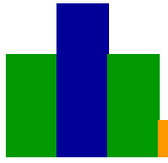


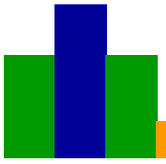
Lecţia 10:

Procesorul:
Tehnica de pipeline - II

G Stefănescu — Universitatea Bucureşti

Arhitectura sistemelor de calcul, Sem.1
Octombrie 2005—Februarie 2006

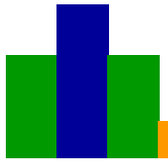
După: D. Patterson and J. Hennessy, Computer Organisation and Design



Procesorul: Tehnica de pipeline

Cuprins:

- Tehnica de pipeline
- Calea de date cu pipeline
- Controlul pentru implementari cu pipeline
- *Hazard de date si avansari*
- Hazard de date si stationari
- Hazard la ramificatii
- Exceptii
- Concluzii, diverse, etc.



Dependenta de date

Dependenta de date vs. pipeline:

- Potenţialul extraordinar oferit de tehnica de pipeline este atenuat de dificultatea de a trata eficient excepţiile produse de diversele tipuri de *hazard*, care întârzie procesarea.
- Aici analizăm hazardul creat de *interdependenţa datelor*.
- In exemplul de mai jos, ultimile 4 instrucţiuni sunt depedente de prima prin intermediul registrului $2.

```
sub $2,$1,$3;          (1)
and $12,$2,$5;         (2)
or  $13,$6,$2;         (3)
add $14,$2,$2;         (4)
sw  $15,100($2);       (5)
```

..Dependenta de date

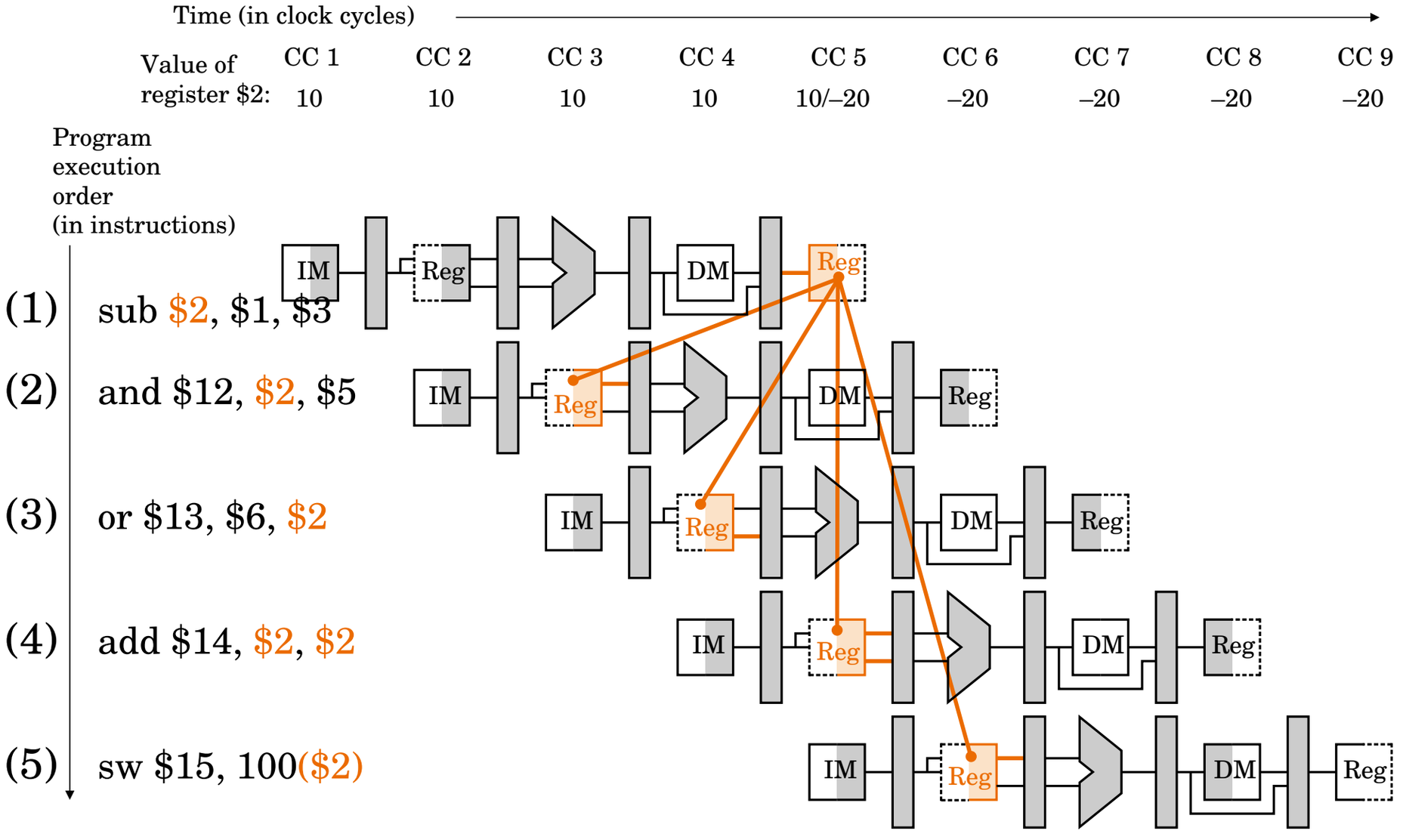
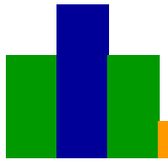


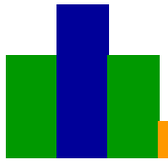
Figura ilustrează *dependenţele de date* între 5 instrucţiuni procesate cu *pipeline*; ilustrarea este schematică spre a reliefa dependenţele.



..Dependenta de date

Dependenta de date vs. pipeline:

- Dependenţa de date este ilustrată în figura de mai sus.
- Figura descrie cazul când `$2` conţine valoarea `10` înainte de scădere şi `-20` după.
- Care sunt hazardurile care apar?
  - pentru perechile *(1,2)* şi *(1,3) avem* hazarduri;
  - pentru perechea *(1,4) nu avem hazard*, dacă se aplică convenţia de a avea scriere în prima jumătate de ceas şi citire în a doua;
  - pentru perechea *(1,5) nu avem* hazard.



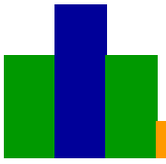
..Dependenta de date

Interfata hardware-software:

- O soluţie facilă la hazardurile de mai sus este de a folosi o instrucţiune specială `nop` care nu face nimic.
- Compilatorul poate genera secvenţa echivalentă

```
sub $2,$1,$3;                (1)
nop
nop
and $12,$2,$5;               (2)
or  $13,$6,$2;               (3)
add $14,$2,$2;               (4)
sw  $15,100($2);             (5)
```

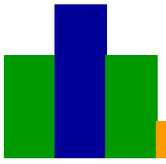
  care este corectă, dar inserează o întârziere de 2 cicluri.
- Rămân de căutat soluţii mai performante.



Avansarea

Avansarea:

- Putem folosi tehnica de *avansare (forwarding)* pentru a soluţina hazardurile de date.
- *Sursa:* Noile valori produse de instrucţiuni pentru registrul destinaţie pot proveni:
  - din ALU (instrucţiune de format-R) şi, înainte de finalizare, se găsesc în regiştrii pipeline `EX/MEM` şi `MEM/WB`;
  - din memorie (instrucţiune load) şi, înainte de finalizare, se găsesc în registrul pipeline `MEM/WB`.
- *Destinaţia:* Pentru un hazard soluţionat prin avansare, utilizarea acestor valori se face
  - în faza de execuţie, folosindu-se pentru unul din argumentele ALU.



..Avansarea

Avansarea (cont.)

- Tipurile de *hazard* care pot fi *rezolvate* cu tehnica de avansare se pot clasifica astfel:
  - *1a:* `EX/MEM.RegisterRd = ID/EX.RegisterRs`
  - *1b:* `EX/MEM.RegisterRd = ID/EX.RegisterRt`
  - *2a:* `MEM/WB.RegisterRd = ID/EX.RegisterRs`
  - *2b:* `MEM/WB.RegisterRd = ID/EX.RegisterRt`

  Pe scurt, regiştri `rs,rt` pentru ALU provin din `EX/MEM` la tipul 1 şi din `MEM/WB` la tipul 2.

- In exemplul anterior, perechea (1,2) este hazard de tip 1a, iar perechea (1,3) este de tip 2b.

..Avansarea

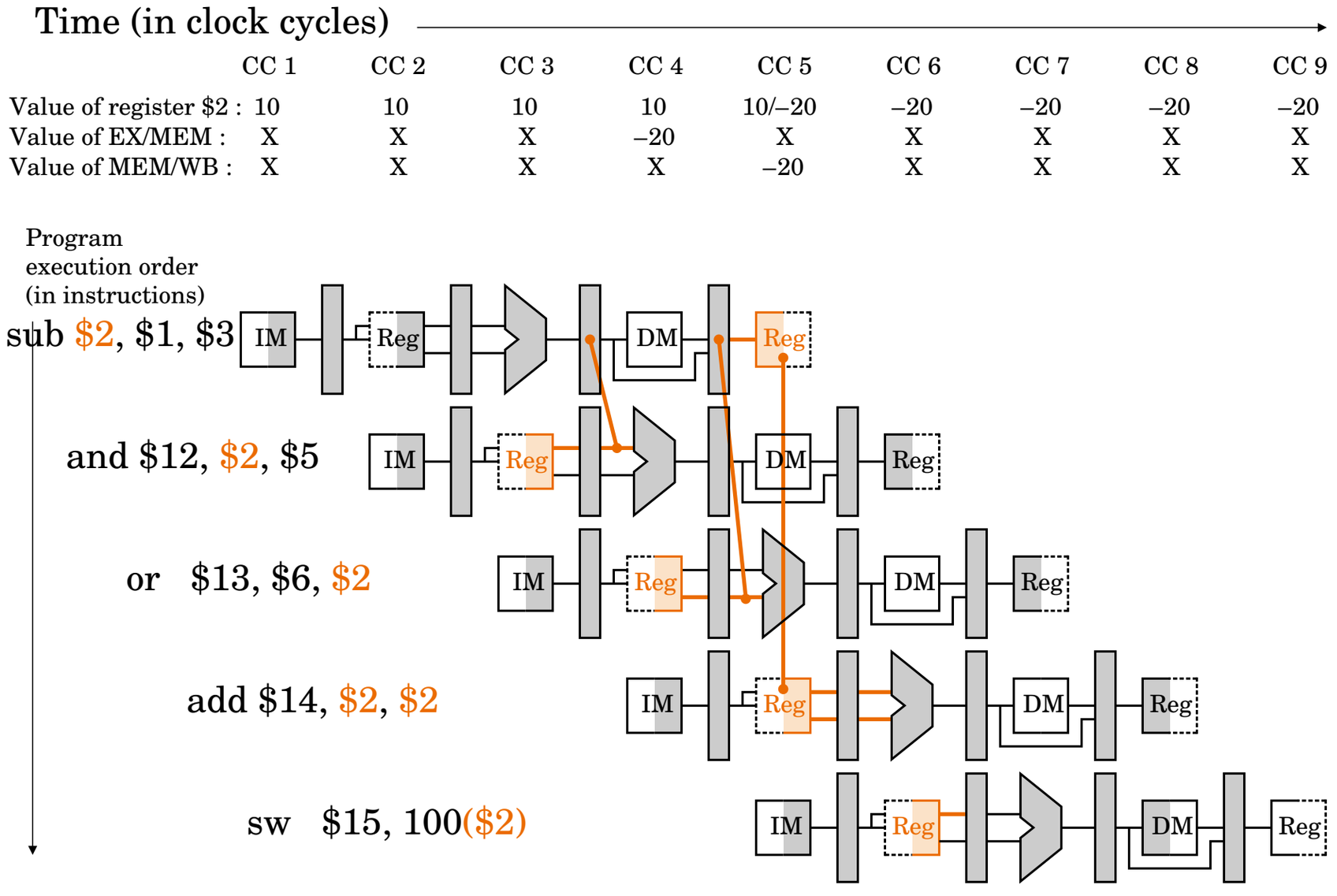
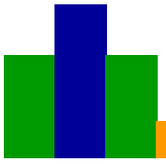


Figura ilustrează tehnica de *avansare* pentru hazardurile de date: luăm datele necesare *înainte de finalizarea* procesării instrucţiunii.



..Avansarea

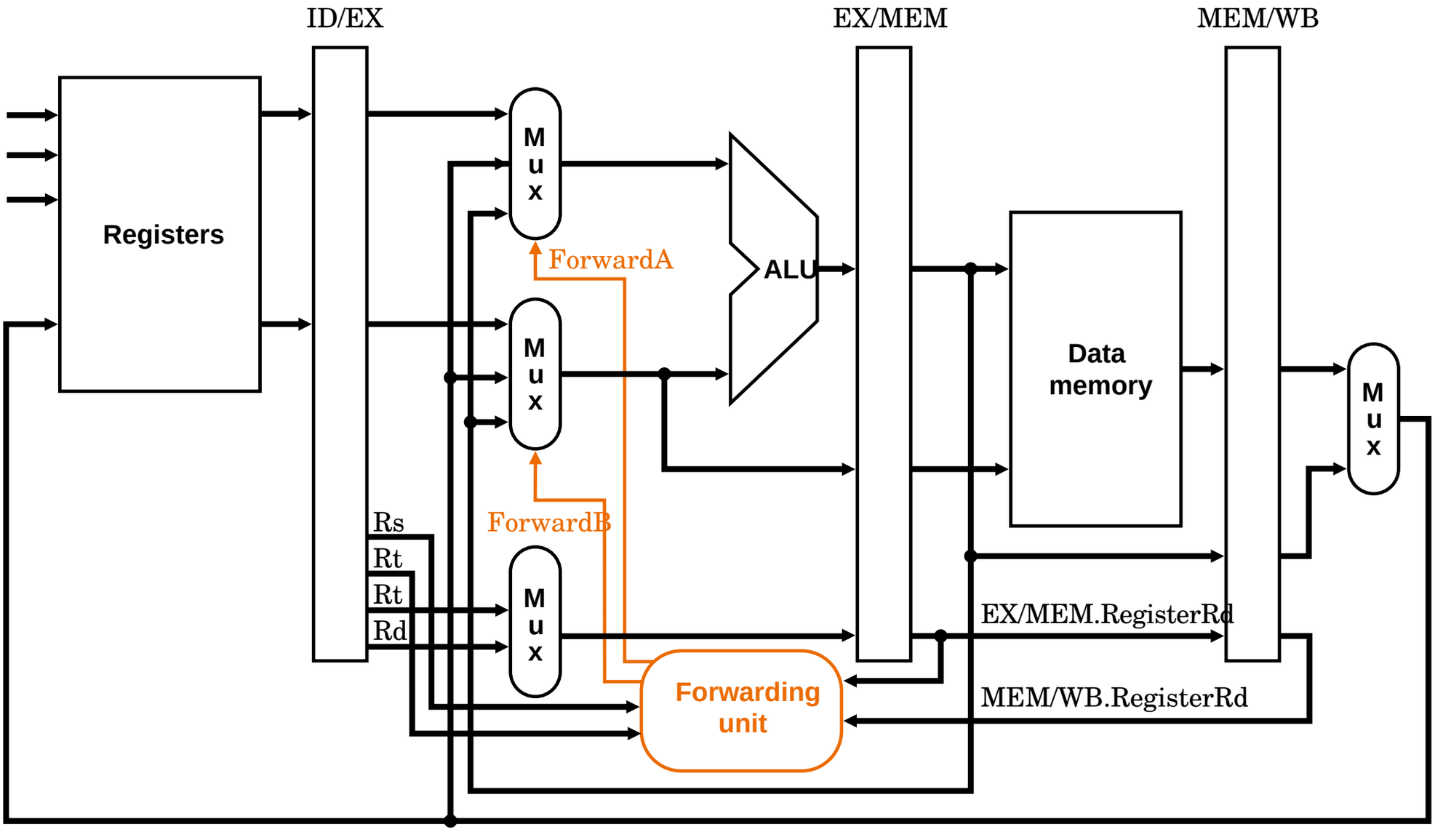
Avansarea (cont.)

Condiţii pentru *detectarea hazardului*:

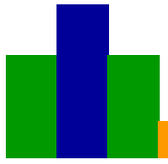
- Pentru un hazard de *tip 1*, condiţiile sunt:
  - semnalul `EX/MEM.RegWrite` este setat (i.e., instrucţiunea curentă scrie într-un registru);
  - `EX/MEM.RegisterRd` $\neq$ `0` (i.e., registrul destinaţie pentru avansare nu este registrul 0, care nu poate fi modificat);
- Similar pentru un hazard de *tip 2*, dar folosim registrul pipeline `MEM/WB`.

Condiţii pentru *avansarea datelor*

- Folosim 2 semnale noi de control pentru avansarea datelor `ForwardA` şi `ForwardB` - vezi figura.



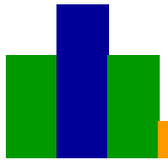
In roşu sunt prezentate noile *semnalele de control* necesare pentru rezolvarea hazardului de date cu tehnica de *avansare* (pe o cale de date simplificată).



..Dependenta de date

Condiţiile de *detectare hazard* şi rezolvare cu *avansări*:

- *Tip 1: în faza* EX (execuţie):
  - $\texttt{EX/MEM.RegWrite} \wedge \texttt{EX/MEM.RegisterRd} \neq 0 \wedge \texttt{EX/MEM.RegisterRd} = \texttt{ID/EX.RegisterRs} \Rightarrow \texttt{ForwardA} = 10$
  - $\texttt{EX/MEM.RegWrite} \wedge \texttt{EX/MEM.RegisterRd} \neq 0 \wedge \texttt{EX/MEM.RegisterRd} = \texttt{ID/EX.RegisterRt} \Rightarrow \texttt{ForwardB} = 10$
- *Tip 2: în faza* MEM (access memorie) - *provizoriu*:
  - $\texttt{MEM/WB.RegWrite} \wedge \texttt{MEM/WB.RegisterRd} \neq 0 \wedge \texttt{MEM/WB.RegisterRd} = \texttt{ID/EX.RegisterRs} \Rightarrow \texttt{ForwardA} = 01$
  - $\texttt{MEM/WB.RegWrite} \wedge \texttt{MEM/WB.RegisterRd} \neq 0 \wedge \texttt{MEM/WB.RegisterRd} = \texttt{ID/EX.RegisterRt} \Rightarrow \texttt{ForwardB} = 01$



..Dependenta de date

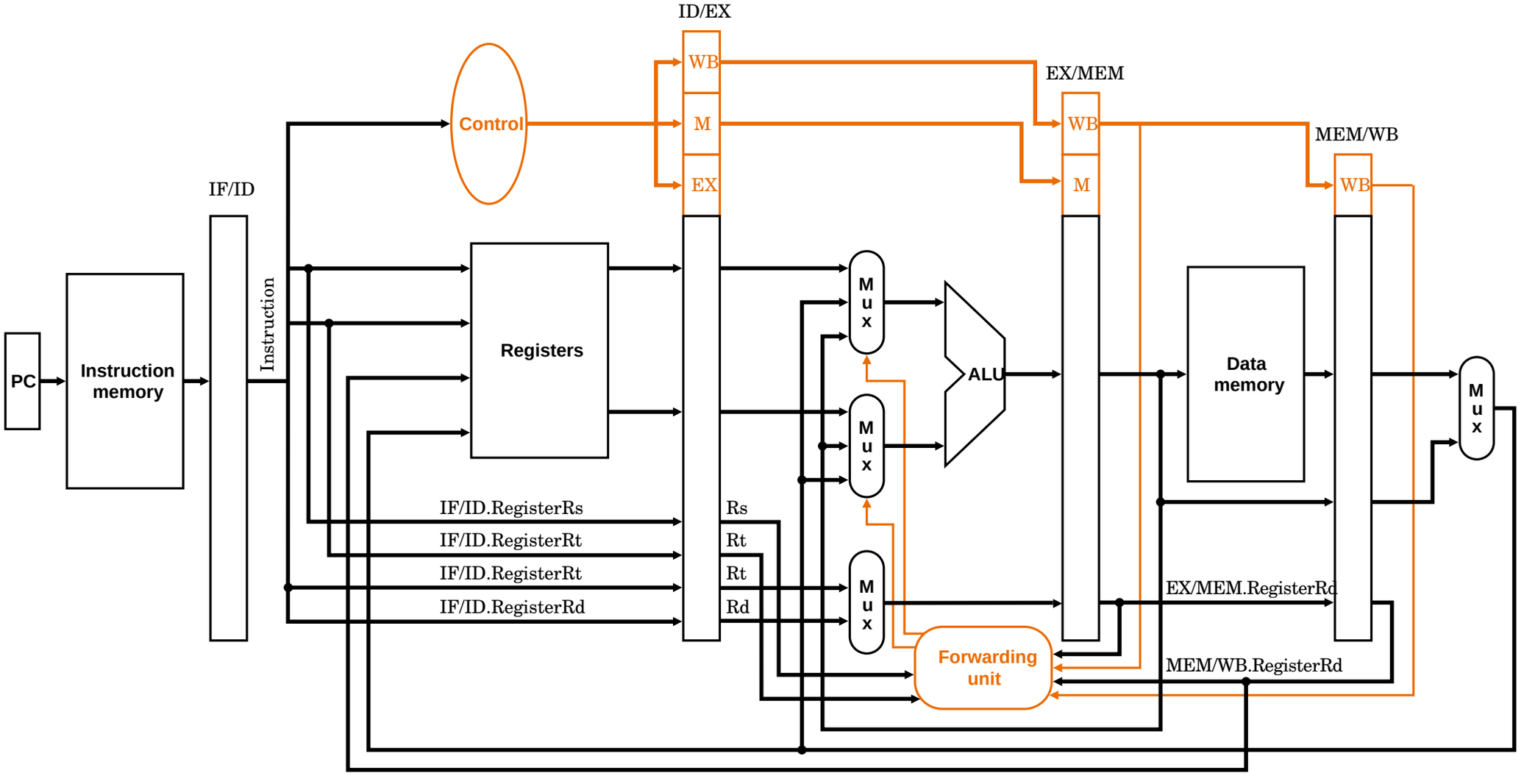
- Condiţiile de mai sus *nu* sunt *disjuncte*; e.g., ele se pot suprapune, ca în cazul

```
add $1,$1,$2;
add $1,$1,$3;
add $1,$1,$4;
```

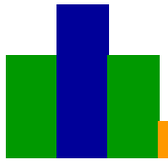
  Hazardul se rezolvă luând cel mai recent rezultat (din faza EX, nu MEM). Deci apare o condiţie suplimentară pentru Tipul 2.

- *Tip 2: în faza* MEM (access memorie) - *final*:
  - $\texttt{MEM/WB.RegWrite} \wedge \texttt{MEM/WB.RegisterRd} \neq 0$
    $\wedge \texttt{EX/MEM.RegisterRd} \neq \texttt{ID/EX.RegisterRs}$
    $\wedge \texttt{MEM/WB.RegisterRd} = \texttt{ID/EX.RegisterRs} \Rightarrow \texttt{ForwardA} = 01$
  - $\texttt{MEM/WB.RegWrite} \wedge \texttt{MEM/WB.RegisterRd} \neq 0$
    $\wedge \texttt{EX/MEM.RegisterRd} \neq \texttt{ID/EX.RegisterRt}$
    $\wedge \texttt{MEM/WB.RegisterRd} = \texttt{ID/EX.RegisterRt} \Rightarrow \texttt{ForwardB} = 01$

..Dependenta de date



Schiţă a căii de date cu *componenta de control pentru avansare* utilizată pentru rezolvarea hazardului de date (ilustrată pe o cale de date simplificată).



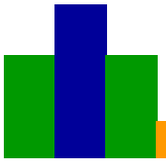
..Dependenta de date

Exemplu:

- In exemplele de mai jos ilustrăm cum funcţionează tehnica de avansare pe următoarea secvenţă de program

```
sub   $2,$1,$3;        (a)
and   $4,$2,$5;        (b)
or    $4,$4,$2;        (c)
add   $9,$4,$2;        (d)
```

- Notaţii:
  - k_a valoarea registrului k *la intrarea* în instrucţiunea a.
  - k_{+a} valoarea registrului k *la ieşirea* din instrucţiunea a.
  - $k_{<a}$ o valoarea nedefinită a registrului k *înainte* de instrucţiunea a.



..Dependenta de date

Exemplu (cont.)

- Condiţiile de corectitudine sunt:

$$1_a - 3_a = 2_{+a} = 2_b = 2_c = 2_d;$$
$$2_b \wedge 5_b = 4_{+b} = 4_c;$$
$$4_c \vee 2_c = 4_{+c} = 4_d$$

- Tabelul de mai jos indică evoluţia calculului, unde:
  - o linie conţine toate valorile accesibile la un ceas dat;
  - culorile indică de unde se iau valorile corecte pentru regiştri ALU: albastru - din `EX/MEM`; roşu - din `MEM/WB`;

..Dependenta de date

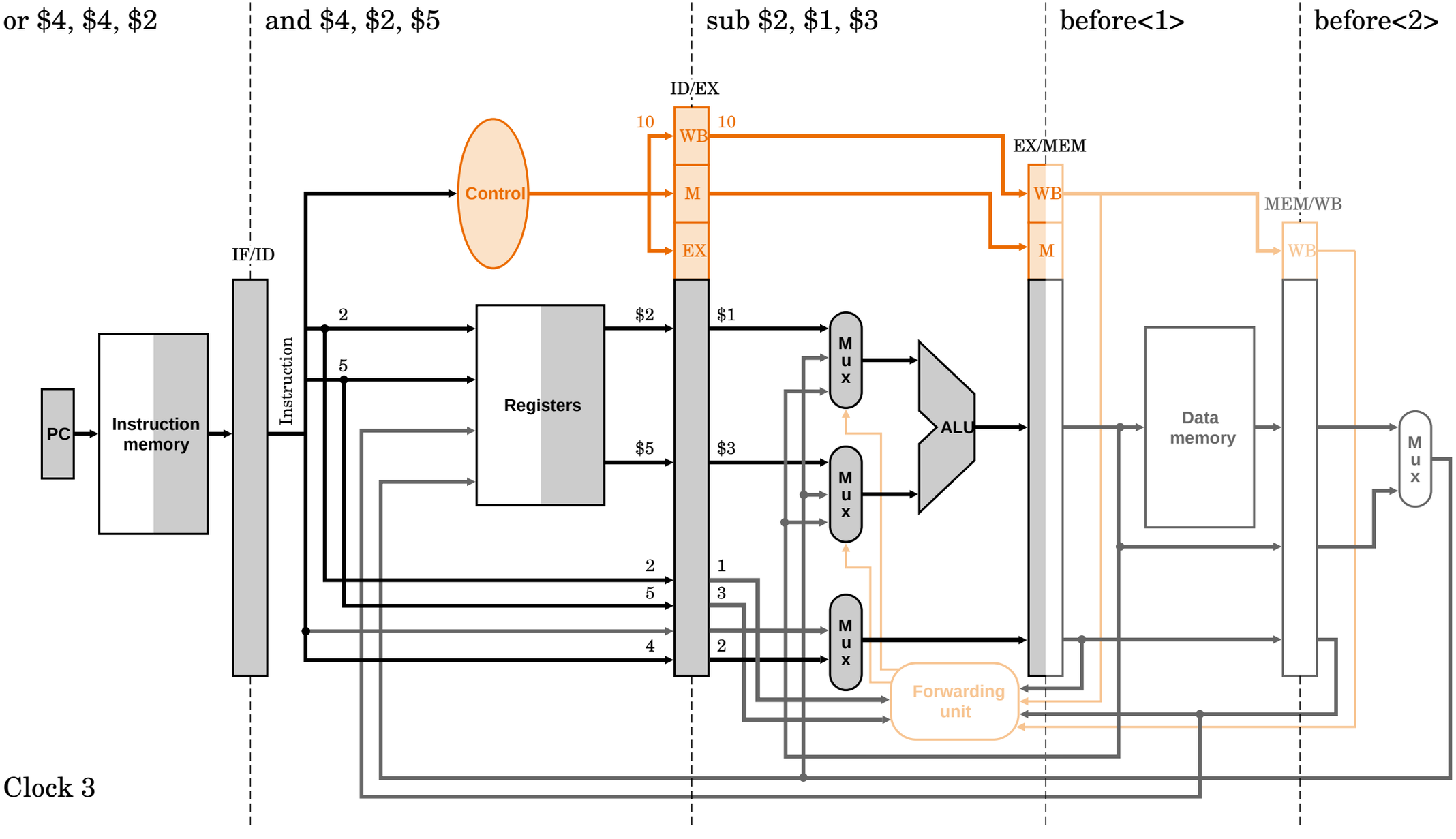
Exemplu (cont.)

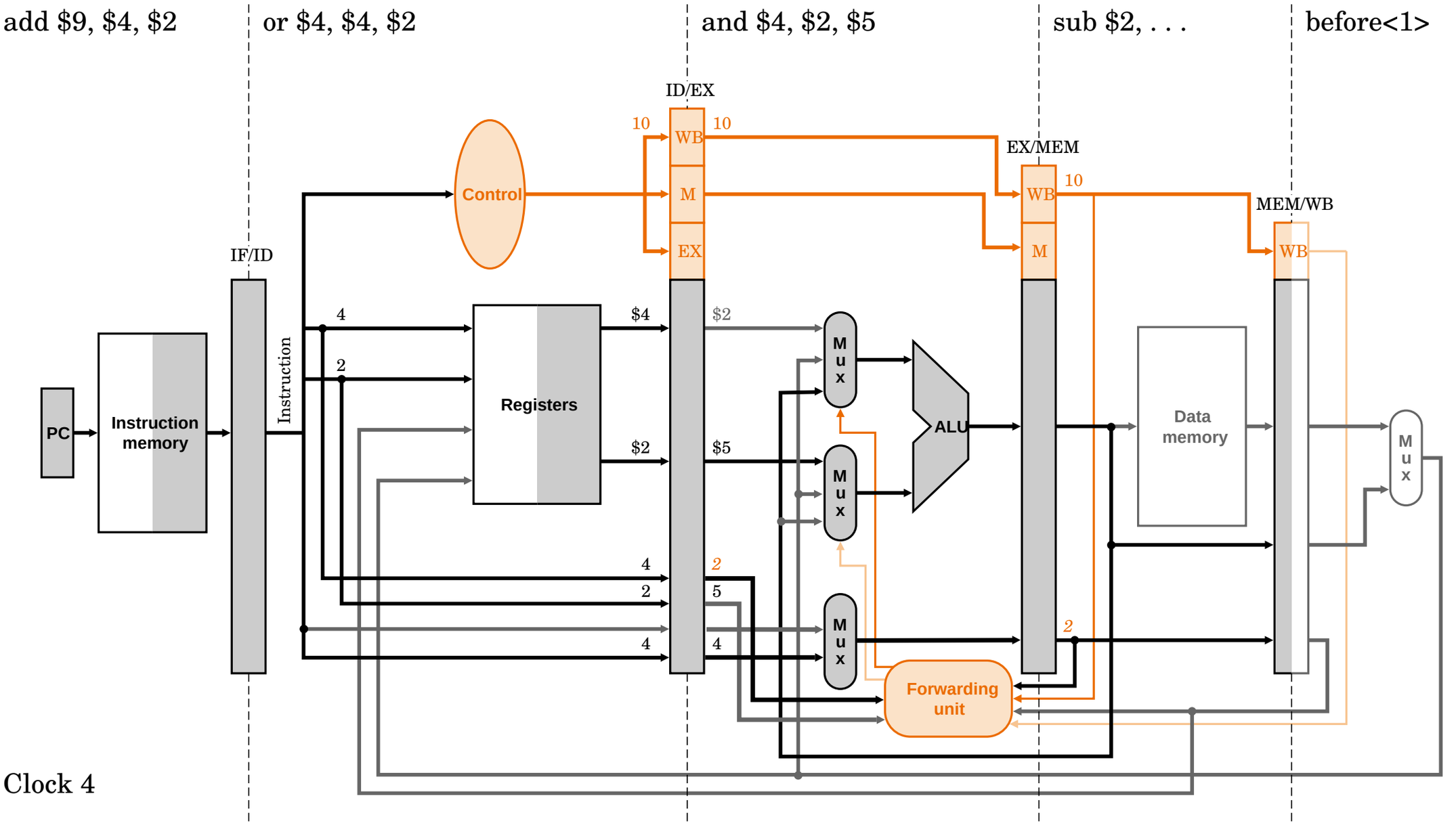
- Tabel cu evoluţia regiştrilor semnificativi:

Ceas	PC	IF/ID	ID/EX(rs,rt)	EX/MEM	MEM/WB
1	a				
2	b	I_a			
3	c	I_b	$1_a, 3_a$		
4	d	I_c	$2_{<b}, 5_b$	$1_a - 3_a$	
5		I_d	$4_{<c}, 2_{<c}$	$2_b \wedge 5_b$	$1_a - 3_a$
6			$4_{<d}, 2_d$	$4_c \vee 2_c$	$2_b \wedge 5_b$
7				$4_d + 2_d$	$4_c \vee 2_c$
8					$4_d + 2_d$

- De notat că în pasul 6, registrul 4 de folosit este cel din EX/MEM, nu cel din MEM/WB.

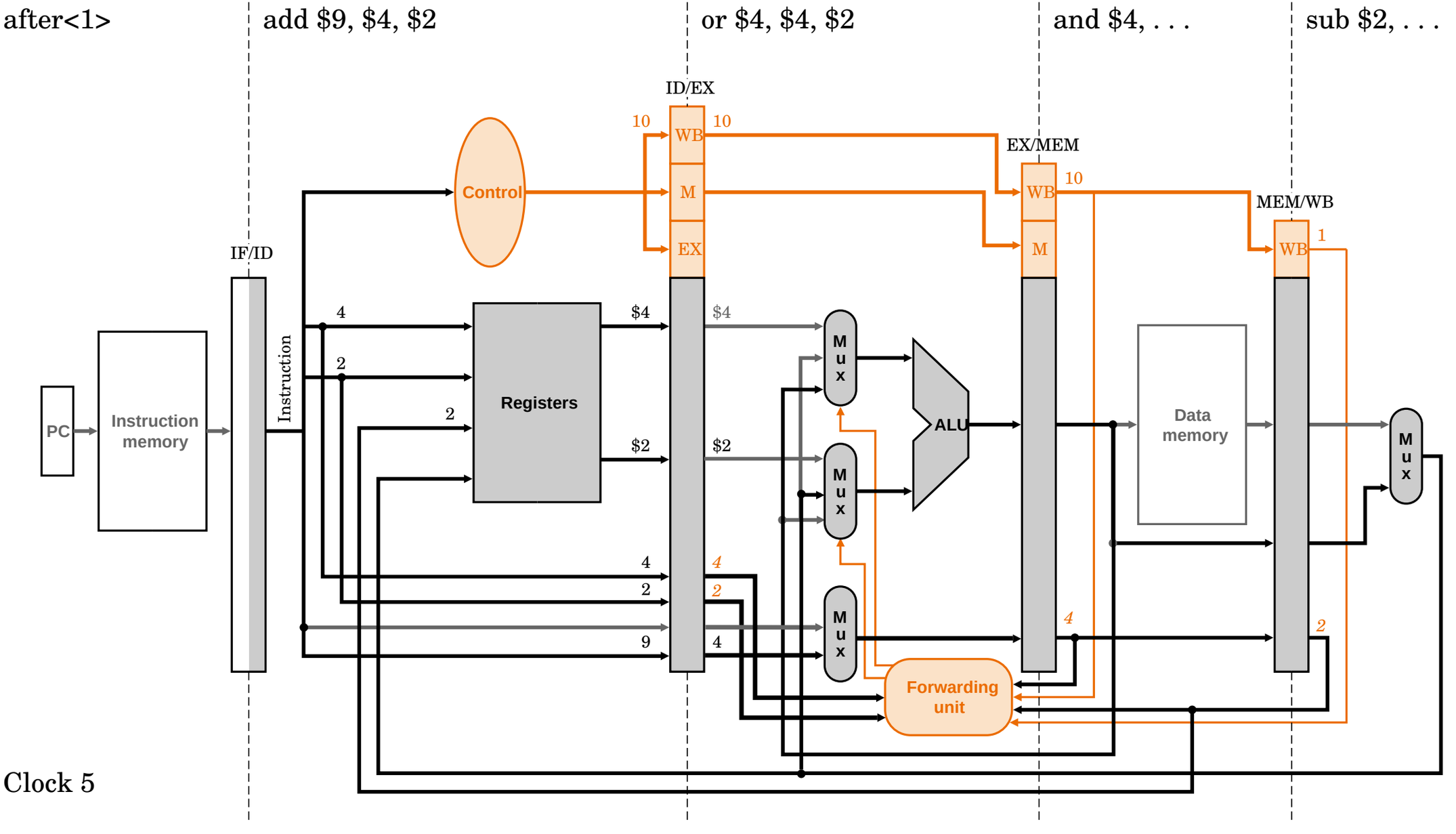
(Ciclul 3)

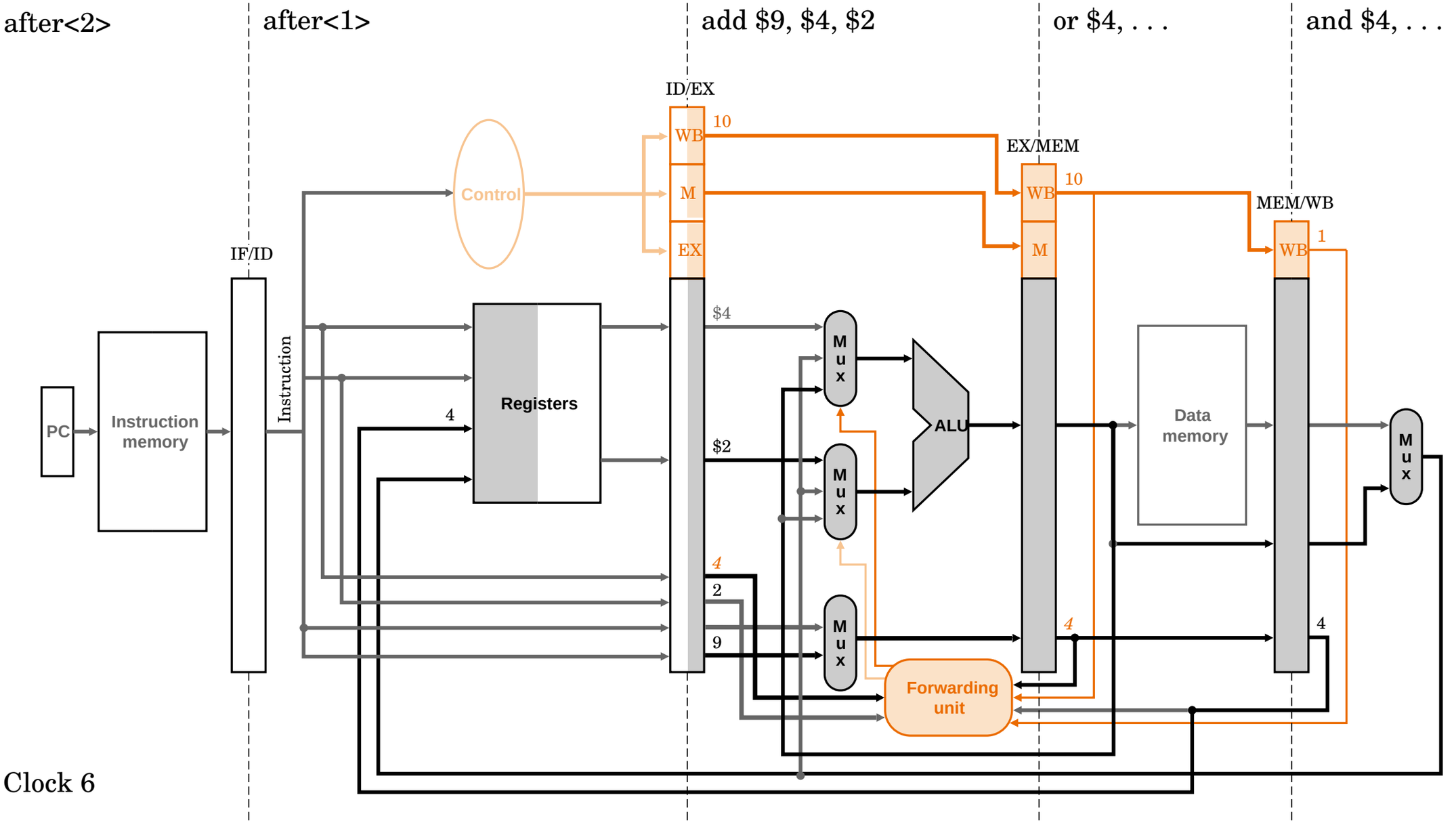

or $4, $4, $2
and $4, $2, $5
sub $2, $1, $3
before<1>
before<2>
ID/EX
EX/MEM
MEM/WB
IF/ID
10
10
WB
M
EX
WB
M
WB
Control
PC
Instruction memory
Instruction
2
5
Registers
$2
$5
$1
$3
2
5
4
1
3
2
M u x
M u x
M u x
ALU
Data memory
M u x
Forwarding unit
Clock 3


add $9, $4, $2
or $4, $4, $2
and $4, $2, $5
sub $2, . . .
before<1>
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
10
10
10
PC
Instruction memory
Instruction
4
2
Registers
$4
$2
$2
$5
Mux
ALU
Data memory
Mux
4
2
4
2
5
4
2
Forwarding unit
Clock 4

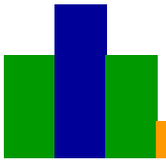
(Ciclul 4)

(Ciclul 5)


after<1>
add $9, $4, $2
or $4, $4, $2
and $4, . . .
sub $2, . . .
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
10
10
10
1
PC
Instruction memory
Instruction
Registers
4
2
2
$4
$2
$4
$2
4
2
9
4
4
2
4
2
Mux
Mux
Mux
Mux
ALU
Data memory
Forwarding unit
Clock 5


after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .
ID/EX
EX/MEM
MEM/WB
WB
M
EX
10
10
1
Control
IF/ID
Instruction
PC
Instruction memory
Registers
4
$4
$2
4
2
9
Mux
ALU
Data memory
Forwarding unit
Clock 6

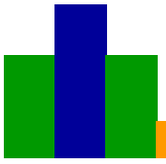
(Ciclul 6)



..Dependenta de date

Comentarii:

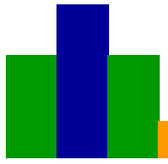
- Analiza de mai sus nu este exhaustivă: am utilizat instrucţiuni particulare (de *format R*); desenele simplificate au folosit această cale particulară de date.
- Hazard care poate fi rezolvat cu avansări poate apărea şi la *alte combinaţii* de instrucţiuni: Spre exemplu, la `load` urmat de `store` data de scris în memorie de ia din `MEM/WB`;
- In MIPS operaţia `nop` are codul format numai din zerouri, anume este codul pentru `sll $0,$0,$0` (shift left logical cu 0 poziţii), instrucţiune care într-adevăr nu modifică nimic;



Procesorul: Tehnica de pipeline

Cuprins:

- Tehnica de pipeline
- Calea de date cu pipeline
- Controlul pentru implementari cu pipeline
- Hazard de date si avansari
- *Hazard de date si stationari*
- Hazard la ramificatii
- Exceptii
- Concluzii, diverse, etc.



Hazard de date si stationari

Intarzieri:

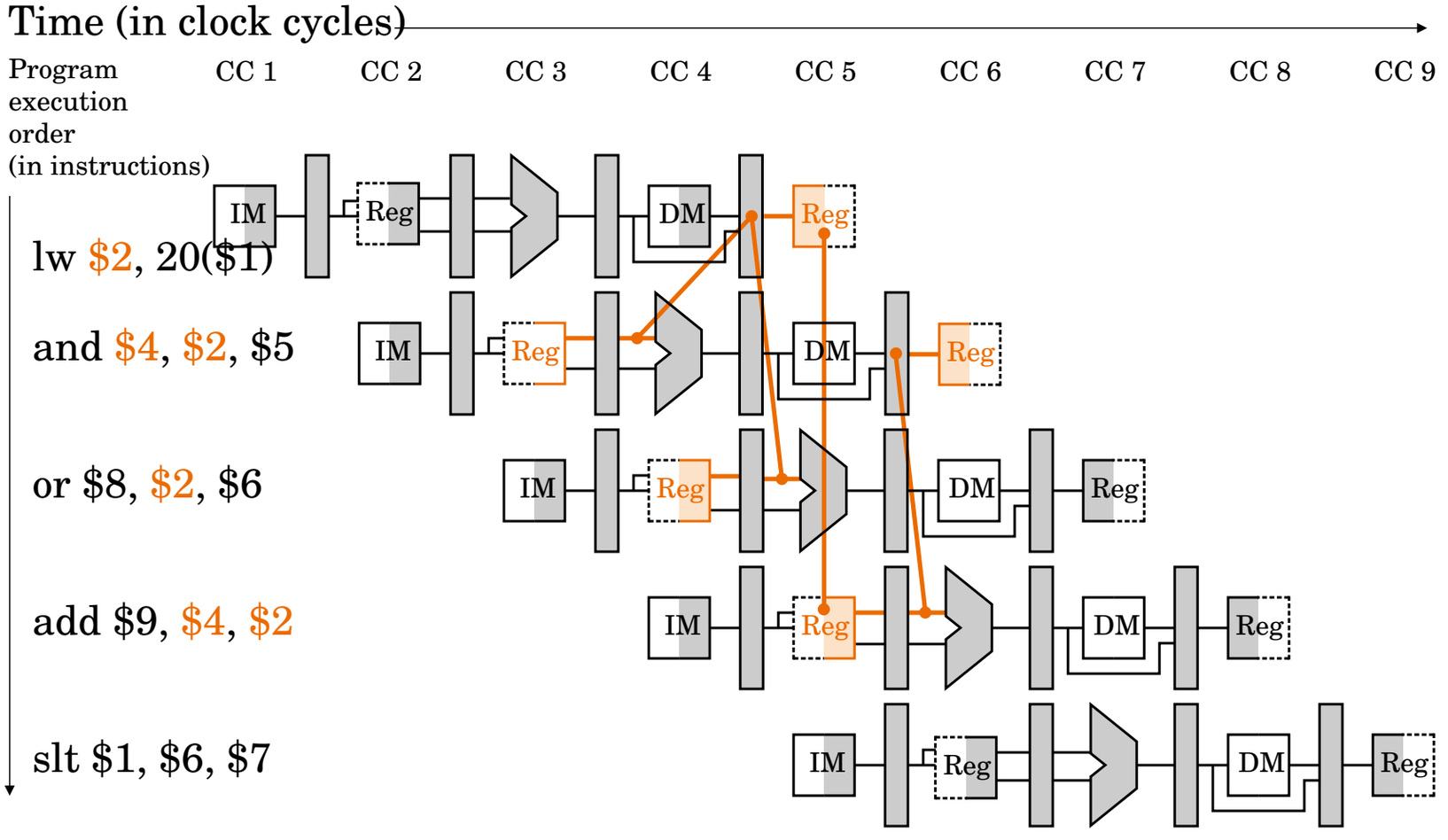
- Există cazuri în care avansarea nu poate rezolva hazardul, e.g., *load*, urmat de *citirea registrului*;
- In astfel de situaţii trebuiesc utilizate *întârzieri*, anume procesarea în pipleline este întârziată cu unul ori mai multe cicluri;
- Pentru detectarea acestor hazarduri folosim o *unitate de detectarea hazardului*, care implementează condiţiile necesare;
- La exemplu de mai sus (i.e., load urmat de citire registru), hazardul este detectat în faza `ID` cu condiţiile

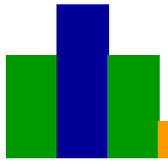
$$[\texttt{ID/EX.MemRead} \ \wedge \ (\texttt{ID/EX.RegisterRt} = \texttt{IF/ID.RegisterRs} \vee \ \texttt{ID/EX.RegisterRt} = \texttt{IF/ID.RegisterRt})] \Rightarrow \ \texttt{intarziere pieline}$$

..Hazard de date si stationari

Intarzieri (cont.)

- *Exemplu:* o *dependenţă de date* în pipeline care nu poate fi rezolvată cu avansări, ci cu *întârzieri* ale pipeline-ul:





Hazard de date si stationari

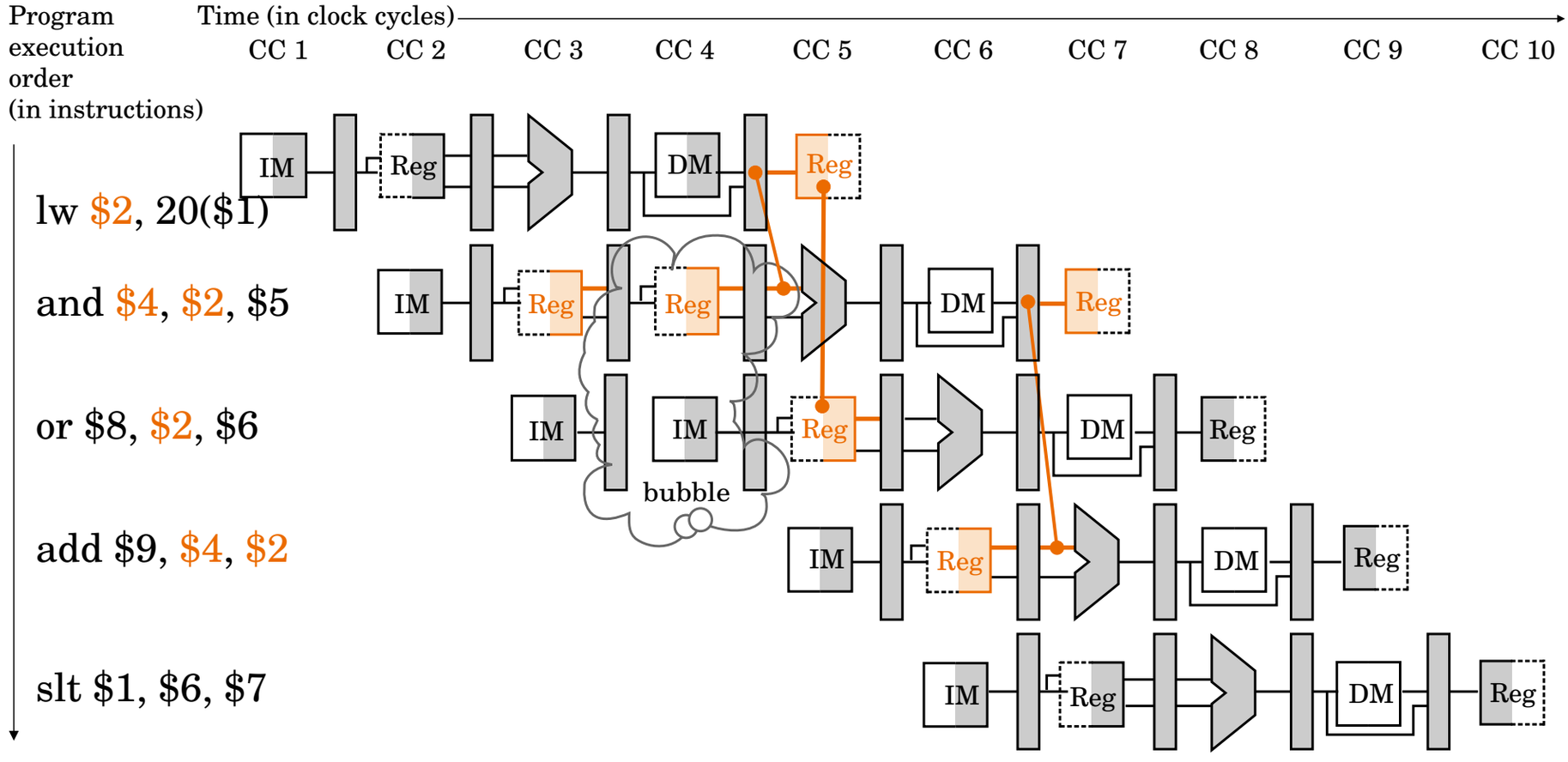
Intarzieri (cont.)

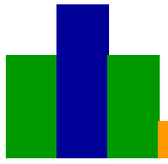
- Intârzierea trebuie să aibă acelaşi efect ca intrucţiunea `nop`, dar acum începând din faza de execuţie `EX` din pipeline;
- Pe exemplul dat întârzierea se simulează astfel:
  - în fazele `IF,ID` se *refolosesc* acelaşi date, *desetând* semnalele `PCWrite,IF/IDWrite`;
  - în fazele `EX,MEM,WB` se *desetează* cele 9 semnalele de control, deci nici regiştrii, nici memoria nu se pot modifica, neputându-se scrie (suficient: `MemWrite,RegWrite`);

..Hazard de date si stationari

Intarzieri (cont.)

- Figura ilustrează *rezolvarea cu întârzieri* a dependenţei anterioare de date:





..Hazard de date si stationari

Exemplu:

- Ilustrăm cum funcţionează tehnica de întârziere pe următoarea secvenţă de program

```
lw    $2,20($1);        (a)
and   $4,$2,$5;         (b)
or    $8,$2,$6;         (c)
add   $9,$4,$2;         (d)
slt   $1,$6,$7;         (e)
```

- Adăugăm convenţia că $M_a[v]$ (resp. $M_{+a}[v]$) reprezintă valoarea din memorie de la adresa v la intrarea în (resp. ieşirea din) instrucţiunea a. Condiţiile de corectitudine sunt:

$$M_a[1_a + 20] = 2_{+a} = 2_b = 2_c = 2_d$$
$$2_b \wedge 5_b = 4_{+b} = 4_d$$

..Hazard de date si stationari

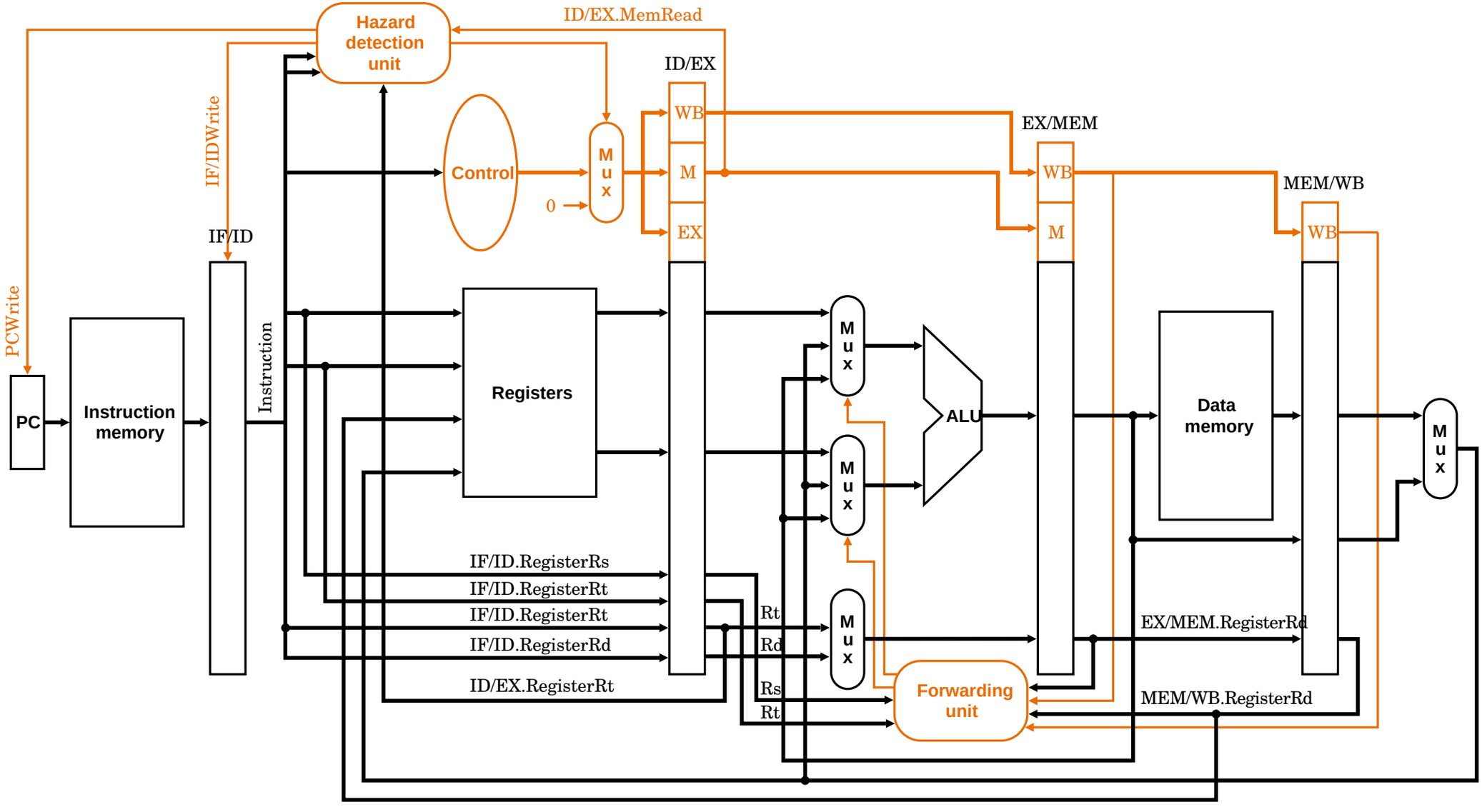
Exemplu (cont.)

- Tabelul de mai jos indică evoluţia calculului:

Ceas	PC	IF/ID	ID/EX(rs,rt)	EX/MEM	MEM/WB
1	a				
2	b	I_a			
3	c	I_b	1_a		
4	c	I_b	$2_{<b}, 5_b$	$1_a + 20$	
5	d	I_c	$2_{<b}, 5_b$	$2_{<b} \wedge 5_b$	$M_a[1_a + 20]$
6	e	I_d	$2_c, 6_c$	$2_b \wedge 5_b$	$2_{<b} \wedge 5_b$
7		I_e	$4_{<d}, 2_d$	$2_c \vee 6_c$	$2_b \wedge 5_b$
8			$6_e, 7_e$	$4_d + 2_d$	$2_c \vee 6_c$
9				6_e ?= 7_e	$4_d + 2_d$
10					6_e ?= 7_e

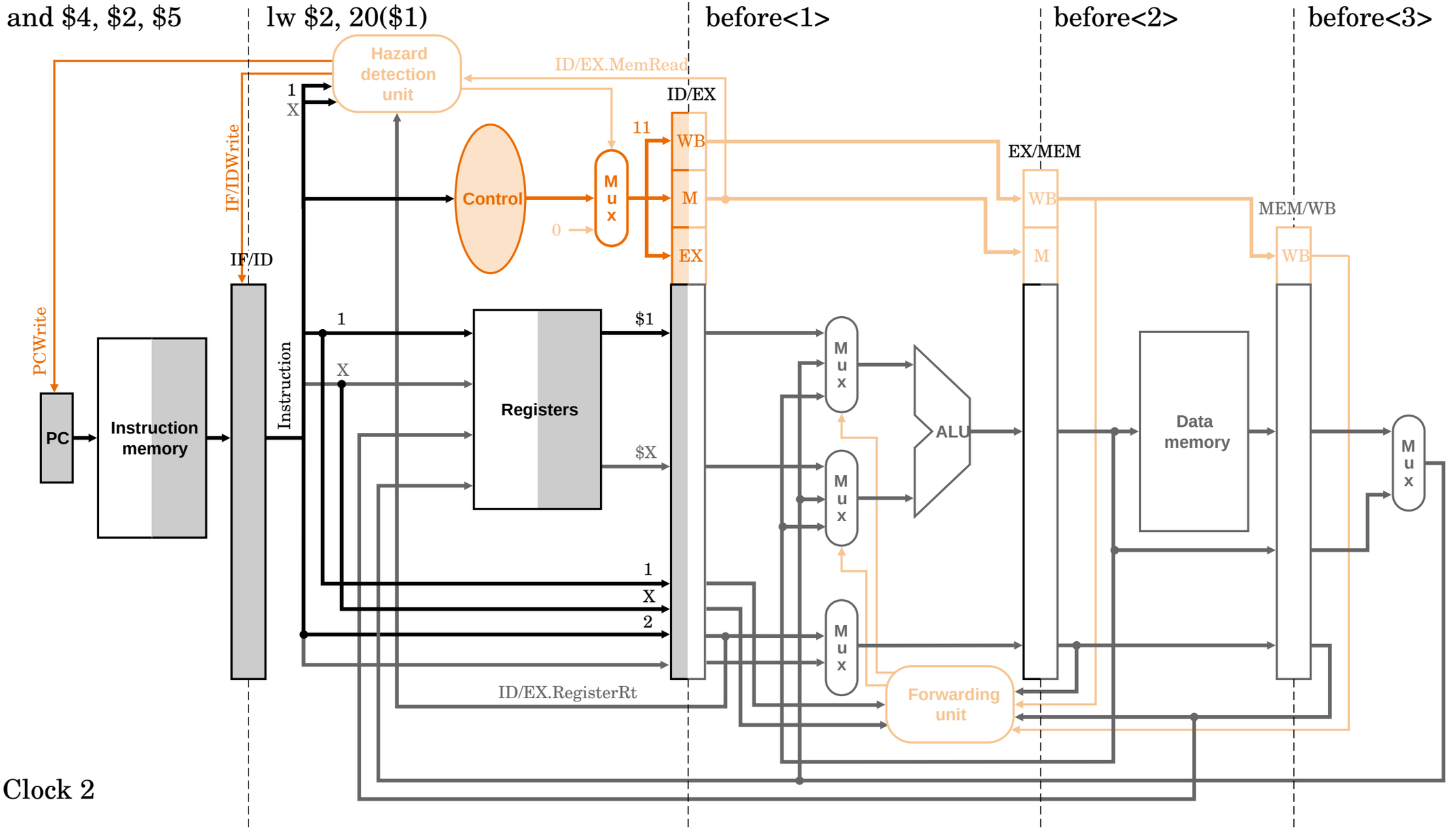
- In pasul 3 detectăm un hazard; procesarea se întârzie cu un ciclu spre a putea folosi valoarea corectă pentru 2_b în pasul 5.

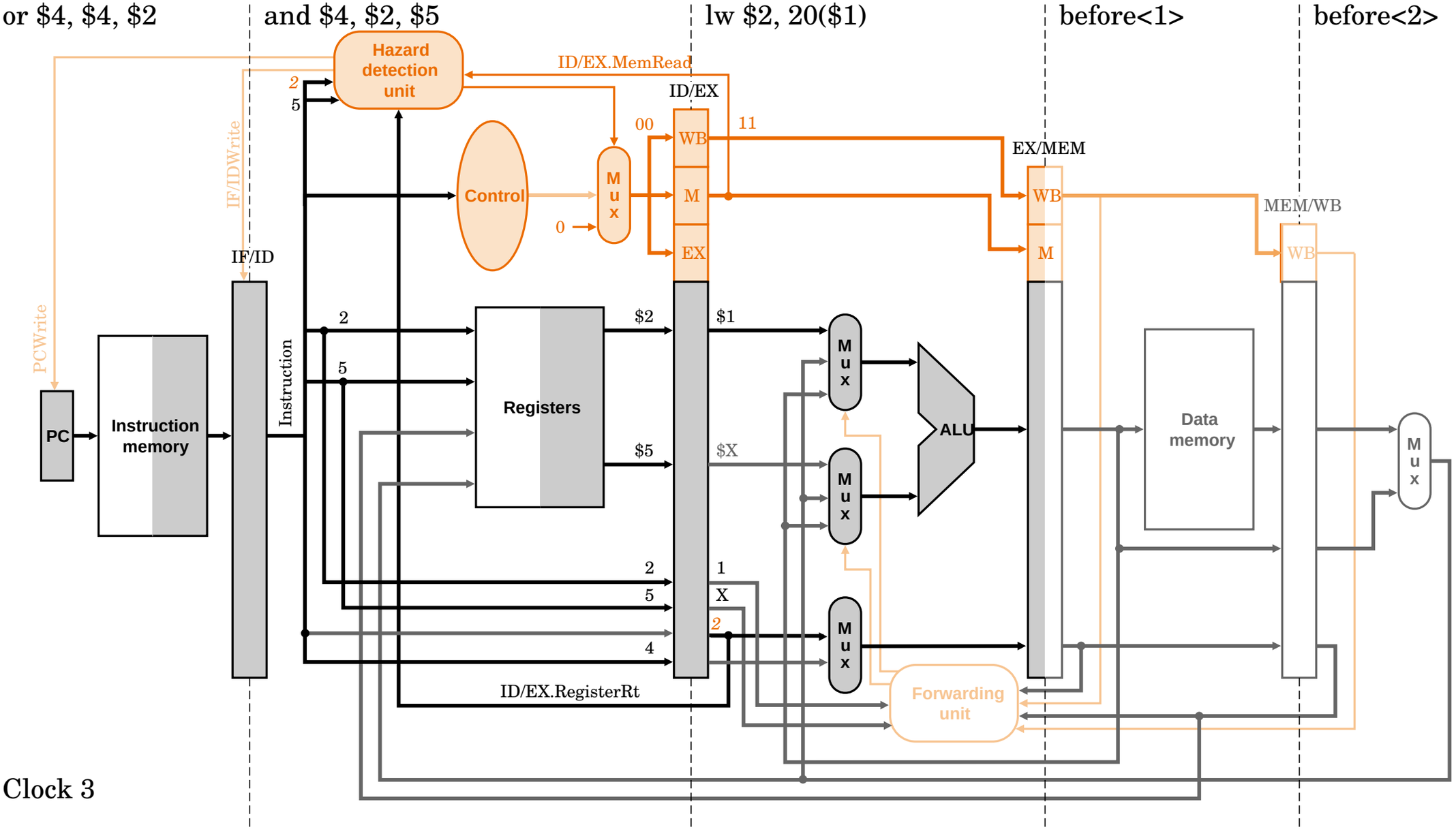
..Hazard de date si stationari



Schiţă a căii de date cu componenta de *detecţie hazarduri*, adăugată la cea pentru avansări (ilustrată pe o cale de date simplificată).

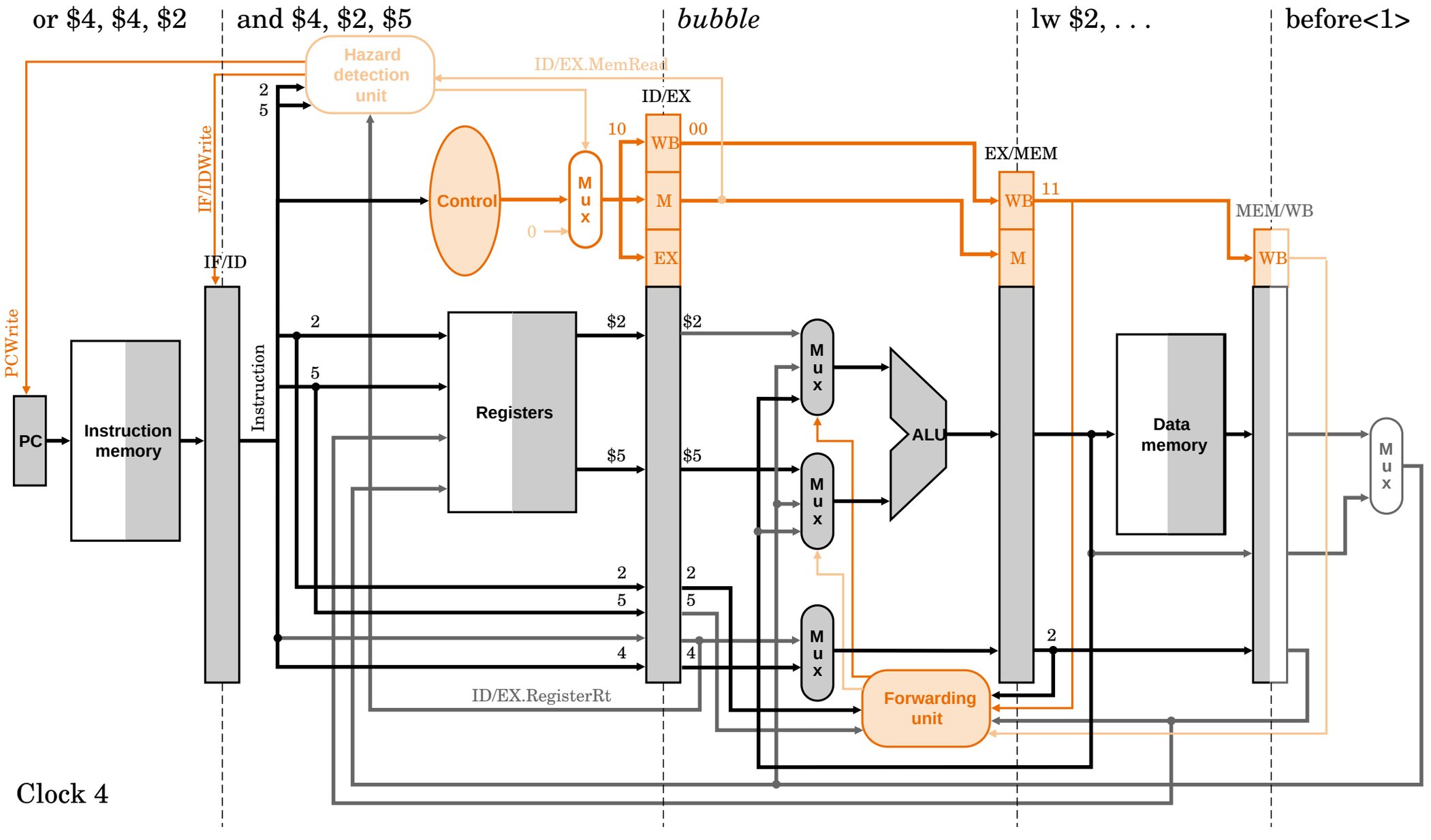
(Ciclul 2)

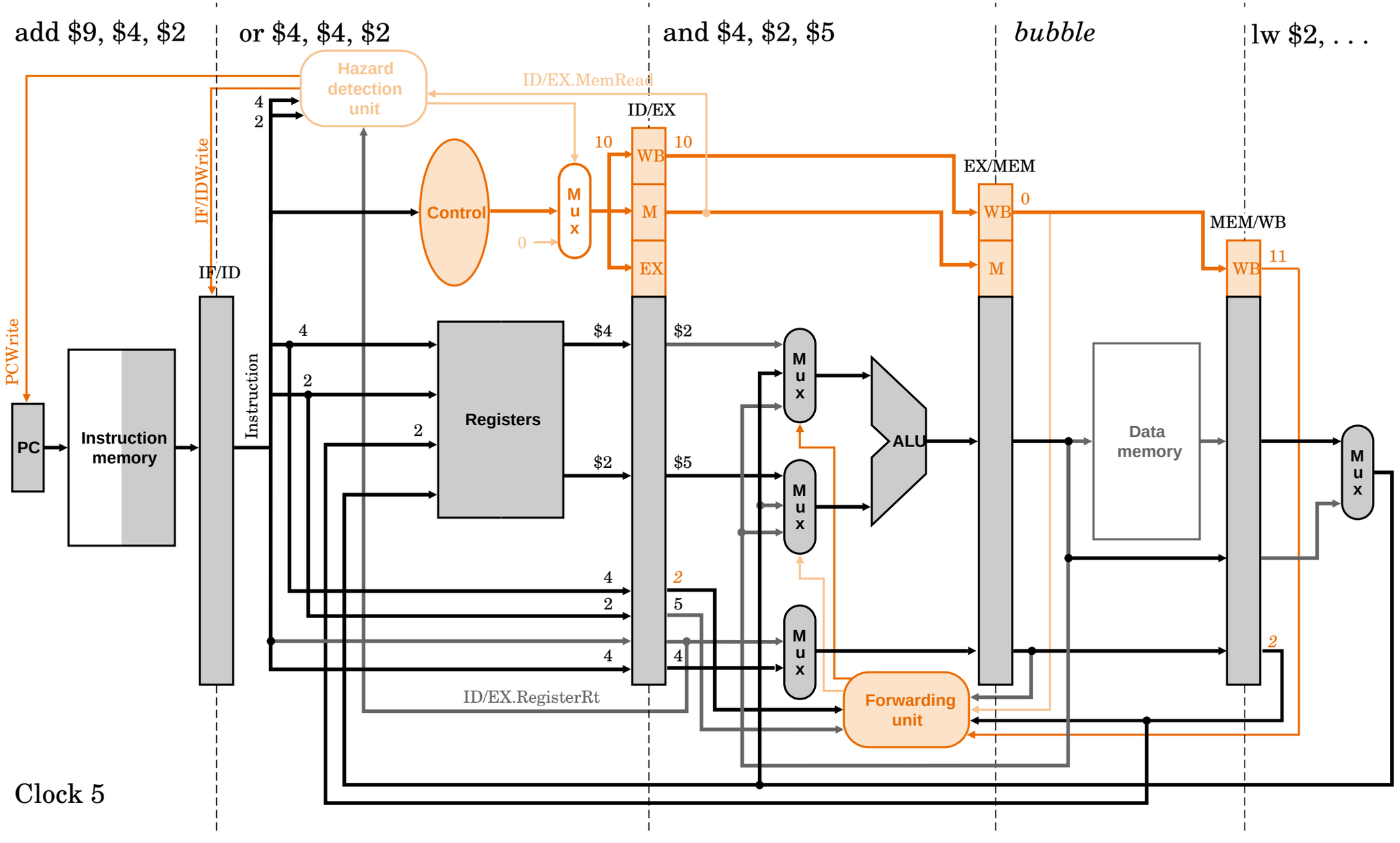

and $4, $2, $5
lw $2, 20($1)
before<1>
before<2>
before<3>
Hazard detection unit
ID/EX.MemRead
IF/IDWrite
PCWrite
Control
M u x
11
0
WB
M
EX
ID/EX
EX/MEM
MEM/WB
IF/ID
PC
Instruction memory
Instruction
Registers
$1
$X
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
Clock 2


or $4, $4, $2
and $4, $2, $5
lw $2, 20($1)
before<1>
before<2>
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/IDWrite
PCWrite
Control
M u x
00
11
0
WB
M
EX
IF/ID
PC
Instruction memory
Instruction
Registers
$2
$5
$1
$X
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
Clock 3

(Ciclul 3)

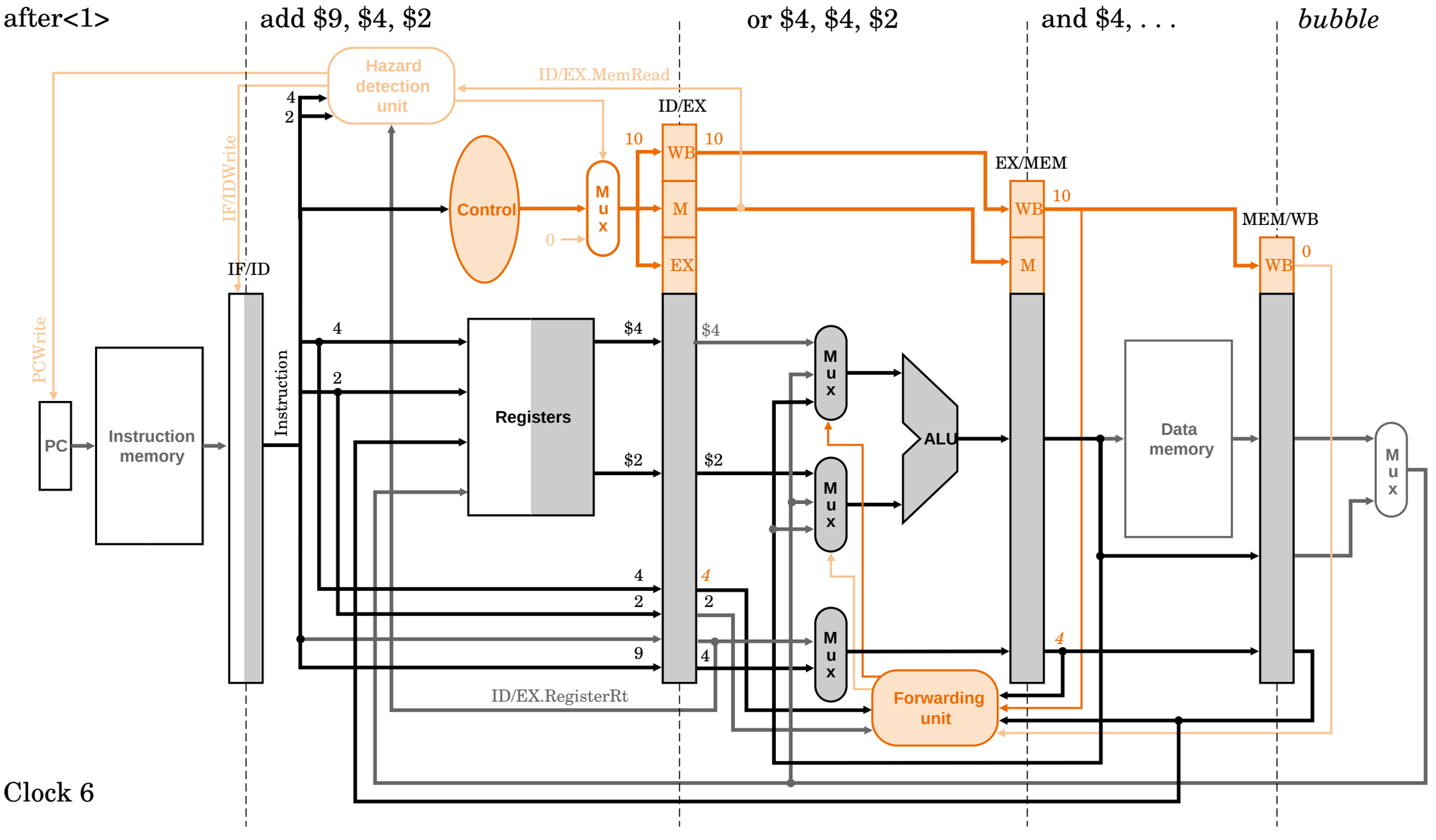
(Ciclul 4)

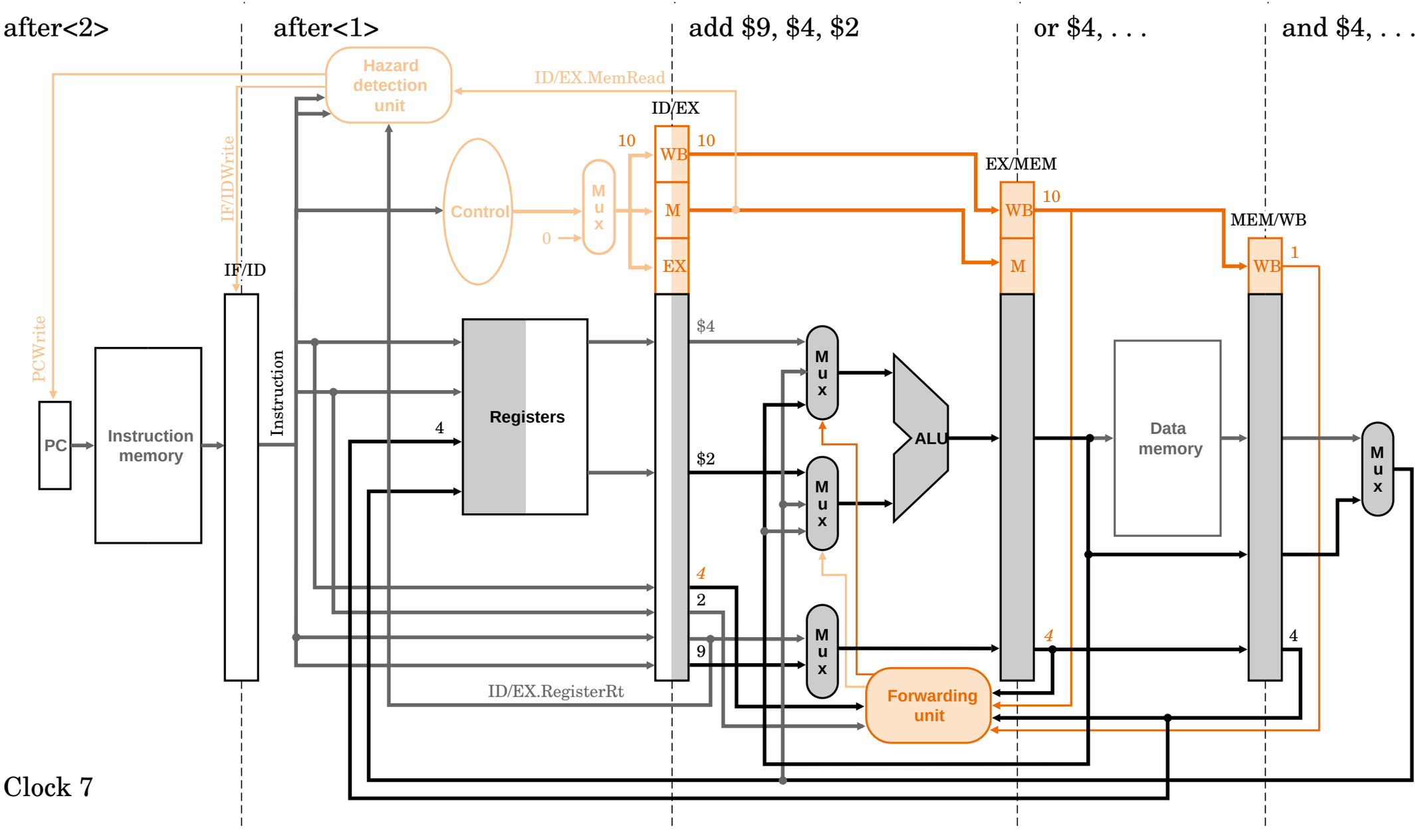

or $4, $4, $2
and $4, $2, $5
bubble
lw $2, . . .
before<1>
Hazard detection unit
ID/EX.MemRead
IF/IDWrite
PCWrite
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
Mux
WB
M
EX
10
00
11
0
PC
Instruction memory
Instruction
Registers
ALU
Data memory
$2
$5
2
5
4
ID/EX.RegisterRt
Forwarding unit
Clock 4


add $9, $4, $2
or $4, $4, $2
and $4, $2, $5
bubble
lw $2, . . .
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/IDWrite
PCWrite
IF/ID
Control
Mux
WB
M
EX
PC
Instruction memory
Instruction
Registers
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
Clock 5

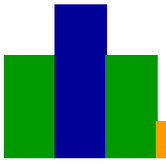
(Ciclul 5)

(Ciclul 6)


after<1>
add $9, $4, $2
or $4, $4, $2
and $4, . . .
bubble
Hazard detection unit
ID/EX.MemRead
IF/IDWrite
PCWrite
ID/EX
EX/MEM
MEM/WB
Control
M u x
WB
M
EX
10
0
IF/ID
Instruction
PC
Instruction memory
Registers
$4
$2
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
4
2
9
Clock 6


after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .
Hazard detection unit
ID/EX.MemRead
IF/IDWrite
PCWrite
Control
M u x
0
10
WB
M
EX
ID/EX
10
EX/MEM
WB
M
10
MEM/WB
WB
1
IF/ID
PC
Instruction memory
Instruction
Registers
4
$4
$2
4
2
9
ALU
Data memory
4
4
ID/EX.RegisterRt
Forwarding unit
Clock 7

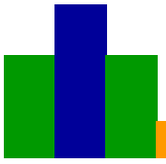
(Ciclul 7)



Procesorul: Tehnica de pipeline

Cuprins:

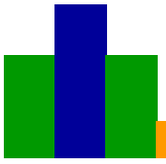
- Tehnica de pipeline
- Calea de date cu pipeline
- Controlul pentru implementari cu pipeline
- Hazard de date si avansari
- Hazard de date si stationari
- *Hazard la ramificatii*
- Exceptii
- Concluzii, diverse, etc.



Hazard la ramificatii

Generalitati:

- In afară de instrucţiunile de format R ori de acces la memorie, *hazarduri* pot produce şi instrucţiunile de *ramificare (branch)*.
- Abordarea este mai simplă, căci nu există tehnici generale de detecţie şi de rezolvare timpurie (precum avansările în cazul hazardul de date).
- In genere, apar mai puţin frecvent, iar tehnicile folosite sunt de a *reduce aşteptarea* ori de a *prezice rezultatul*.

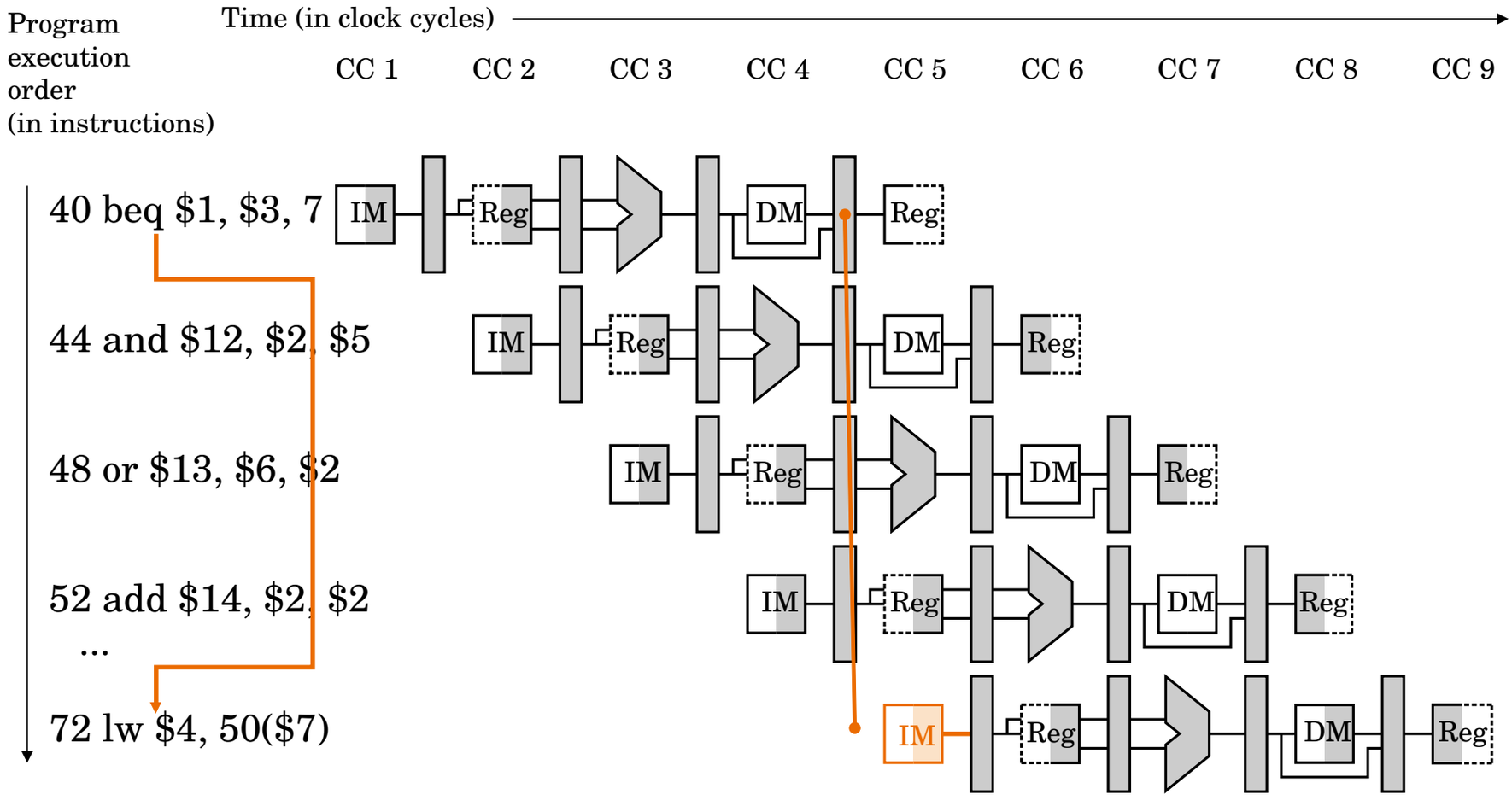


..Hazard la ramificatii

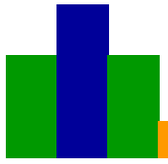
Ipoteza de neacceptare a ramifi carii: Procedura bazată pe ipoteza de a *nu accepta ramificarea* este următoarea:

- Se continuă cu secvenţa următoare.
- Decizia despre test o aflăm abia în faza MEM;
  - dacă decizia este de a accepta testul, se *curăţă (*flush*)* pipeline-ul şi se continuă cu instrucţiunea respectivă;
  - dacă nu se acceptă testul se continuă normal.

..Hazard la ramificatii



Impactul produs de instrucţiunile *branch* asupra pipeline-ului: se continuă în ipoteza că *ramificaţia nu este acceptată*; decizia exactă se află abia în faza MEM.

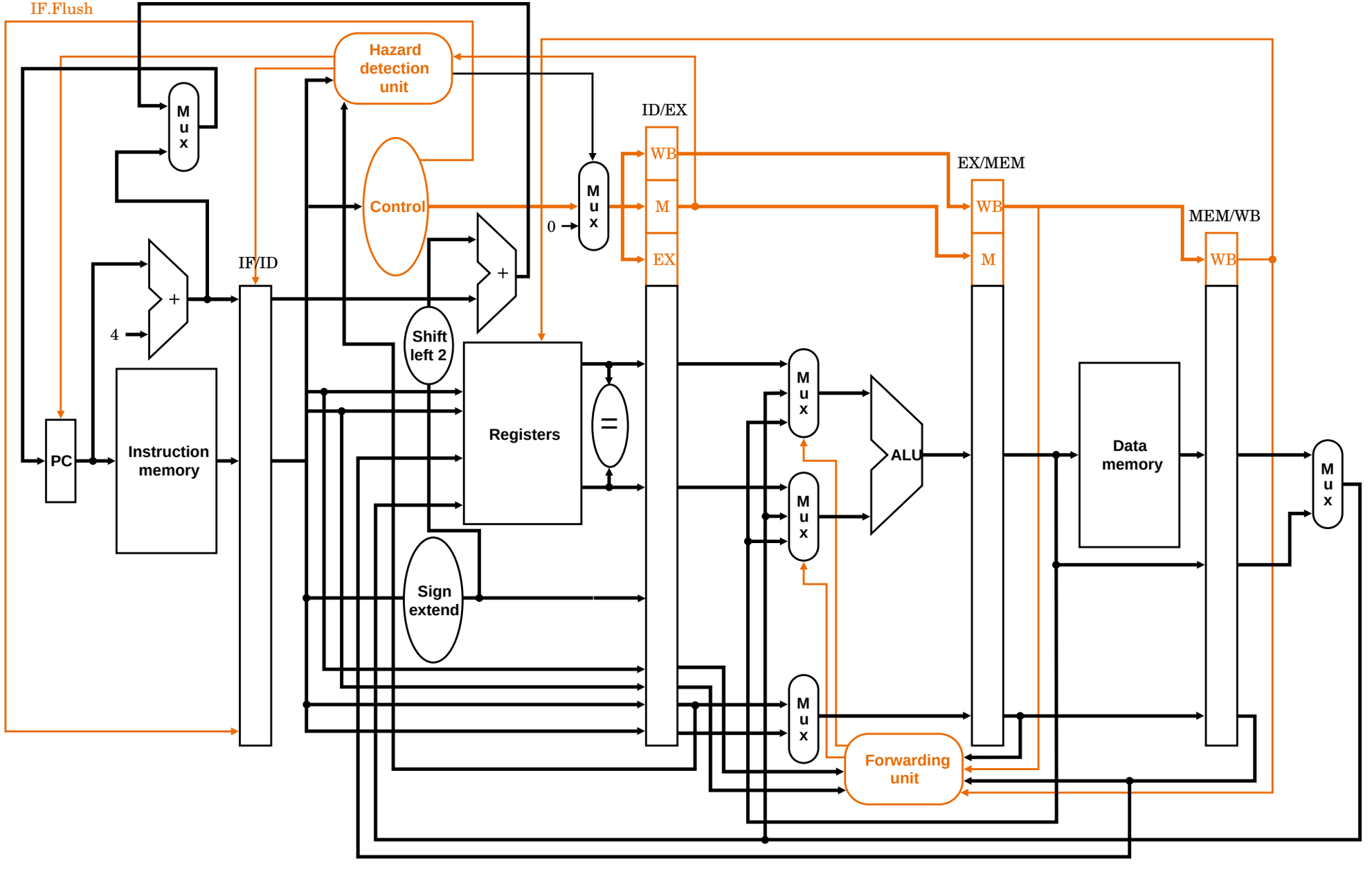


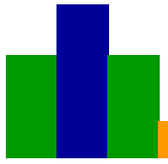
..Hazard la ramificatii

Reducerea întârzierii:

- Uneori, ca în cazul `beq`, *testul* se poate face *mai simplu* (nu cu scădere ALU): testez egalitatea bit-cu-bit, operaţie care poate fi făcută în faza `ID` cu hardware simplu.
- [Complicaţia este că mutând decizia de test în faza `ID` şi alte componente hardware dependente trebuiesc copiate acolo, spre exmplu cele pentru avansări.]
- Cu cele de mai sus se reduce aşteptarea de la 3 cicluri la 1.
- Istrucţiunea care trebuie curăţată este acum în faza `IF` şi se curăţă uşor *setând la 0 registrul* `IF/ID` (echivalent cu execuţia unei instrucţiunii `nop`, de cod 0).

Cale cu branch şi hardware pentru flush





..Hazard la ramificatii

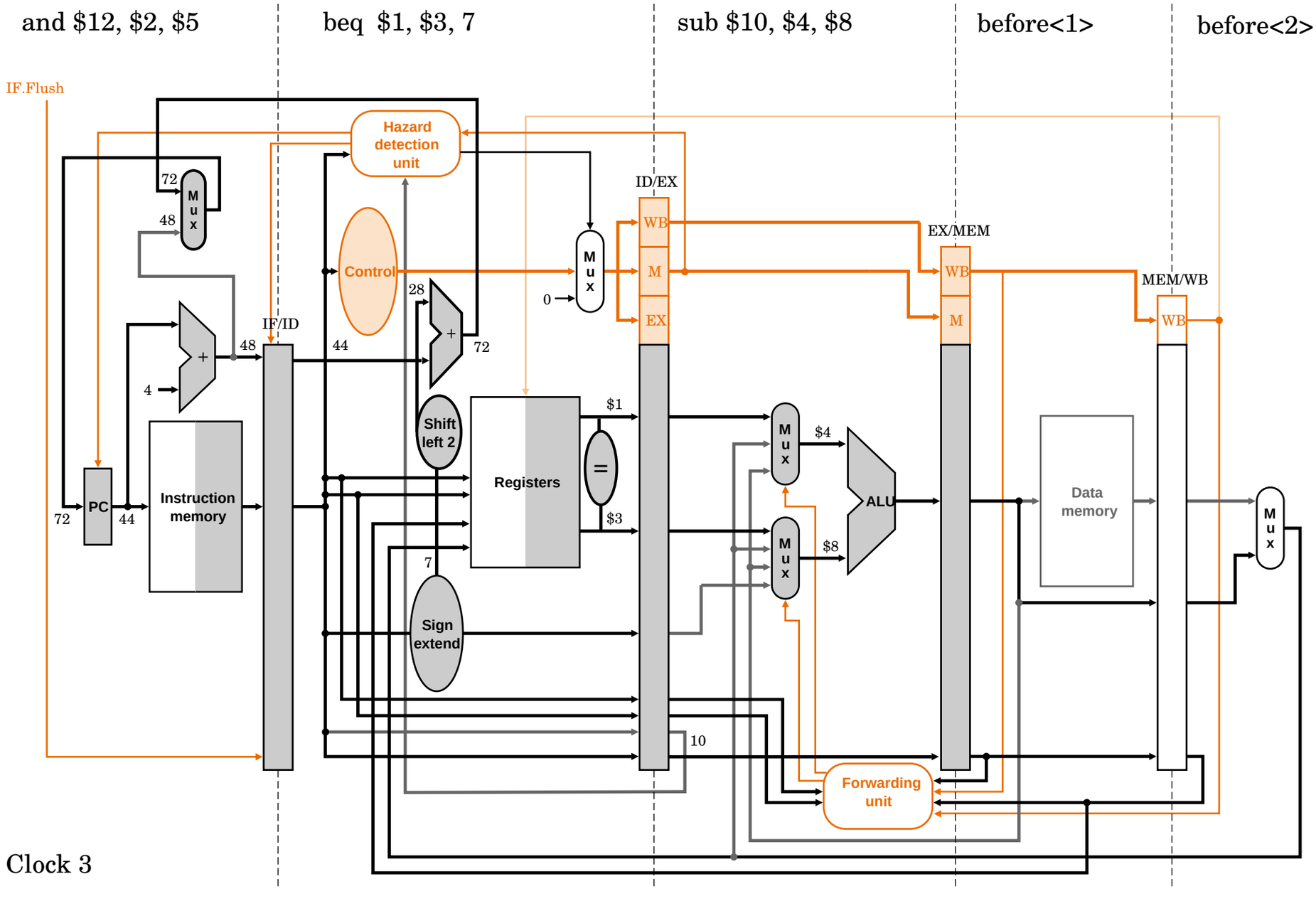
Exemplu:

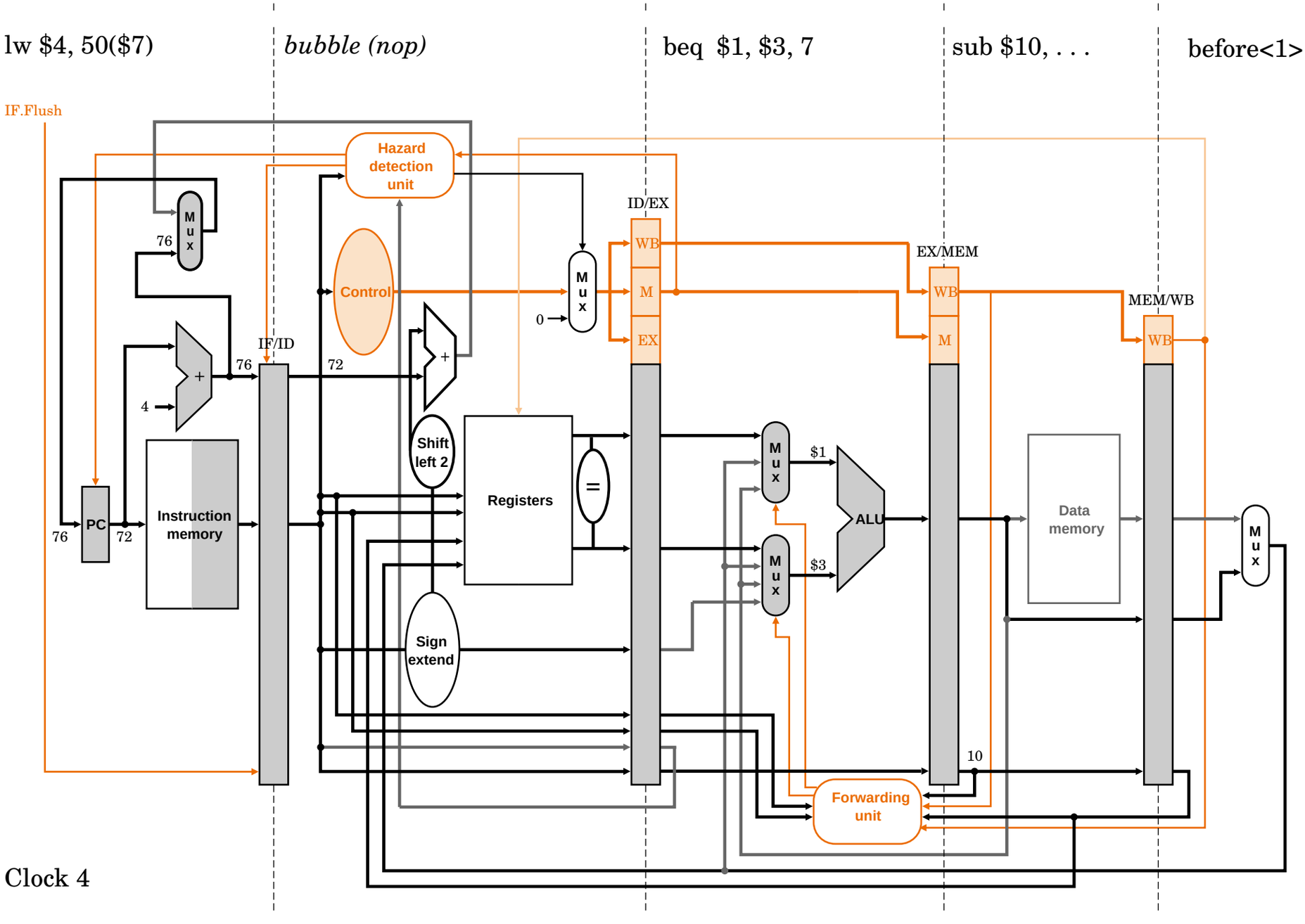
- Ilustrăm cum funcţionează tehnica pe următoarea secvenţă de program

```
36 sub   $1,$3,7;
40 beq   $10,$4,$8; # salt la 40+4+7*4=72
44 and   $12,$2,$5;
48 or    $13,$2,$6;
52 add   $14,$4,$2;
...
72 lw    $4,50($7);
```

- Desenul indică numai paşii semnificativi (ciclurile 3-4).

(Ciclul 3)


and $12, $2, $5
beq $1, $3, 7
sub $10, $4, $8
before<1>
before<2>
IF.Flush
Hazard detection unit
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
Mux
0
72
48
28
44
IF/ID
4
Shift left 2
Registers
$1
$3
=
$4
$8
ALU
PC
Instruction memory
Data memory
7
Sign extend
10
Forwarding unit
Clock 3

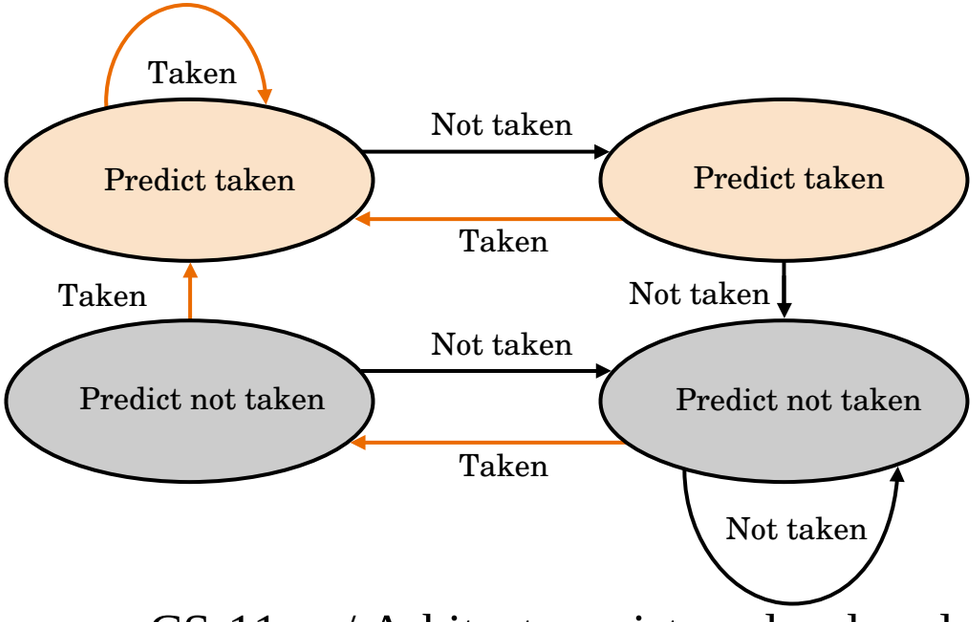

lw $4, 50($7)
bubble (nop)
beq $1, $3, 7
sub $10, . . .
before<1>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Mux
0
76
4
IF/ID
72
Shift left 2
Registers
=
Sign extend
PC
Instruction memory
$1
$3
ALU
Data memory
10
Forwarding unit
Clock 4

(Ciclul 4)

..Hazard la ramificatii

Predictii:

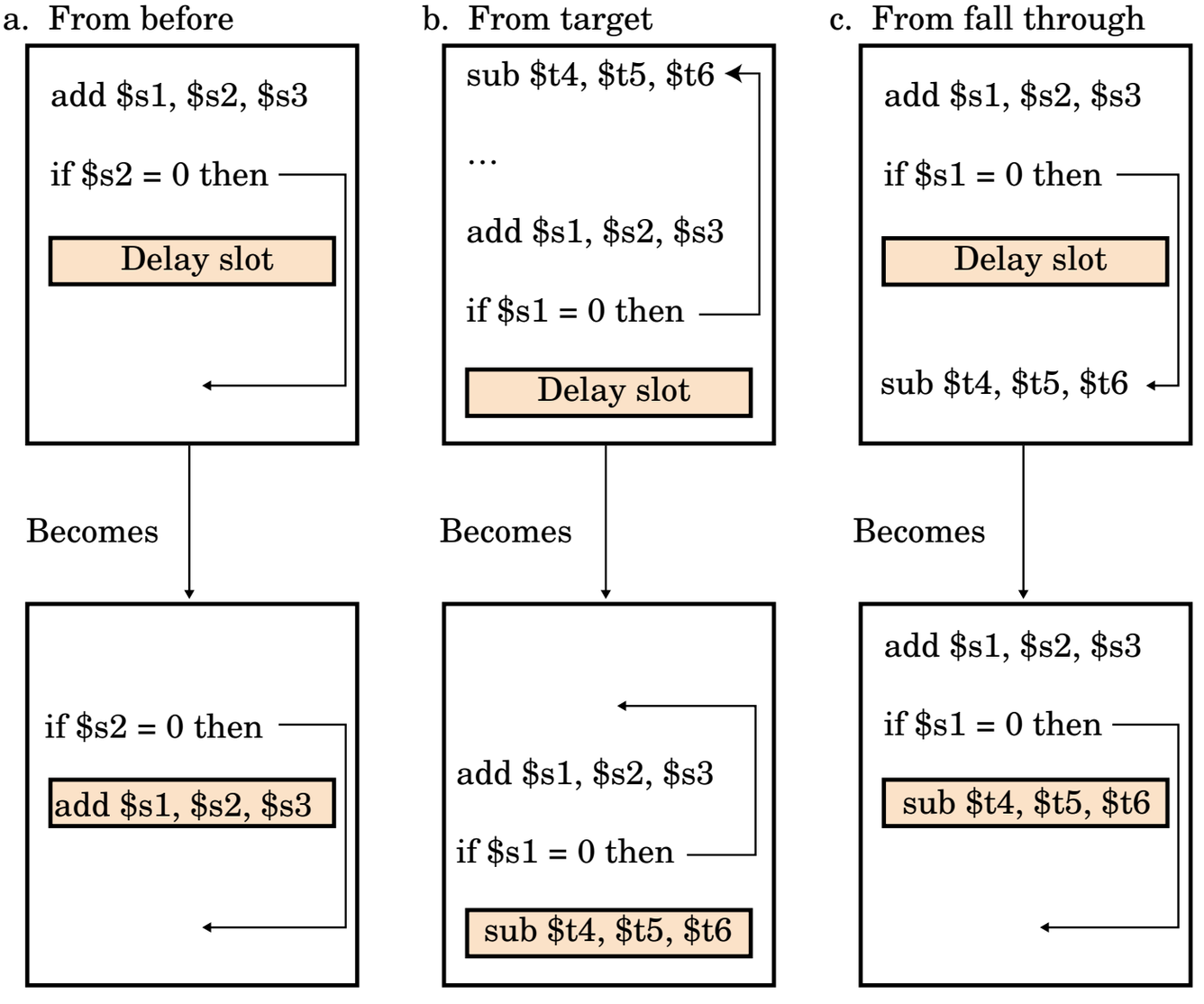
- Continuarea în “ipoteza că nu se acceptă testul” este un caz extrem de *predicţie la ramificării*.
- In general, se foloseşte un *buffer* (tabelă a istoriei selecţiei) pe baza căruia se face predicţia.
- *Exemplu:* O prezicere a selecţiei la branch care foloseşte 2 biţi (spre a codifica cele 4 stări) - schimbă prezicerea la 2 erori:



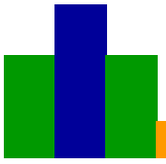
..Hazard la ramificatii

Slot de intarziere:

Exemple de planificări de activităţi spre a umple ciclul liber produs de întârzierile generate de ramificaţii.



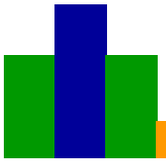
(a) este preferabilă; dacă nu se poate, se încearcă (b) ori (c).



Procesorul: Tehnica de pipeline

Cuprins:

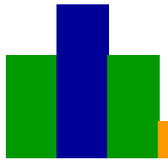
- Tehnica de pipeline
- Calea de date cu pipeline
- Controlul pentru implementari cu pipeline
- Hazard de date si avansari
- Hazard de date si stationari
- Hazard la ramificatii
- *Exceptii*
- Concluzii, diverse, etc.



Exceptii

Exceptii:

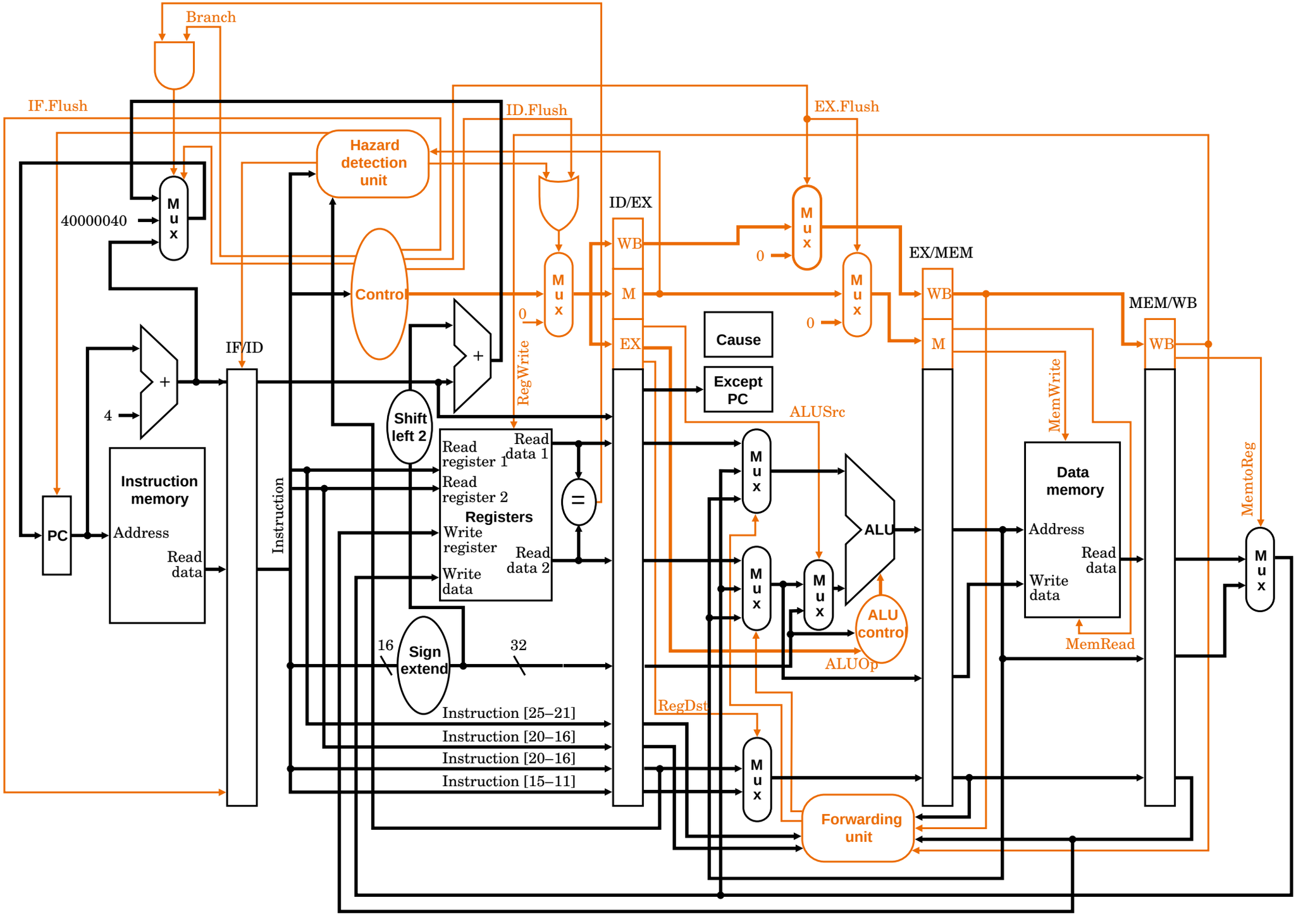
- Excepţiile se tratează similar, folosind regiştri EPC (Exception PC) şi Cause Register.
- Dacă apare o excepţie overflow în ALU, *transferăm controlul* la o adresa 4000 0040_{hex} (ce conţine adresa procedurii pentru excepţii) şi *curăţăm* pipeline-ul.
- Ultima operaţie se face cu un semnal de control nou `EX.Flush`, care setează semnalele uzuale de control la zero (suficient: `MemWrite` şi `IRWrite`, spre a nu modifica memoria ori regiştrii).
- Instrucţiunea care a produs eroarea poate fi mai greu de localizat în pipeline; alternativ, se pot folosi *excepţii imprecise* - totuşi, MIPS suportă *excepţii precise*.



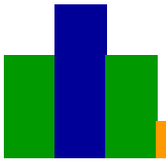
Performanta

Cu si fara pipeline: Pe exemplul anterior (compilatorul `gcc`) avem:

- *23% load*: 1 ciclu, uzual; 2, dacă este urmat de utilizare; medie: 1.5
- *13% store, 43% ALU*: 1 ciclu
- *19% branch*: 1 ciclu, corect; 2, dacă este predicţie greşită; medie: 1.25 (aproximativ 1/4 sunt predicţii greşite)
- *2% jump*: 2 cicluri
- *Total*: $1.5 \times .25 + 1 \times .13 + 1 \times .43 + 1.25 \times .19 + 2 \times .02 = 1.18$
- *Performanţă* (ciclu de 2ns): 8ns (1ciclu) / $1.18 \times$2ns (pipe) = 3.40.



Schiţă cu implementarea finală de pipeline.



Procesorul: Tehnica de pipeline

Cuprins:

- Tehnica de pipeline
- Calea de date cu pipeline
- Controlul pentru implementari cu pipeline
- Hazard de date si avansari
- Hazard de date si stationari
- Hazard la ramificatii
- Exceptii
- Pipeline superscalar si dinamic
- *Concluzii, diverse, etc.*