SISTEM DE CALCUL IN TIMP REAL

SCTR

-SZOKE ENIKO -

Curs 5 - continuare Curs 3

Arhitecturi de 3-adrese

- Sunt arhitecturile de calcul care se regasesc in microprocesoarele care echipeaza majoritatea calculatoarelor personale actuale.

- Instructiunile ofera posibilitatea de a descrie fiecare cate trei adrese in campul operand, care pot fi toate adrese explicite (registri generali sau adrese de memorii), fie una este implicit registrul acumulator celelalte doua fiind adrese explicite.

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP	X	M [X] ← TOS

$$X = (A + B) * (C + D)$$

Arhitecturi de 0-adrese

TOS top of stack
varful stivei

LOAD	A	AC ← M [A]
ADD	B	AC ← A [C] + M [B]
STORE	T	M [T] ← AC
LOAD	C	AC ← M [C]
ADD	D	AC ← AC + M [D]
MUL	T	AC ← AC * M [T]
STORE	X	M [X] ← AC

Arhitecturi de 1-adresa

MOV	R1, A	R1 ← M [A]
ADD	R1, B	R1 ← R1 + M [B]
MOV	R2, C	R2 ← M [C]
ADD	R2, D	R2 ← R2 + M [D]
MUL	R1, R2	R1 ← R1*R2
MOV	X, R1	M [X] ← R1

$$X = (A + B) * (C + D)$$

Arhitecturi de 2-adrese

ADD	R1, A, B	R1 ← M [A] + M [B]
ADD	R2, C, D	R2 ← M [C] + M [D]
MUL	X, R1, R2	M [X] ← R1 * R2

Arhitecturi de 3-adrese

Example: Y=(A-B)/(C+D x E)

- One Addresses:

```
LOAD  D      # AC← D
MPY   E      # AC← ACxE
ADD   C      # AC← AC+C
STOR  Y      # Y← AC
LOAD  A      # AC←A
SUB   B      # AC←AC-B
DIV   Y      #...
STOR  Y      #...
```

- Stack (0 address)

```
Push  A
Push  B
SUB
Push  C
Push  D
Push  E
MPY
ADD
DIV
Pop   Y
```

Example: Y=(A-B)/(C+D x E)

- Three Addresses:

```
SUB   Y, A, B      # Y← A-B
MPY   T, D, E      # T← DxE
ADD   T, T, C      # T← T+C
DIV   Y, Y, T      # Y← Y/T
```

- Two Addresses

```
MOV   Y, A         #Y← A
SUB   Y, B         #Y←Y-B
MOV   T, D         #T←D
MPY   T,E          #T←TxE
ADD   T,C          # T←T+C
DIV   Y, T         # Y←Y/T
```

- SC bazate pe arhitecturi von Neumann sunt limitate in viteza lor de calcul de constrangerea ca secventele de incarcare (fetch)si executie (execute) ale fiecarei instructiuni sa se faca serial. Din acest motiv, doua sau mai multe instructiuni nu pot fi executate simultan, ceaa ce limiteaza performantele procesoarelor.

- Solutia cea mai utilizata se bazeaza pe faptul ca secventele de fetch si de executie sunt secvente distincte si ca ele reprezinta numai o parte dintre secventele parcurse in executia unei instructiuni (mai exista decode) si ca secvente diferite din instructiuni diferite pot fi suprapuse.

- **Pipelining.**

- In timp ce o instructiune se afla in secventa de execute, urmatoarea instructiune in secventa de fetch.

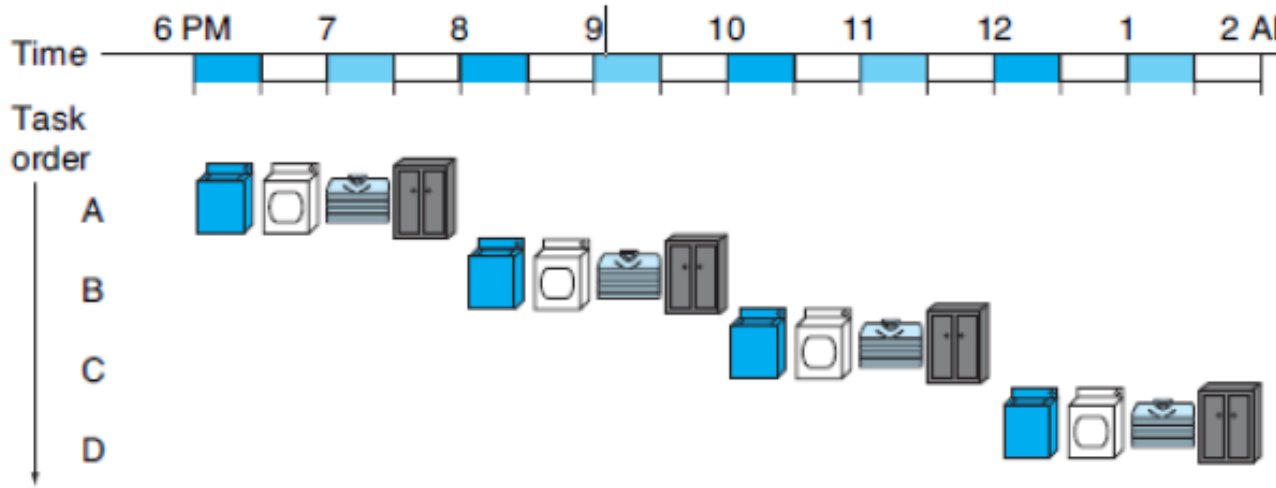
- Tehnica **pipelining** se poate aplica la majoritatea secventelor de program. Problemele apar in momentul in care urmeaza sa fie executate instructiuni de salt, care duc la modificarea brusca a contorului de program, rupandu-se secventialitatea programului. In acest caz, ca si in cazul aparitiei intreruperilor interne sau externe, cea mai veche instructiune aflata in executie in regim pipeline trebuie incheiata iar cele care urmeaza si sunt deja in regim pipeline trebuiesc sa fie memorate in starea in care se afla, fie trebuie retrase si reinceputa executia lor atunci cand vor fi apelate.

Pipeline

- Tehnica de *pipeline* îmbunătăţeşte eficienţa prin execuţia suprapusă a mai multor instrucţiuni

- Conceptual, tehnica de pipeline:
  - ✓ sparge un proces complex în mai multe etape / faze
  - ✓ dacă fiecare fază necesită resurse diferite, atunci acestea se pot executa concomitent

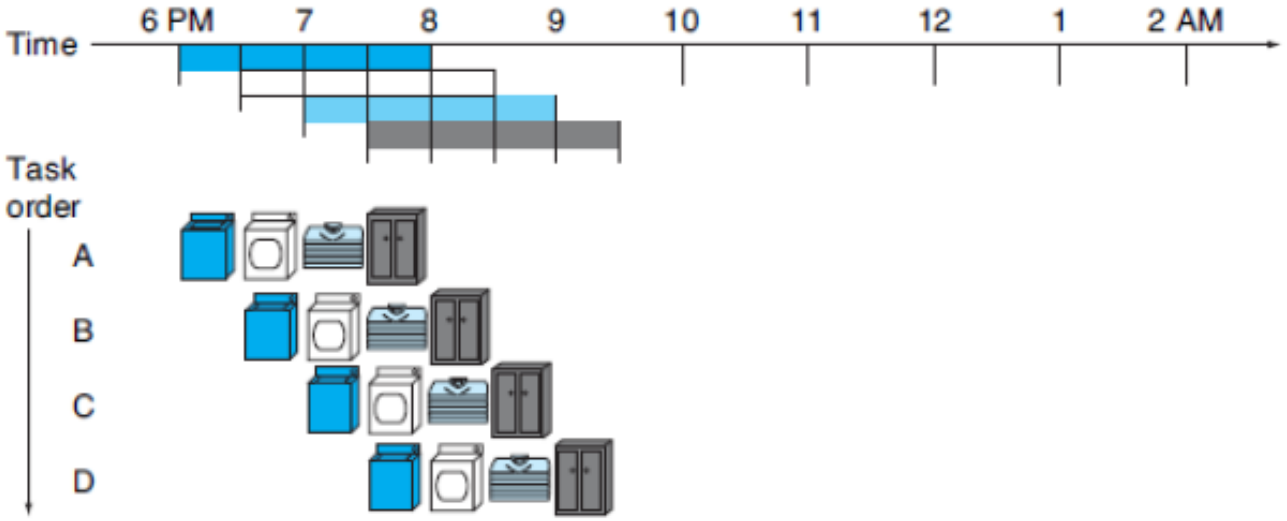
Exemplu:



[COD]

- Fiecare etapă (spălare, uscare, călcare, aşezare) se realizează secvenţial, pe fiecare set de rufe
- Când s-a finalizat un set de rufe, se trece la următorul

- Dacă se aplică tehnica de pipeline [COD]:

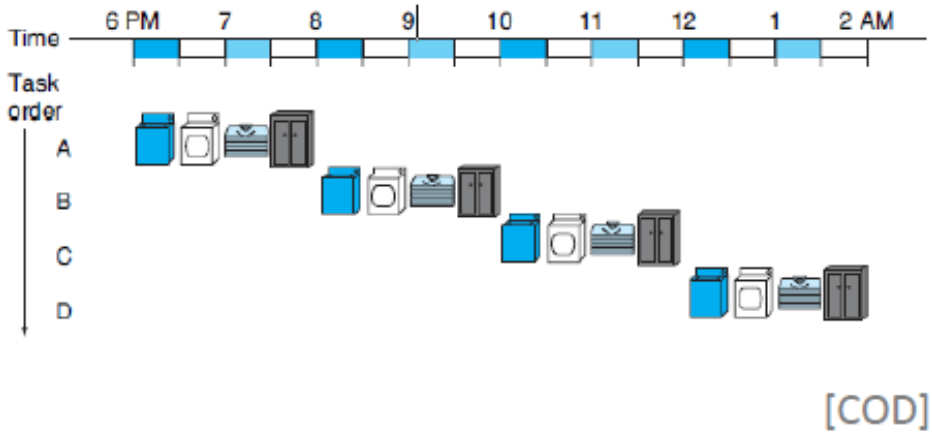


[COD]

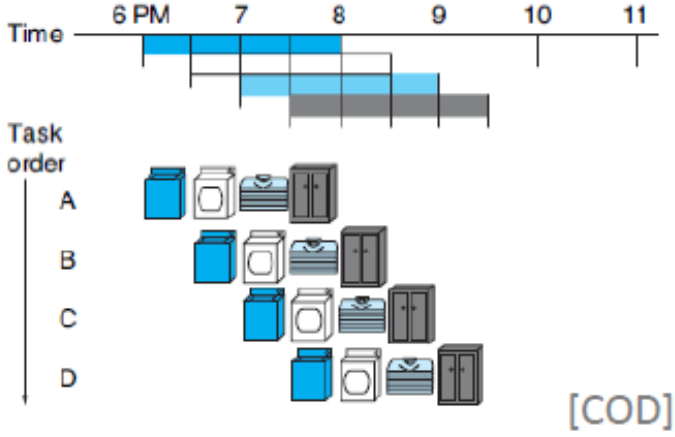
- Când pentru un set s-a finalizat o etapă, atunci trece în etapa următoare şi se poate introduce un alt set în etapa iniţială
- Spre exemplu, după ce setul A este spălat trece la uscat, timp în care se poate spăla un alt set B

- *Întrebare:* Cât durează spălarea setului A fără se se folosească tehnica de pipeline? Dar când se foloseşte pipeline?

- *Răspuns:* 2h în ambele cazuri



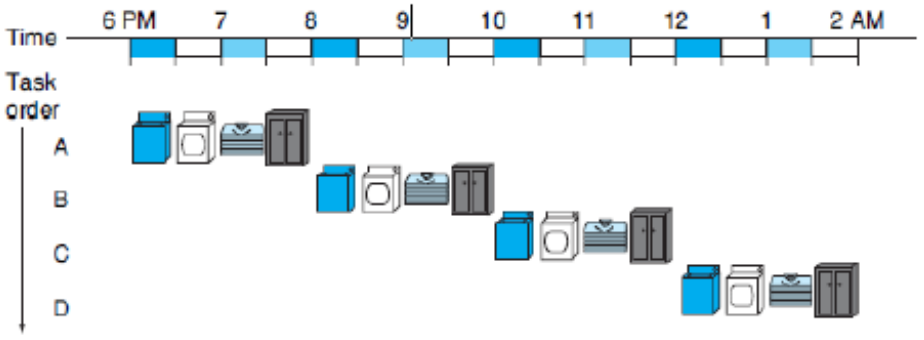
[COD]



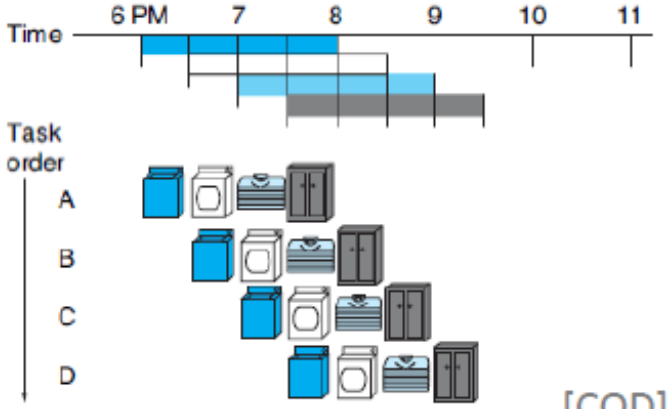
[COD]

- *Întrebare:* Cât durează spălarea seturilor A, B, C, D fără se se folosească tehnica de pipeline? Dar când se foloseşte pipeline?

- *Răspuns:* 8h, respectiv 3.5h (implementarea pipeline este de aprox. 2.28 mai rapidă)



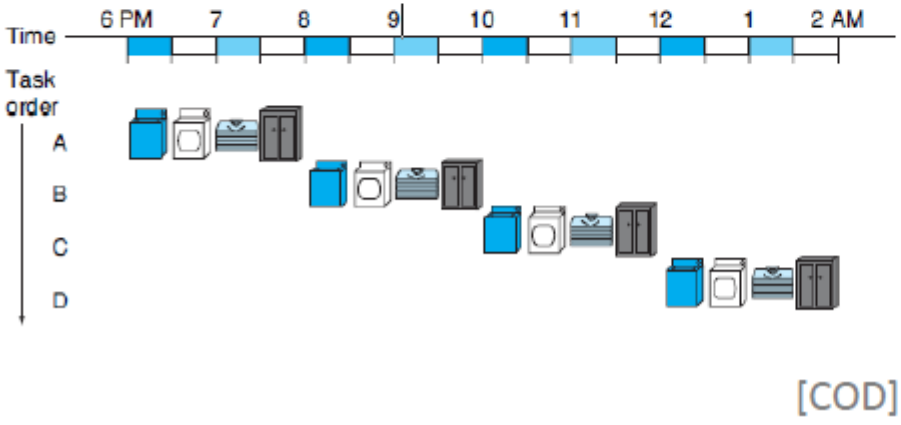
[COD]



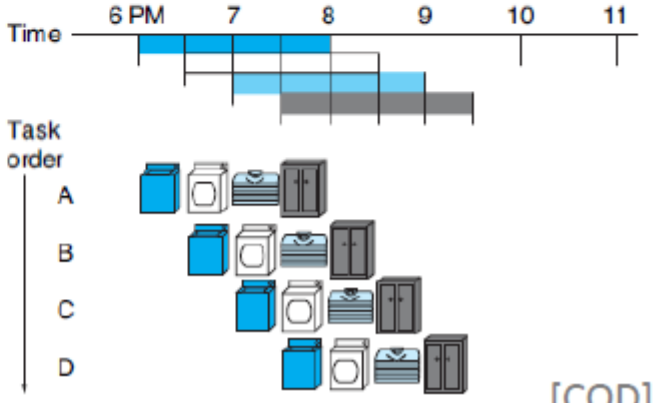
[COD]

- *Întrebare:* Considerând un proces continuu (un şir infinit de seturi A, B, C...) de câte ori este mai rapidă varianta care implementează pipeline?

- *Răspuns:* De 4 ori (cu excepţia unei întârzieri la început, la fiecare oră se va finaliza un set)



[COD]



[COD]

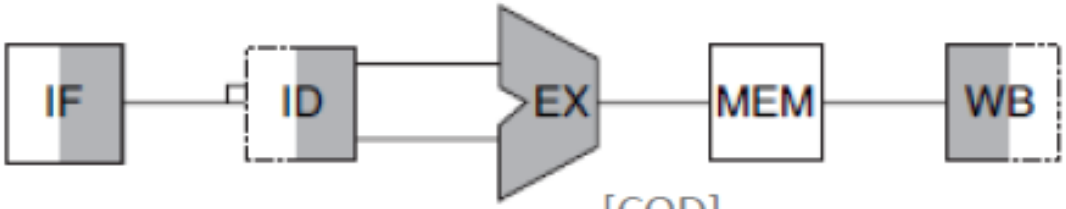
- Utilizarea pipeline NU ajută dacă volumul procesat este foarte mic (în practică poate chiar întârzia, din cauza unor procesări suplimentare); ex.: timpul necesar pentru finalizarea unui singur set este acelaşi

- Pipeline **îmbunătăţeşte productivitatea** (throughput) prin creşterea volumului procesat într-un anumit timp, şi nu prin scăderea timpului necesar pentru o entitate; ex.: timpul pentru un set nu se micşorează, dar în total se termină mai multe seturi de rufe)

- Pentru MIPS tehnica de pipeline prezintă 5 faze:

1. **Încărcarea** instrucţiunilor din memorie – *IF* (*I*nstruction *F*etch)
2. **Decodarea** instrucţiunii şi citirea regiştrilor – *ID* (*I*nstruction *D*ecode)
3. **Execuţia** operaţiei (ex.: `add`) sau calculul unei adrese (ex.: `lw`) – *EX* (*EX*ecution)
4. **Accesarea** memoriei de date (ex.: `sw`) – *MEM* (*MEM*ory access)
5. **Scrierea** rezultatului în registru (ex.: `add`) – *WB* (*W*rite *B*ack)

- **Observaţie!** Nu toate instrucţiunile necesită toţi paşii; ex.: `add` nu foloseşte pasul 4 pentru că nu utilizează memoria de date



[COD]

MIPS – (million instructions per second)

- Accesul la memorie numai prin 2 instructiuni Load/Store

Instructiuni aritmetice si logice

```
add $rd, $rs, $rt                ;$rd = $rs + $rt
addi $rt, $rs, imm               ;$rt = $rs + imm
sub $rd, $rs, $rt                ;$rd = $rs - $rt
mult $rs, $rt                    ;$LO = $rs * $rt
div $rs, $rt      ;$LO=$rs/$rt; $HI=$rs % $rt
and $rd, $rs, $rt                ; $rd = $rs & $rt
andi $rt, $rs, imm               ; $rt = $rs & imm
or $rd, $rs, $rt                 ; $rd = $rs | $rt
ori $rt, $rs, imm                ; $rt = $rs | imm
```

Instructiuni LOAD/STORE

- Load word

```
lw $rt, offset($rs)        ; $rt = MEM[$rs + offset]
```

- Load byte

```
lb $rt, offset($rs)        ; $rt = MEM[$rs + offset]
```

- Store word

```
sw $rt, offset($rs)        ; MEM[$rs + offset] = $rt
```

- Store byte

```
sb $t, offset($s)   ; MEM[$s + offset] = (0xff & $t)
```

Instructiuni de salt

Salturi neconditionate

j target ; PC = (PC & 0xf0000000) | (target << 2)

jr $rs ; salt cu registru PC = $rs;

Salturi conditionate (ramificari – branch)

- Branch on equal

beq $rs, $rt, offset ; if $rs=$rt PC=PC+(offset<<2))

- Branch on greater than or equal with zero

bgez $rs, offset ; if $rs >= 0 PC=Pc+ (offset << 2))

Pipeline

- *Întrebare:* Care paşi sunt necesari pentru fiecare din instrucţiunile `lw`, `sw`, `add`, `beq`?

- *Răspuns:*

	IF	ID	EX	MEM	WB
lw	✓	✓	✓	✓	✓
sw	✓	✓	✓	✓	×
add	✓	✓	✓	×	✓
beq	✓	✓	✓	×	×

Hazard

- Există situaţii când o etapă de procesare a unei instrucţiuni nu se poate executa în următoarea etapă din pipeline

- O astfel de situaţie poartă denumirea de *hazard*

- Exista 3 tipuri de hazard:
  - ✓ *Hazard structural* (**restricţie fizică**) : cauzat de hardware

  ex.: se foloseşte aceeaşi componentă hardware pentru 2 etape succesive
  - ✓ *Hazard de date* (**restricţie logică**) : cauzat de dependenţa unei variabile (registru, valoare, etc.)

  ex.: nu se cunoaşte încă valoarea unui registru, dar se foloseşte într-o altă instrucţiune
  - ✓ *Hazard de control* (**restricţie logică**): cauzat de instrucţiunile de salt

  ex.: trebuie luată o decizie pe baza unui rezultat incă necalculat, se încarcă (IF) următoarea instrucţiune, dar nu aceasta este cea care trebuie executată următoarea

Hazard structural

- *Hazardul structural* apare când nu se pot executa toate operaţiile necesare într-o etapa de pipeline din cauza unor restricţii hardware

- Fiindcă este introdus de o limitare hardware, este o restricţie **fizică**

- În exemplul iniţial (spălatul rufelor), apare hazard structural dacă:
  - ✓ aceeaşi maşină realizează spălarea şi călcarea
  - ✓ aceeaşi persoană calcă şi aşează rufele

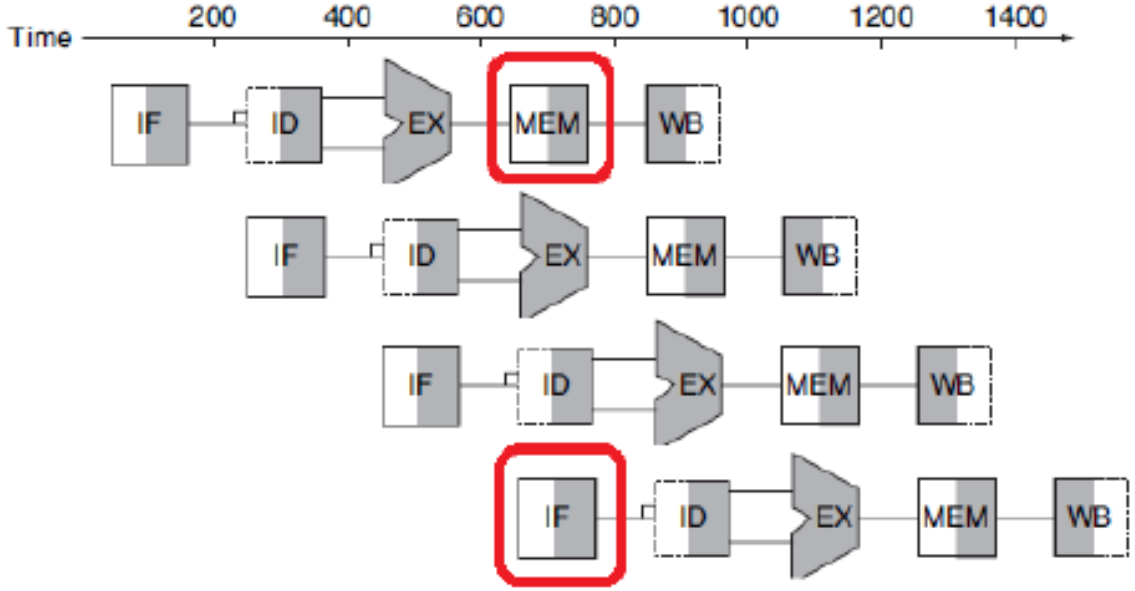
- Pentru un procesor cu o memorie comună de date şi instructiuni (care să nu poată realiza operaţii paralele de acces), un exemplu de hazard structural este când se încearcă scrierea simultană în memorie (sau citirea simultană din memorie)

Hazard structural

```
lw $s1,0($s1)
lw $s2, 0($s2)
lw $s3,0($s3)
lw $s4, 0($4)
```

- *Întrebare:* Unde apare hazardul structural ?

- *Răspuns:* La al 4-lea tact, pentru instr.1 citeşte datele din memorie şi instr.4 se încarcă în memorie (avem deci acces simultan pentru citire din memorie)



Hazard de date

- *Hazardul de date* apare când o instrucţiune nu se poate executa în tactul de ceas corespunzător pentru că datele necesare execuţiei nu sunt încă disponibile

- Fiindcă nu este introdus de o limitare hardware, este o restricţie **logică**

- În exemplul iniţial (spălatul rufelor), apare hazard de date dacă:
  - ✓ la etapa de aşezare a rufelor se găseşte o şosetă fără pereche (trebuie să se aştepte perechea pentru a putea fi strânse şi aşezate corespunzător la loc)

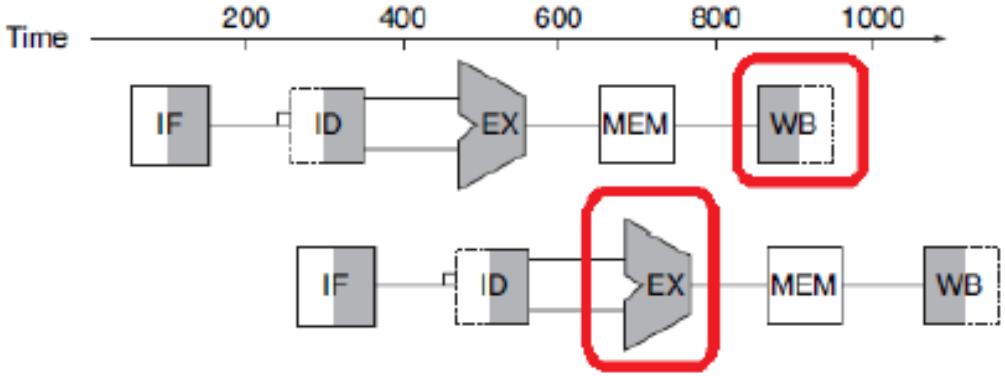
- În calculator, hazardul de date apare dacă se foloseşte o valoare care încă nu este calculată

Hazard de date

- *Întrebare:* Unde apare hazardul de date ?

```
add $t0, $t0, $t1
add $t4, $t0, $t3
```

- *Răspuns:* Valorea sumei pentru instr.1 se scrie în registrul `$t0` în etapa WB, în timp ce instr.2 necesită valoarea în etapa EX, care este anterioară ca timp



Hazard de date

- O metoda de rezolvare a hazardului de date este *forwarding* (sau *bypassing*; sau *tehnica de avansare*): valorile se preiau din regiştrii care delimitează cele 5 etape pipeline (ex.: EX/MEM)

- Aceasta funcţionează dacă timpul sursei datelor este anterior timpului de utilizare

- În unele cazuri acest lucru nu se întâmplă şi atunci este nevoie să se introducă **întârzieri** suplimentare (*nop* = no operation sau *bubble* = pipeline stall); aceasta se numeşte *tehnica de întârziere*

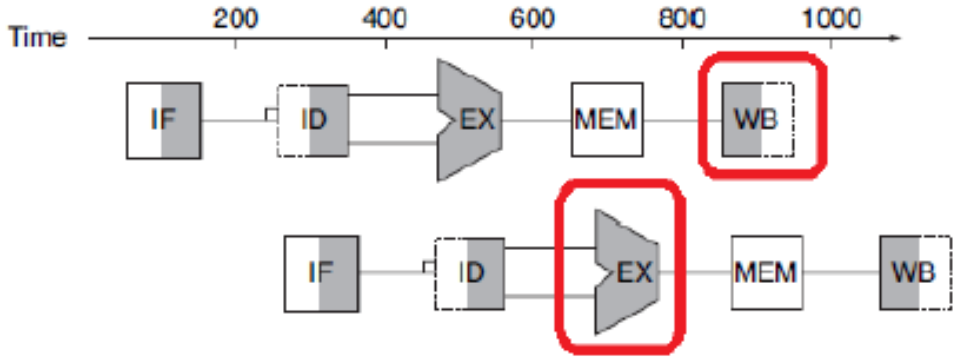
- O tehnică **generală** de evitare a hazardului (de date, dar şi de control) este **reordonarea instrucţiunilor**

Hazard de date

- *Întrebare:* Unde apare hazardul de date ?

```
add $s0, $t1, $t2
sub $t2, $t0, $t3
```

- *Răspuns:* Valorea sumei pentru instr.1 se scrie în registrul `$t0` în etapa WB, în timp ce instr.2 necesită valoarea în etapa EX, care este anterioară ca timp

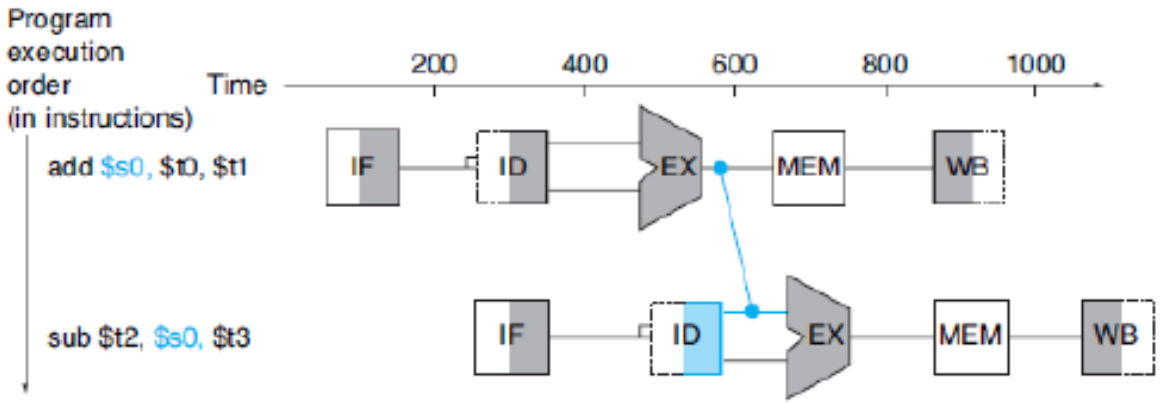


Hazard de date

- *Întrebare:* Cum poate rezolva tehnica de avansare hazardul de date?

```
add $s0, $t1, $t2
sub $t2, $t0, $t3
```

- *Răspuns:* Se preia valoarea lui `$s0` direct după calculul acesteia din instr.1, pentru a putea fi utilizată direct în etapa de execuţie a instr.2

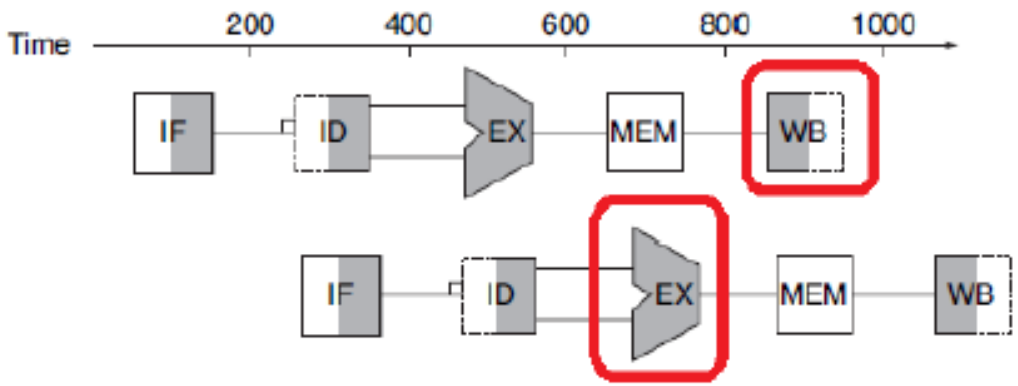


Hazard de date

- *Întrebare:* Unde apare hazardul de date ?

```
lw $s0, 20($t1)
sub $t2, $s0, $t3
```

- *Răspuns:* Valorea încărcată din memorie pentru instr.1 se scrie în registrul `$s0` în etapa WB, în timp ce instr.2 necesită valoarea în etapa EX, care este anterioară ca timp

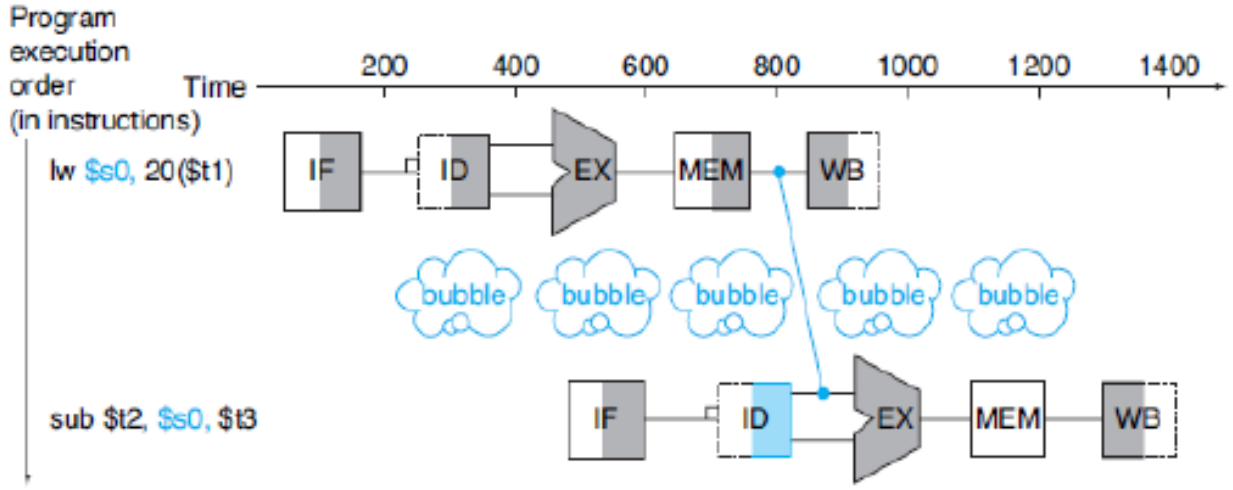


Hazard de date

- *Întrebare:* Cum poate rezolva tehnicile de avansare şi întârziere hazardul de date?

```
lw $s0, 20($t1)
sub $t2, $s0, $t3
```

- *Răspuns:* Se preia valoarea lui `$s0` direct după citirea din memorie din instr.1, pentru a putea fi utilizată cu 1 singur tact întârziere în etapa de execuţie a instr.2



[COD]

Hazard de date

- *Întrebare:* Ce face secvenţa de cod de mai jos?

Ne referim la variabilele stocate la locaţiile de memorie astfel:

```
a = 0($t0); b= 4($t0); c = 8($t0); d=12($t0); e=16($t0)
```

```
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1,$t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1,$t4
sw $t5, 16($t0)
```

- *Răspuns:* Calculează:

```
d = a + b
e = c + a
```

Hazard de date

- *Întrebare:* **Cum poate rezolva reordonarea codului hazardul de date? (care apare spre exemplu la ambele instr. add)**

- *Răspuns:*

```
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1,$t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1,$t4
sw $t5, 16($t0)
```

```
lw $t1, 0($t0)
lw $t2, 4($t0)
lw $t4, 8($t0)
add $t3, $t1,$t2
sw $t3, 12($t0)
add $t5, $t1,$t4
sw $t5, 16($t0)
```

Hazard de control

- *Hazardul de control* apare când trebuie luată o decizie bazată pe rezultatul unei instrucţiuni în timp ce altele se execută

- Fiindcă nu este introdus de o limitare hardware, este o restricţie **logică**

- În exemplul inţial (spălatul rufelor), apare hazard de control dacă:
  - ✓ se doreşte ajustarea (cantităţii sau a tipului) detergentului folosit în funcţie de cât de bine este spălat un set de rufe

- Pentru procesor, un exemplu de hazard de control apare la condţionări: se încarcă (IF) următoarea instrucţiune, dar nu aceasta este cea care trebuie executată următoarea (din cauza condiţiei de salt care se poate îndeplini sau nu)

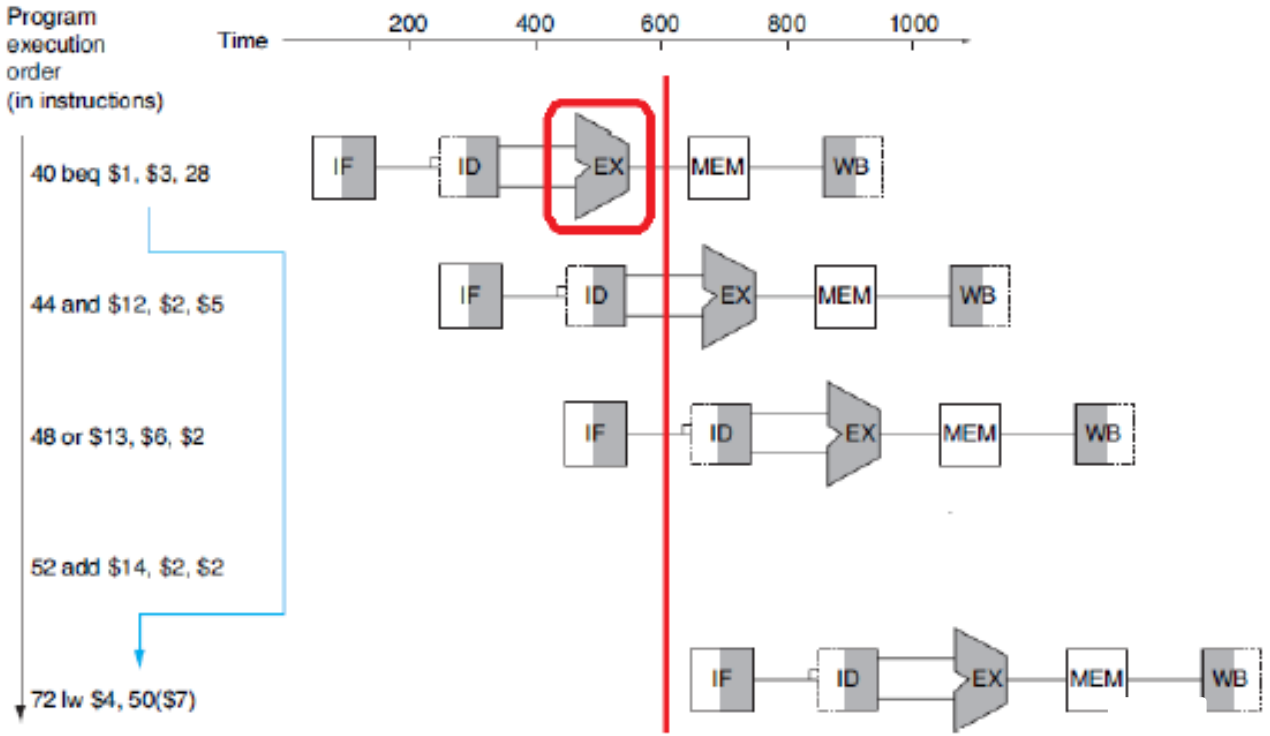
Hazard de control

- *Întrebare:* Unde apare hazardul de control?

```
beq $1, $3, 28
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
lw $4, 50($7)
```

- *Răspuns:* Decizia din instr.1 (salt sau nu) se cunoaşte abia după etapa EX, timp în care se pot încărca în pipeline următoarele instrucţiuni pentru a fi executate

(am presupus salt)



Hazard de control

- O metoda de rezolvare a hazardului de control este *tehnica de întârziere*): se întârzie până când se cunoaşte dacă se realizează salt sau nu, altfel se introduc întârzieri suplimentare (*nop* = no operation sau *bubble* = pipeline stall)

- Pentru a nu aştepta până la aflarea deciziei, se poate utiliza *tehnica de predicţie*: se presupune un anumit rezultat al deciziei şi se continua cu încărcarea în pipeline a instrucţiunii respective

- Se poate lua o decizie statică (ex. pentru un loop mereu se va considera întoarcerea în buclă pentru ca are probabilitate mai mare) sau dinamică (prin contorizarea frecvenţei de apariţie a fiecărei ramuri)

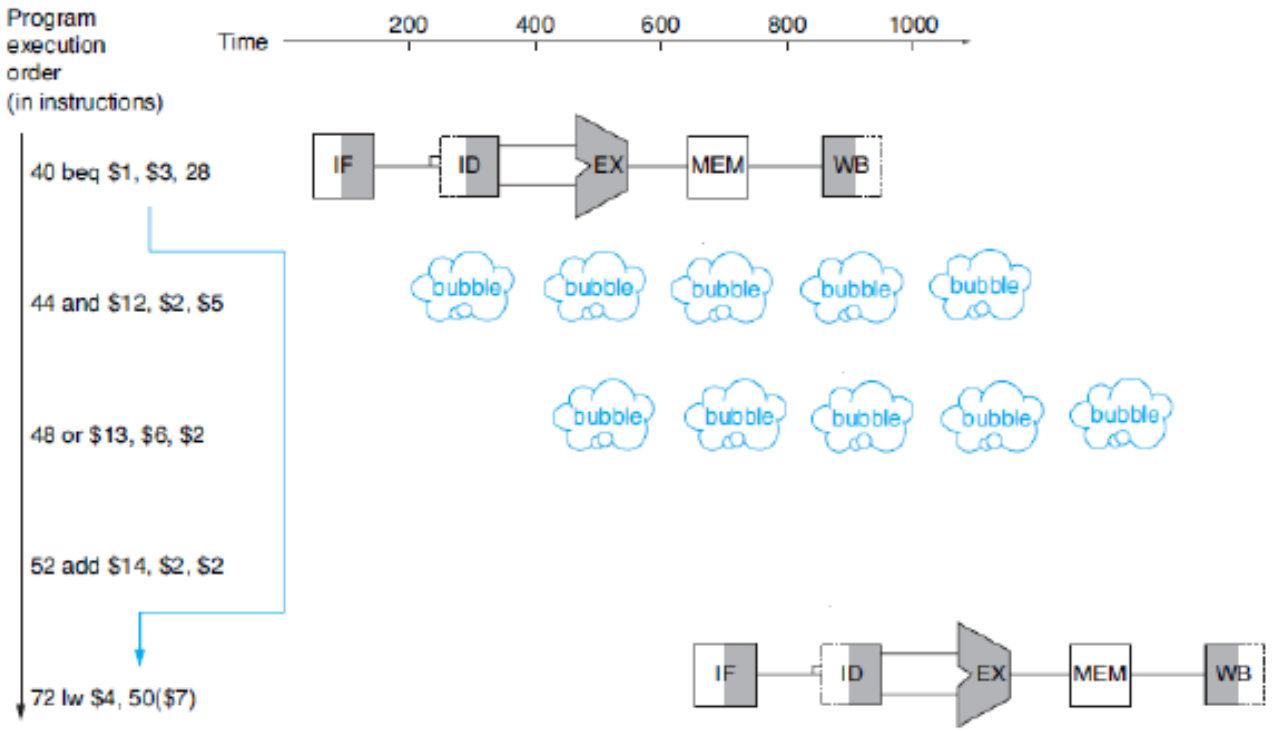
Hazard de control

- *Întrebare:* Cum poate rezolva tehnica de întârziere hazardul de control?

```
beq $1, $3, 28
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
lw $4, 50($7)
```

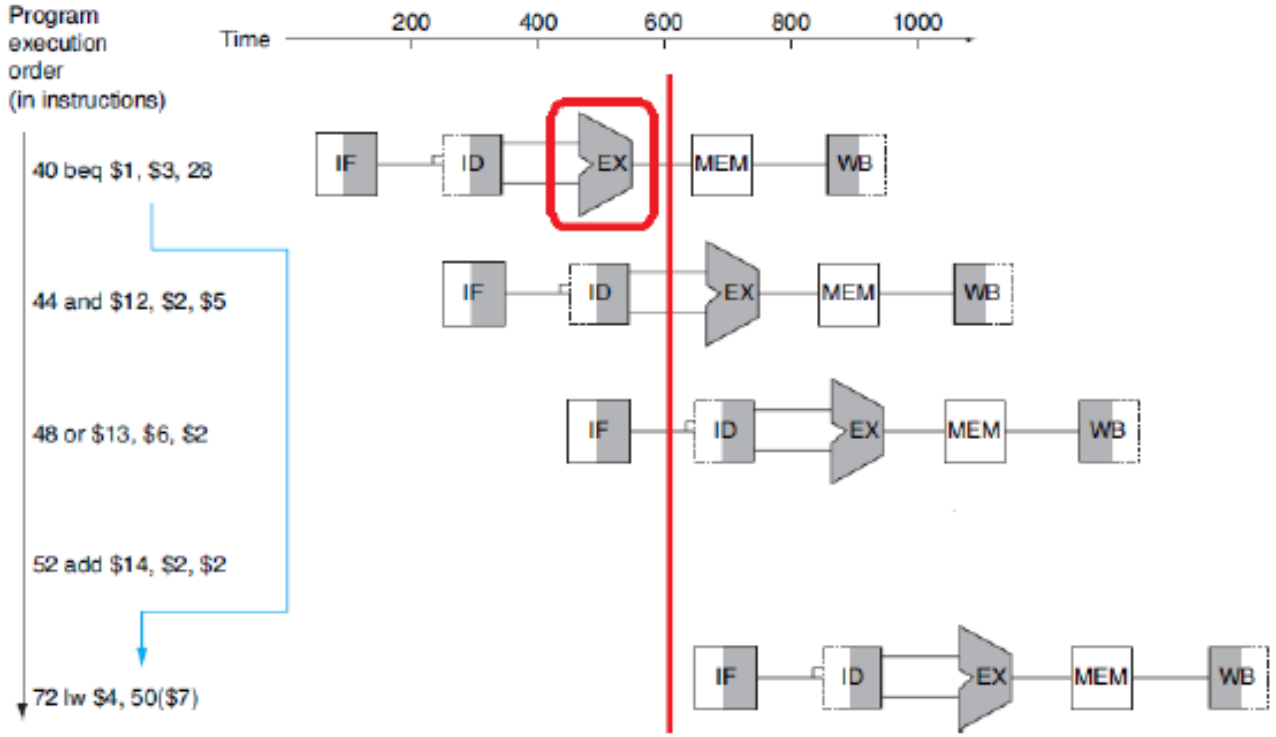
- *Răspuns:*

Se întârzie încărcarea instrucţiunilor până se cunoaşte rezultatul deciziei (am presupus salt)



Hazard de control

- Am folosit deja *tehnica de predicţie*, considerând că nu s-a realizat saltul (predicţie eronată în cazul ilustrat; dar dacă saltul nu se realiza, i.e. predicţia era corectă, nu mai apărea întârzierea de 2 tacturi introdusă prin tehnica întârzierii):



Hazard de control

- Ca să se evite execuţia parţială a instrucţiunilor în caz de predicţii eronate sau întârzierea prin adăugarea bubble / nop, se poate folosi *reordonarea codului*: până se cunoaşte decizia saltului se execută instrucţiuni care se executau indiferent de decizie (situate spre exemplu înainte de instrucţiunea branch)

- Tehnica *delayed branch* este utilizată de procesoarele MIPS: se execută întotdeauna instrucţiunea următoare instrucţiunii condiţionate;

- Pentru programator, acest lucru este ascuns, pentru că se reordonează codul şi se obţine acelaşi rezultat.

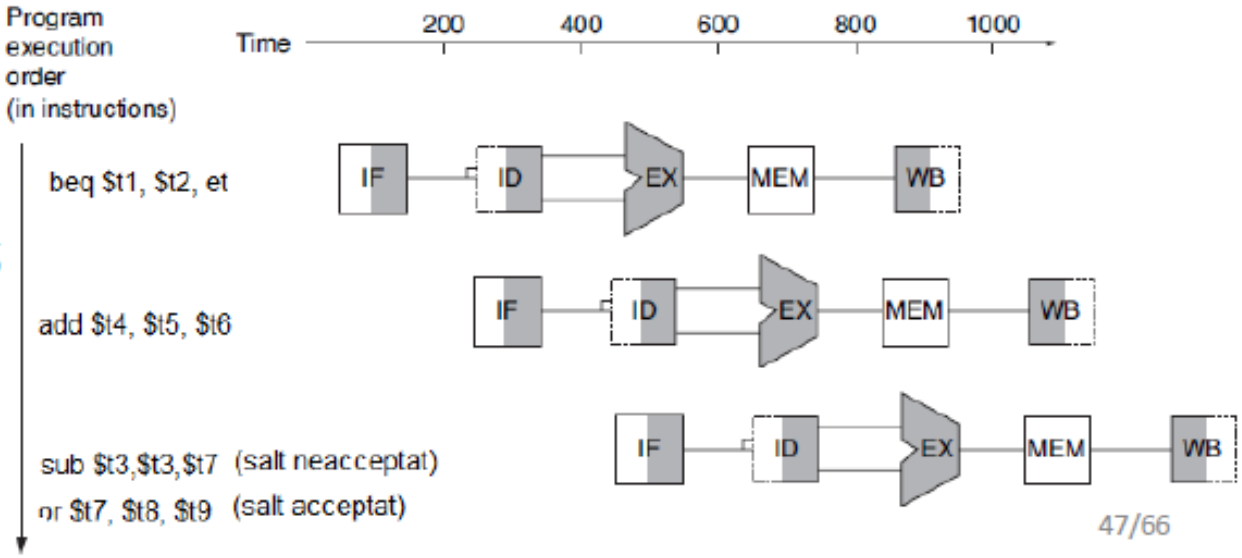
Hazard de control

- *Întrebare:* Cum se execută pe un procesor MIPS secvenţa de instrucţiuni?

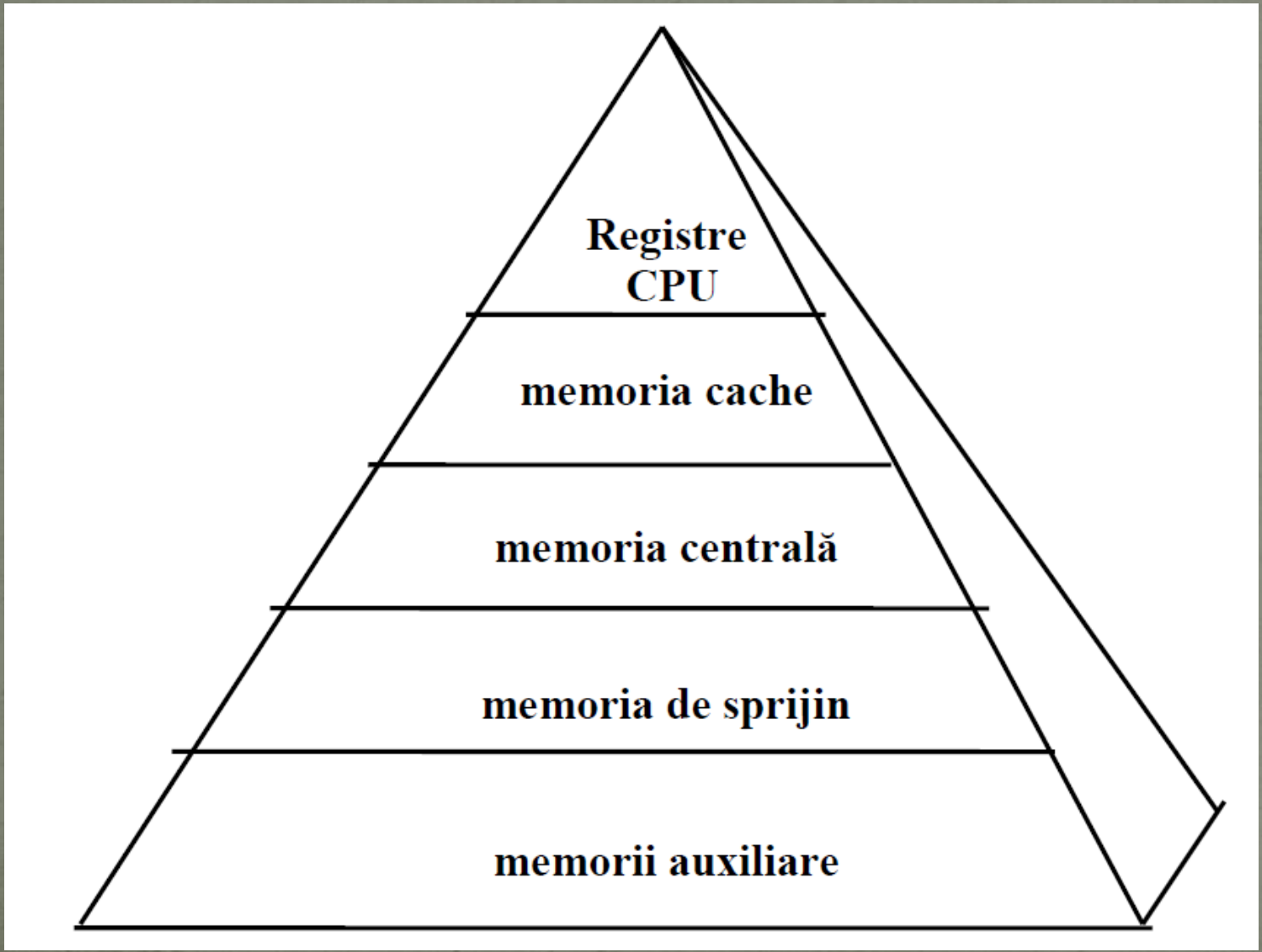
```
add $t4, $t5, $t6
beq $t1, $t2, et
sub $t3, $t3, $t7
et:
or $t7, $t8, $t9
```

- *Răspuns:* Instrucţiunea `add` se execută imediat după `beq` (indiferent de rezultatul condiţiei). Am ţinut cont că se încarcă instrucţiunea corespunzătoare în acelaşi tact de ceas cu determinarea condiţiei (tactul 3).

```
beq $t1,$t2,et
add $t4,$t5,$t6
sub $t3,$t3,$t7
et:
or $t7,$t8,$t9
```



Memorii: Ierarhia memoriilor



Memorii:

- O **memorie** este un dispozitiv capabil sa inregistreze sa conserve si sa restitue informatii.
- Se constata ca pe masura ce ne indepartam de CPU, timpul de acces si capacitatea memoriei cresc, in timp ce costul pe bit se diminueaza.
- **Registrele** sunt elemente de memorie situate in CPU si sunt caracterizate printr-o mare viteza, servind in principal stocarii operanzilor si a rezultatelor intermediare.
- **Memoria cache**, este o memorie rapida, de capacitate redusa (in raport cu memoria centrala) utilizata ca memorie intermediara intre CPU si memoria centrala. Aceasta memorie permite minimizarea numarului de accese la memoria centrala, realizand astfel castig considerabil de timp.

Memorii:

- **Memoria de sprijin (zona tampon)** serveste drept memorie intermediara intre memorie centrala si memoriile auxiliare (externe).

- Clasificare:
  - memorii interne
  - memorii externe

Memoria interna

Memoria internă reprezintă cea mai costisitoare şi importantă componentă fizică a unui calculator personal, prin intermediul căreia vom putea aprecia performanţele unui calculator. Aceasta este unitatea funcţională a calculatorului destinată păstrării permanente sau temporare a programelor şi a datelor necesare utilizatorului şi bineînţeles a sistemului de operare.
Memoria internă a unui calculator este caracterizată de doi parametri:

1. dimensiunea;
2. timpul maxim de răspuns;

Dimensiunea acestei memorii este în strânsă legătură cu microprocesorul folosit (în speţă cu limitările impuse de acesta). O valoare des întâlnită pentru această mărime este de 1 Mbyte. Cu cât aceasta este mai mare, cu atât performanţele calculatorului sunt mai bune.

Timpul maxim de răspuns se referă la intervalul de timp care este necesar memoriei interne pentru a citi sau scrie date. Mai exact, intervalul de timp ce se scurge din momentul în care primeşte de la microprocesor comanda de citire şi momentul în care depune pe magistrala de date valoarea citită (similar este şi pentru scriere). Valoarea medie a acestui parametru este de 70 ns. Cu cât această valoare este mai mică, cu atât calculatorul este mai rapid.

În configuraţia unui sistem electronic de calcul în funcţie de modul în care se realizează accesul la memorie, pot fi întâlnite simultan două mari tipuri de memorii: memorii ROM şi memorii RAM.

Memoria ROM (Read Only Memory – memorie care poate fi doar citită) – este un tip de memorie nevolatilă (informaţia conţinută de acest tip de memorie nu se pierde la oprirea calculatorului). Este o memorie de tip special, care prin construcţie nu permite programatorilor decât citirea unor informaţii înscrise aici de constructorul calculatorului prin tehnici speciale. Memoriile de tip ROM se clasifică la în funcţie de modalitatea de scriere a datelor în PROM şi EPROM.

1. memorii PROM (Programabile ROM), memorii ROM programabile, care permit o singură rescriere de programe;
2. memorii EPROM (Programabile Electric PROM), care pot fi şterse şi reprogramate din nou de mai multe ori, utilizând tehnici electronice speciale.

Programele aflate în ROM sunt livrate odată cu calculatorul şi alcătuiesc aşa numitul firmware. Calculatoarele din familia IBM – PC conţin şi o memorie CMOS (de tip RAM, alimentată în permanenţă de o baterie pentru a nu-şi pierde conţinutul informaţional. În această memorie se stochează informaţii referitoare la configuraţia hardware a sistemului electronic de calcul.

Memoria RAM reprezintă un spaţiu temporar de lucru unde se păstrează datele şi programele pe toată durata execuţiei lor. Programele şi datele se vor pierde din memoria RAM, după ce calculatorul va fi închis, deoarece aceasta este volatilă, păstrând informaţia doar atâta timp cât calculatorul este sub tensiune.

În funcţie de circuitele din care sunt implementate memoriile RAM acestea se clasifică în: memorii statice (SRAM) şi memorii dinamice (DRAM). La rândul său memoriile DRAM se împart în:

1. memorii FPM (Fast Page Mode) – caracteristica acestui tip de memorie o reprezintă facilitatea de a lucra cu pagini de memorie. O pagină de memorie este o secţiune de memorie, disponibilă prin selectarea unei adrese de rând.

2. memorii EDO (extended Data Out) – funcţionează la fel ca şi memoriile FPM dar accesul la datele din celulele de memorie este mai rapid cu 10 – 15 % faţă de FPM

3. memorii SDRAM (Syncronous DRAM) – un astfel de tip de memorie reprezintă un modul DRAM ce lucrează în mod sincron cu procesorul (prin construcţie, la origini memoriile DRAM convenţionale funcţionau în mod asincron)

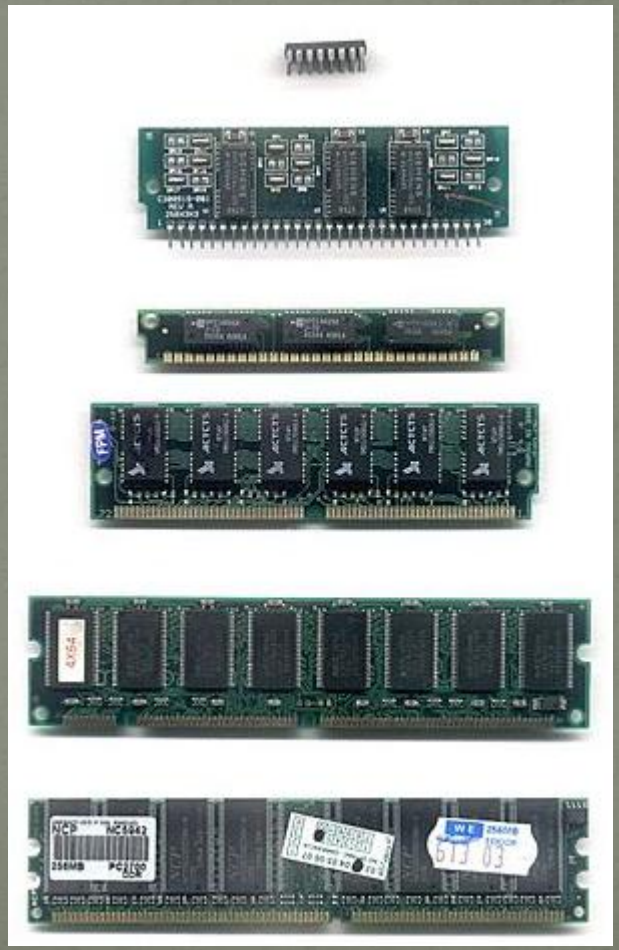
4. memoriile VRAM (Video RAM) – este o memorie rapidă folosită în special pentru plăcile video.

5. memorii SGRAM (Syncronuos Graphics RAM)- este un SDRAM adaptat cerinţelor foarte mari din domeniul graficii 3D.

6. memorii DDR (Double Data Rate)- prin această tehnologie se pot transfera date de două ori mai rapid faţă de tehnologiile anterioare.

Fizic memoria RAM este constituită din elemente care prezintă două stări stabile, reprezentate convenţional prin simbolurile 0 şi 1 denumite biţi sau cifre binare. Aceste elemente sunt constituite din milioane de perechi de tranzistori şi condensatori. Rolul condensatorilor este de a reţine sarcină electrică, iar al tranzistorului acela de a încărca cu sarcină electrică condensatorul. Aceste perechi de condensatori şi tranzistori sunt dispuse sub formă de coloane şi rânduri formând o matrice. Prin construcţie, accesul la memorie se realizează la nivelul unui grup de biţi denumit celulă sau locaţie de memorie. Fiecărei locaţii de memorie îi este asociată o adresă, care identifică în mod unic aceea locaţie. Numărul de biţi care se poate memora într-o locaţie de memorie reprezintă lungimea cuvântului de memorie. Numărul total de locaţii de memorie reprezintă capacitatea memoriei şi se exprimă de regulă în octeţi.

O altă caracteristică a memoriei RAM o reprezintă timpul de acces la informaţie care se defineşte prin intervalul de timp scurs dintre momentul furnizării adresei de către procesor şi momentul obţinerii informaţiei. Timpul de acces la informaţie la memoriile noi este de ordinul nanosecundelor



Memorie RAM	Memorie ROM
poate fi citită și scrisă	poate fi scrisă și, doar dacă vorbim despre EPROM, reprogramată
viteze mai mari față de ROM	viteze mai mici decât RAM
este mai scumpă	este mai ieftină
dimensiuni mai mari	dimensiuni mult mai mici

Memoria externa:

• Este o memorie suplimentara care comunica cu microprocesorul tot prin intermediul magistralei de date si magistralei de comenzi. Este o memorie nevolatila din care se poate citi si in care se poate scrie.

• Memoria externa are deobicei o capacitate mult superioara celei interne, in care se pot inmagazina mai multe programe si datele corespunzatoare.

• Este alcatuita in principal din discuri fixe (hard-disk) si flexibile (CD, DVD. Discurile fixe sunt montate de obicei in interiorul unitatii centrale si nu pot fi detasate de calculator decat prin demontatarea acesteia.

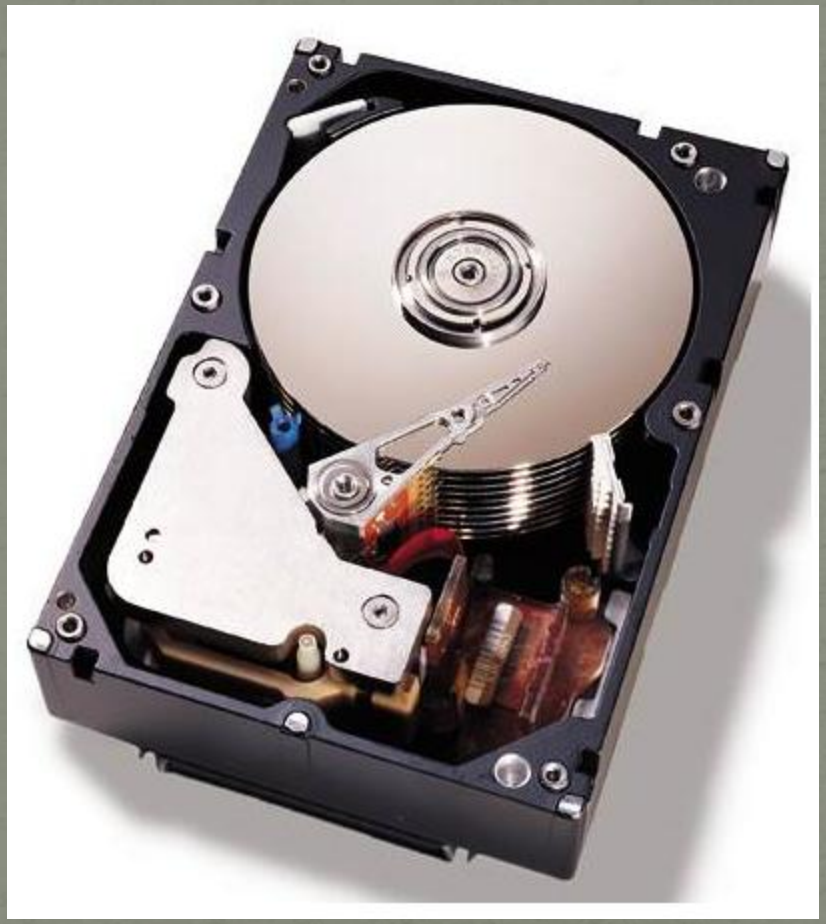
Discurile flexibile

HARD-DISCUL (HD) reprezintă o unitate fixă de stocare a datelor. Acesta este încorporat în cutia care conţine unitatea centrală, încasetat într-un dispozitiv la care nu avem acces pentru a-l înlocui cu altul. În caz de defectare se înlocuieşte întreg ansamblul. Acest ansamblu se mai numeşte disc fix sau disc Winchester, după numele tehnologiei de construcţie. Denumirea de disc fix, atribuită iniţial, a avut în vedere faptul că acesta se fixează în interiorul calculatorului şi nu poate fi detaşat cu uşurinţă de către un utilizator obişnuit. În ultimul timp însă, această denumire a devenit improprie, deoarece au fost create şi HD care pot fi cu uşurinţă conectate şi deconectate în exteriorul calculatorului prin porturile de intrare/ieşire ale acestuia.

În funcţie de interfaţa de conectare hard discurile se clasifică în:

1. Hard discuri **SCSI (Small Computer System Interface)** – hard discuri având caracteristici deosebite fiind conectate la o interfaţă SCSI, interfaţă ce este controlată de sisteme inteligente (controlere) acestea având menirea de a coordona fluxul de informaţii dintre hard disc şi sistem. Acest tip de unităţi de stocare se folosesc cu precădere montate pe servere sau pe acele calculatoare unde se doreşte o performanţă ridicată privind transferul de date.

2. Hard discuri **EIDE (Enhanced Integrated Drive Electronics)** – termen general aplicat tuturor unităţilor care au un controler inclus în unitate. De-a lungul timpului unităţile de stocare de acest gen au cunoscut o serie de implementări printre care amintim protocolul Ultra ATA care mai este denumit şi Ultra DMA/ ATA-33/ DMA-33, Ultra ATA 66, Ultra ATA 100. Aceste denumiri se referă direct la realizarea transferului rapid de date. Legat de hardurile EIDE în ultimul timp şi-au făcut apariţia pe piaţă cele SATA (Serial ATA), hard discuri ce reuşesc să obţină o viteză de transfer de 150 M/s.



Principalele caracteristici ale HD se referă la:

I. capacitatea de stocare a informaţiilor/capacitatea de manipulare a datelor de către PC (PC Data Handling);

II. timpul de căutare (seek time) - este o măsură exprimată în milisecunde a rapidităţii cu care hard discul îşi poate deplasa capetele de scriere citire de la o locaţie la alta. Întârzierea produsă de rotaţie reprezintă timpul necesar pentru ca sectorul dorit să ajungă în dreptul capului de scriere/citire, odată ce capul s-a poziţionat pe pista respectivă.

III. rata de transfer a sistemului gazdă – este reprezentată de cantitatea de date ce poate fi trensferată prin magistralele de date ale sistemului;

IV. rata de transfer a hard-discului (media rate) - reprezintă viteza cu care datele sunt transferate spre şi dinspre platan. Unitatea uzuală de măsură a acestei caracteristici este numărul de biţi pe secundă. Parametrul care influenţează rata de transfer pe lângă viteza de rotaţie este dat şi de densitatea datelor pe platan exprimată fie prin număr de piste / inch fie prin cantitate de biţi / inch.

V. numărul de rotaţii/minut (rpm) -reprezintă viteza de rotaţie a discului. Particularitatea acestui parametru o reprezintă faptul ca această viteză este constantă. Cu cât această viteză este mai mică cu atât întârzierile datorate poziţionării mecanismelor fizice sunt mai mari având un impact direct asupra aşteptării generate de mişcarea de rotaţie şi implicit asupra ratei de transfer a discului;

VI. cantitatea de memorie cache – influenţează în mod direct performanţele hard discului, reducând timpii de aşteptare.

HDD (Hard Disk Drive) vs SSD (Solid State Drive)

Durata de viață

Una dintre cele mai notabile diferențe dintre HDD-uri și SSD-uri este durata de viață, generată de modul în care funcţionează dispozitivele. SSD-urile nu au componente în mişcare care să cedeze din punct de vedere mecanic, însă fiecare bloc al memoriei flash are un număr limitat de cicluri de scriere. Totuşi, în general, durata de viaţă a SSD-urilor este mai mare decât cea a HDD-urilor.

Atenţie însă la penele de curent: SSD-urile rezistă semnificativ mai puţin la întreruperile bruşte de energie electrică comparativ cu clasicele HDD-uri.

Viteză

Dacă sunteți în căutarea unui dispozitiv de stocare mai rapid, atunci opțiunea potrivită este un tot SSD. Acest dispozitiv de stocare a informației sunt mult mai rapide pentru că dispun de procesor . De asemenea, pentru HHD-uri este necesară sincronizarea unui cap de citire/scriere cu un plantan rotativ.

În prezent, vitezele de transfer ale SSD-urilor variază semnificativ de la un producător la altul şi de la un produs la altul, însă se încadrează în general intre 100-600 MB/s. În schimb, viteza unui HDD ajunge, teoretic, până la circa 140 MB/s, viteza reală fiind însă semnificativ mai mică în cele mai multe situaţii.

HDD vs SSD

Rezistență

Și la capitolul rezistență SSD-urile stau mai bine. Aceste dispozitive sunt mai rezistente la accidente, spre exemplu, pentru că nu dispun de părți mobile, fiind bazate în principal pe memorie flash sau DRAM. De asemenea, acestea rezistă mai bine și la fluctuațiile de temperatură decât HDD-urile.

HDD vs SSD

Alte caracteristici

În ceea ce privește alte diferențe dintre aceste două dispozitive, ar trebui menționat faptul că SSD-urile consumă mai puțină energie decât HDD-urile. SSD-urile sunt mai silențioase decât HDD-urile și nu se încălzesc atât de mult ca acestea. De asemenea SSD-urile nu produc niciun fel de zgomot in functionare.

În plus, trebuie avut în vedere și faptul că SSD-urile sunt alegerea potrivită pentru stocarea de sisteme de operare și programe, întrucât sunt mai rapide, de vreme ce HDD-urile sunt mai potrivite pentru stocarea de materiale.

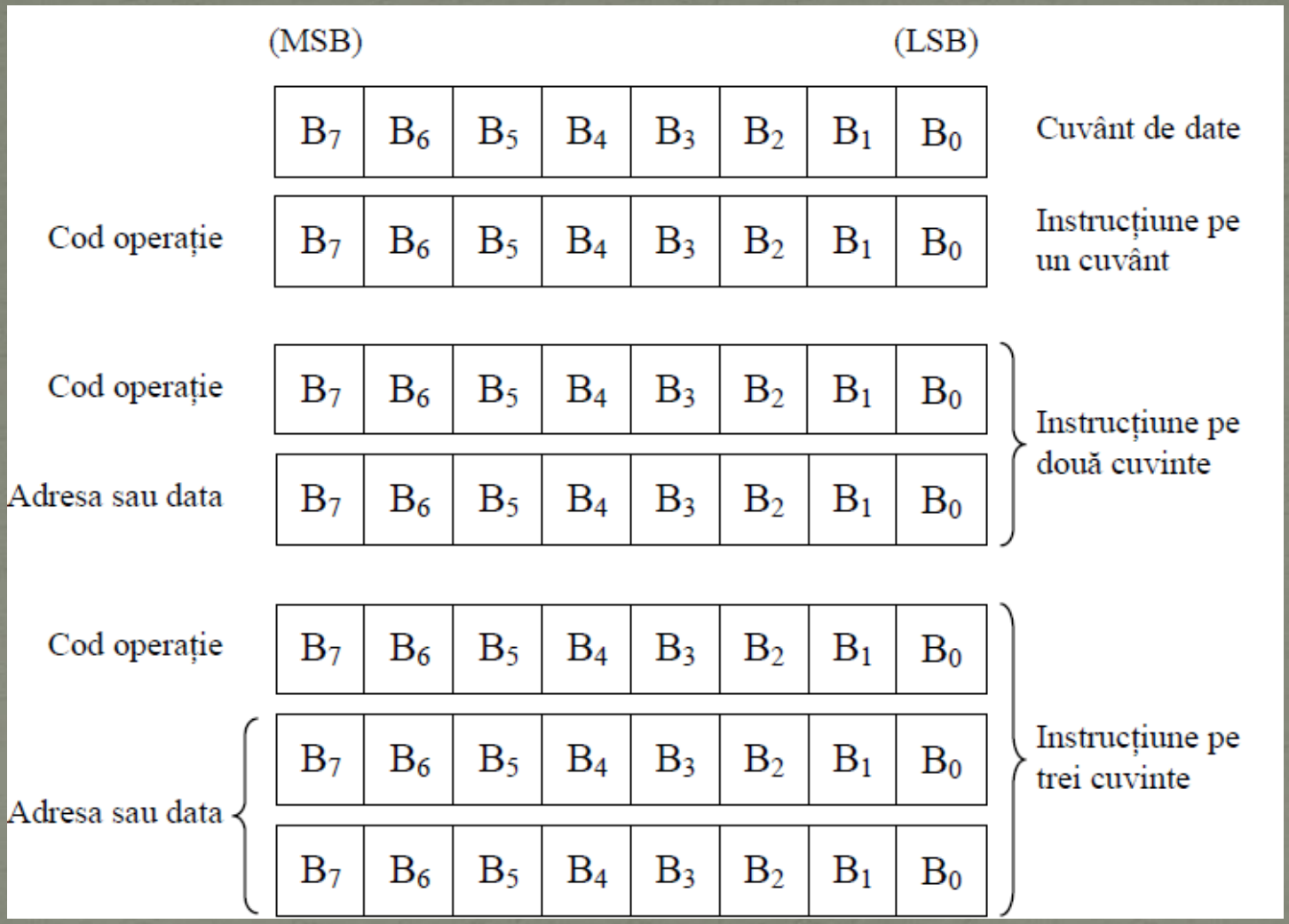
Astfel, dacă ești în căutarea unui dispozitiv pentru stocarea informației digitale de calitate și dispui de bugetul necesar opțiunea potrivită ar fi un SSD. Cu toate acestea, dacă urmărești să achizitionezi un dispozitiv mai ieftin și care oferă mai mult spațiu de stocare atunci ar trebui să alegi un HDD.

Pentru maximum de performanţe, contează foarte mult şi tipul de conexiune suportată de SSD-ul cumpărat. În prezent, cele mai multe SSD-uri se conectează prin SATA3, iar noi îţi recomandăm să eviţi achiziţionarea unor dispozitive care se conectează prin cabluri SATA2 sau, şi mai rău, SATA, întrucât aceste interfeţe sunt mai lente. Desigur, pentru a beneficia de viteză superioară, este necesar ca şi placa de bază a computerului să suporte aceeaşi tehnologie (cu alte cuvinte, degeaga cumperi un SSD SATA3 dacă ai un computer vechi care abia suporta SATA).

- Structura uzuală a memoriei unui SCTR cuprinde atât memorii RAM cât si memorii ROM. Organizarea si dimensiunile acestora depind de aplicatia în care se utilizează sistemul.
- Zonele de adrese ocupate de memoria RAM si ROM depind de proiectantul sistemului, de obicei, memoria ROM conținând adresa la care trebuie să se găsească prima instructiune din program, la care procesorul face acces după primirea unui semnal de RESET.

- Fie de exemplu un procesor de 8 biti avand in codul operatiei o instructiune cu trei octeti, în care octetii 2 si 3 contin o adresă din memorie, al cărei continut va fi adus în acumulator, succesiunea de operatii va fi următoarea:

- se citeste primul octet al instructiunii;
- se decodifică codul operatiei;
- conform codului, se mai citesc doi octeti suplimentari (la fiecare citire, PC – program counter - este incrementat) (instructiunea are cuvintele unul după altul în memoria program);
- se configurează pe magistrala de adrese adresa desemnată de către cei doi octeti suplimentari cititi si se citeste această celulă de memorie, continutul ei fiind transferat în acumulator;
- se trece la citirea si executarea altei instructiuni.



Structura instructiunilor unui procesor

Unitati de intrare si iesire

- Informatiile (date sau instructiuni) circula pe caile de sistem format din busul de date si busul de adrese si sunt gestionate de unitatile de intrare/iesire.

- Gestionarea marimilor de intrare/iesire:
  - Prin intrari/iesiri programabile
  - Prin intrari/iesiri mapate in memorie
  - Prin metoda de acces direct la memorie sau DMA (Direct Memory Acces)

Unitati de intrare si iesire

- **In modul de lucru cu intrari/iesiri programabile** se utilizeaza instructiuni speciale din setul de instructiuni al procesorului pentru a transfera informatii intre CPU si periferice sau CPU si memorii.
- IN, OUT - in care dispozitivul periferic este recunoscut ca o adresa de catre procesor iar in CPU adresa este a unui registru implicit.
- Acest mod de lucru utilizeaza capacitatile CPU pentru toate operatiunile de I/E si deci se mareste timpul de raspuns al SC  probleme la unele aplicatii in TR

Unitati de intrare si iesire

- **In modul de lucru cu intrari/iesiri mapate in memorie** utilizeaza un mecanism cu instructiuni LOAD si SAVE.
- Procesorul va impartii memoria cu alte dispozitive ale SC. Anumite locatii de memorie vor fi considerate ca porturi virtuale de I/E.
- SAVE - trimiterea acelei informatii al dispozitivului periferic caruia i s-a alocat acea adresa de memorie.
- Problema apare la viteze diferite de lucru intre procesor si periferic.(Daca perifericul mai lent, atunci trebuiesc generate stari de asteptare pentru procesor)

Unitati de intrare si iesire

- **In modul de lucru DMA** accesul la memorie este dispus de catre un alt dispozitiv.
- Folosirea DMA conduce la creşterea vitezei sistemului prin degrevarea microprocesorului de controlul transferurilor de date, accesul la memorie fiind efectuat de periferic.
- Când ne referim la DMA, ne referim de fapt la transferuri de date şi la controlul acestora între memorie şi porturile I/E. Activitatea este supravegheată în cazul transferurilor de tip DMA de către un circuit specializat numit controller DMA.
- Dacă o setare DMA a fost realizată greşit, de exemplu acelaşi canal DMA a fost alocat pentru mai multe dispozitive, placa nou conectată nu va funcționa sau chiar va bloca sistemul. În funcție de aplicație, se pot folosi unul sau mai multe canale DMA.

Daca mai multe dispozitive doresc sa obtina in acelasi timp accesul la un bus de date se obtine “contencios de bus”.

Daca un dispozitiv are controlul busului iar altul primeste accesul in acelasi timp rezulta “coliziune.”

Datorita timpului de acces redus la memorii si a timpului mai scurt de lucru a procesorului rezulta utilizarea in aplicatii TR.

O parte a memoriei SC este definita ca memorie cu acces DMA.

Oricare din aceste trei metode permite vehicularea datelor prin modul de lucru cu intrerupere.

Intreruperea este un semnal electronic generat de catre o unitate functionala, de exp. canal sau controler de periferice si acest semnal este transmis spre CPU pentru a provoca o ruptura de secventa in vederea executie unui program prioritar care trateaza cauza intreruperii.

Semnale de intreruperi:

- interne: depasire de capacitate, coduri de operatii inexistente, erori de adresare, pana de curent.
- externe: starea unei unitati periferice, sfarsitul unui transfer de date.

Tratarea intreruperii:

- oprirea exectiei programului in curs
- salvarea starii sistemului
- executarea programului de serviciu de intrerupere
- restaurarea starii sistemului
- reluarea executiei programului intrerupt.

- Cauzele intreruperilor sunt afisate in **vector indicatori**, iar programul care trateaza intreruperea trebuie sa testeze acesti indicatori.

Majoritatea sistemelor de calcul moderne sunt prevăzute cu sisteme de întrerupere ierarhizate [priority interrupt systems], acestea fiind sistemele cu nivele de prioritate.

Problemele care trebuiesc rezolvate sunt următoarele:

- sosirea mai multor semnale de întrerupere în timpul execuţiei unei instrucţiuni;
- sosirea unui semnal de întrerupere în timpul execuţiei unui program de prelucrare a unei întreruperi anterioare.

În aceste sisteme, fiecare nivel este asociat unui anumit număr de întreruperi, fiecărui nivel îi corespunde un anumit nivel de prioritate, iar orice program poate fi întrerupt în vederea realizării unei întreruperi mai prioritare.

Un sistem de întrerupere modern trebuie să permită programatorului următoarele acțiuni:

- invalidarea/activarea [disable/enable] sistemului de întrerupere;
- mascarea/demascarea individuală a întreruperilor;
- stabilirea unei ierarhii în mulţimea cauzelor intreruperilor și definirea mai multor nivele de prioritate, de preferinţă dinamic;
- asocierea unui program specific fiecărei întreruperi, permiţând acțiuni elementare realizabile într-o singură instrucţiune;
- posibilitatea de utilizare a tuturor registrelor care caracterizează starea mașinii și refacerea acestora la sfârşitul programului de întrerupere.

Tehnici de comunicare cu dispozitivele de intrare/iesire

- Gestiunea comunicarii aplicatiei timp-real cu dispozitivele I/E presupune utilizarea unor **tehnici de programare specifice**, destinate sa sincronizeze operatiile de I/E cu prelucrarile interne.

- Daca de exemplu datele sunt citite înainte ca un dispozitiv sa le poata furniza, atunci rezultatul este eronat.

- Daca aplicatia nu preia datele suficient de rapid, dispozitivul periferic poate sa le înlocuiasca cu altele noi fara ca acest lucru sa poata fi depistat. Prin urmare, este necesar ca în fluxul prelucrarilor sa existe secvente de program care sa realizeze **sincronizarea** operatiilor de I/E cu programele de aplicatie (detectarea daca un periferic este gata sa furnizeze/primeasca date si sa realizeze transferul acestora).

Programarea dispozitivelor I/E în aplicatii timp - real

- Pentru sincronizarea comunicarii între aplicatie si echipamentele periferice exista 2 tehnici de baza:
  - aplicatia interogheaza periodic dispozitivele (**polling**) citind registrele de stare pentru a determina când se poate comunica cu acesta;
  - dispozitivele periferice întrerup procesorul pentru executia unor proceduri specifice de comunicare.

Metoda Polling

- Procesorul interogheaza dispozitivele periodic, citind unul sau mai multe registre de stare a caror valoare permite procesorului sa decida când dispozitivul este pregatit pentru comunicare. Daca dispozitivul solicita serviciile procesorului, este apelata o rutina de tratare specifica, în caz contrar procesorul fie continua interogarea fie executa alte prelucrari.

- **Bucla de polling** poate sa fie implementata în doua moduri:

    1. Asteptare în bucla de test pâna când dispozitivul este gata si apoi transfer de date.
    2. Daca dispozitivul nu este gata în momentul interogarii se continua cu alte prelucrari (inclusiv interogarea altor dispozitive), iar când este gata se face transferul de date.

Metoda Polling

- structura generală

```
do {
    if (eveniment1) actiune1;
    if (eveniment2) actiune2;
    ....
    if (evenimentN) actiuneN;
} while(1) //bucla infinita
```

- Desi polling este cea mai simpla metoda de comunicare, prezinta unele dezavantaje:

  - Aplicatia trebuie sa fie capabila sa execute întreaga bucla suficient de rapid pentru a putea tine cont de toate cerintele perifericelor.

  - În functie de încarcarea sistemului, o bucla polling poate fi suficient de rapida în anumite conditii de functionare a dispozitivelor, iar în alte conditii poate fi extrem de lenta (de exemplu daca se executa multe sarcini într-un anumit pas al buclei sau daca sunt interogate mai multe dispozitive si toate cer servicii simultan).

- Când complexitatea programului creste ca urmare a introducerii de noi prelucrari, o bucla polling, care la origine lucra bine, poate deveni prea lunga.

- Daca se cere executarea unor actiuni cu o baza de timp prestabilita, o bucla polling nu permite întotdeauna ca aceasta sa fie controlata pentru a asigura precizia adecvata. Daca baza de timp se calculeaza tinând cont de durata de executie a instructiunilor masina, schimbarea tactului procesorului implica modificarea programului.

Metoda întreruperilor externe

- Prin aceasta metoda dispozitivul periferic atentioneaza aplicatia generând întreruperi externe. Tratarea prompta a întreruperilor de la toate dispozitivele este posibila atât timp cât cererile catre procesor sunt rezonabile, acesta este capabil sa lanseze rapid rutinele de tratare iar timpul de executie al acestora este suficient de mic.

- În particular, metoda întreruperilor externe este de preferat în aplicatii care cer precizie pentru timpul de achizitie de date si control, în timp ce procesorul executa si alte sarcini. De asemenea, este utila daca mai multe dispozitive **solicita asincron servicii**, la intervale de timp nepredictibile.

- Intreruperile sunt o buna modalitate de a controla achizitia de date atunci când rata de achizitie este suficient de scazuta si exista timp disponibil suficient între întreruperi.
- De asemenea întreruperile sunt mai avantajoase decât polling-ul daca programul trebuie sa comunice asincron cu mai multe dispozitive sau daca programul trebuie sa execute mai multe task-uri în paralel cu sarcinile de achizitie de date si comenzi.

- Se dau doua sate.
  Primul sat este populat de canibali mincinosi iar al doilea sat este populat de oameni buni si sinceri.
  Se stie deci ca oamenii buni spun adevarul tot timpul iar canibalii mint tot timpul.
  Te afli la intersectia dintre cele doua sate si te intalnesti cu un om din cele doua sate, dar nu sti din ce sat provine el.
  Ai dreptul sa-i pui o singura intrebare si sa afli directia incotro trebuie sa mergi ca sa nu ajungi in satul de canibali.

- In ce directie e satul lui?