Arhitectura Calculatoarelor

Curs 4: Proiectarea MIPS cu ciclu de ceas unic

E-mail: florin.oniga@cs.utcluj.ro

Web: http://users.utcluj.ro/~onigaf, secțiunea Teaching/AC

Conținutul / obiectivul cursului

- Se prezintă o **metodă sistematică, pe pași, de proiectare a unui procesor**
- Se pornește de la arhitectura setului de instrucțiuni
- Rezultatul este procesorul format din calea de date și unitatea de control
- Studiul de caz din acest curs: **Procesorul MIPS pe 32 de biți, cu un ciclu de ceas pe instrucțiune** – MIPS ciclu-unic

Faze de proiectare pentru un procesor

1. **Se analizează setul de instrucțiuni la nivel RTL→ cerințe pentru căi de date**
   - Secvențe de micro-operații (RTL) pentru instrucțiunile din ISA
   - Expresiile RTL specifică componentele și conexiunile pentru calea de date
   - Diagrama de procesare (vezi curs 3).

2. **Se selectează setul de componente și disciplina semnalului de ceas**
   - Se definește citirea/scrierea elementelor de stare, ex. FF - basculare pe front +
   - Calea critică în căile de date pentru definirea perioadei semnalului de ceas

3. **Se asamblează calea de date conform cerințelor**
   - Se formează o cale de date inițială (ex. registre, ALU, memorii).
   - Daca mai *multe surse sunt conectate la o destinație*, se adaugă un MUX / Magistrală
   - Se completează secvența de micro-operații pentru toate instrucțiunile, și se adaugă componentele și conexiunile/multiplexoarele necesare

Faze de proiectare pentru un procesor

4. **Se identifică și se definesc semnalele de comandă ale componentelor în calea de date**
   - Pentru fiecare instrucțiune se stabilește valoarea semnalelor de comandă (pe fiecare tact, la multi-ciclu), necesară pentru asigurarea transferurilor corespunzătoare între registre

5. **Se asamblează logica de comandă / control**
   - Proiectarea unității de comandă pe baza semnalelor de comandă identificate
   - 3 tipuri de UC
     - Logică combinațională → single cycle CPU (fiecare instrucțiune se execută într-un ciclu)
     - Hard-Wired, secvențial: Implementare tip mașină de stare
     - Micro-programat.

Proiectarea MIPS ciclu unic – Pas 1

În această secțiune se urmărește descrierea din materialul obligatoriu, care poate fi consultată opțional pentru detalii suplimentare (versiunea este derivata din [1]).

Pas de proiectare 1: selecția setului de instrucțiuni - Subset MIPS-lite

- Se selectează un număr de instrucțiuni reprezentative

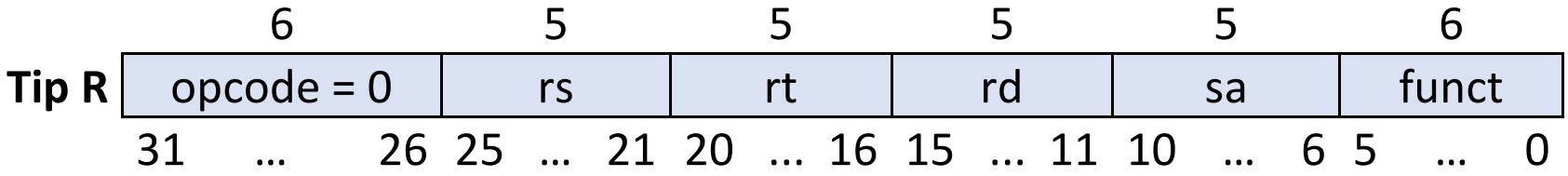
Instrucțiune	RTL	PC – modificare
op $rd, $rs, $rt	RF[rd] ← RF[rs] op RF[rt], op poate fi add, sub, or, and	PC ← PC + 4
ori $rt, $rs, imm	RF[rt] ← RF[rs] or Z_Ext(imm)	PC ← PC + 4
addi $rt, $rs, imm	RF[rt] ← RF[rs] + S_Ext(imm)	PC ← PC + 4
lw $rt, imm($rs)	RF[rt] ← M[RF[rs] + S_Ext(imm)]	PC ← PC + 4
sw $rt, imm($rs)	M[RF[rs] + S_Ext(imm)] ← RF[rt]	PC ← PC + 4
beq $rs, $rt, imm	If(RF[rs] == RF[rt]) then	PC ← PC + 4 + S_Ext(imm) <<2
	else	PC ← PC + 4
j target_addr	PC ← (PC+4)[31:28] & target_addr & 00	

- RTL abstract definește comportamentul fiecărei instrucțiuni
  - Faze de Execuție pentru o instrucțiune (descriere independentă de ceas!, curs anterior): IF, ID (ID/OF), EX, MEM, WB
  - IF și ID sunt la fel pentru toate instrucțiunile

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip R

- Generic: op $rd, $rs, $rt
- Operația de bază: RF[rd] ← RF[rs] op RF[rt]
- PC la următoarea instrucțiune: PC ← PC + 4
- OPCODE este tot timpul 0 pentru tip R



	op code				sa	funct
add	000000	rs	rt	rd	00000	100000 = add
sub	000000	rs	rt	rd	00000	100010 = sub
or	000000	rs	rt	rd	00000	100101 = or
and	000000	rs	rt	rd	00000	100100 = and

Exemple

add $1, $2, $3 **cod mașină: 000000 00010 00011 00001 00000 100000**

and $9, $2, $3 **cod mașină: 000000 00010 00011 01001 00000 100100**

Proiectarea MIPS ciclu unic – Pas 1

Următorul pas ? …analiza RTL si/sau diagrama de procesare, vezi cursul 3.

Proiectarea MIPS ciclu unic – Pas 1

Tip R

Din analiza descrierii RTL rezultă resursele necesare (atenție, în RTL nu apare explicit citirea instrucțiunii - IF, dar se consideră implicită):

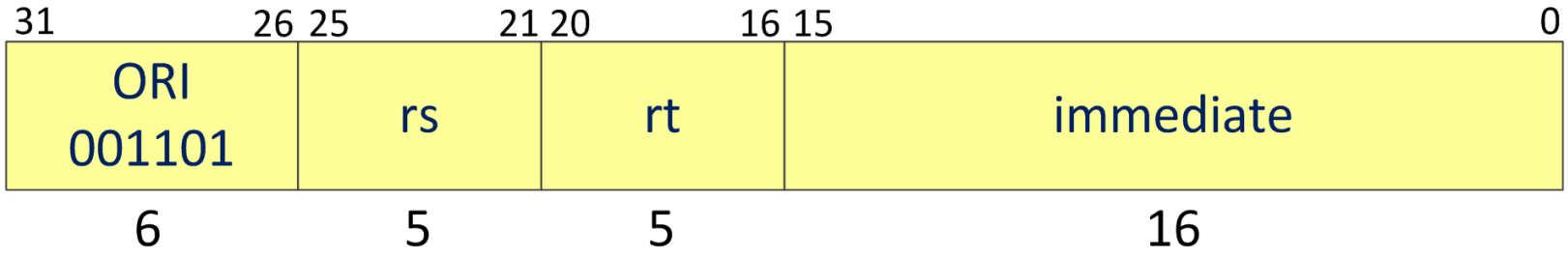
Resurse necesare pe faze de execuție	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	Bloc de registre, *Unitatea principală de control*
EX	ALU, *Unitatea de control ALU*
MEM	No operation
WB	Bloc de registre

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip I: OR Immediate – ORI

ori $rt, $rs, imm

Scop: sau logic între un registru ($rs) și o constantă (imm), rezultatul în $rt



Adresare:

- cu registru și imediat

RTL abstract

- RF[rt] ← RF[rs] | Z_Ext(imm)
- PC ← PC + 4

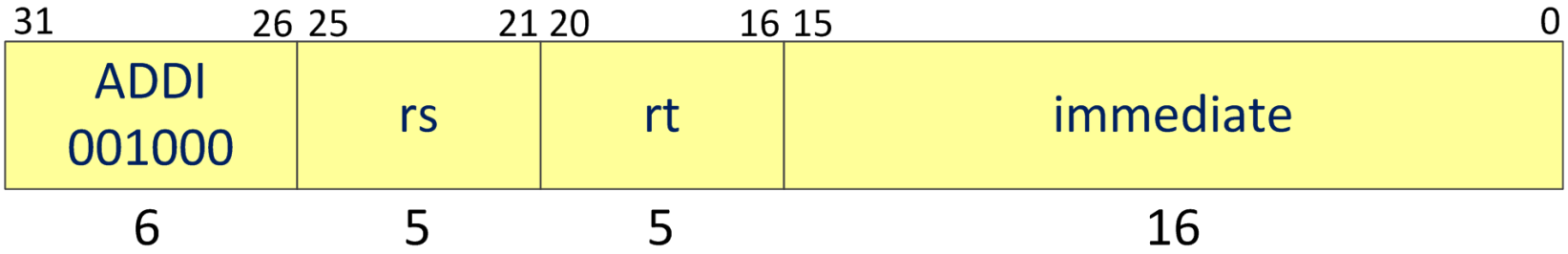
Resurse necesare	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	Bloc de registre, Unit. princ. de control, Extensie
EX	ALU, Unitatea de control ALU
MEM	No operation
WB	Bloc de registre

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip I: ADD Immediate – ADDI

addi $rt, $rs, imm

Scop: adunare între un registru ($rs) și o constantă (imm), rezultatul în $rt



Adresare:

- cu registru și imediat

RTL abstract

- RF[rt] ← RF[rs] + S_Ext(imm)
- PC ← PC + 4

Resurse necesare	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	Bloc de registre, Unit. princ. de control, Extensie
EX	ALU, Unitatea de control ALU
MEM	No operation
WB	Bloc de registre

Proiectarea MIPS ciclu unic – Pas 1

Exemple de codificare:

ori $11, $11, 2 **cod mașină: 001101 01011 01011 0000000000000010**

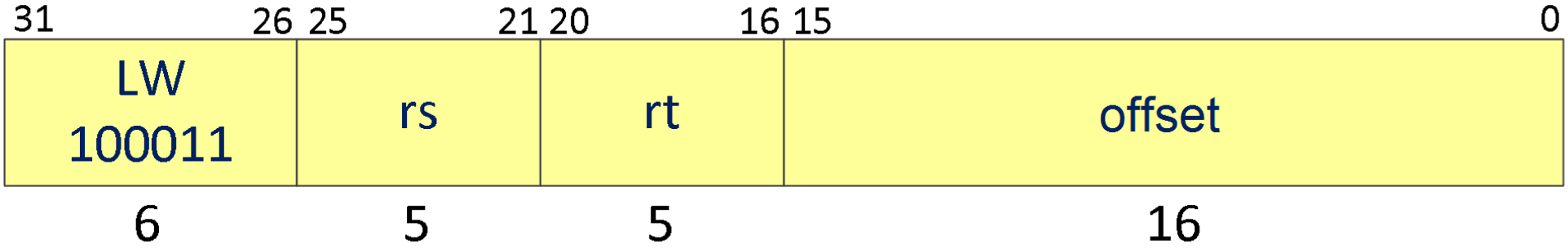
addi $5, $3, -2 **cod mașină: 001000 00011 00101 1111111111111110**

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip I: Load Word - LW

lw $rt, offset($rs)

Scop: Încarcă un cuvânt de 32 de biți din memoria de date într-un registru



Adresare:

- cu registru
- cu bază a memoriei

RTL abstract

- RF[rt] ← M[RF[rs] + S_Ext(offset)]
- PC ← PC + 4

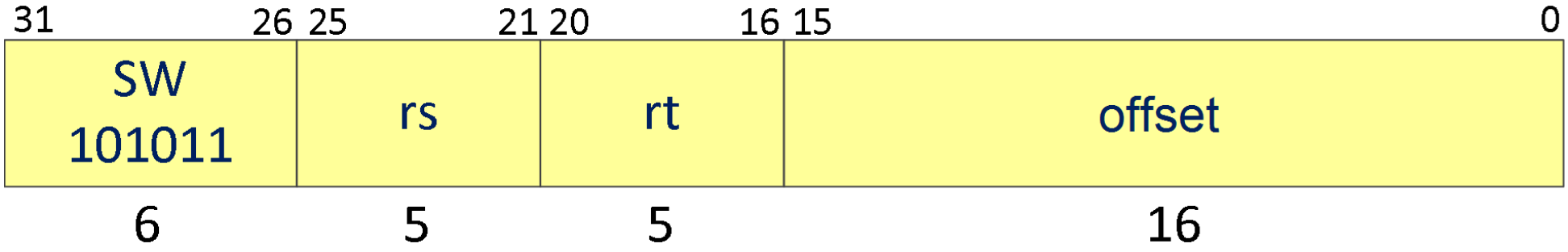
Resurse necesare	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	Bloc de registre, Unit. princ. de control, Extensie
EX	ALU, Unitatea de control ALU
MEM	Memoria de date
WB	Bloc de registre

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip I: Store Word - SW

sw $rt, offset($rs)

Scop: Încarcă un cuvânt de 32 de biți dintr-un registru în memoria de date



Adresare:

- cu registru
- cu bază a memoriei

RTL abstract

- M[RF[rs] + S_Ext(offset)] ← RF[rt]
- PC ← PC + 4

Resurse necesare	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	Bloc de registre, Unit. princ. de control, Extensie
EX	ALU, Unitatea de control ALU
MEM	Memoria de date
WB	No operation

Proiectarea MIPS ciclu unic – Pas 1

Exemple de codificare:

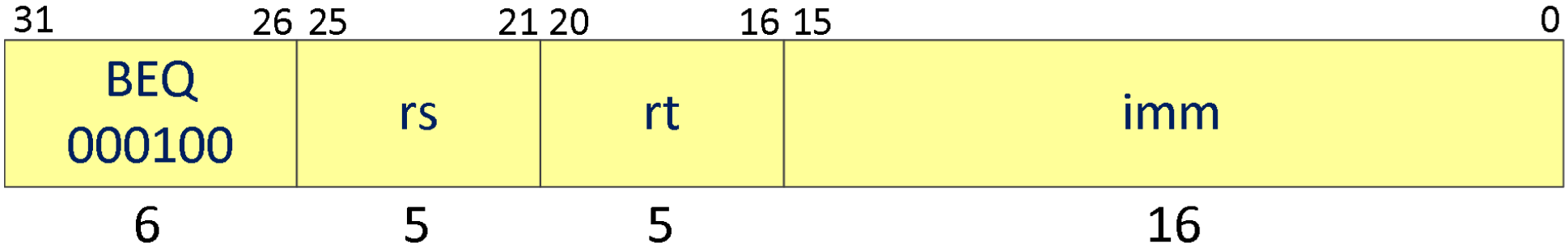
lw $3, -1($8)	cod mașină: **100011 01000 00011 1111111111111111**
lw $31, 254($8)	cod mașină: **100011 01000 11111 0000000011111110**
sw $7, 30($5)	cod mașină: **101011 00101 00111 0000000000011110**
sw $12, -11($2)	cod mașină: **101011 00010 01100 1111111111110101**

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip I: Branch on Equal – BEQ

beq $rt, $rs, imm

Scop: compară două registre, apoi face salt condiționat relativ la PC



Adresare:

- Relativă la PC

RTL abstract:

If (RF[rs] == RF[rt]) then

PC ← PC + 4 + S_Ext(imm) <<2

else

PC ← PC + 4

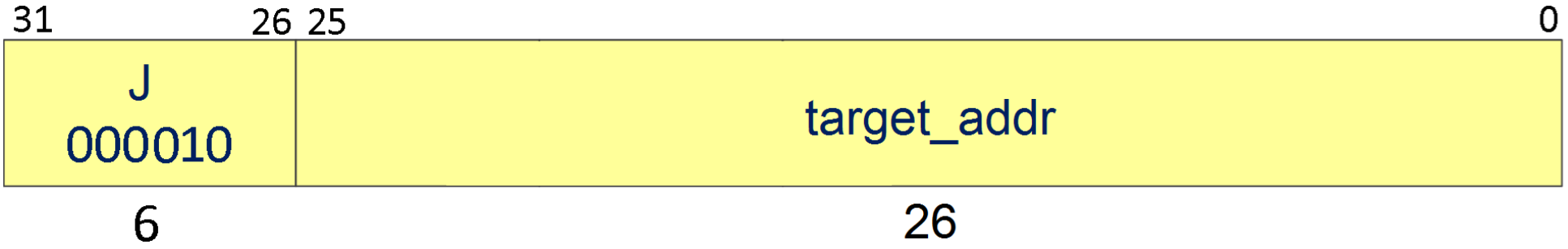
Resurse necesare	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	Bloc de registre, Unit. princ. de control, Extensie
EX	ALU, Unitatea de control ALU, **sumator, circuit deplasare, MUX**
MEM	No operation
WB	No operation

Proiectarea MIPS ciclu unic – Pas 1

Instrucțiuni de tip J: Jump

j target_addr

Scop: salt necondiționat la adresă pseudo-absolută (în zona de 256 MB unde e programul)



Adresare:

- Pseudo-absolută (în segment)

RTL abstract:

PC ← (PC+4)[31:28] & target_addr & 00

Resurse necesare	
IF	PC, Memoria de instrucțiuni, sumator
ID/OF	No operation
EX	Circuit de deplasare
MEM	No operation
WB	No operation

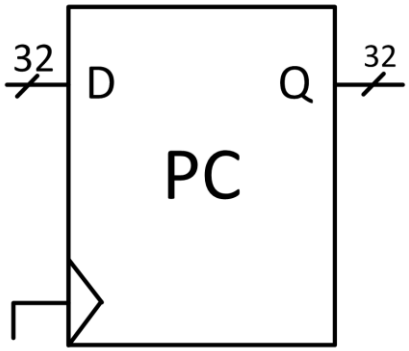
Proiectarea MIPS ciclu unic – Pas 1

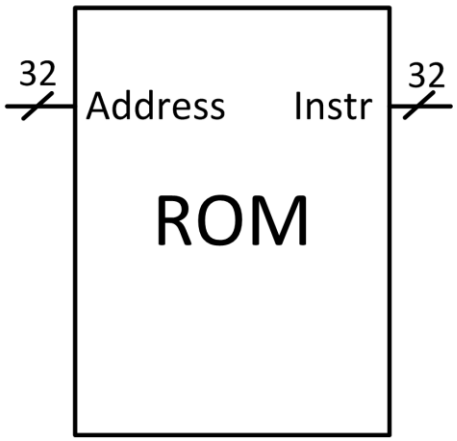
Cerințe de resurse pentru instrucțiunile analizate până acum

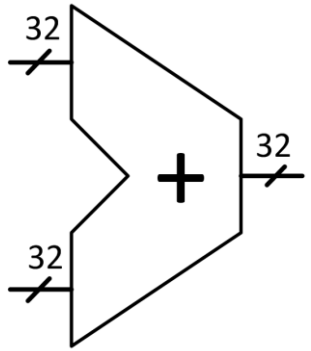
- PC – contorul de program
- Memorii
  - Instrucțiuni și Date
- Blocul de Registre (32 x 32)
  - Citire R[rs], Citire R[rt]
  - Scrie în R[rt] sau R[rd]…o singură adresă de scriere…=> MUX
- Extensie cu semn / zero pentru câmpul address / immediate, <<2
- Unitatea aritmetică-logică – ALU => MUX
  - Operații cu două registre
  - Operații cu un registru și cu o valoare imediată extinsă
- Pentru valoarea următoare a PC => MUX
  - Sumator pentru PC+4
  - Sumator pentru (PC+4) + Imediat extins cu semn

Proiectarea MIPS ciclu unic – Pas 2

Pas de proiectare 2: Componentele căilor de date (varianta ciclu unic)

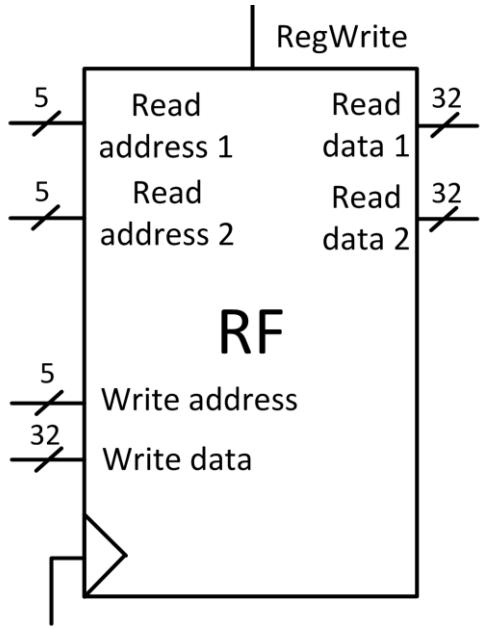






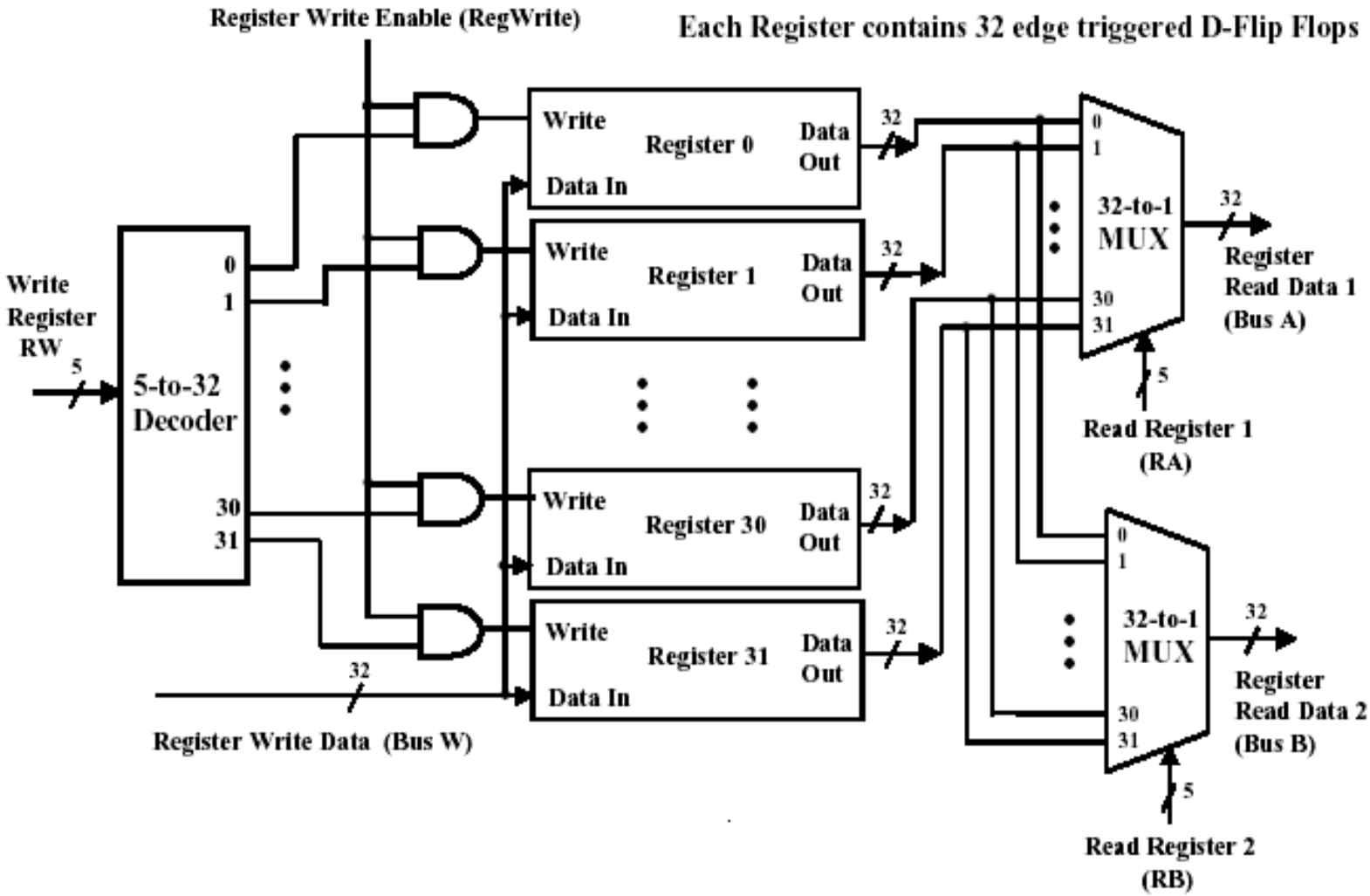
- PC contorul de program
  - Registru 32-biti, bistabil D cu încărcare pe front pozitiv
- Memoria de instrucțiuni (model ROM idealizat)
  - O intrare: Adresă de instrucțiune
  - O ieșire: Instrucțiunea
  - Cuvântul de memorie este selectat combinațional de adresă, fără alte semnale de control
- Sumator (x2) pentru formarea următoarei adrese de instrucțiune
  - Ex. 32-bit ripple carry

Proiectarea MIPS ciclu unic – Pas 2



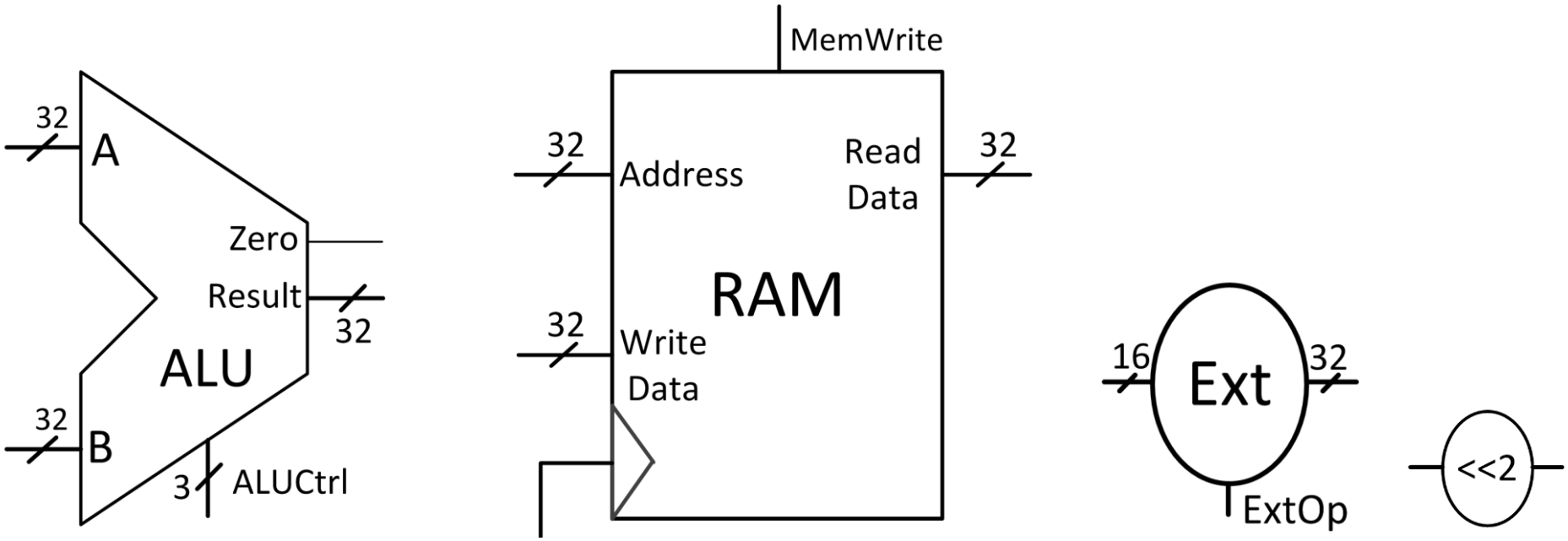
- Bloc de Registre, 32x32 biți
  - Model didactic cu flip-flop-uri D vs. multi-acces SRAM în procesoare reale
  - Două iesiri de date, (2 x 32-biti): Read data 1 si Read data 2
  - O intrare de date, 32-biti: Write data
  - Multi-acces: 2 Citiri asincrone, 1 Scriere pe front (+); în aceeași perioadă de ceas
  - Selectia Registrelor:
    - **Read addres 1,2** selectează data pentru ieșirea **Read data 1,2**
    - **Write address** - adresă, selectează registrul în care se va scrie (Write data) dacă se activează RegWrite și frontul pozitiv al ceasului
      - La citire, blocul de registre se comportă ca un bloc logic combinaţional

Proiectarea MIPS ciclu unic – Pas 2



Model de Bloc de Registre (Intern / Exemplu)

Proiectarea MIPS ciclu unic – Pas 2



- ALU - proiectată conform cerințelor (se va discuta în cursul viitor)

- Memoria de date (model SRAM idealizat)
  - Două intrări: Adresa și data de scris
  - O ieșire: Data citită, asincron, *MemRead rol de activare* (**nefolosit la ciclu unic!**)
  - Scrierea se face sincron, front de ceas, cu semnalul MemWrite cu rol de activare
  - Semnal de comandă MemWrite

- Unitatea de Extensie și circuit de deplasare cu 2 poziții (stânga)
  - ExtOp = 1 → extensie cu semn
  - ExtOp = 0 → extensie cu zero

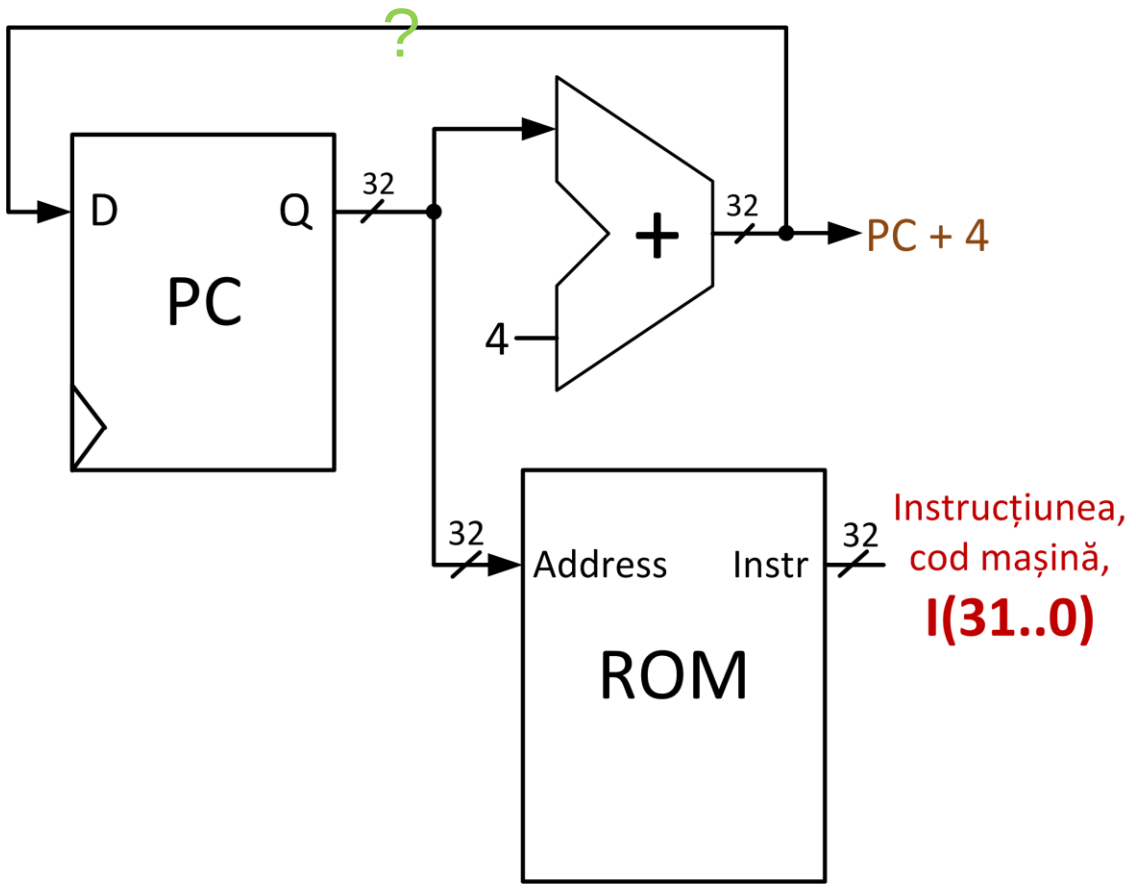
Proiectarea MIPS ciclu unic – Pas 2

Pas de proiectare 2: Disciplina semnalului de ceas

- Disciplina de ceas
  - Definește momentul de scriere și / sau citire pentru semnale
  - Impune momentele când datele trebuie sa fie valide și stabile față de ceas (…Tsetup, Thold…).
- Alternative de sincronizare / temporizare (momentul scrierii)
  - **activare pe front crescător**
  - activare pe front descrescător
  - activare pe două faze (ambele fronturi)
- Pentru toate elementele de stocare (ex. PC, Bloc de registre, Memoria de date) scrierea se face pe același front de ceas
  - În mod implicit, elemente de stocare sunt scrise la fiecare ciclu de ceas
  - Altfel, sunt necesare semnale explicite de control a scrierii
  - Scrierea se face doar când semnalul de control este activat și apare frontul de ceas

Proiectarea MIPS ciclu unic – Pas 3 (doar IF)

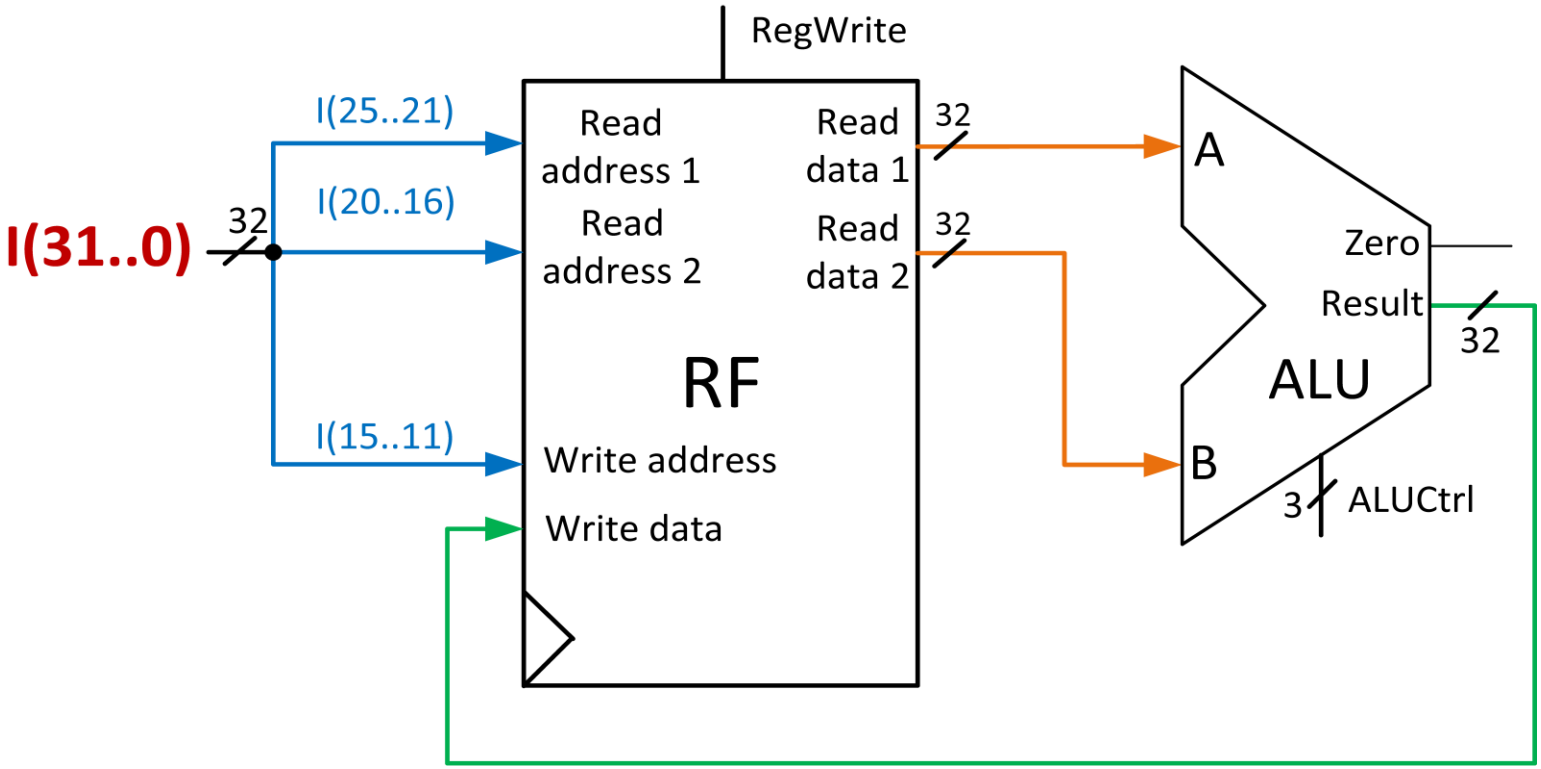
Pas de proiectare 3: Asamblarea căilor de date



Instruction Fetch;

- RTL IF: **I(31..0)** ← IM [PC], PC ← PC + 4;
- PC – registru 32 biți, Add – Sumator 32 biți, 4 – increment 4 octeți pentru un cuvânt, Memoria de instrucțiuni ROM

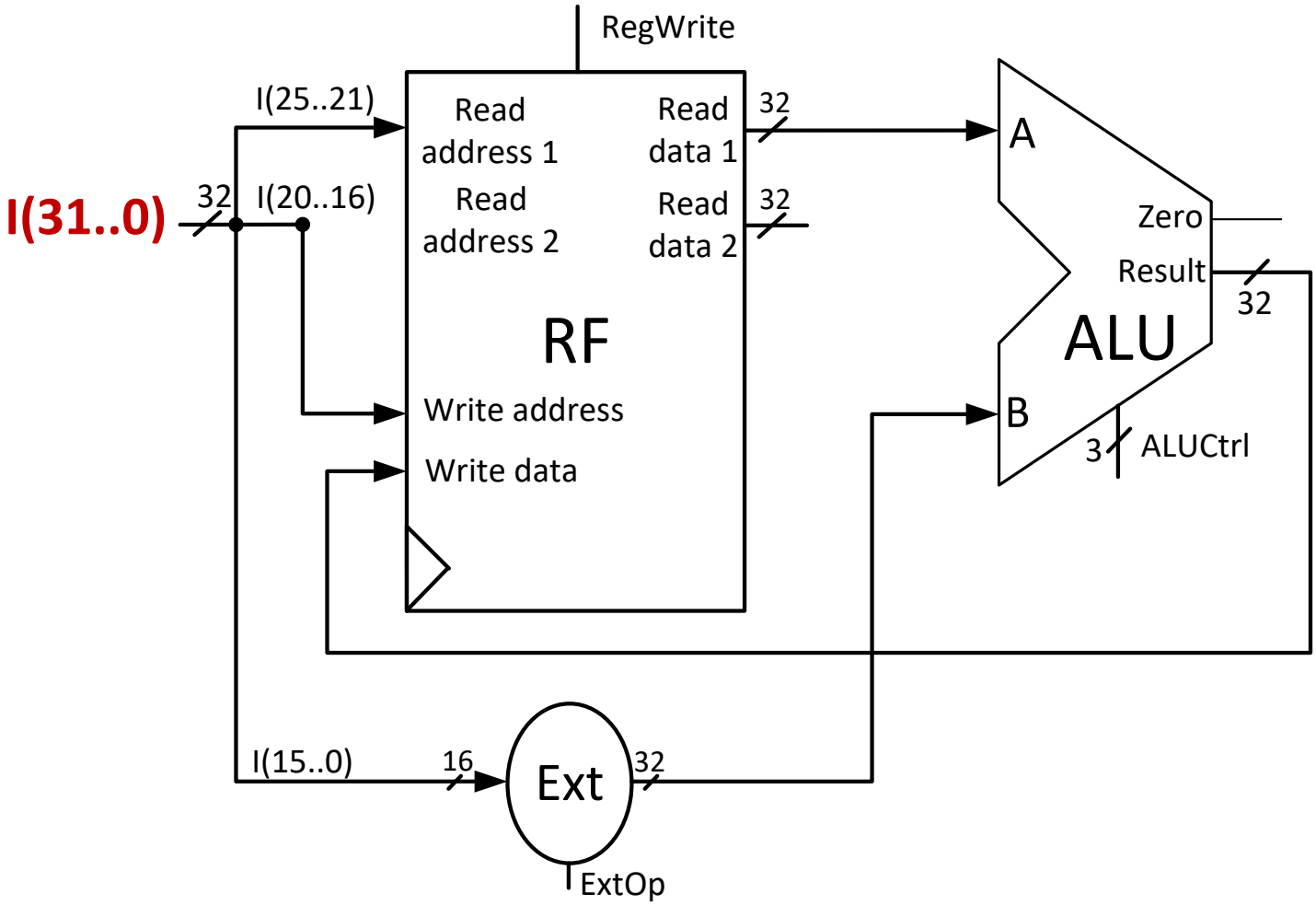
Proiectarea MIPS ciclu unic – Pas 3 (ID, EX, MEM, WB)



Tip R: RTL RF[rd] ← RF[rs] op RF[rt]

1. RF[**rd**] ← RF[**rs**] op RF[**rt**]: rd, rs și rt câmpuri din codul mașină al instrucțiunii I(31..0). **rs** – Read address 1, **rt** – Read address 2, **rd** – Write address
2. RF[rd] ← **RF[rs]** op **RF[rt]**: RF[rs] și RF[rt] sunt ieșirile de date din RF, operația op este executată de ALU. Read data 1 – intrare A din ALU, Read data 2 – intrare B din ALU
3. RF[rd] ← **RF[rs] op RF[rt]**: valoarea (RF[rs] op RF[rt]) este disponibilă pe ieșirea ALU și trebuie scrisă în RF. ALU Result – Write Data.

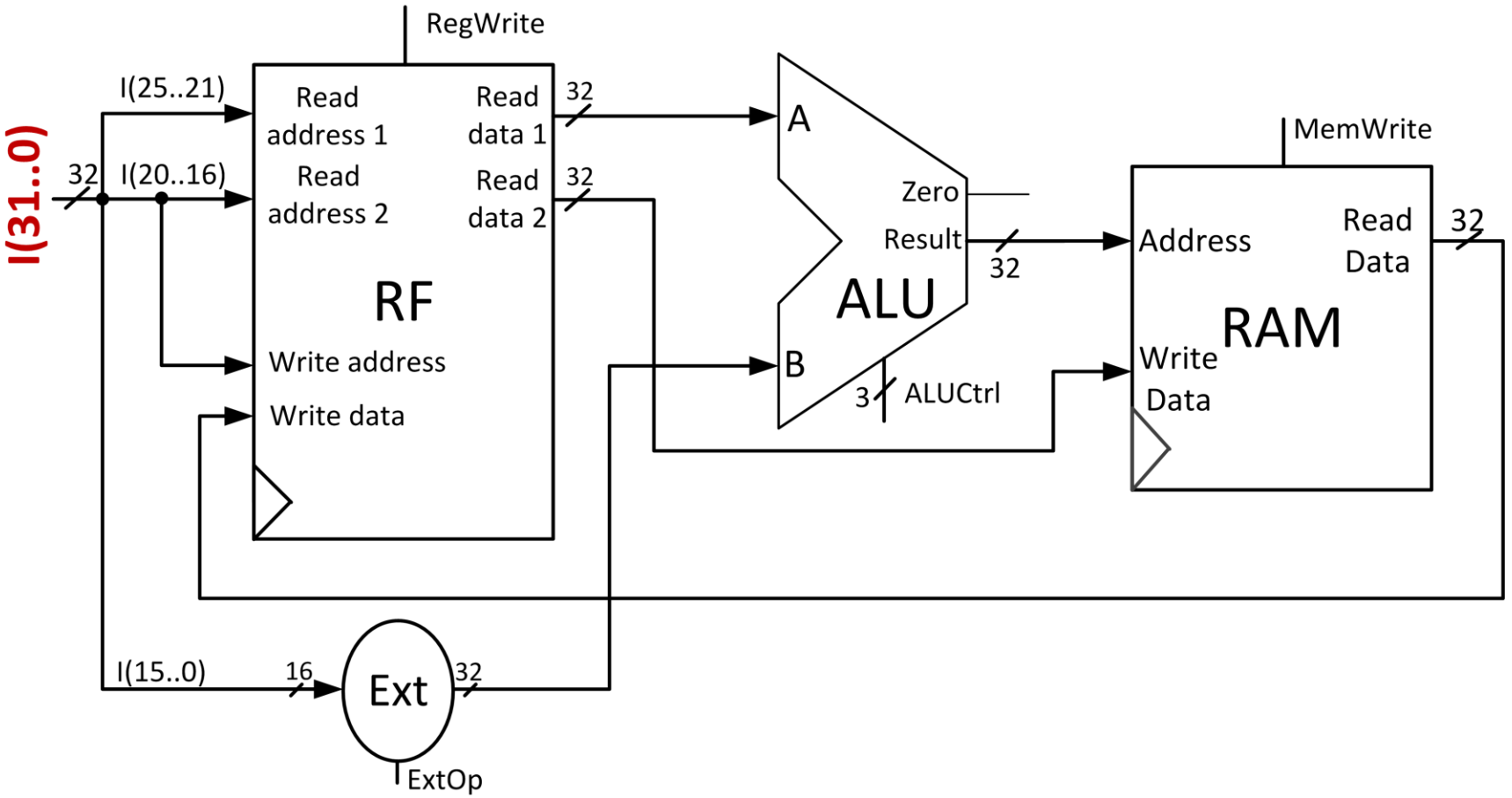
Proiectarea MIPS ciclu unic – Pas 3



Tip I: ori: RF[rt] ← RF[rs] or Z_Ext(imm)

addi: RF[rt] ← RF[rs] + S_Ext(imm)

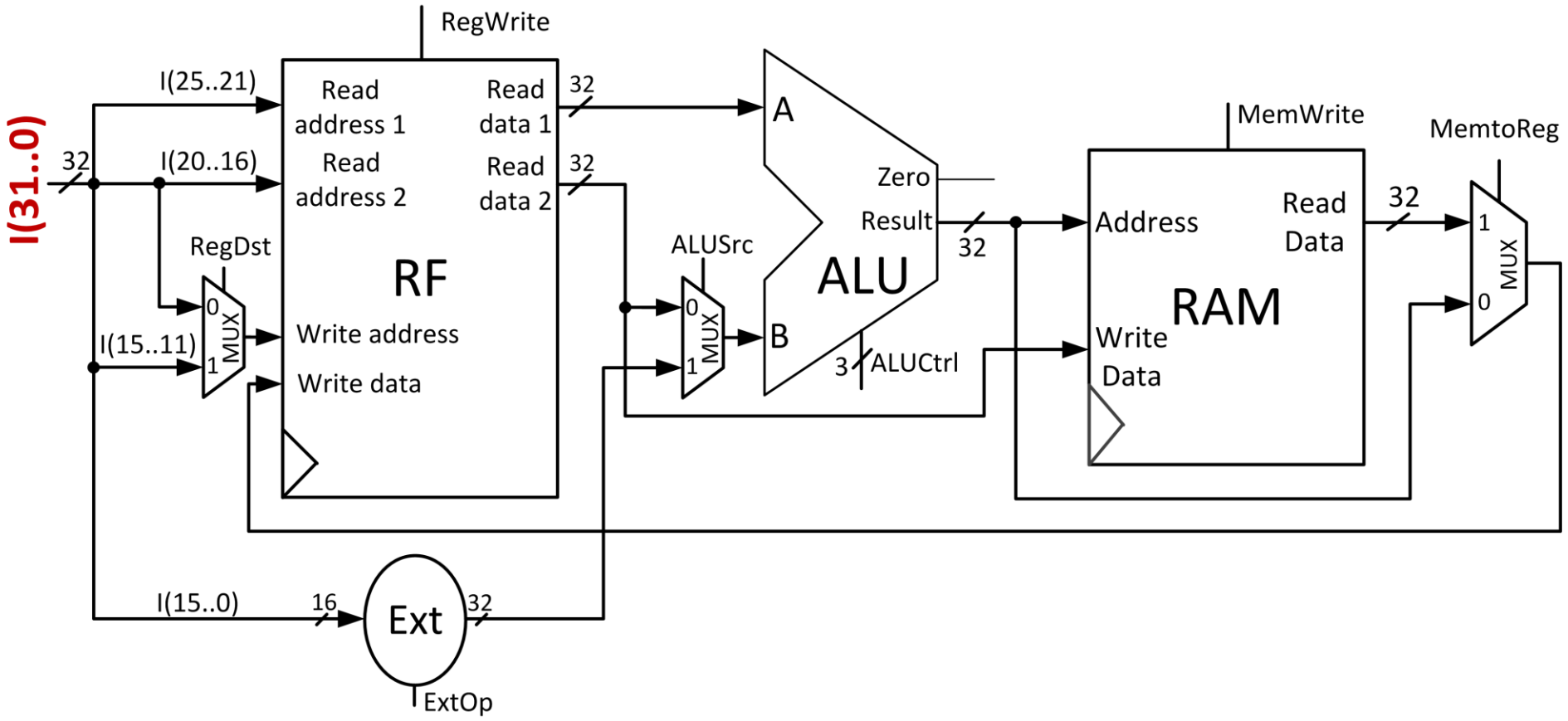
Proiectarea MIPS ciclu unic – Pas 3



Tip I: load word: RF[rt] ← M[RF[rs] + S_Ext(imm)]

store word: M[RF[rs] + S_Ext(imm)] ← RF[rt]

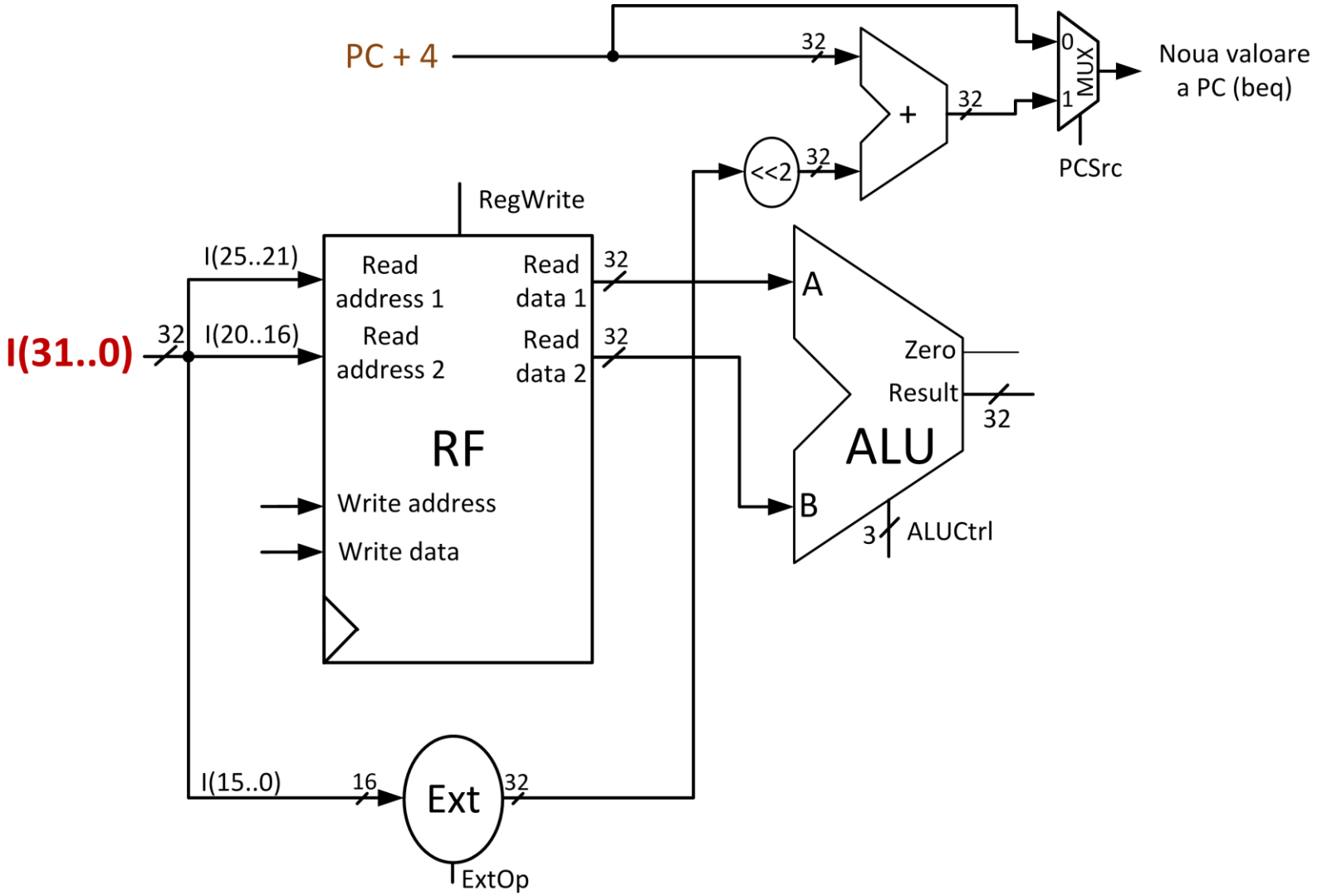
Proiectarea MIPS ciclu unic – Pas 3



Unificare căi de date parțiale Tip I (fără beq)+R: Tip R, addi, ori, lw, sw

- Apar 3 mux-uri cu semnale de control RedDst, ALUSrc, MemtoReg !

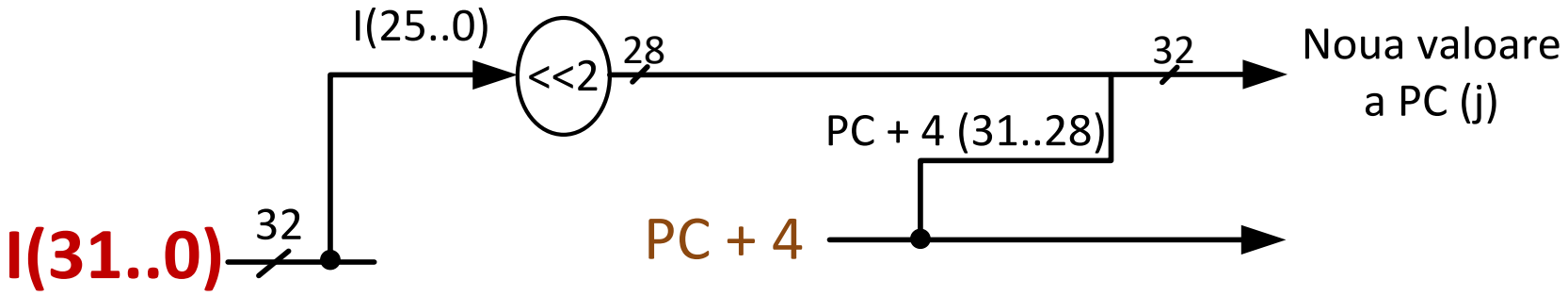
Proiectarea MIPS ciclu unic – Pas 3



Tip I:

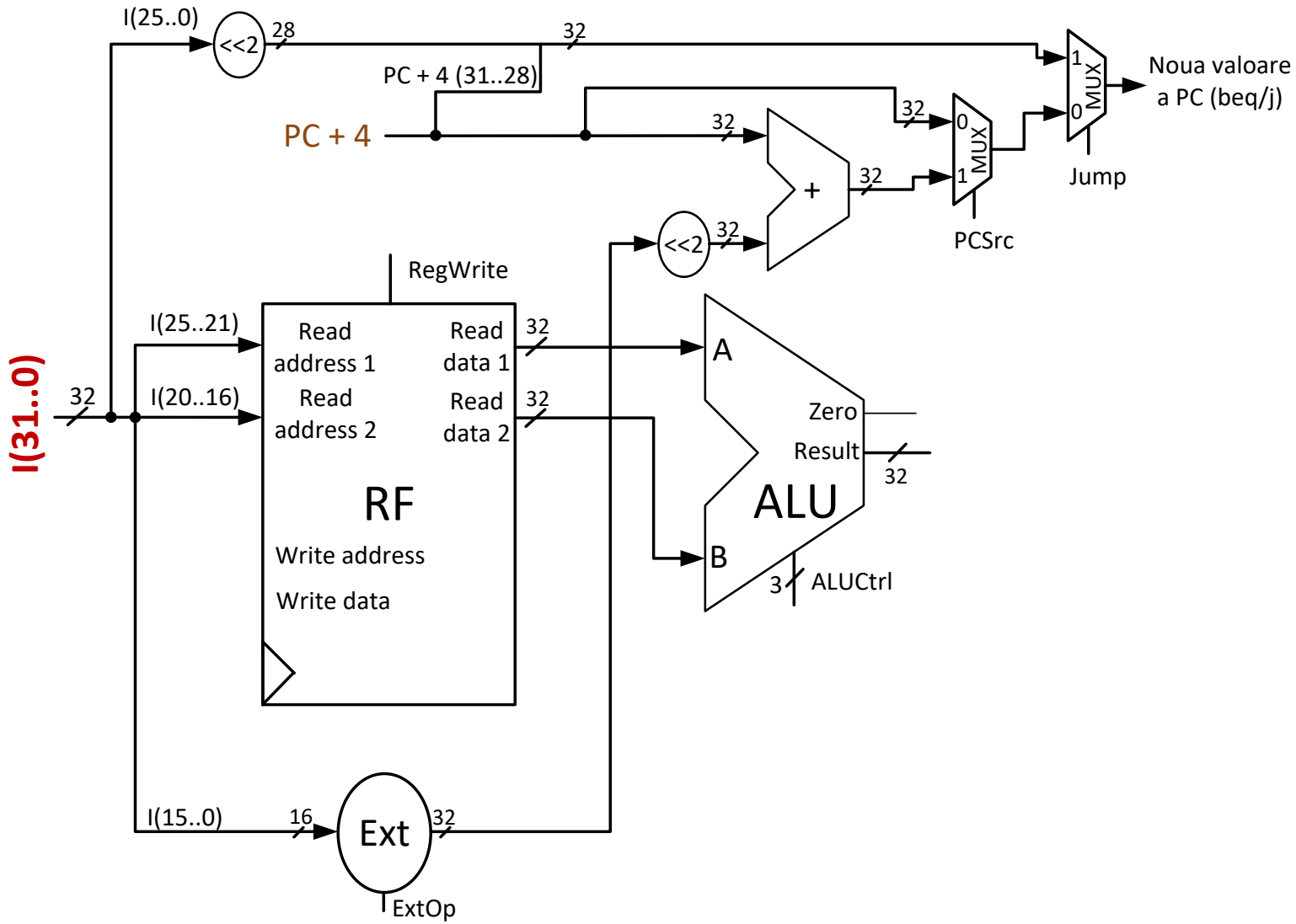
beq: If(RF[rs] == RF[rt]) then PC ← PC + 4 + S_Ext(imm) <<2
else PC ← PC + 4

Proiectarea MIPS ciclu unic – Pas 3



Tip J: j: PC ← (PC+4)[31:28] & target_addr & 00

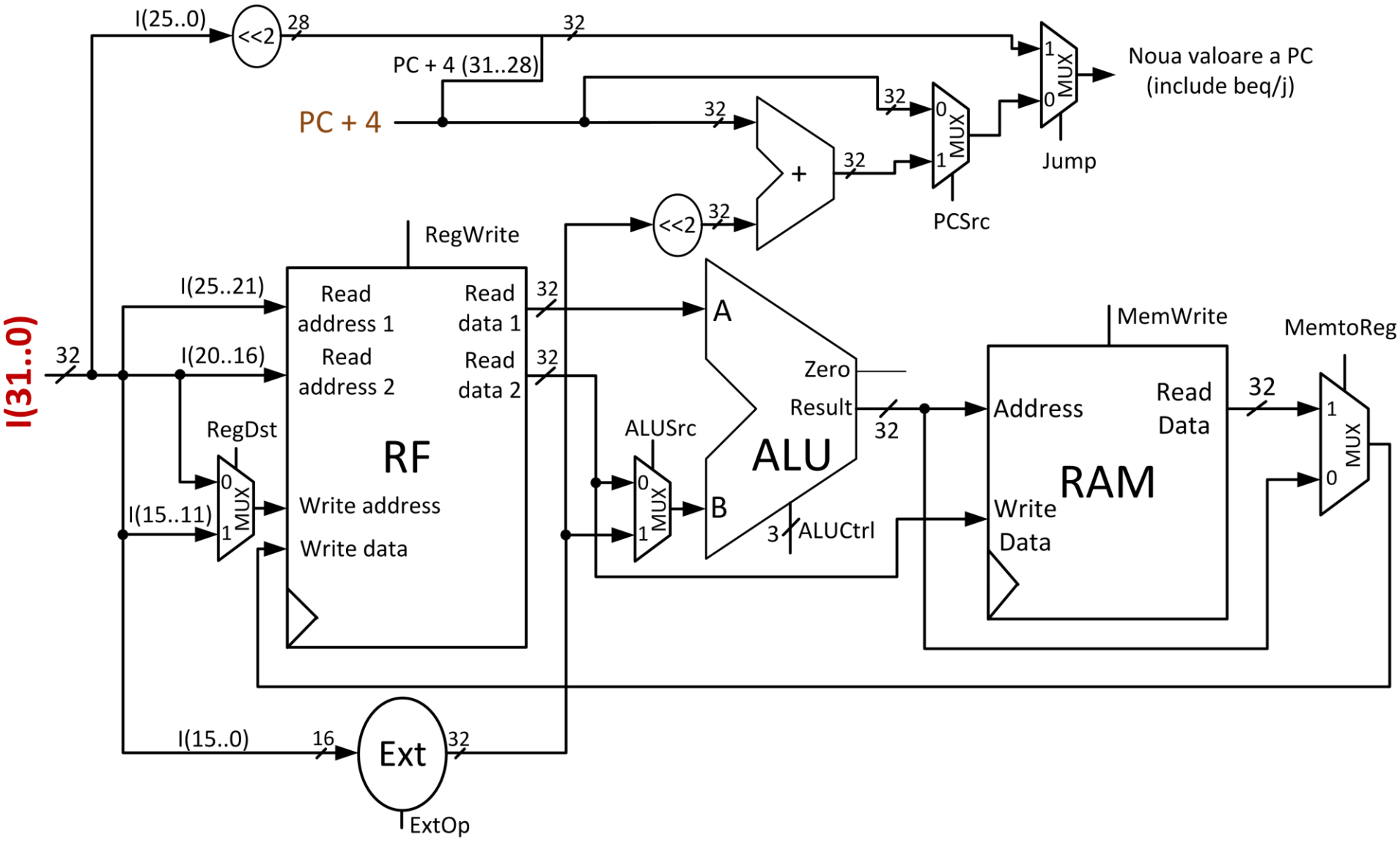
Proiectarea MIPS ciclu unic – Pas 3



Unificare Tip I beq și Tip J, jump

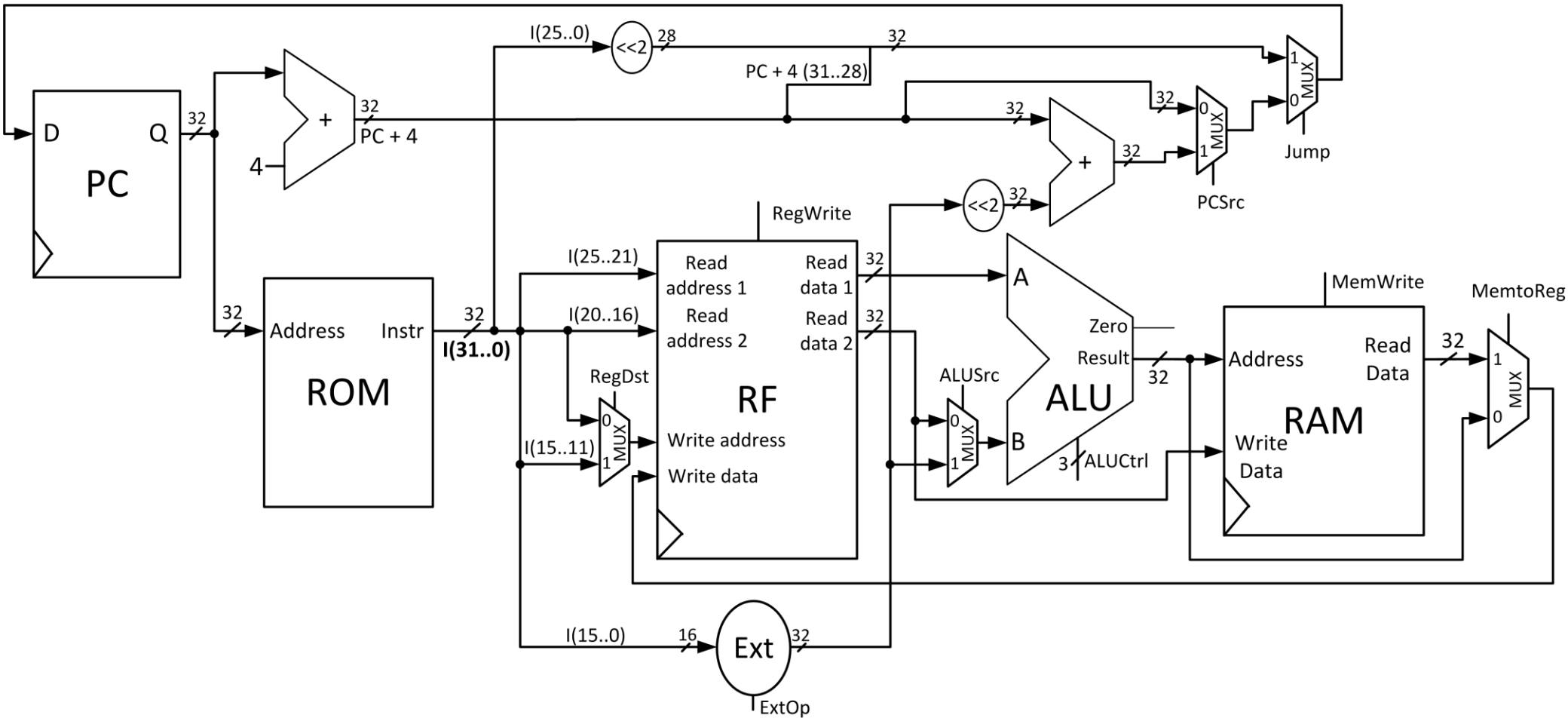
- Se adaugă un nou mux, semnal de control Jump

Proiectarea MIPS ciclu unic – Pas 3



Unificare toate instrucțiunile pentru etapele ID, EX, MEM, WB

Proiectarea MIPS ciclu unic – Pas 3



Unificare toate instrucțiunile etapele ID, EX, MEM, WB cu etapa IF!

Proiectarea MIPS ciclu unic – Pas 4

Pas de proiectare 4: Specificarea comenzii

- Se identifică și se definesc semnalele de comandă necesare pentru funcționarea componentelor și transferul datelor
- Pentru fiecare instrucțiune se stabilesc valorile semnalelor de comandă

Proiectarea MIPS ciclu unic – Pas 4

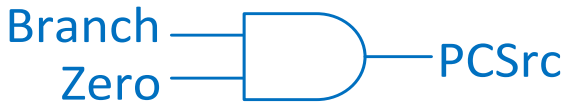
Semnificația semnalelor de control (1):

- RegDst –mux-ul de la adresa de scriere Write address a RF. Dacă are valoarea 0, atunci trece mai departe valoarea câmpului rt. Dacă are valoarea 1, atunci trece rd
- RegWrite – validarea scrierii în RF. Dacă are valoarea 0, nu se execută scriere, altfel, dacă e 1, se scrie în blocul de registre
- ExtOp – controlează funcționarea unității Ext. Dacă are valoarea 0, extinde fără semn, cu 0. Dacă are valoarea 1, extensie cu semn
- ALUSrc –mux-ul de la intrarea B a ALU. Dacă are valoarea 0, atunci pe intrarea B a ALU ajunge Read data 2. Dacă are valoarea 1, va ajunge ieșirea Ext
- PCSrc – mux-ul de salt condiționat. Dacă are valoarea 0, trece valoarea PC + 4. Altfel, dacă are valoarea 1, trece adresa de salt (pentru *beq*)
- Jump – mux-ul de salt necondiționat. Dacă are valoarea 0, spre intrarea PC trece ieșirea mux-ului precedent (PCSrc). Dacă are valoarea 1, spre intrarea PC trece valoarea adresei absolute de salt (pentru *j*)
- MemWrite – validarea scrierii în memoria RAM de date. Dacă are valoarea 0 nu se execută scriere, altfel, dacă e 1, se scrie în memorie
- MemtoReg – mux-ul prin care se scrie înapoi rezultatul. Dacă e 0, atunci trece rezultatul de la ALU. Dacă e 1, atunci trece valoarea din memoria de date.

Proiectarea MIPS ciclu unic – Pas 4

Semnificația semnalelor de control (2):

- PCSrc – depinde de semnalul de stare Zero generat de ALU, doar pentru instrucțiuunea beq. Se introduce un semnal nou Branch, și se generează PCSrc cu o poartă SI

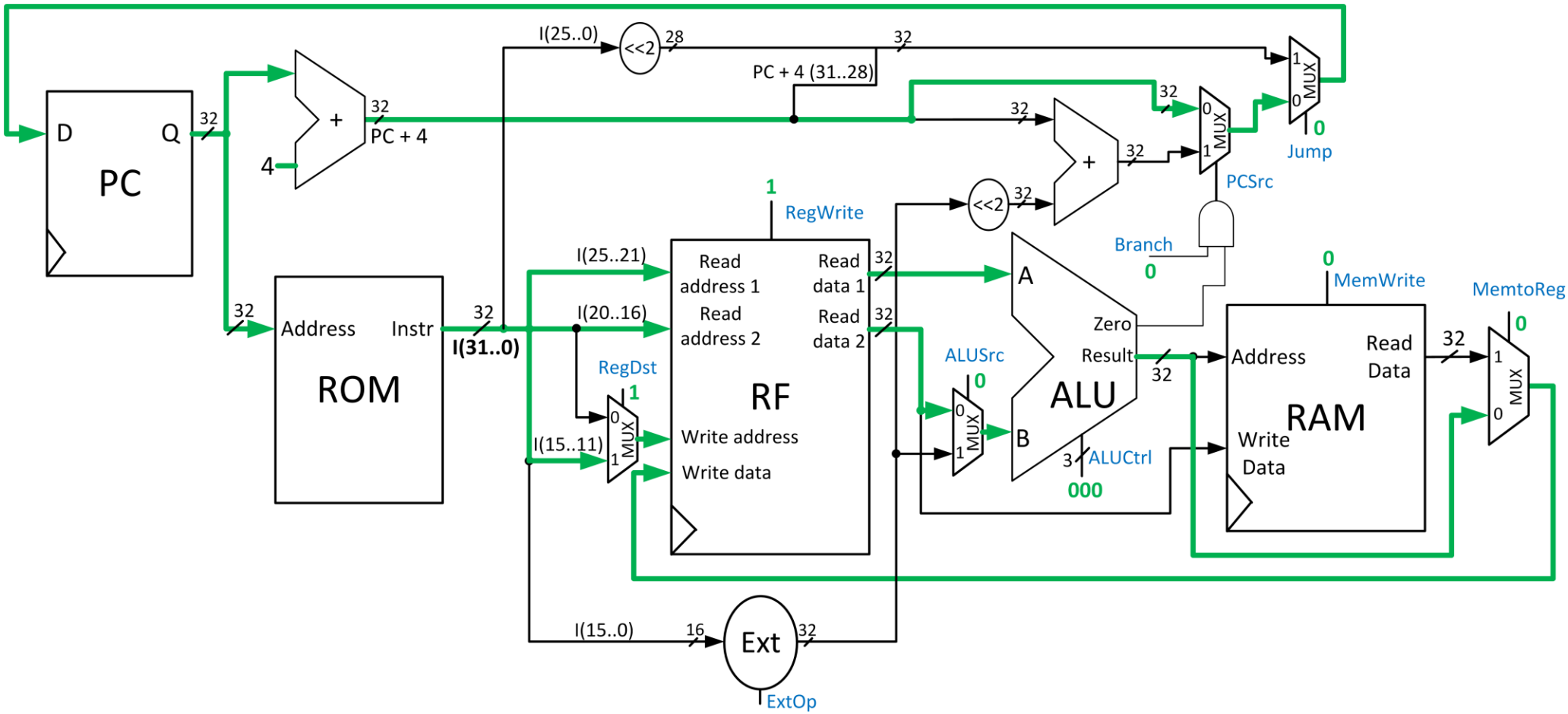


- ALUCtrl – 3 biți, codificat ca în tabelul de mai jos (se va discuta pe larg în cursul următor)

$AluCtrl_{2...0}$	Operație ALU
0 0 0	Adunare
1 0 0	Scădere
0 0 1	ȘI
0 1 0	SAU

Proiectarea MIPS ciclu unic – Pas 4

Stabilirea valorilor pentru semnalele de control se face ținând cont de fluxul de date (RTL)

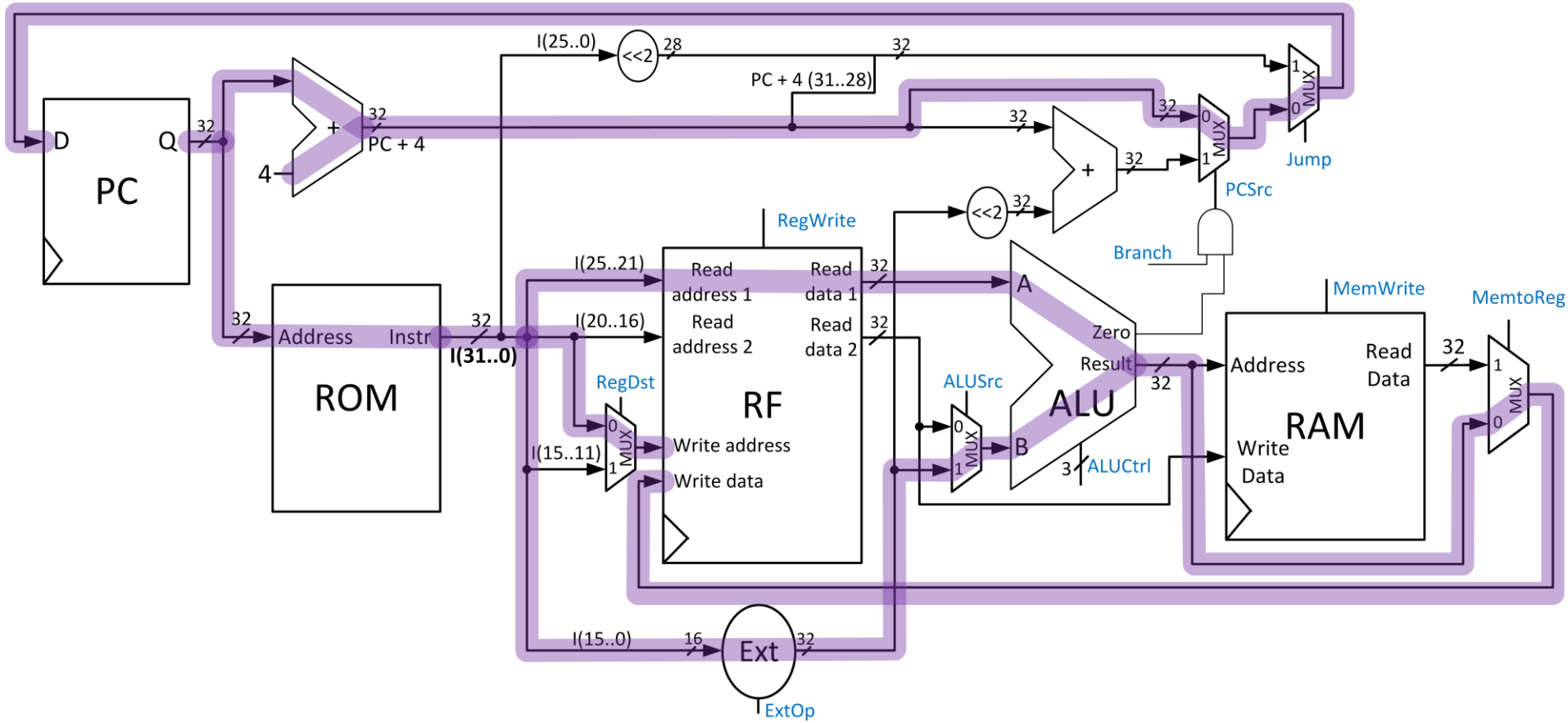


RF[rd] ← RF[rs] + RF[rt], PC ← PC + 4

Instrucțiune	Reg Dst	Reg Write	ALU Src	Ext Op	ALU Ctrl	Mem Write	Memto Reg	Branch	Jump
add (tip R)	1	1	0	x	000 (+)	0	0	0	0

Proiectarea MIPS ciclu unic – Pas 4

Stabilirea valorilor pentru semnalele de control se face ținând cont de fluxul de date (RTL)



RF[rt] ← RF[rs] + S_Ext(imm) sau RF[rt] ← RF[rs] or Z_Ext(imm), PC ← PC + 4

Instrucțiune	Reg Dst	Reg Write	ALU Src	Ext Op	ALU Ctrl	Mem Write	Memto Reg	Branch	Jump
ori	0	1	1	0	010 (sau)	0	0	0	0
addi	0	1	1	1	000 (+)	0	0	0	0

Proiectarea MIPS ciclu unic – Pas 4

Valoarea semnalelor de control pentru setul complet selectat

Instrucțiune	Reg Dst	Reg Write	ALU Src	Ext Op	ALU Ctrl	Mem Write	Memto Reg	Branch	Jump
add (tip R)	1	1	0	x	000 (+)	0	0	0	0
sub (tip R)	1	1	0	x	100 (-)	0	0	0	0
or (tip R)	1	1	0	x	010 (sau)	0	0	0	0
and (tip R)	1	1	0	x	001 (si)	0	0	0	0
ori	0	1	1	0	010 (sau)	0	0	0	0
addi	0	1	1	1	000 (+)	0	0	0	0
lw	0	1	1	1	000 (+)	0	1	0	0
sw	x	0	1	1	000 (+)	1	x	0	0
beq	x	0	0	1	100 (-)	0	x	1	0
j	x	0	x	x	xxx	0	x	x	1

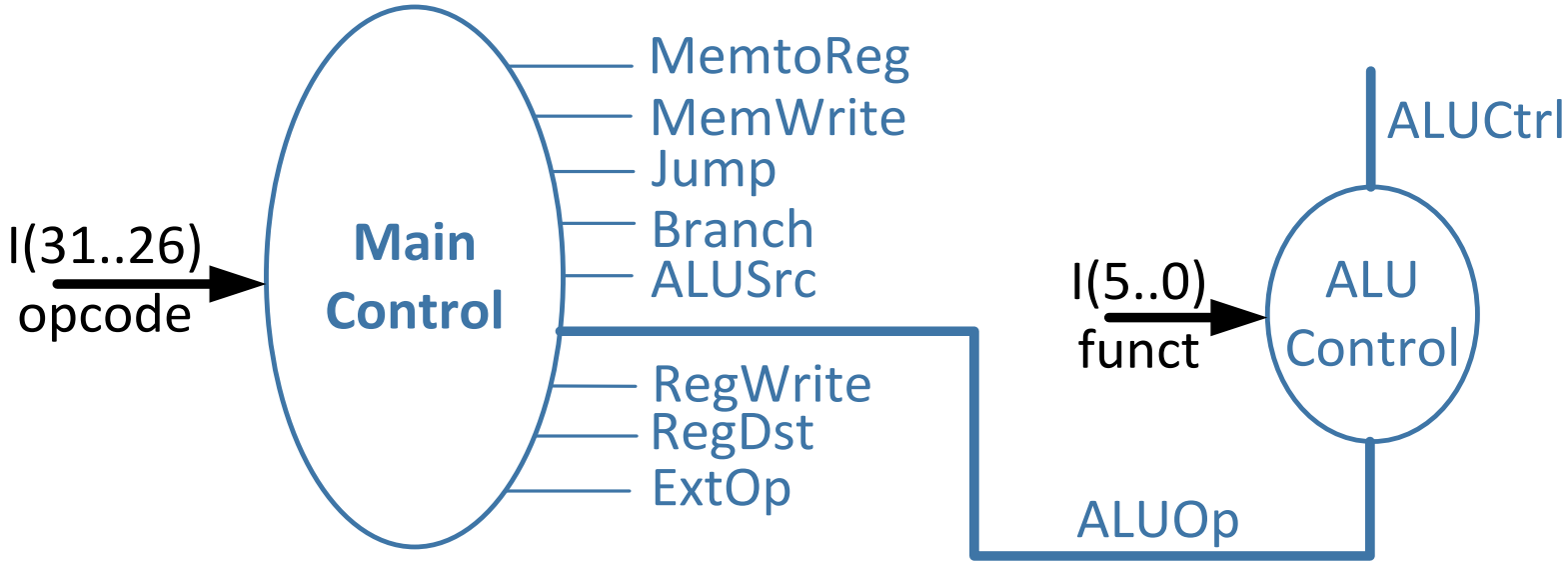
Exercițiu acasă: cu calea de date în față, scrieți valorile semnalelor de control pentru fiecare instrucțiune

Proiectarea MIPS ciclu unic – Pas 5

Unitatea de control este:

1. Combinațională (…e procesor cu ciclu unic)

2. Are o structură ierarhică Main Control (master) + ALU Control (slave), în principal datorită codificării Tip R (opcode=0 + funct). Semnal intermediar de control ALUOp!



Instrucțiune	ALUOp	Semnificația comenzii pentru ALU Control
tip R	10	folosește funct
ori	11	comandă SAU
addi / lw / sw	00	comandă +
beq	01	comandă –

Proiectarea MIPS ciclu unic – Pas 5

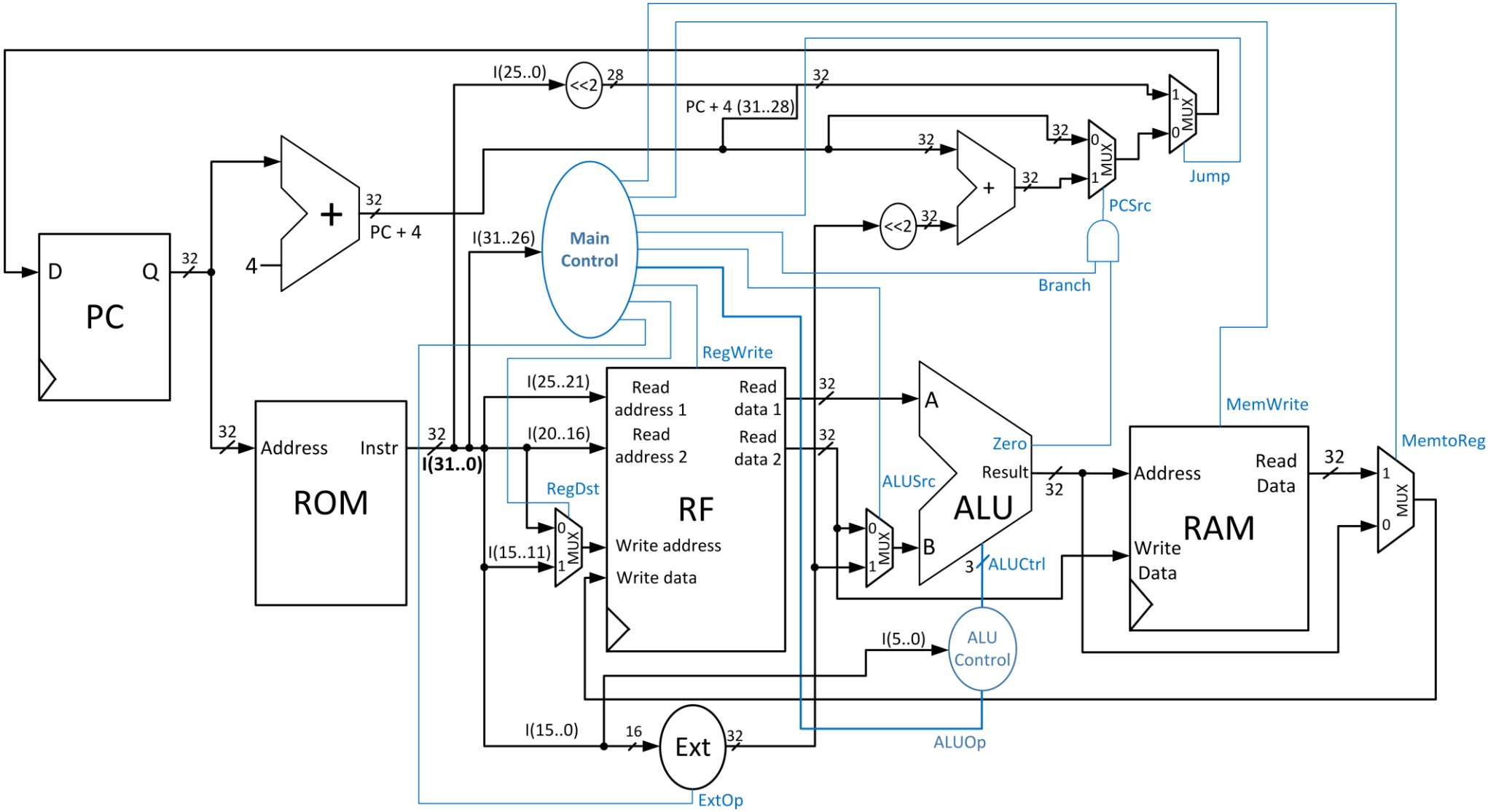
Unitatea Main Control generează valori pentru semnalele următoare:

opcode (instr)	Reg Dst	Reg Write	ALU Src	Ext Op	$ALUOp_{1..0}$	Mem Write	Memto Reg	Branch	Jump
000000 (tip R)	1	1	0	x	10 (funct)	0	0	0	0
001101 (ori)	0	1	1	0	11 (sau)	0	0	0	0
001000 (addi)	0	1	1	1	00 (+)	0	0	0	0
100011 (lw)	0	1	1	1	00 (+)	0	1	0	0
101011 (sw)	x	0	1	1	00 (+)	1	x	0	0
000100 (beq)	x	0	0	1	01 (–)	0	x	1	0
000010 (j)	x	0	x	x	xx	0	x	x	1

Unitatea ALU Control generează valori pentru semnalul ALUCtrl, în funcție de ALUOp și funct:

Instrucțiune	ALUOp	*funct*	ALUCtrl	**operația ALU**
add (tip R)	10	100000	000	adunare
sub (tip R)	10	100010	100	scădere
or (tip R)	10	100101	010	SAU
and (tip R)	10	100100	001	ȘI
ori	11	xxxxxx	010	SAU
addi / lw / sw	00	xxxxxx	000	adunare
beq	01	xxxxxx	100	scădere

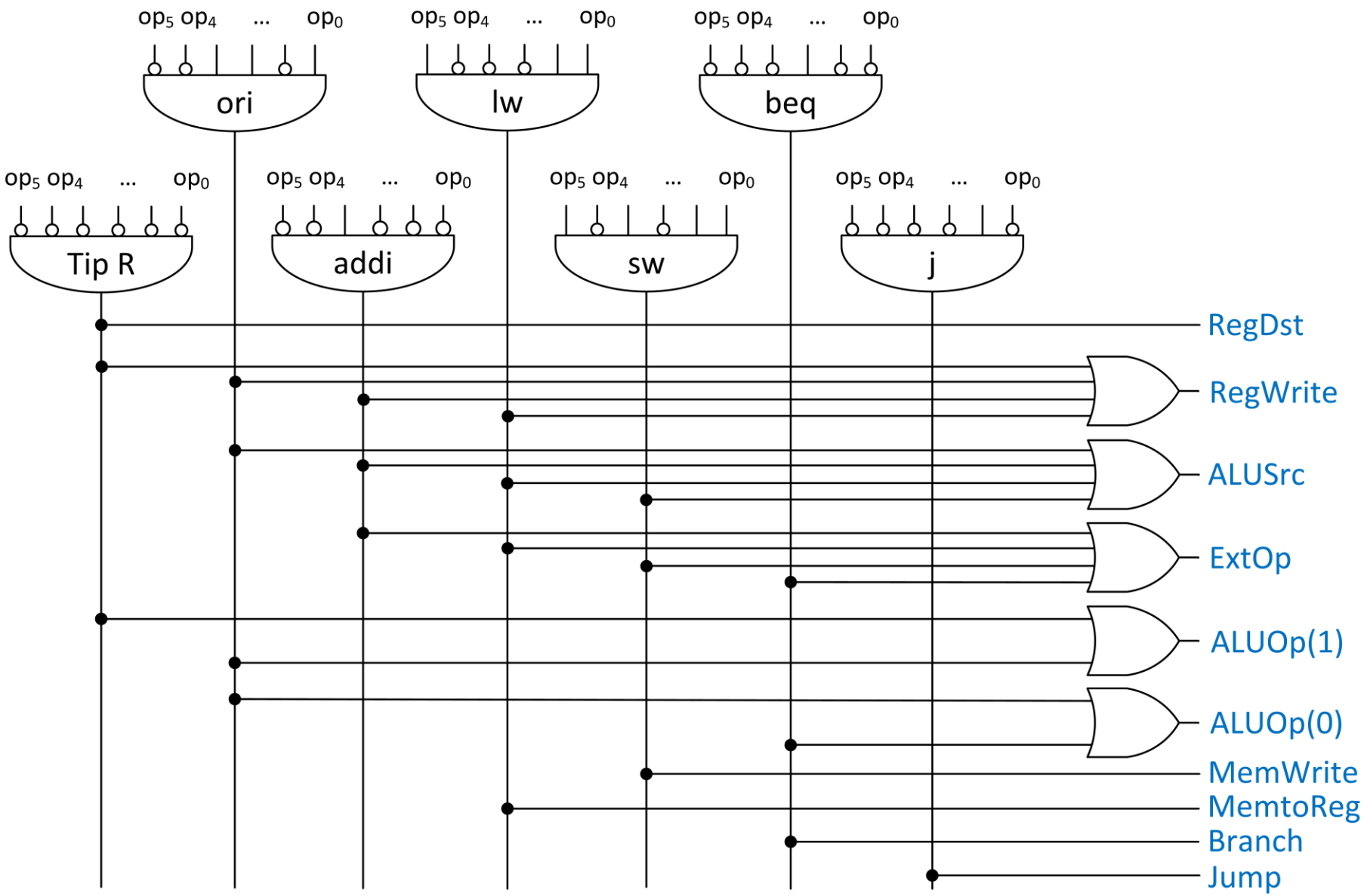
Proiectarea MIPS ciclu unic – Pas 5



Cale de date MIPS + unitate de control, ciclu unic de ceas

Proiectarea MIPS ciclu unic – Pas 5

Pas de proiectare 5: Implementarea unității de comandă (PLA)

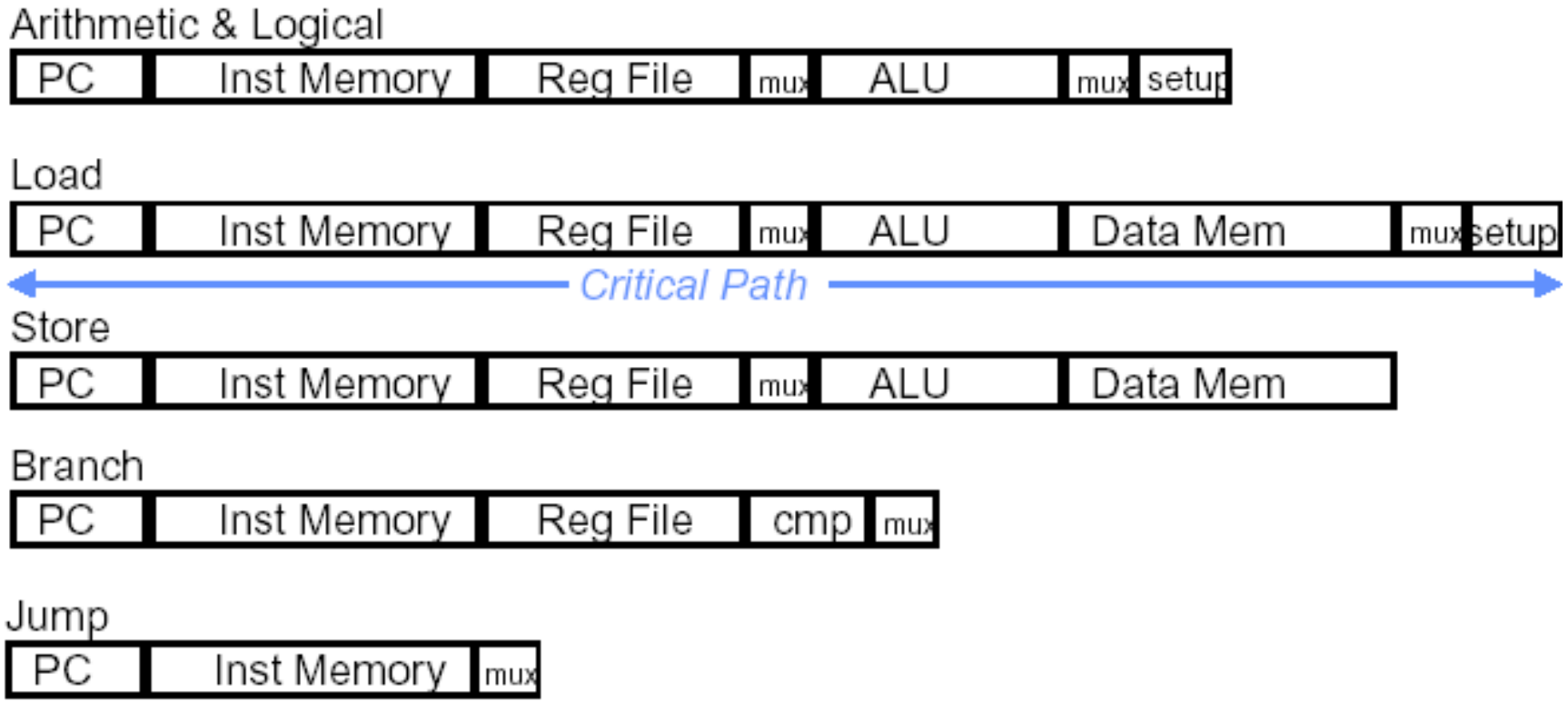


Proiectare UC, pe baza valorilor semnalelor de comandă

Proiectarea MIPS ciclu unic

Perioada de ceas este limitată de calea critică:

- **Calea critică (Load Word)** = PC's Clk-to-Q + Instruction Memory's Access Time + Register File's Access Time + ALU to Perform a 32-bit Add + Data Memory Access Time + Setup Time for Register File Write + Clock Skew

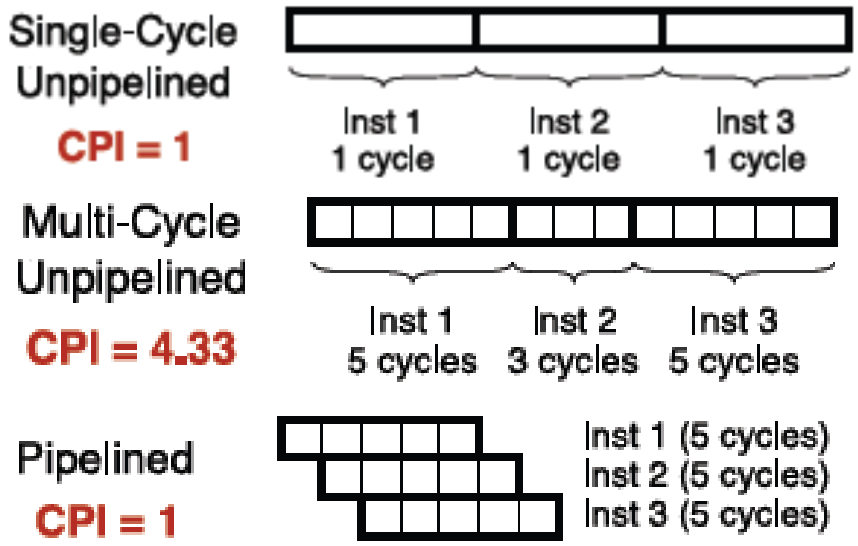


MIPS cu ciclu unic: Compararea duratelor Instrucțiunilor [1]

Proiectarea MIPS ciclu unic

Dezavantajul implementării cu ciclu unic de ceas:

- Durata ciclu ceas: Se alege conform LW => lent
- Durata ciclului pentru Load/Store mult mai mare decât minimul necesar pentru restul instrucțiunilor => timp pierdut prin inactivitate.



CPI **Clocks Per Instruction**

Tip procesor	CPI	Perioadă CLK
Ciclu unic	1	ciclu lung
Multi-ciclu	>1	ciclu scurt
Pipelined	~1	ciclu scurt

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

Performanta procesorului

Proiectarea MIPS ciclu unic - Posibile extensii

Conectare cu dispozitive de intrare / ieșire (I/O)

Problemă: definirea unor porturi pentru comunicare (in, out)

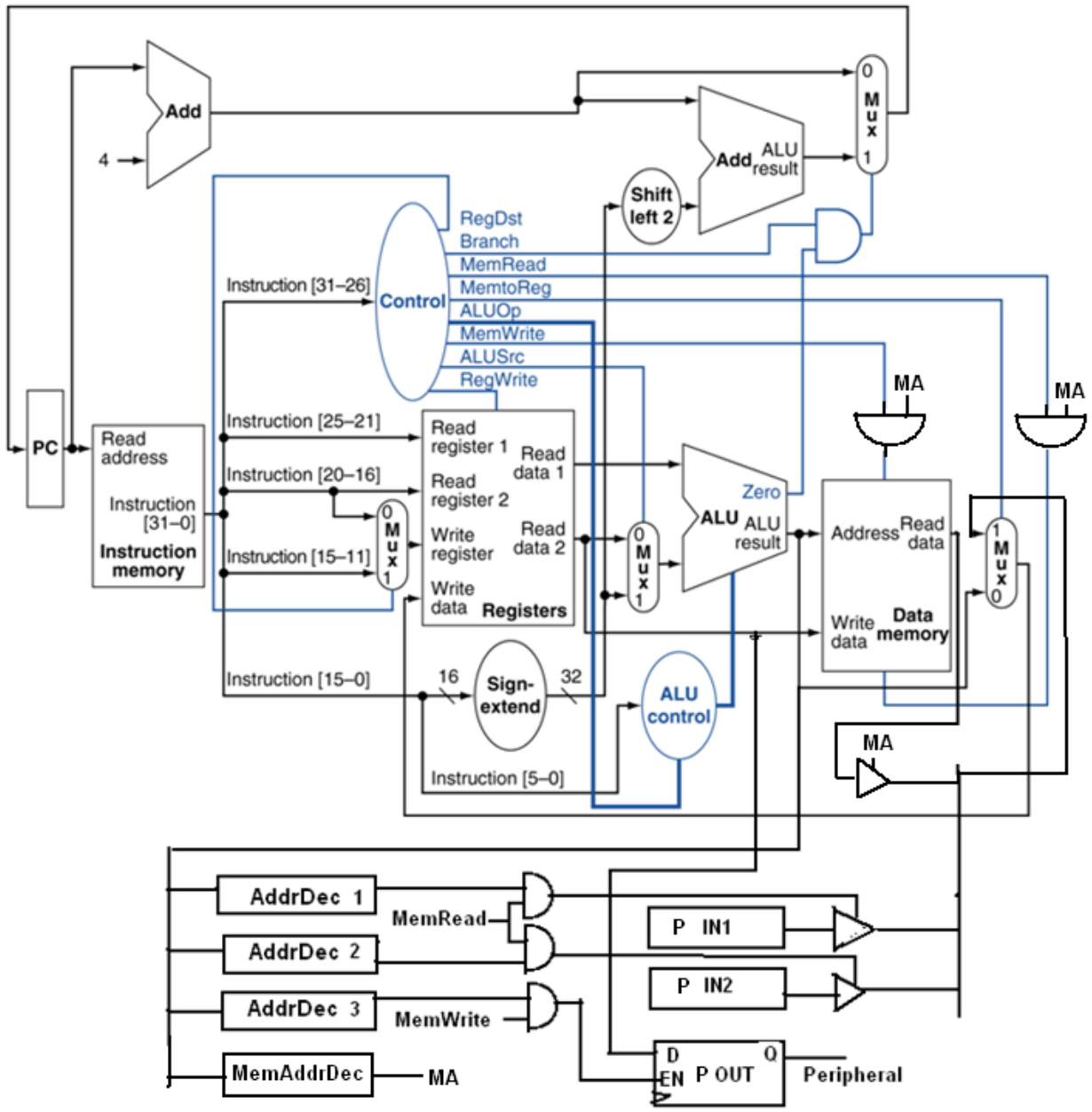
Soluţie: o variantă convenabilă (! fără a introduce instrucțiuni dedicate) este I/O mapat prin spațiul de adresare a memoriei:

- anumite adrese din memorie se rezervă pentru porturi de I/O
- scrierea, respectiv citirea de pe aceste porturi se face folosind instrucţiunile standard de lucru cu memoria
- LW şi SW pentru transfer de cuvânt (cu posibilitatea extinderii cu LH, SH pentru semi-cuvânt, respectiv LB, SB pentru octet)

Pentru proiectare se rescrie descrierea RTL la LW şi SW pentru a trata diferit (cum ?...if…) situațiile când adresa efectivă este în setul de adrese rezervate. De aici rezultă componentele suplimentare necesare (ex. decodificatoare pentru a detecta adresele).

În figura următoare se prezintă căile de date modificate pentru a suporta două porturi de IN şi unul de OUT.

Proiectarea MIPS ciclu unic - Posibile extensii



Conectare cu dispozitive de I/O

Dispozitivele sunt mapate prin spaţiul de adresare a memoriei:

- Cele trei adrese sunt „interceptate”:
  - 2 IN
  - 1 OUT
- Semnale de control:
  - MemRead
  - MemWrite
  - MA (în plus)

Proiectarea MIPS ciclu unic – Mecanism întreruperi

…vezi cursul 3, descrierea din ISA a mecanismului de întrerupere…
…căile de date se extind în mod corespunzător…

Proiectarea MIPS ciclu unic

Implementarea altor instrucțiuni (urmăriți pașii de proiectare => calea de date plus controlul pentru instrucțiunile implementate):

- Instrucțiuni tipice: add, sub, and, or, lw, sw, beq, bne, j, addi, andi, ori, sll, srl, sra
- Alte instrucțiuni (unele sunt atipice!, necesită modificări ale unor componente de bază):
  - BLTZ
  - BNE
  - JAL
  - JAL: Indiciu – realizați o conexiune / cale de la ieșirea sumatorului PC+4 la intrarea de date a blocului de registre RF, iar pe Write address a RF puneți 31 (registrul unde se va salva PC+4, vezi JAL) prin adăugarea unei intrări la MUX-ul legat pe Write address
  - JR Jump register
  - JM (jump memory) – formatul este similar cu LW, diferența constând în nefolosirea câmpului rt, astfel conținutul memoriei se va scrie în PC
  - SWAP two registers
  - Lwnew - adresare a memoriei (ca LW), care însumează două registre pentru a obține adresa efectivă de unde se ia un cuvânt din memoria de date. Format de tip R.
  - Load/Store – ca LW / SW, fără offset / imm, doar adresa dată de registru
  - ADD3, operație aritmetică cu 4 operanzi care adună 3 numere în loc de două: add3 $t5, $t6, $t7, $t8 => $t5 = $t6 + $t7 + $t8
  - Arithmetic with memory operand
    - addm $t2, 100($t3) => $t2 = $t2 + M[$t3+100]
  - WAI (where am I), pune locația instrucțiunii, reprezentată de valoarea lui PC, într-un registru specificat de câmpul rt în biții instrucțiunii. Variație: salvarea lui PC+4, în loc de PC

Referințe

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013.
2. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011.
3. MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
4. MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.
5. Material obligatoriu (facultativ?), vezi site