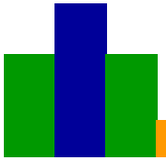


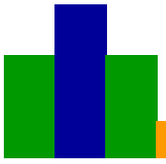
Lecţia 7:

Procesorul:
Calea de date si controlul - I

G Stefănescu — Universitatea Bucureşti

Arhitectura sistemelor de calcul, Sem.1
Octombrie 2016—Februarie 2017

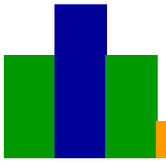
După: D. Patterson and J. Hennessy, Computer Organisation and Design



Procesorul: Calea de date si controlul

Cuprins:

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

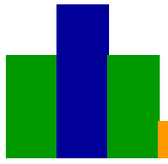


Generalitati

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

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

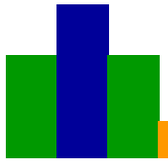
Nota: Nu contine operatii de inmultire, impartire, operatii in virgula mobila, etc., dar principiile de constructie ale procesorului sunt similare.



..Generalitati

Pasi de procesare:

- Primii paşi sunt comuni pentru toate instrucţiunile de mai sus:
  - Se trimite contorul de instrucţiuni PC în memorie unde este programul şi se *extrage instrucţiunea* de executat;
  - Se *citesc* unul ori doi *regiştri* asupra cărora operează instrucţiunea;
- Ce urmează depinde de tipul instrucţiunii, dar, din fericire, există anumite asemănări care uniformizează procesarea:
  - Toate operaţiile *folosesc ALU* pentru calcule specifice
    - tip I: operaţii aritmetice ori logice;
    - tip II: găsirea adresei de memorie;
    - tip III: operaţii pentru decizia de la test.



..Generalitati

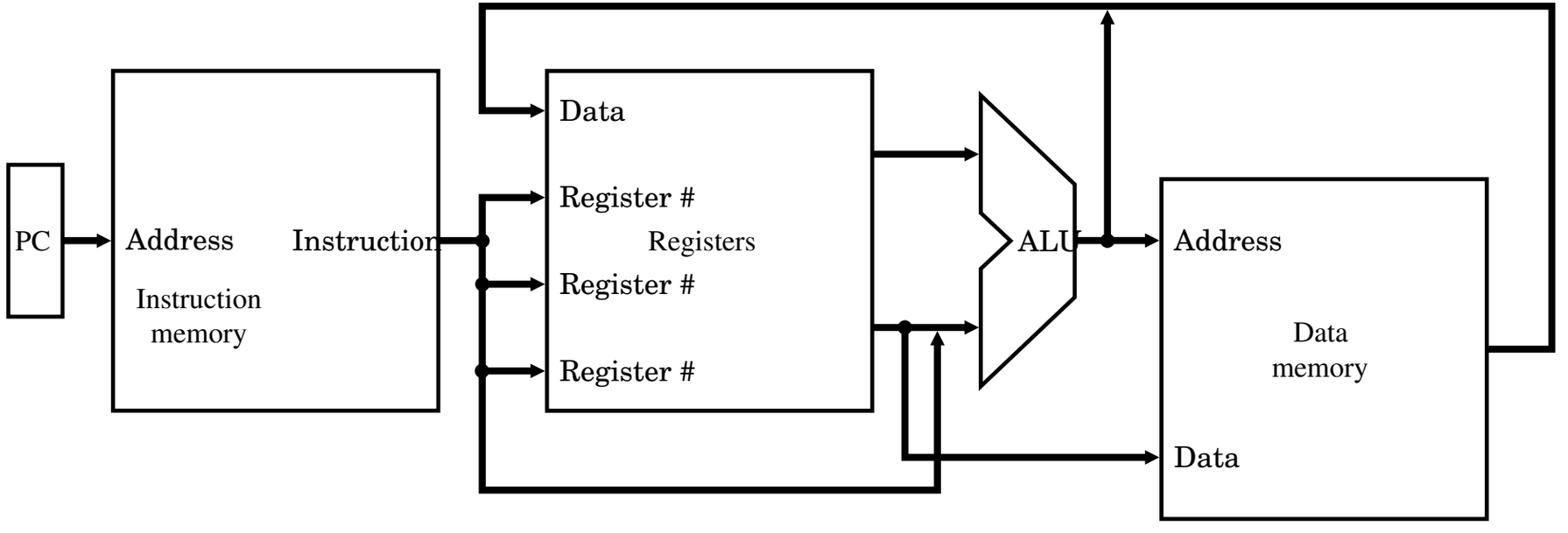
Pasi de procesare (cont.)

- Paşii de finalizare a instrucţiunilor sunt mai diferiţi la cele trei tipuri, necesitând mai mult efort de uniformizare (anume, un control mai complex):
  - Paşi de *finalizare*
    - tip I: se scrie rezultatul în registru;
    - tip II: se accesează memoria pentru citit/scris;
    - tip III: se decide care este noua instrucţiune.

..Generalitati

Schema de ansamblu:

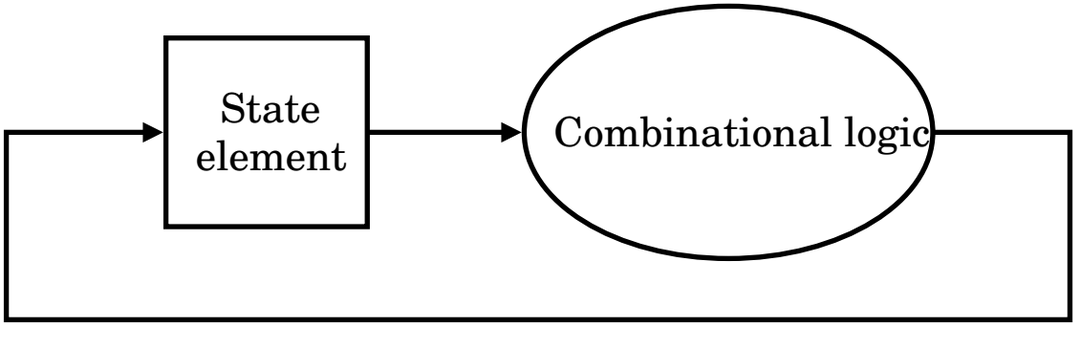
- Figura conţine schema de ansamblu pentru procesor, cu principalele sale *componente* (partea pentru instrucţiunile de tip III este incomplet prezentată).

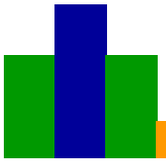


..Generalitati

Tipul circuitelor:

- Instrucţiunile din ALU sunt pur *combinaţionale*.
- Cum procesorul se foloseşte *cu repetiţie* (avem de executat mai multe instrucţiuni), avem nevoie de un *ceas* şi de *circuite cu memorie*.
- Modelul teoretic este de *maşină cu stări finite*.

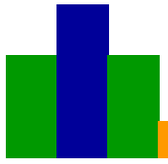




Procesorul: Calea de date si controlul

Cuprins:

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



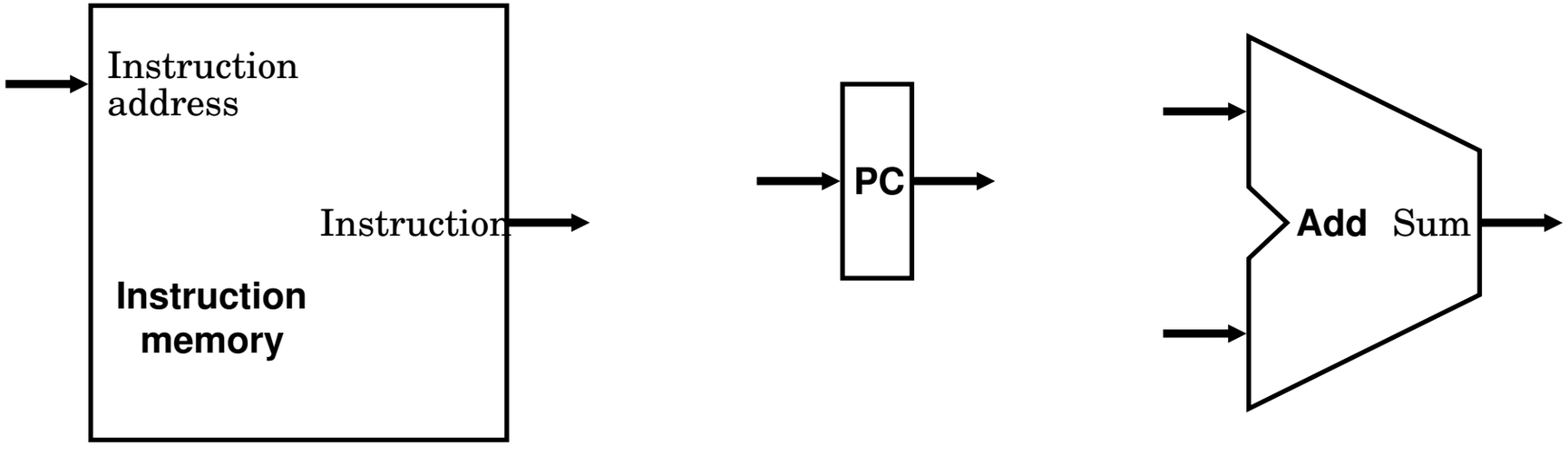
Calea de date

Calea de date: Vom exemplifica pe rând:

- cum se extrage o instrucţiune;
- cum se operează cu regiştri la instrucţiuni aritmetice şi logice;
- cum se tratează operaţiile de accesare a memoriei (load / store);
- cum procesăm salturile condiţionate (branch) ori pe cele necondiţionate (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:



a. Instruction memory b. Program counter c. Adder

Componente pentru a accesa o instrucţiune şi a calcula noua adresă:

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

..Extragerea unei instructiuni

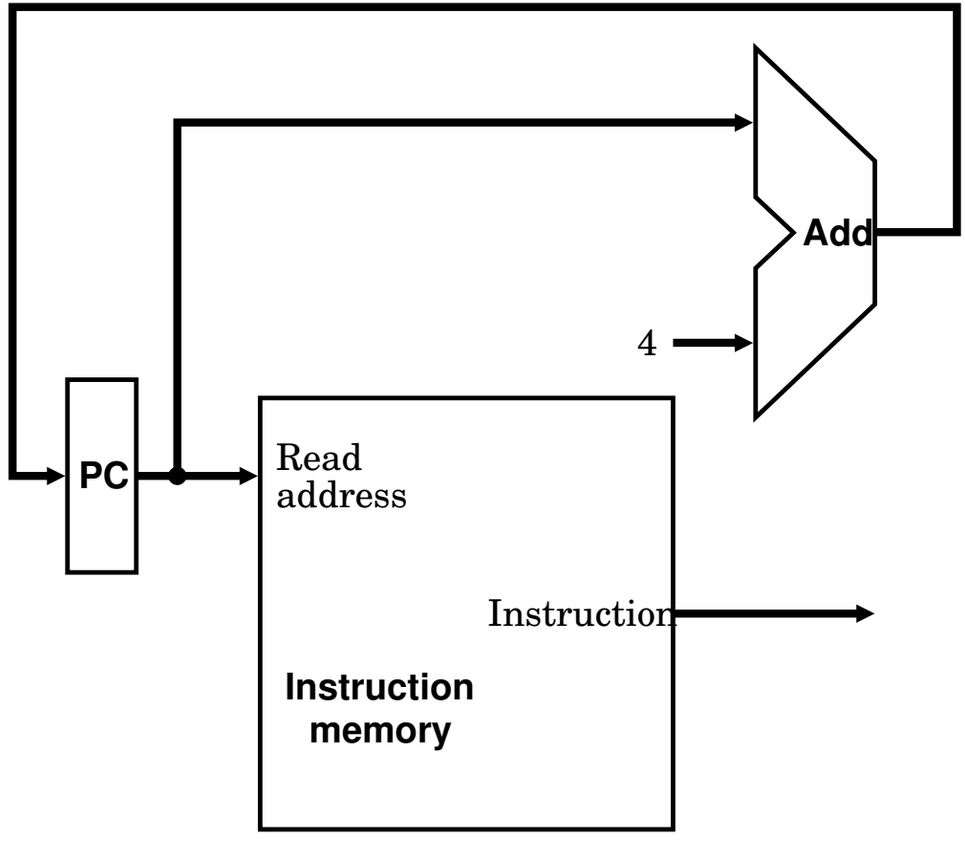
Extragerea unei instructiuni (cont.)

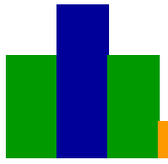
O parte din calea de date pentru extragerea instrucţiunii:

–instrucţiunile sunt pe 32 biţi, deci 4 octeţi;

–adresa instrucţiunii următoare (fără salt) se obţine adunând 4;

–componenta PC memorează valoarea pentru ciclul următor





..Extragerea unei instructiuni

Extragerea unei instructiuni (cont.)

- Fie

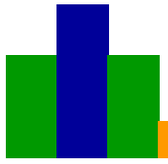
  $IM : 32 \to 32$ – memoria cu instrucţiuni;
  $ALU\_SUM : 32 + 32 \to 32$ – un sumator;
  $PC : 32 \to 32$ – memorie pentru contorul de instrucţiuni, şi
  $FOUR : 0 \to 32$ – o componentă ce produce la ieşire numărul 4.

- Componenta de mai sus este

  $$C01 = [PC \cdot \wedge_2^{32} \cdot (IM \star (FOUR \star \mathsf{I}_{32}) \cdot ALU\_SUM)] \uparrow^{32} : 0 \to 32$$

  unde avem

  – La intrare: nimic;
  – La ieşire: o poartă de 32b pentru instrucţiunea rezultată.



Instructiuni aritmetice si logice

Instructiuni aritmetice si logice:

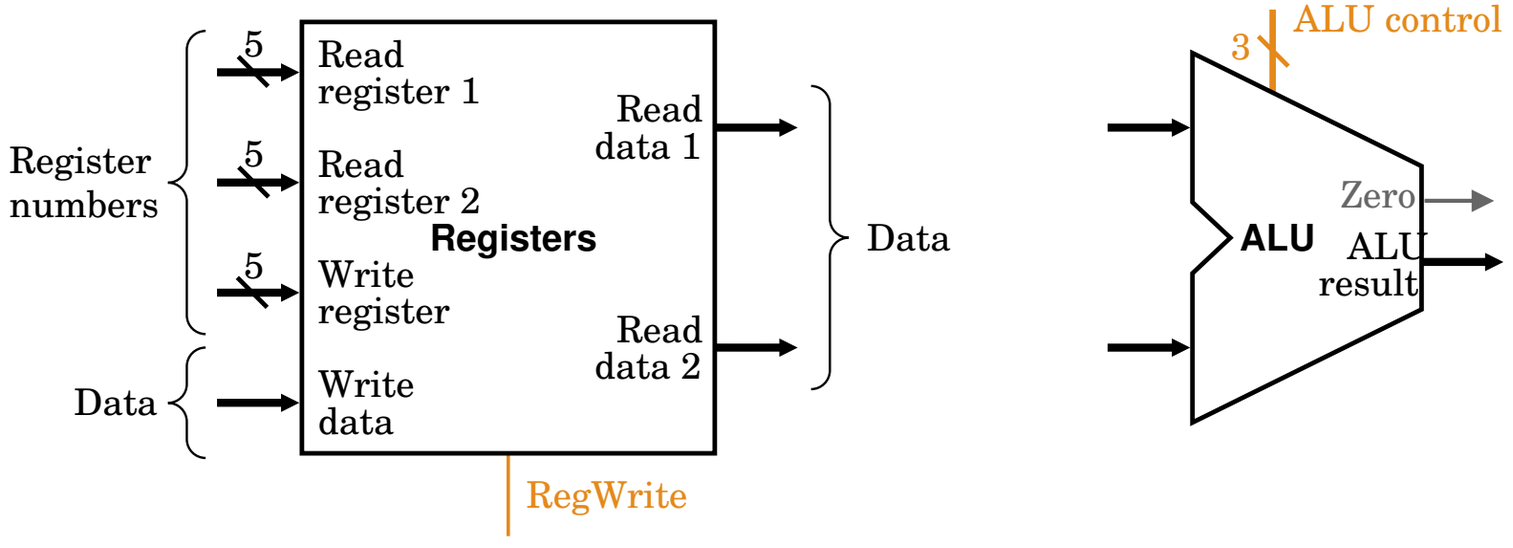
- Instrucţiunile de tip I considerate sunt în *format R*, i.e., folosesc *2 regiştri* pentru argumente şi *1 registru* pentru rezultat.
- Exemplu tipic este

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

  Se citesc regiştri `$t1,$t2` şi rezultatul se pune în `$t3`.
- Folosim o memorie cu regiştri "*register file*", care are 32 de regiştri cu 2 porţi de citire şi una de scriere (toate în paralel).
- Pentru *citirea* unui registru se dă *numărul registrului*.
- Pentru *scrierea* unui registru se dă *numărul registrului* şi *data* de scris.

..Instructiuni aritmetice si logice

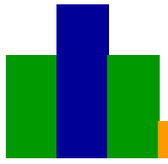
Instructiuni aritmetice si logice (cont.)



a. Registers b. ALU

Componente necesare pentru a opera pe regiştri:

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



..Instructiuni aritmetice si logice

Instructiuni aritmetice si logice (cont.)

- Memoria de regiştri M permite 2 citiri şi o scriere simultane; mai exact tipul este

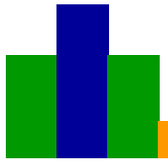
$$M : 5+5+5+32+1 \rightarrow 32+32$$

unde, la intrare avem

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

iar la ieşire avem

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



..Instructiuni aritmetice si logice

Instructiuni aritmetice si logice (cont.)

- Unitarea aritmetică şi logică $ALU1$ se obţine din cea standard ALU, neglijând ieşirile de overflow şi carryout, anume

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

deci $ALU1$ are tipul

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

unde, la intrare avem

  - două porţi de 32b pentru 2 argumente;
  - 3b de control `ALU operation` indicând operaţia;

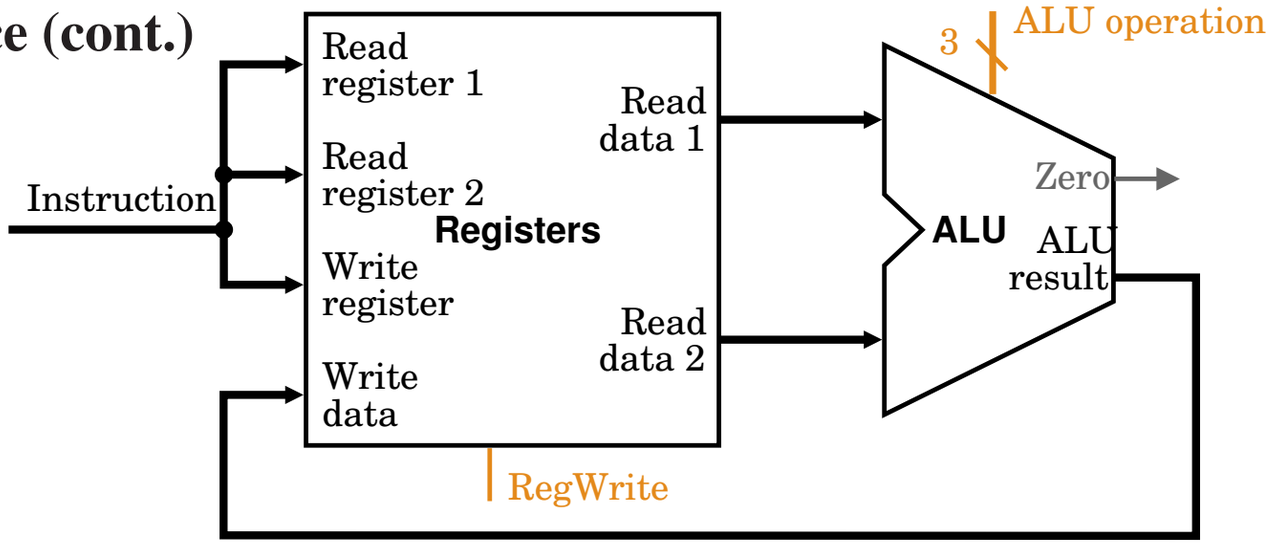
iar la ieşire avem

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

..Instructiuni aritmetice si logice

Instructiuni aritmetice si logice (cont.)

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

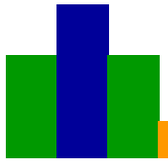


Componenta rezultată $C02 = [(\mathsf{I}_{15} \star {}^{4}\mathsf{X}^{32}) \cdot (RF \star \mathsf{I}_3) \cdot ALU\,1] \uparrow^{32}$ are tipul

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

unde avem

- la intrare: 3 porţi de 5b pentru 3 regiştri; 1b control scriere; 3b selecţie operaţie;
- la ieşire: 1b, dacă rezultatul este zero.



Instructiuni de transfer

Instructiuni de transfer:

- Instrucţiunile de tip II considerate sunt în *format I*, i.e., folosesc *1 registru* pentru dată şi *1 registru* + *offset* pentru adresă (ultimul aflat pe 16b din instrucţiune, poziţiile 15-0).
- Primul exemplu este

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

  Se calculează adresa de memorie `$t2+const` şi acolo se scrie data din `$t1`.

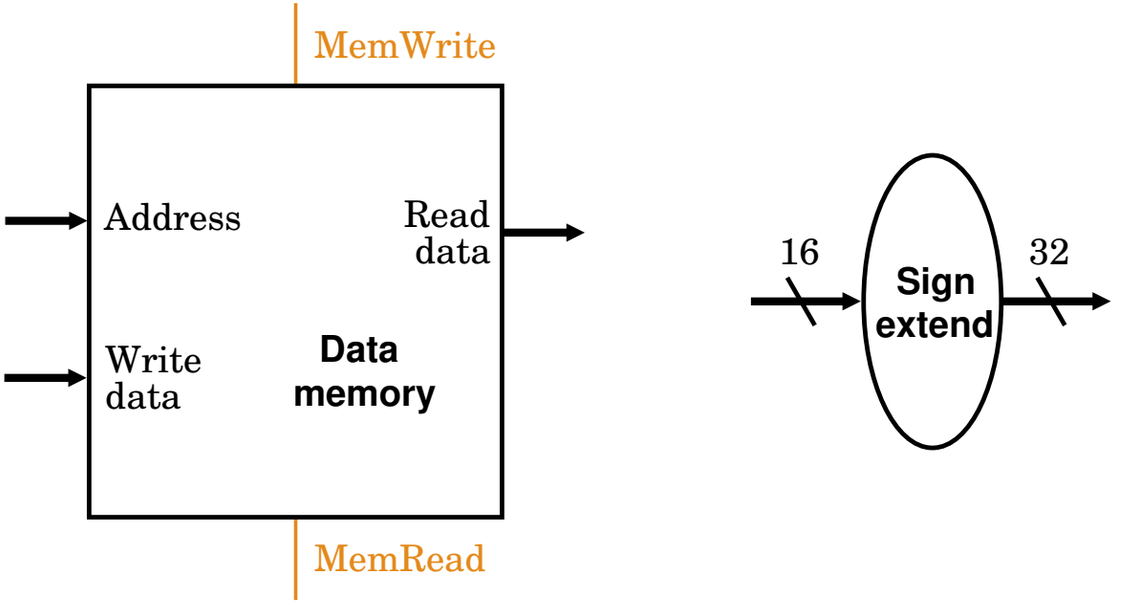
- Al doilea este

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

  Se calculează adresa `$t2+const` şi data de acolo se încarcă în `$t1`.

..Instructiuni de transfer

Instructiuni de transfer (cont.)



a. Data memory unit

b. Sign-extension unit

Componente pentru instrucţini de transfer suplimentare:

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

..Instructiuni de transfer

Instructiuni de transfer (cont.)

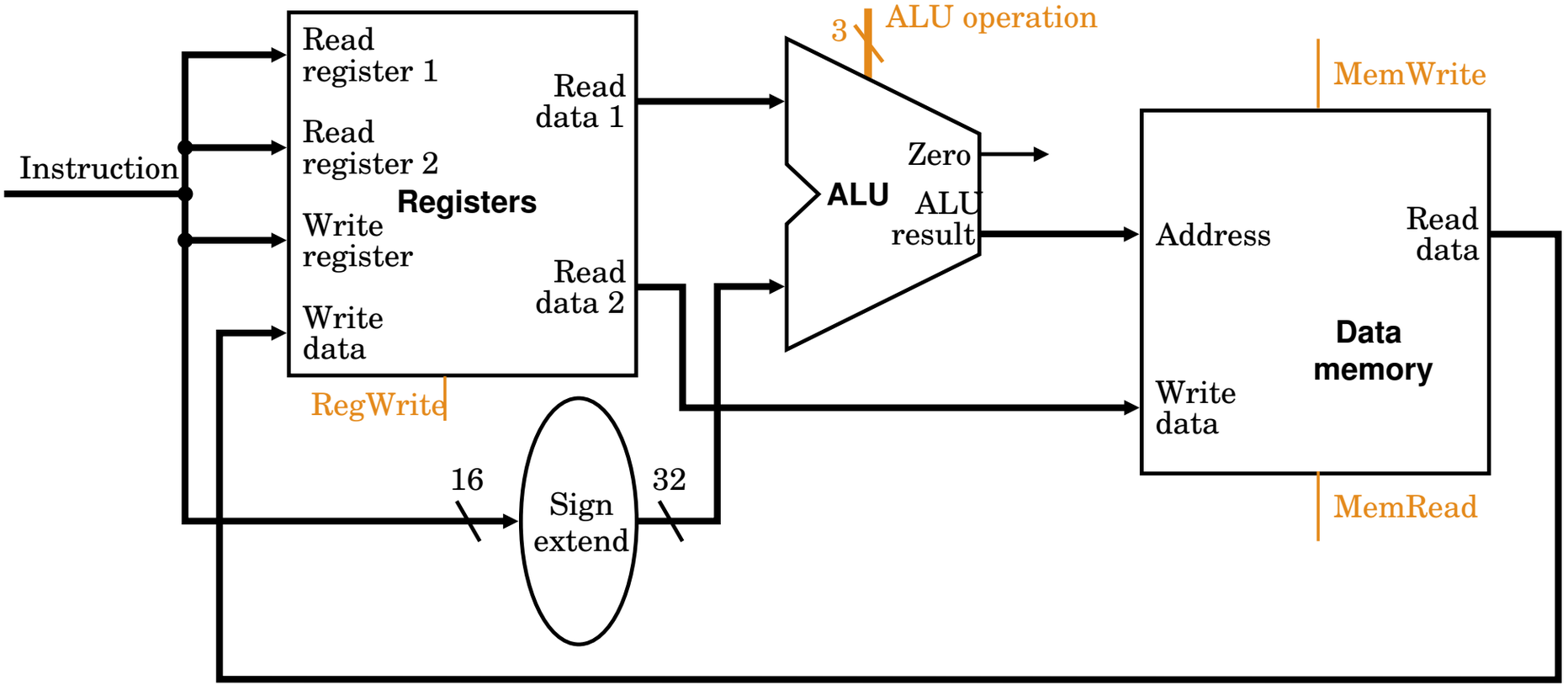
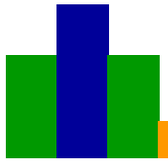


Figura conţine o parte $C03 : 32 + 6 \rightarrow 1$ din calea de date pentru instructiuni de transfer: intrări = instrucţiunea (32b) + control (6b); ieşiri = testul la zero (1b).

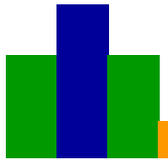


..Instructiuni de transfer

Instructiuni de transfer (cont.)

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

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



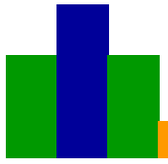
..Instructiuni de transfer

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

$$
\begin{aligned}
C03 = \{ & [(\wedge_0^6 \star \mathsf{I}_{10} \star \wedge_2^5 \star \mathsf{I}_{11}) \star \mathsf{I}_6 \star \mathsf{I}_{32}] \\
& \cdot [\mathsf{I}_{15} \star ({}^{16}\mathsf{X}^1 \star \mathsf{I}_{37}) \cdot {}^{22}\mathsf{X}^{32}] \\
& \cdot (RF \star SignExt \star \mathsf{I}_5) \\
& \cdot (\mathsf{I}_{32} \star {}^{32}\mathsf{X}^{35} \star \mathsf{I}_2) \\
& \cdot (ALU1 \star \mathsf{I}_{34}) \\
& \cdot (\mathsf{I}_1 \star DM) \} \uparrow^{32}
\end{aligned}
$$

Porţile de conexiune sunt:

- La intrare: instrucţiunea (32b) şi 6b de control pentru (în ordine) `RegWrite` (1b), `ALU operation` (3b), `MemWrite` (1b), `MemRead` (1b).
- La ieşire: test pentru rezultat zero (1b).



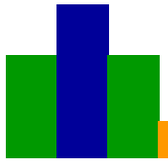
Instructiuni conditionale

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

Primul exemplu este

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

- Dacă conţinutul regiştrilor `$t1,$t2` este egal se trece la instrucţiunea $\texttt{PC} + 4 + \texttt{offset} \times 4$, altfel la instrucţiunea $\texttt{PC} + 4$.
- Notă: `offset`-ul este faţă de instrucţiunea următoare şi este exprimat în cuvinte; deplasarea reală se obţine înmulţind cu 4.
- Decizia despre următoarea instrucţiune depinde de rezultatul testului (ieşire ALU). Dacă testul este adevărat (operanzi egali), *ramificaţia este acceptată*, altfel *nu este acceptată*.



Instructiuni conditionale

Instructiuni conditionale (cont.)

Al doilea exemplu este

```
j address;
```

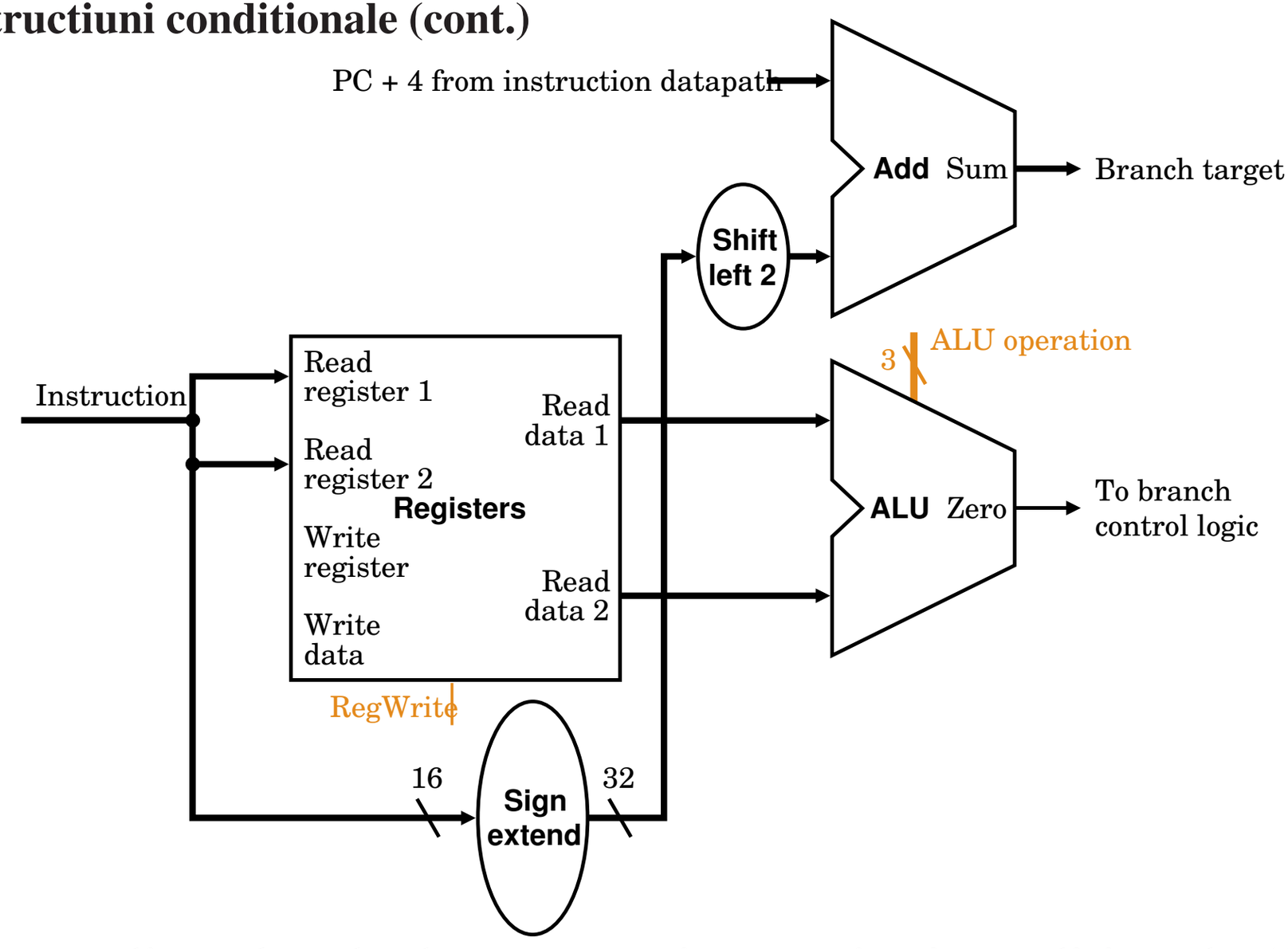
- Se face un salt necondiţionat la adresa, exprimată în cuvinte, specificată de ultimii 26 biţi din instrucţiune.
- Ca implementare, se înlocuiesc biţii 27-0 din PC cu biţii 25-0 din instrucţiune, shift-aţi la stânga cu 2 biţi.

Remarcă:

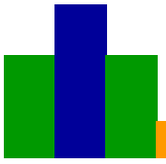
- In MPIS ramificaţiile sunt *întârziate*, anume se execută mereu instrucţiunea următoare, decizia despre test venind mai târziu.
- Dacă testul este pozitiv, se continuă normal; altfel, se revine, se restaurează regiştri şi se acceptă ramificaţia (saltul).

..Instructiuni conditionale

Instructiuni conditionale (cont.)



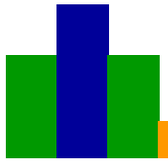
O parte din calea de date pentru instrucţiuni condiţionale (format *I*).



Procesorul: Calea de date si controlul

Cuprins:

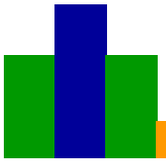
- Generalitati
- Calea de date
- *O prima implementare*
- Implementare cu cicluri multiple
- Microprogramare
- Exceptii
- Concluzii, diverse, etc.



O prima implementare

O prima implementare:

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

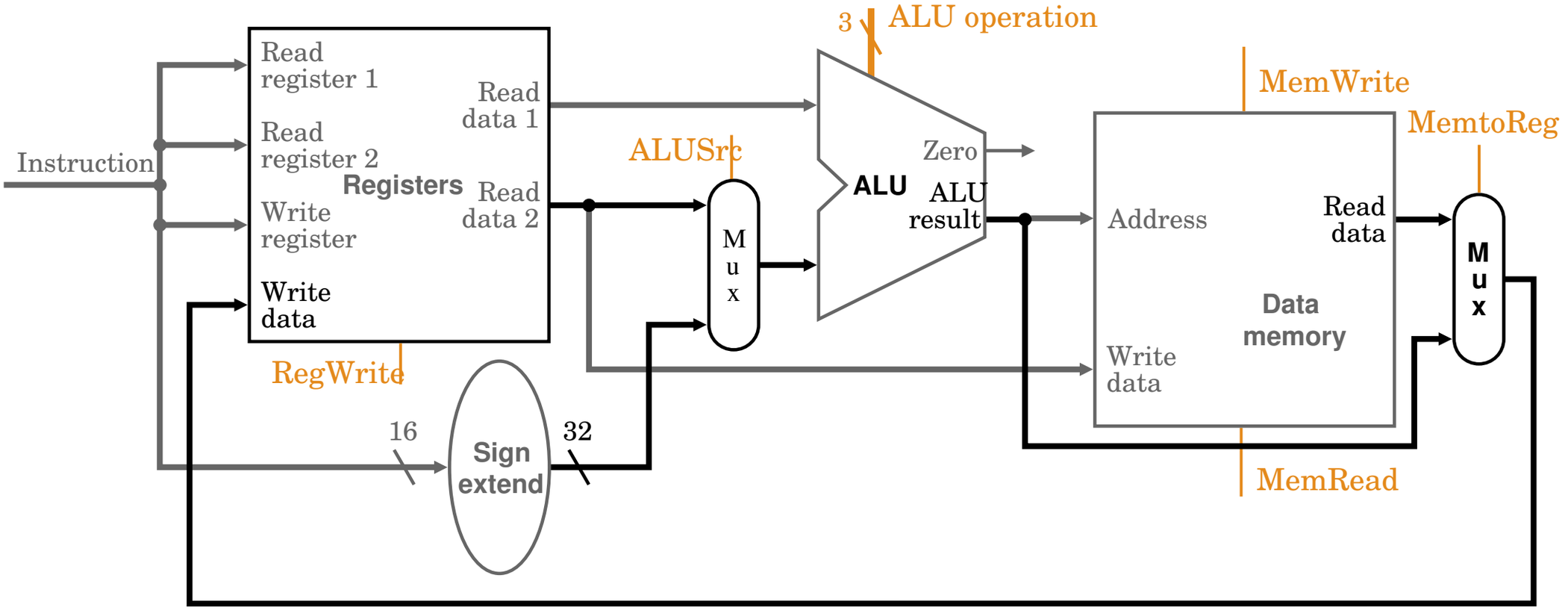


Integrarea cailor de date

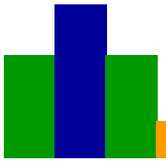
Integrare: operaţii ALU + accesare memorie (tip I + tip II):

- 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 operaţie ALU, iar dacă `(ALUSrc,MentoReg) = (1,0)` se face operaţie de accesare memorie (load).

..Integrarea cailor de date



Combinaţie: *operaţii ALU* + *load / store*.

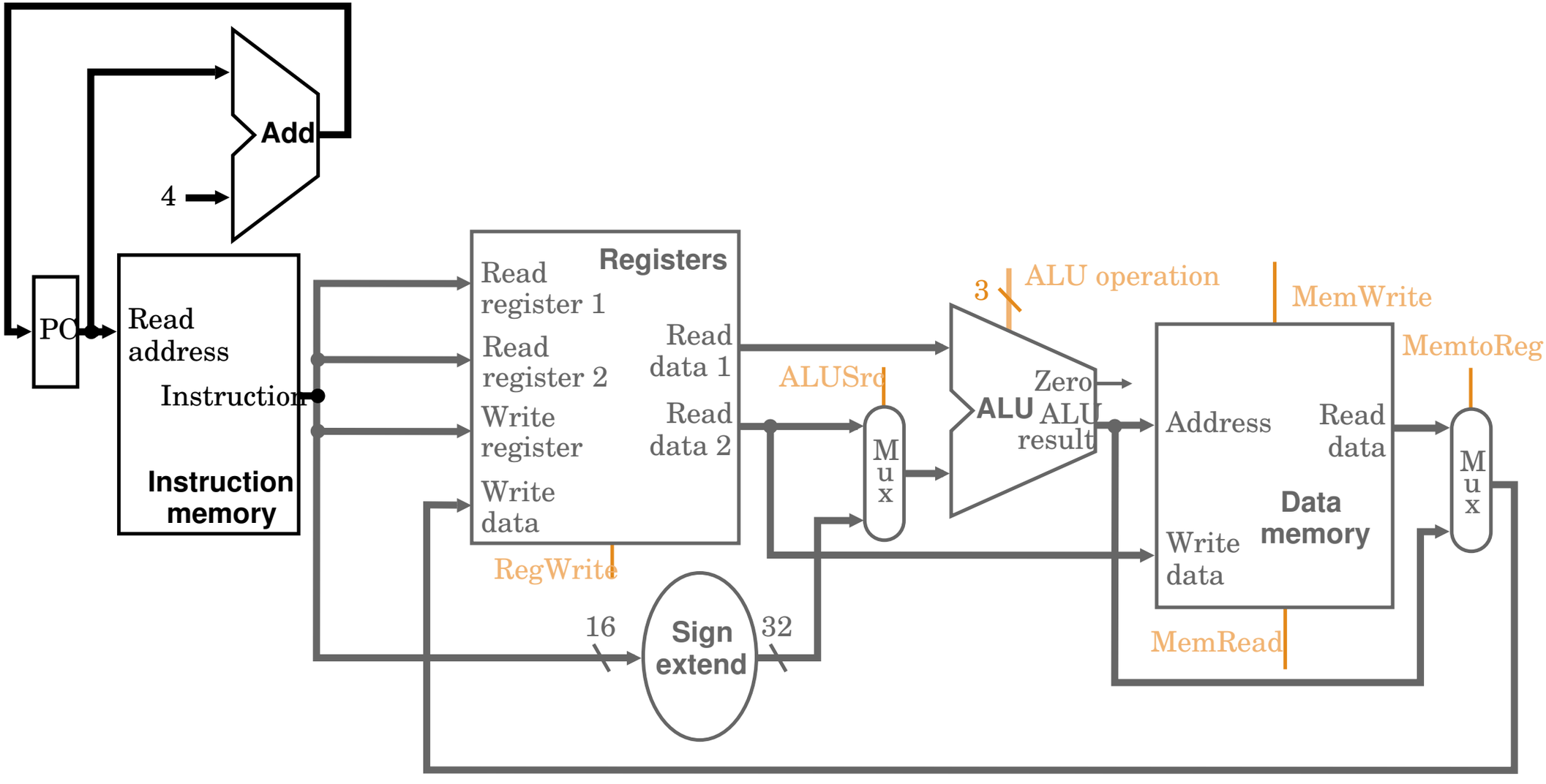


..Integrarea cailor de date

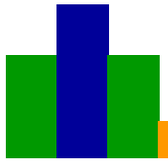
Adaugare - Extractie instructiune:

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

..Integrarea cailor de date



Combinaţie: operaţii ALU + load / store + *extracţie instrucţiune*.

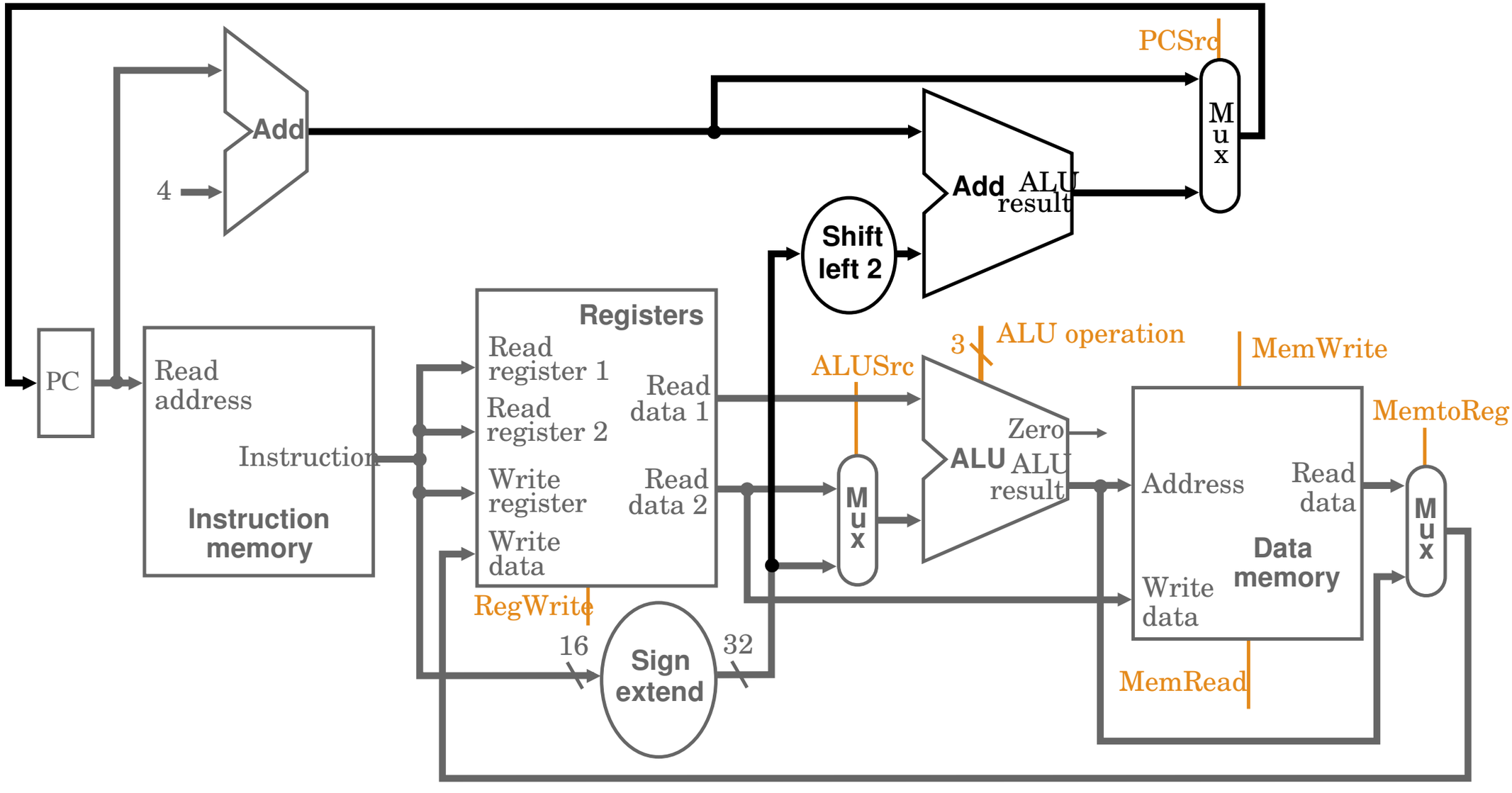


..Integrarea cailor de date

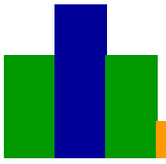
Adaugare - Ramificatii conditionale:

- Integrarea *ramificaţiei condiţionale*:
  - folosim un multiplexor cu controlul `PCSrc` care decide de unde se ia adresa instrucţiunii de exectat;
  - normal adresa este $\texttt{PC} + 4$, iar dacă ramificaţia este acccep-tată, adresa se ia din ieşirea componentei speciale ALU (v. fila a 2-a in continuare).

..Integrarea cailor de date



Combinaţie: operaţii ALU + load / store + extracţie instrucţiune + *ramificaţii condiţionale*



Control ALU

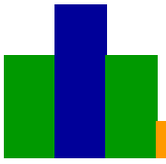
Control ALU:

- 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 *funcţie* al instrucţiunii şi un control de 2b numit `ALUOp`.
- `ALUOp` indică operaţia executată astfel: adunare (00) pentru `lw/sw`, scădere (01) pentru `beq`, ori operaţia din codul de funcţie 5-0 al instrucţiunii (10).

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

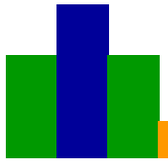


..Control ALU

Control ALU (cont.)

- Tabelul indică cum depinde controlul ALU de `ALUOp` şi biţi 5-0 din instrucţiune (cămpul de funcţie):

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



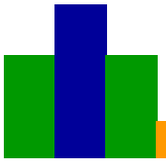
..Control ALU

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 $2^8 = 256$ combinaţii de intrări nu au importanţă (X = "don't care").
- Din tabelă se obţine o *formulă logică*, se optimizează şi se produce un *circuit* pentru controlul ALU.



Unitatea principala de control

Unitatea principala de control: Formatele de instrucţiuni pe care le avem sunt:

- format R pentru operaţii ALU

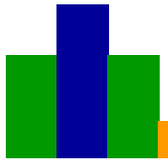
31-26	25-21	20-16	15-11	10-6	0-5
0	rs	rt	rd	shamt	funct

- format I pentru load / store

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

- format I pentru salt condiţionat

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



..Unitatea principala de control

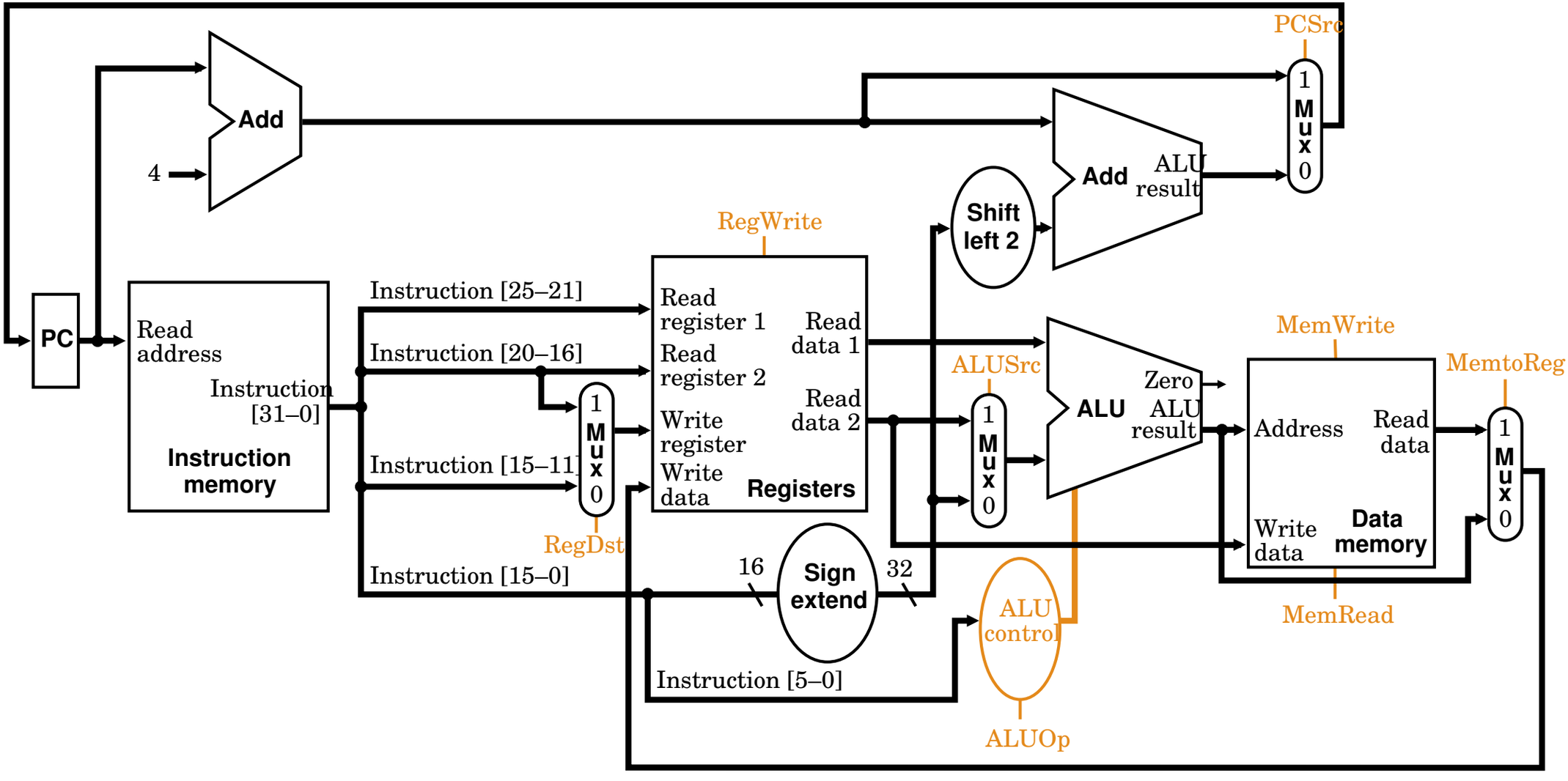
Unitatea principala de control (cont.)

Observaţii:

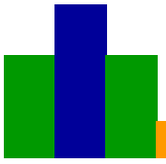
- codul de operaţie, notat `Op[5-0]` este în biţii `31-26`;
- cei 2 regiştri de citit pentru operaţile ALU, branch şi store sunt în `rs` şi `rt` (zonele `25-21` şi `20-16`);
- la load / store registrul de bază este în `rs` (zona `25-21`);
- la branch, load şi store offset-ul este pe 16b în zona `15-0`;
- registrul destinaţie este
  - la operaţiile ALU în `rd` (zona `15-11`);
  - la load în `rt` (zona `20-16`);

Obţinem *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 şi 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 cânpul 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 ramificaţiei;

MemRead: 0 - nimic; 1 - la ieşire 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 principala de control

Unitatea principala de control (cont.)

- Cu o excepţie, toate semnalele de mai sus se deduc *complet* din corpul instrucţiunii.
- Excepţia este `PCSrc`, pentru care este necesară şi ieşirea `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

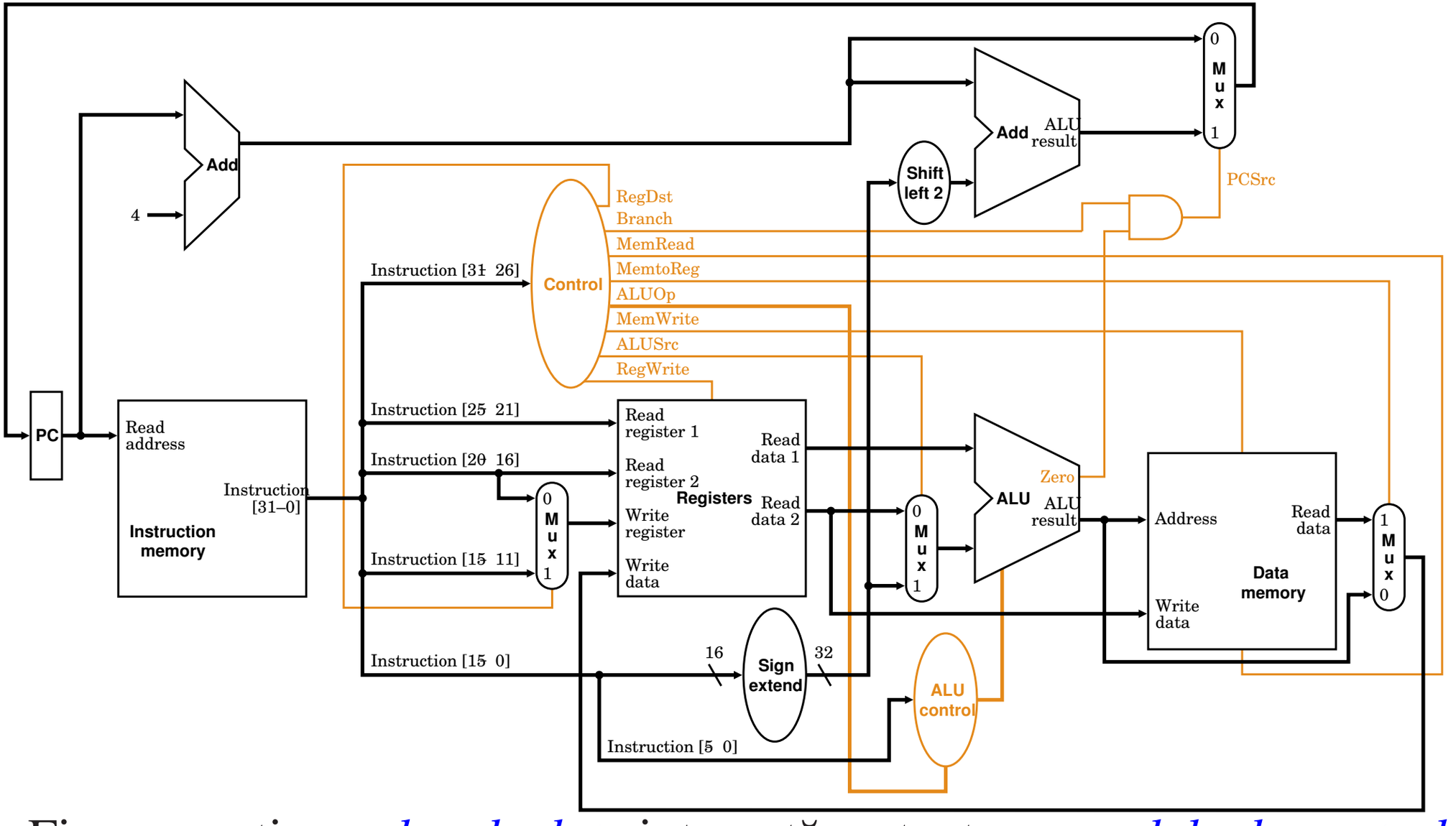
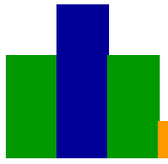


Figura conţine *calea de date* integrată cu toate *semnalele de control* şi *unităţile de control* (ALU Control + Control).



Procesarea unei instructiuni

Procesarea unei instructiuni: Exemplificăm procesorul obţiunt pe câteva instrucţiuni tipice:

- Instrucţiune de format *R* (operaţie ALU)

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

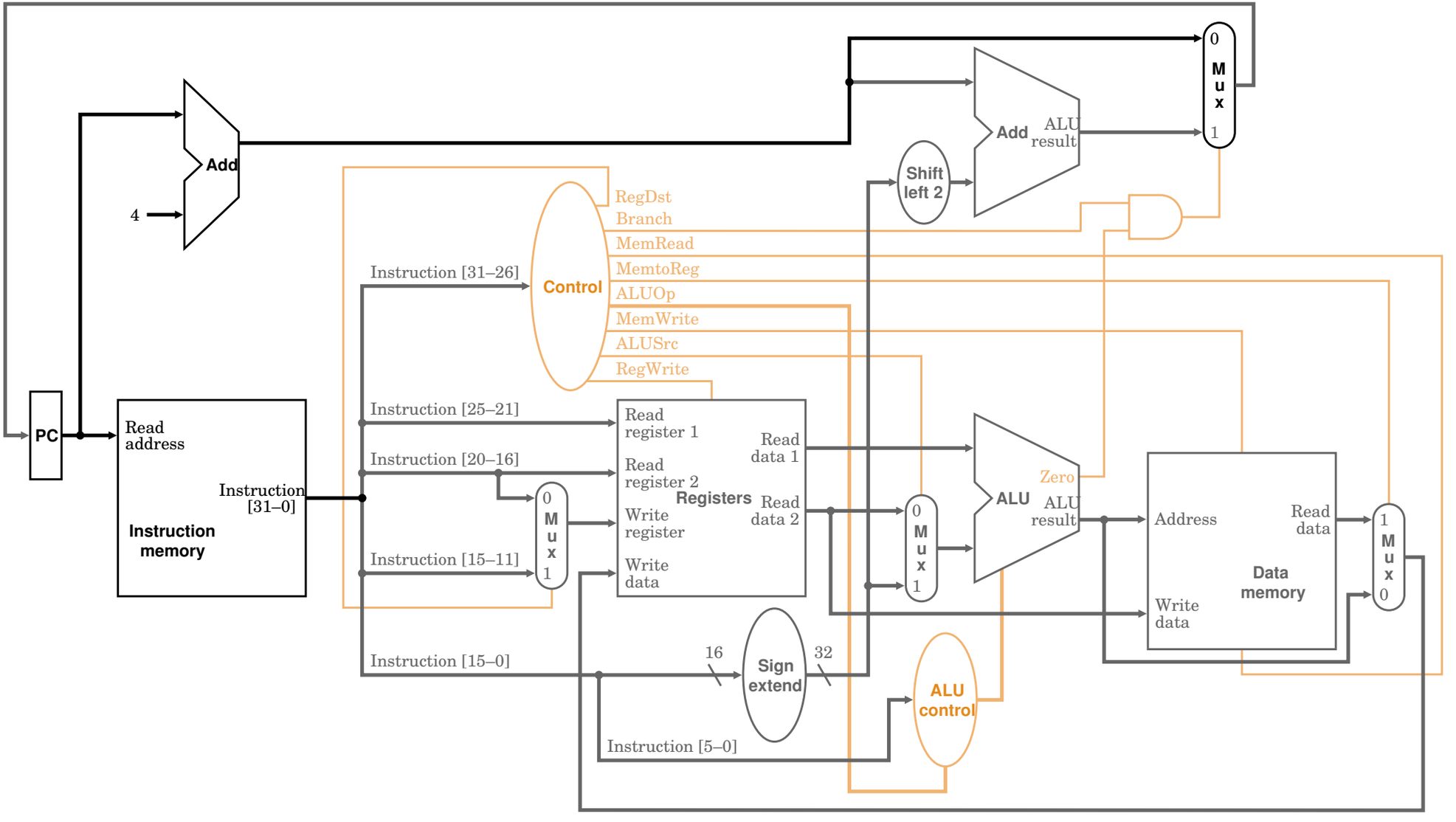
- Instrucţiune de accesare a memoriei

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

- Instrucţiune condiţională

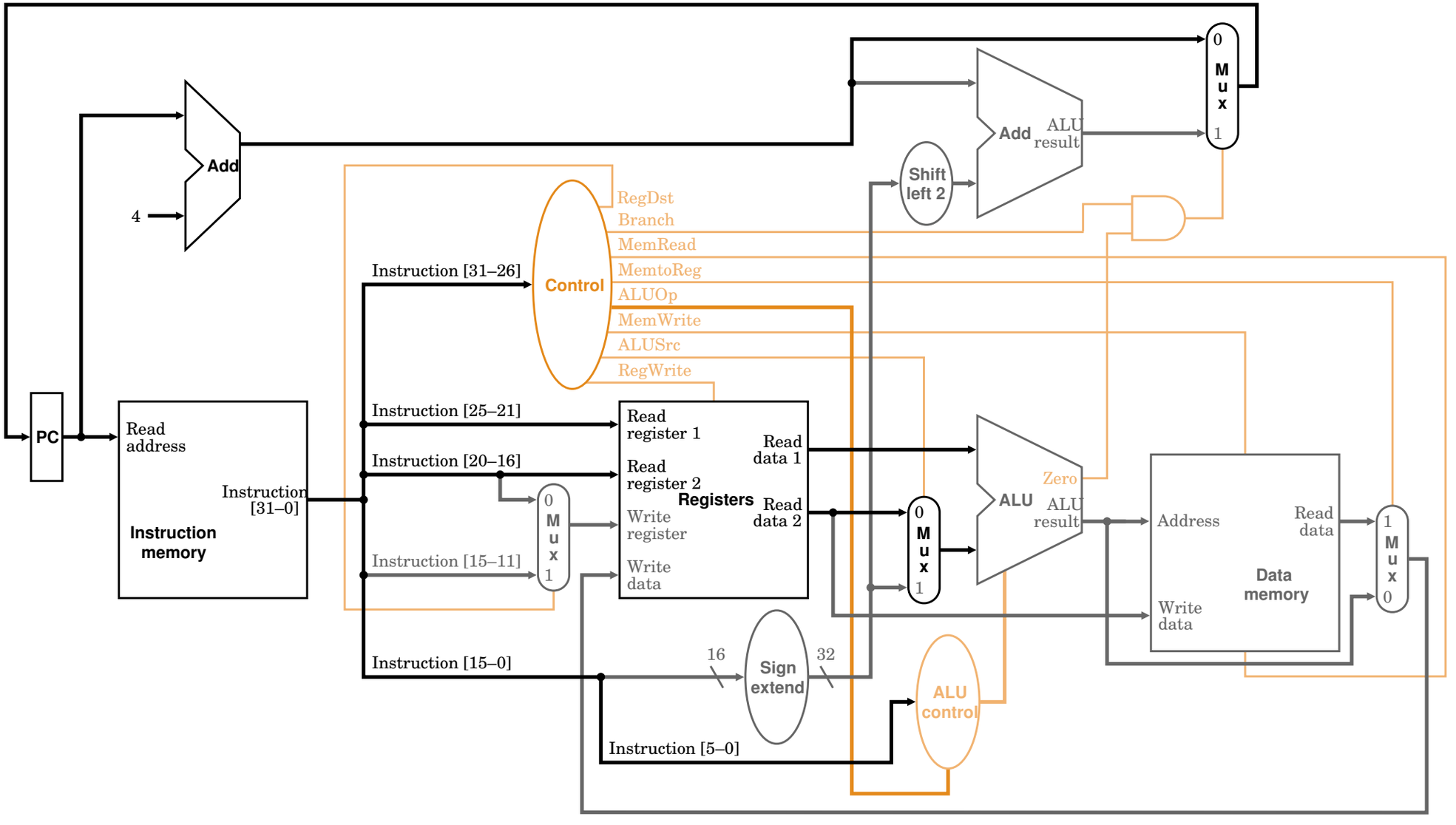
```
beq $t1,$t2,offset;
```

Procesarea unei instructiuni - format R



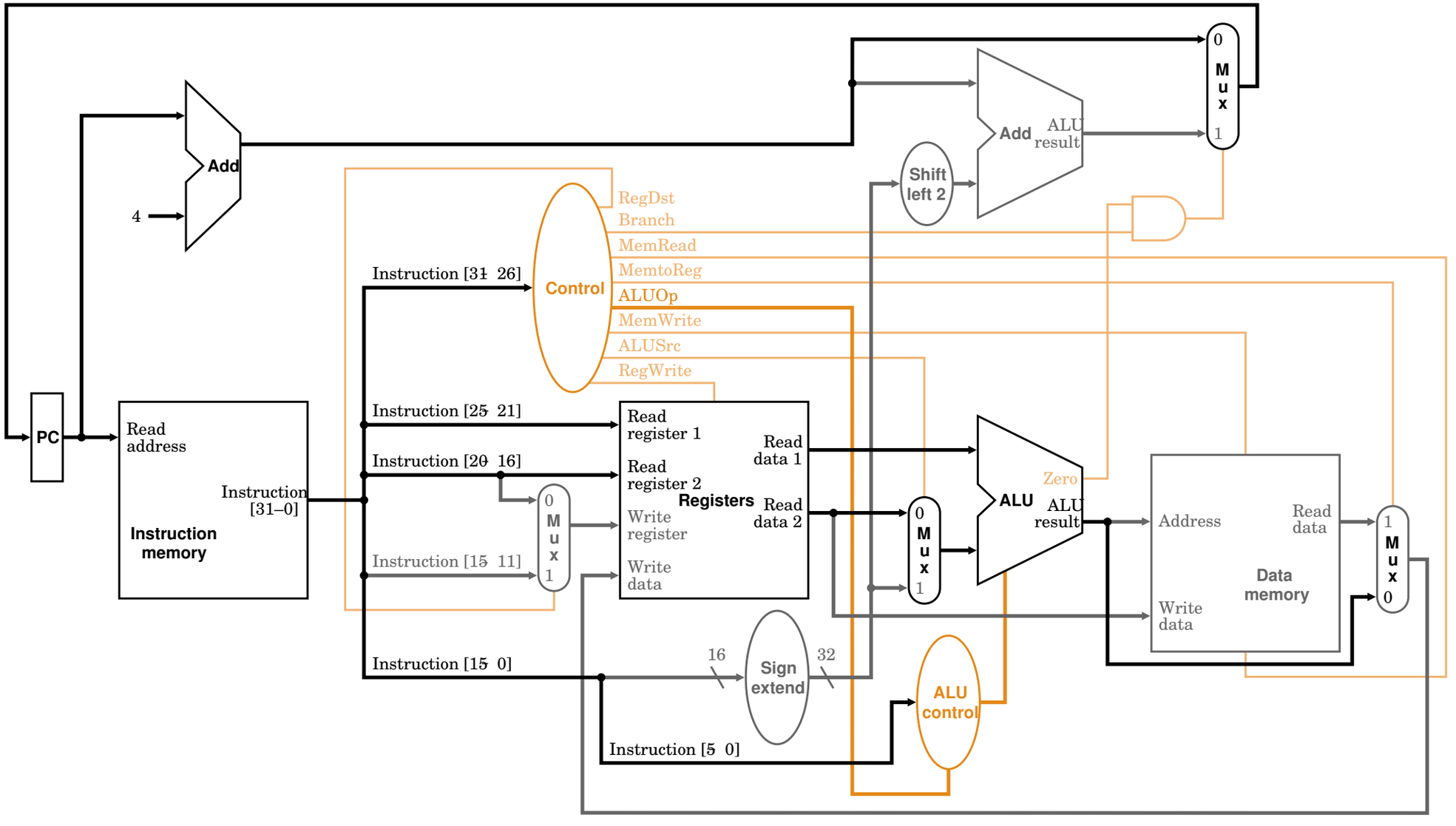
Primul pas la procesarea unei instrucţiuni de *tip R*: se *extrage instrucţiunea* şi se *incrementează PC*.

..Procesarea unei instructiuni - format R



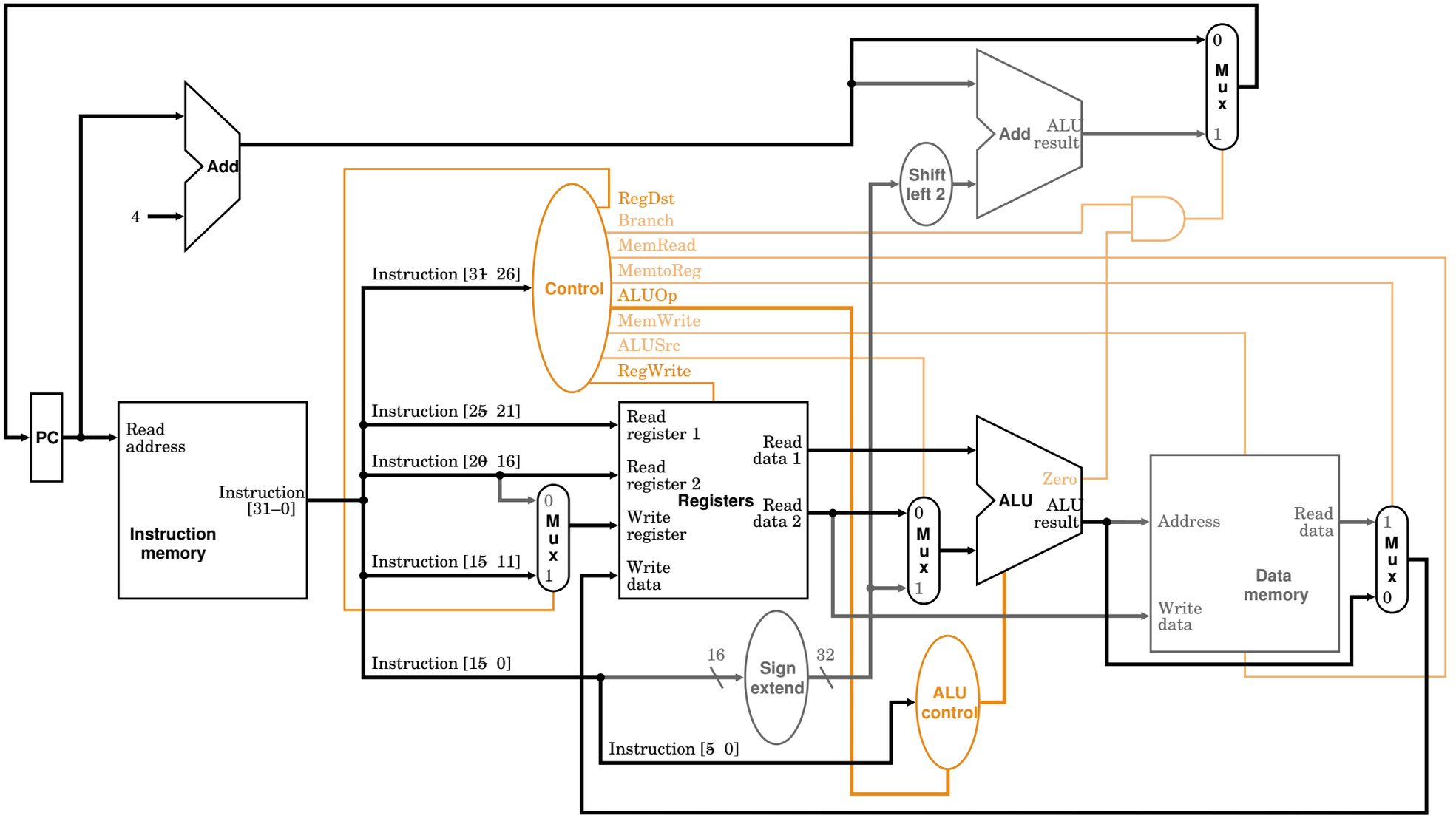
Al doilea pas la procesarea unei instrucţiuni de *tip R*: se *citesc 2 regiştri* sursă din RF (Register File).

..Procesarea unei instructiuni - format R



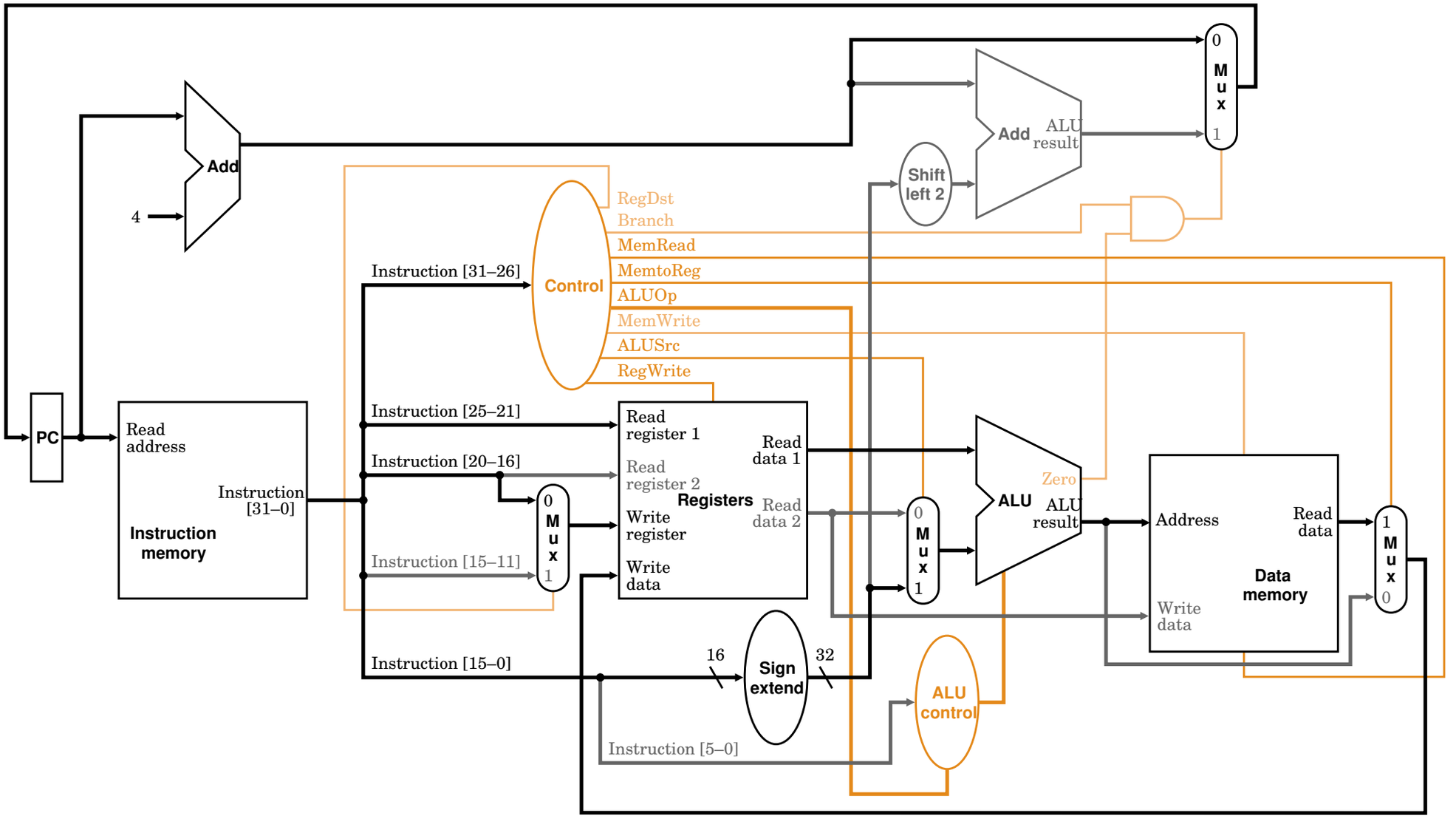
Al treilea pas la procesarea unei instrucţiuni de *tip R*: se *calculează în ALU* folosind datele din regiştri sursă.

..Procesarea unei instructiuni - format R



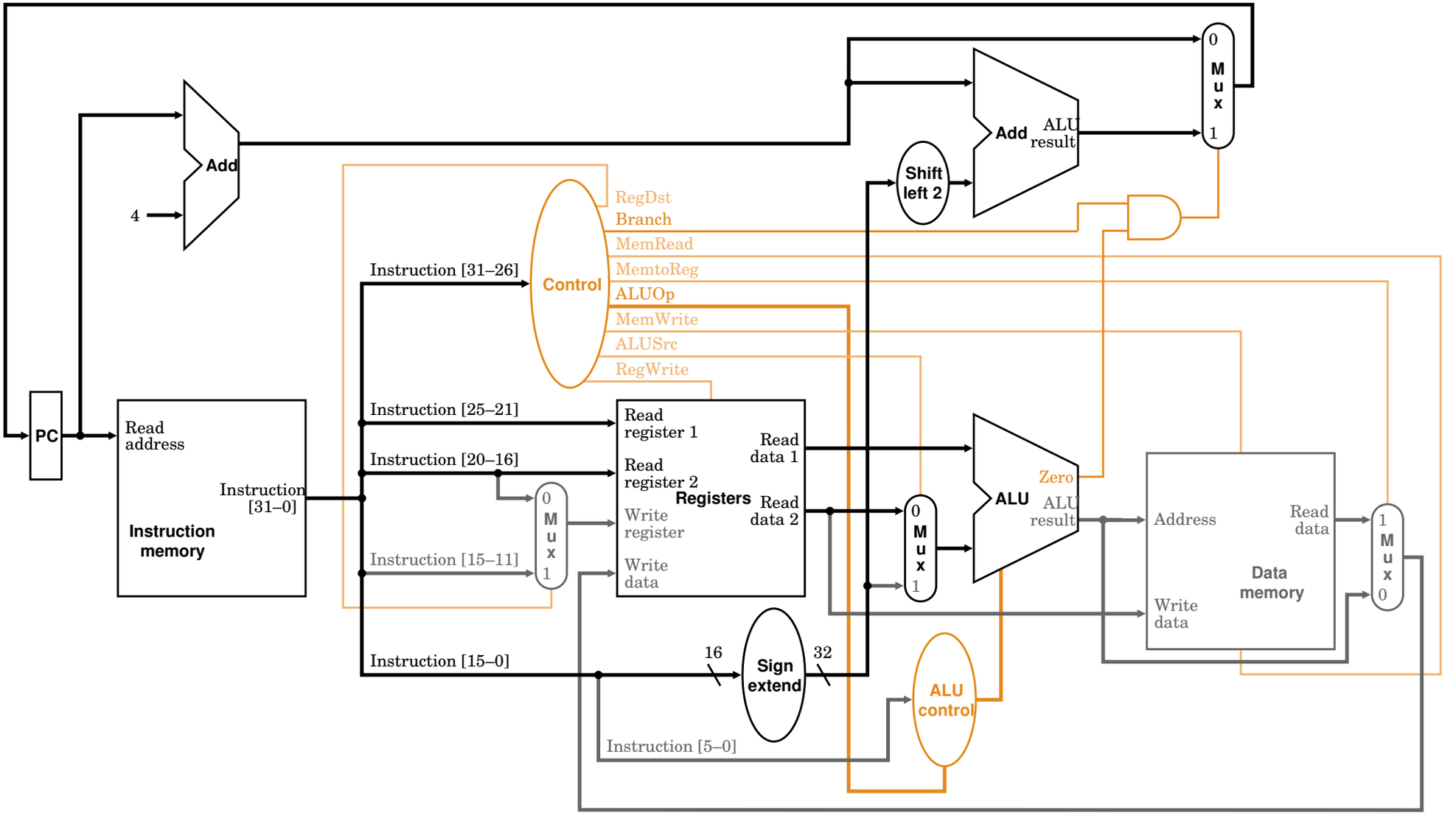
Pasul final la procesarea unei instrucţiuni de *tip R*: se *scrie în registru* rezultatul obţinut în ALU.

Procesarea unei instructiuni - LOAD

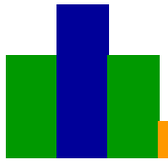


Procesarea unei instrucţiuni *load*. Pentru *store* este similar, dar se scrie în memorie (nu se citeşte) o data ce vine din registru.

Procesarea unei instructiuni - BNE



Procesarea unei instrucţiuni *condiţionale*. De notat că ieşirea `zero` din ALU se foloseşte spre a decide noua instrucţiune.



Prima implementare de procesor

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

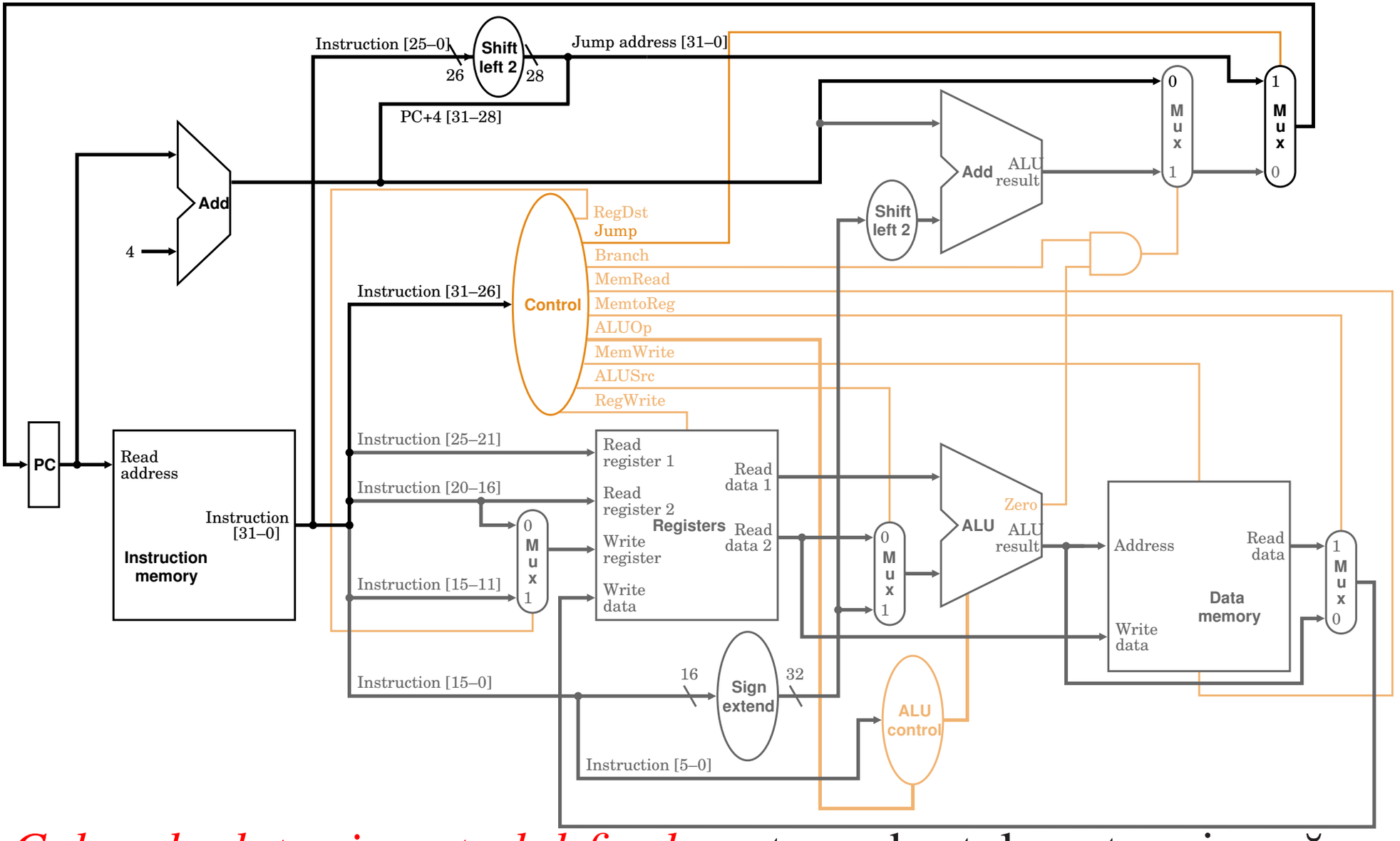
- formatul pentru jump este formatul J

31-26	25-0
2	address

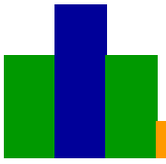
- adresa este în cuvinte, deci se multiplică cu 4 (shift la stânga cu 2 biţi); cei 28 biţi rezultaţi se plasează în poziţiile `27-0` ale rezultatului;
- poziţiile din faţă `31-28` ale rezultatului se copiază din `PC+4`.

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

..Prima implementare de procesor



Calea de date şi controlul final pentru subsetul nostru: inserăm şi procesarea saltului necondiţionat *jump*.



Procesorul: Calea de date si controlul

Cuprins:

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