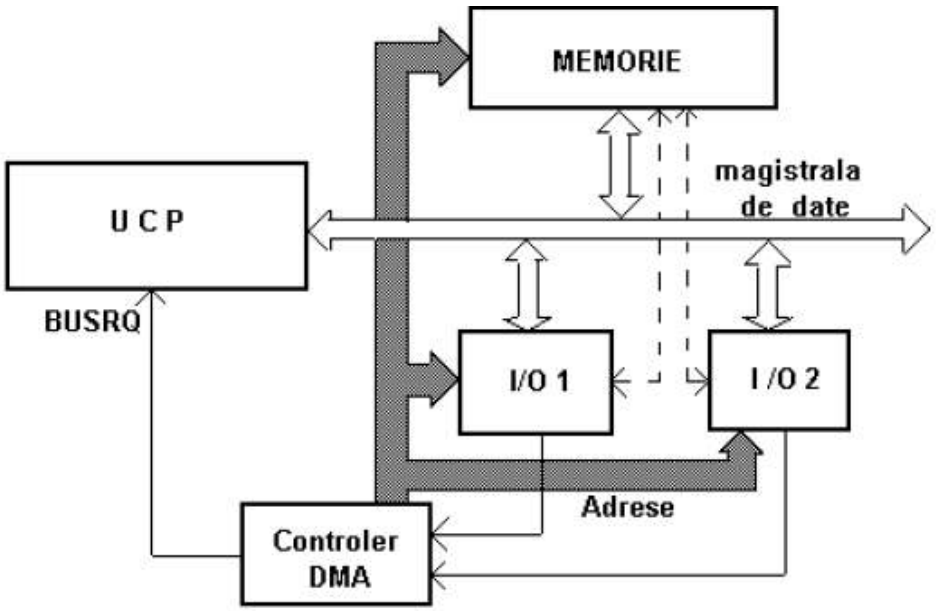
6. **la revenirea în programul curent se recupereaza din stiva toate informatiile**

OBSERVATII:

1. raspunsul la o cerere de intrerupere pentru μP CISC foloseste stiva primara sau propriu-zisa
2. procesorul salveaza in stiva F, PC deci locul unde se face intreruperea si optional alte registre attribute de arhitectura
3. resetarea lui IF inseamna invalidarea anumitor cereri de intreruperi care ar putea aparea in rezolvare, in cursul raspunsului intreruperii curente.
4. O etapa in cadrul secventei de raspuns este apelarea etapei nr.5.(diverse tipuri de intrerupere si dupa modul in acre adr este prestabilita)
5. reiau din stiva ceea ce era acolo pentru a putea continua programul initial

…:::6.3 Intreruperi caracteristice µP de uz general :::…

a) Accesul direct la memorie:



BUSRQ → bus request (controllerul preia controlul asupra magistralei)

Caracteristici:

1. este o intrerupere de prioritate maxima (se raspunde prima)
2. este un criteriu de performanta , tipul de raspuns nu asteapta ciclul masinii current)
3. raspunsul la intrerupere este hard nu soft (µP isi ingheata activitatea)
4. controllerul DMA care preia controlul magistralei si care are drept unica sarcina transferal de date direct intre memorie si porturi poate fi integrat pe chipul µP sau extern
5. transferal de date direct intre memorie si porturi considerat a fi transferal cel mai rapid mod de transfer de date pe magistrala µP a calculatorului

controlerul DMA poate avea setata o proprietate a unuor porturilor asupra altora
tehnica PUR hard DMA → cel mai rapid DMA

b) intreruperi nemascabile

* este formulata pe un terminal specializat (“NMI”)
* nu poate fi invalidata de catre utilizator(=nemascabila)
* ca prioritate, urmeaza dupa cererea de acces direct la memorie(DMA)
* asteapta terminarea instructiunii curente
* urmeaza algoritmul general de raspuns la o cerere de întrerupere
* rutina de deservire a întreruperii are o adresa prestabilita

c) Întreruperi mascabile:

* este formulata pe un terminal specializat (“INT”)
* poate fi invalidata de catre utilizator (fanion / fanioane de validare a întreruperilor)
* ca prioritate, urmeaza dupa întreruperile nemascabile
* asteapta terminarea instructiunii curente
* urmeaza algoritmul general de raspuns la o cerere de întrerupere
* rutina de deservire a întreruperii are o adresa care depinde de “modul de raspuns” prestabilit:
  - → “modul 0” – perifericul stabileste adresa dar si instructiunea de apel a rutinei de deservire a întreruperii
  - → “modul 1” – adresa rutinei de deservire a întreruperii este prestabilita
  - → “modul 2” – ***întreruperi vectorizate***(compromise modul 0 si 1)

Întreruperi vectorizate

Un periferic trimite, indirect, un vector de întrerupere dintr-o multime de vectori posibili, alegând astfel o rutina de deservire din mai multe rutine potentiale

Se utilizeaza adresarea indirecta cu memoria folosind o **tabela cu vectori de intrerupere.**

-mecanismul este de a primi o informatie de la periferic si de la acea informatie µP calculeaza pozitia in care se afla vectorul de intrerupere, citeste vectorul de intrerupere din acea locatie

- exista corespondenta biunivoca intre informatia trimisa de periferic si adresele vectorilor de intrerupere posibili

- exista corespondenta biunivoca intre informatia primita de la periferic si procedura de raspuns

-din aceste corespondente biunivoce ➔ perifericul se autoidentifica in acest fel si ca in aceasta maniera µP allege procedura dedicate acestui periferic

-informatia primita de la periferic trebuie sa fie un compromise intre compromisul perifericului de a trimite un numar de biti si numarului de periferice se poate raspunde cu proceduri distincte

µP → impune dimensiunea codului

➔aceasta imformatie primita de la periferic se numeste **tip**

-intreruperile vectorizate folosesc adresarea indirecta cu memoria

1) µP primeste prin intermediul unui port un cod pe care il numim tip
2) µP calculeaza adresa unui vector de intrerupere folosind aceasta informatie numita tip
3) adresa astfel calculata este o adresa intr-o tabela cu vectori de intrerupere
4) µP citeste din tabela vectorul de intrerupere corespunzator pe care il foloseste ca sa apeleze procedura de raspuns

Obs. Conform specificului adresarii cu memoria perifericul nu impune direct vectorul ci adresa ➔pozitia vectorilor este prestabilita in memorie darn u si valorile efective.

Probleme:

1. Marimea vectorului de întrerupere
2. Dimensiunea informatiei furnizata de periferic (“**tip**”)
3. Marimea tabelei cu vectori de întrerupere
4. Localizarea tabelei cu vectori de întrerupere în harta memoriei

1.vectorul de intreruperi este o adresa completa
dimensiunea vectorului de intreruperi = dimensiunea adresei complete

2.dimensiunea tip impusa µP se realizeaza un compromis intre complexitatea informatiei primita de la periferic si numarul de periferice deservibile

3.dimeniunea tabelei → produs intre dimeniunea vectorilor si nr. de vectori

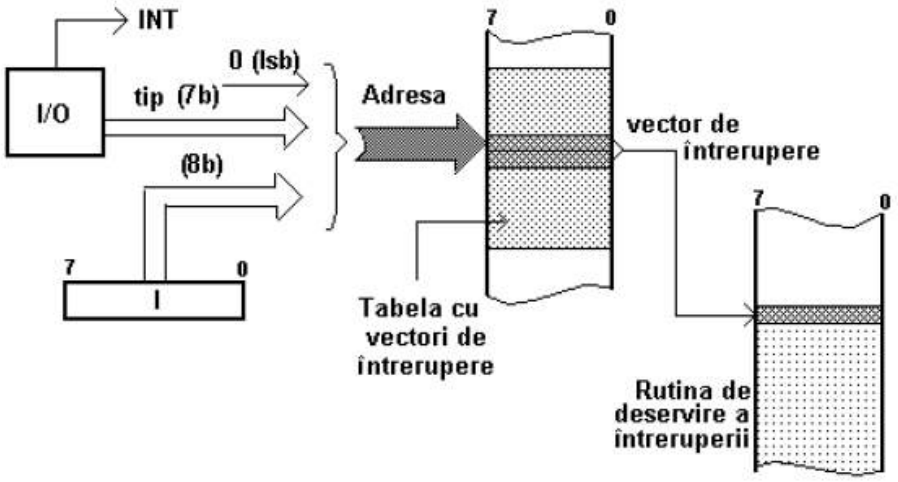
4. doua categorii de Mp din punct de vedere al pozitiei tabelei cu vectori de intrerupere

1. µP care spun unde se afla tabela
2. µP poate sa foloseasca anumite zone

Regula…. Dupa codul de "tip" are n biti si vectorul de intrerupere are m octeti dimensiunea:

$m*2^n$ octeti

EXEMPLU:
-fie µP CISC
- organizare liniara a memoriei, memorie organizat pe octeti, adrese complete 16 biti, cod tip de 7 biti, registru suplimentar atribut de arhitectura, poate sa aleaga o anumita zona de memorie.

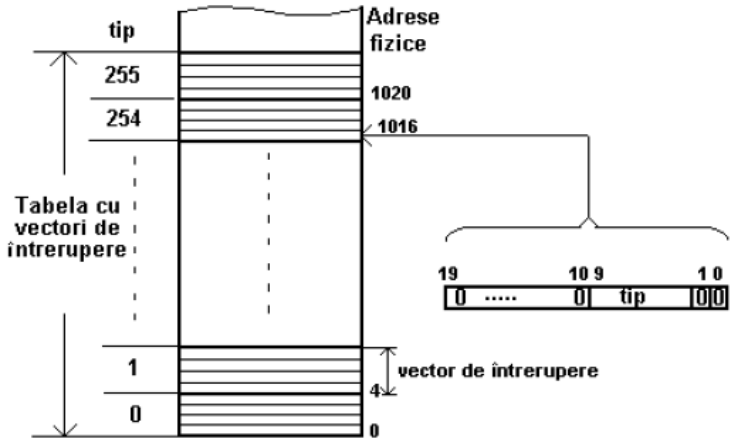


vector_de_întrerupere = ((I) ↑ tip ↑ 0) ↑ ((I) ↑ tip ↑ 0 + 1)

procesorul ma obliga sa introduce in tabela din 2 in 2 de aceea *2 adica pun 0 la coada

1. vectorul de intrerupere are 2 octeti
2. tip are 7 biti
3. tabela cu vectori de intrerupere 256B (2^8 octeti)
4. localizarea tabelei este la alegerea utilizatorului

…:::6.4 Tipuri de întreruperi pentru µP Intel în modul real:::…



1. Vectorul de întrerupere are 4 octeti
2. **tip** are 8 biti
3. Tabela cu vectori de întrerupere 1kB
4. Localizarea tabelei este prestabilita

µP Intel x86 folosesc tehnica intreruperilor pentru sincronizarea µP

-urmatoarele tipuri de intreruperi la care Intel x86 raspunde:

a) cerere de access direct la memorie (BUSRQ)
-mod de raspuns este de 2 stari
-controler access direct la memorie externa µP
-timp de raspuns de 2 stari

b) cereri de intreruperi nemascabile, formulate pe terminal specializate NMI
-Intel imi impune nu direct valoarea adresei procedurii de raspuns ci adresa acestei adrese → undeva in cadrul tabelei vectorilor de intrerupere

c) Intel x86 in mod real , cereri de intrerupere mascate vectorizabile, cererile de intrerupere sunt formulate pe terminal specializat numit INT

Intelx86 → un singur fanion validare cereri de intreruperi (IF)

Caracteristica interesanta Intel x86 → intreruperi software
→instructiuni din setul de instructiuni la care raspunsul µP este identic la o cerere de intrerupere mascabila vectorizata

Intreruperi → software
→ hardware

- intreruperile software folosesc aceeasi tabela cu vectori de intreruperi ca intreruperile hardware, iar desfasurarea in timp a instructiune este de fapt secventa de instructiuni care reprezinta raspunsul la o cerere de intreruperi.

din punct de vedere al μP mecanismul este acelasi deoarece octetul numit tip este primit similar pe magistrala de date

a) cazul intreruperilor
b) exact la fel in memoria de program

(tip-octetul instructiunii curente)

```
INT [tip] ;   (SP) ← (SP) - 2                          salvez fanioane stiva
              ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP)) ← (F)
              (IF) ← 0     → fanion general valid intreruperi
              (TF) ← 0  → invalidez o cerere de functionare pas cu pas
              (SP) ← (SP) - 2
              ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP)) ← (CS)
              if tip then
                    (CS) ← (4 * tip + 3) ↑ (4 * tip + 2)   →octetul exista
              else
                    (CS) ← (0000FH) ↑ (0000EH)
              (SP) ← (SP) - 2                          →octetul nu exista
              ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP)) ← (IP)
              if tip then
                    (IP) ← (4 * tip + 1) ↑ (4 * tip)
              else
                    (IP) ← (0000DH) ↑ (0000CH)
```

↙dedicata situatiei cand tin seama de depasirea unei operatii aritmetice

```
INTO ;    if (OF) = 1 then
                (SP) ← (SP) - 2
                ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP)) ← (F)
                (IF) ← 0
                (TF) ← 0
                (SP) ← (SP) - 2
                ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP)) ← (CS)
                (CS) ← (00013H) ↑ (00012H)
                (SP) ← (SP) - 2
                ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP)) ← (IP)
                (IP) ← (00011H) ↑ (00010H).
```

intoarcere din procedura de raspuns

IRET ; **(IP) ← ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP))**
(SP) ← (SP) + 2
(CS) ← ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP))
(SP) ← (SP) + 2
(F) ← ((SS) ↑ 0H + (SP) +1) ↑ ((SS) ↑ 0H + (SP))
(SP) ← (SP) + 2.

Tipul întreruperii	Adresa vectorului de întrerupere	Funcția implicită
0	**00H ÷ 03H**	Cerere de întrerupere generată de "împărțire la 0"
1	**04H ÷ 07H**	Cerere de întrerupere pentru funcționare pas cu pas
2	**08H ÷ 0BH**	Asociat *obligatoriu* întreruperii hard nemascabile
3	**0CH ÷ 0FH**	Asociat instrucțiunii **INT** cu format minim (un octet)
4	**10H ÷ 13H**	Asociat instrucțiunii **INTO**

Rezervate

VII. DIMENSIUNEA TEMPORALA A ARHITECTURII µPUG

…:::7.1 DESFASURAREA IN TIMP A INSTRUCTIUNII PT µP RISC:::…

Desfasurarea in timp a instructiunilor CISC este neuniforma.
Ea depinde de complexitatea instructiunilor

Fiecare instructiune are:
mai multe cicluri masina (M1, M2, M3,....)

Fiecare ciclu masina are:
mai multe stari (T1, T2, T3, ...)

Câteva tipuri de cicluri masina

„fetch" (M1) – identifica codul instructiunii curente si il aduce in µP, apoi il decodifica
prelucare date
citeste din memorie
scrie în memorie
citeste din stiva
scrie în stiva
citeste din porturi
scrie în porturi

Masina are schemele bloc functionale ca in cap 2

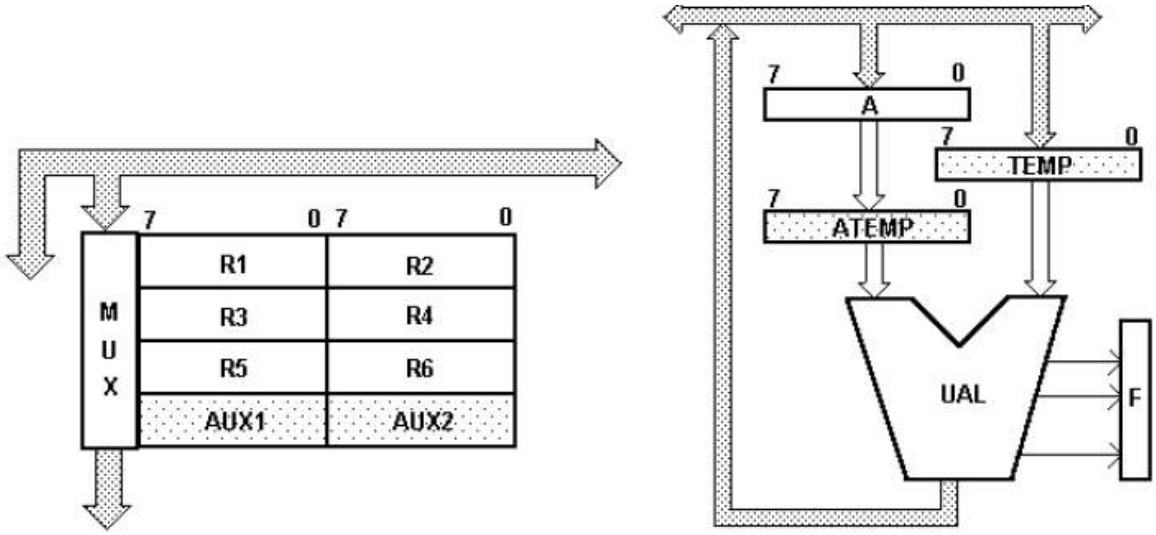
Exemplu

- magistala de date interna si externa pe 8 biti
- organizare liniara a memoriei
- adrese fizice pe 16 biti
- memoria oragnizata pe octeti
- registre generale pe 8 biti concatenabile cîte doua: R1, R2,R3, R4, R5, R6
- Acumulator pe 8 biti A

- Registru de fanioane F pe 8 biti
- Numarator de program PC
- Indicator de stiva SP

Registru index IX
Registru de instructiuni RI pe 8 biti
si Registru de date RD → NU SUNT ATRIBUTE DE ARHITECTURA
Registru de adrese RA
Registre temporare ATEMP, TEMP, AUX1, AUX2

AUX1 si AUX2 – registre cache
ATEMP si TEMP – sincronizarea activitatii UAL



- **regula micului indian**

Instructiunea: (R1)←(R3)

T1 - adresez codul instructiunii curente, sistemul stie ca se afla intr-o stare speciala

T2 – aducerea cosului instructiunii curente

T3 – codul este adus in R1

(R1) ← (R3)

```
M1:  T1:  (RA) ← (PC)
          (RD) ← "Stare µP"
          MREAD
     T2:  (PC) ← (PC) + 1 ,   (RD) ← ((RA))
     T3:  (RI) ← (RD)
     T4:  decodificarea octetului de cod din (RI)
          (TEMP) ← (R3)
     T5:  (R1) ← (TEMP).
```

Concluzii:

1. Orice desfasurare in timp se bazeaza pe o schema bloc functionala data.
2. Desf in timp are in vedere optimizarea timpului de executie pentru o schema bloc functionala data
3. Din acest motiv sunt stari in care am mai mult de o actiune elementara.

   Actiuni puse una sub alta =succesive(T1)
   Actiuni puse pe acelasi rand=simultane(T2)

4. Se pot desfasura simultan mai multe actiuni elementare daca numai una dintre ele afecteaza magistrala interna de date (vezi ex T2)
5. Nu se transfera direct continutul lui R3 in R1 pentru ca organizarea fizica a setului de registre este de tip RAM fizic –nu se poate scrie si citi simultan.
   - Pe post de registru tampon, intermediar

Registru tampon - nu trebuie sa fie atribut de arhitectura
- trebuie sa fie accesibil

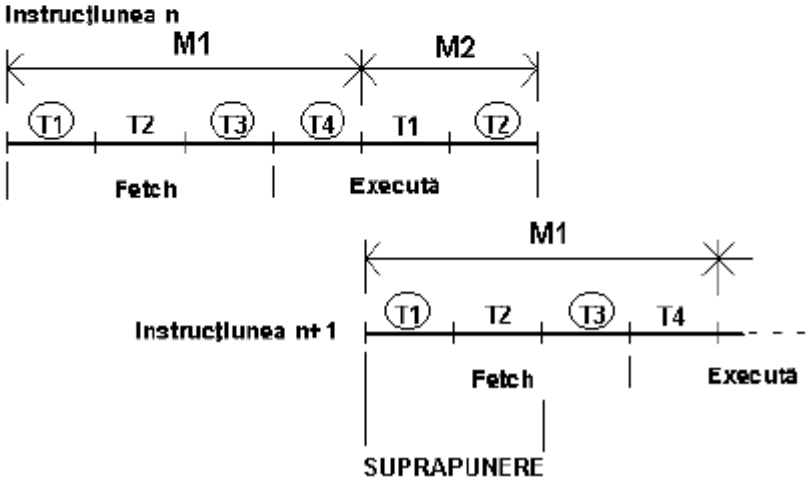
6. Aceste instructiuni se desfasoara intr-un singur ciclu masina modificat

Instructiunea (A)←(A)+(R1)

(A) ← (A) + (R1)

M1: T1: (RA) ← (PC)
(RD) ← "Stare µP"
MREAD
T2: (PC) ← (PC) + 1 , (RD) ← ((RA))
T3: (RI) ← (RD)
T4: decodificarea octetului de cod din (RI)
(TEMP) ← (R1) , (ATEMP) ← (A)

M2: T1: nefolosită!
T2: (A) ← (ATEMP) + (TEMP)



Observatie:

1. T1, T2, T3 sunt identice pentru M1 – se repeta pentru fiecare instructiune
2. Pentru *criteriul de viteza de executie* a se observa T4 din M1 : -2 actiuni simultane
3. Tot pentru *criteriul de viteza de executie*, in mod paradoxal se declanseaza un nou ciclu masina care lasa starea T1 nefolosita

-de fapt are 4 stari – executia pipeline

-cele 2 instructiuni pot fi suprapuse partial
- orice stare in cerc acces seaza magistrala interna de date, nu exista 20 pe ac. Verticala
- executia lui **n+1** se realizeaza dupa 4 stari

4. Desfasurarea propusa pentru instructiunea curenta n permite o suprapunere partiala cu instructiunea urmatoare pe principiul benzii rulante.

Se castiga o stare in executia instructiunii.

4. Suprapunerea nu este uniforma pentru ca instructiunea nu se desfasoara uniform

=>SUPRAPUNERE PARTIALA SI NEUNIFORMA

5. Atentie la starile care afecteaza magistrala interna de date.

Instructiunea (A)←(A)+((R5,R6))

- ne scoate afara din procesor

(A) ← (A) + ((R5,R6))

M1:	T1:	(RA) ← (PC)
		(RD) ← "Stare µP"
		MREAD
	T2:	(PC) ← (PC) + 1, (RD) ← ((RA))
	T3:	(RI) ← (RD)
	T4:	decodificarea octetului de cod din (RI)
		(ATEMP) ← (A)
M2:	T1:	(RA) ← (R5,R6)
		MREAD
	T2:	(RD) ← ((RA))
	T3:	(TEMP) ← (RD)
M3:	T1:	nefolosită
	T2:	(A) ← (ATEMP) + (TEMP)

Adresare indirecta prin registru
-un octet adus din memoria de date

Observatii:

1. Avem 3 cicluri masina
   - Fetch
   - M2 – citire din memorie un operand pe 8b
   - M3 – prelucrare

"stare µP" este obligatoriu numai la inceputul instructiunii

2. A se remarca T4 din M1 pentru criteriul de eficienta in viteza
3. Si aceasta instructiune poate fi suprapusa cu urmatoarele
   Si aici suprapunerea este partiala si neuniforma
   Castigam o stare din 8 efective.
4. Atentie la actiunile elementare care ar putea sa nu incapa intr-o stare
   T1 din M2, depinde de structura concreta

(R5, R6) – 16 biti, transferul se face pe magistrala de 8b

O singura data accesez memoria de program => formatul instructiunii este de 1 octet;
-ori de cate ori in RA se varsa continutul lui PC

Instructiunea (A)←(adr)

- adresare directa/absoluta a sursei si implicita a destinatiei
- M1 si M2 - citesc din memoria de program cei 2 octeti care constituie memoria absoluta
- M1, M2, M3 – accesez mem de program
  - un octet de cod
  - 2 octeti de adresa
- M4 - asamblez octetii sa formeze o adresa, accesez memoria de date

(A) ← (adr)

M1:	T1:	(RA) ← (PC)
		(RD) ← "Stare µP"
		MREAD
	T2:	(PC) ← (PC) + 1, (RD) ← ((RA))
	T3:	(RI) ← (RD)
	T4:	decodificarea octetului de cod din (RI)
M2:	T1:	(RA) ← (PC)
		MREAD
	T2:	(PC) ← (PC) + 1, (RD) ← ((RA))
	T3:	(AUX2) ← (RD)
M3:	T1:	(RA) ← (PC)
		MREAD
	T2:	(PC) ← (PC) + 1, (RD) ← ((RA))
	T3:	(AUX1) ← (RD)
M4:	T1:	(RA) ← (AUX1,AUX2)
		MREAD
	T2:	(RD) ← ((RA))
	T3:	(A) ← (RD)

Observatii:

1. O instructiune complexa , 4 cilcuri masina, 13 stari
   M1 – fetch pur
   M2, M3 – citire din memoria de program
   M4 – citire din mem de date
2. Formatul instructiunii are 3 octeti;
   M2,M3 acceseaza mem de prg pt a citi formatul complet al instructiunii, ce am in plus fata de cod
3. Atentie la acele actiuni elementare care pot dura mai mult de o stare
   Ex: T1 din M4
4. Act elementare formal identice cu semnificatii diferite:
   T2 din M1 aduc codul
   T2 din M2, M3 aduc o adresa
   T2 din M4 aduc un operand

Instructiunea (PC)←adr

- salt cu adresare directa (adr completa)
- M4 = saltul

(PC) ← adr

M1: T1: (RA) ← (PC) ...., T2: ...., T3: ...., T4: ....

M2: T1: (RA) ← (PC)
MREAD
T2: (PC) ← (PC) + 1, (RD) ← ((RA))
T3: (AUX2) ← (RD)

M3: T1: (RA) ← (PC)
MREAD
T2: (PC) ← (PC) + 1, (RD) ← ((RA))
T3: (AUX1) ← (RD)

M4: T1: (RA) ← (AUX1,AUX2)
MREAD
T2: (PC) ← (AUX1,AUX2)
T3: (PC) ← (PC) + 1

Observatii:

1. Instructiune cu patru cilcuri masina. Format pe 3 octeti – M1, M2, M3 – identice cu cazul precedent
2. A se nota utilizarea reg AUX1, AUX2
   - registre tampon

- am nevoie sa stochez temporar adresa operandului (o aduc in memorie nu o am in procesor)
   - AUX2 si pe urma AUX1 datorita micului indian

3. M4 poate fi suprapus cu o instructiune urmatoare pentru ca saltul inseamna inceputul unei noi instructiuni si se pot suprapune

Din motive de eficienta in timp adresa de salt se incarca in RA

Saltul se face in T1 din M4
Ulterior actualizez PC

Concluzii finale pt 7.1

Procesoarele CISC, bazate pe nucleul prez in capitolul 2 – un CISC pur – prezinta urmatoarele caracteristici a desfasurarii in timp a instructiunii:

- instructiunile se desfasoara neuniform in timp, depinde de complexitatea instructiunii
- criteriul de performanta este viteza in conditiile date
- sunt posibile suprapuneri a 2 instructiuni dintr-o secventa, mai mult de 2, dar in mod neuniform si ocazional

..:::7.2 CRESTEREA VITEZEI DE EXECUTIE PENTRU CISC EVOLUATE:::…

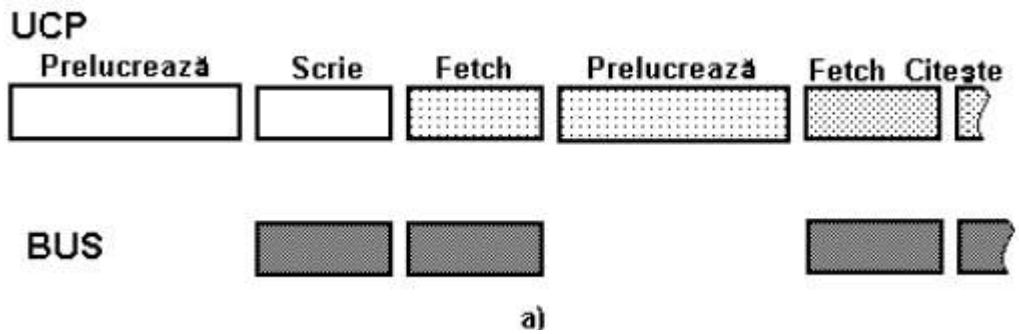
- are in vedere schema bloc functionala din 3.1
- se refera la CISC incepand cu generatia a 3-a

Avem in vedere 2 caracteristici fundamentale prezentate in 3.1:

1. Existenta mai multor unitati care functioneaza in paralel pentru ca au functii distincte
2. Existenta conceptului de coada de instructiuni.

Fie un procesor CISC clasic conform cap 2. Are de efectuat o secventa de instructiuni care constau in cicluri masina ca mai jos. Alegerea este intamplatoare. Succesiunea de instructiuni din punct de vedere al ciclurilor masina este intamplatoare.

Diagrama de tip simplificat cu ipoteze simplificatoare:

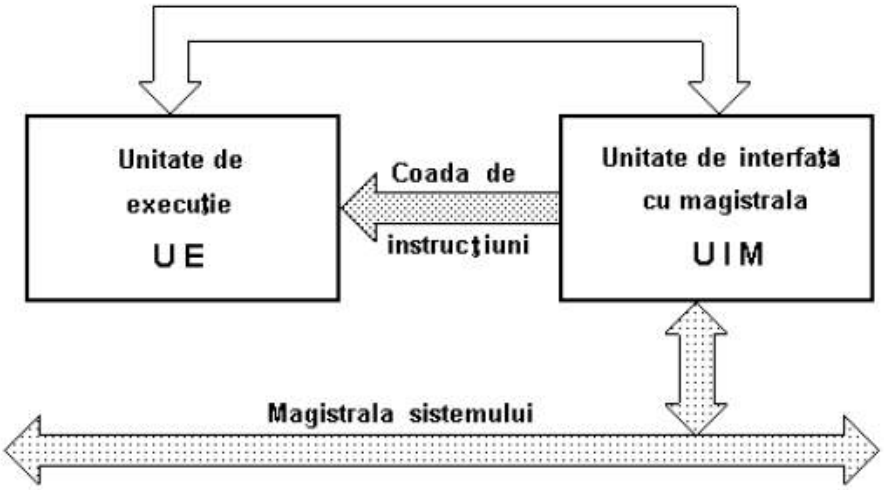


a)

fetch	prelucrează	scrie rezultat
fetch	prelucrează	
fetch	citeşte operand	prelucrează
fetch	.....	
fetch	.....	

1. Orice accesare in memorie dureaza la fel
2. Toate prelucrarile de operanzi dureaza la fel, sunt mai lungi decat accesul in memorie
3. Neglijam suprapunerile ocazionale si neuniforme dintre 2 instructiuni succesive

Exemplu:
Procesorul are 2 unitati:
- UE
- UIM
Au functii distincte si pot functiona in paralel



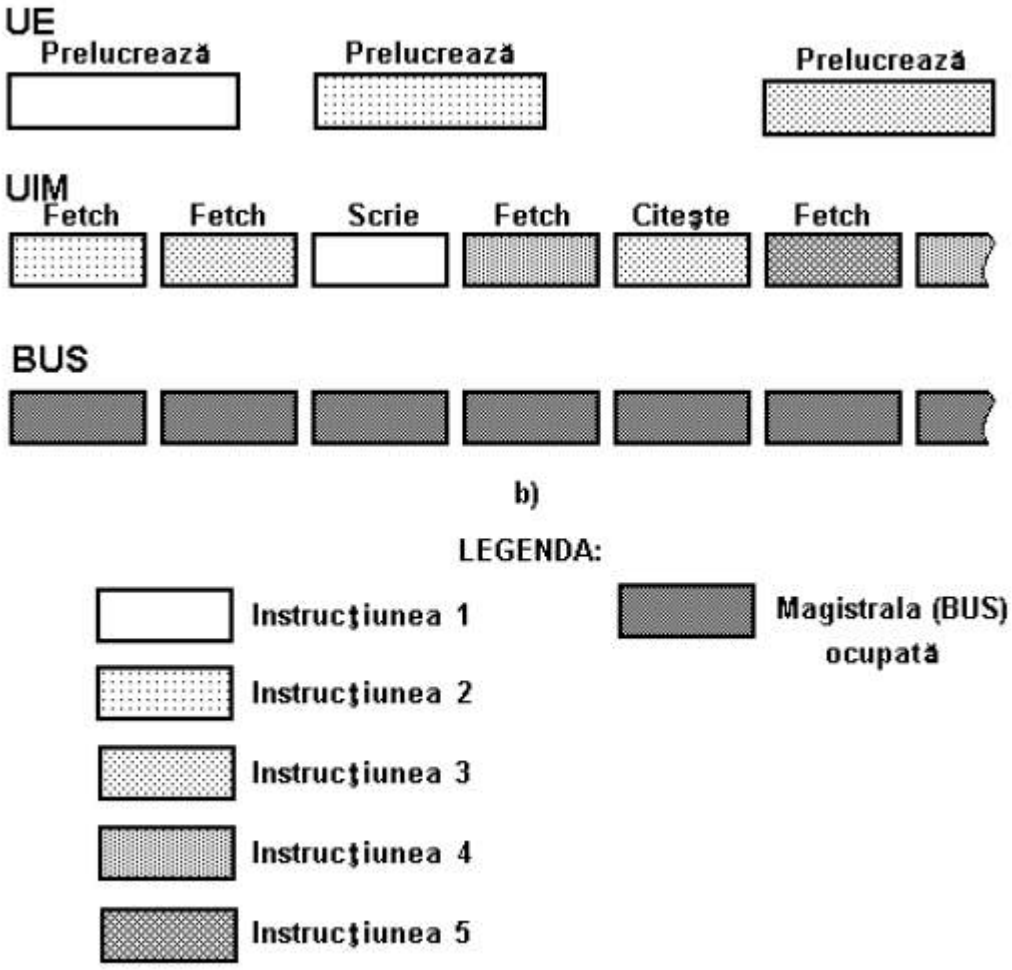
1. UE prelucreaza operanzi. Toate informatiile de la magistrala nu vin direct ci de la UIM

Orice informatie spre magistrala nu merge direct ci la UIM

2. UIM acceseaza memoria de date si de program. Ea e legata de magistrala microcalculatorului.

- mentine coada de instructuni plina – **principala sarcina UIM**

= accesul memoriei de program este prioritar fata de accesul memoriei de date.



Concluzii:

1. Executia secventelor de instructiuni este mai rapida
2. Viteza nu e dubla
3. Magistrala de date este ocupata complet, ceea ce nu era cazul la un CISC clasic

Concluzii pentru 7.2

1. CISC incepand cu generatia a 3-a , prezinta urmatoarele caracteristici de desfasurare in timp a instructiunilor
2. Desfasurarea in timp permite suprapunerea sistematica dar tot neuniforma a instructiunilor dintr-o secventa, eventual mai multe instructiunilor
3. Magistrala sistemului unica conceptului Von-Neumann este complet ocupata

…:::7.3 CONCEPTELE DESFASURARII IN TIMP A INSTRUCTIUNIILOR PENTRU RISC:::…

Desfasurarea in timp pt un RISC se bazeaza pe urmatoarele premise:

- **instructiunile** sunt uniforme ca desfasurare in timp, **toate dureaza acelasi numar de stari**. Acest lucru este posibil deoarece:
  a) prelucrarea operanzilor este realizata cablat
  b) exista multe registre interne ceea ce permite accesarea memoriei in mult mai putine ocazii
  c) avem putine instructiuni, codificarea este simpla si rapida
  d) avem putine moduri de adresare, ea este simpla si rapida
  e) format identic ca numar de octeti pentru toate instructiunile si putine tipuri de format
  Acestea duc la simplificarea unitatii de control care este cablata si nu microprogramata

Exista premise caracteristice tuturor tipurilor de procesoare:

- existenta cozii de instructiuni
- memorii mici rapid accesate care sunt memorii tampon pentru date si instructiuni sau date

Toate aceste premise permit o desfasurare uniforma si rapida.
Uniformitatea permite folosirea pipeline.

Daca presupunem ca toate instructiunile dureaza **n** stari atunci banda rulanta se zice ca are **n etaje** sau **stadii** => intr-o stare gasim **n** instructiuni in diverse etape de desfasurare.

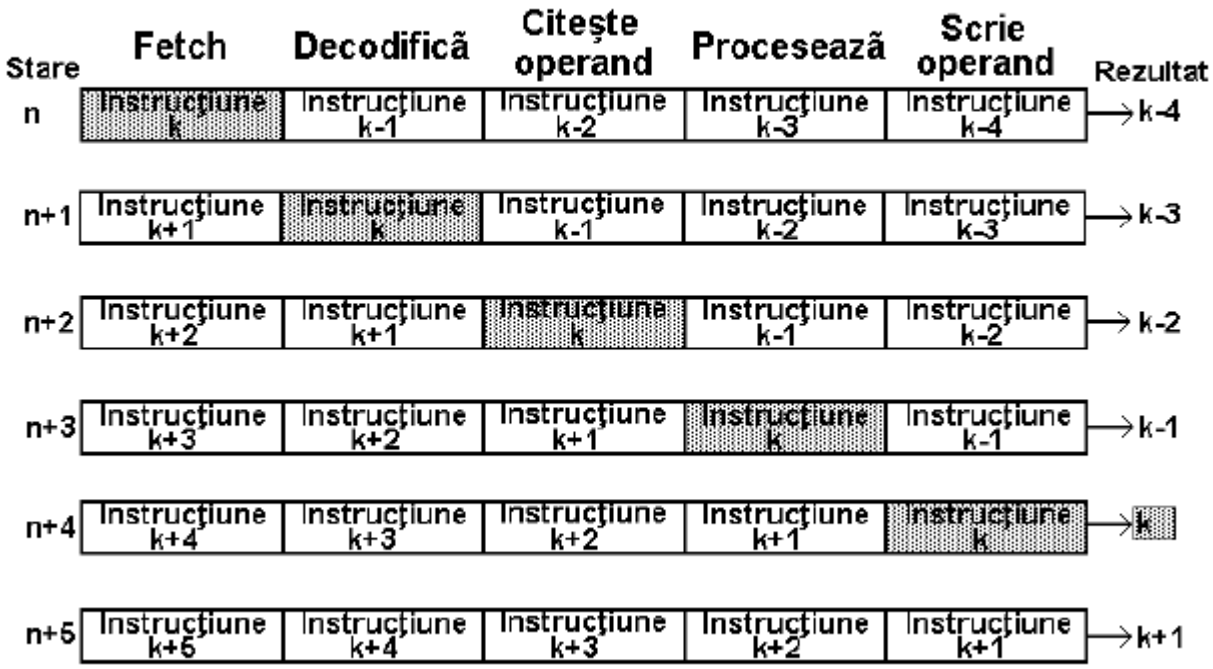
Premisele+ pipleine => **CPI=1 - caracteristica definitorie CISC**

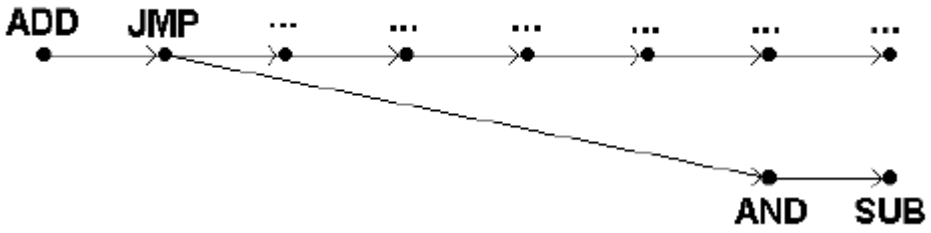
Exemplu:

Fie instructiunea RISC care au 5 stari.

Semnificatii :
S1 – fetch
S2 – decodifica
S3, S4, S5 – depind de tipul de instructiuni

a) pentru instructiunile de prelucrari de date:
   S3 – citeste operand
   S4 - prelucreaza
   S5 – scrie rezultat
b) pentru accesarea memoriei de date si/sau de program (load, store, salt)
   S3, S4, S5 – accesarea memoriei





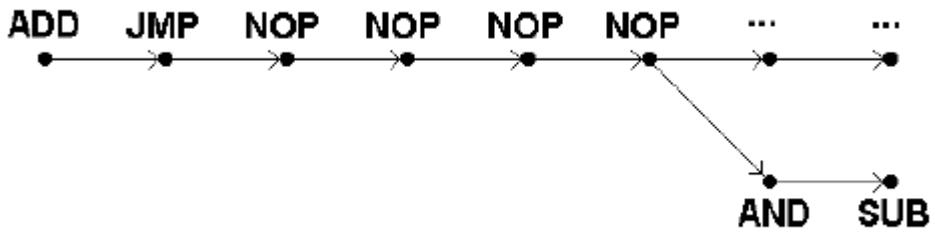
- pentru ca rezultatul pentru salt stie unde se face saltul

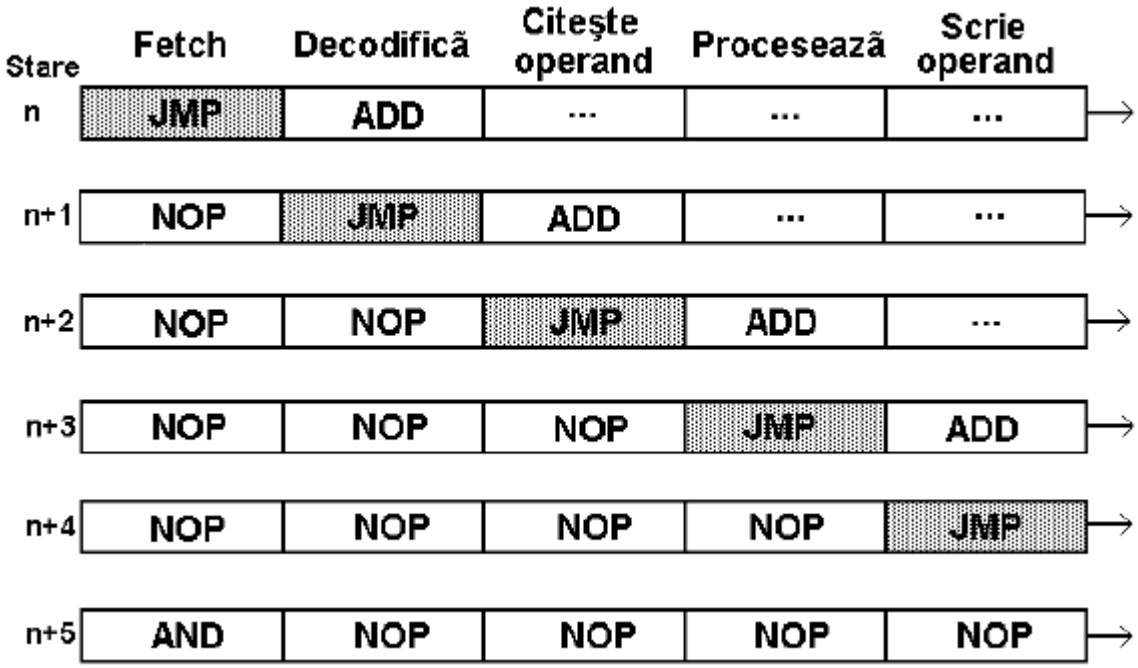
Fetch	Decodifica	Citeste operand	Proceseaza	Scrie operand
JMP	ADD	....	....	....
XOR	JMP	ADD	....	....
ADD	XOR	JMP	ADD	....
OR	ADD	XOR	JMP	ADD
ADD	OR	ADD	XOR	JMP
AND	ADD	OR	ADD	XOR
SUB	AND	ADD	OR	ADD
	SUB	AND	ADD	OR
		SUB	AND	ADD
			SUB	AND

- cand JMP ajunge la **scrie operand** - procesorul stie unde se face saltul in acest punct
- CPI≠1 pentru intercalarea cu NOP
- **AND** este instructiune utila, restul instructiunilor in starea **n+5** sunt rezultate gresite

Solutii pentru impiedicarea obtinerii de rezultate eronate

- imping secventa de instructiuni nedorita mai departe





- primul rezultat apare in **n+9**

Optimizarea prevenirii blocarii unitatii de control din cauza salturilor

```
        ADD r3, r2, r1 – se aduna continutul lui r1 la cont lui r2 si
              rezultatul se depune in r3
        AND r0, r5, r6
        JMPZ r0, eticheta
        NOP
        .....................
eticheta  SUB r1, r5, r6
```

- se face un sir logic intre continutul lui r5 si continutul lui r6 si rezultatul se depune in r0
Numarul de NOP-uri care trebuie adaugate depinde de strutura benzii rulante

```
        AND  r0, r5, r6
        JMPZ r0, eticheta
        ADD r3, r2 ,r1 – intra pe teava si se executa
        ..................
eticheta   SUB r1, r5, r6
```

Pot schimba ordinea instructiunilor pentru ca:

- am suficiente registre pentru ca instructiunile sa fie pe anumite secvente cvasiindependente
- pipeline-ul permite intrarea instructiunilor chiar dupa salt

Intarzieri din cauza accesului in memorie

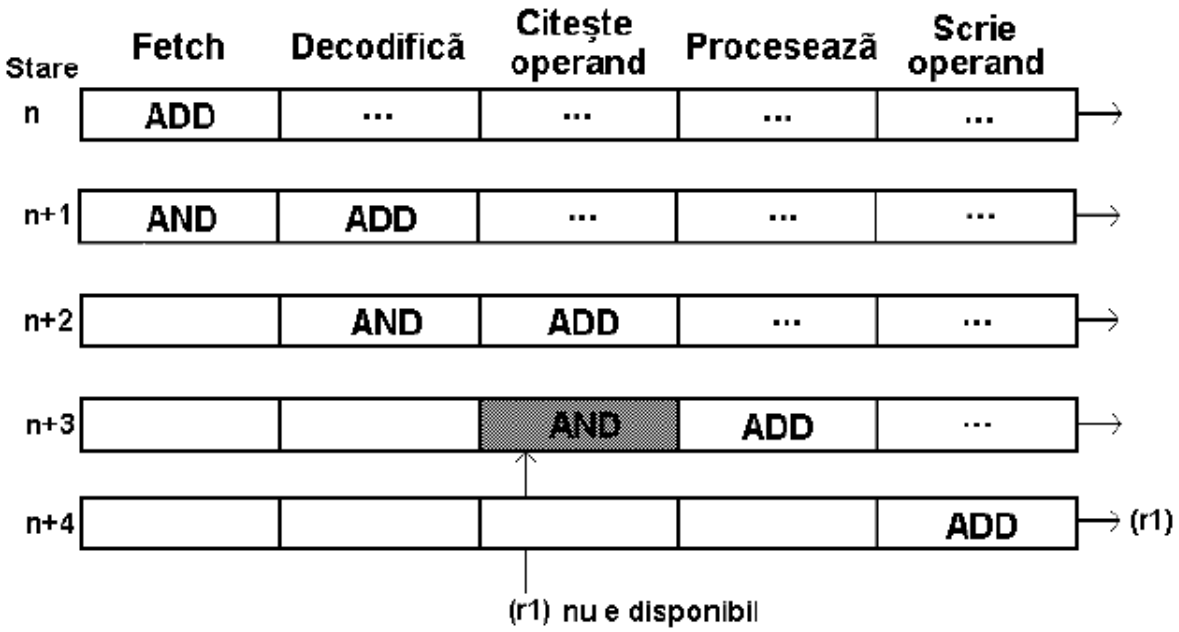
```
LOAD  r1, mem
ADD   r3,r2,r1  - continutul lui R1 nu e cel corect ; trebuie intercalate NOP-uri
LOAD  r1, mem
NOP
NOP
....
NOP
ADD   r3, r2, r1
```

Numarul de instructiuni NOP depinde de:

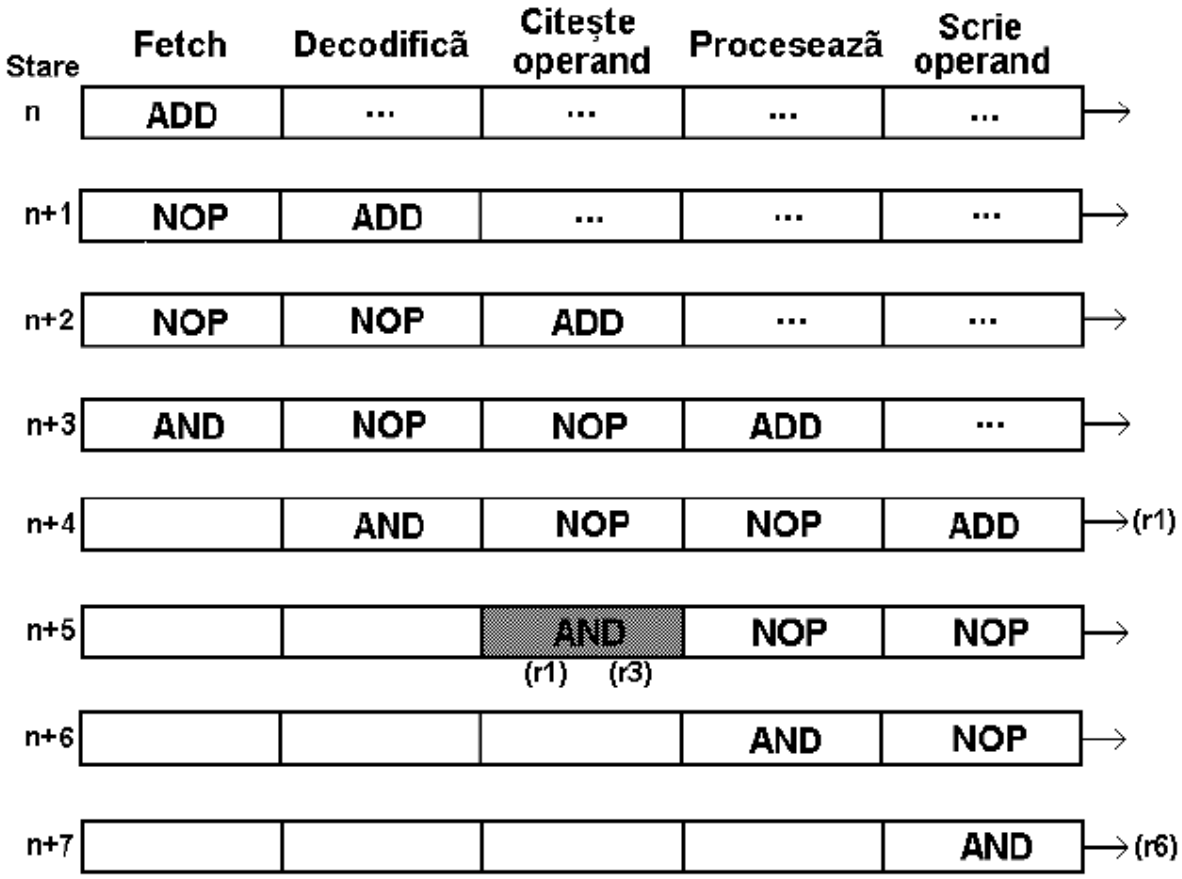
a) structura benzii rulante
b) numarul de stari necesare accesarii memoriei

- din cauza dependentei datelor de utilizare a registrelor

```
ADD     r1, r2, r7
AND     r6, r1, r3
```



```
ADD     r1, r2, r7
NOP
NOP
AND     r6, r1, r3
```



Concluzii preliminare:

1. Masurile care trebuie luate pentru buna functionare ce proceduri tin de utilizarea corecta a resurselor procesorului (Soft-ul pentru aceste procesoare este scris de fabricant)
2. Utilizand aceste masuri pentru buna functionare CPI>1

CONCLUZII PENTRU 7.3

1. **RISC au instructiuni uniform desfasurate in timp. Acelasi numar de stari cu stari similare.**
2. **Aceasta desfasurare uniforma este rezultatul unor caracteristici de structura si arhitectura care au fost trecute in revista.**

3. **Aceasta uniformitate in timp permite folosirea tehnicii benzii rulante de o maniera sistematica si uniforma. Suprapunerea duce la CPI=1.**
4. **Exista motive pentru o suprapunere neuniforma a instructiunii (salturi) accesul memoriei, dependenta datelor de utilizare a registrelor. In aceste cazuri CPI>1. In anumite conditii, masurile luate pentru functionarea corecta pot fi optimizate (vezi cazul salturilor).**