

Lecția 9:

Procesorul:

Tehnica de pipeline - I

G Ștefănescu — Universitatea București

Arhitectura sistemelor de calcul, Sem.1

Octombrie 2016—Februarie 2017

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 execuției mai multor instrucțiuni*.

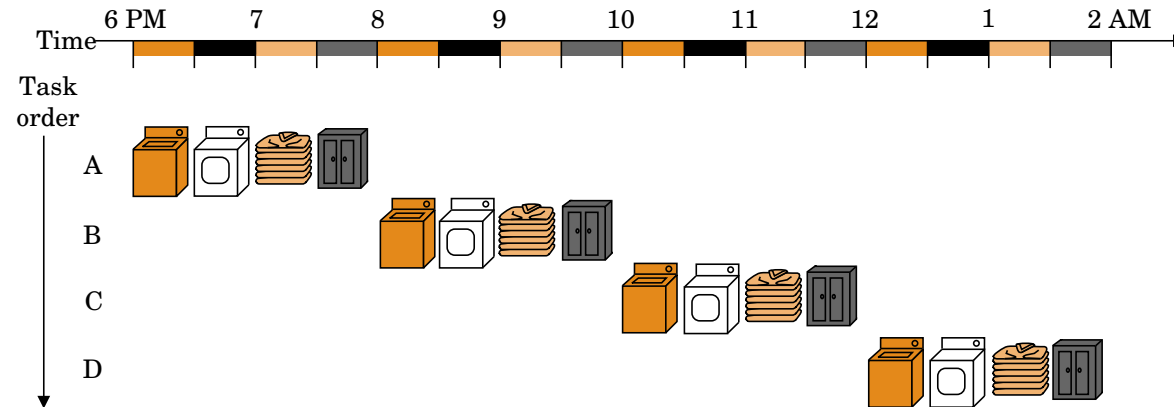
Exemplu: Avem de curățat mai multe seturi de rufe murdare, operațiile necesare fiind, în ordine: *spălare (s)*, *uscare (u)*, *călcare (c)* și *aranjare (a)* (în dulap).

- Secvențial: se repetă secvența $s(i), u(i), c(i), a(i)$ pentru fiecare set i ;
- Paralel, cu pipeline: în fazele 1, 2, 3, 4, 5, ... avem acțiunile
 $s(1),$
 $u(1) \& s(2),$
 $c(1) \& u(2) \& s(3),$
 $a(1) \& c(2) \& u(3) \& s(4),$
 $a(2) \& c(3) \& u(4) \& s(5),$

...

Tehnica de pipeline, într-o imagine

secvențial



*paralel
cu pipeline*

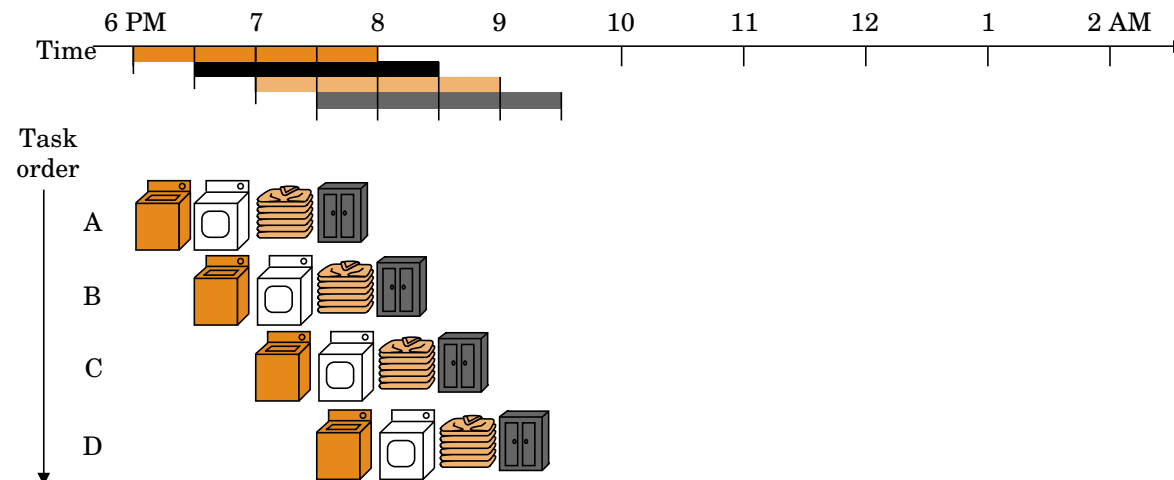


Figura descrie *tehnica de pipeline*, într-o imagine ilustrând operațiile de *spălat*, *uscat*, *călcat* și *aranjat* seturi de rufe A, B, . . .



Tehnica de pipeline

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

- **Pas 1:** Se *extrage* instrucțiunea din memorie;
- **Pas 2:** Se citesc regiștrii și se *decodifică* instrucțiunea;
- **Pas 3:** Se *execută* operația 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 distribuție 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
Format-R (add, sub, and, or, slt)	2ns	1ns	2ns		1ns	7ns
Branch (beq)	2ns	1ns	2ns			5ns

Să considerăm un program simplu cu 3 instrucțiuni load

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

- Execuția cu un ciclu durează $8 \times 3 = 24ns$;
- Cu o variantă de pipeline ca mai sus (exemplul cu rufe), am avea $2 + 2 + 8 = 12ns$;
- Performanța: $Perf_pipe / Perf_1ciclu = 24 / 12 = 2.00$.



..Performanta: Ciclu simplu vs. pipeline

Ciclu simplu vs. pipeline (cont.)

- Intr-o fază de pipeline pot fi active mai multe instrucțiuni; *durata unei faze* va fi *durata celei mai lungi operații* dintre cele procesate în faza respectivă (la noi este $2ns$);
- La limită, dacă în loc de 3 avem k instrucțiuni 1_w ($k \geq 5$), timpii sunt $8k$ și $8 + 2k$, indicând o *viteză de 4 ori mai mare*;
- Sporul de viteză dat de tehnica de pipeline provine din *creșterea numărului de instrucțiuni procesate* într-un timp dat și nu din focalizarea pe *micșorarea timpului de execuție al unei instrucțiuni* individuale.



MIPS: pentru pipeline

Suport MIPS pentru pipeline:

- *Instrucțiunile* MIPS au *acceași lungime*, facilitând extragerea și decodarea în 2 faze. [Arhitectura 80x86 are instrucțiuni de lungime variabilă, între 1B și 17B!]
- Numărul de *formate de instrucțiuni* MIPS este *redus*. Instrucțiunile au un *registru sursă comun*, care poate fi *citit odată cu decodarea* instrucțiunii. [Altfel, ar fi necesară o fază în plus.]
- *Accesul la memorie* se face numai prin *load/store*. [Dacă se admit operații 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 regiștri, în Pasul 2).



Proiectarea unui pipeline

Proiectarea unui pipeline:

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



..Proiectarea unui pipeline

Hazard structural:

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



..Proiectarea unui pipeline

Hazard de control:

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

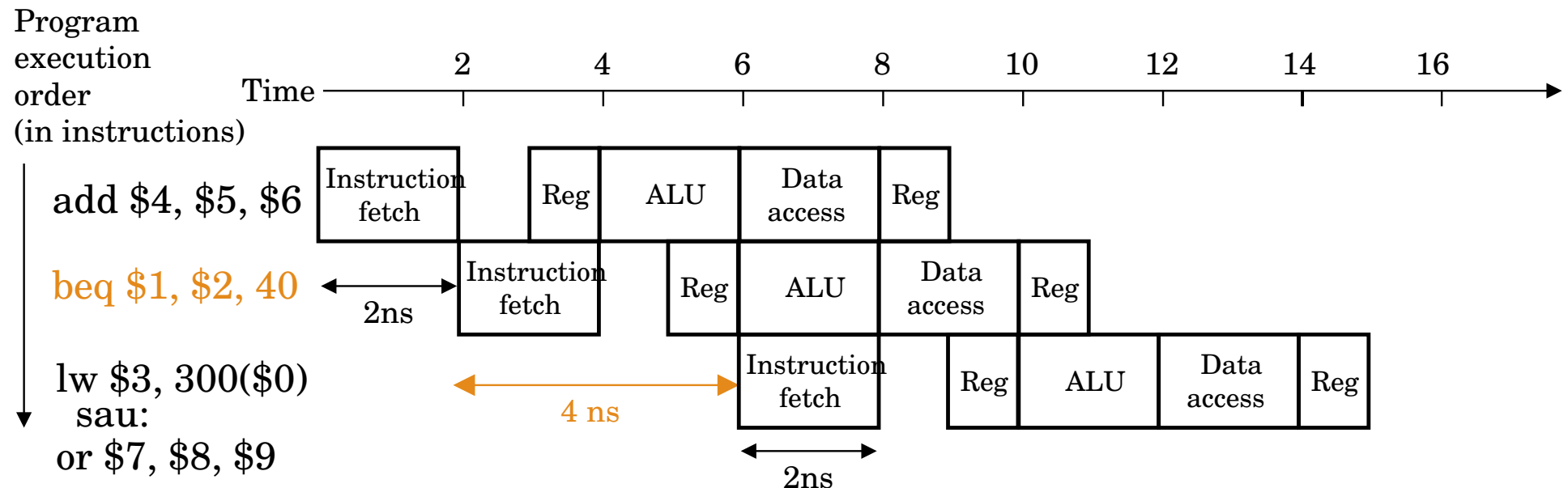
Soluții:

- *Intârzieri* (engl. *stalls*): Intârziem procesarea setului 2 până ce setul 1 este spălat și uscat. (Corect, dar lent.)
- *Predicții*: Continuăm spalarea setului 2 *cu speranța* 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 predicție greșită este costisitoare.)

..Proiectarea unui pipeline

Întârzieri:

- Figura descrie un *hazard de control*, pentru instrucțiunea condițională beq rezolvat cu *tehnica de întârziere*.
- Pentru a diminua întârzierea, rezultatul testului din ALU (ieșirea zero) se folosește *în același ciclu* pentru a încărca noua instrucțiune. (Normal, trebuie o întârziere de 3 cicluri.)





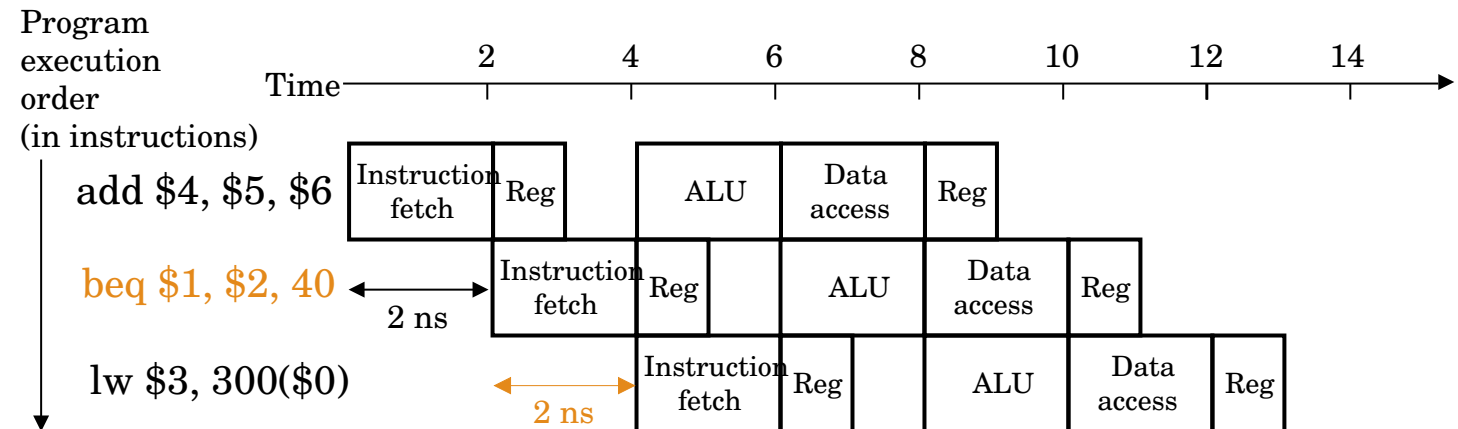
..Proiectarea unui pipeline

Predicții:

- Alternativ, spre a nu întârzia pipeline-ul, se pot folosi *predicții* relativ la *ce ramură* a instrucțiunii condiționale *va fi urmată*.
- Uneori se poate lua o simplă *decizie statică*: la o instrucțiune “loop” ne așteptăm ca bucla să fie urmată cu o frecvență mare, deci îi vom acorda o probabilitate corespunzătoare.
- Alteori putem folosi o *decizie dinamică*, anume contorizăm frecvența de acceptare a fiecărei ramuri și o folosim pentru a prezice ramura care va fi urmată în instrucțiunea curentă.
- *Costul unei predicții greșite* poate fi ridicat, mai ales în cazul în care pipeline-ul are mai multe faze.

..Proiectarea unui pipeline

*ramificație
neacceptată*



*ramificație
acceptată*

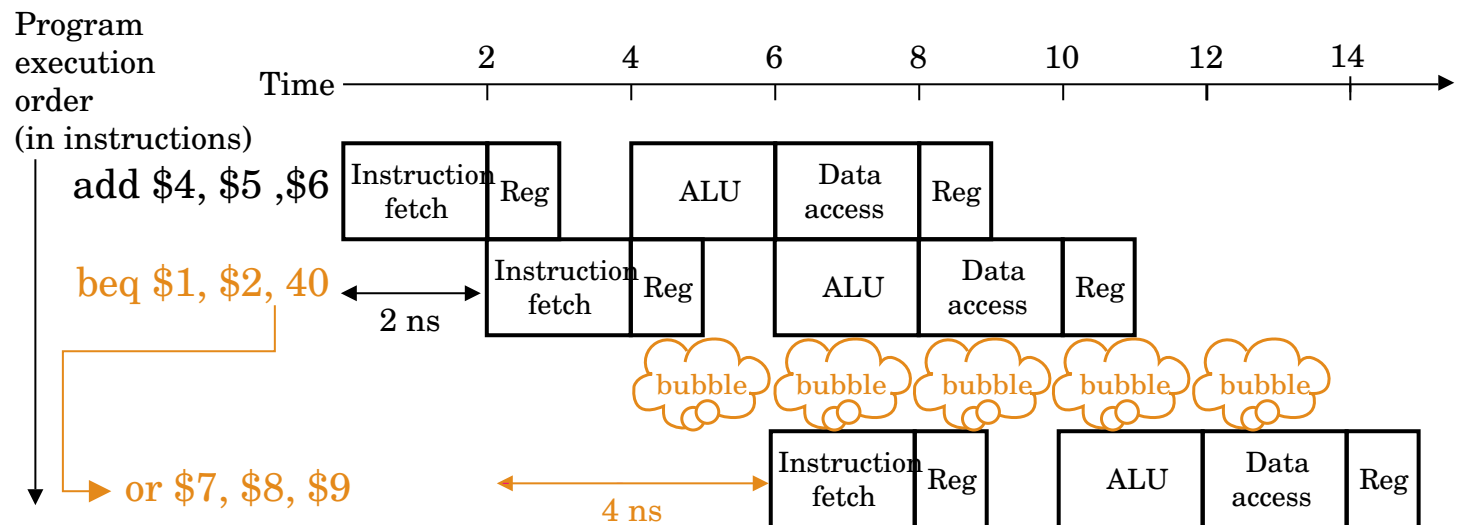
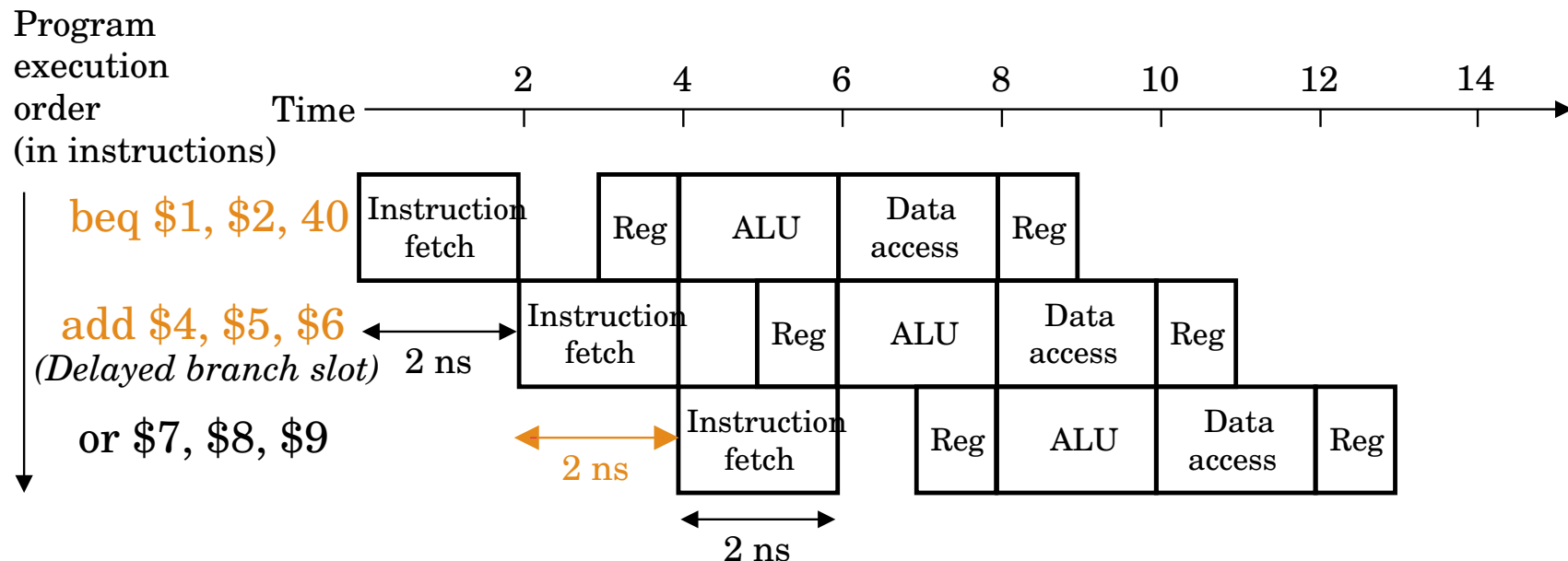


Figura descrie același *hazard de control* pentru instrucțiunea condițională beq rezolvat cu *tehnica de predicție*.

..Proiectarea unui pipeline

Decizii intarziate:

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





..Proiectarea unui pipeline

Hazard de date:

- *Hazardul de date* este generat de interdependența instrucțiunilor realizată prin intermediul datelor (variabilelor).
- Exemplu: O instrucțiune poate necesita rezultatul unei instrucțiuni anterioare, care este încă în pipeline, ca în exemplul

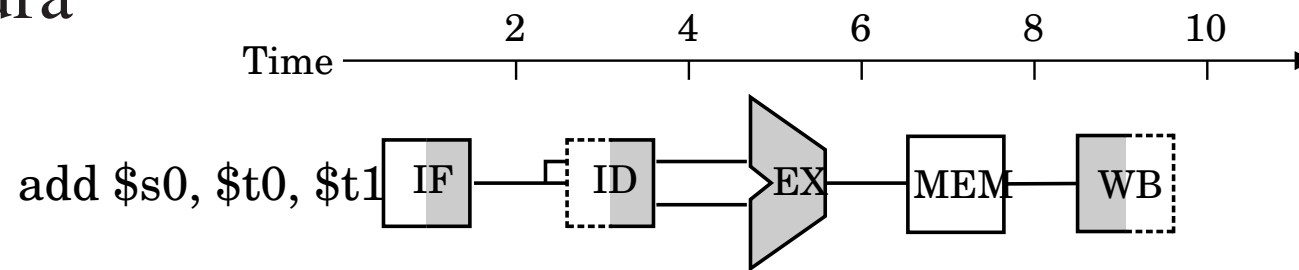
```
add $s0, $t0, $t1;  
sub $t2, $s0, $t3;
```

- O *soluție* de a evita întârzierea pipeline-ului este *avansarea (forwarding / bypassing)*, tehnică prin care *data necesară* într-o instrucțiune următoare se folosește *imediat ce a fost obținută* în instrucțiunea curentă.

..Proiectarea unui pipeline

Hazard de date (cont.)

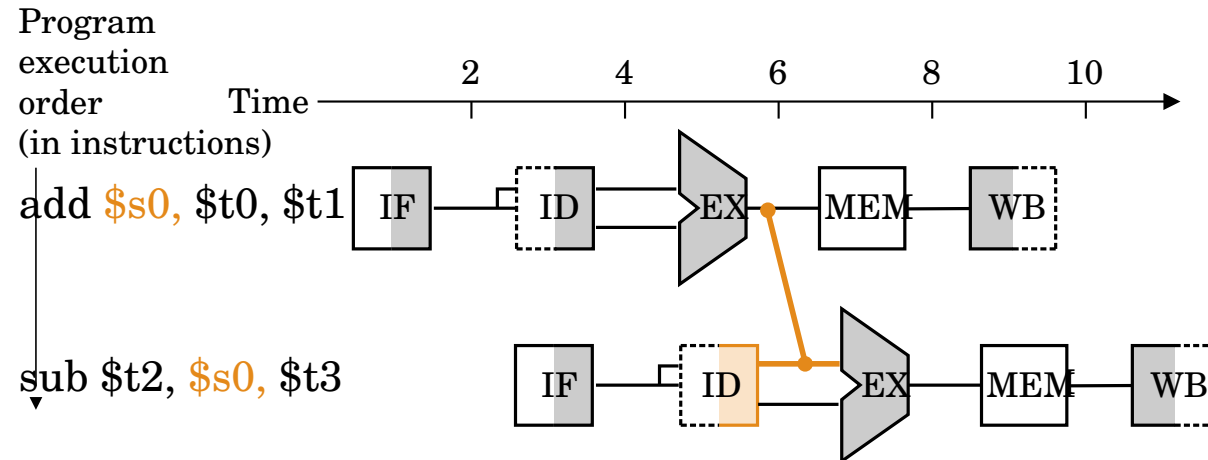
- Se poate folosi tehnica de *avansare* numai dacă *timpul* la care apare *data sursă* este *anterior* celui din faza de *utilizare*.
- Pentru o analiza mai fină, putem împărți fazele unei procesări ca în figură



- Fazele sunt: *IF* - extragere instrucțiune; *ID* - decodare instrucțiune; *EX* - execuție 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 memoriei, cu *scriere* pe *prima jumătate* și *citire* pe *a doua*.

..Proiectarea unui pipeline

*hazard
rezolvat
cu avansare*



*hazard
rezolvat
cu avansare
și întârziere*

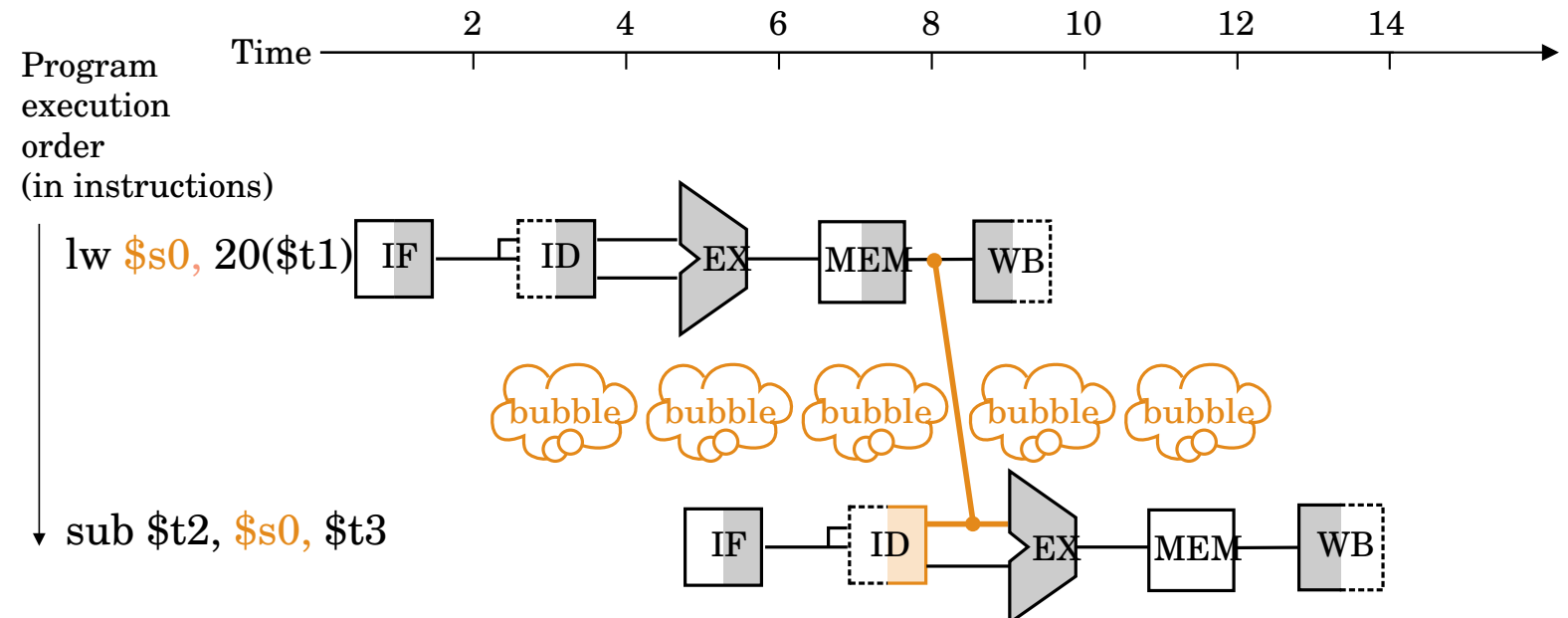


Figura descrie 2 cazuri de *hazard de date* rezolvate cu *tehnica de avansare*. Jos, chiar și cu avansare, pipeline-ul trebuie întârziat.



..Proiectarea unui pipeline

Rearanjarea codului:

- O tehnică generală de *evitare a hazardul* este *reordonarea instrucțiunilor*, fără a modifica funcția 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.

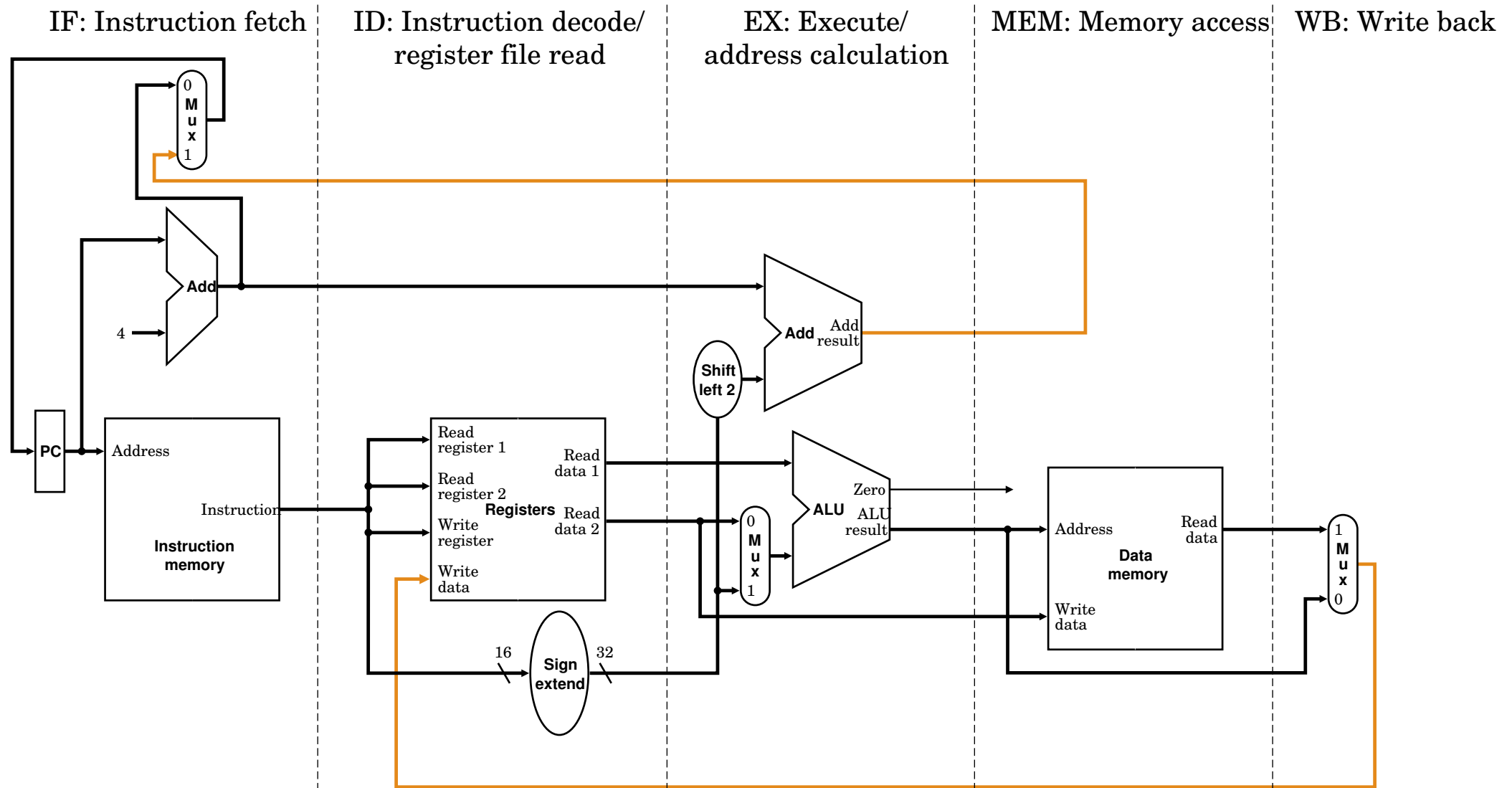


Procesorul: Tehnica de pipeline

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, *împărțită în faze*. (In roșu sunt arce de “întoarcere” care complică pipeline-ul.)



Calea de date cu pipeline

Primii pasi

- Ca anterior, putem împărți implementarea cu un ciclu în 5 faze:
 - IF* - extragere instrucțiune;
 - ID* - decodare instrucțiune;
 - EX* - execuție în ALU;
 - MEM* - acces memorie;
 - WB* - scriere în registru
- Toate conexiunile sunt *de la stînga la dreapta*, cu *2 excepții*:
 - scris PC în caz de instrucțiune condițională;
 - scris în registru (la load);

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

..Calea de date cu pipeline

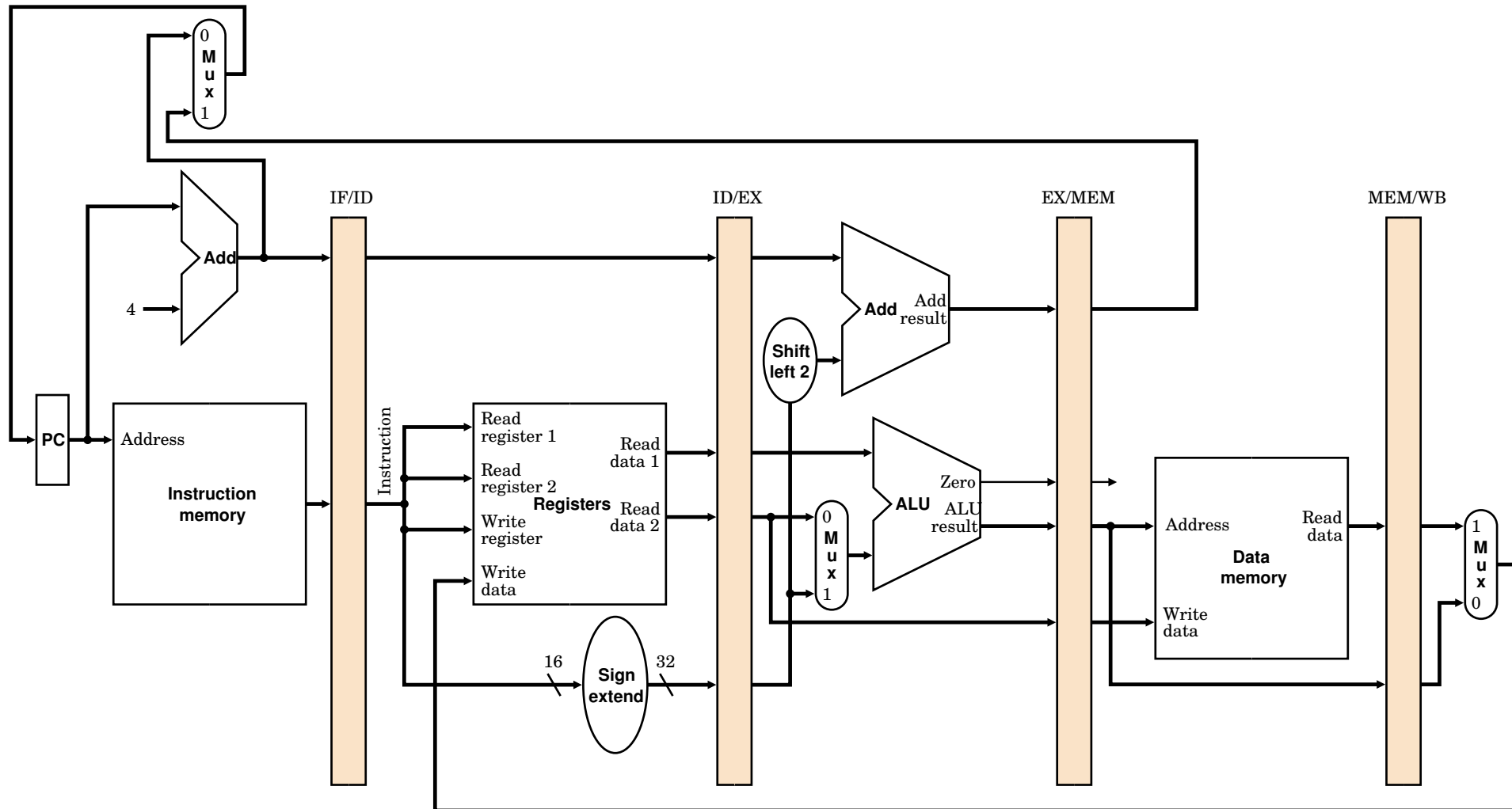


Figura descrie un *pipeline* pentru *implementarea cu un singur ciclu*. Se introduc *regiștri specifici* care să rețină datele între fazele de pipeline: IF/ID, ID/EX, EX/MEM, MEM/WB.



..Calea de date cu pipeline

Exemplu - load:

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

- *IF - extragere instrucțiune:* Se extrage instrucțiunea, se calculează noul PC (pentru instrucțiunea următoare $PC + 4$). Se rețin în registrul IF/ID instrucțiunea și $PC + 4$ (ultimul, pentru branch).
- *ID - decodare instrucțiune și citire regiștri:* Instrucțiunea din IF/ID se folosește pentru a citi 2 regiștri și a face extensia deplasării la 32b. Cele 3 date obținute se rețin în registrul ID/EX (și $PC + 4$, dar inutilă acum).

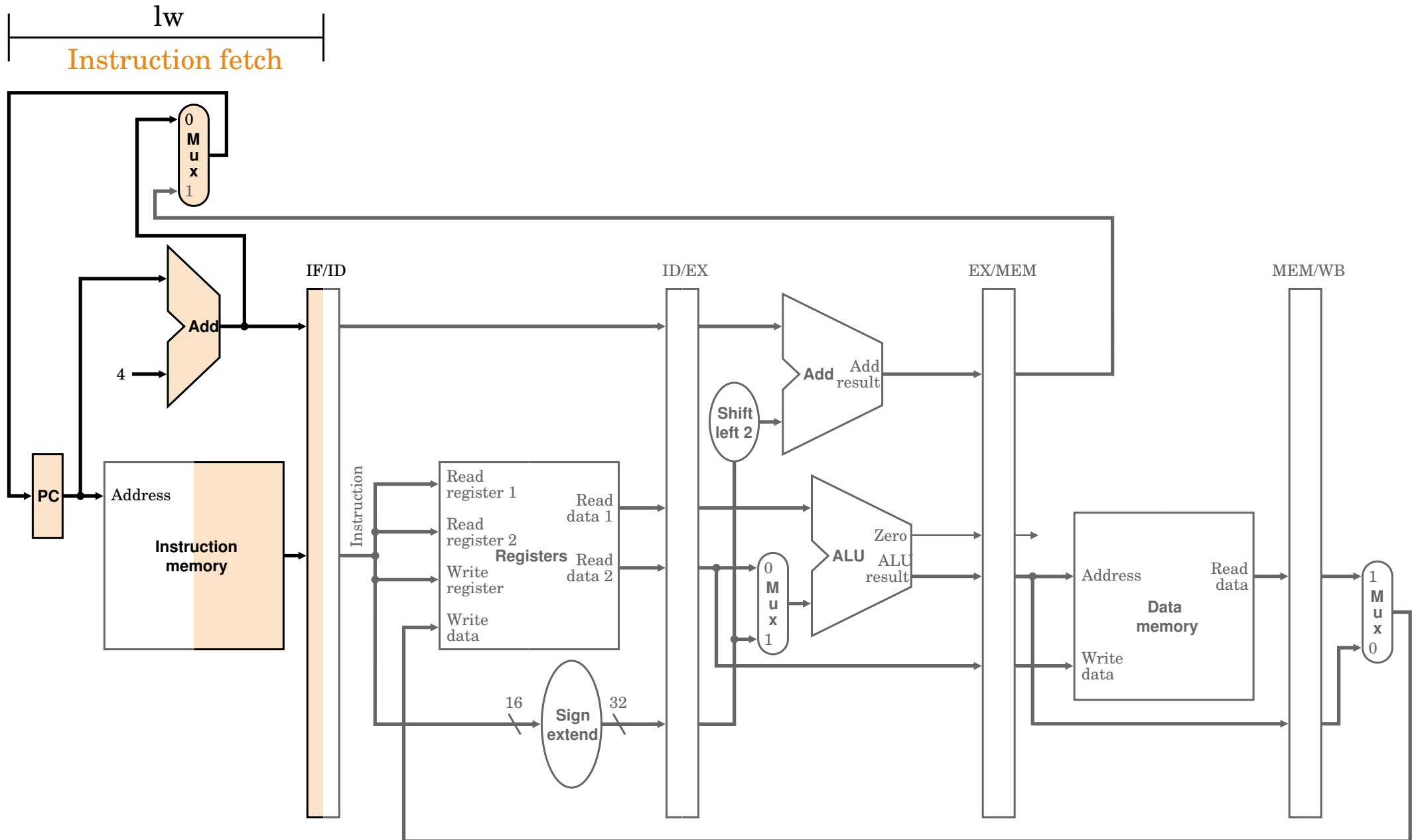


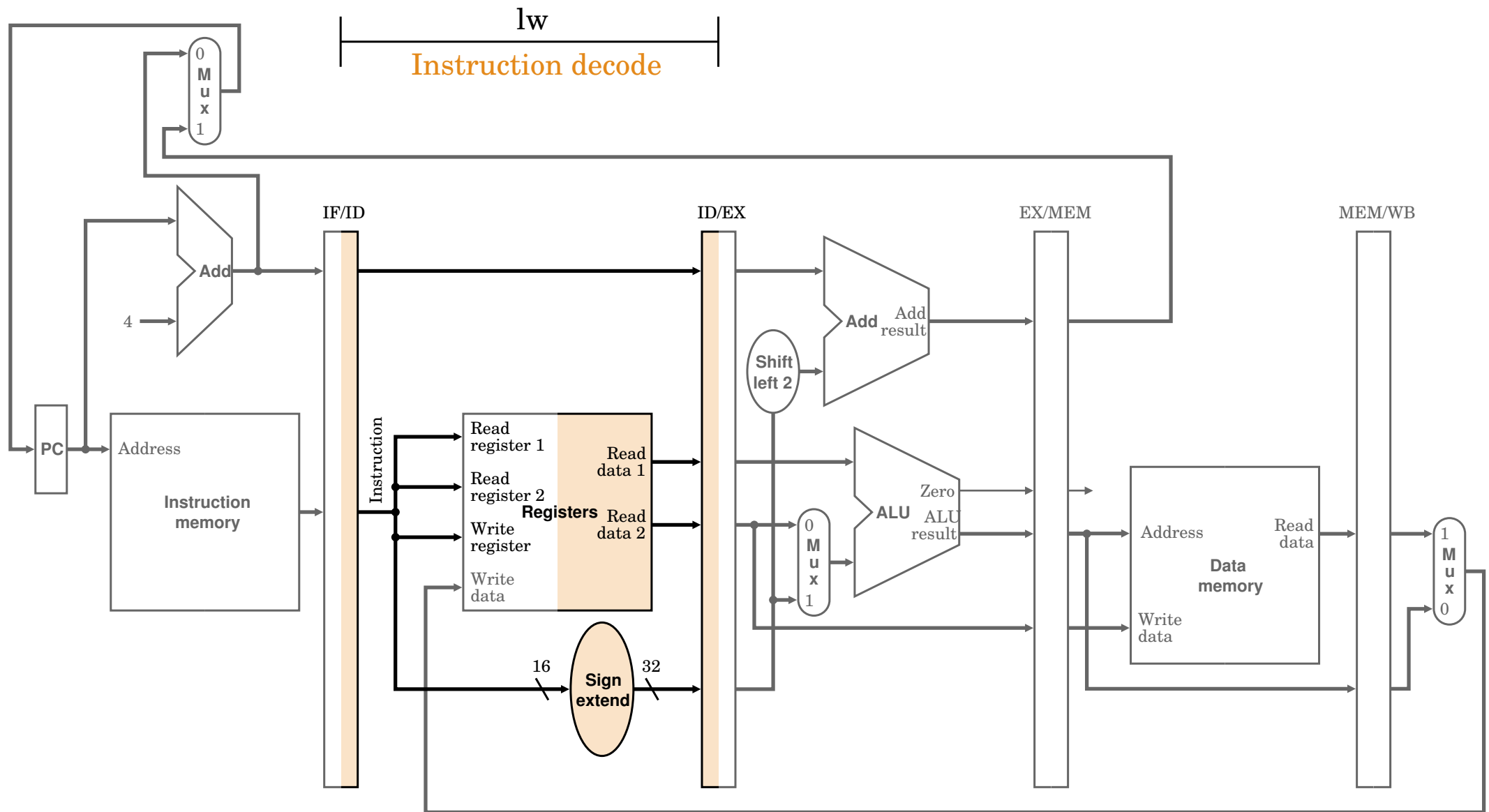
..Calea de date cu pipeline

Exemplu - load (cont.)

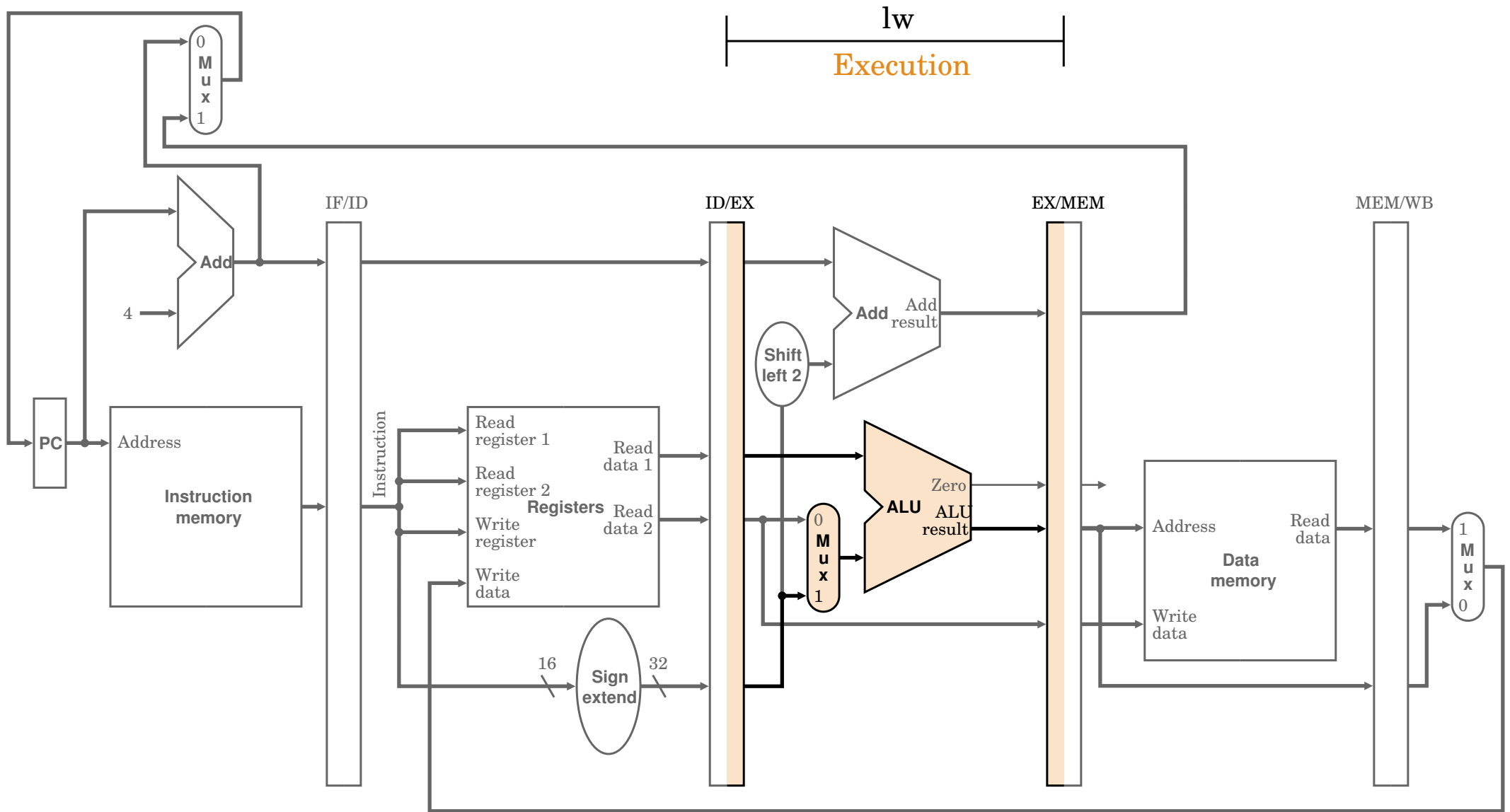
- *EX - execuție:* Folosind valoarea registrului 1 și deplasarea (reținute în ID/EX), calculăm în ALU adresa de memorie, pe care o punem în registrul EX/MEM.
- *MEM - acces memorie:* Se accesează memoria cu adresa de memorie din registrul EX/MEM, rezultatul reținându-se în MEM/WB.
- *WB - scriere în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

Faza 1: IF (extragere instrucțiune)



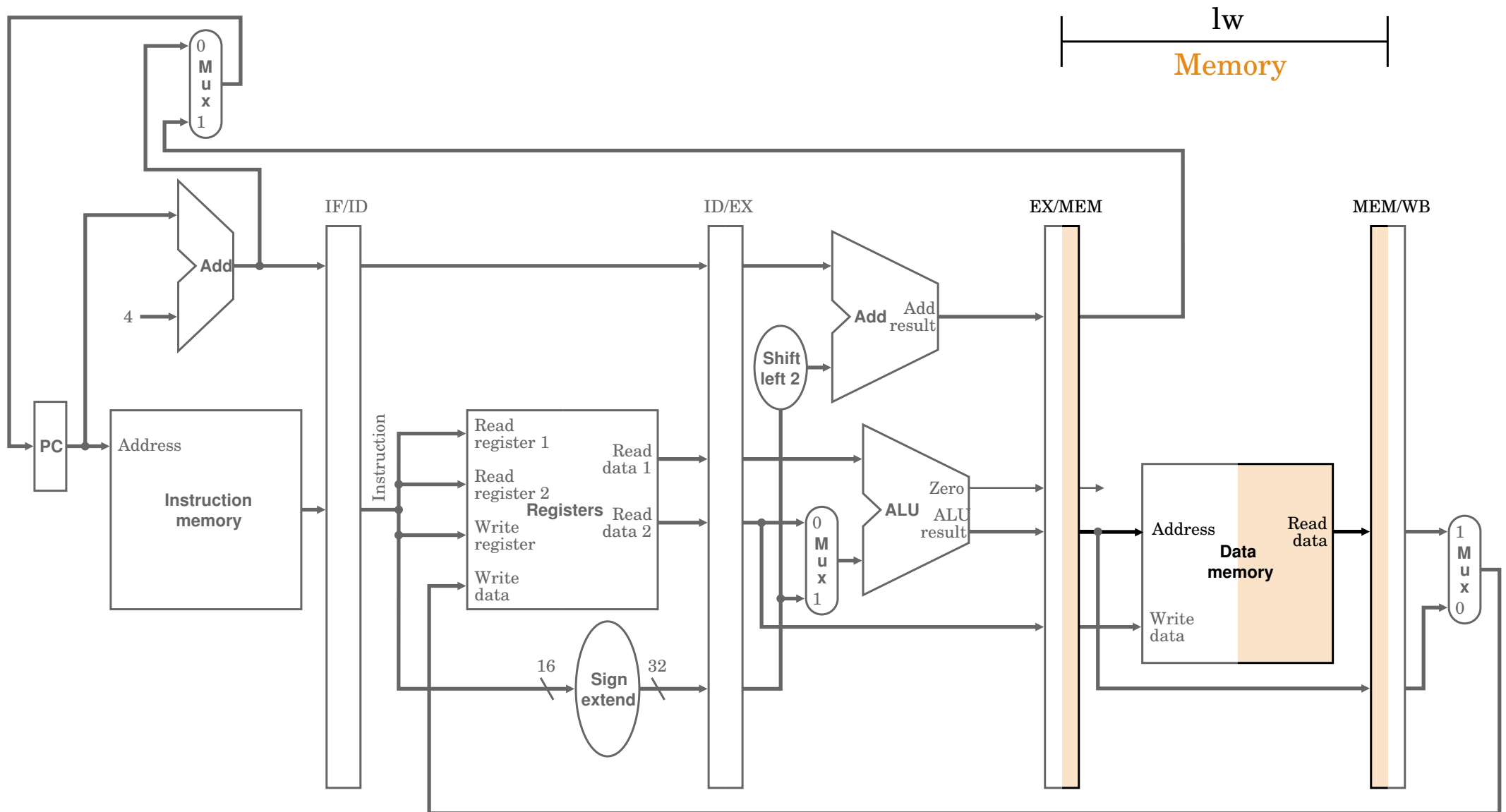


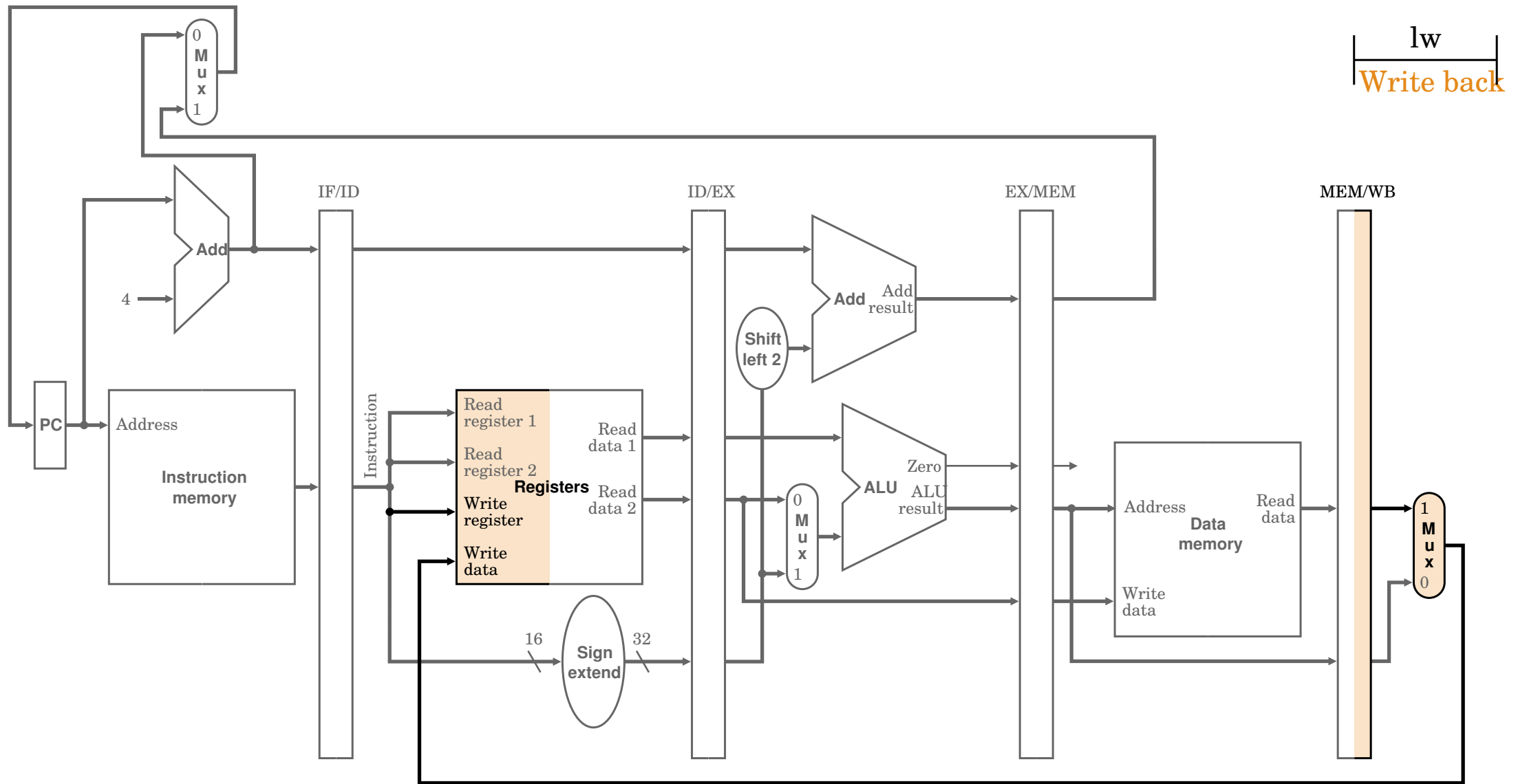
Faza 2: ID (decodare instrucțiune)



Faza 3: EX (execuție)

Faza 4: MEM (acces memorie)





Faza 5: WB (scriere în registru)



..Calea de date cu pipeline

Comentarii:

- In fiecare fază trebuie salvate în regiștri speciali de pipeline *toate* datele necesare fazelor următoare.
- In versiunea ilustrată, arcul de feedback din ultima fază WB reliefează *o eroare* de referință: *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 instrucțiuni: store, add, etc.

..Calea de date cu pipeline

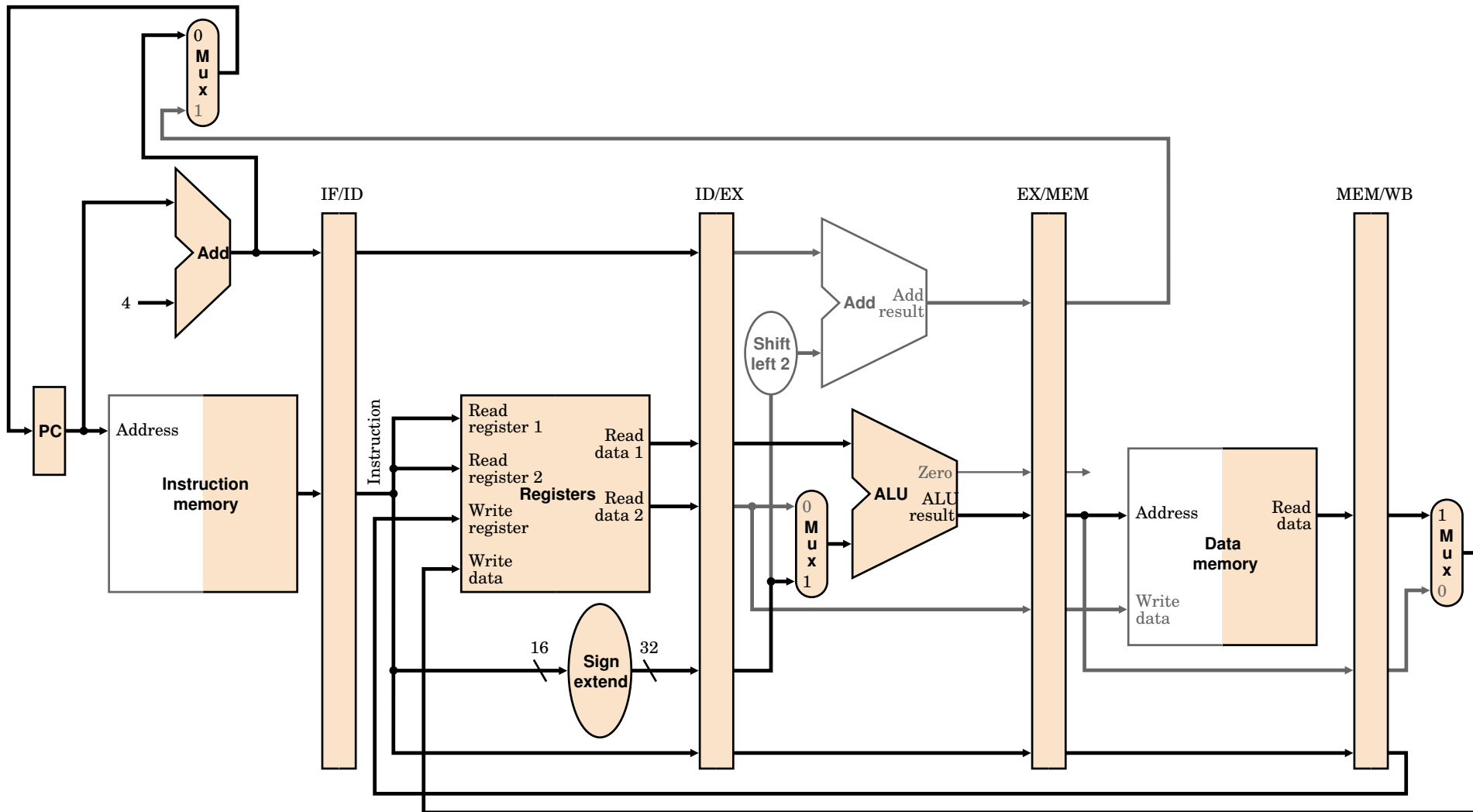


Figura descrie un *pipeline* (corect) pentru implementarea instrucțiunii *load*. Am inserat conexiunea de propagare a *adresei registrului de scriere*.

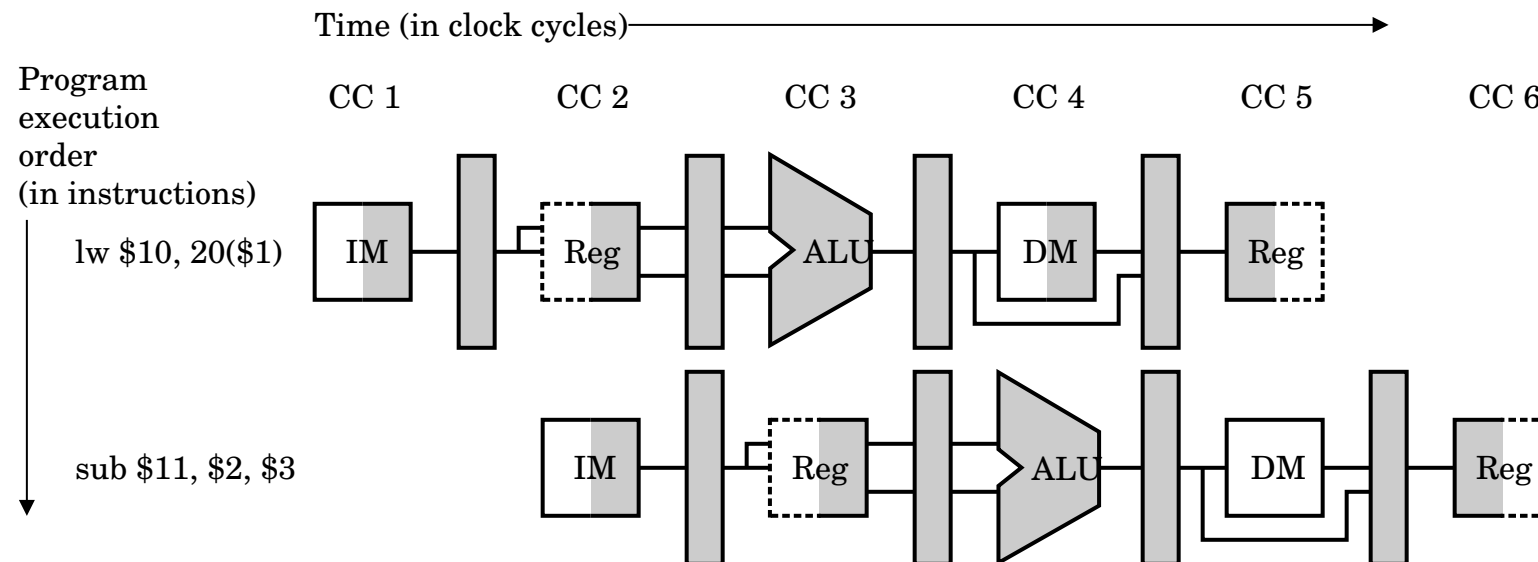
Multiple instructiuni in pipeline

Multiple instructiuni in pipeline:

- Ilustrăm trecerea a 2 instrucțiuni prin pipeline

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

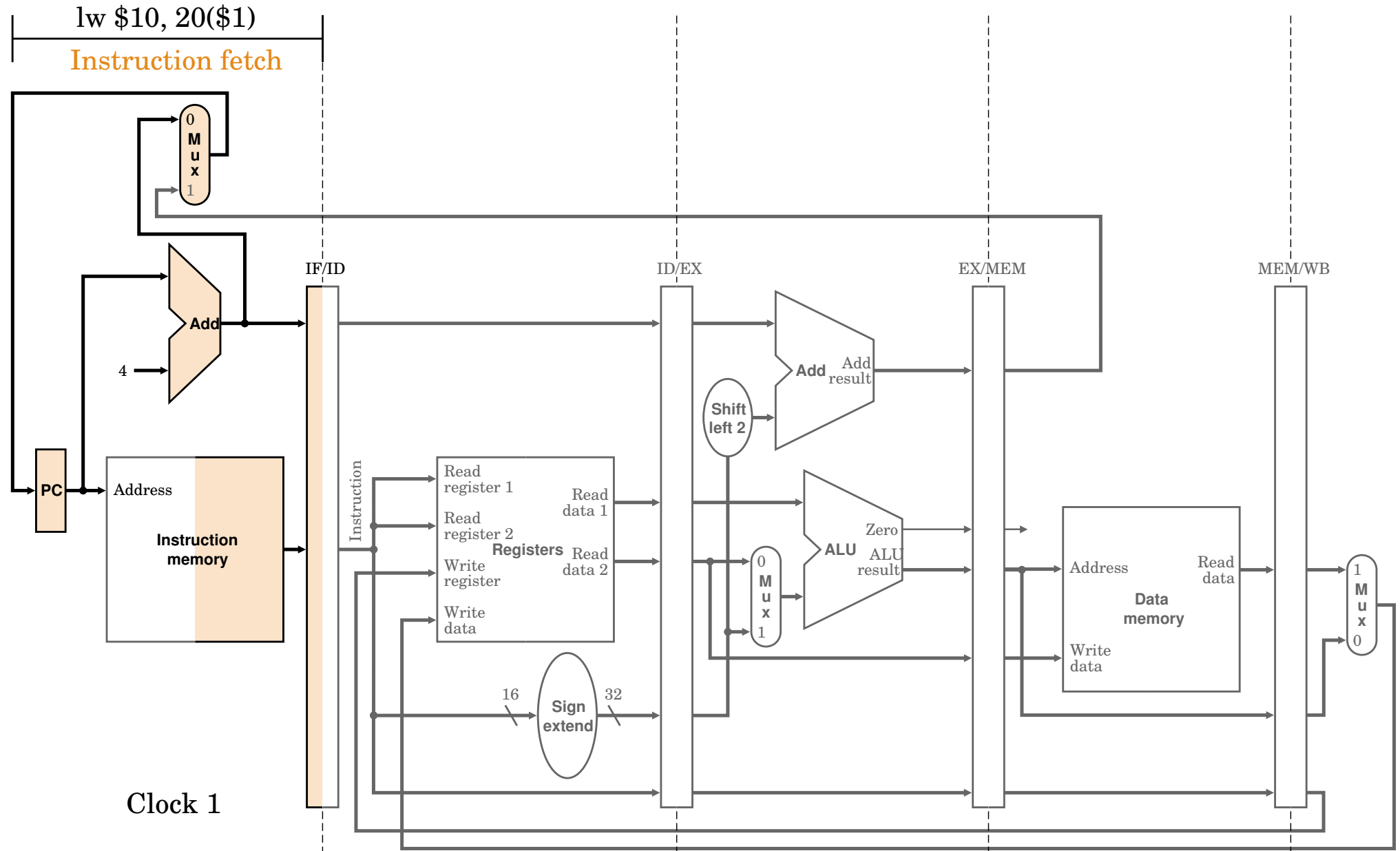
- Se pot utiliza *diagrame pipeline cu multiple cicluri de ceas* de tipul

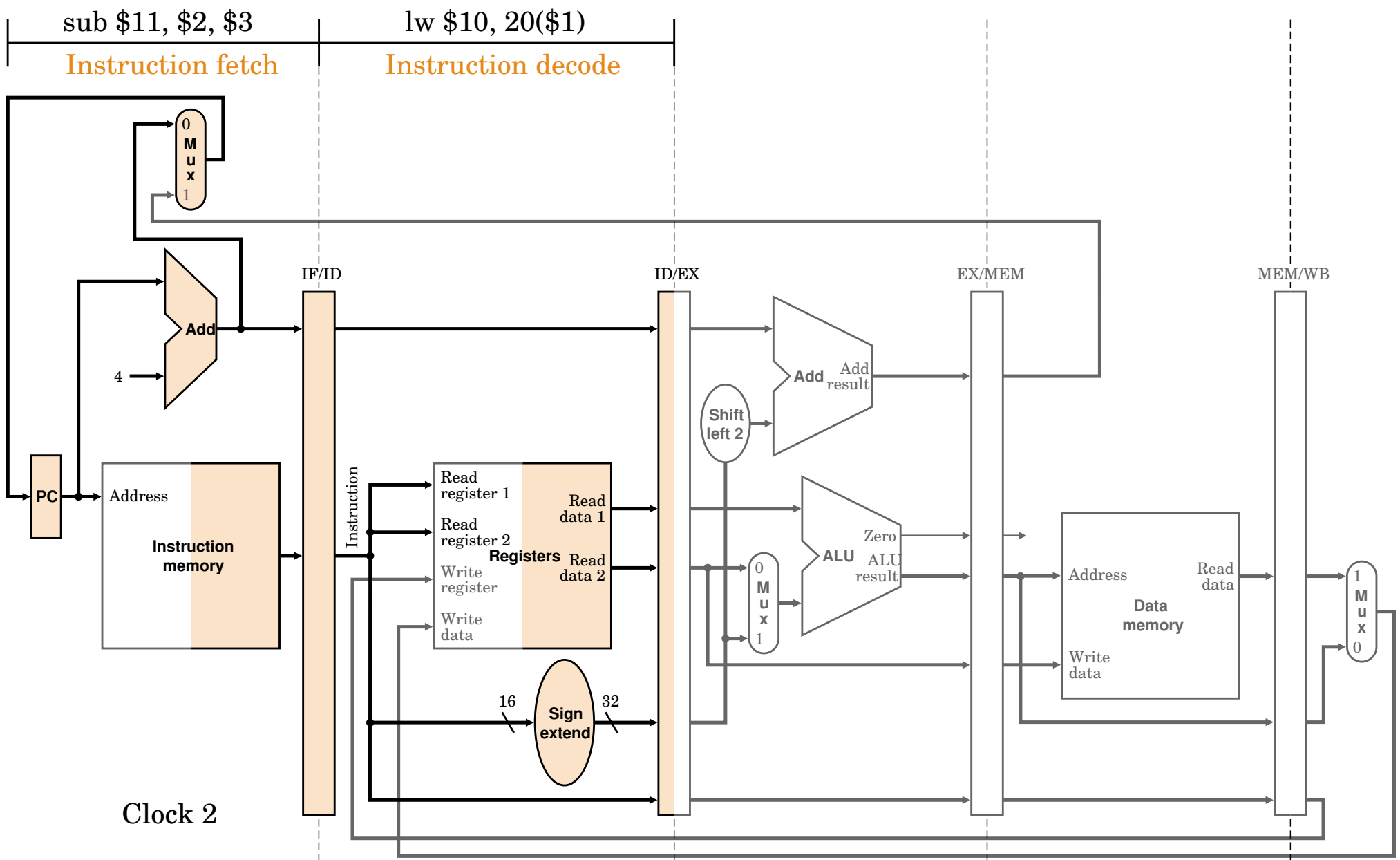


ori *secvențe de diagrame pipeline cu un singur ciclu de ceas*, ca în figurile ce urmează.

Fazele de execuție în *pipeline* pentru o secvență “*load; sub*”:

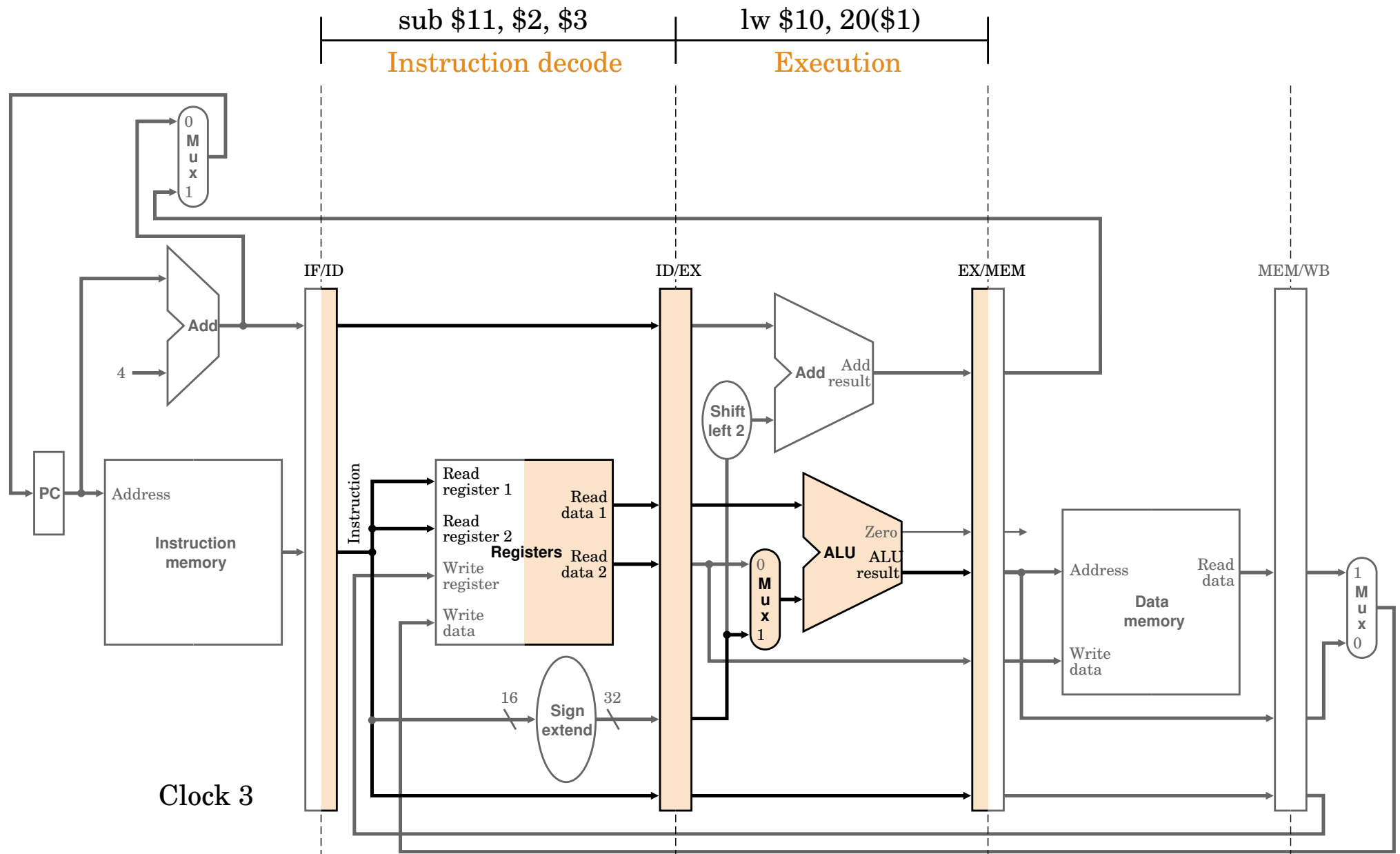
Faza 1: IF-load

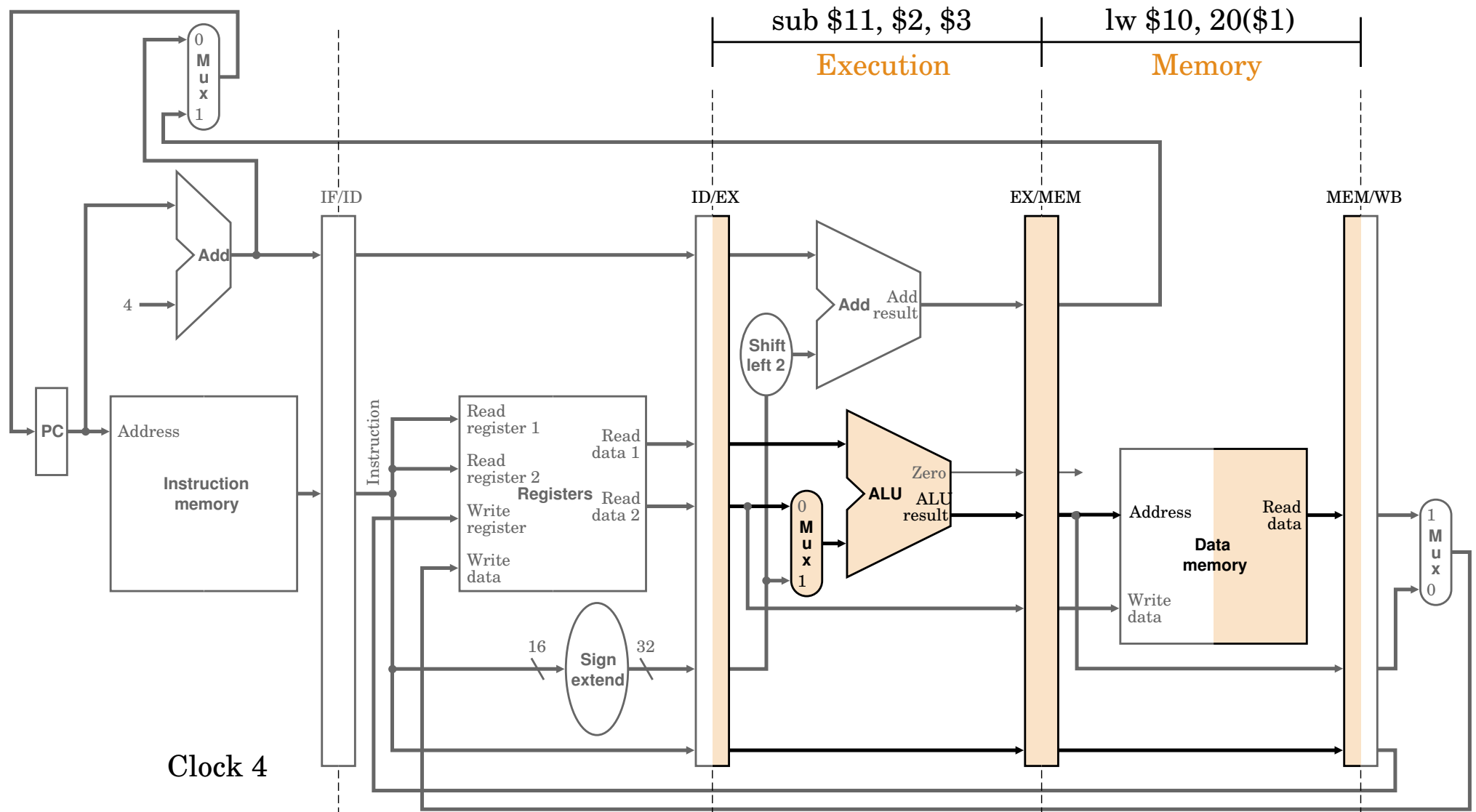




Faza 2: IF-sub & ID-load

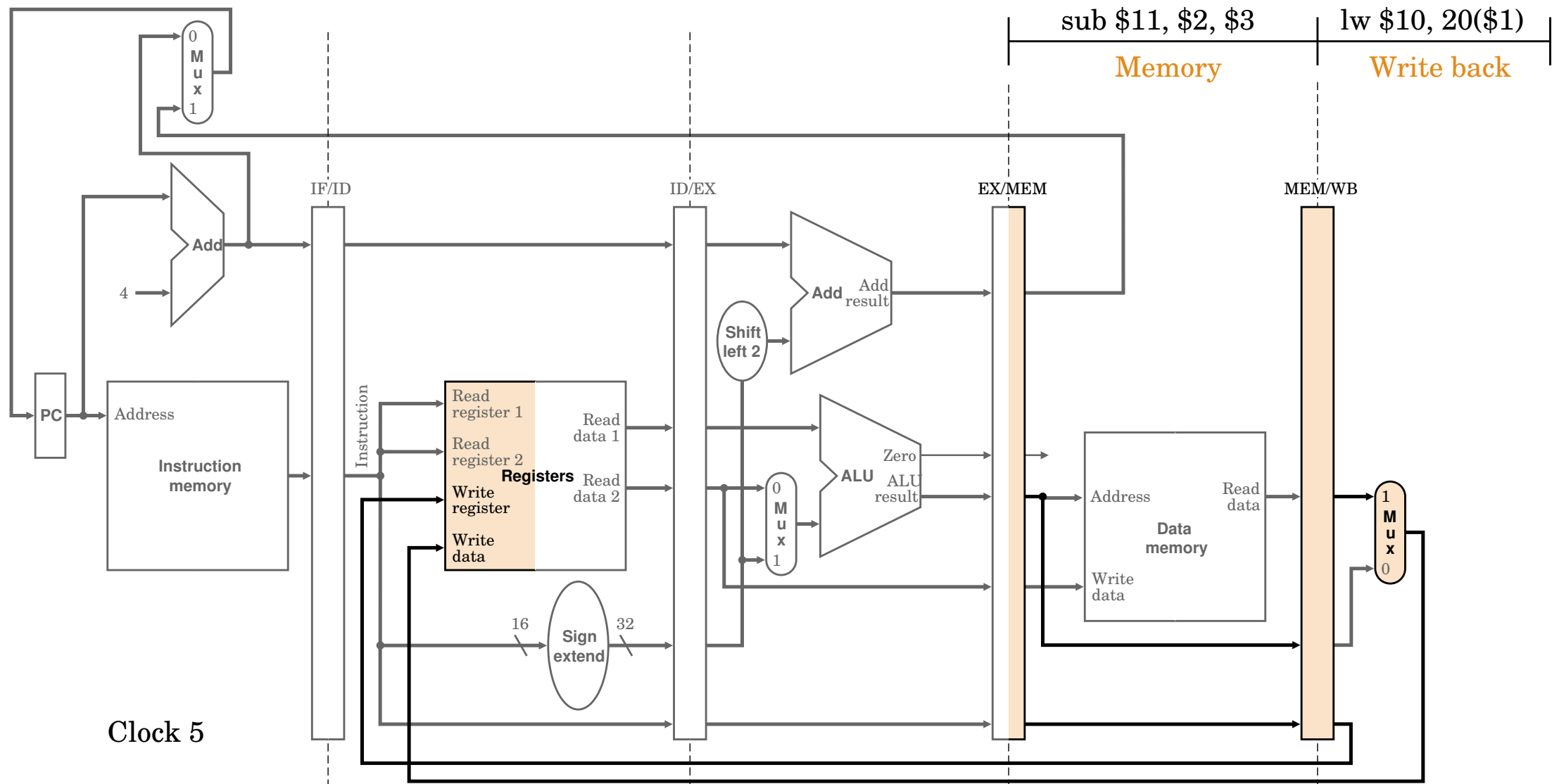
Faza 3: ID-sub & EX-load

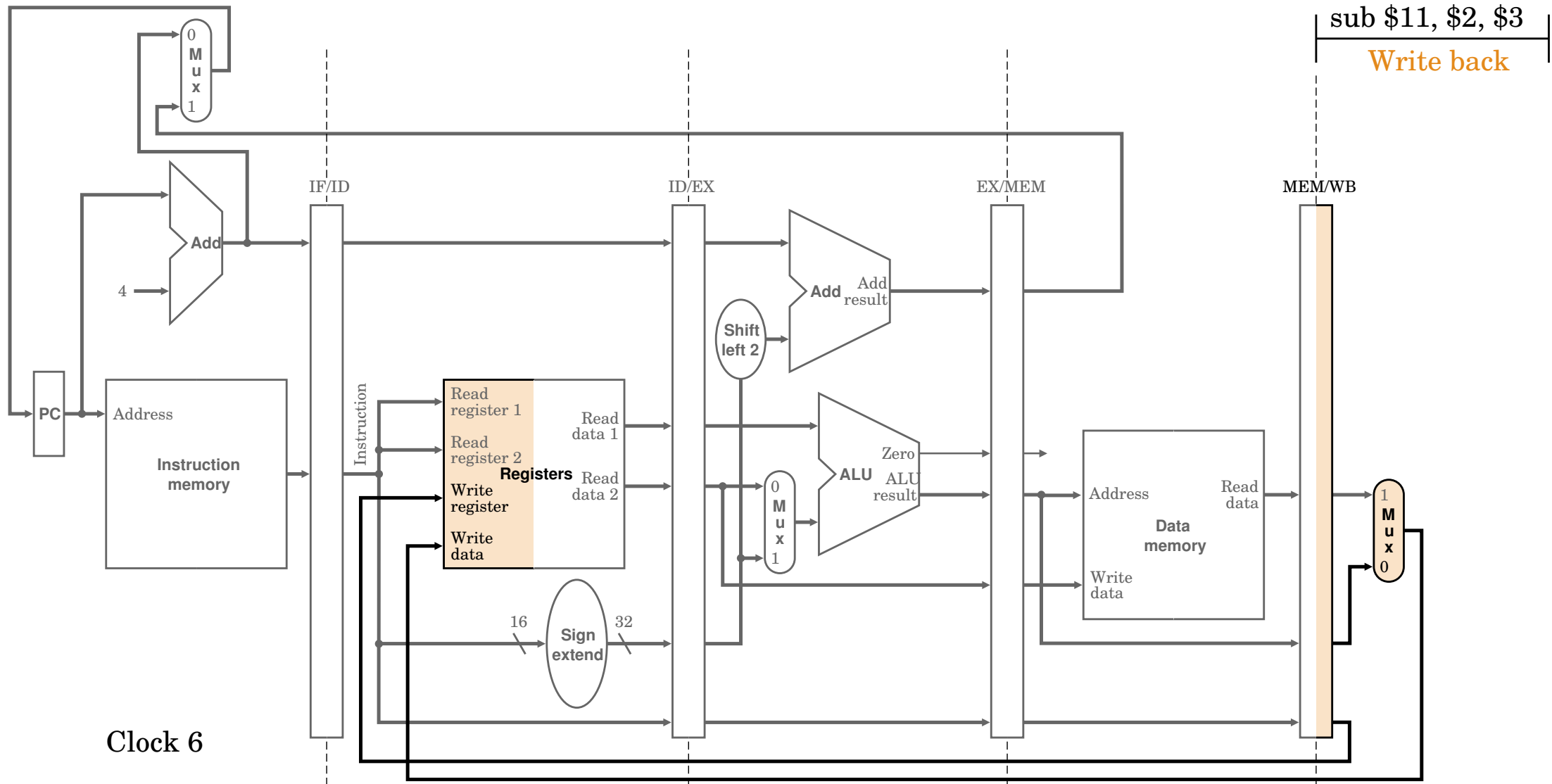




Faza 4: EX-sub & MEM-load

Faza 5: MEM-load & WB-load





Faza 6: WB-sub

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: Prezintă 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 și toți ceilalți regiștri 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 instrucțiune se folosesc în faze pipeline distincte, deci unele trebuiesc *propagate* de la o fază la alta.

Semnale de control la pipeline

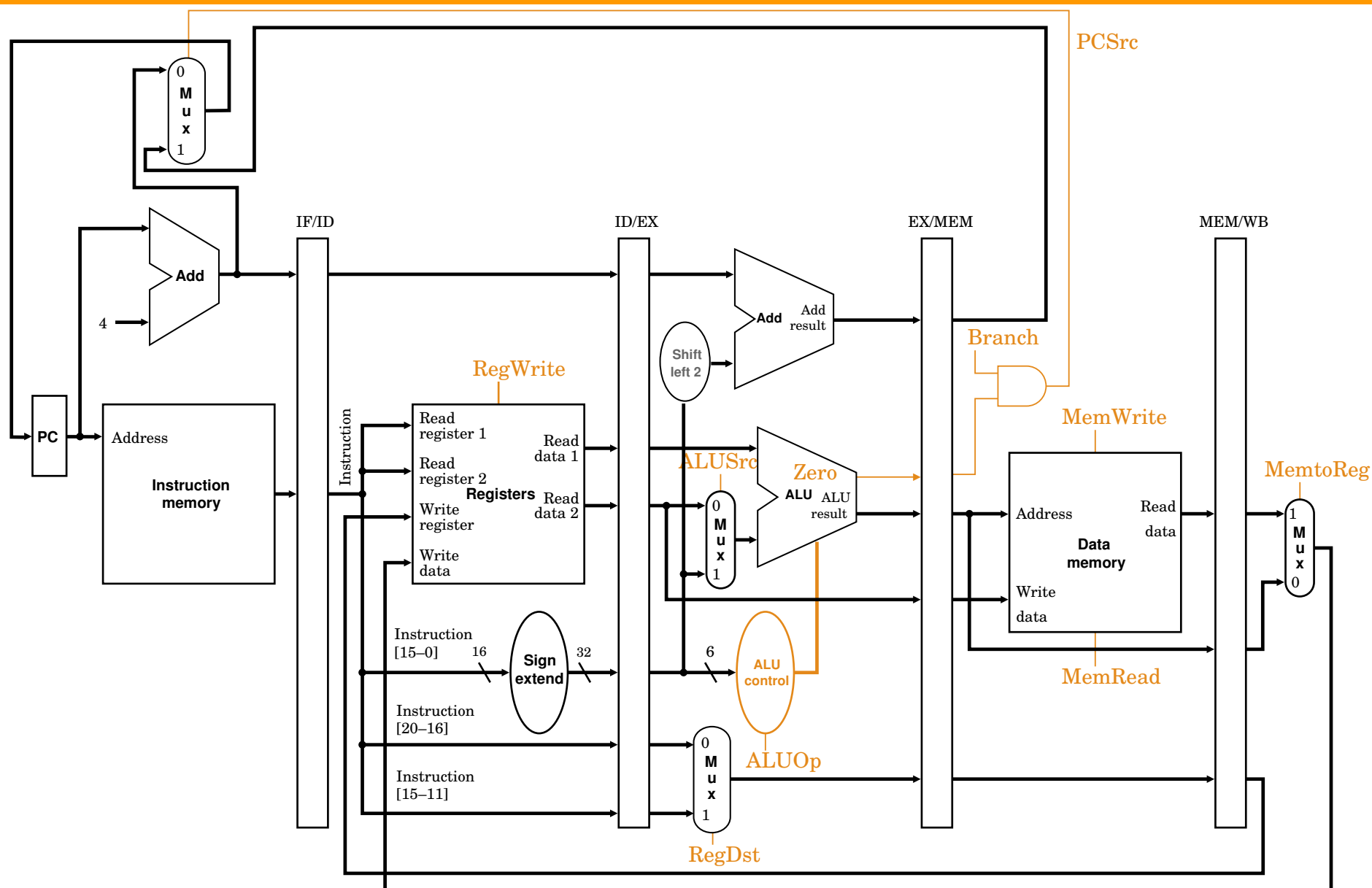


Figura prezintă *calea de date cu pipeline* cu *semnalele de control* necesare. (Versiune simplificată, ignorând hazardurile.)

Controlul cu pipeline

Semnale de control:

- Folosim semnalele de control de la implementarea cu 1 ciclu (Fila 7.41), împărțite 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



O implementare simplă de control pentru pipeline:

-
- The diagram illustrates the data flow in a 5-stage MIPS pipeline. It shows four stages: IF/ID, ID/EX, EX/MEM, and MEM/WB. A Control unit (orange oval) is connected to the ID/EX stage. The pipeline registers (WB, M, EX, M, WB) are shown as vertical bars. The flow of instructions is indicated by orange arrows. The diagram shows multiple instructions being processed simultaneously, with the first instruction in the IF/ID stage and subsequent instructions in the ID/EX, EX/MEM, and MEM/WB stages. The Control unit is connected to the ID/EX stage, and the pipeline registers are connected to the EX/MEM and MEM/WB stages. The diagram shows the flow of instructions through the pipeline, with the first instruction in the IF/ID stage and subsequent instructions in the ID/EX, EX/MEM, and MEM/WB stages.

Semnale de control la pipeline

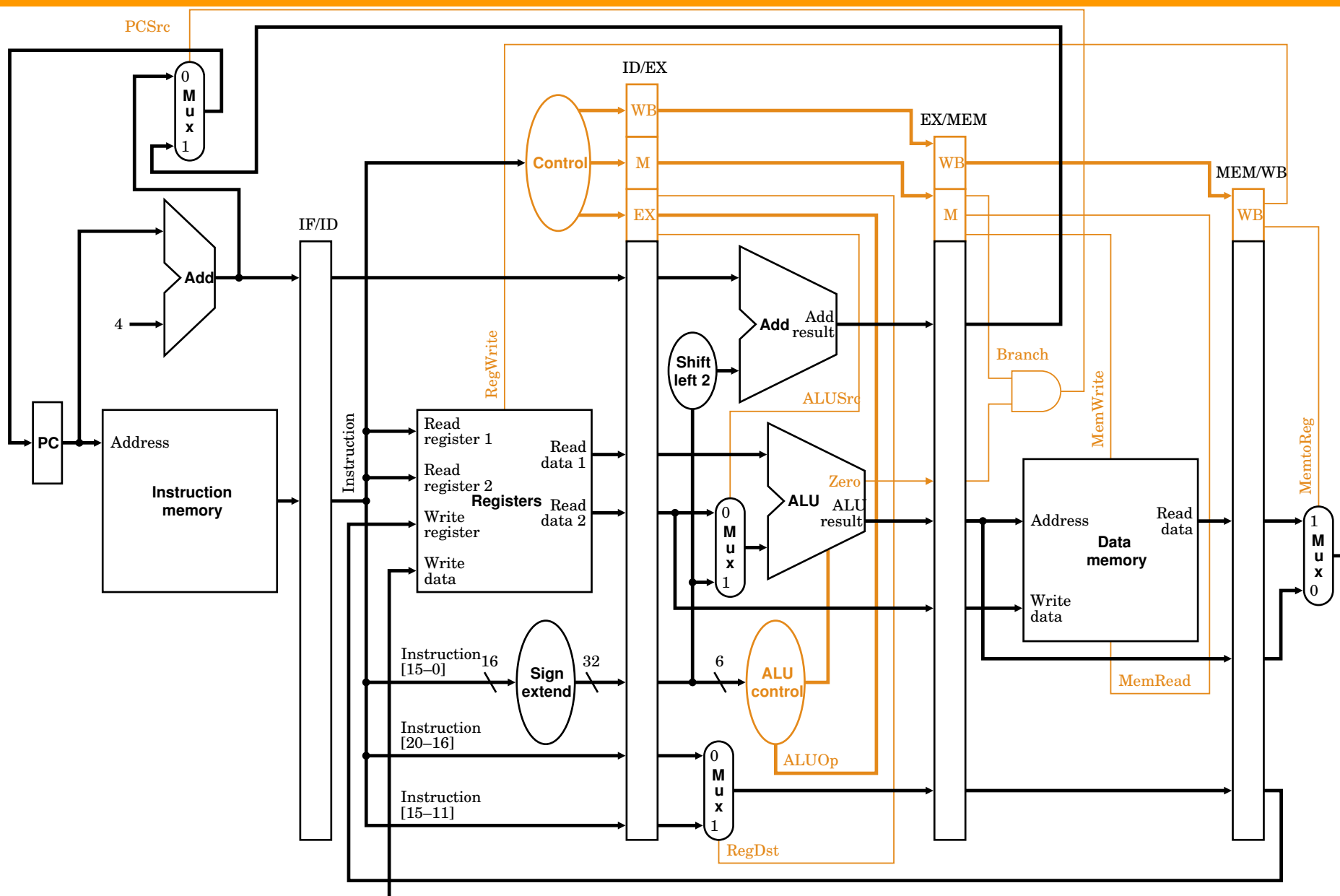


Figura prezintă *calea de date cu pipeline* cu *semnalele de control* necesare *conectate*. (Versiune simplificată, ignorând hazardurile.)



Controlul cu pipeline

Exemplu:

- Ilustrăm *activitatea din procesorul cu pipeline* obținut mai sus pe o secvență 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>

...

evidențiind și *semnalele de control* care apar.

(Ciclul 1)

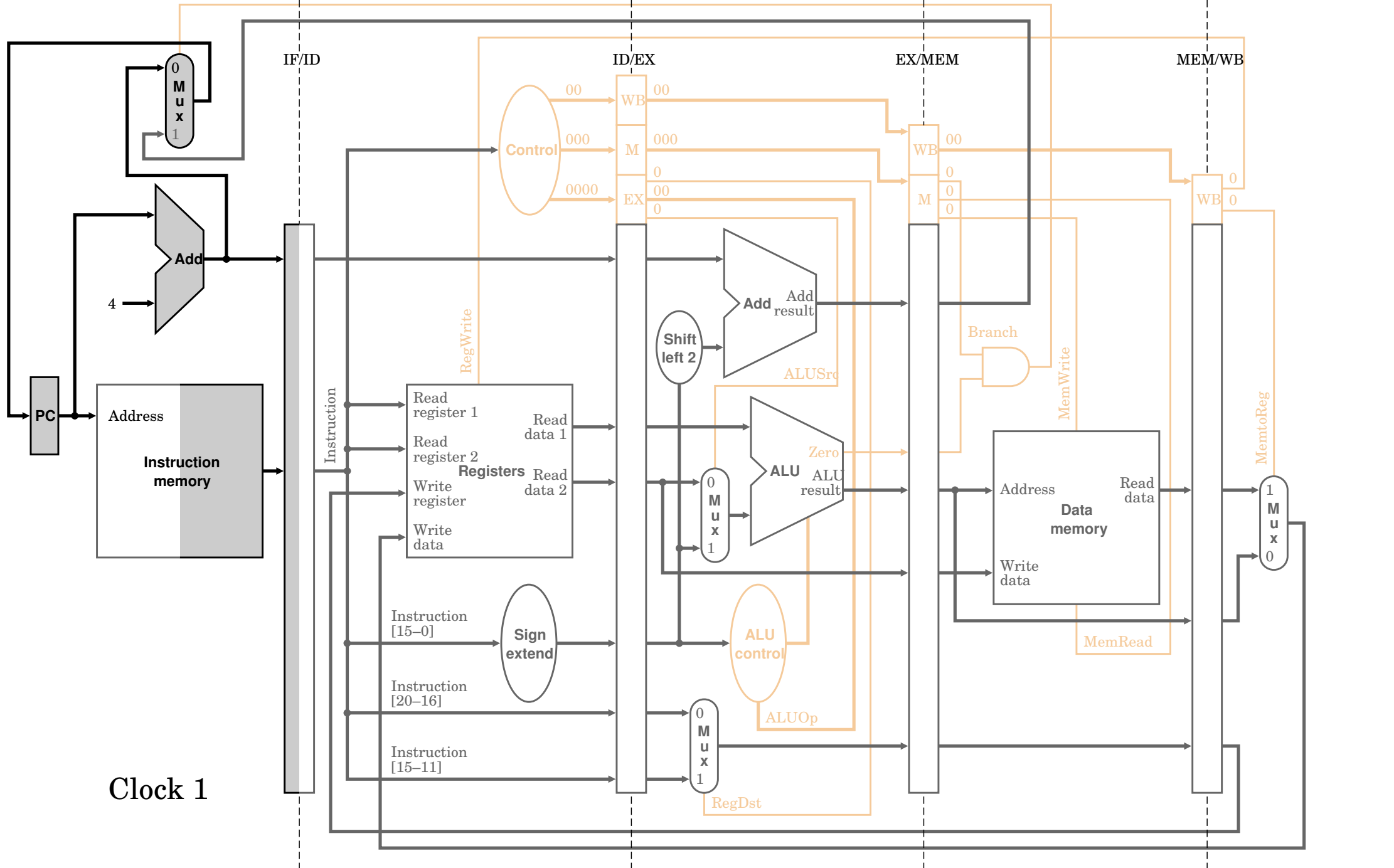
IF: lw \$10, 20(\$1)

ID: before<1>

EX: before<2>

MEM: before<3>

WB: before<4>



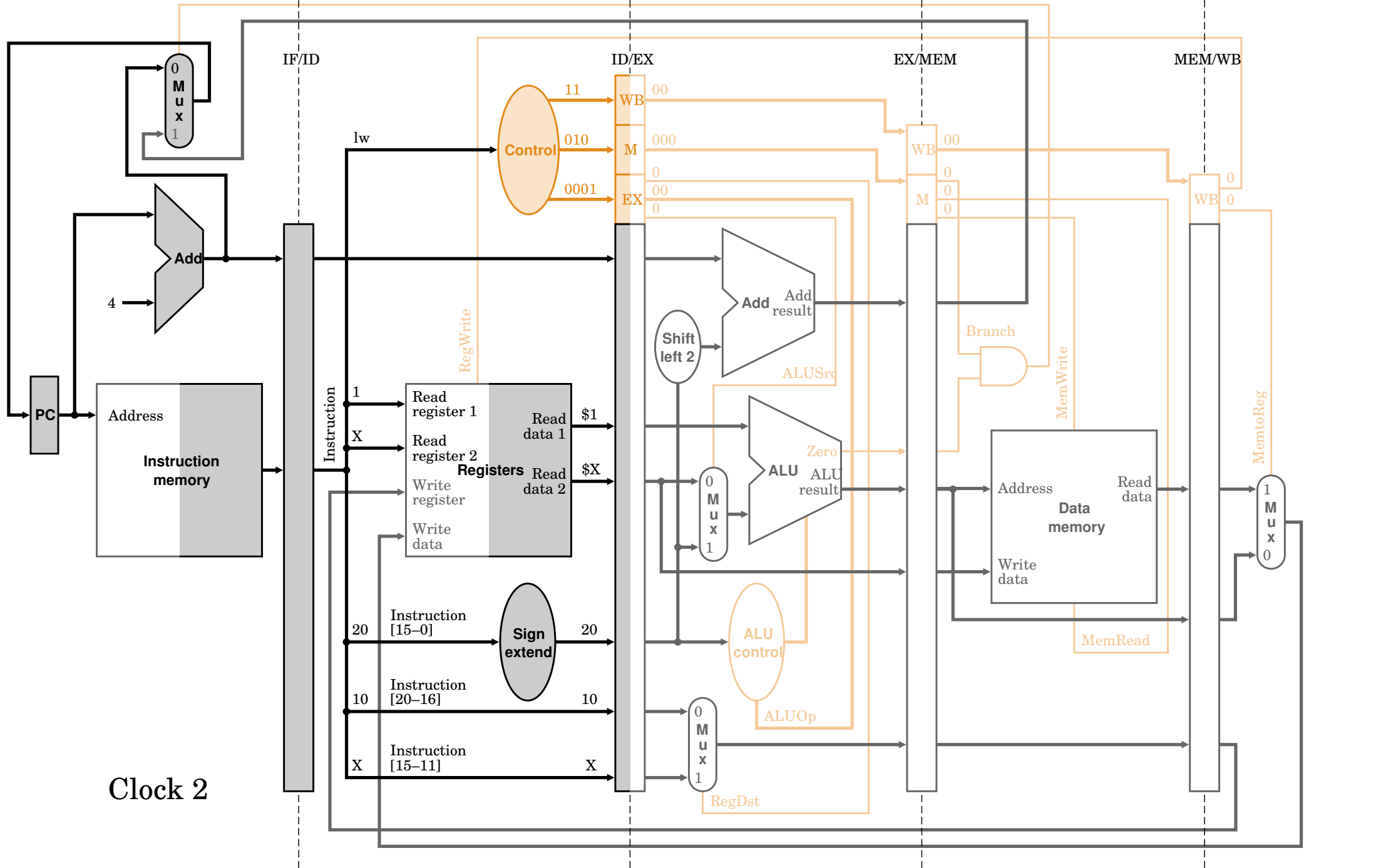
IF: sub \$11, \$2, \$3

ID: lw \$10, 20(\$1)

EX: before<1>

MEM: before<2>

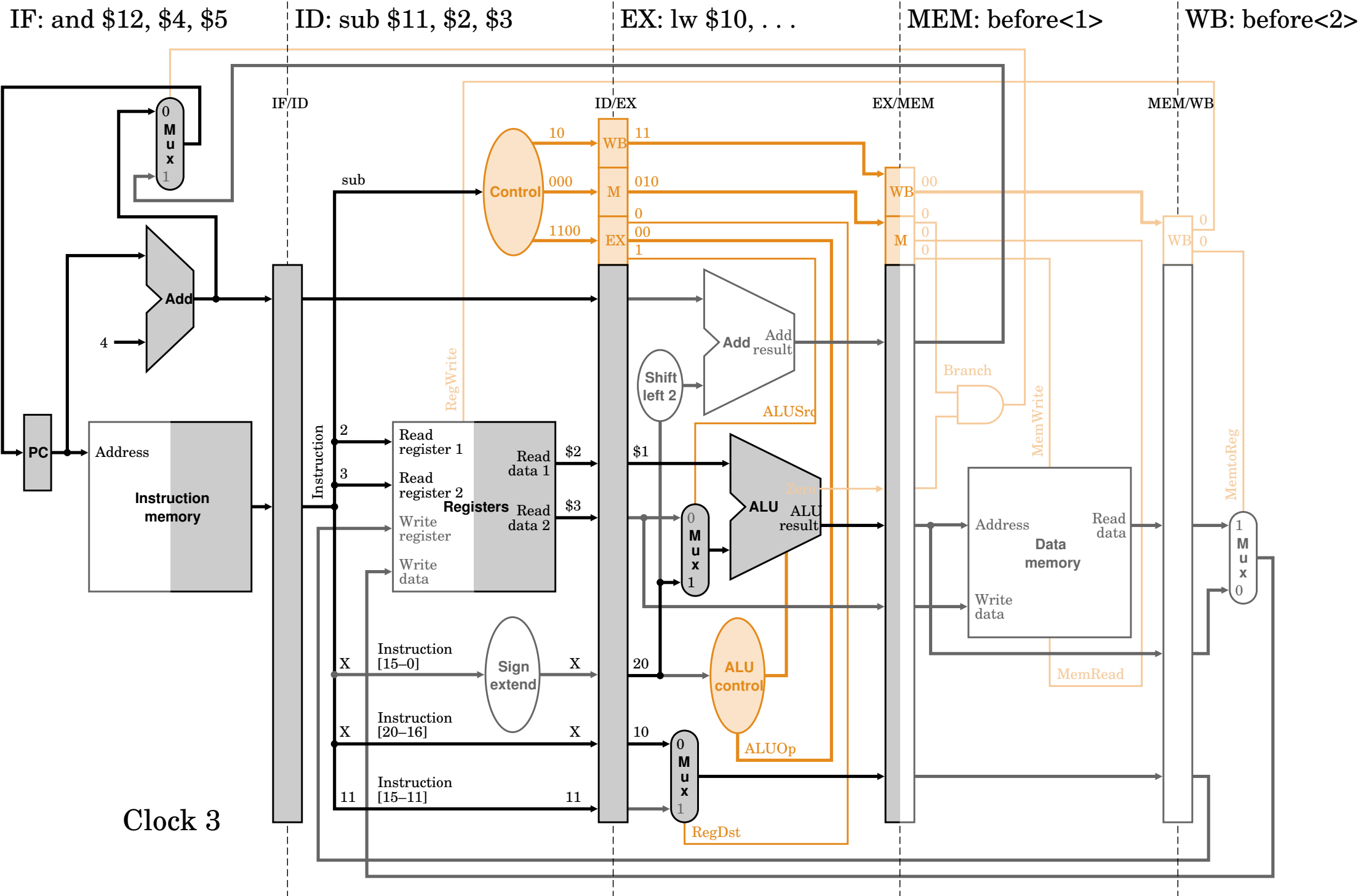
WB: before<3>



Clock 2

(Ciclul 2)

(Ciclul 3)



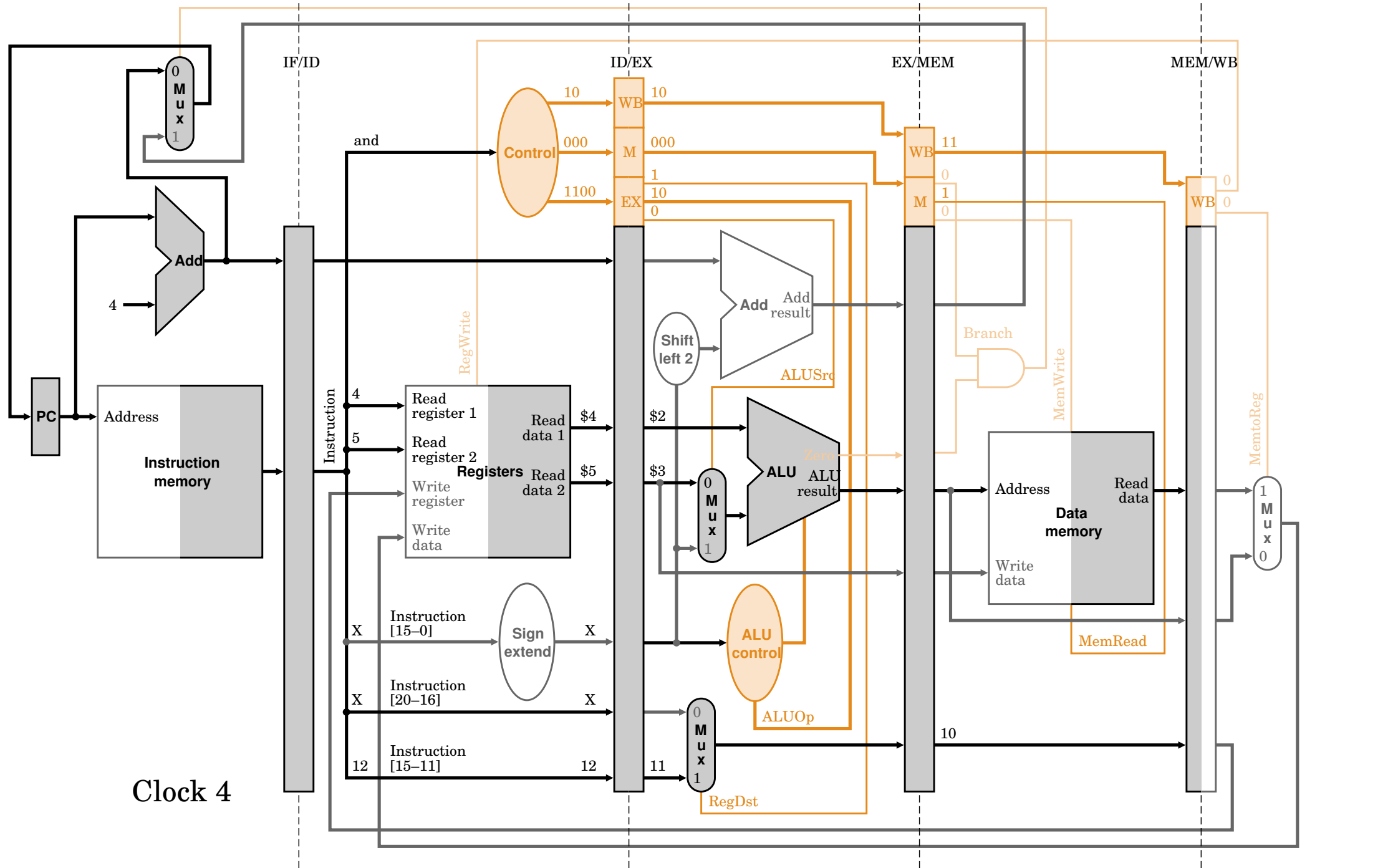
IF: or \$13, \$6, \$7

ID: and \$12, \$2, \$3

EX: sub \$11, ...

MEM: lw \$10, ...

WB: before<1>



(Ciclul 4)

(Ciclul 5)

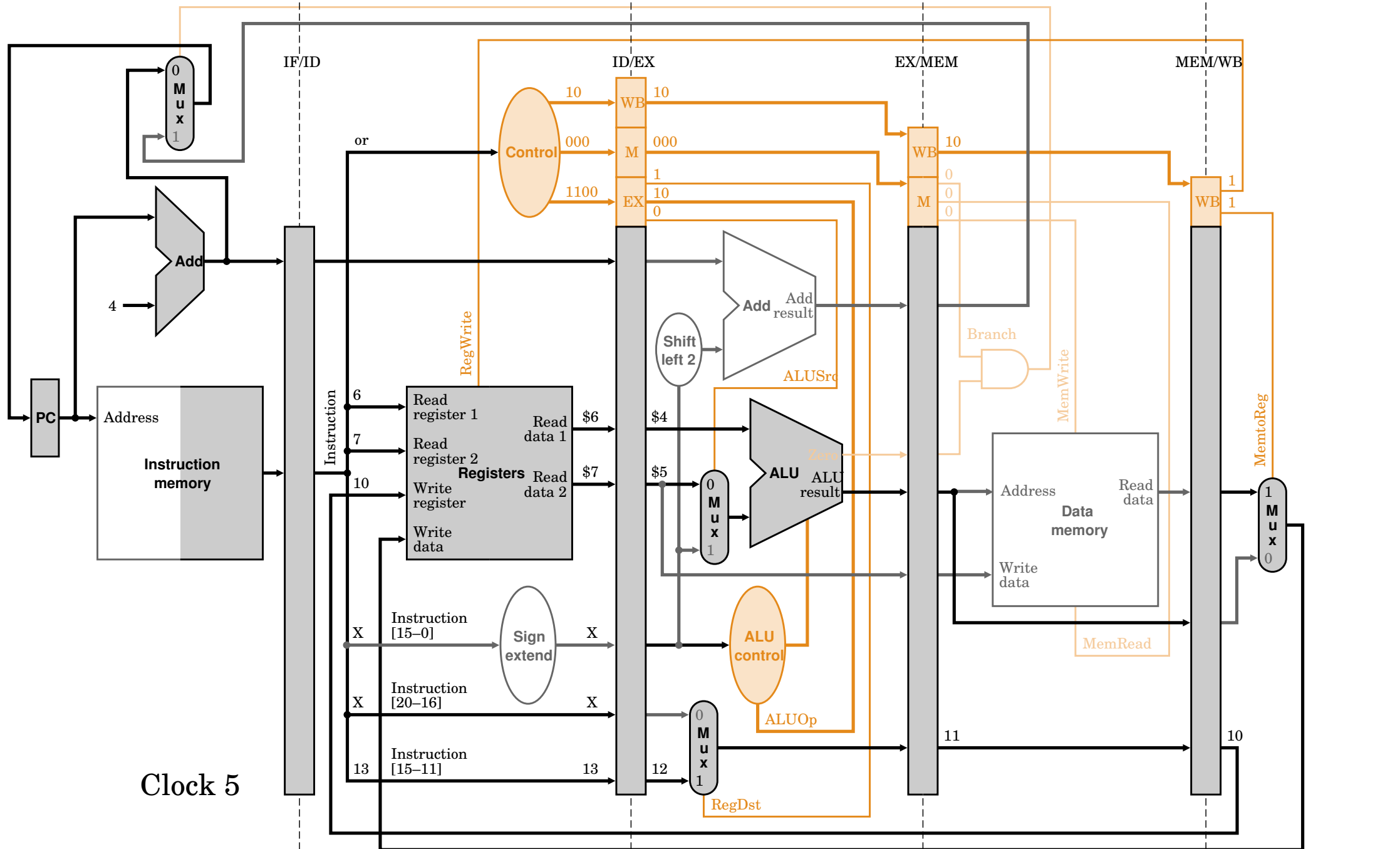
IF: add \$14, \$8, \$9

ID: or \$13, \$6, \$7

EX: and \$12, ...

MEM: sub \$11, ...

WB: lw \$10, ...



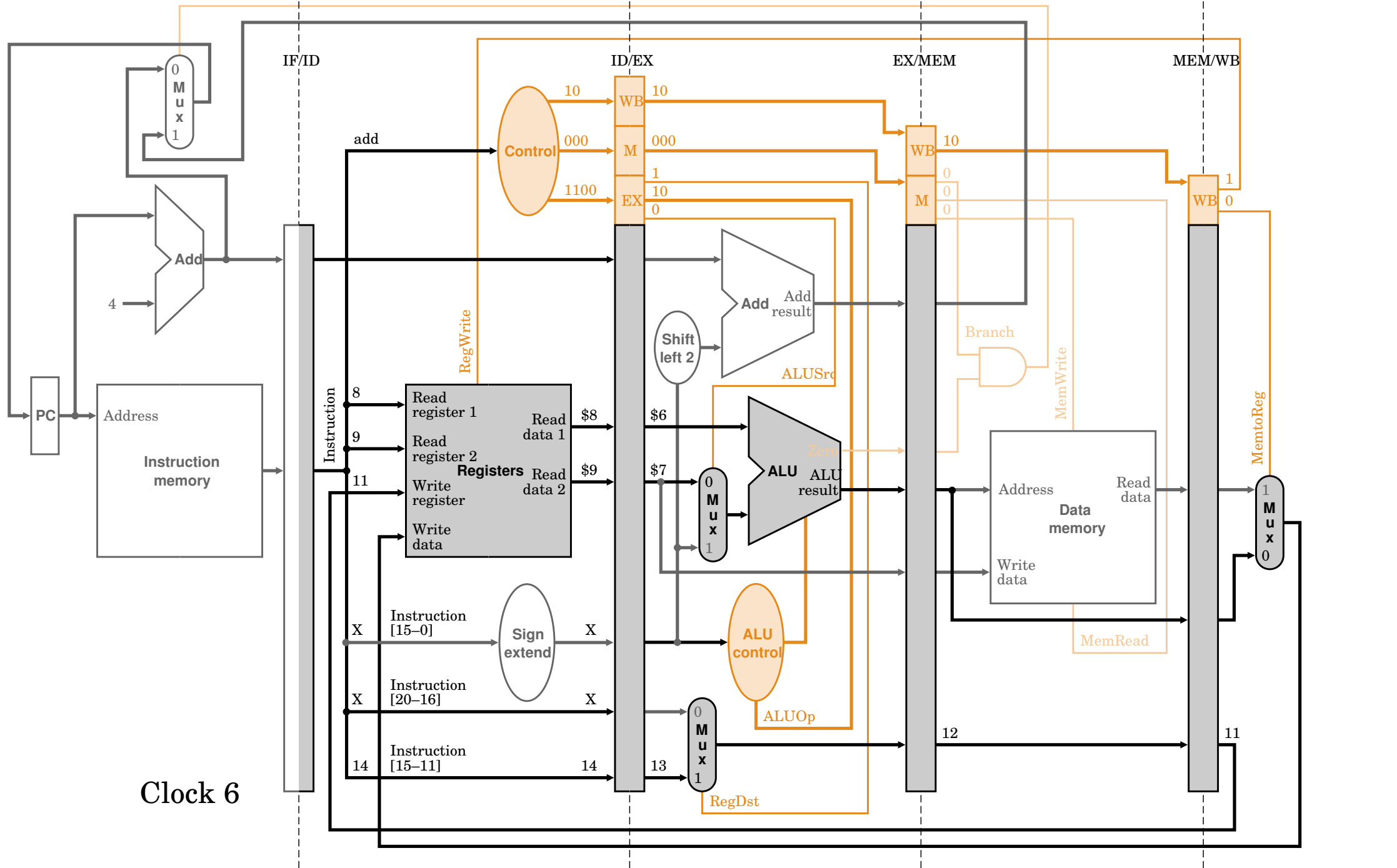
IF: after<1>

ID: add \$14, \$8, \$9

EX: or \$13, ...

MEM: and \$12, ...

WB: sub \$11, ...



Clock 6

(Ciclul 6)

(Ciclul 7)

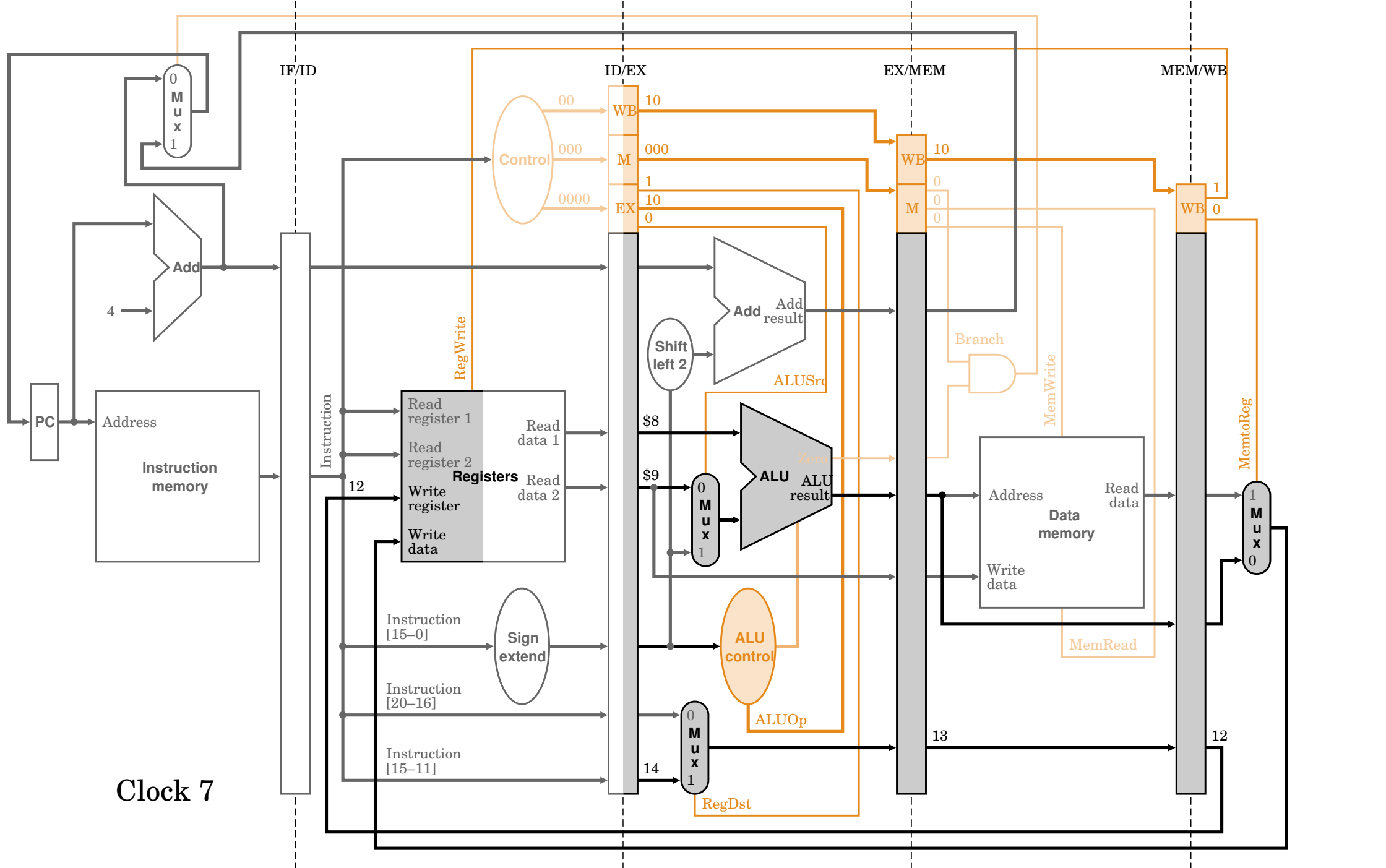
IF: after<2>

ID: after<1>

EX: add \$14, ...

MEM: or \$13, ...

WB: and \$12, ...



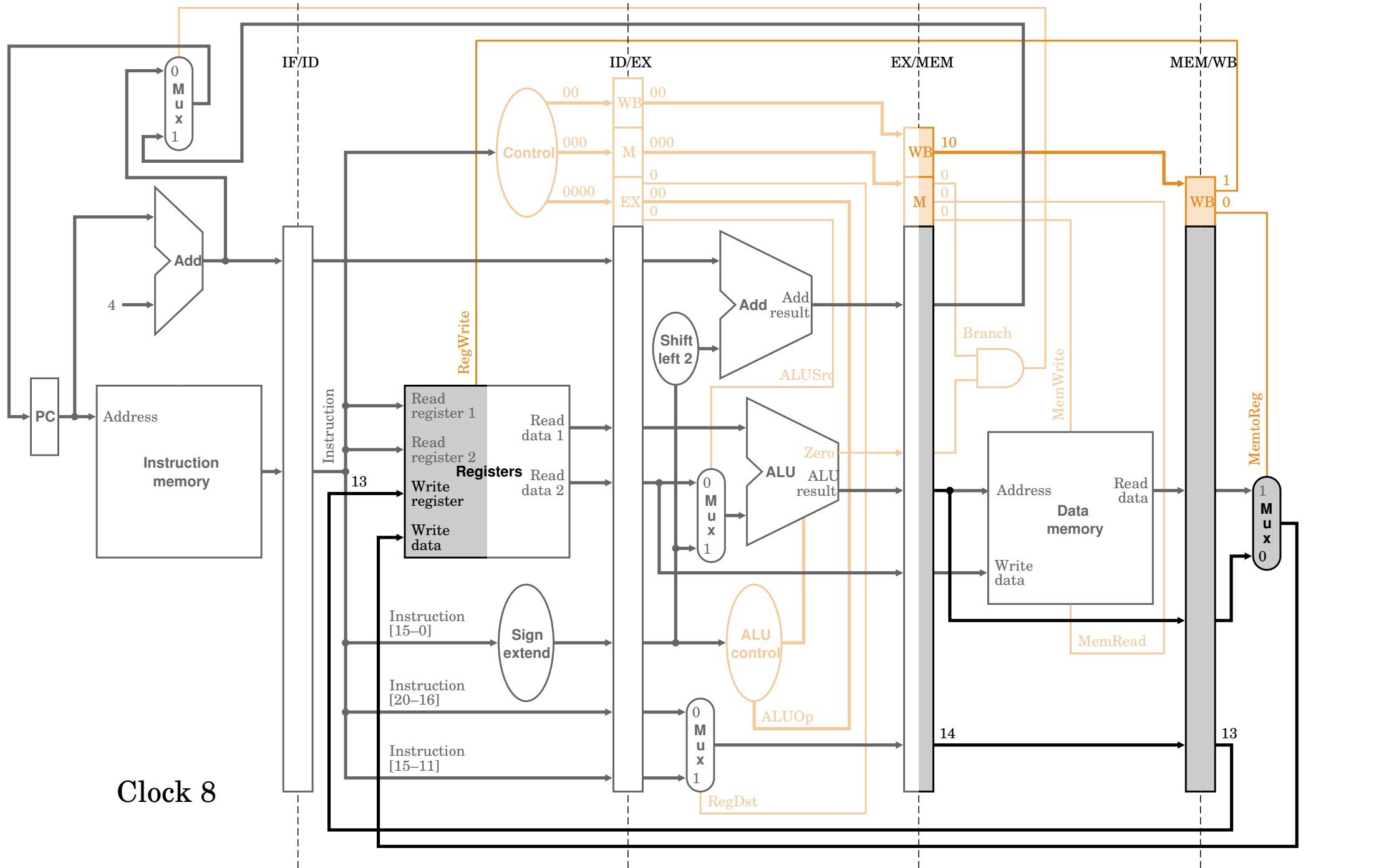
IF: after<3>

ID: after<2>

EX: after<1>

MEM: add \$14, ...

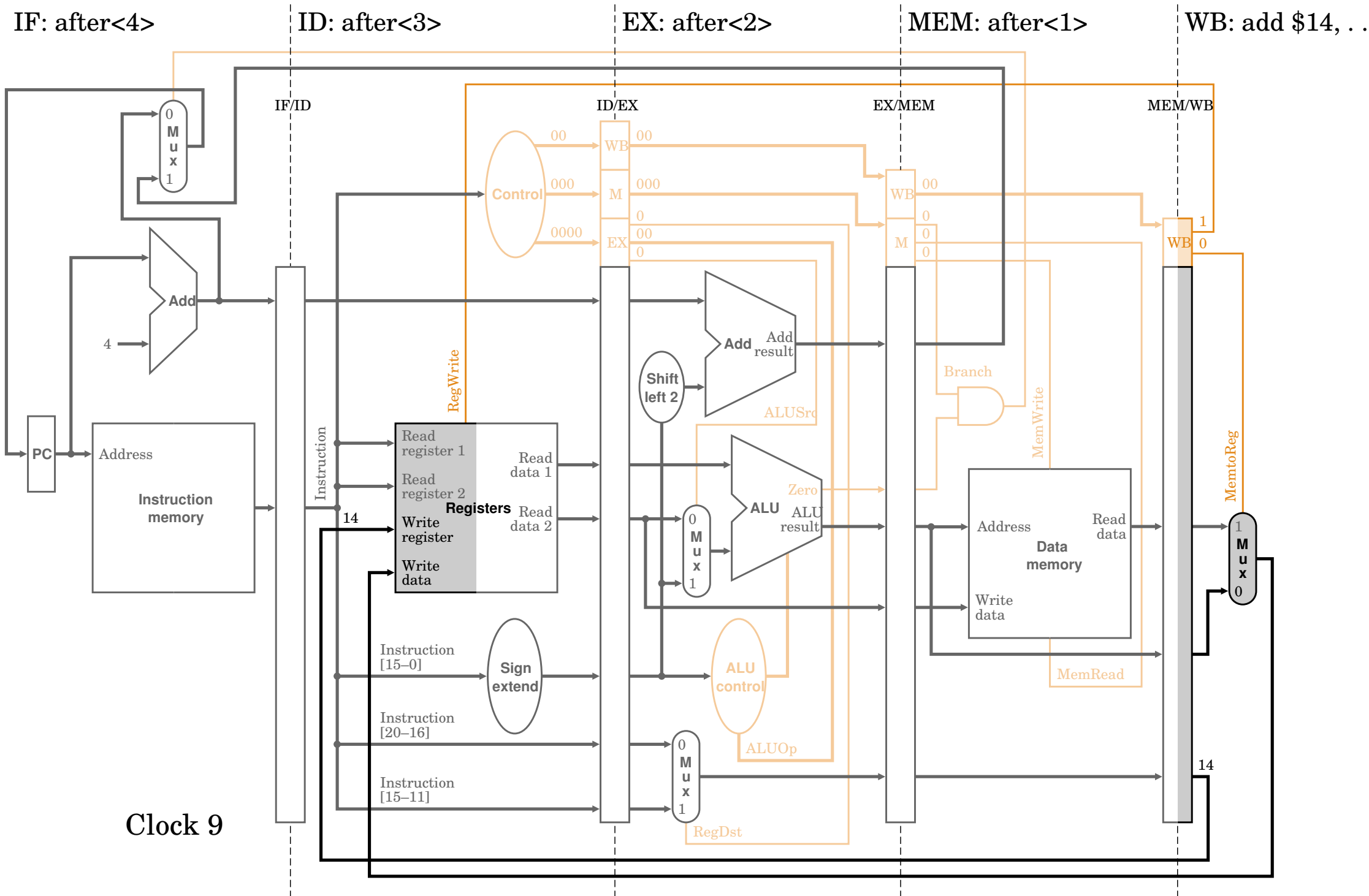
WB: or \$13, ...



Clock 8

(Ciclul 8)

(Ciclul 9)



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.*