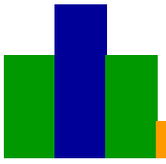
Lecţia 9:

Procesorul:
Tehnica de pipeline - I

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
- Pipeline superscalar si dinamic
- Concluzii, diverse, etc.

Tehnica de pipeline

Generalitati:

Pipeline: Este o tehnică de implementare în care se permite *suprapunerea execuţiei mai multor instrucţiuni*.

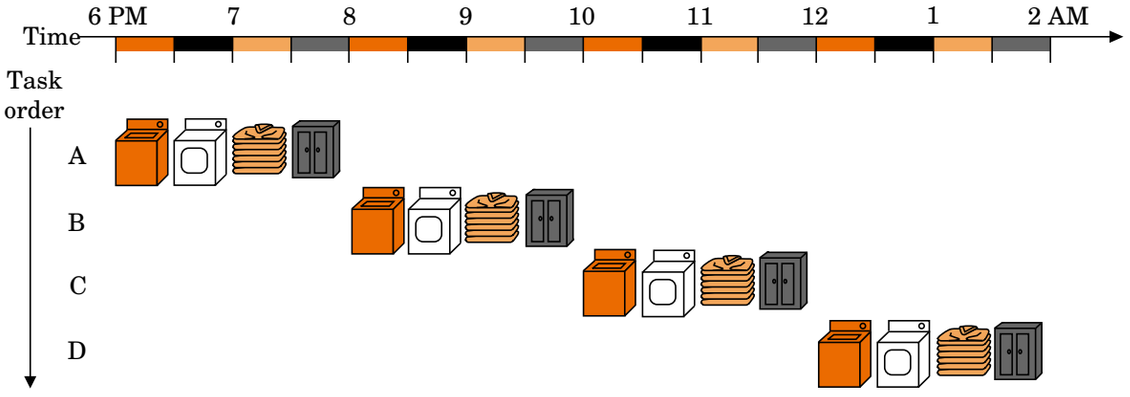
Exemplu: Avem de curăţat mai multe seturi de rufe murdare, operaţiile necesare fiind, în ordine: *spălare (s), uscare (u), călcare (c)* şi *aranjare (a)* (în dulap).

- Secvenţial: se repetă secvenţa $s(i), u(i), c(i), a(i)$ pentru fiecare set i;
- Paralel, cu pipeline: în fazele $1, 2, 3, 4, 5, \ldots$ avem acţiunile

$$
\begin{aligned}
& s(1), \\
& u(1) \,\&\, s(2), \\
& c(1) \,\&\, u(2) \,\&\, s(3), \\
& a(1) \,\&\, c(2) \,\&\, u(3) \,\&\, s(4), \\
& a(2) \,\&\, c(3) \,\&\, u(4) \,\&\, s(5), \\
& \ldots
\end{aligned}
$$

Tehnica de pipeline, intr-o imagine

secvenţial



paralel cu pipeline

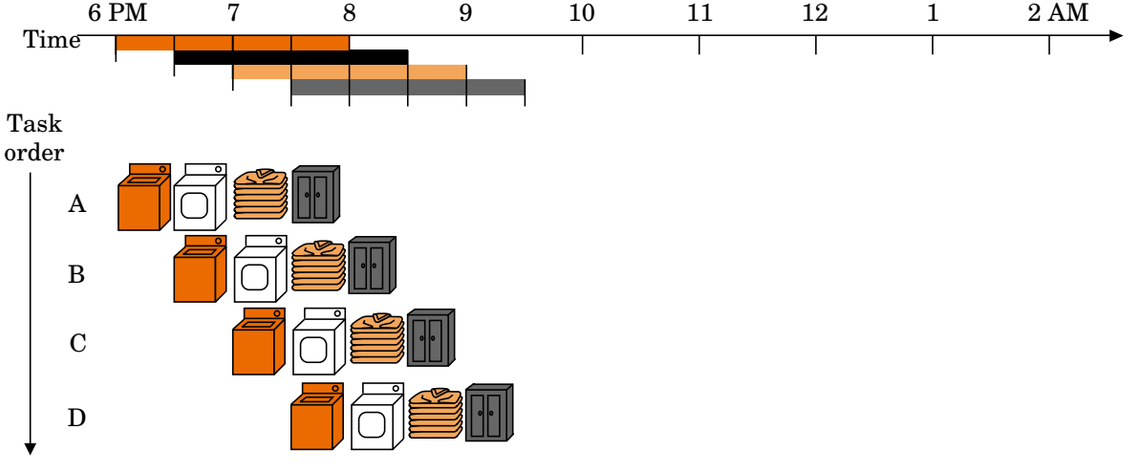
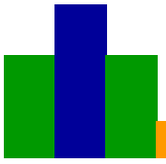


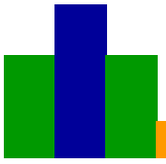
Figura descrie *tehnica de pipeline*, într-o imagine ilustrând operaţiile de *spălat, uscat, călcat* şi *aranjat* seturi de rufe `A`, `B`, ...



Tehnica de pipeline

Tehnica de pipeline - pe procesor: Tehnica de pipeline se poate aplica pe implementarea procesorului folosind cei *5 paşi* identificaţi anterior:

- **Pas 1:** Se *extrage* instrucţiunea din memorie;
- **Pas 2:** Se citesc regiştrii şi se *decodifică* instrucţiunea;
- **Pas 3:** Se *execută* operaţia ori se calculează adresa;
- **Pas 4:** Se *accesează memoria* pentru date;
- **Pas 5:** Se *scrie* rezultatul într-un *registru*.



Performanta: Ciclu simplu vs. pipeline

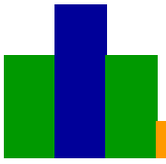
Ciclu simplu vs. pipeline: Presupunem o distribuţie a timpilor ca în figura

Tip instructiune	Pas 1	Pas 2	Pas 3	Pas 4	Pas 5	Total
Load (`lw`)	2ns	1ns	2ns	2ns	1ns	8ns
Store (`sw`)	2ns	1ns	2ns	2ns		7ns
Fromat-R (`add,sub,and,or,slt`)	2ns	1ns	2ns		1ns	7ns
Branch (`beq`)	2ns	1ns	2ns			5ns

Să considerăm un program simplu cu 3 instrucţiuni load

```
lw $1,100($0); lw $2,200($0); lw $3,200($0);
```

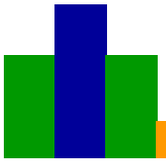
- Execuţia cu un ciclu durează $8 \times 3 = 24ns$;
- Cu o variantă de pipeline ca mai sus (exemplul cu rufe), am avea $2+2+8 = 12ns$;
- Performanţa: $Perf\_pipe/Perf\_1ciclu = 24/12 = 2.00$.



..Performanta: Ciclu simplu vs. pipleline

Ciclu simplu vs. pipeline (cont.)

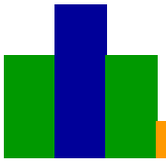
- Intr-o fază de pipeline pot fi active mai multe instrucţiuni; *durata unei faze* va fi *durata celei mai lungi operaţii* dintre cele procesate în faza respectivă (la noi este $2ns$);
- La limită, dacă în loc de 3 avem k instrucţiuni `lw` ($k \geq 5$), timpii sunt $8k$ şi $8+2k$, indicând o *viteză de 4 ori mai mare*;
- Sporul de viteză dat de tehnica de pipeline provine din *creşterea numărului de instrucţiuni procesate* într-un timp dat şi nu din focalizarea pe *micşorarea timpului de execuţie al unei instrucţiuni* individuale.



MIPS: pentru pipeline

Suport MIPS pentru pipeline:

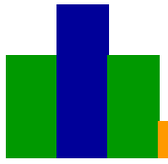
- *Instrucţiunile* MIPS au *acceaşi lungime*, facilitând extragerea şi decodarea în 2 faze. [Arhitectura 80x86 are instrucţiuni de lungime variabilă, între 1B şi 17B!]
- Numărul de *formate de instrucţiuni* MIPS este *redus*. Instrucţiunile au un *registru sursă comun*, care poate fi *citit odată cu decodarea* instrucţiunii. [Altfel, ar fi necesară o fază în plus.]
- *Accesul la memorie* se face numai prin *load/store*. [Dacă se admit operaţii cu operanzi din memorie ca la 80x86, fazele 3-4 ar adăuga o fază în plus de operare cu valorile citite din memorie.]
- In MIPS, *memoria este aliniată la cuvânt*, deci putem folosi *2 citiri simultane din memorie* (e.g., 2 regiştri, în Pasul 2).



Proiectarea unui pipeline

Proiectarea unui pipeline:

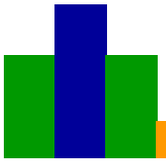
- Uneori, *următorul pas* de procesare a unei instrucţiuni *nu* poate fi procesat *în următoarea fază* din pipeline.
- Intr-o astfel de situaţie, spunem că aparare un *hazard*.
- Principalele tipuri de hazard sunt:
  - *structural* - generat de lipsa de suport hardware
  - *de control* - generat de instrucţiunile de salt din program
  - *de date* - generat de dependenţa variabilelor din program
- Prima restricţie este o restricţie *fizică*, pe când ultimile 2 sunt restricţii *logice*.



..Proiectarea unui pipeline

Hazard structural:

- *Hazardul structural* este generat de *lipsa de suport hardware* pentru a putea executa toate operaţiile *dintr-o fază* de pipeline.
- In exemplul cu rufe, apare hazard structural dacă nu există suficienţii operatori pentru a face toate cele 4 operaţii odată. (Exemplu: Dacă spălatul şi călcatul se fac manual, trebuie cel puţin 2 operatori.)
- La procesarea programului cu `lw`-uri ($k \geq 4$), o fază de pipeline poate conţine *acces la memorie* atât pentru *date*, cât şi pentru *instrucţiuni*. Pentru a nu avea hazard structural ar trebuie să avem *2 memorii* (ori una mai complexă cu multiple operaţii paralele de acces).



..Proiectarea unui pipeline

Hazard de control:

- *Hazardul de control* apare când trebuie decis ceva *bazându-ne pe rezultatul unei instrucţiuni* în timp ce *alte instrucţiuni se execută*.
- In exemplu cu rufe, să zicem că vrem să ajustăm detergentul în funcţie de cât de bine este spalat un set de rufe.

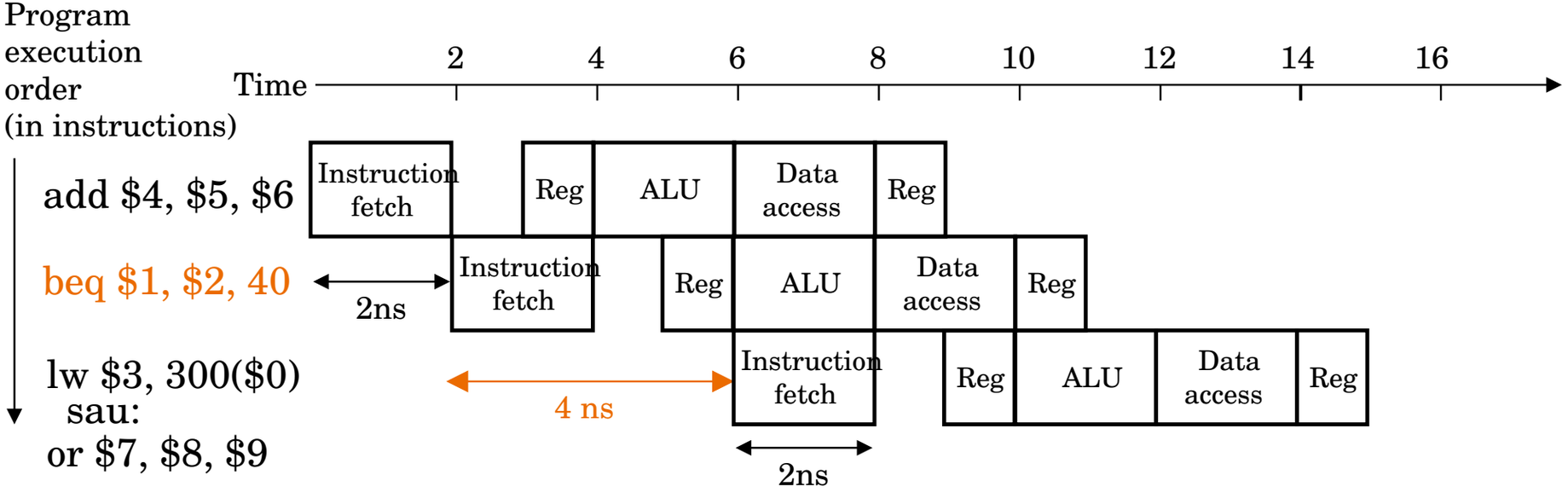
Soluţii:

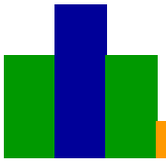
- *Intârzieri* (engl. *stalls*): Intârziem procesarea setului 2 până ce setul 1 este spălat şi uscat. (Corect, dar lent.)
- *Predicţii*: Continuăm spalarea setului 2 *cu speranţa* că detergentul este bun. Dacă după uscare constatăm că nu a fost bun, alegem alt detergent, dar trebuie *respălate* seturile 1,2. (Nu blochează pipeline-ul, dar o predicţie greşită este costisitoare.)

..Proiectarea unui pipeline

Intârzieri:

- Figura descrie un *hazard de control*, pentru instrucţiunea condiţională `beq` rezolvat cu *tehnica de întârziere*.
- Pentru a diminua întârzierea, rezultatul testului din ALU (ieşirea `zero`) se foloseşte *în acelaşi ciclu* pentru a încărca noua instrucţiune. (Normal, trebuie o întârziere de 3 cicluri.)





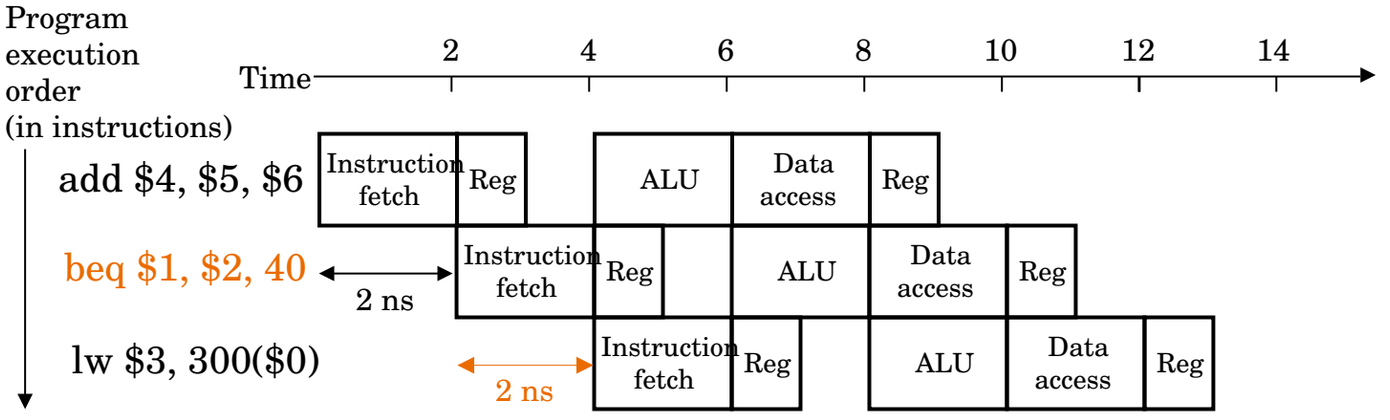
..Proiectarea unui pipeline

Predicţii:

- Alternativ, spre a nu întârzia pipeline-ul, se pot folosi *predicţii* relativ la *ce ramură* a instrucţiunii condiţionale *va fi urmată*.
- Uneori se poate lua o simplă *decizie statică*: la o instrucţiune "loop" ne aşteptăm ca bucla să fie urmată cu o frevenţă mare, deci îi vom acorda o probabilitate corespunzătoare.
- Alteori putem folosi o *decizie dinamică*, anume contorizăm frecvenţa de acceptare a fiecărei ramuri şi o folosim pentru a prezice ramura care va fi urmată în instrucţiunea curentă.
- *Costul unei predicţii greşite* poate fi ridicat, mai ales în cazul în care pipeline-ul are mai multe faze.

..Proiectarea unui pipeline

ramificaţie neacceptată



ramificaţie acceptată

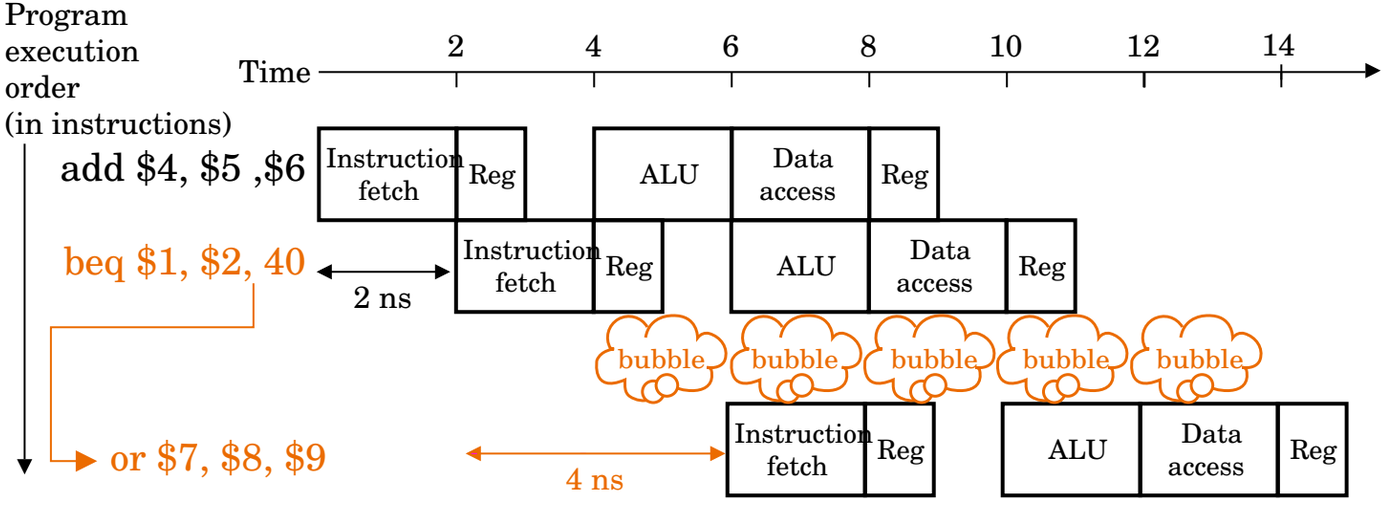
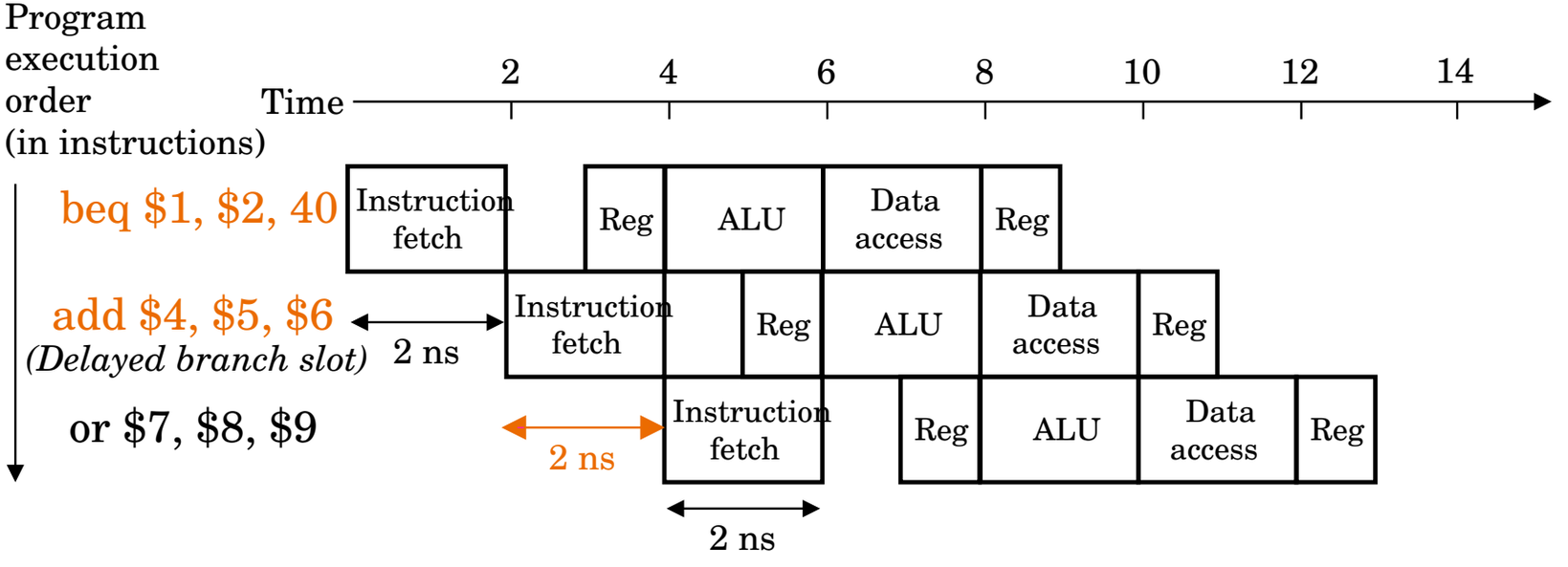


Figura descrie acelaşi *hazard de control* pentru instrucţiunea condiţională beq rezolvat cu *tehnica de predicţie*.

..Proiectarea unui pipeline

Decizii intarziate:

- Se poate evita blocarea pipeline-ului prin *decizii întârziate*: *aşteptăm* ca o decizie să fie luată *executând alte sarcini*.
- De regulă, compilatoarele acoperă circa 50% din astfel de sloturi de aşteptare cu instrucţiuni utile.
- In exemplul anterior, putem muta în slotul de aşteptare instrucţiunea anterioară, evitând blocarea - vezi figura.



..Proiectarea unui pipeline

Hazard de date:

- *Hazardul de date* este generat de interdependenţa instrucţiunilor realizată prin intermediul datelor (variabilelor).
- Exemplu: O instrucţiune poate necesita rezultatul unei instrucţiuni anterioare, care este încă în pipeline, ca în exemplul

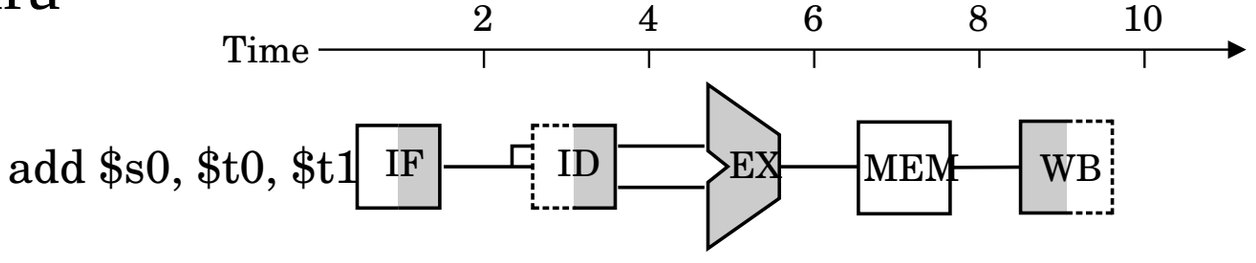
```
add $s0,$t0,$t1;
sub $t2,$s0,$t3;
```

- O *soluţie* de a evita întârzierea pipeline-ului este *avansarea* (*forwarding / bypassing*), tehnică prin care *data necesară* într-o instrucţiune următoare se foloseşte *imediat ce a fost obţinută* în instrucţiunea curentă.

..Proiectarea unui pipeline

Hazard de date (cont.)

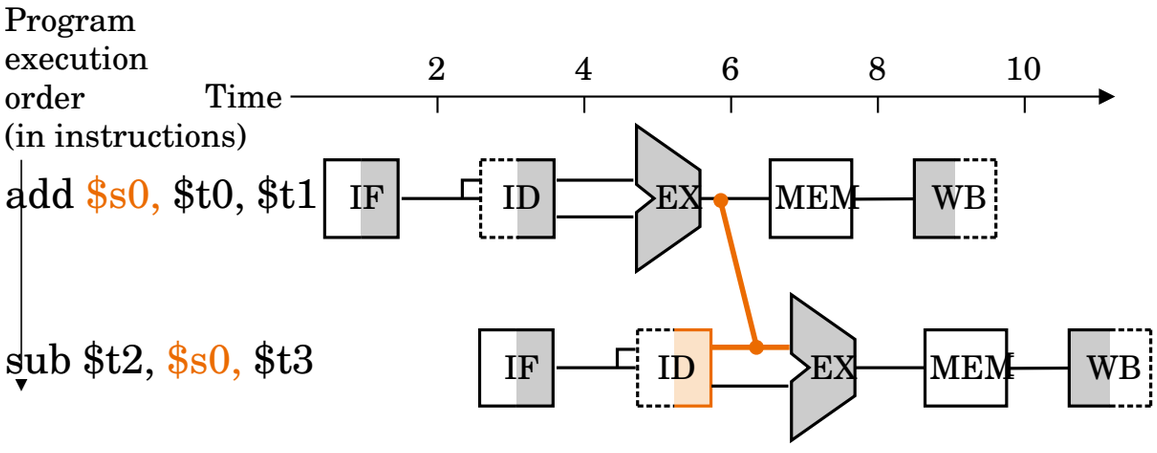
- Se poate folosi tehnica de *avansare* numai dacă *timpul* la care apare *data sursă* este *anterior* celui din faza de *utilizare*.
- Pentru o analiza mai fină, putem împărţi fazele unei procesări ca în figură



- Fazele sunt: *IF* - extragere instrucţiune; *ID* - decodare instrucţiune; *EX* - execuţie în ALU; *MEM* - acces memorie; *WB* - scriere în registru (engl. “write back”).
- *Umbrele* indică dacă *componenta este utilizată* în procesarea curentă; umbririle pe jumătate sunt pentru acces memorii, cu *scriere* pe *prima jumătate* şi *citire* pe *a doua*.

..Proiectarea unui pipeline

hazard rezolvat cu avansare



hazard rezolvat cu avansare şi întârziere

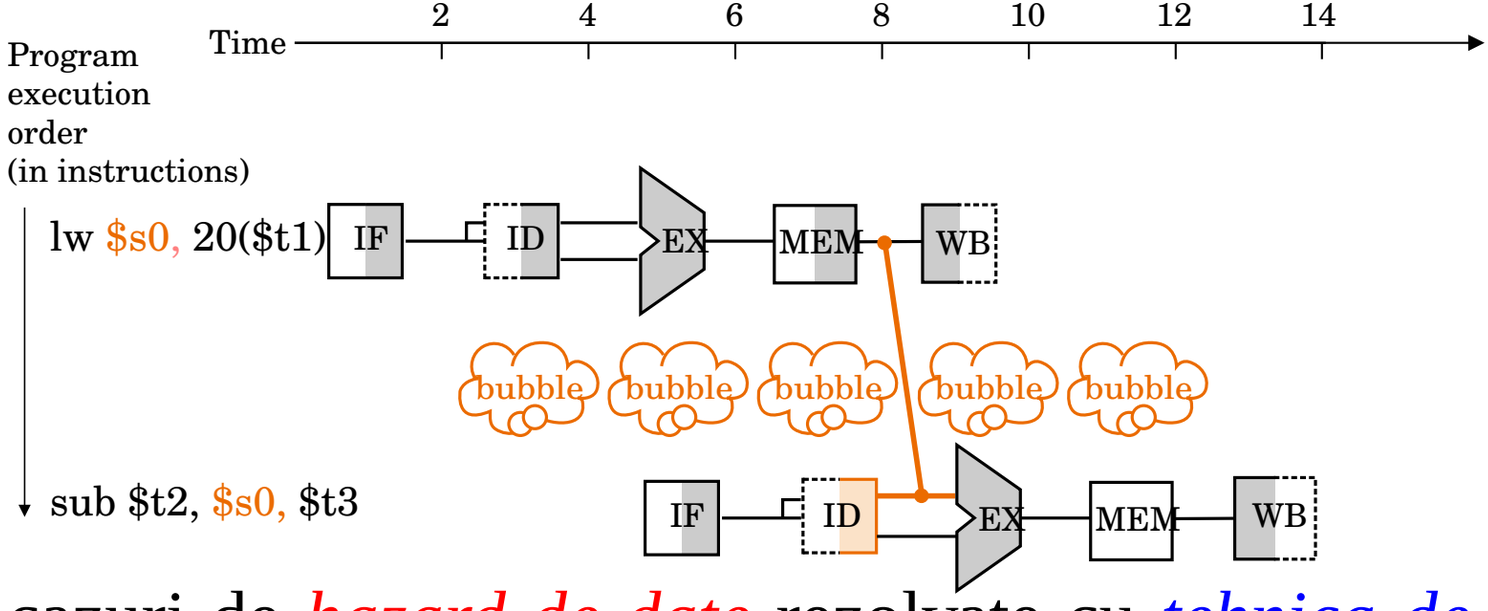
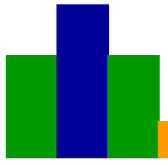


Figura descrie 2 cazuri de *hazard de date* rezolvate cu *tehnica de avansare*. Jos, chiar şi cu avansare, pipeline-ul trebuie întârziat.



..Proiectarea unui pipeline

Rearanjarea codului:

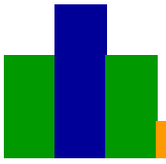
- O tehnică generală de *evitare a hazardul* este *reordonarea instrucţiunilor*, fără a modifica funcţia calculată.

- *Exemplu*: Permutarea a 2 variabile se poate face cu codul din stânga, care are o întârziere (la trecerea de la `lw` la `sw`).

```
lw $t0,0($t1)
lw $t2,4($t1)
sw $t2,0($t1)
sw $t0,4($t1)
```

```
lw $t0,0($t1)
lw $t2,4($t1)
sw $t0,4($t1)
sw $t2,0($t1)
```

Rearanjând codul ca în dreapta întârzierea dispare.

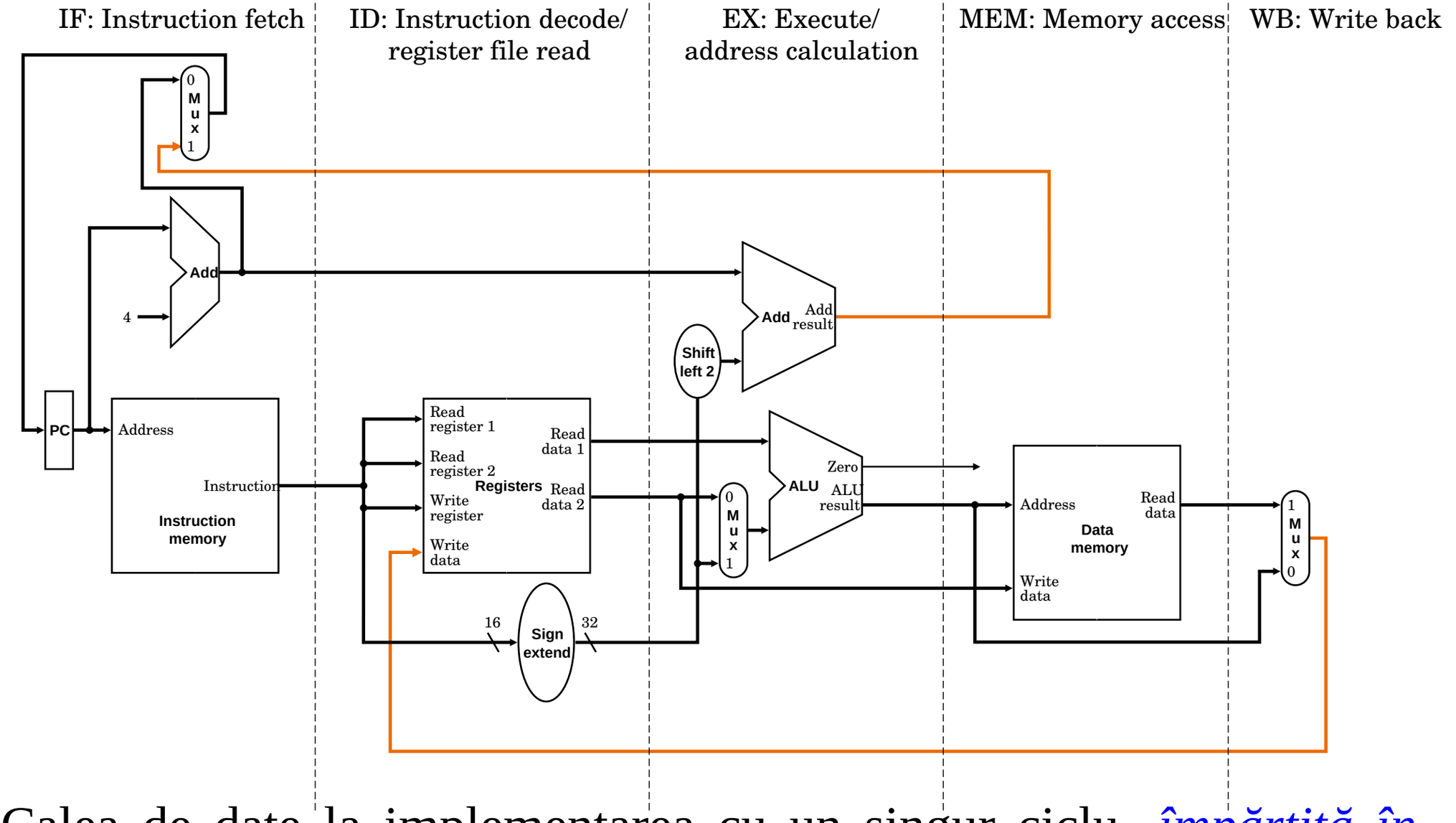


Procesorul: Tehnica de pipeline

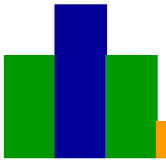
Cuprins:

- 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.

Implementarea cu un ciclu, in faze



Calea de date la implementarea cu un singur ciclu, *împărţită în faze*. (In roşu sunt arce de "întoarcere" care complică pipeline-ul.)



Calea de date cu pipeline

Primii pasi

- Ca anterior, putem împărţi implementarea cu un ciclu în 5 faze:

  IF - extragere instrucţiune;
  ID - decodare instrucţiune;
  EX - execuţie în ALU;
  MEM - acces memorie;
  WB - scriere în registru

- Toate conexiunile sunt *de la stînga la dreapta*, cu *2 excepţii*:
  - scris PC în caz de instrucţiune condiţională;
  - scris în registru (la load);

  Aceste excepţii produc *hazarduri*: prima de control, a doua de date.

..Calea de date cu pipeline

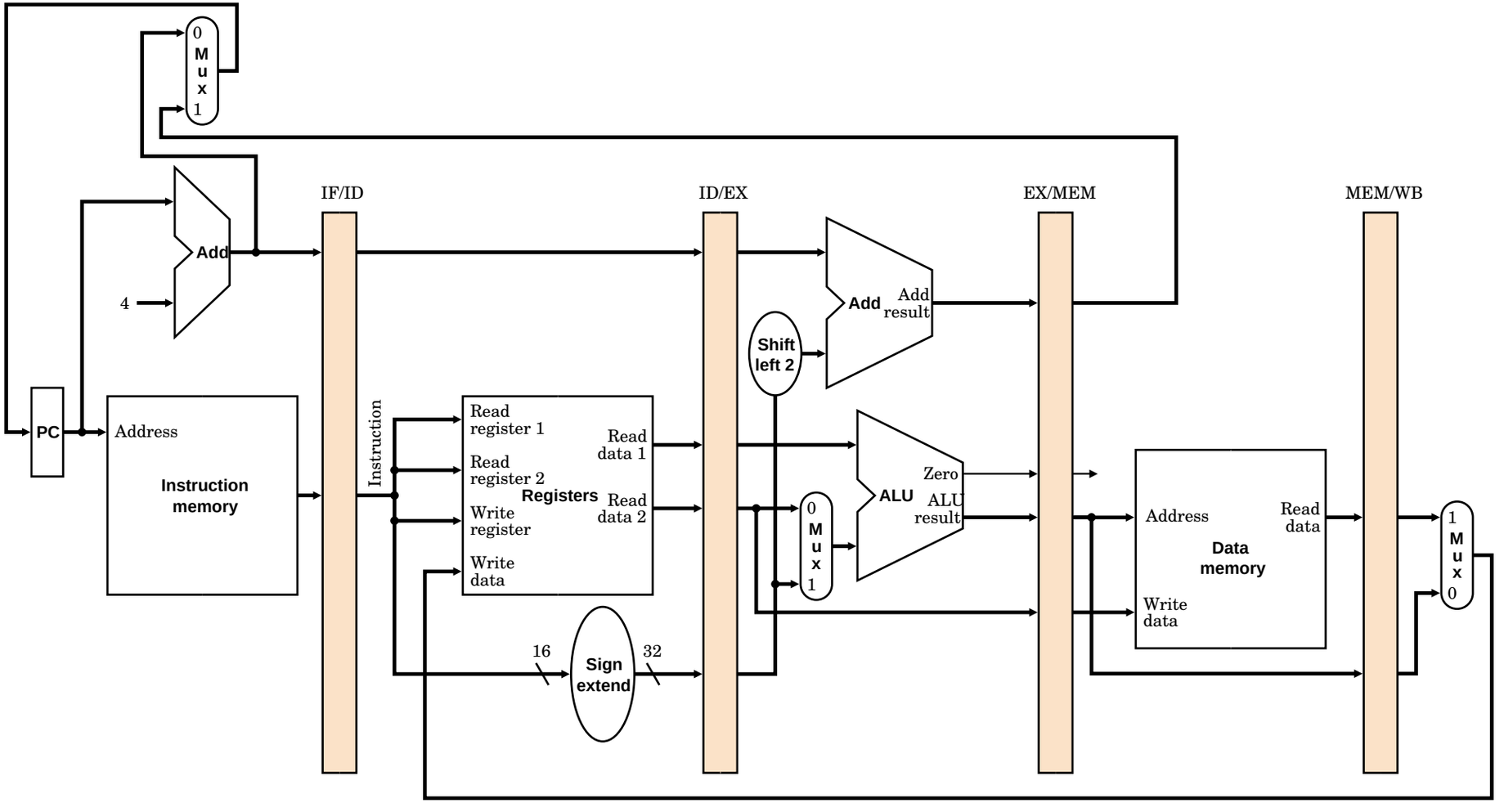
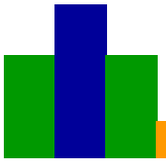


Figura descrie un *pipeline* pentru *implementarea cu un singur ciclu*. Se introduc *regiştri specifici* care să reţină datele între fazele de pipeline: `IF/ID, ID/EX, EX/MEM, MEM/WB`.



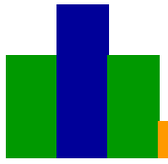
..Calea de date cu pipeline

Exemplu - load:

Ilustrăm fazele de pipeline cu procesarea instrucţiunii *load* (vezi desenele următoare):

- *IF - extragere instrucţiune:* Se extrage instrucţiunea, se calculează noul `PC` (pentru instrucţiunea următoare $\texttt{PC}+4$). Se reţin în registrul `IF/ID` instrucţiunea şi $\texttt{PC}+4$ (ultimul, pentru branch).

- *ID - decodare instrucţiune şi citire regiştri:* Instrucţiunea din `IF/ID` se foloseşte pentru a citi 2 regiştri şi a face extensia deplasării la 32b. Cele 3 date obţinute se reţin în registrul `ID/EX` (şi $\texttt{PC}+4$, dar inutilă acum).



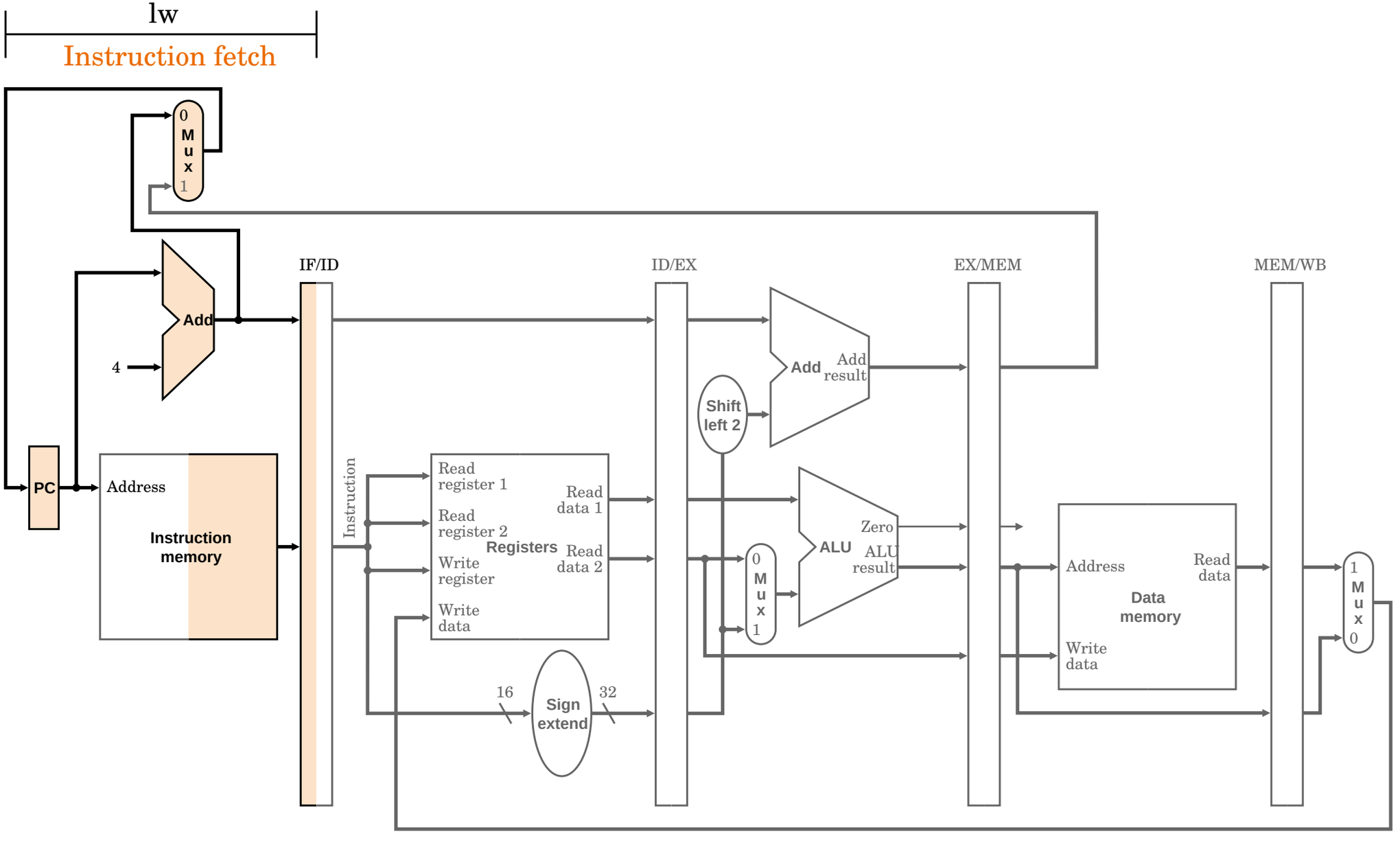
..Calea de date cu pipeline

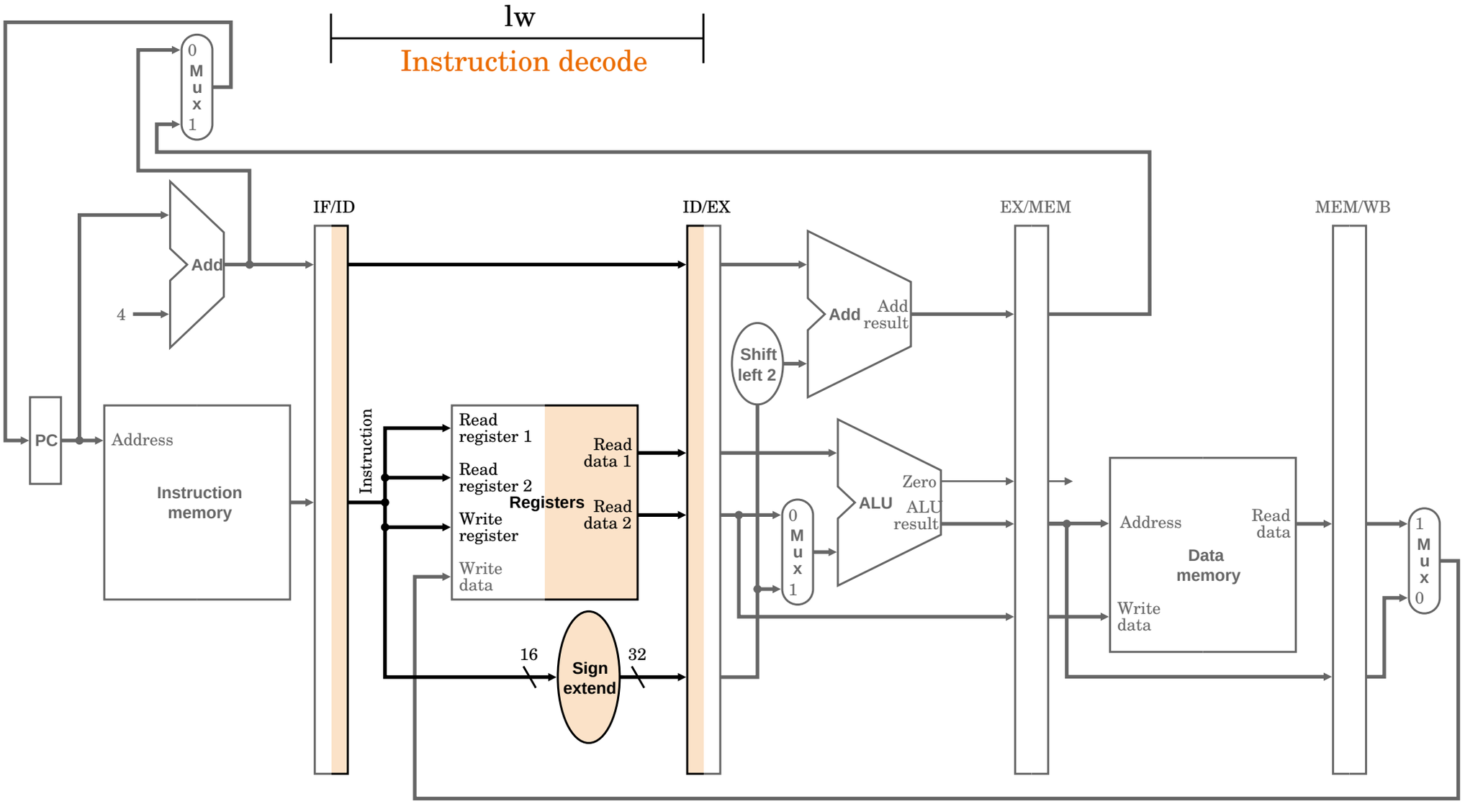
Exemplu - load (cont.)

- *EX - execuţie:* Folosind valoarea registrului 1 şi deplasarea (reţinute în `ID/EX`), calculăm în ALU adresa de memorie, pe care o punem în registrul `EX/MEM`.
- *MEM - acces memorie:* Se accesează memoria cu adresa de memorie din registrul `EX/MEM`, rezultatul reţinându-se în `MEM/WB`.
- *WB - scriere în registru:* Valoarea citită din memorie, aflată în `MEM/WB` se scrie în registrul 2 (specificat în instrucţiune)!

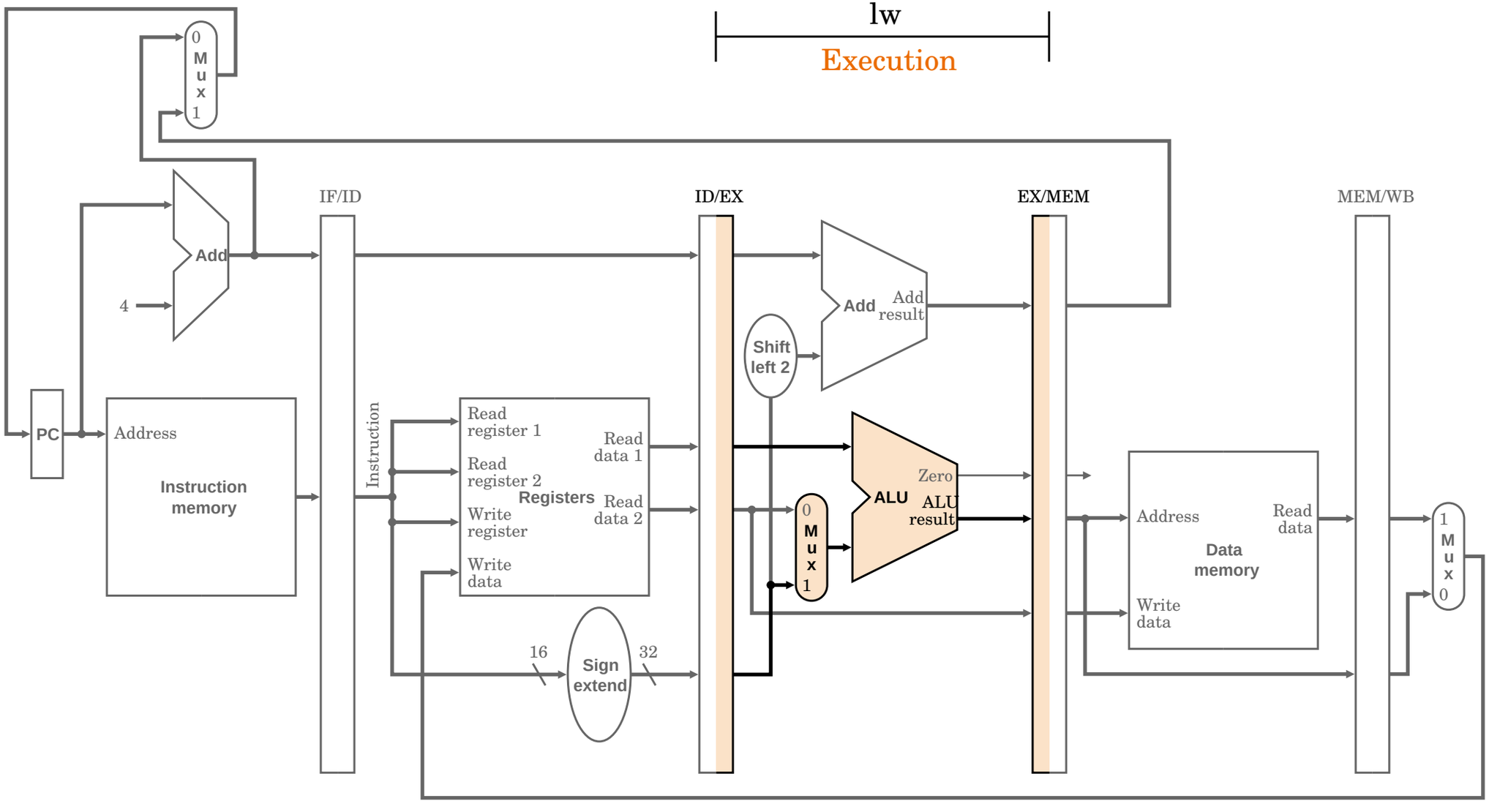
Fazele de execuţie în *pipeline* pentru instrucţiunea *load*:

Faza 1: IF (extragere instrucţiune)



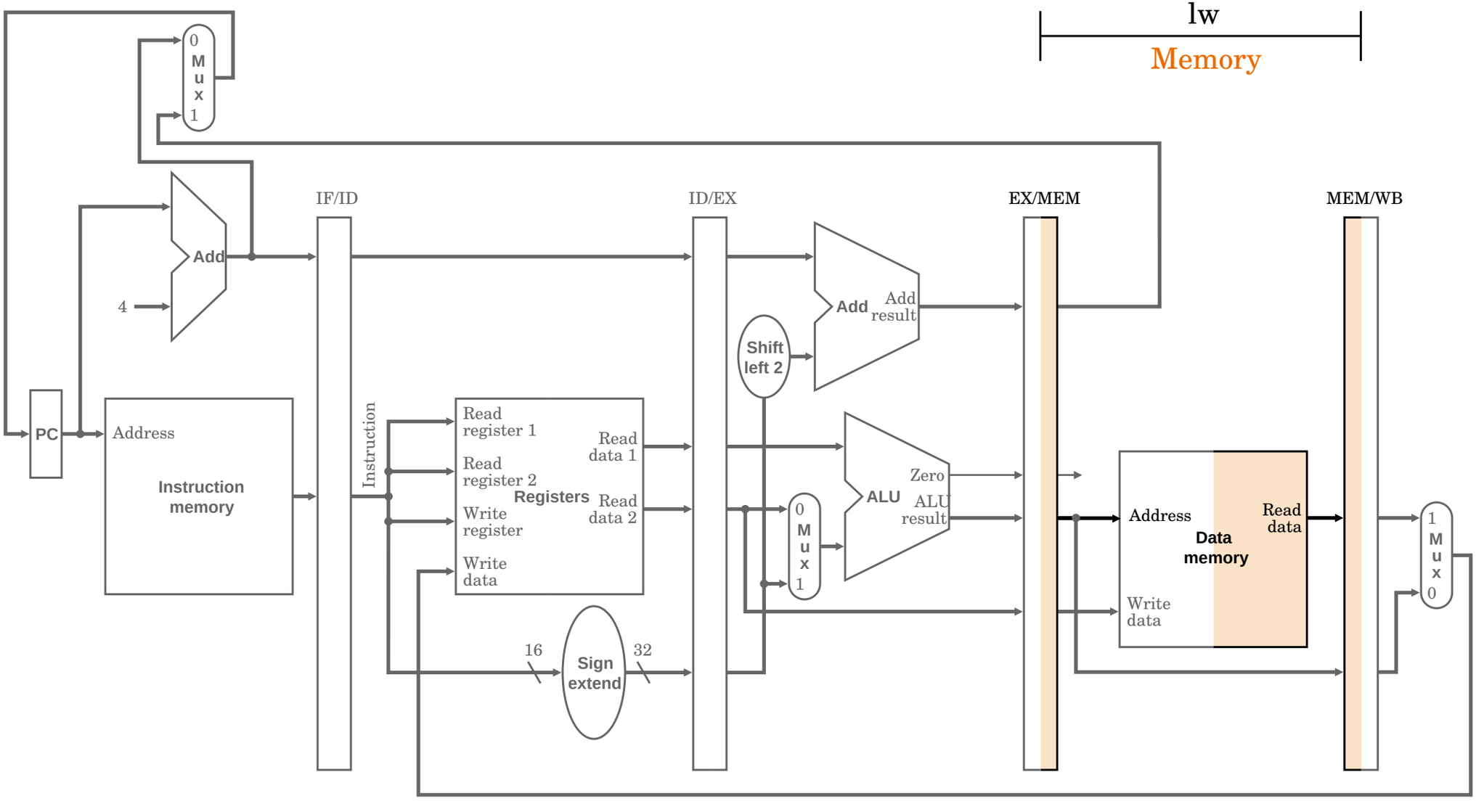


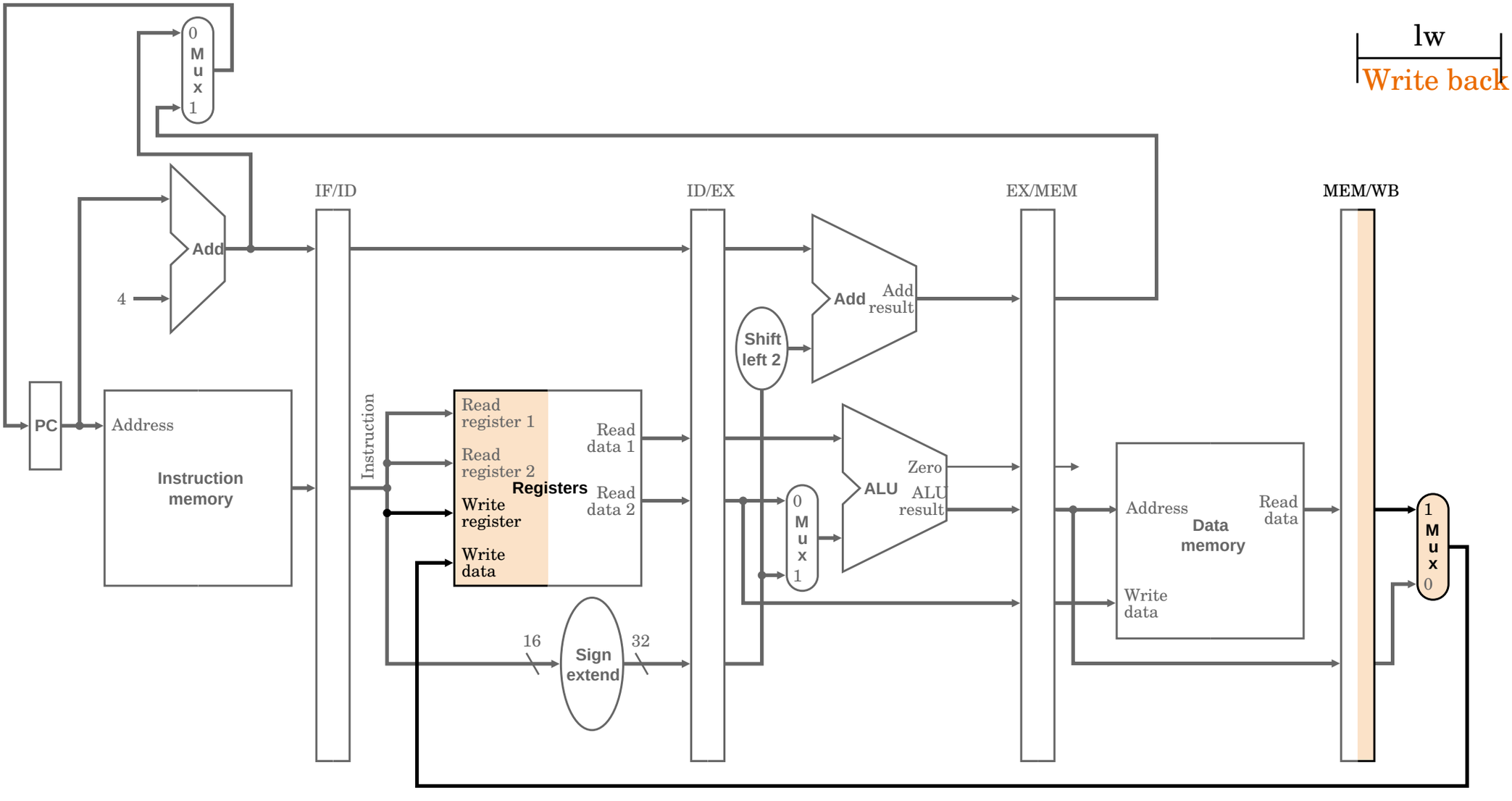
Faza 2: ID (decodare instrucţiune)



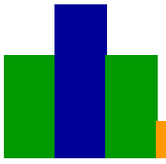
Faza 3: EX (execuţie)

Faza 4: MEM (acces memorie)





Faza 5: WB (scriere în registru)



..Calea de date cu pipeline

Comentarii:

- In fiecare fază trebuie salvate în regiştri speciali de pipeline *toate* datele necesare fazelor următoare.
- In versiunea ilustrată, arcul de feedback din ultima fază `WB` reliefează *o eroare* de referinţă: *adresa registrului* de scriere trebuia *propagată* prin pipeline.
- Figura care urmează prezintă o versiune corectată pentru load.
- Se pot face analize similare pentru trecerea prin pipeline a altor instrucţiuni: store, add, etc.

..Calea de date cu pipeline

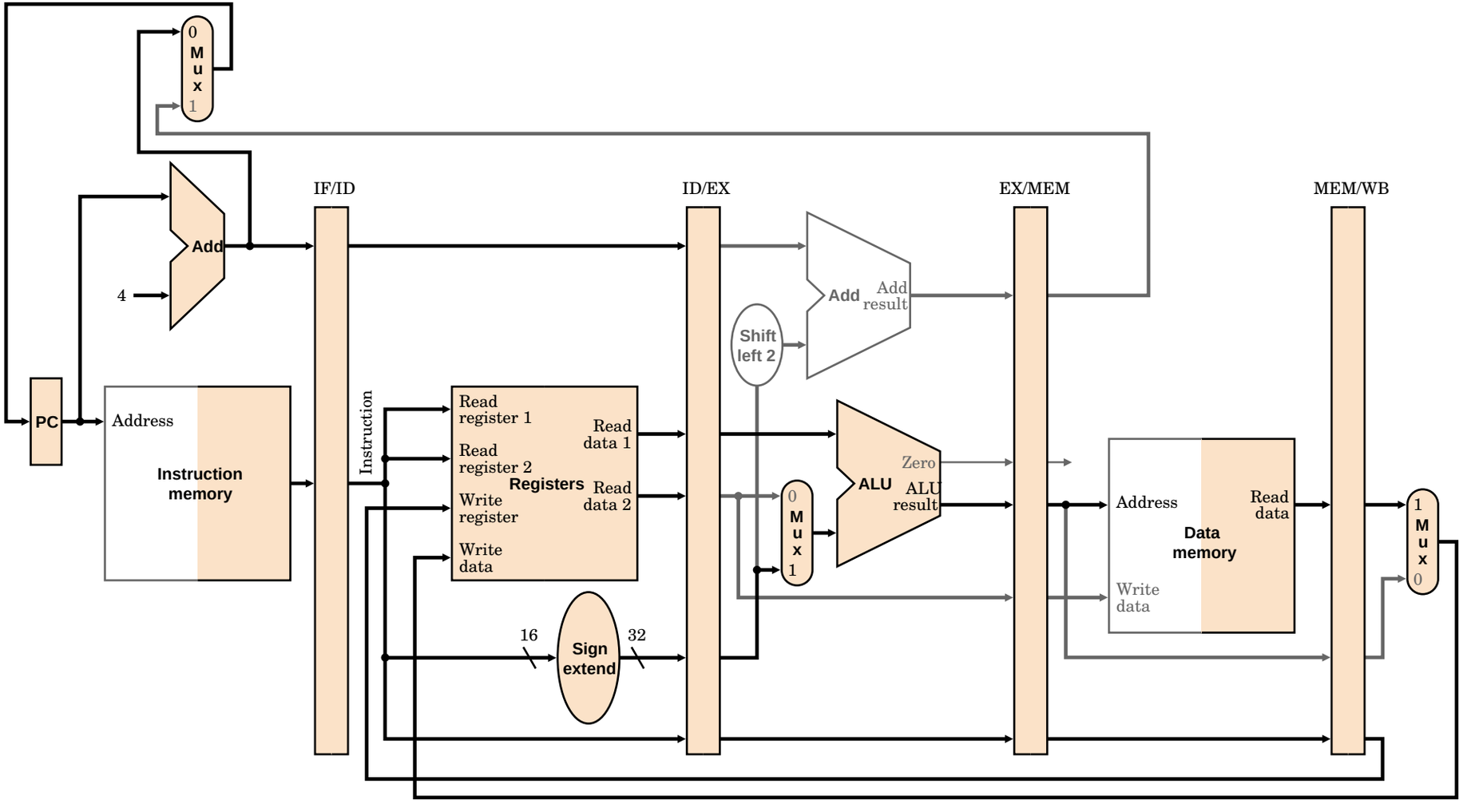


Figura descrie un *pipeline* (corect) pentru implementarea instrucţiunii *load*. Am inserat conexiunea de propagare a *adresei registrului de scriere*.

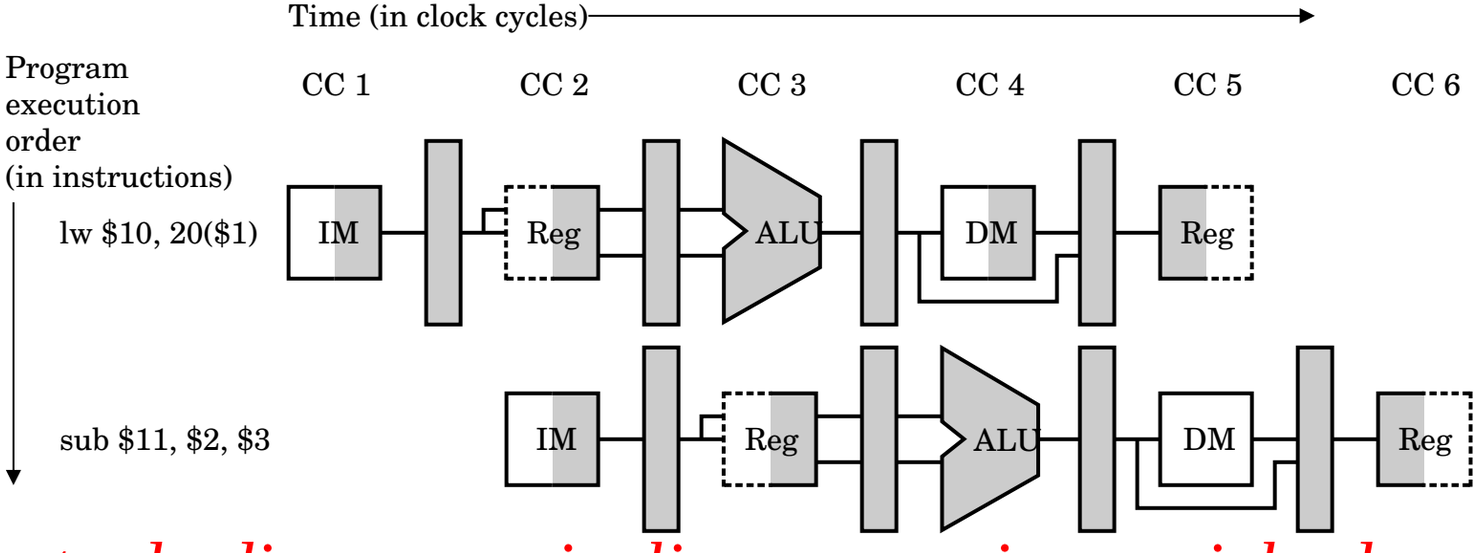
Multiple instructiuni in pipeline

Multiple instructiuni in pipeline:

- Ilustrăm trecerea a 2 instrucţiuni prin pipeline

```
lw $10,20($1);
sub $11,$2,$3
```

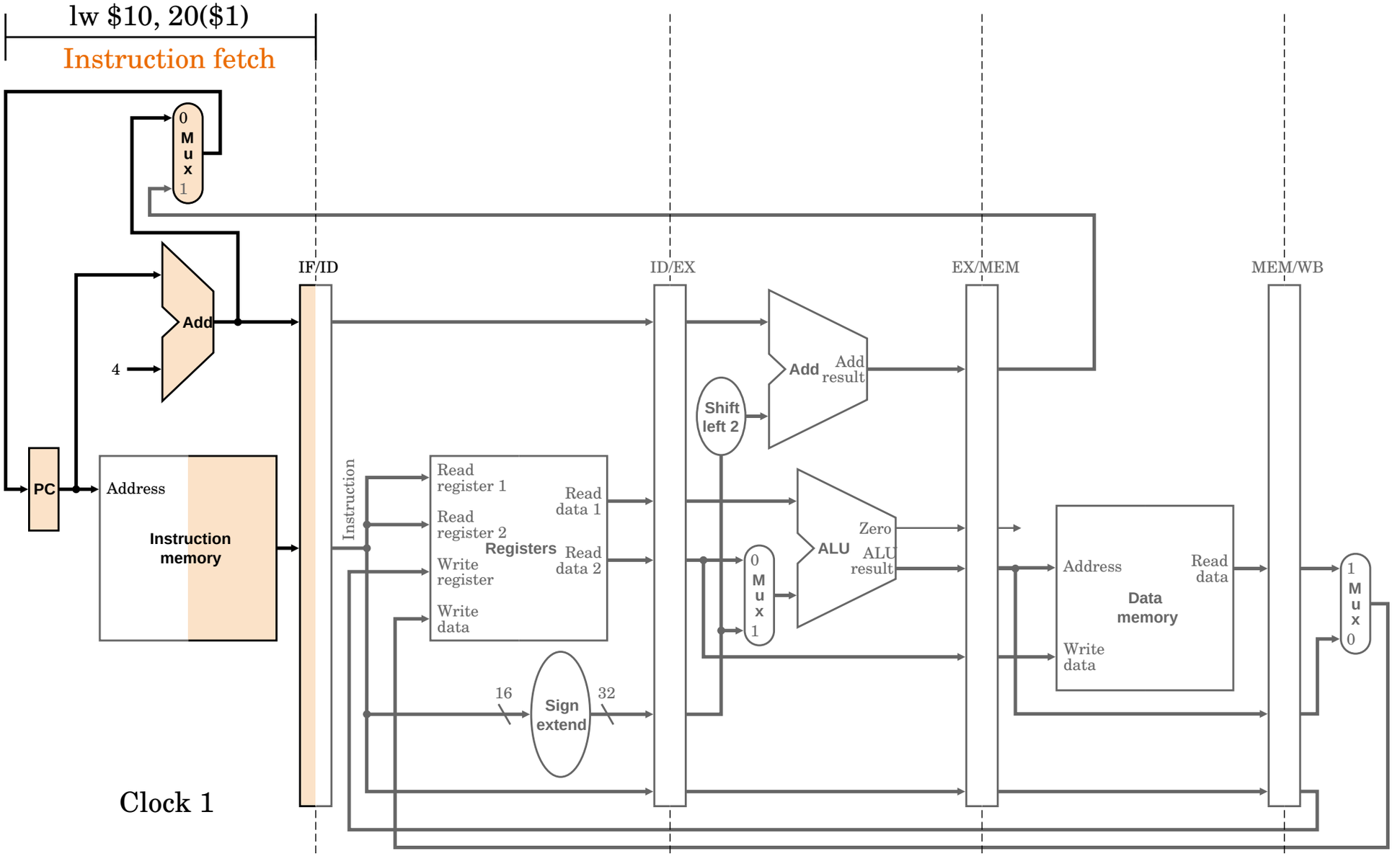
- Se pot utiliza *diagrame pipeline cu multiple cicluri de ceas* de tipul

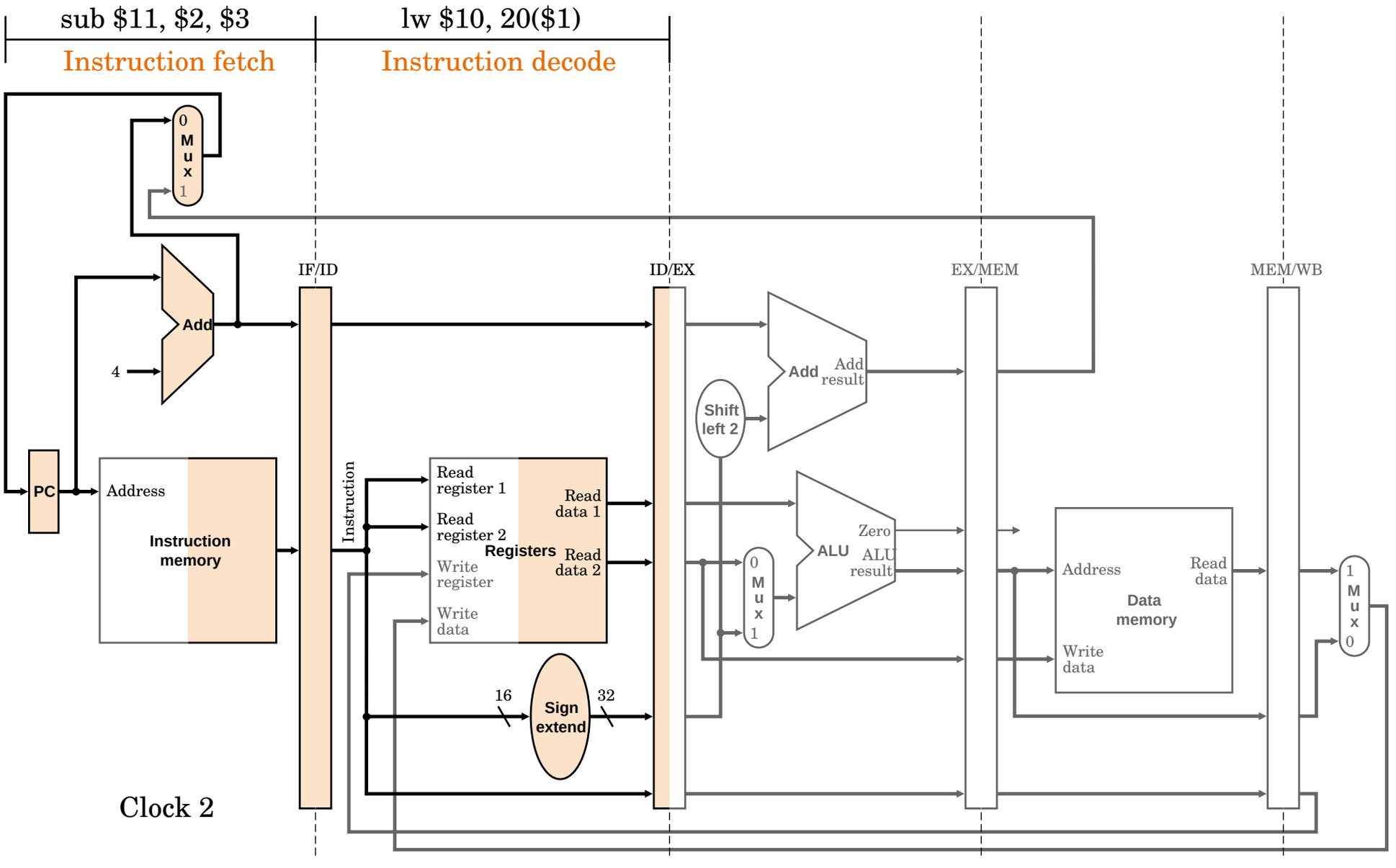


ori *secvenţe de diagrame pipeline cu un singur ciclu de ceas*, ca în figurile ce urmează.

Fazele de execuţie în *pipeline* pentru o secvenţă “*load; sub*”:

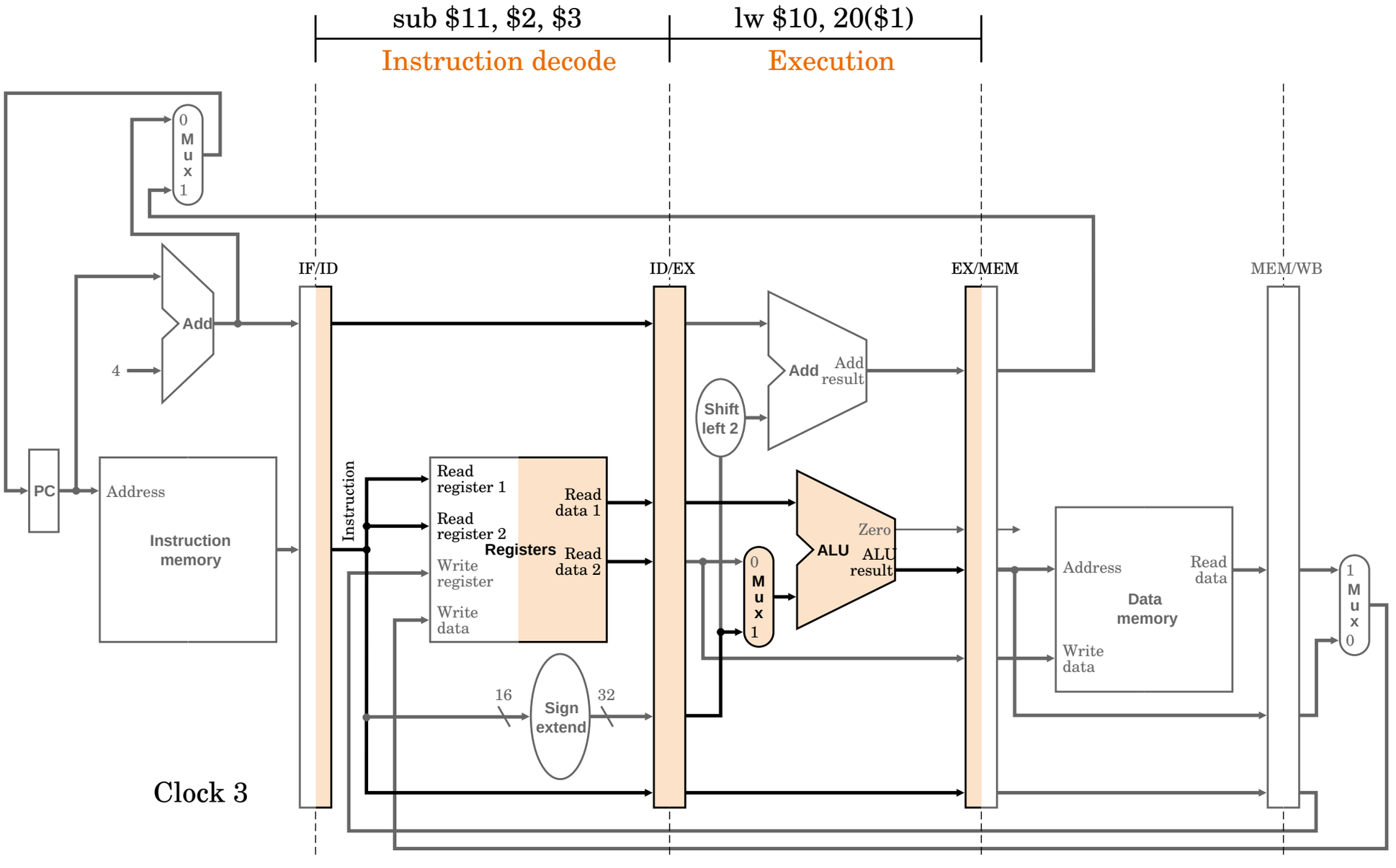
Faza 1: IF-load

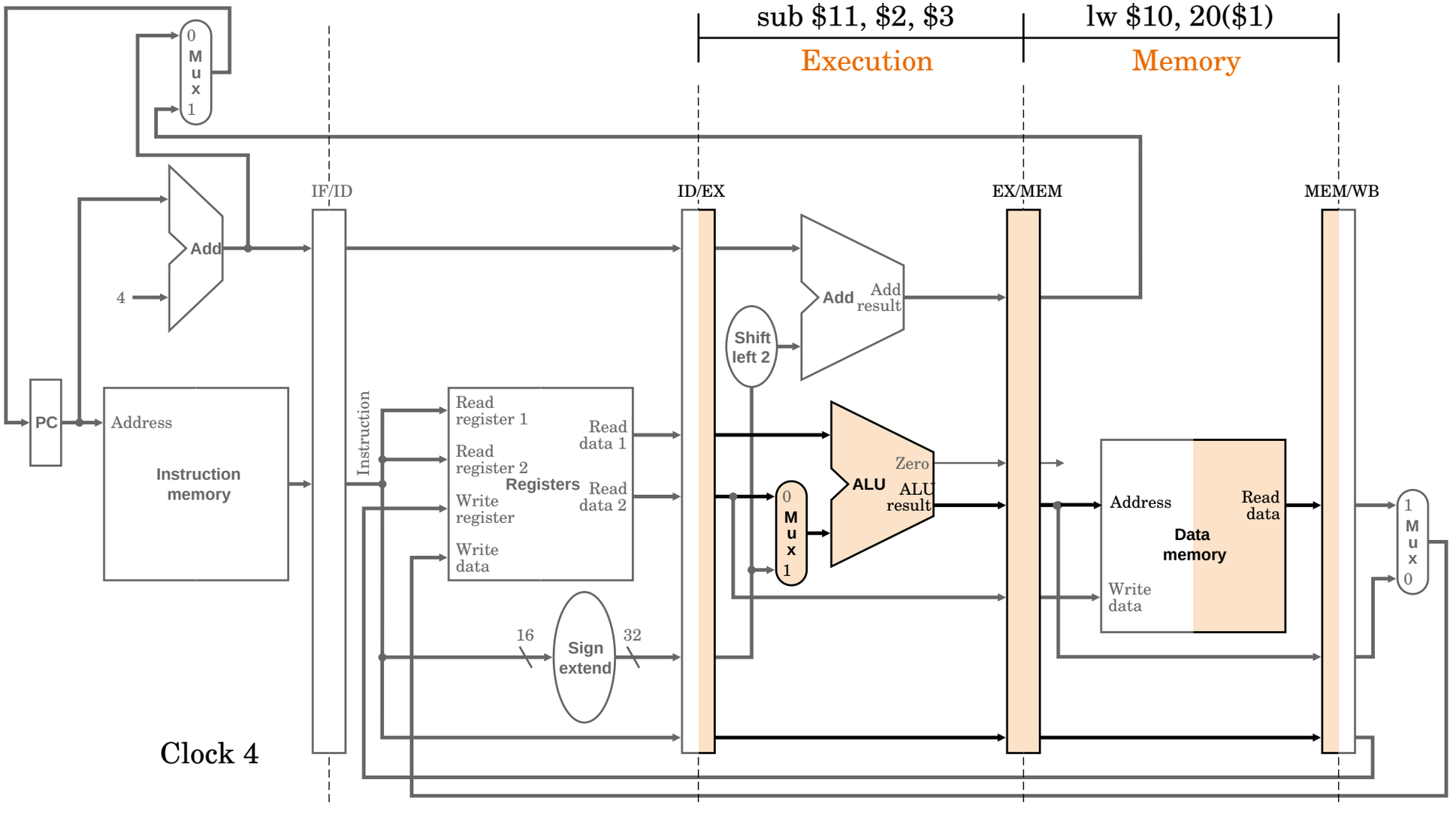




Faza 2: IF-sub & ID-load

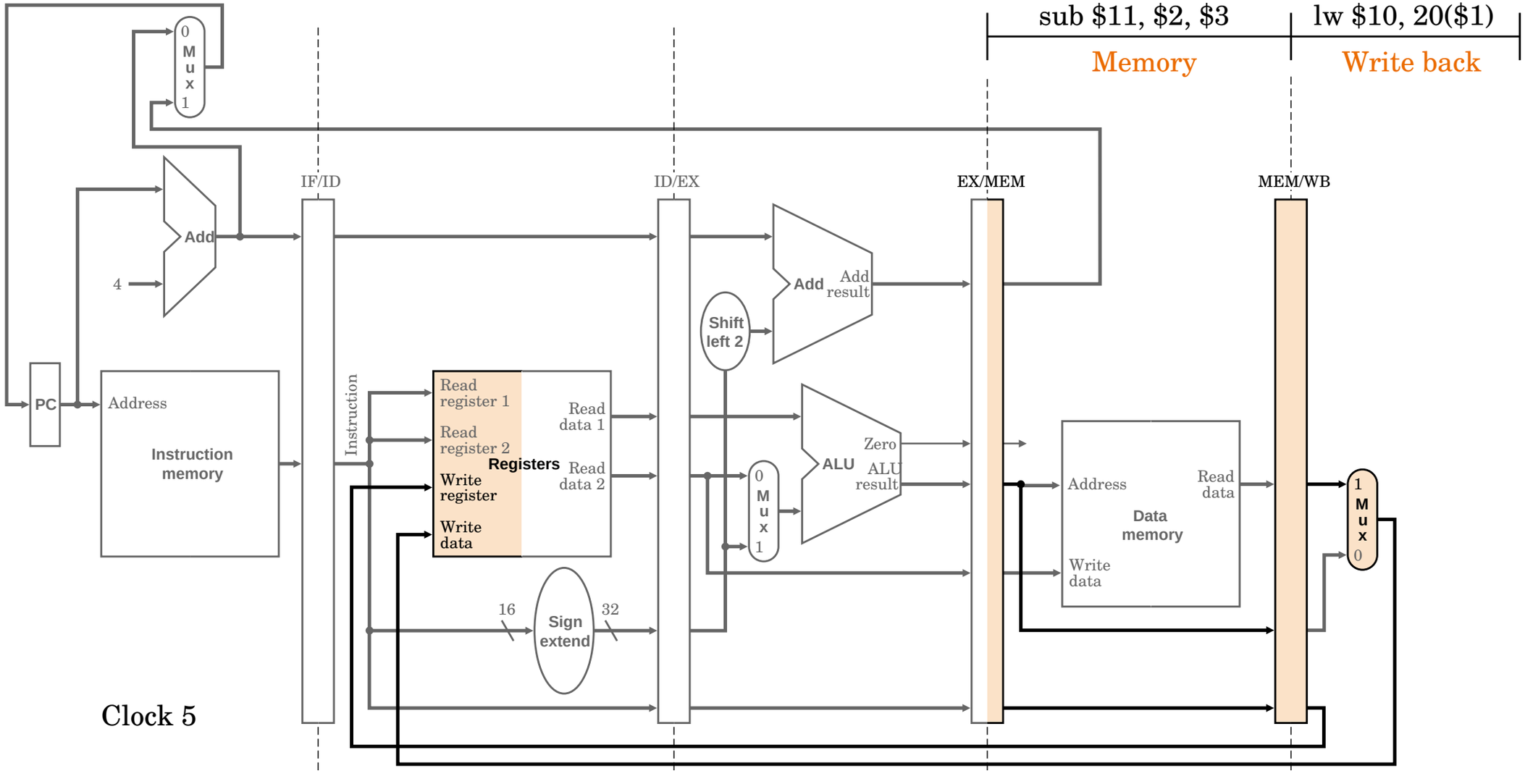
Faza 3: ID-sub & EX -load

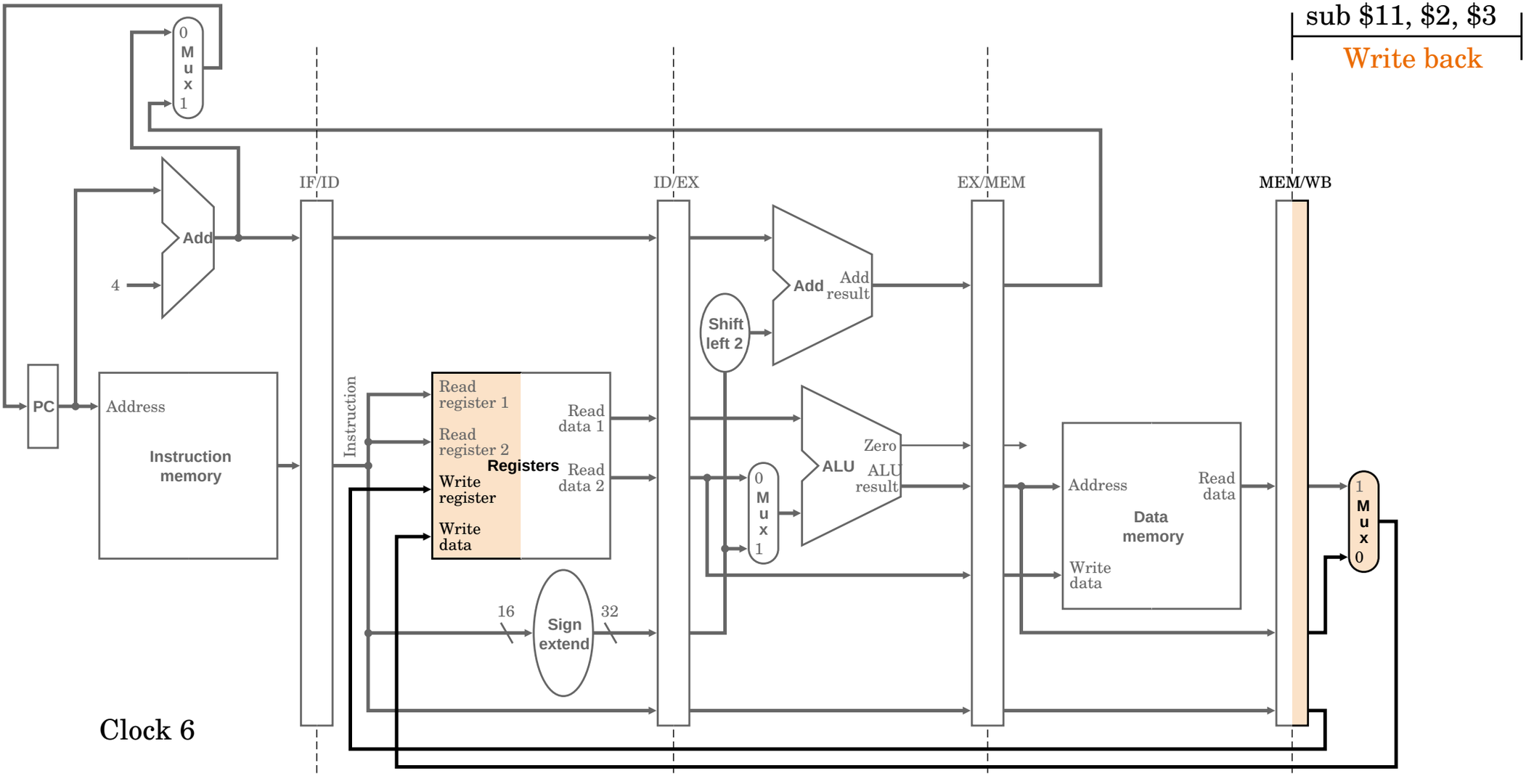




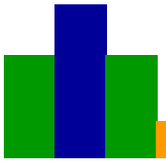
Faza 4: EX-sub & MEM-load

Faza 5: MEM-load & WB-load




sub $11, $2, $3
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign extend
32
Clock 6

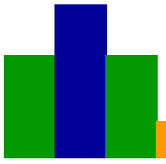
Faza 6: WB-sub



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.



Controlul pentru pipeline

Generalitati: Prezentăm schema generală de control:

- Incepem cu o versiune simplificată, *neglijând hazardurile*.
- Folosim semnale de control similare ca cele din prima implementare de procesor (cu 1 ciclu).
- Notăm că `PC` şi toţi ceilalţi regiştri de pipeline (`IF/ID`, `ID/EX`, `EX/MEM`, `MEM/WB`) se scriu în fiecare ciclu, deci *nu necesită control* special de scriere.
- Semnalele de control generate de instrucţiune se folosesc în faze pipeline distincte, deci unele trebuiesc *propagate* de la o fază la alta.

Semnale de controlul la pipeline

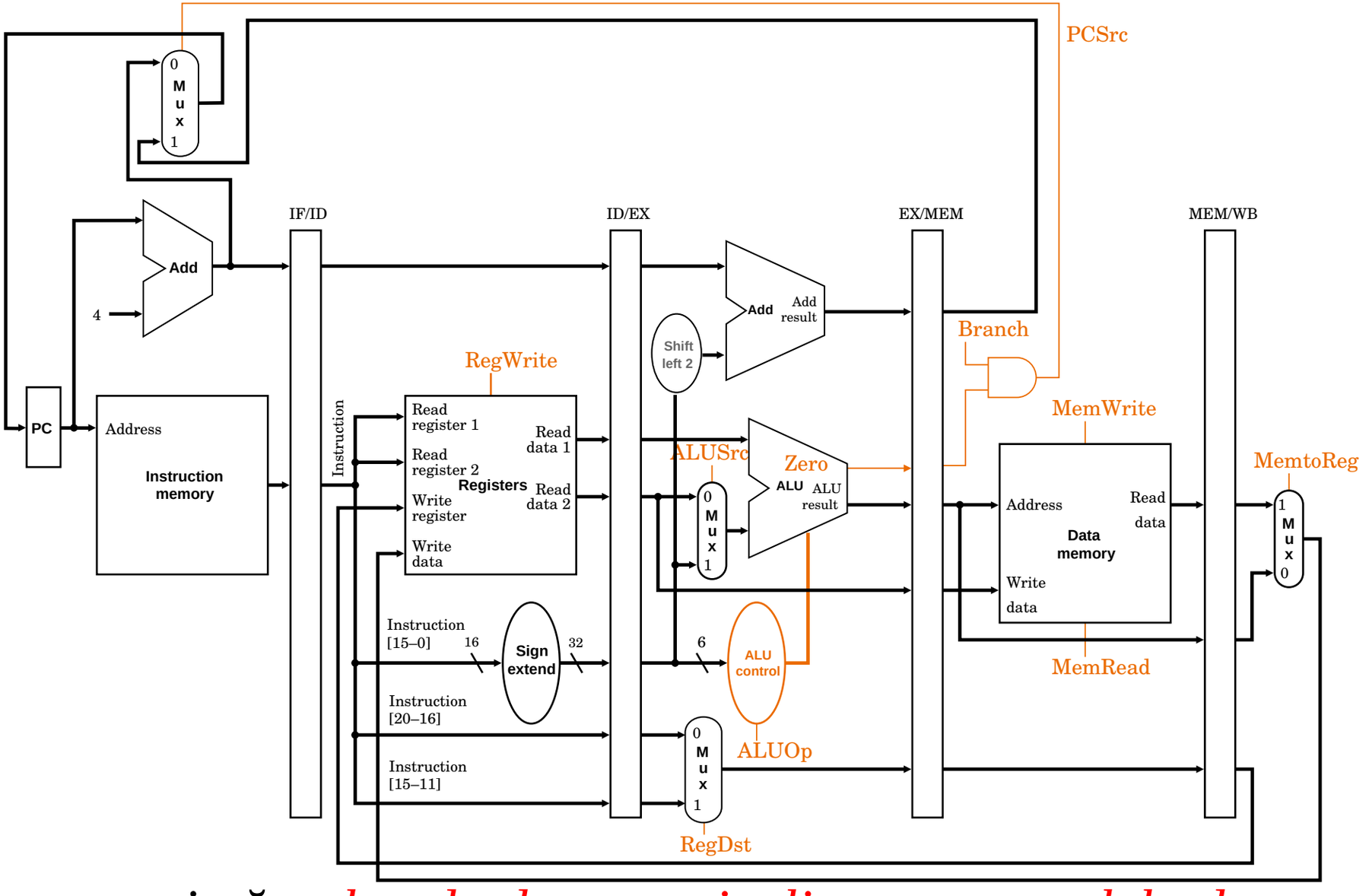


Figura prezintă *calea de date cu pipeline* cu *semnalele de control* necesare. (Versiune simplificată, ignorând hazardurile.)

Controlul cu pipeline

Semnale de control:

- Folosim semnalele de control de la implementarea cu 1 ciclu (Fila 7.41), împărţite pe faze pipeline astfel:

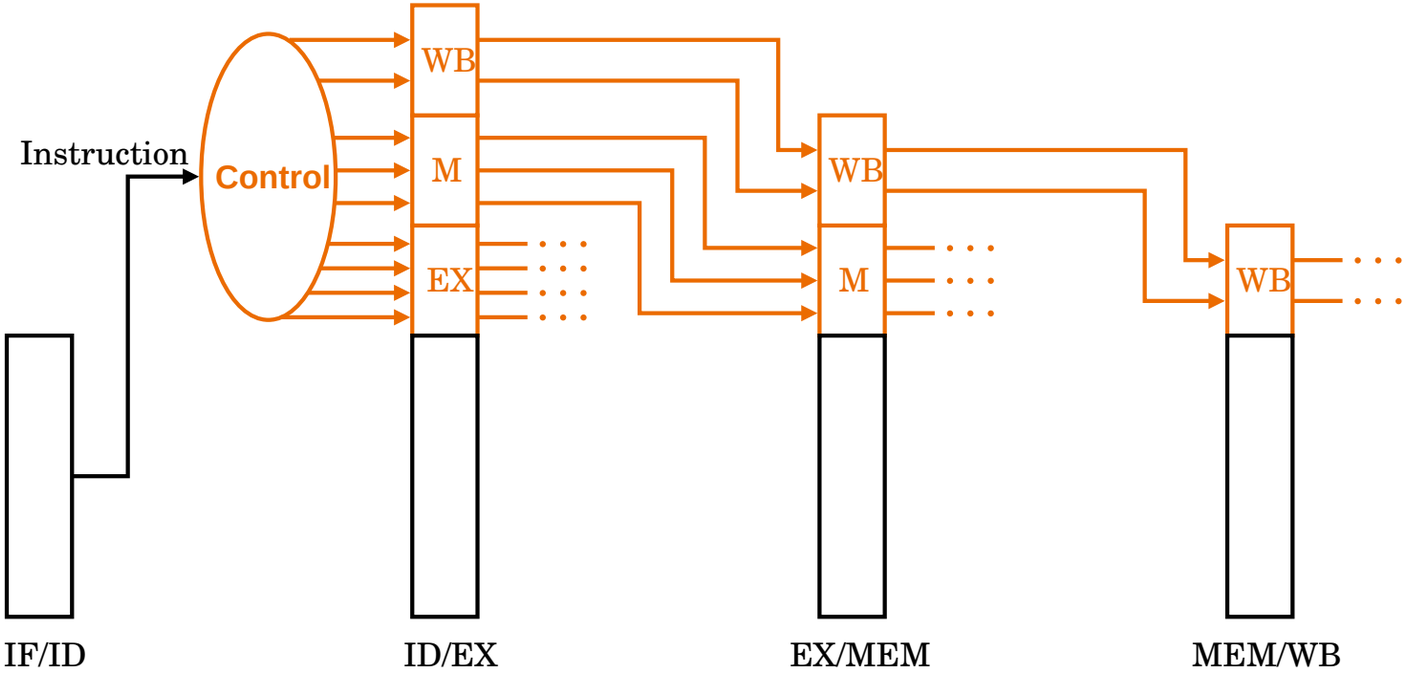
Instruc.	Executie si Calcul Adresa				Access Memorie			Scriere in Registru	
	Reg Dst	ALU Op1	ALU Op2	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem toReg
Format R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	x	0	0	1	0	0	1	0	x
beq	x	0	1	0	1	0	0	0	x

Controlul cu pipeline

Implementarea controlului:

O implementare simplă de control pentru pipeline:

- (1) folosim vechea implementare a componentei de control;
- (2) propagăm semnalele de control necesare în faze ulterioare



Semnale de controlul la pipeline

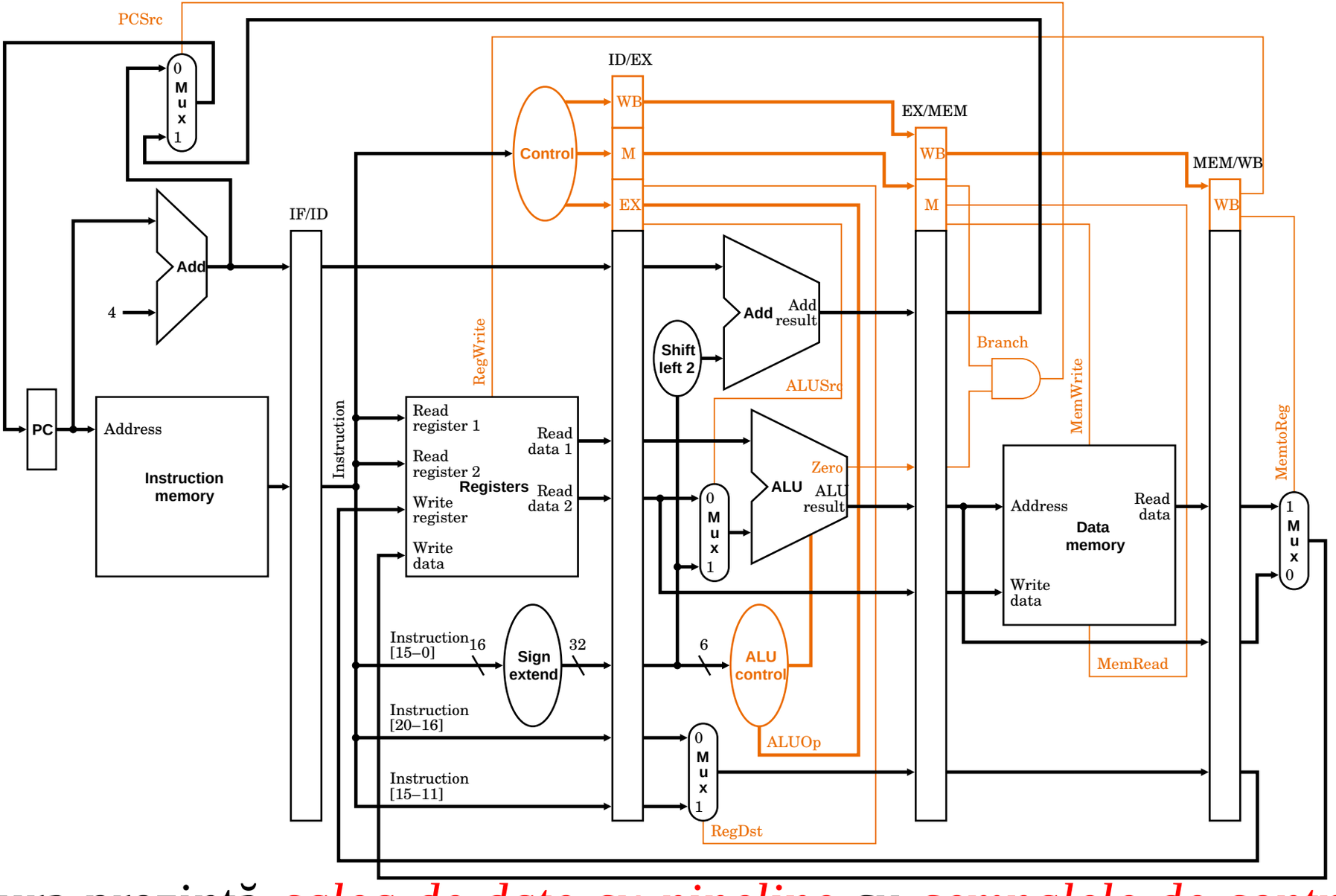
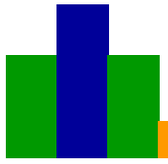


Figura prezintă *calea de date cu pipeline* cu *semnalele de control* necesare *conectate*. (Versiune simplificată, ignorând hazardurile.)



Controlul cu pipeline

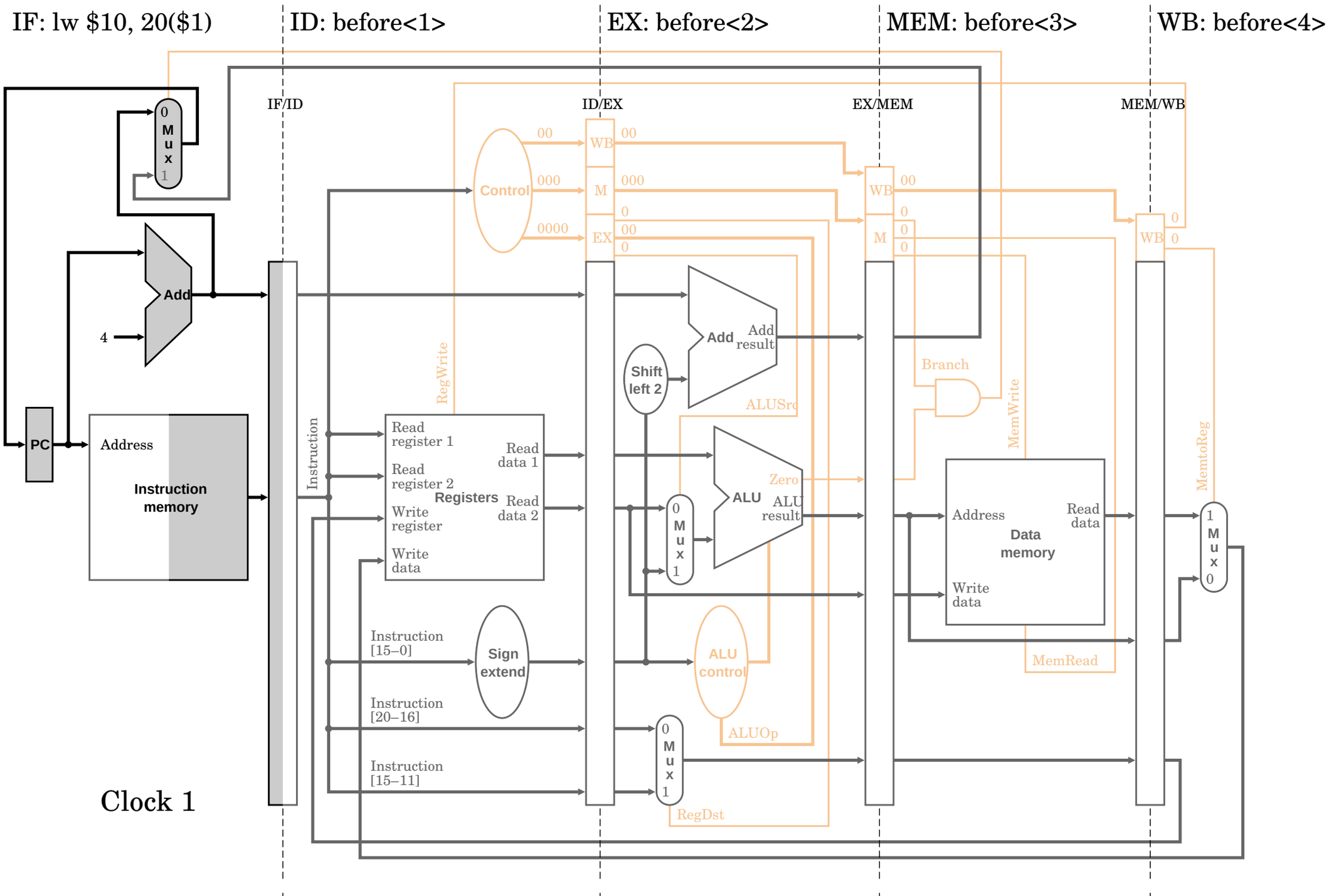
Exemplu:

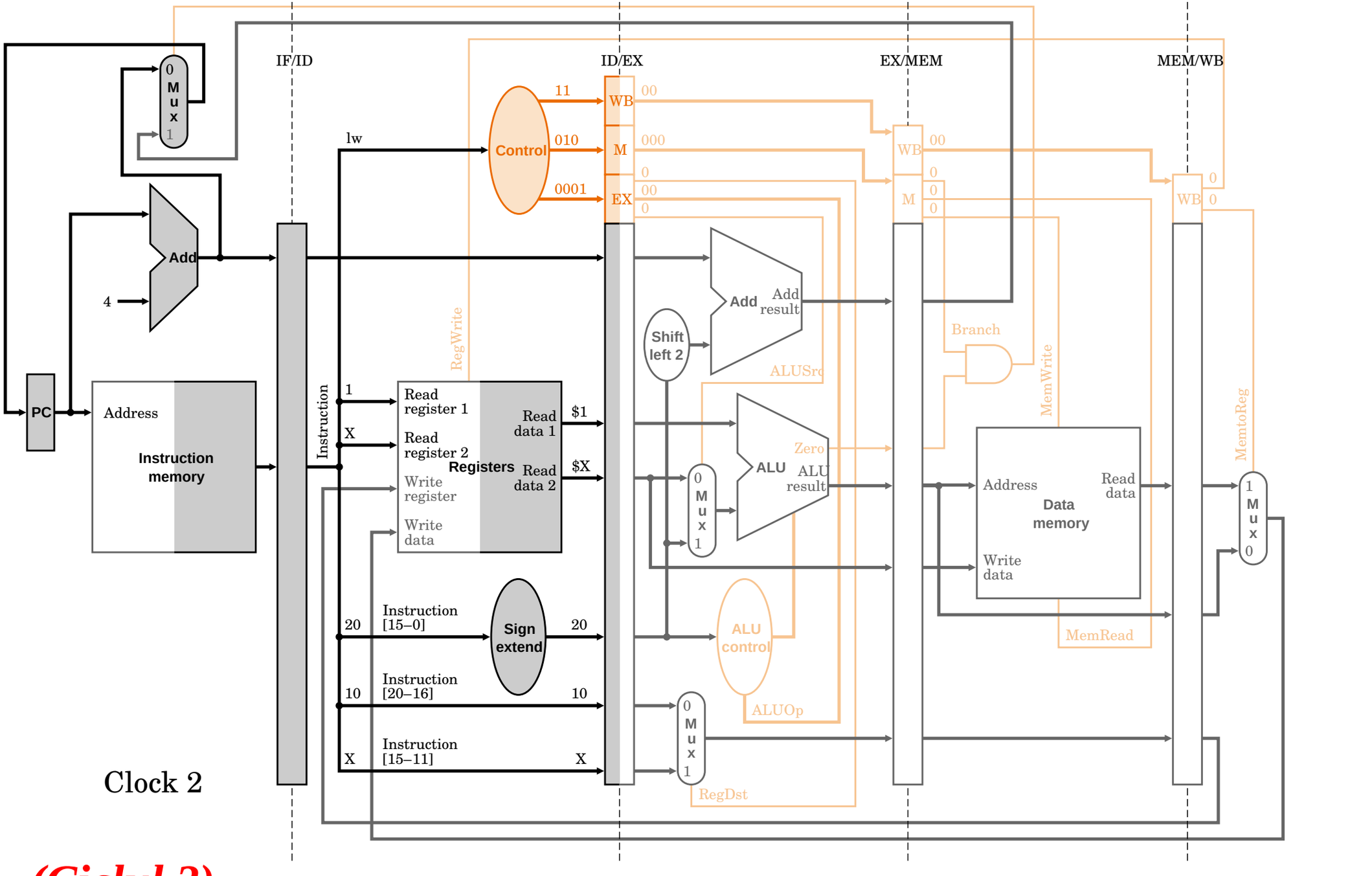
- Ilustrăm *activitatea din procesorul cu pipeline* obţinut mai sus pe o secvenţă simplă de program

```
...
before <1>
lw $10,20($1);
sub $11,$2,$3;
and $12,$4,$5;
or $13,$6,$7;
add $14,$8,$9;
after<1>
...
```

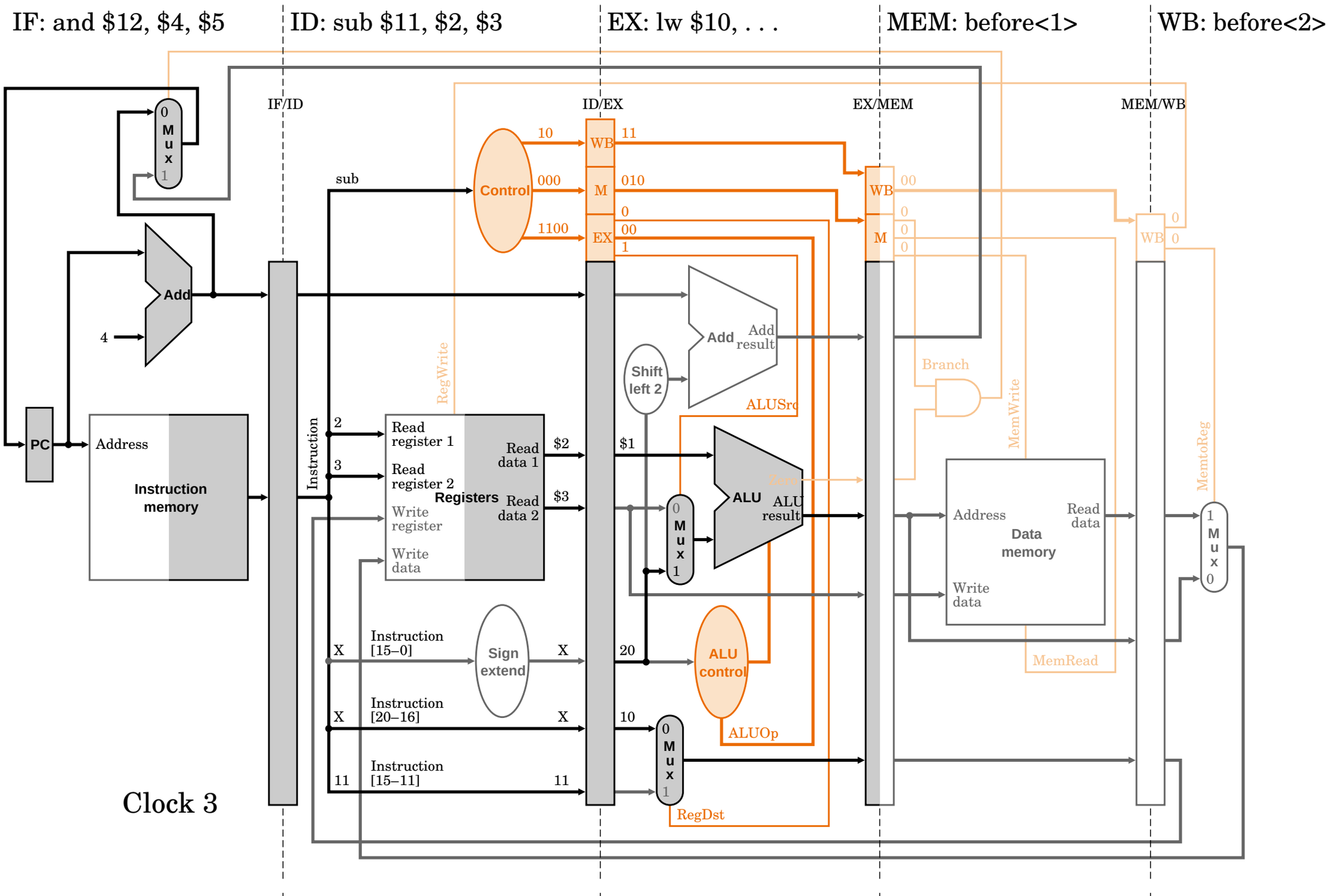
evidenţiind şi *semnalele de control* care apar.

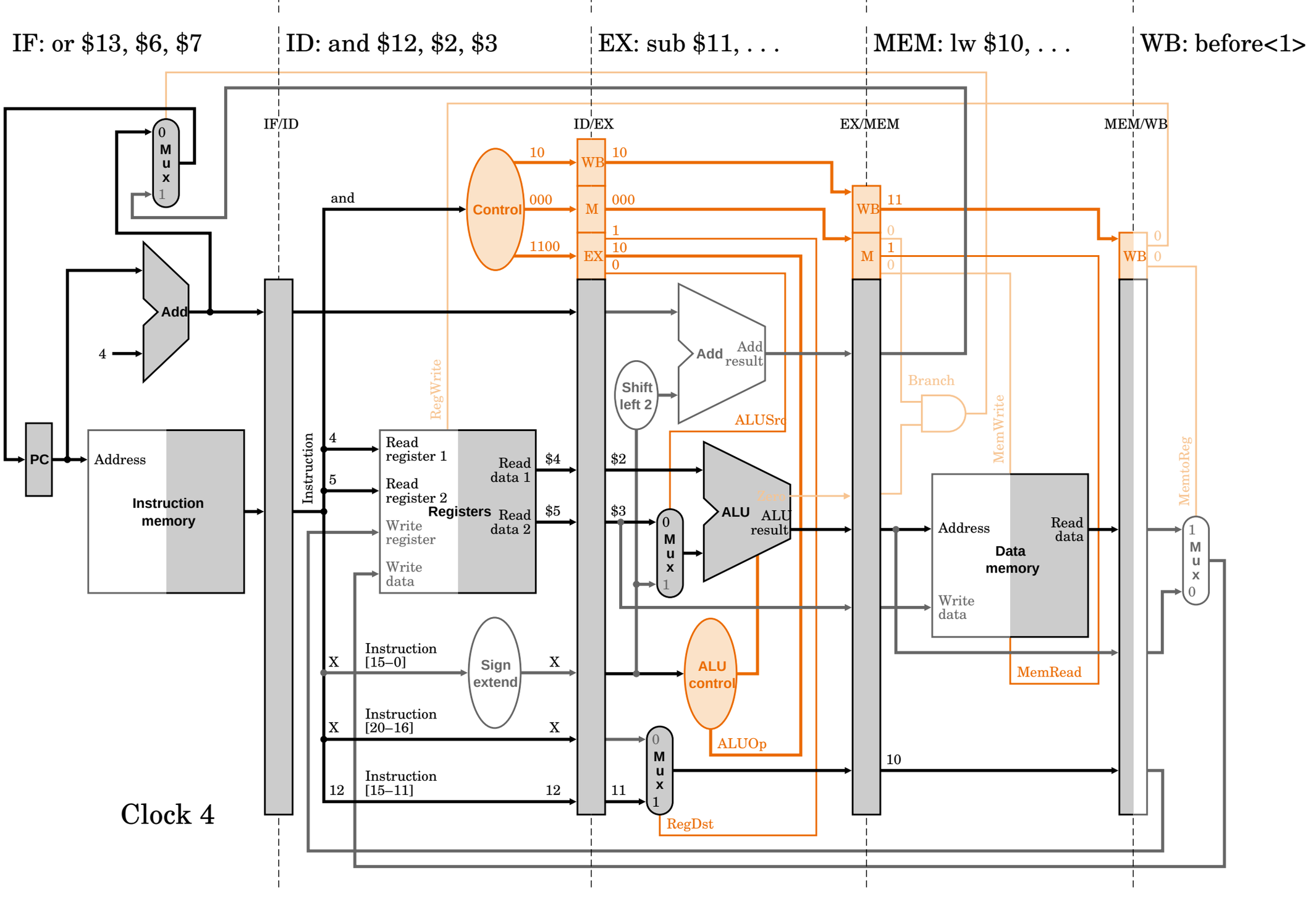
(Ciclul 1)


IF: lw $10, 20($1)
ID: before<1>
EX: before<2>
MEM: before<3>
WB: before<4>
IF/ID
ID/EX
EX/MEM
MEM/WB
0
M
u
x
1
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Control
00
000
0000
WB
M
EX
Shift left 2
Add
Add result
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Instruction [15–0]
Sign extend
Instruction [20–16]
Instruction [15–11]
ALU control
ALUOp
RegDst
Clock 1


IF: sub $11, $2, $3
ID: lw $10, 20($1)
EX: before<1>
MEM: before<2>
WB: before<3>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 2
(Ciclul 2)

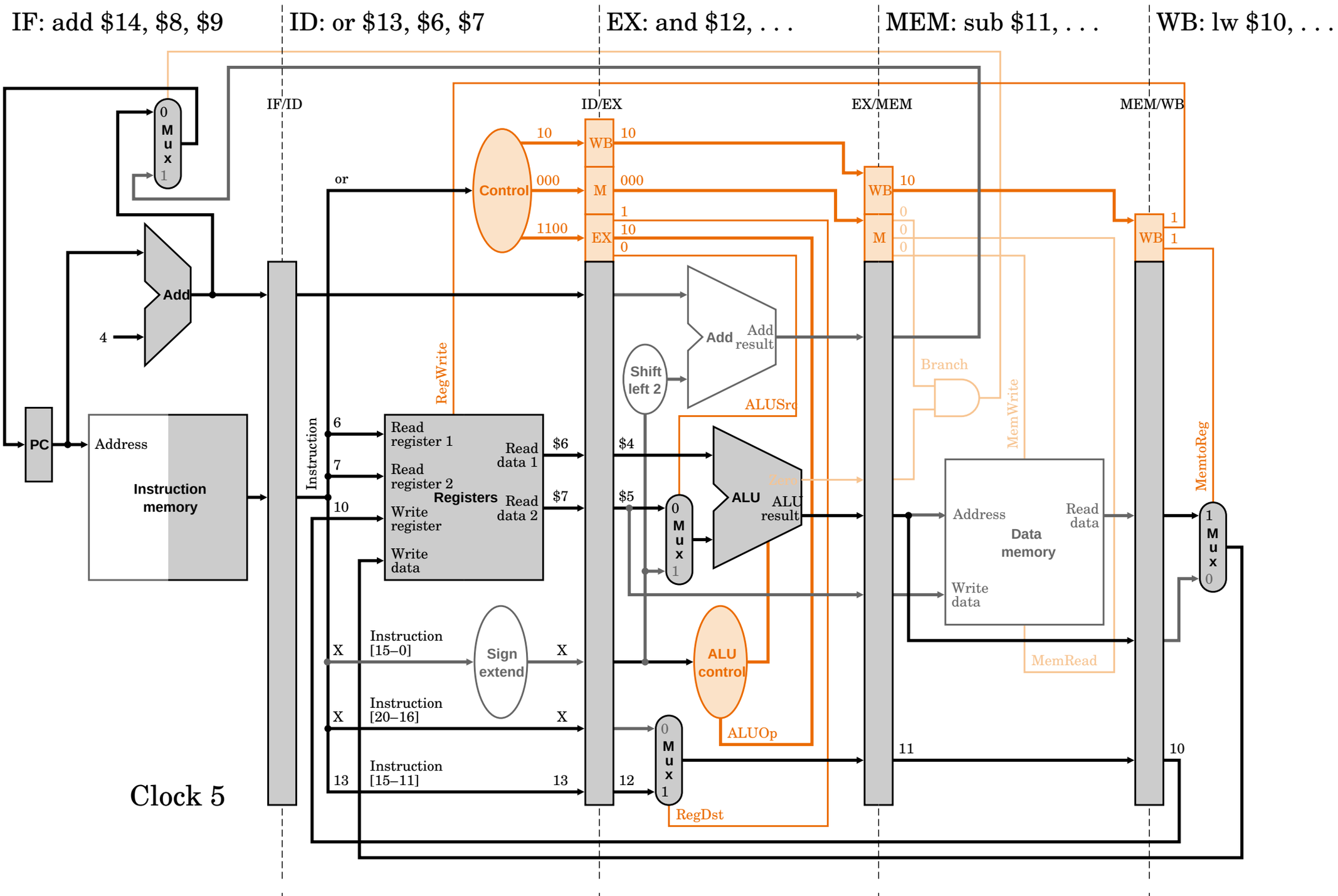
(Ciclul 3)

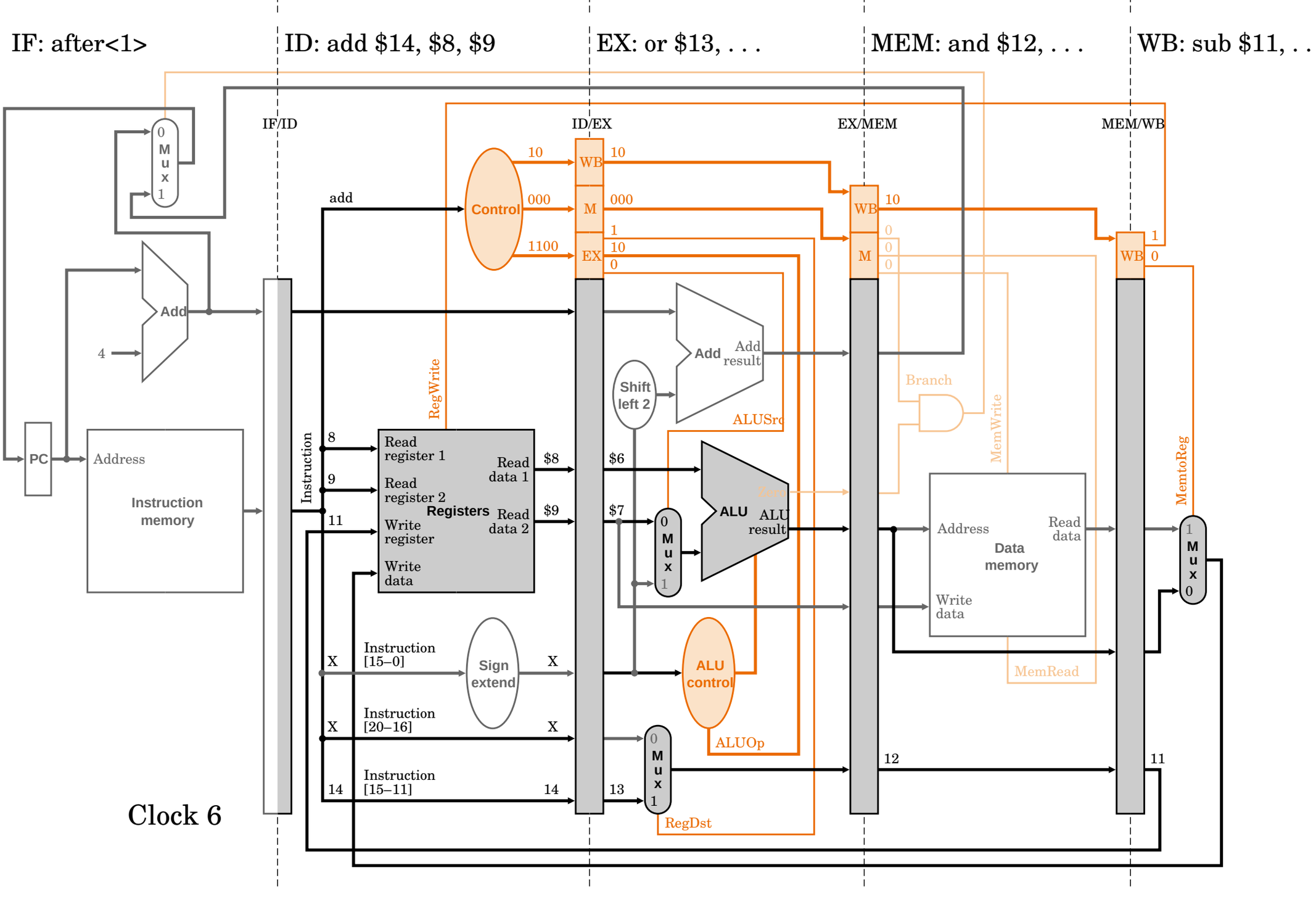

IF: and $12, $4, $5
ID: sub $11, $2, $3
EX: lw $10, . . .
MEM: before<1>
WB: before<2>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 3
PC
Address
Instruction memory
Instruction
Add
4
sub
Control
10
000
1100
WB
M
EX
11
010
00
1
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
2
3
$2
$3
Shift left 2
Add result
ALUSrc
$1
20
10
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Sign extend
X
11
ALU
ALU result
Zero
ALU control
ALUOp
RegDst
Branch
MemWrite
MemRead
MemtoReg
Address
Data memory
Read data
Write data
Mux


IF: or $13, $6, $7
ID: and $12, $2, $3
EX: sub $11, . . .
MEM: lw $10, . . .
WB: before<1>
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 4

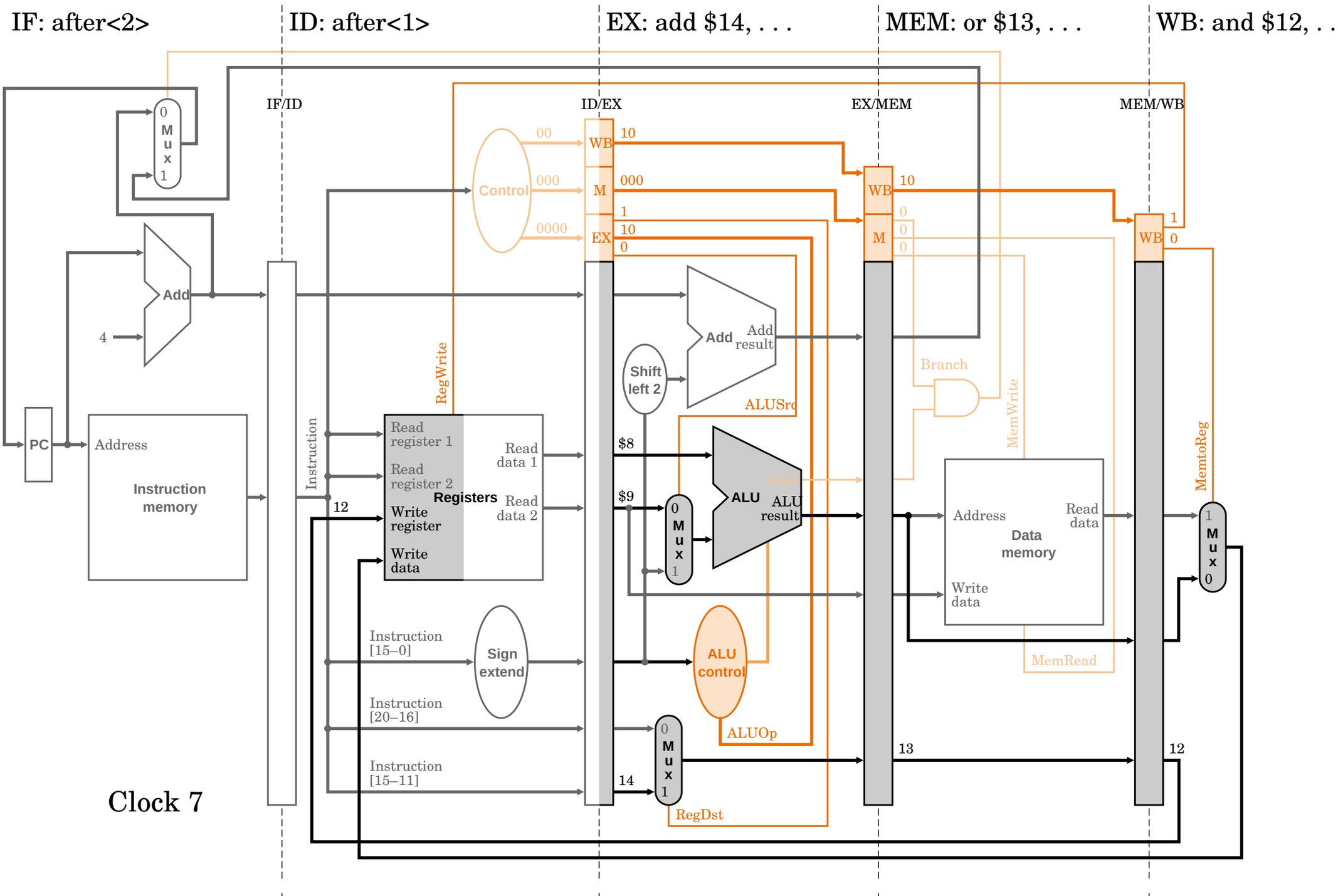
(Ciclul 4)

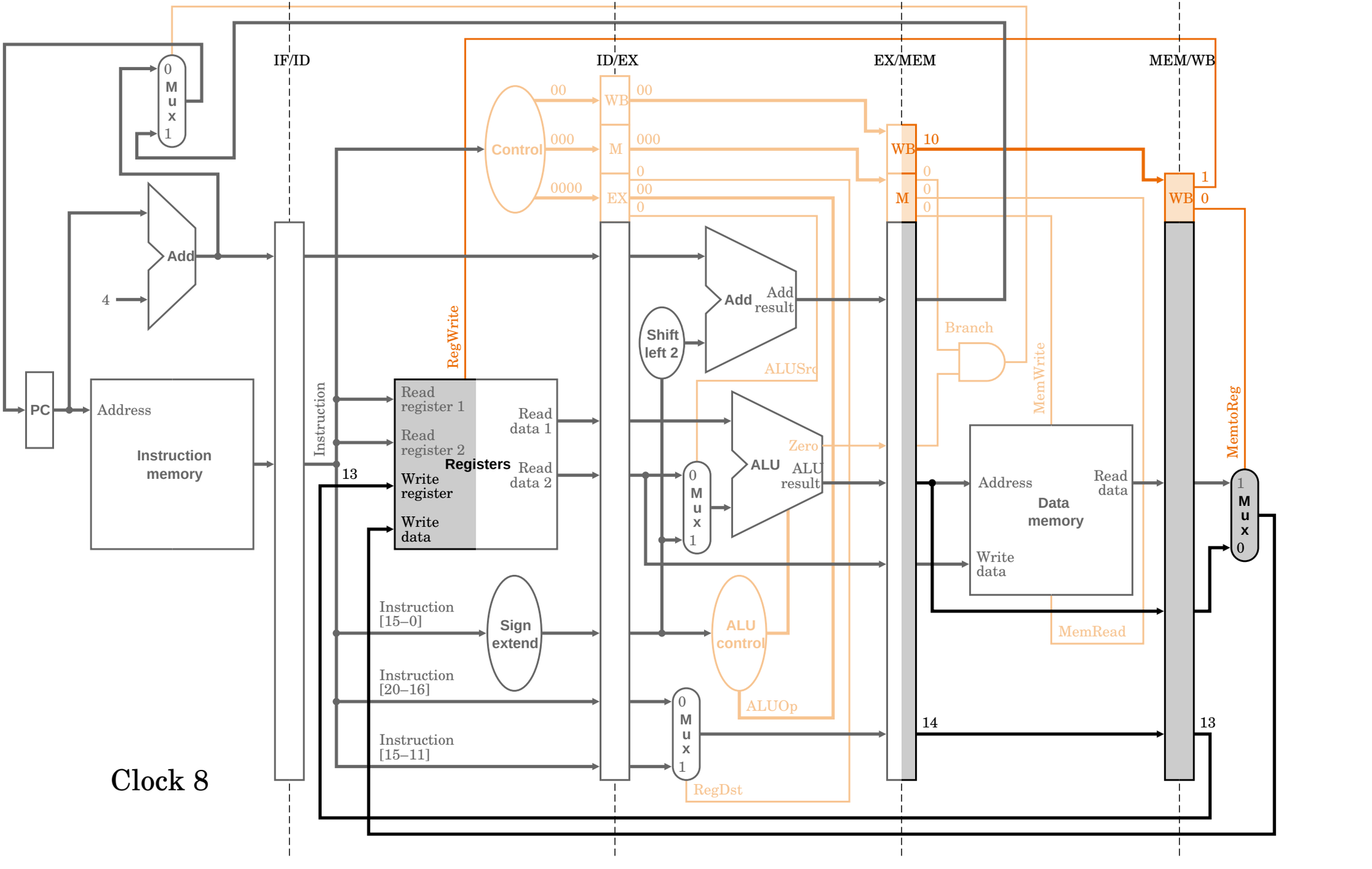
(Ciclul 5)


IF: add $14, $8, $9
ID: or $13, $6, $7
EX: and $12, . . .
MEM: sub $11, . . .
WB: lw $10, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 5
PC
Address
Instruction memory
Instruction
Add
4
0 M u x 1
or
Control
10
000
1100
WB
M
EX
10
000
1
10
0
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
6
7
10
$6
$7
$4
$5
Add
Add result
Shift left 2
ALUSrc
ALU
ALU result
Zero
Branch
MemWrite
MemRead
Address
Data memory
Read data
Write data
MemtoReg
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Sign extend
X
13
12
ALU control
ALUOp
RegDst
11


IF: after<1>
ID: add $14, $8, $9
EX: or $13, . . .
MEM: and $12, . . .
WB: sub $11, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 6
(Ciclul 6)

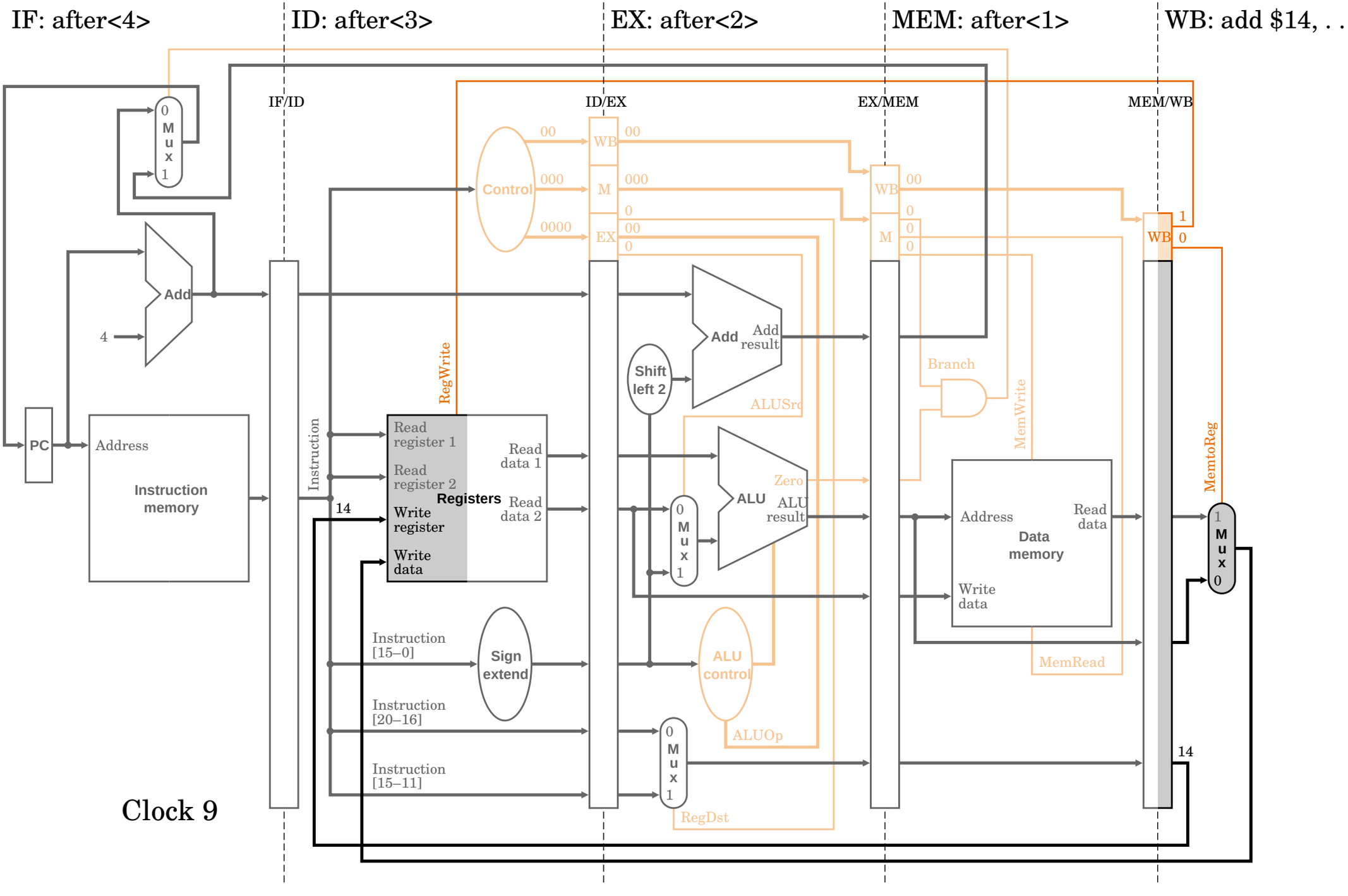
(Ciclul 7)

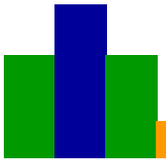

IF: after<2>
ID: after<1>
EX: add $14, . . .
MEM: or $13, . . .
WB: and $12, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 7
PC
Address
Instruction memory
Instruction
Add
4
0 M u x 1
Control
00
000
0000
WB
M
EX
10
000
1
10
0
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
12
$8
$9
Shift left 2
Add
Add result
ALUSrc
0 M u x 1
ALU
Zero
ALU result
Instruction [15–0]
Sign extend
Instruction [20–16]
Instruction [15–11]
ALU control
ALUOp
14
RegDst
Branch
MemWrite
MemRead
Address
Data memory
Read data
Write data
13
12
MemtoReg
1 M u x 0


IF: after<3>
ID: after<2>
EX: after<1>
MEM: add $14, . . .
WB: or $13, . . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 8

(Ciclul 8)

(Ciclul 9)


IF: after<4>
ID: after<3>
EX: after<2>
MEM: after<1>
WB: add $14, . .
IF/ID
ID/EX
EX/MEM
MEM/WB
Clock 9
PC
Address
Instruction memory
Instruction
Add
4
0 M u x 1
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
14
Control
00
000
0000
WB
M
EX
Shift left 2
Add
Add result
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
Address
Data memory
Read data
Write data
MemRead
MemtoReg
Instruction [15–0]
Sign extend
Instruction [20–16]
Instruction [15–11]
ALU control
ALUOp
RegDst



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.*