

Lecția 7:

Procesorul:

Calea de date si controlul - I

G Ștefănescu — Universitatea București

Arhitectura sistemelor de calcul, Sem.1

Octombrie 2005—Februarie 2006

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



Procesorul: Calea de date si controlul

Cuprins:

- *Generalitati*
- Calea de date
- O prima implementare
- Implementare cu cicluri multiple
- Microprogramare
- Exceptii
- Concluzii, diverse, etc.



Generalitati

Subset MIPS: Vom prezenta o implementare de procesor care cuprinde un *subset de bază* de instrucțiuni MIPS, anume:

- Tip I - instrucțiuni *aritmetice și logice*: `add`, `sub`, `and`, `or`, `slt`
- Tip II - instrucțiuni de *transfer* în și din memorie: `lw`, `sw`;
- Tip III - instrucțiuni *condiționale*: `beq`, `j`

Nota: Nu contine operatii de inmultire, impartire, operatii in virgula mobila, etc., dar principiile de constructie ale procesorului sunt similare.



..Generalitati

Pasi de procesare:

- Primii pași sunt comuni pentru toate instrucțiunile de mai sus:
 - Se trimite contorul de instrucțiuni PC în memorie unde este programul și se *extrage instrucțiunea* de executat;
 - Se *citesc* unul ori doi *registri* asupra cărora operează instrucțiunea;
- Ce urmează depinde de tipul instrucțiunii, dar, din fericire, există anumite asemănări care uniformizează procesarea:
 - Toate operațiile *folosesc ALU* pentru calcule specifice
 - * tip I: operații aritmetice ori logice;
 - * tip II: găsirea adresei de memorie;
 - * tip III: operații pentru decizia de la test.



..Generalitati

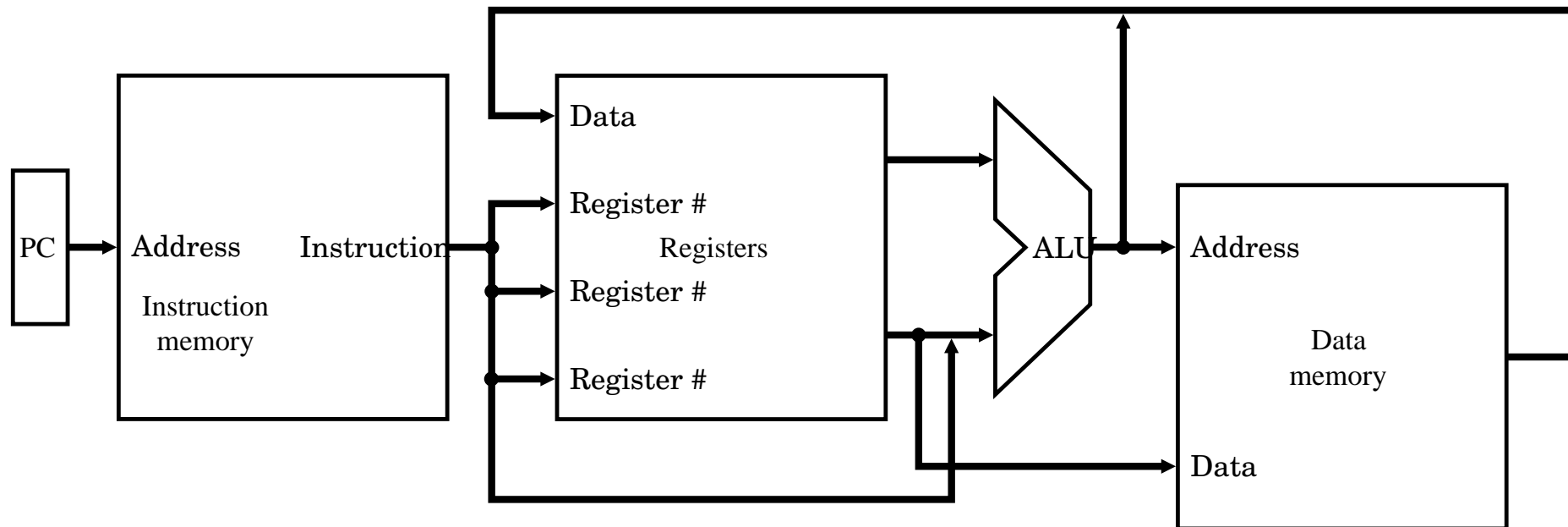
Pasi de procesare (cont.)

- Pașii de finalizare a instrucțiunilor sunt mai diferiți la cele trei tipuri, necesitând mai mult efort de uniformizare (anume, un control mai complex):
 - Pași de *finalizare*
 - * tip I: se scrie rezultatul în registru;
 - * tip II: se accesează memoria pentru citit/scriș;
 - * tip III: se decide care este noua instrucțiune.

..Generalitati

Schema de ansamblu:

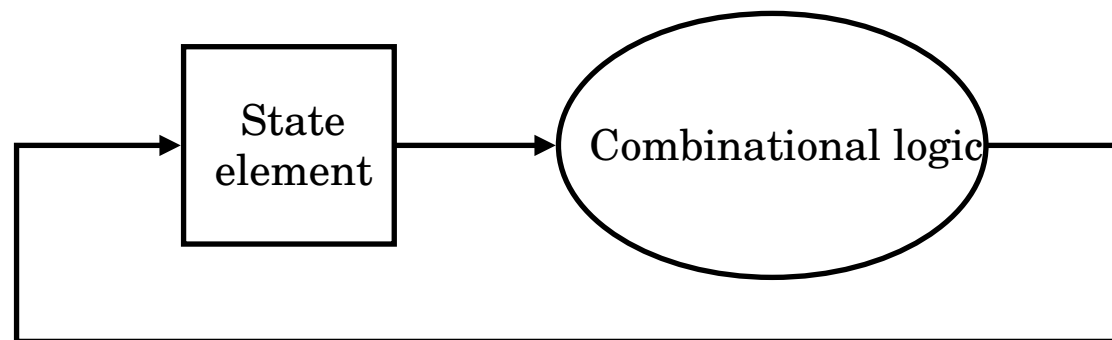
- Figura conține schema de ansamblu pentru procesor, cu principalele sale *componente* (partea pentru instrucțiunile de tip III este incomplet prezentată).



..Generalitati

Tipul circuitelor:

- Instrucțiunile din ALU sunt pur *combinazionale*.
- Cum procesorul se folosește *cu repetiție* (avem de executat mai multe instrucțiuni), avem nevoie de un *ceas* și de *circuite cu memorie*.
- Modelul teoretic este de *mașină cu stări finite*.





Procesorul: Calea de date si controlul

Cuprins:

- Generalitati
- *Calea de date*
- O prima implementare
- Implementare cu cicluri multiple
- Microprogramare
- Exceptii
- Concluzii, diverse, etc.



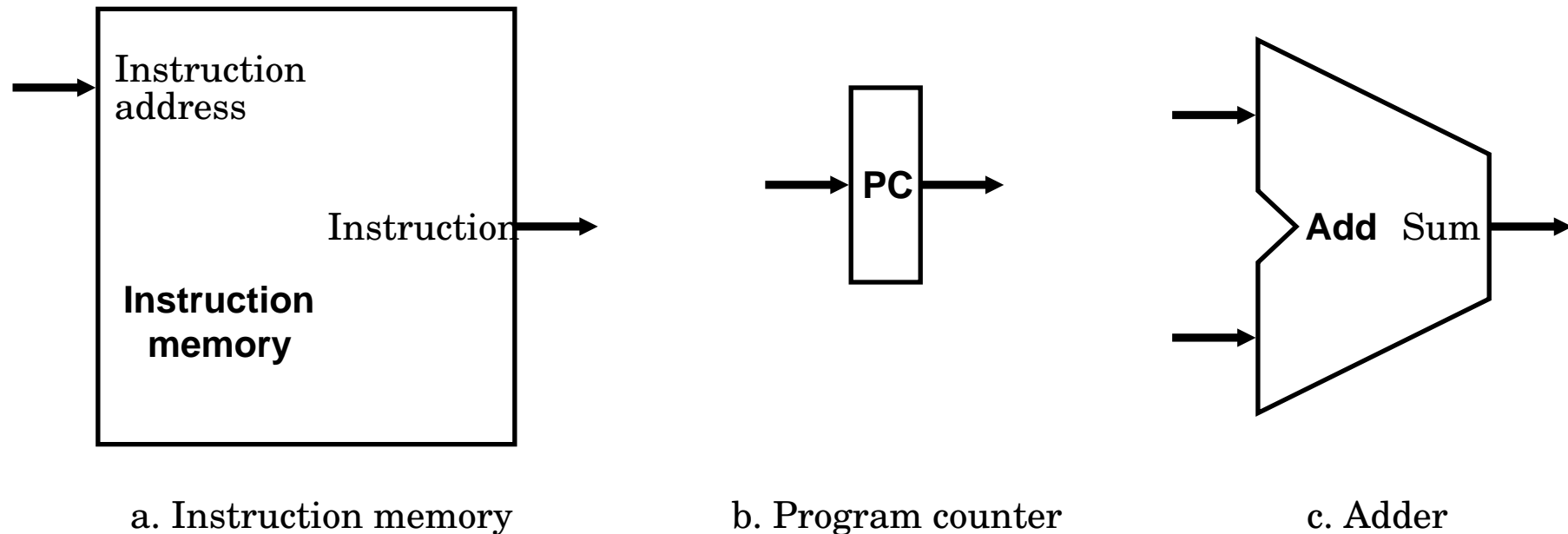
Calea de date

Calea de date: Vom exemplifica pe rând:

- cum se extrage o instrucțiune;
- cum se operează cu regiștri la instrucțiuni aritmetice și logice;
- cum se tratează operațiile de accesare a memoriei (load / store);
- cum procesăm salturile condiționate (branch) ori pe cele necondiționate (jump);
- cum integrăm componentele de mai sus *într-o unică cale de date*, folosind multiplexori cu semnale de control adecvate;
- etc.

Extragerea unei instructiuni

Extragerea unei instructiuni:



Componente pentru a accesa o instrucțiune și a calcula noua adresă:

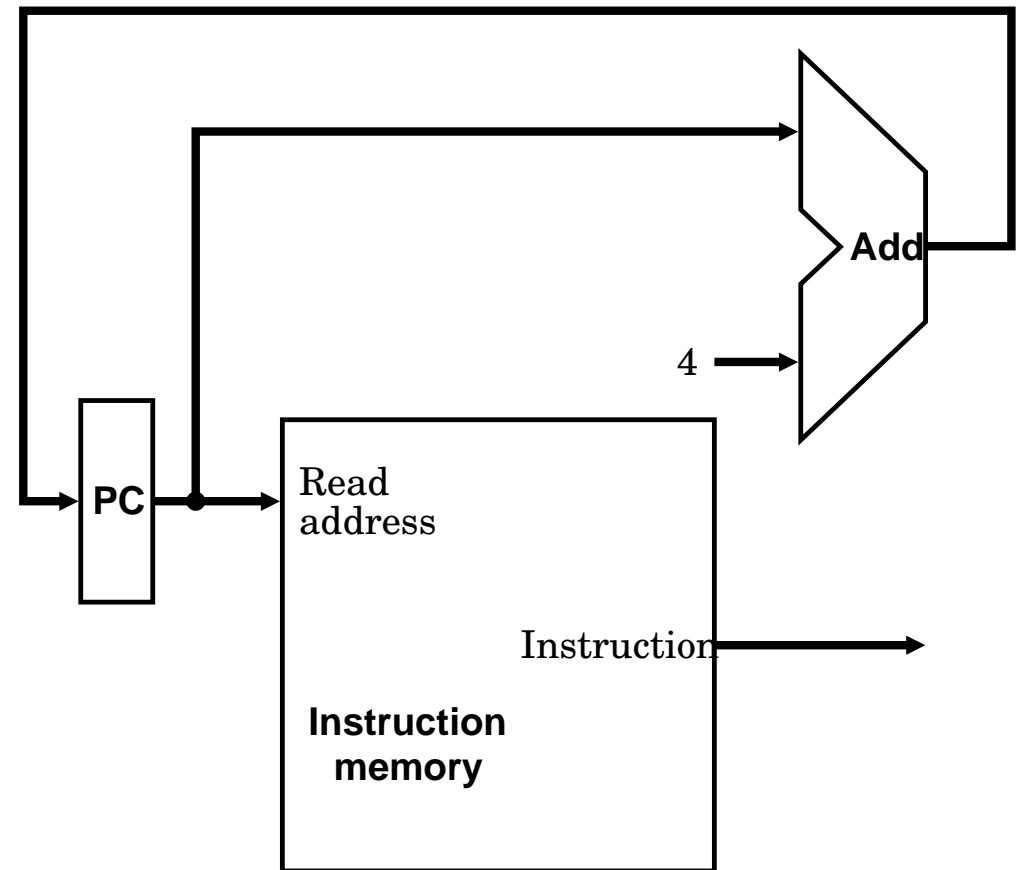
- o memorie cu instrucțiunile programului, accesată cu adresa;
- o memorie pentru contorul instrucțiunii curente *PC*;
- un circuit combinațional de sumare (pentru a calcula noua adresă)

..Extragerea unei instructiuni

Extragerea unei instructiuni (cont.)

O parte din calea de date
pentru extragerea
instrucțiunii:

- instrucțiunile sunt pe 32 biți, deci 4 octeți;
- adresa instrucțiunii următoare (fără salt) se obține adunând 4;
- componenta PC memorează valoarea pentru ciclul următor





..Extragerea unei instructiuni

Extragerea unei instructiuni (cont.)

- Fie

$IM : 32 \rightarrow 32$ – memoria cu instrucțiuni;

$ALU_SUM : 32 + 32 \rightarrow 32$ – un sumator;

$PC : 32 \rightarrow 32$ – memorie pentru contorul de instrucțiuni, și

$FOUR : 0 \rightarrow 32$ – o componentă ce produce la ieșire numărul 4.

- Componenta de mai sus este

$$C01 = [PC \cdot \wedge_2^{32} \cdot (IM \star (FOUR \star I_{32}) \cdot ALU_SUM)] \uparrow^{32} : 0 \rightarrow 32$$

unde avem

- La intrare: nimic;
- La ieșire: o poartă de 32b pentru instrucțiunea rezultată.



Instructiuni aritmetice si logice

Instructiuni aritmetice si logice:

- Instrucțiunile de tip I considerate sunt în *format R*, i.e., folosesc *2 regiștri* pentru argumente și *1 registru* pentru rezultat.
- Exemplu tipic este

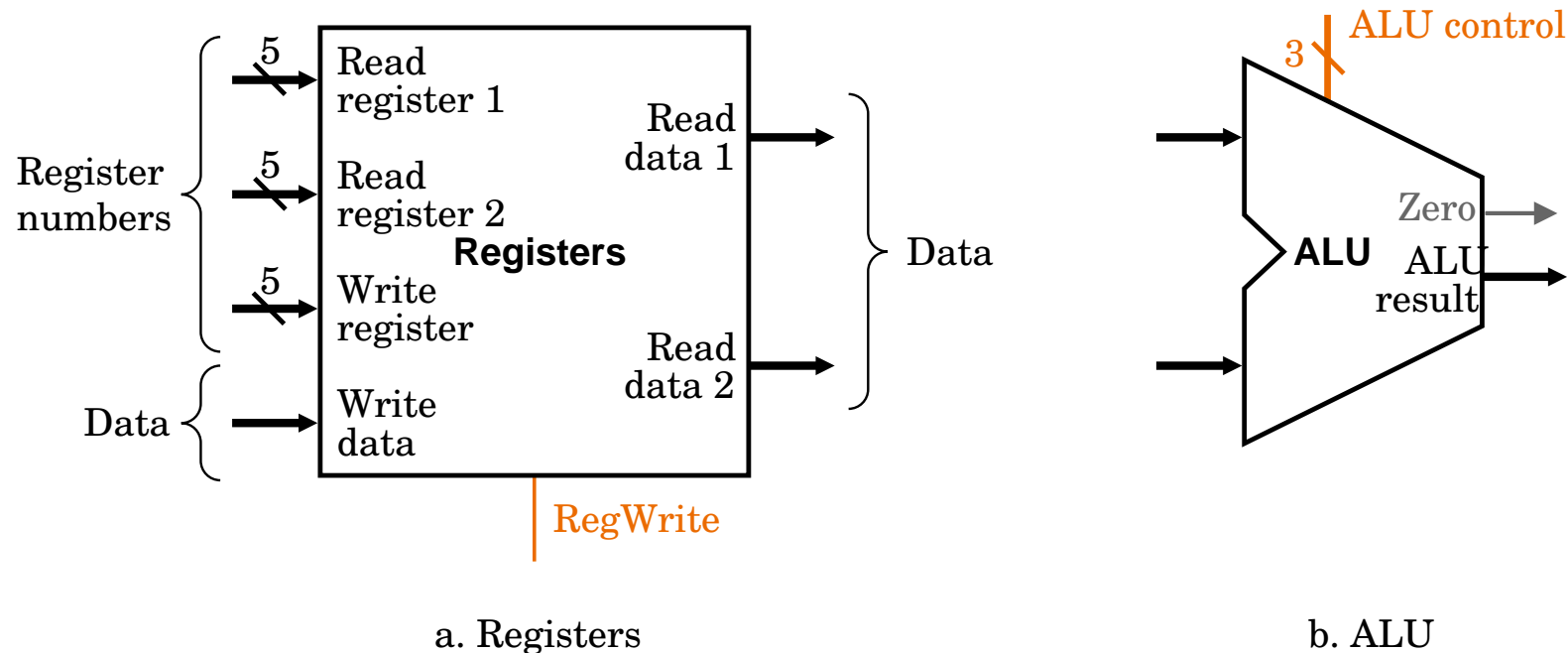
```
add $t1,$t2,$t3;
```

Se citesc regiștri \$t1, \$t2 și rezultatul se pune în \$t3.

- Folosim o memorie cu regiștri “*register file*”, care are 32 de regiștri cu 2 porți de citire și una de scriere (toate în paralel).
- Pentru *citirea* unui registru se dă *numărul registrului*.
- Pentru *scrierea* unui registru se dă *numărul registrului* și *data* de scris.

..Instrucțiuni aritmetice și logice

Instrucțiuni aritmetice și logice (cont.)



Componente necesare pentru a opera pe regiștri:

- o memorie cu regiștri RF (Register File) ca în figură;
- o componentă ALU de calcul.



..Instrucțiuni aritmetice și logice

Instrucțiuni aritmetice și logice (cont.)

- Memoria de regiștri M permite 2 citiri și o scriere simultane; mai exact tipul este

$$M : 5 + 5 + 5 + 32 + 1 \rightarrow 32 + 32$$

unde, la intrare avem

- 3 porți de 5b pentru a indica 3 regiștri (operând în paralel);
- 32b pentru o dată de scris;
- 1b de control pentru scriere `RegWrite`;

iar la ieșire avem

- 2 porți de 32b pentru cele 2 date citite (operând în paralel);



..Instructiuni aritmetice si logice

Instructiuni aritmetice si logice (cont.)

- Unitatea aritmetică și logică *ALU1* se obține din cea standard *ALU*, neglijând ieșirile de overflow și carryout, anume

$$ALU1 = ALU \cdot (I_{32+32} \star \wedge_0^1 \star \wedge_0^1)$$

deci *ALU1* are tipul

$$ALU1 : 32 + 32 + 3 \rightarrow 1 + 32$$

unde, la intrare avem

- două porți de 32b pentru 2 argumente;
- 3b de control ALU operation indicând operația;

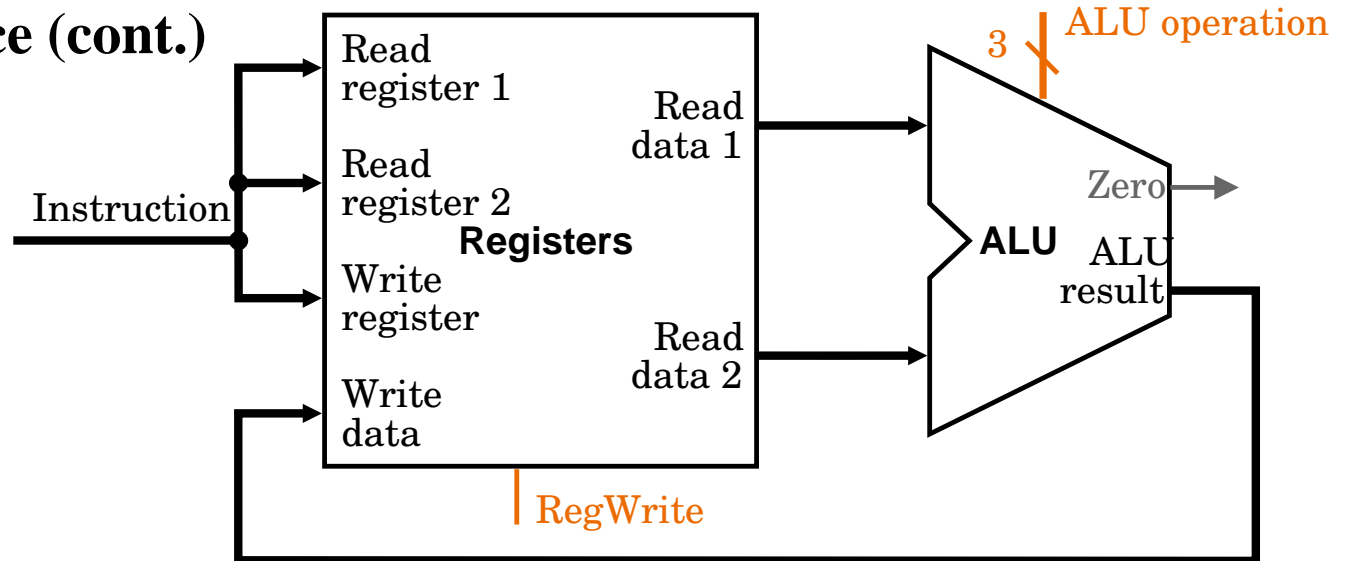
iar la ieșire avem

- 1b indicând dacă rezultatul este zero;
- 32b pentru rezultatul operației.

..Instrucțiuni aritmetice si logice

Instrucțiuni aritmetice si logice (cont.)

O parte din calea de date pentru instrucțiunile de format *R*:



Componenta rezultată $C02 = [(I_{15} \star^4 X^{32}) \cdot (RF \star I_3) \cdot ALU1] \uparrow^{32}$ are tipul

$$C02 : 15 + 4 \rightarrow 1$$

unde avem

- la intrare: 3 porți de 5b pentru 3 regiștri; 1b control scriere; 3b selecție operație;
- la ieșire: 1b, dacă rezultatul este zero.



Instructiuni de transfer

Instructiuni de transfer:

- Instrucțiunile de tip II considerate sunt în *format I*, i.e., folosesc *1 registru* pentru dată și *1 registru* + *offset* pentru adresă (ultimul aflat pe 16b din instrucțiune, pozițiile 15-0).

- Primul exemplu este

```
sw $t1, const($t2);
```

Se calculează adresa de memorie $\$t2 + \text{const}$ și acolo se scrie data din $\$t1$.

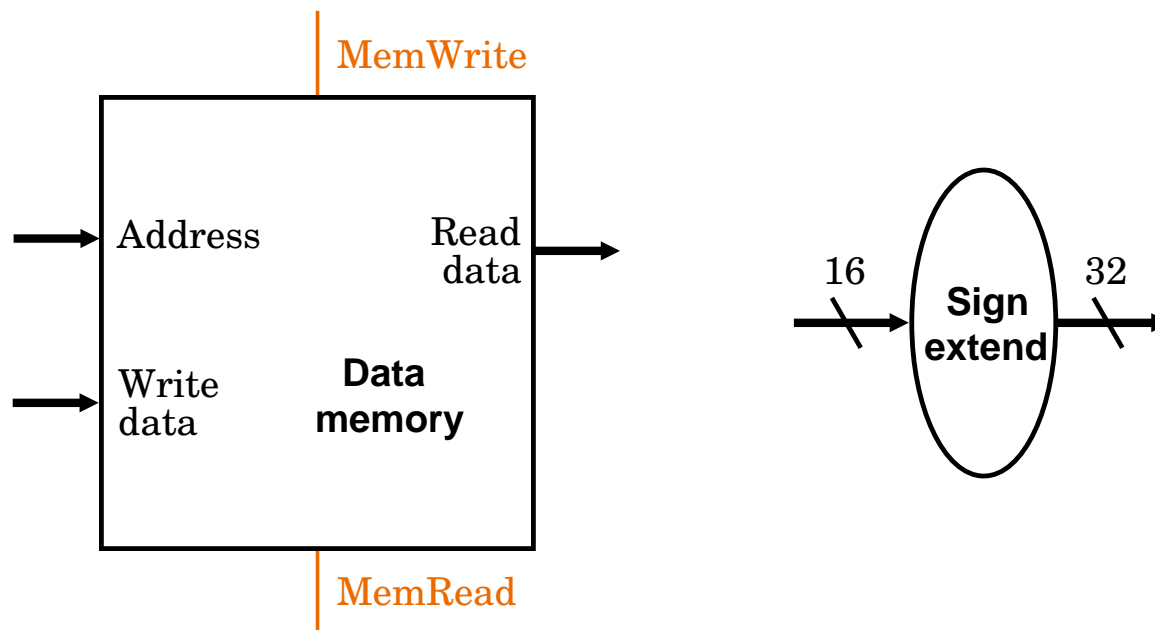
- Al doilea este

```
lw $t1, const($t2);
```

Se calculează adresa $\$t2 + \text{const}$ și data de acolo se încarcă în $\$t1$.

..Instrucțiuni de transfer

Instrucțiuni de transfer (cont.)



a. Data memory unit

b. Sign-extension unit

Componente pentru instrucțiuni de transfer suplimentare:

- o memorie cu 1 port de citire și 1 port de scriere *DM*
- o componentă pentru extensia cu semn de la 16 la 32 biți.

..Instrucțiuni de transfer

Instrucțiuni de transfer (cont.)

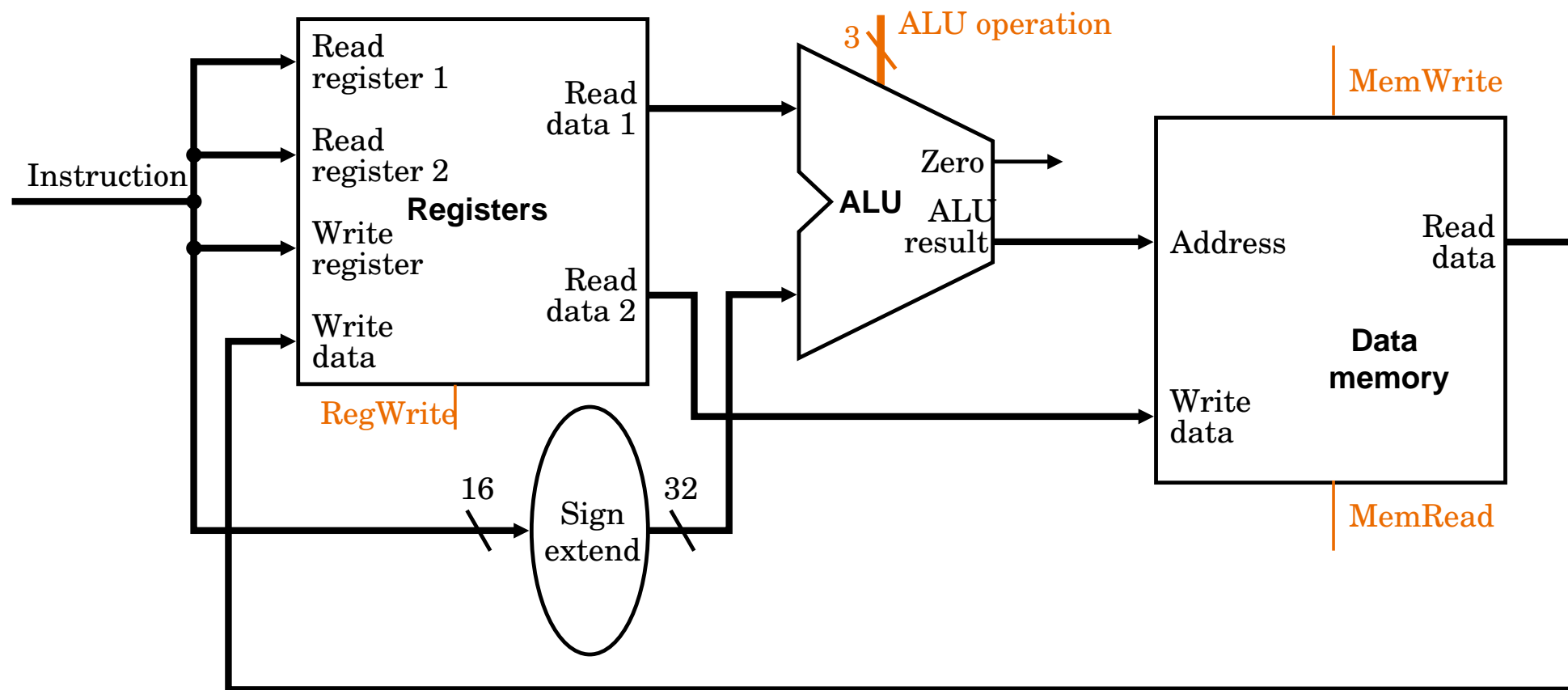


Figura conține o parte C03 : $32 + 6 \rightarrow 1$ din calea de date pentru instrucțiuni de transfer: intrări = instrucțiunea (32b) + control (6b); ieșiri = testul la zero (1b).



..Instrucțiuni de transfer

Instrucțiuni de transfer (cont.)

Pentru instrucțiuni `sw $t1, const($t2)` or `lw $t1, const($t2)`:

- Offset-ul de pe pozițiile 15-0 din instrucțiune se extinde la 32b și se folosește ca al 2-lea argument în ALU (primul este \$t2).
- Dacă avem `sw`, deci controlul `MemWrite` este 1 (iar `MemRead` este 0), a doua dată din registru (\$t1) se folosește pentru a scrie în memorie la adresa dată de ieșirea ALU.
- Dacă avem `lw`, deci controlul `MemRead` este 1 (iar `MemWrite` este 0), ieșirea ALU este folosită ca adresă de citire, iar data de acolo se scrie în registru (\$t1).
- Există o ambiguitate în desen: din instrucțiune, pozițiile 25-21, 20-16, 15-11 merg la cei 3 regiștri și 15-0 la extensia $16 \mapsto 32$ (deci pozițiile 15-11 se duplică).



..Instrucțiuni de transfer

Instrucțiuni de transfer (cont.) O formulă pentru $C03 : 32 + 6 \rightarrow 1$ este (RF = Register File; $SignExt$ = Sign Extended; DM = Data Memory)

$$C03 = \{ [(\wedge_0^6 \star l_{10} \star \wedge_2^5 \star l_{11}) \star l_6 \star l_{32}] \\ \cdot [l_{15} \star ({}^{16}X^1 \star l_{37}) \cdot {}^{22}X^{32}] \\ \cdot (RF \star SignExt \star l_5) \\ \cdot (l_{32} \star {}^{32}X^{35} \star l_2) \\ \cdot (ALU1 \star l_{34}) \\ \cdot (l_1 \star DM) \} \uparrow^{32}$$

Porțile de conexiune sunt:

- La intrare: instrucțiunea (32b) și 6b de control pentru (în ordine) RegWrite (1b), ALU operation (3b), MemWrite (1b), MemRead (1b).
- La ieșire: test pentru rezultat zero (1b).



Instructiuni conditionale

Instructiuni conditionale: Instrucțiunile de tip III considerate sunt una de format *format R* (i.e., *bne*) și una de format *J* (i.e., *j*).

Primul exemplu este

```
beq $t1,$t2,offset;
```

- Dacă conținutul regiștrilor $\$t1, \$t2$ este egal se trece la instrucțiunea $PC + 4 + \text{offset} \times 4$, altfel la instrucțiunea $PC + 4$.
- Notă: *offset*-ul este față de instrucțiunea următoare și este exprimat în cuvinte; deplasarea reală se obține înmulțind cu 4.
- Decizia despre următoarea instrucțiune depinde de rezultatul testului (ieșire ALU). Dacă testul este adevărat (operanzi egali), *ramificația este acceptată*, altfel *nu este acceptată*.



Instructiuni conditionale

Instructiuni conditionale (cont.)

Al doilea exemplu este

`j address;`

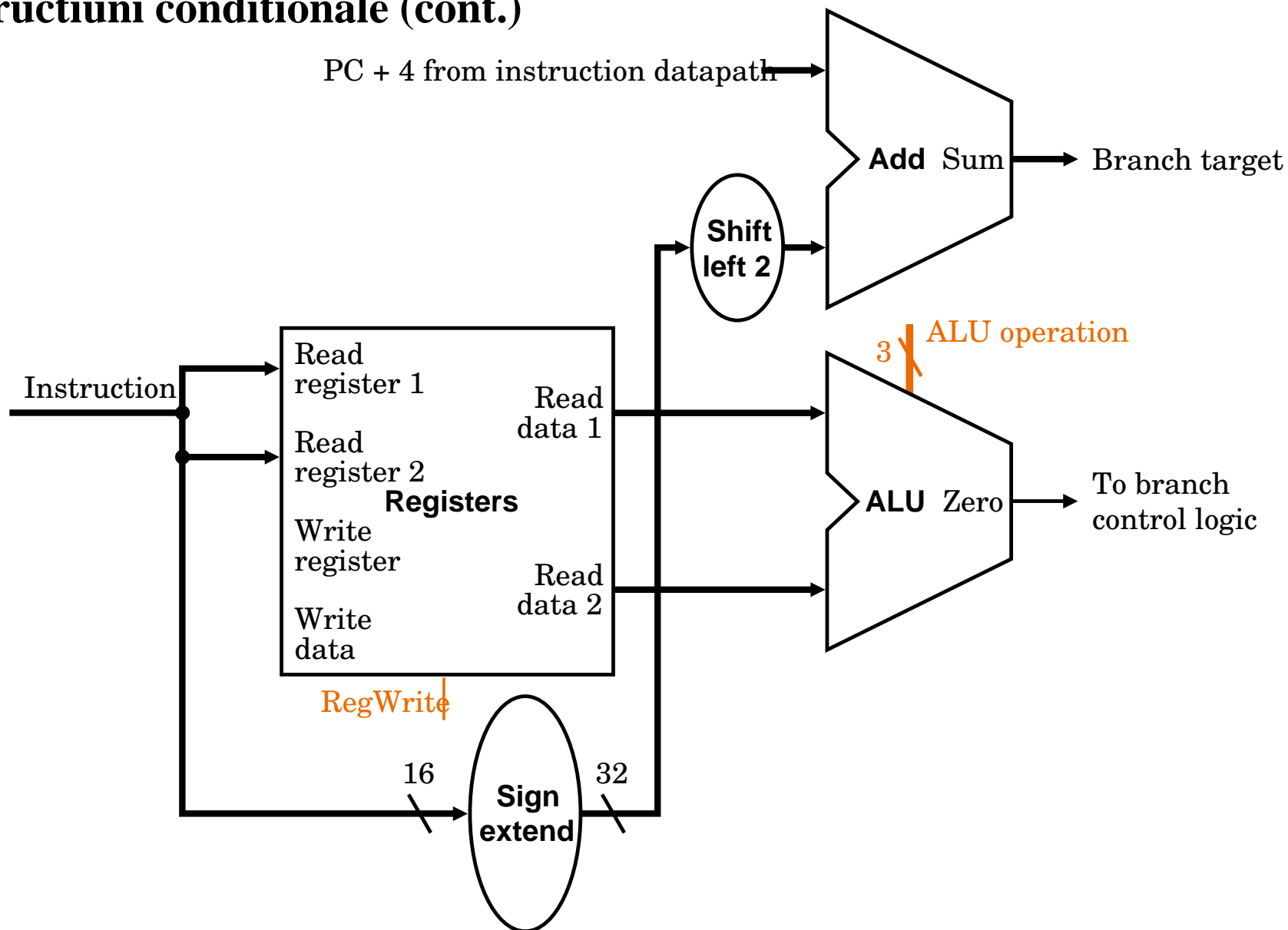
- Se face un salt necondiționat la adresa, exprimată în cuvinte, specificată de ultimii 26 biți din instrucțiune.
- Ca implementare, se înlocuiesc biții 27-0 din PC cu biții 25-0 din instrucțiune, shift-ați la stânga cu 2 biți.

Remarcă:

- In MPIS ramificațiile sunt *întârziate*, anume se execută mereu instrucțiunea următoare, decizia despre test venind mai târziu.
- Dacă testul este pozitiv, se continuă normal; altfel, se revine, se restaurează regiștri și se acceptă ramificația (saltul).

..Instrucțiuni condiționale

Instrucțiuni condiționale (cont.)



O parte din calea de date pentru instrucțiuni condiționale (format *I*).



Procesorul: Calea de date si controlul

Cuprins:

- Generalitati
- Calea de date
- *O prima implementare*
- Implementare cu cicluri multiple
- Microprogramare
- Exceptii
- Concluzii, diverse, etc.



O prima implementare

O prima implementare:

- In această primă implementare, *combinăm componentele* generate separat pentru fiecare instrucțiune de tip I, II, ori III.
- Scopul este de a folosi aceleași componente (spre exemplu, aceeași unitate ALU), dar cu mai multă *flexibilitate în conexiuni* pentru a acoperi toate cazurile.
- In implementare, vom folosi *multiplexori* (ori *selectori de date*) și un *control* adecvat pentru manipularea lor.
- Pentru început, producem o *unică cale de date* pentru instrucțiunile *lw, sw, add, sub, and, or, slt, beq*. Integrarea căii de date pentru saltul necondiționat (*j*) se va face ulterior.

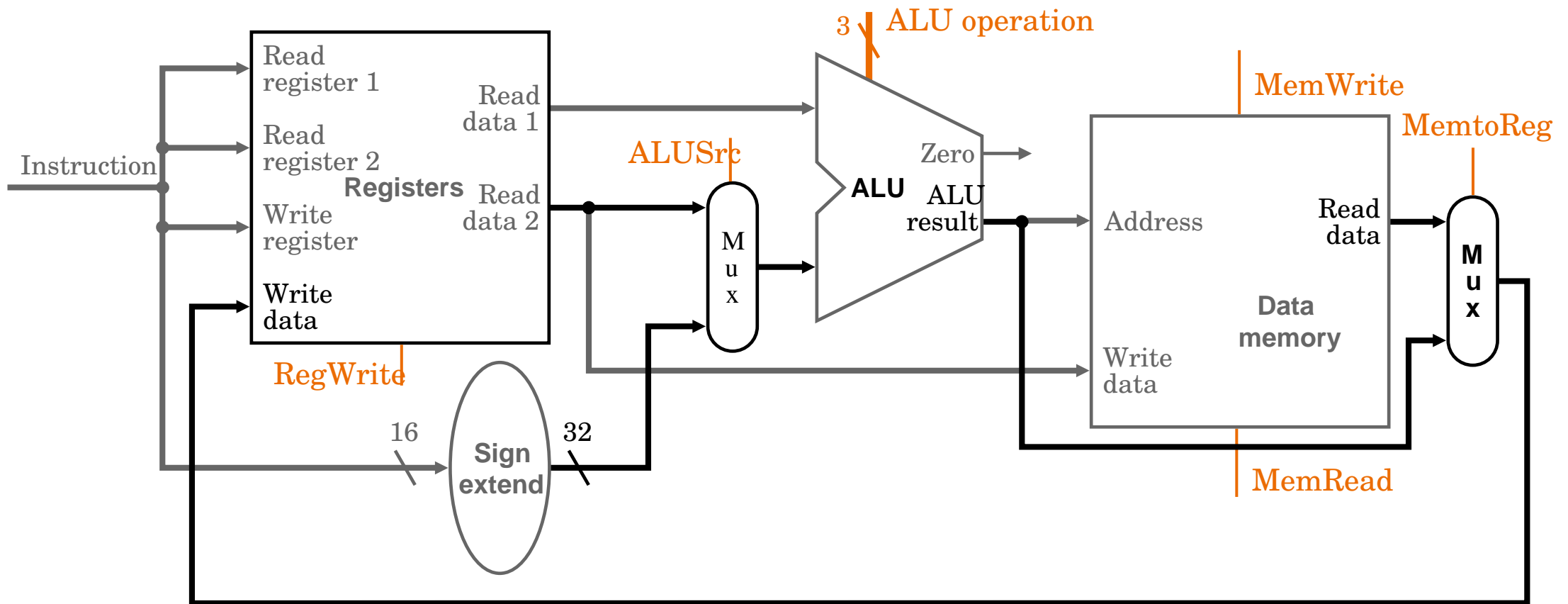


Integrarea cailor de date

Integrare: operații ALU + accesare memorie (tip I + tip II):

- Folosim un selector cu controlul **ALUSrc** care decide de unde se ia al 2-lea argument: la tip I data vine din *RF* (Register File), iar la tip II din *SignExt* (Sign Extend).
- Folosim un selector cu controlul **MentoReg** care decide cine scrie în registru: la tip I data vine din ALU, iar la tip II (load) vine din *DM* (Data Memory).
- Dacă $(ALUSrc, MentoReg) = (0, 1)$ se face o operație ALU, iar dacă $(ALUSrc, MentoReg) = (1, 0)$ se face operație de accesare memorie (load).

..Integrarea cailor de date



Combinatie: operații ALU + load / store.

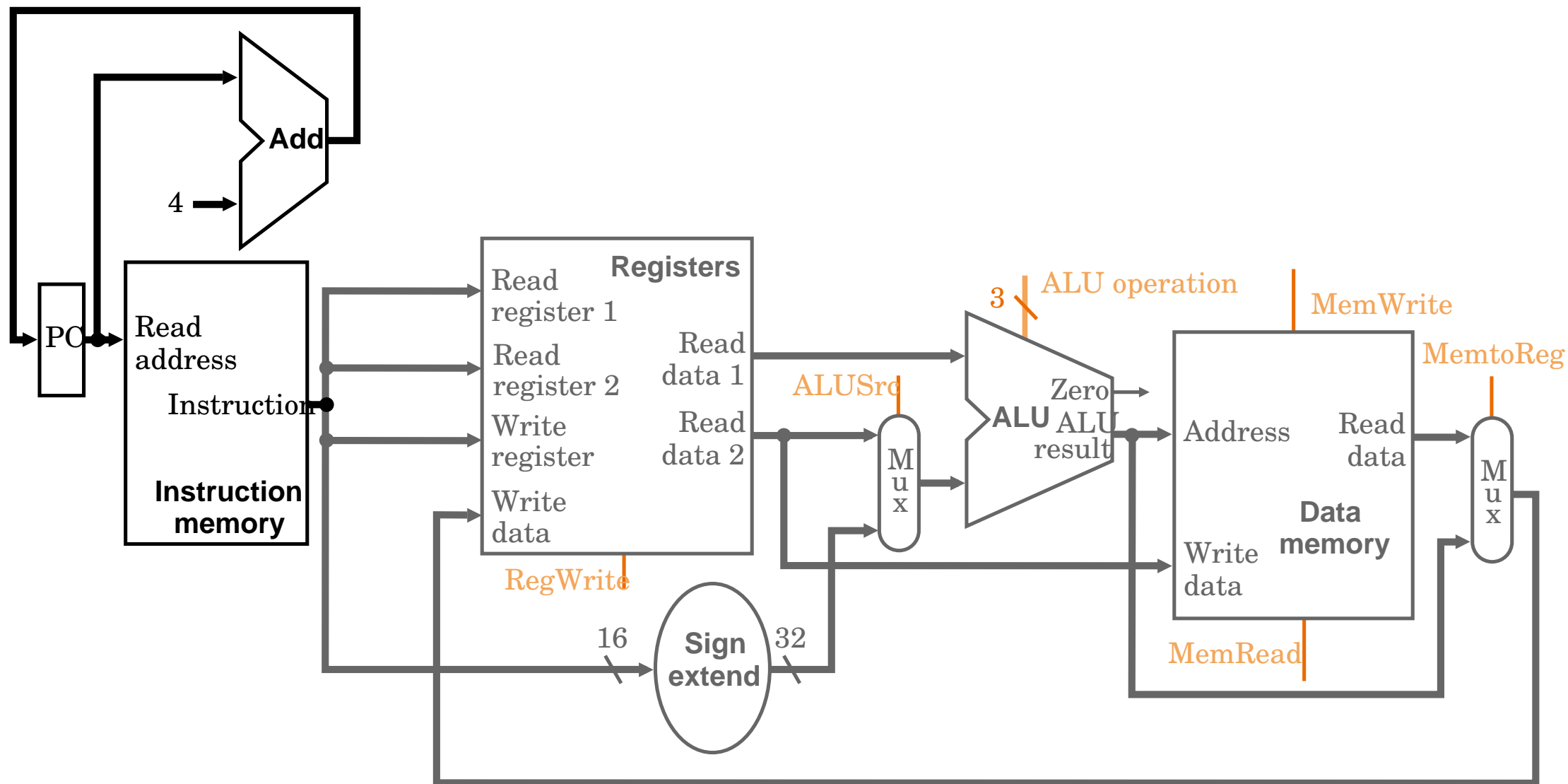


..Integrarea cailor de date

Adaugare - Extractie instructiune:

- Integrarea *extragerii instructiunii*:
 - se adaugă simplu componenta respectivă, lipită de *IM* (v. fila următoare);

..Integrarea cailor de date



Combinatie: operații ALU + load / store + *extracție instrucțiune*.

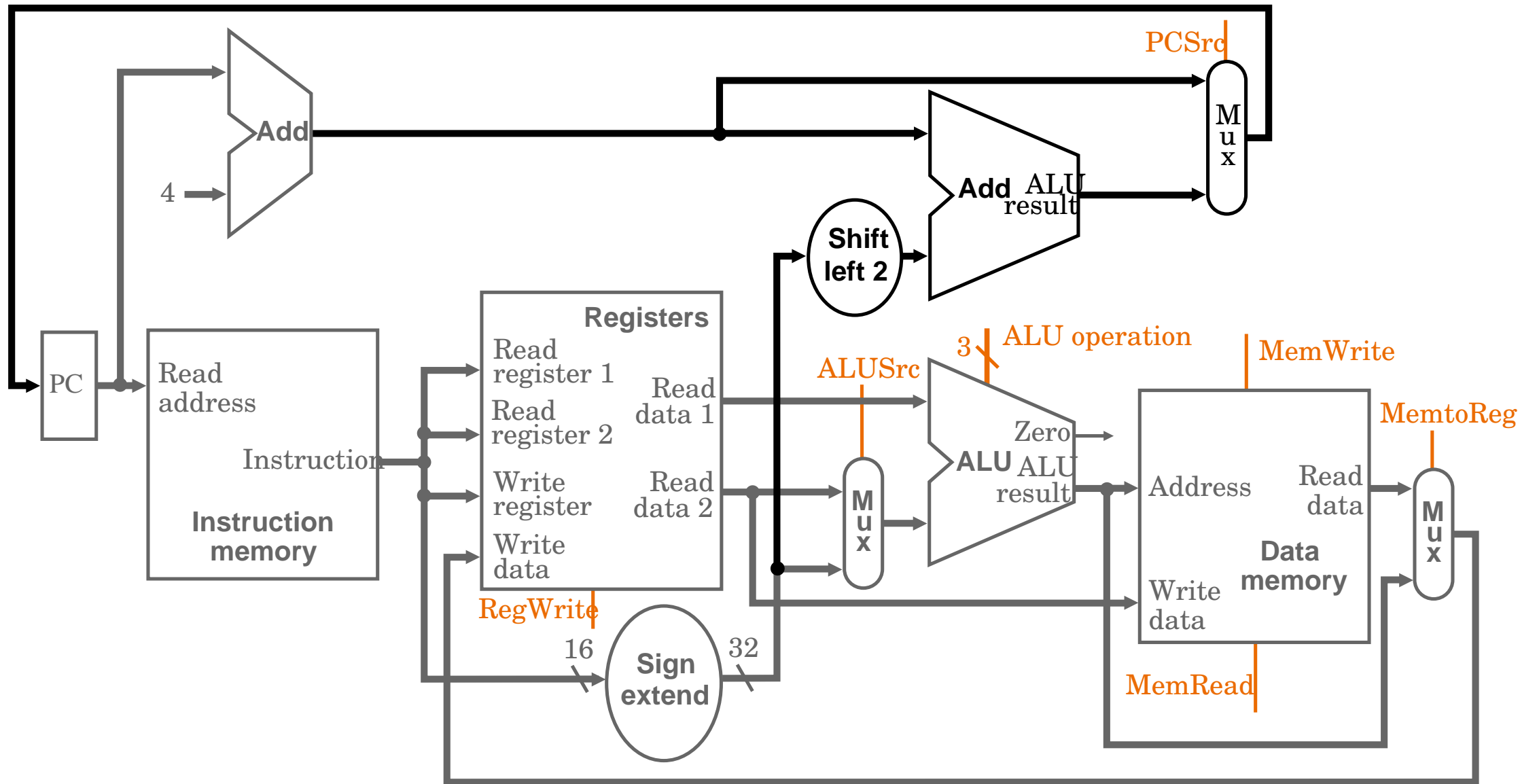


..Integrarea cailor de date

Adaugare - Ramificatii conditionale:

- Integrarea *ramificației condiționale*:
 - folosim un multiplexor cu controlul $PCSrc$ care decide de unde se ia adresa instrucțiunii de executat;
 - normal adresa este $PC + 4$, iar dacă ramificația este acceptată, adresa se ia din ieșirea componentei speciale ALU (v. fila a 2-a în continuare).

..Integrarea cailor de date



Combinatie: operații ALU + load / store + extracție instrucțiune + ramificații condiționale



Control ALU

Control ALU:

- Anterior am stabilit următorul control ALU:

ALU control input	Function
000	and
001	or
010	add
110	subtract
111	set on less than

- Generăm controlul ALU din câmpul de *funcție* al instrucțiunii și un control de 2b numit **ALUOp**.
- ALUOp indică operația executată astfel: adunare (00) pentru lw/sw, scădere (01) pentru beq, ori operația din codul de funcție 5-0 al instrucțiunii (10).

Nota: Bitii ALUOp vor fi generati in unitatea centrala de control.

..Control ALU

Control ALU (cont.)

- Tabelul indică cum depinde controlul ALU de ALUOp și biți 5-0 din instrucțiune (câmpul de funcție):

Opcode	ALUOp	Operatie	Camp functie	Actiune ALU	control ALU
LW	00	incarcare cuvant	XXXXXX	add	010
SW	00	memorare cuvant	XXXXXX	add	010
Branch equal	01	ramificatie cond.	XXXXXX	subtract	110
Tip-R	10	adunare	100000	add	010
Tip-R	10	scadere	100010	subtract	110
Tip-R	10	si	100010	and	000
Tip-R	10	sau	100100	or	001
Tip-R	10	set daca mai mic	100100	set on less than	111

..Control ALU

Control ALU (cont.)

- Din tabelul de mai sus extragem următoare *tabelă de adevăr*:

ALUOp		Camp functie						Operatie
ALUOp ₁	ALUOp ₀	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

- Tabela *favorizează optimizarea*, i.e., multe din cele $2^8 = 256$ combinații de intrări nu au importanță (X = “don’t care”).
- Din tabelă se obține o *formulă logică*, se optimizează și se produce un *circuit* pentru controlul ALU.



Unitatea principală de control

Unitatea principală de control: Formatele de instrucțiuni pe care le avem sunt:

- format R pentru operații ALU

31-26	25-21	20-16	15-11	10-6	0-5
0	rs	rt	rd	shamt	funct

- format I pentru load / store

31-26	25-21	20-16	15-0
35/43	rs	rt	address

- format I pentru salt condiționat

31-26	25-21	20-16	15-0
4	rs	rt	address



..Unitatea principală de control

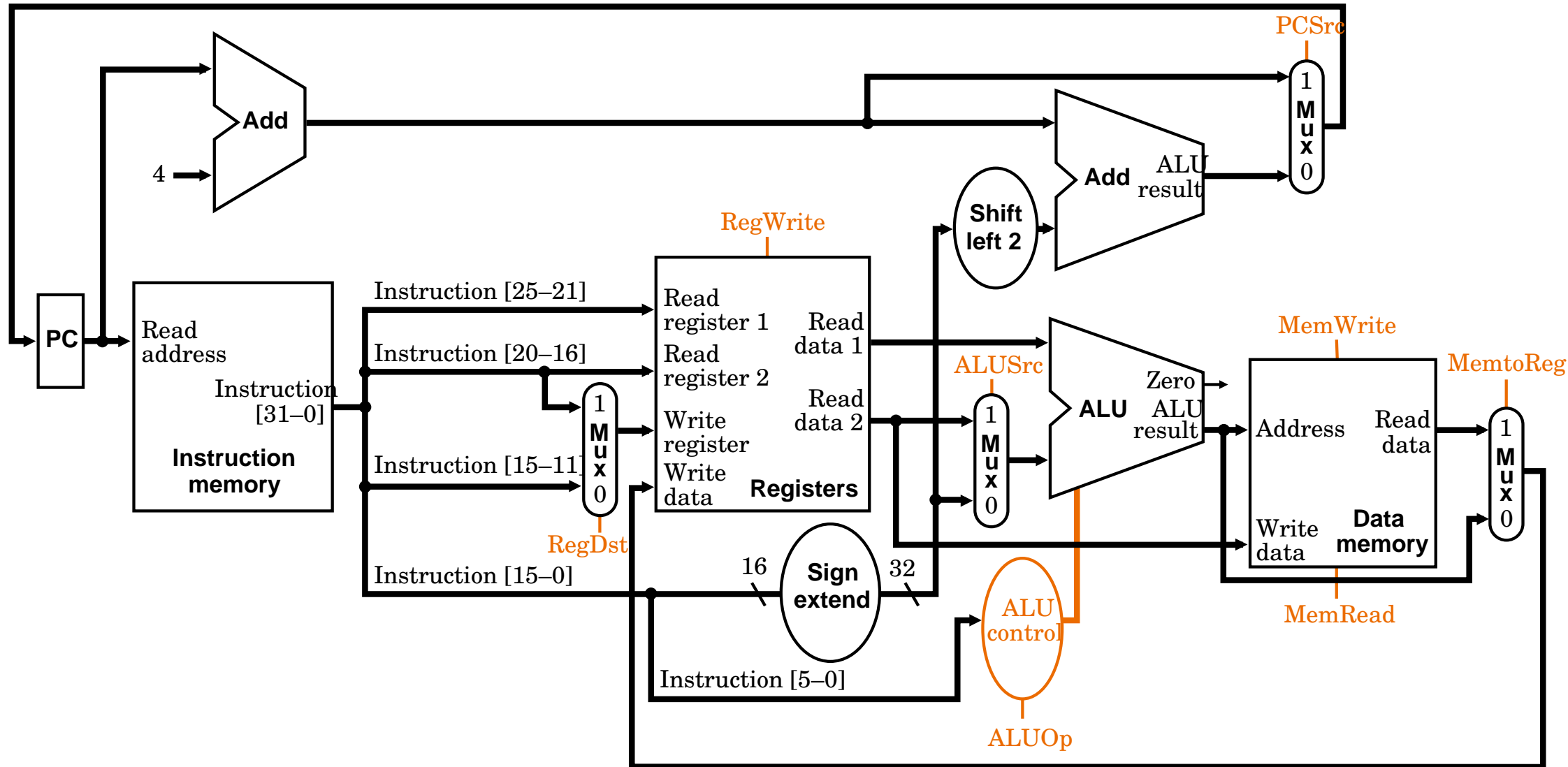
Unitatea principală de control (cont.)

Observații:

- codul de operație, notat $Op[5-0]$ este în biții 31-26;
- cei 2 regiștri de citit pentru operațiile ALU, branch și store sunt în rs și rt (zonele 25-21 și 20-16);
- la load / store registrul de bază este în rs (zona 25-21);
- la branch, load și store offset-ul este pe 16b în zona 15-0;
- registrul destinație este
 - la operațiile ALU în rd (zona 15-11);
 - la load în rt (zona 20-16);

Obținem *setul complet de semnale de control* adăugând un multiplexor cu controlul *RegDest* care decide în ce registru se scrie.

..Unitatea principala de control



Calea de date (cu multiplexori și liniile de control ALU)



..Unitatea principala de control

Unitatea principala de control (cont.) Semnalele de control folosite sunt:

RegDst: 0 - registrul de scriere se ia din câmpul rt (20-16); 1 - se ia din rd (15-11);

RegWrite: 0 - nimic; 1 - permite scrierea în registru;

ALUSrc: 0 - intrarea 2 ALU se ia din registru; 1 - se ia din “Sign extend” (offset-ul din 15-0 multiplicat cu 4);

PCSrc: 0 - noul PC este $PC + 4$; 1 - noul PC rezultă din sumatorul pentru acceptarea ramificației;

MemRead: 0 - nimic; 1 - la ieșire am data de la adresa de la intrare;

MemWrite: 0 - nimic; 1 - data de la intrare se memorează la adresa dată la intrare;

MemtoReg: 0 - valoarea de scris în registru vine din ALU; 1 - valoarea vine din memorie.



..Unitatea principală de control

Unitatea principală de control (cont.)

- Cu o excepție, toate semnalele de mai sus se deduc *complet* din corpul instrucțiunii.
- Excepția este PCSrc, pentru care este necesară și ieșirea zero de la ALU.
- Putem construi o *tabelă de adevăr* pentru aceste semnale de control, ca mai jos.
- Din tabelă, ca la ALU Control, putem construi un circuit pentru această *Unitate principală de control*.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Calea de date + controlul

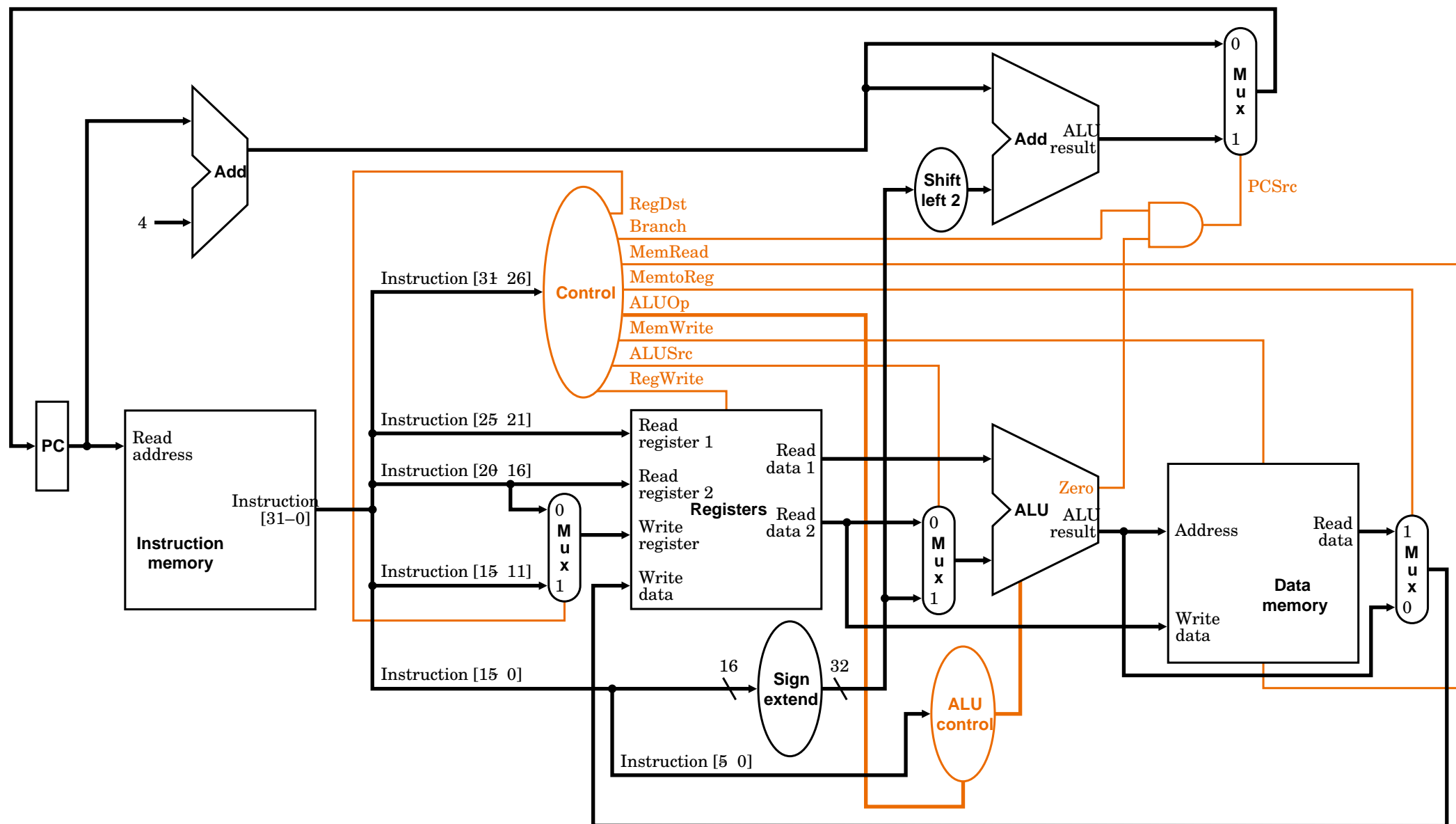


Figura conține *calea de date* integrată cu toate *semnalele de control* și *unitățile de control* (ALU Control + Control).



Procesarea unei instructiuni

Procesarea unei instructiuni: Exemplificăm procesorul obținut pe câteva instrucțiuni tipice:

- Instrucțiune de format *R* (operație ALU)

```
add $t1,$t2,$t3;
```

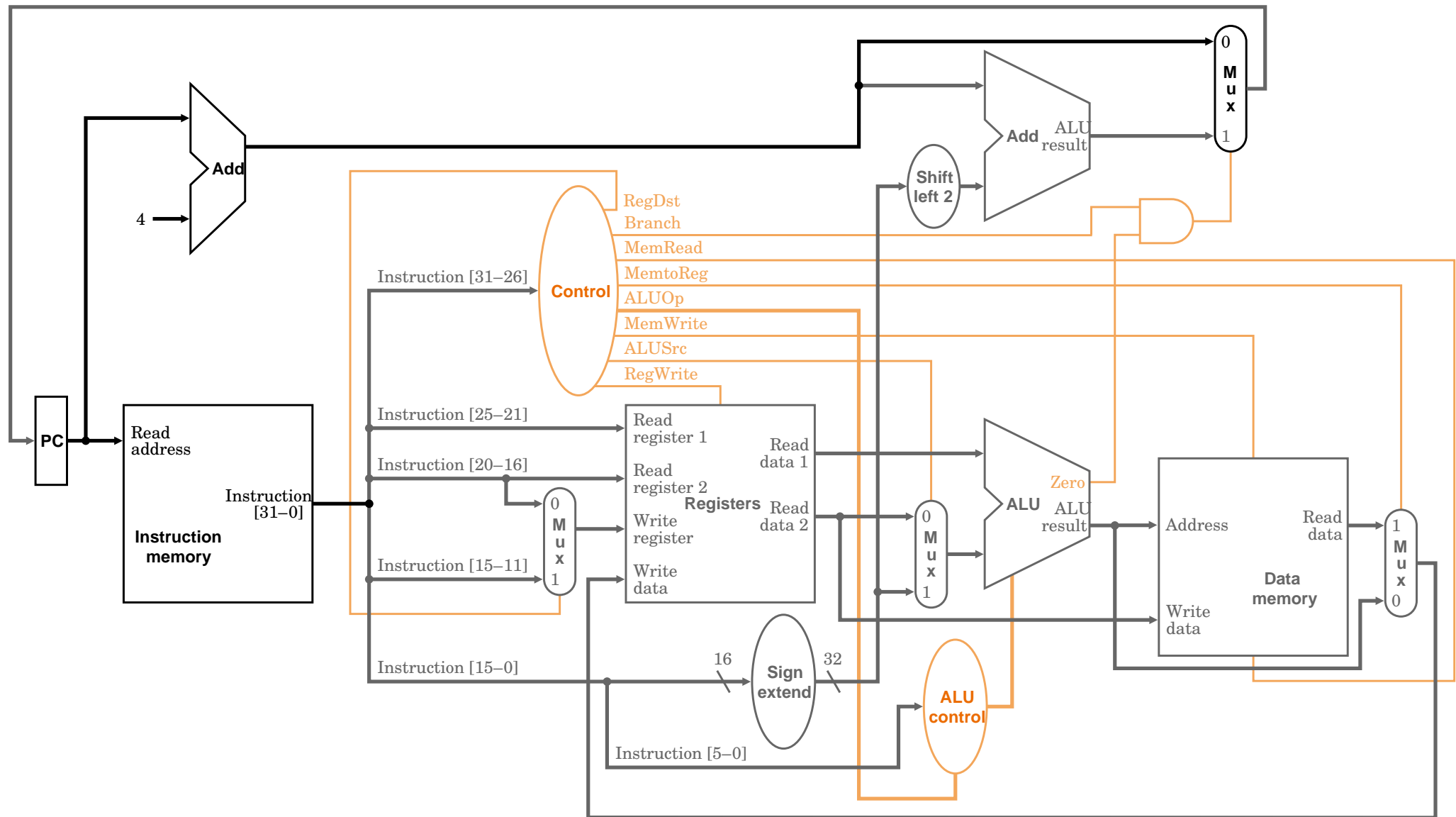
- Instrucțiune de accesare a memoriei

```
lw $t1,offset($t2);
```

- Instrucțiune condițională

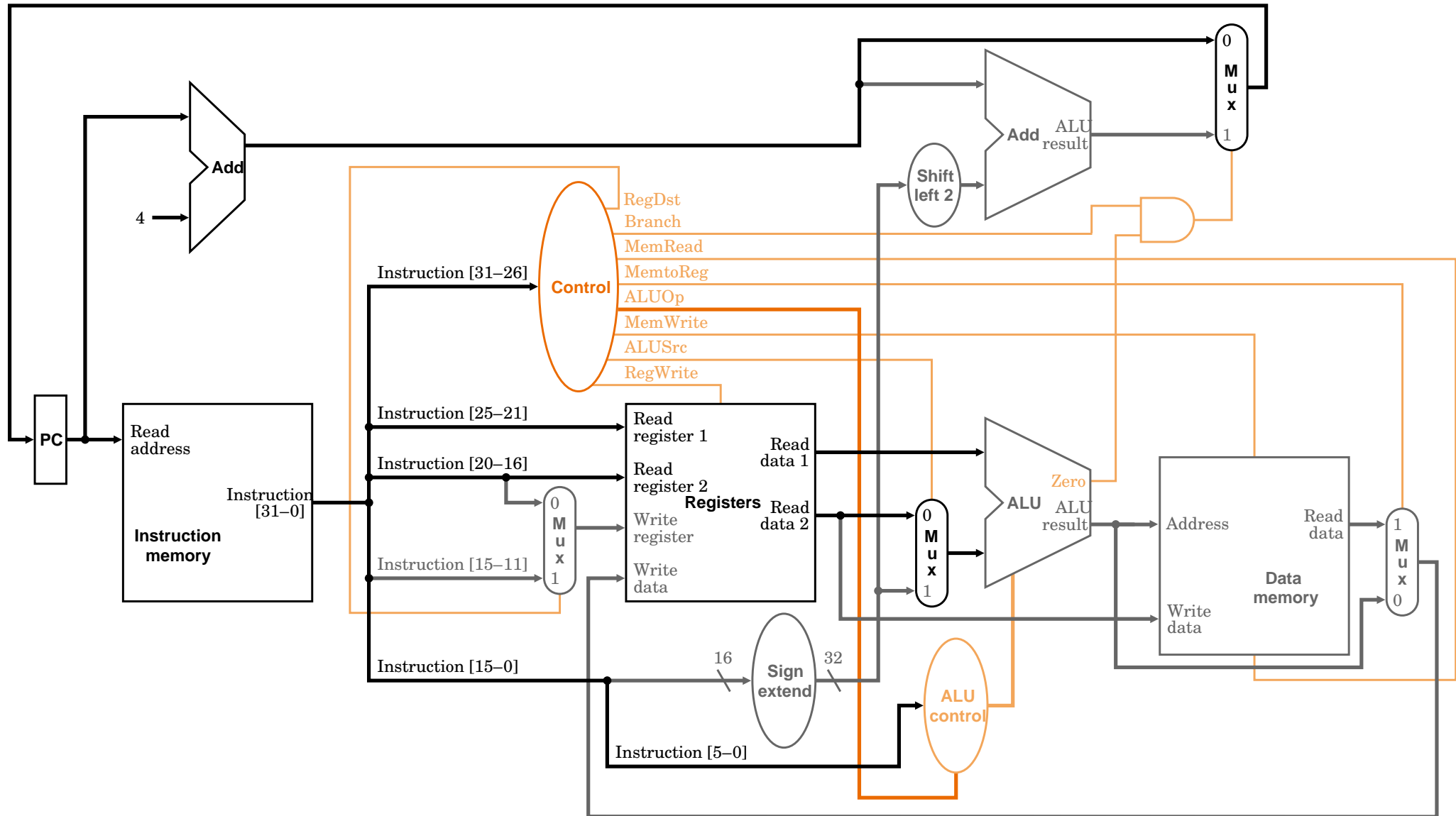
```
beq $t1,$t2,offset;
```

Procesarea unei instrucțiuni - format R



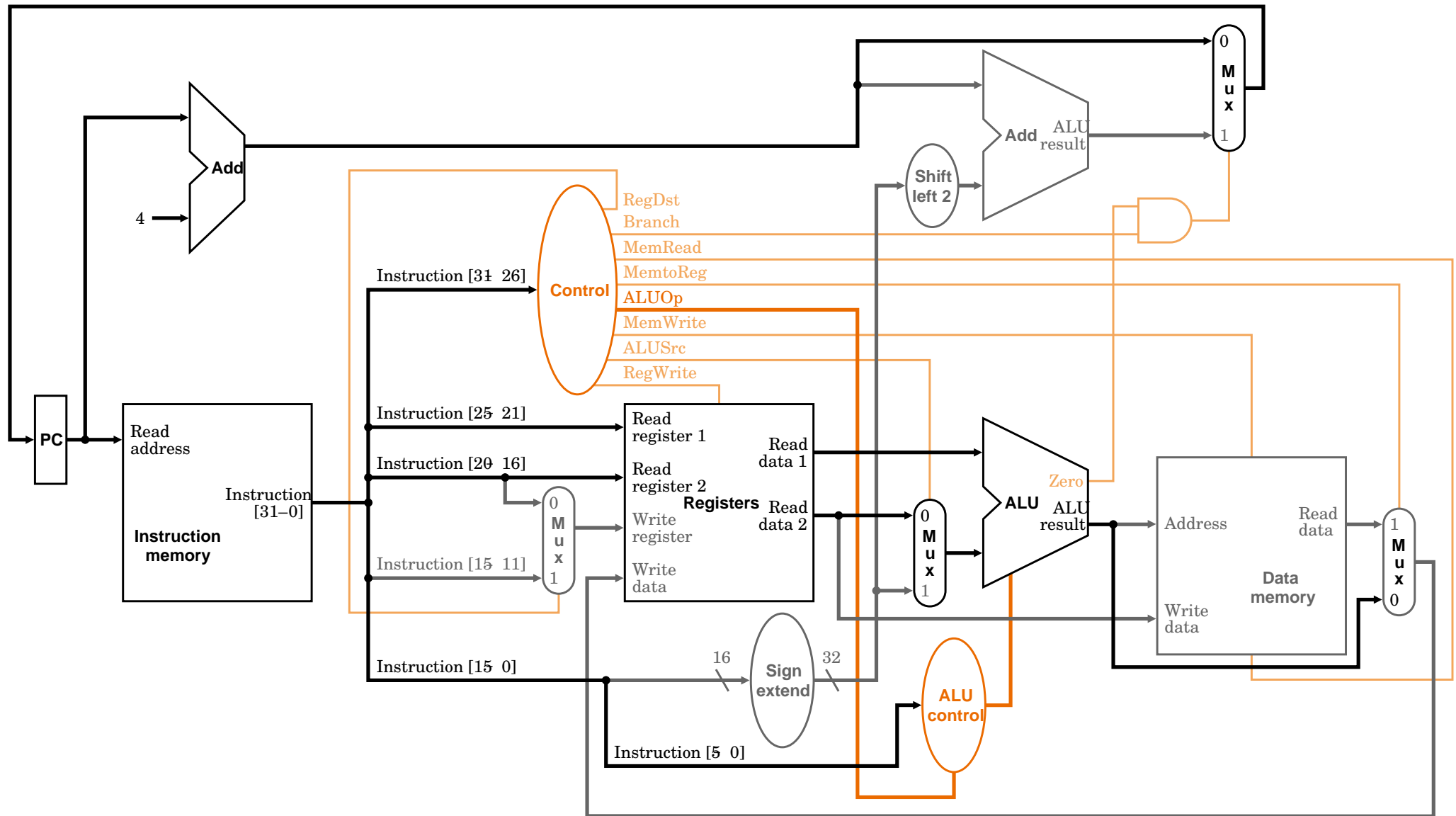
Primul pas la procesarea unei instrucțiuni de *tip R*: se *extrage* *instrucțiunea* și se *incrementează PC*.

..Procesarea unei instructiuni - format R



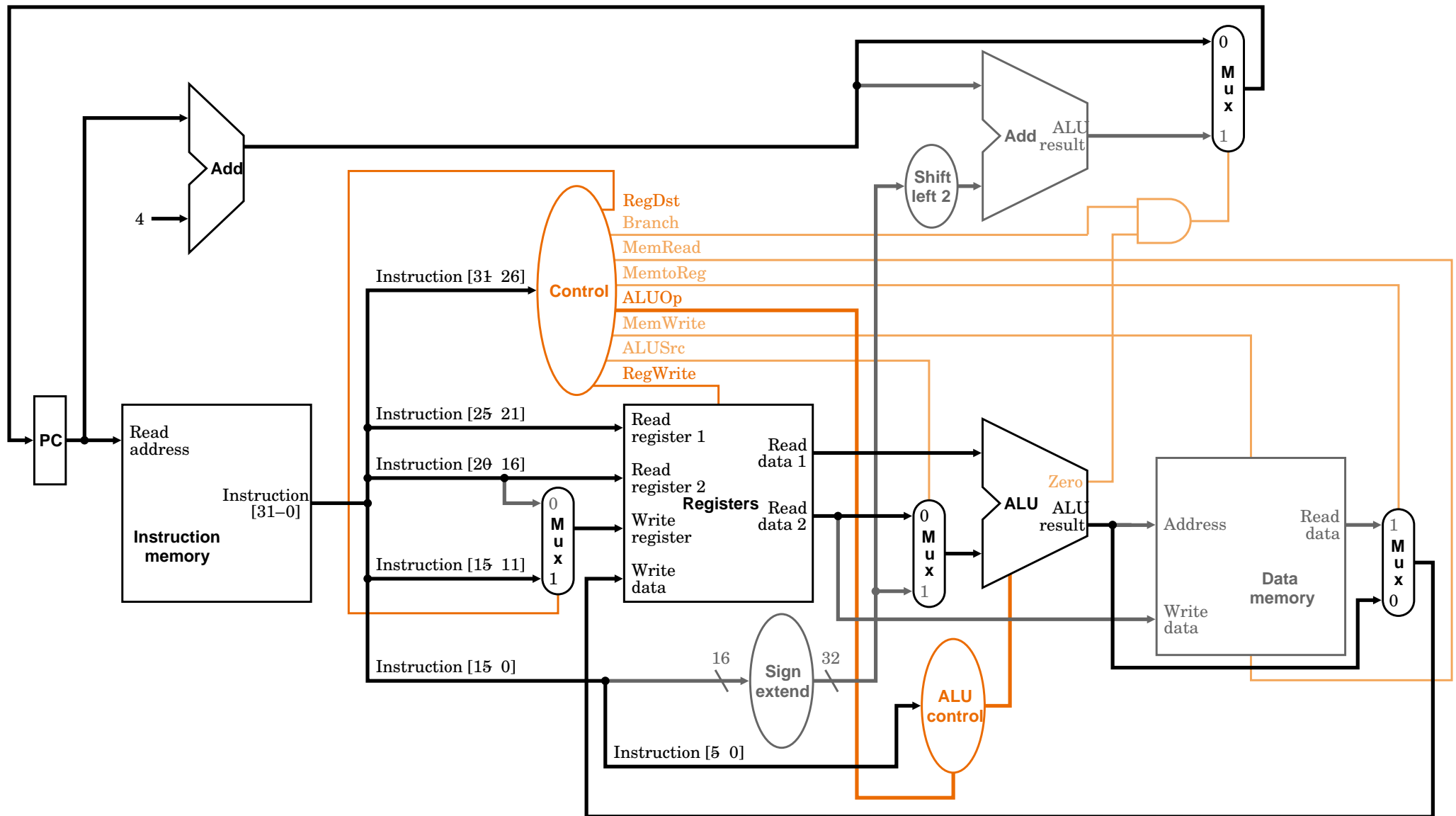
Al doilea pas la procesarea unei instructiuni de *tip R*: se *citesc 2 regiștri* sursă din RF (Register File).

..Procesarea unei instructiuni - format R



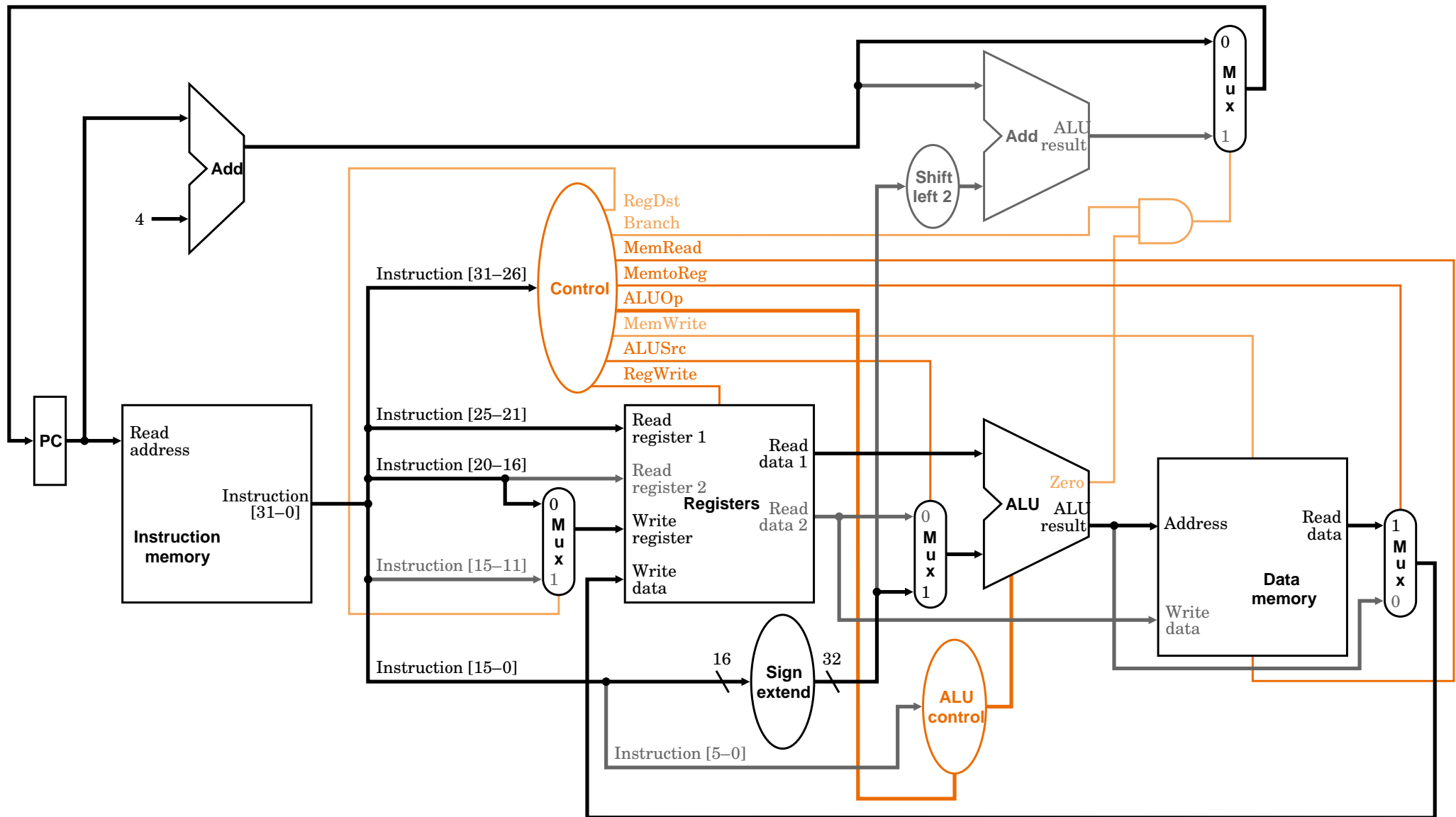
Al treilea pas la procesarea unei instructiuni de *tip R*: se *calculează* în *ALU* folosind datele din regiștri sursă.

..Procesarea unei instructiuni - format R



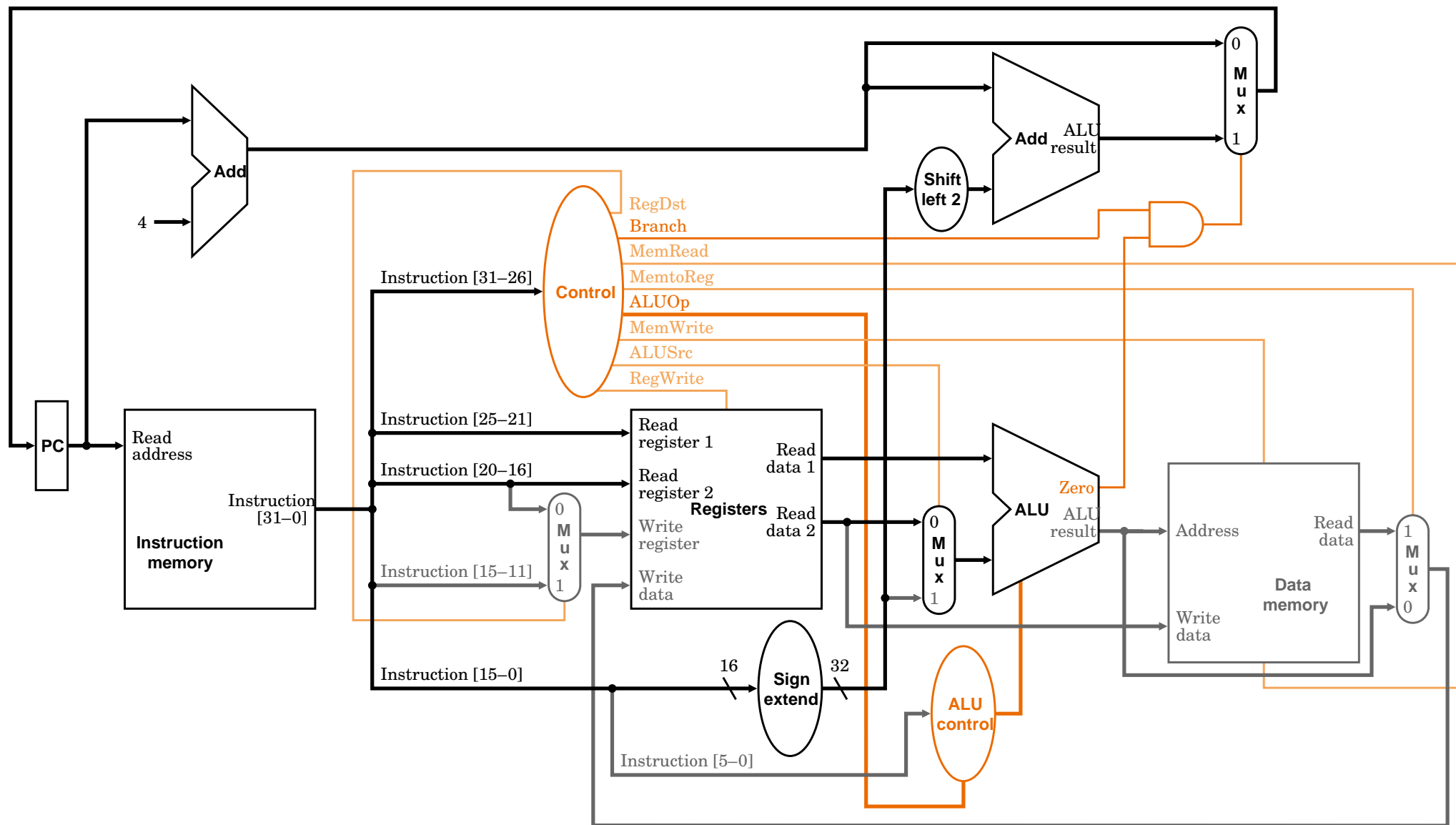
Pasul final la procesarea unei instructiuni de *tip R*: se *scrie în registru* rezultatul obținut în ALU.

Procesarea unei instructiuni - LOAD



Procesarea unei instructiuni *load*. Pentru *store* este similar, dar se scrie în memorie (nu se citește) o data ce vine din registru.

Procesarea unei instructiuni - BNE



Procesarea unei instructiuni *condiționale*. De notat că ieșirea zero din ALU se folosește spre a decide noua instrucțiune.



Prima implementare de procesor

Prima implementare de procesor: Adăugăm saltul necondiționat, j (jump):

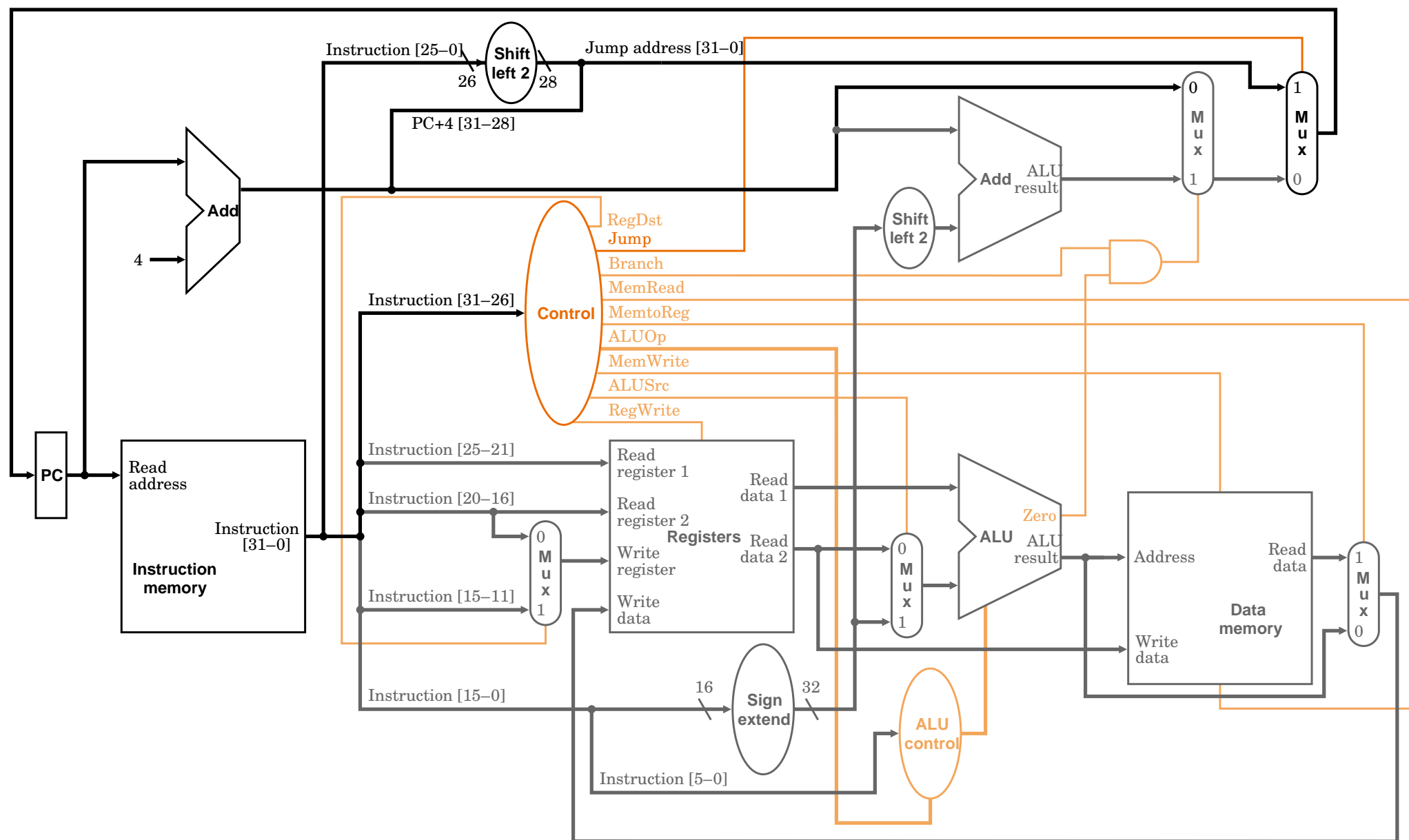
- formatul pentru jump este formatul J

31-26	25-0
2	address

- adresa este în cuvinte, deci se multiplică cu 4 (shift la stânga cu 2 biți); cei 28 biți rezultați se plasează în pozițiile 27-0 ale rezultatului;
- pozițiile din față 31-28 ale rezultatului se copiază din $PC + 4$.

Adăugând componenta pentru jump, obținem *varianta finală* pentru *prima implementare de procesor*.

..Prima implementare de procesor



Calea de date și controlul final pentru subsetul nostru: inserăm și procesarea saltului necondiționat *jump*.



Procesorul: Calea de date si controlul

Cuprins:

- Generalitati
- Calea de date
- O prima implementare
- *Implementare cu cicluri multiple*
- *Microprogramare*
- *Exceptii*
- *Concluzii, diverse, etc.*