Arhitectura Calculatoarelor

Curs 6: Proiectarea MIPS multi-ciclu 1 - Calea de date

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

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

Proiectarea MIPS multi-ciclu

- Proiectarea în detaliu pentru procesorul MIPS multi-ciclu, folosind multiplexoare, partea 1: calea de date, partea 2: unitatea de control (se face în cursul următor)

Proiectarea MIPS multi-ciclu

Proiectarea procesorului pas cu pas → MIPS multi-ciclu (Multi cycle)

- Pas 1: ISA → RTL Abstract
- Pas 2: Componentele căilor de date
- Pas 3: RTL + Componente → Căi de date
- Pas 4: Căi de date + RTL Abstract → RTL Concret
- Pas 5: RTL Concret → Comandă / Control

Problemele MIPS cu ciclu unic

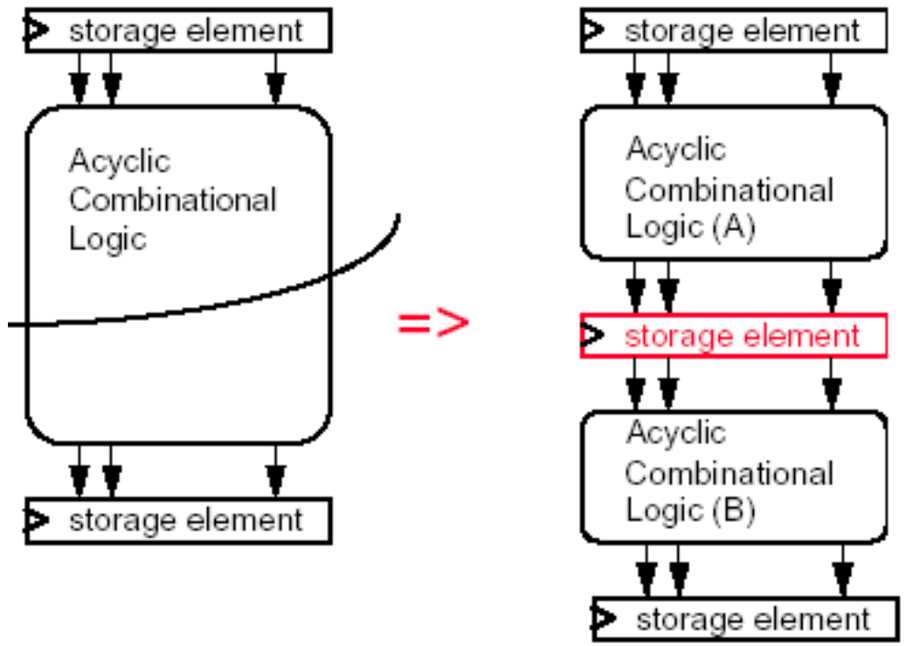
- Durata perioadei de ceas: mare
  - Toate instrucțiunile durează cat cea mai lentă instrucțiune
  - Ce se întâmplă dacă avem instrucţiuni complicate de ex. virgulă mobilă?
- Exploatarea spațiului pe chip: nu se refolosesc componentele

O soluție posibilă

- Folosirea unei perioade de ceas mai redusă
- Instrucţiunile diferite vor avea durate de execuţie diferite
- Rezultă căi de date multi-ciclu

Reducerea duratei ciclului de ceas

- Secționarea căilor combinaționale prin introducerea de registre între componentele rezultate
- Executarea aceleiași sarcini în mai multe cicluri rapide, nu într-un ciclu lent

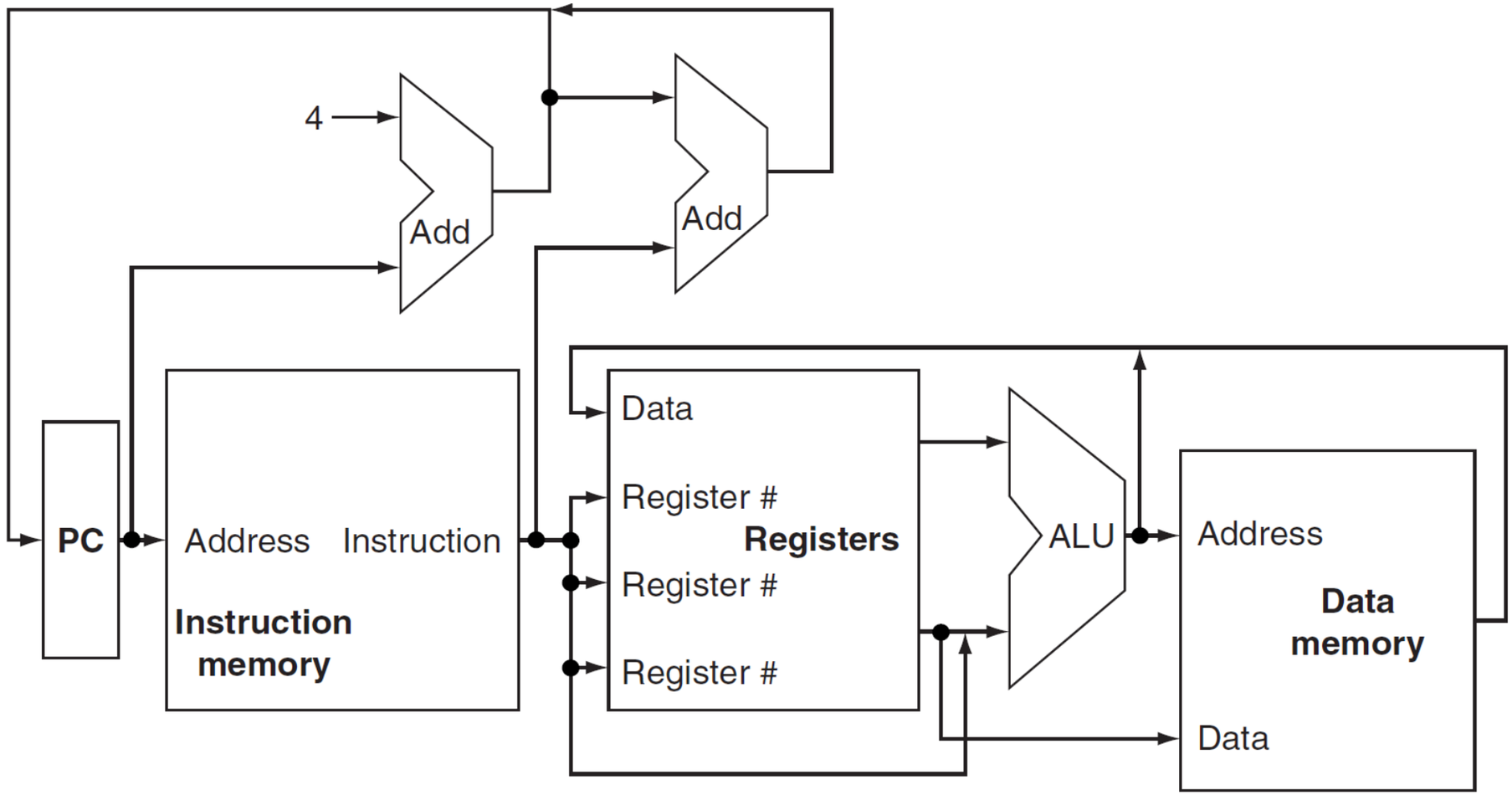


Secţionarea etajelor combinationale lungi

Limitarea duratei ciclului de ceas în etaje diferite MIPS (exemplu):

Logica adresei următoare	PC ← branch ? PC + offset: PC + 4	Durată calculare adresă
Obținerea Instrucțiunii	… din Mem[PC]	Timp de acces Memorie
Accesul la registre	… din RF[rs]	Timp de acces Bloc Registre
Operația ALU	ieșire ← op1 + op2	Durata operaţiei in ALU

Abordarea cu ciclu unic – bază de pornire



[1]

Căile de date MIPS cu ciclu unic, (!) reprezentare de nivel înalt

- Unde se pot introduce registre?
- Ce se poate refolosi?
- Perioada de ceas pentru pașii de execuţie trebuie să fie echilibrată…

Abordarea Multi-ciclu

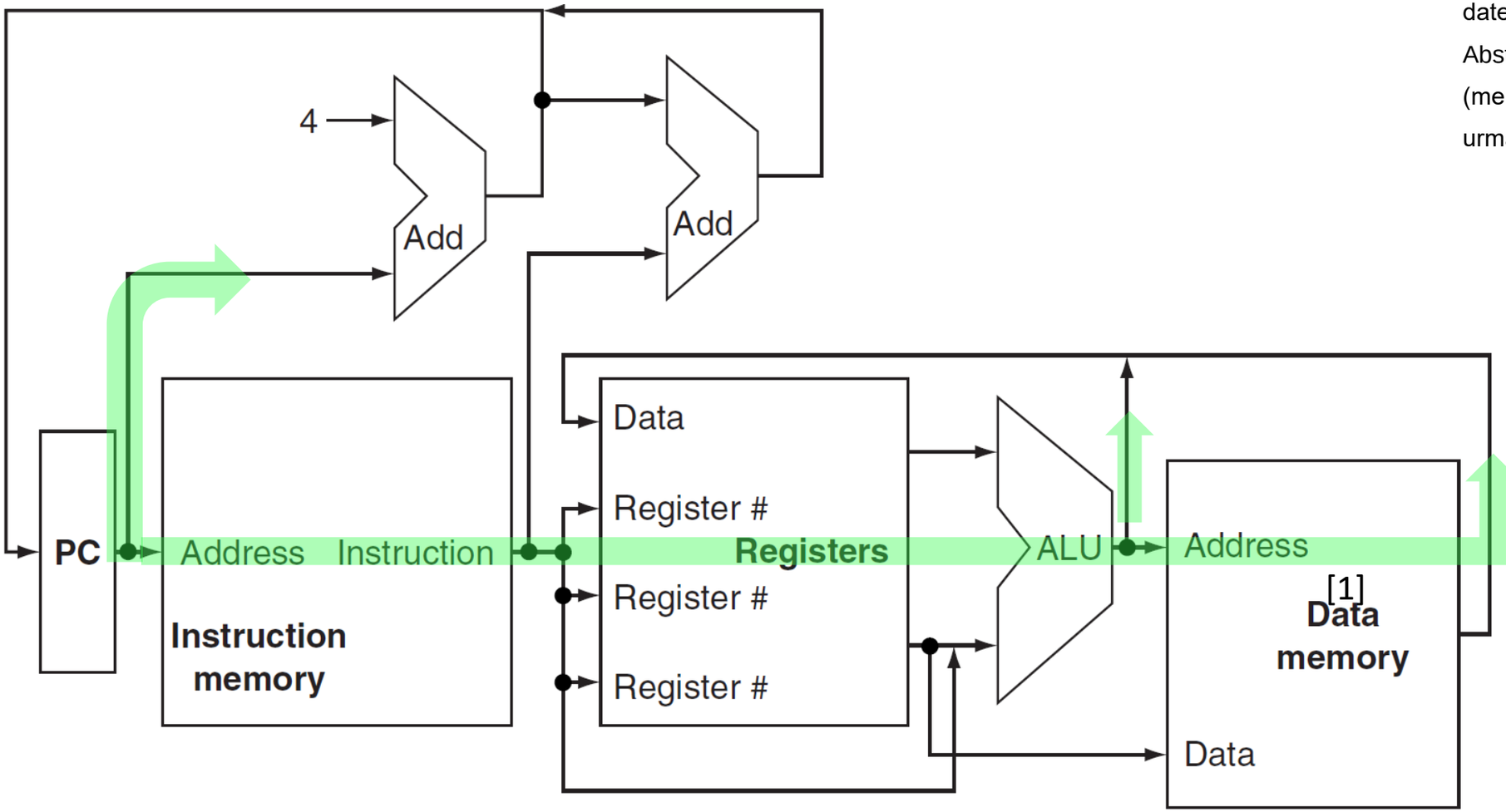
- **Se secționează instrucțiunile în pași simpli de execuție**, fiecare pas ⇔ un ciclu de ceas
  - Se balansează durata sarcinilor în diferite faze
  - Se folosește doar o singură unitate funcțională majoră în fiecare pas.
- **La terminarea unui ciclu**
  - Se memorează rezultatele pasului pentru a fi folosite în ciclurile următoare
  - Se introduc regiștri adiționali “interni” (invizibili pentru programator).
- **Se refolosesc unităţile funcţionale**
  - ALU este folosit pentru calculul adresei și incrementarea PC (pe lângă operațiile ALU uzuale)
  - Aceeaşi memorie se foloseşte pentru instrucţiuni si pentru date
- **Pentru control se va folosi o maşina cu stări finite (FSM)**

(!) Descrierea care urmează în curs este prezentată în detaliu în [1].

Proiectarea MIPS Multi-ciclu – Pas 1, 2

- **Pas 1**: ISA → RTL Abstract - Același subset de instrucțiuni ca pentru MIPS cu ciclu unic
- **Pas 2**: Componentele căilor de date
  - **Simplificare***: partiționăm căile de date deja obținute la MIPS cu ciclu unic

(*) Fără aceasta simplificare, căile de date se obțin la pas 3 din analiza RTL Abstract + constrângerile impuse (memorie comună, logica adresei următoare prin ALU…)

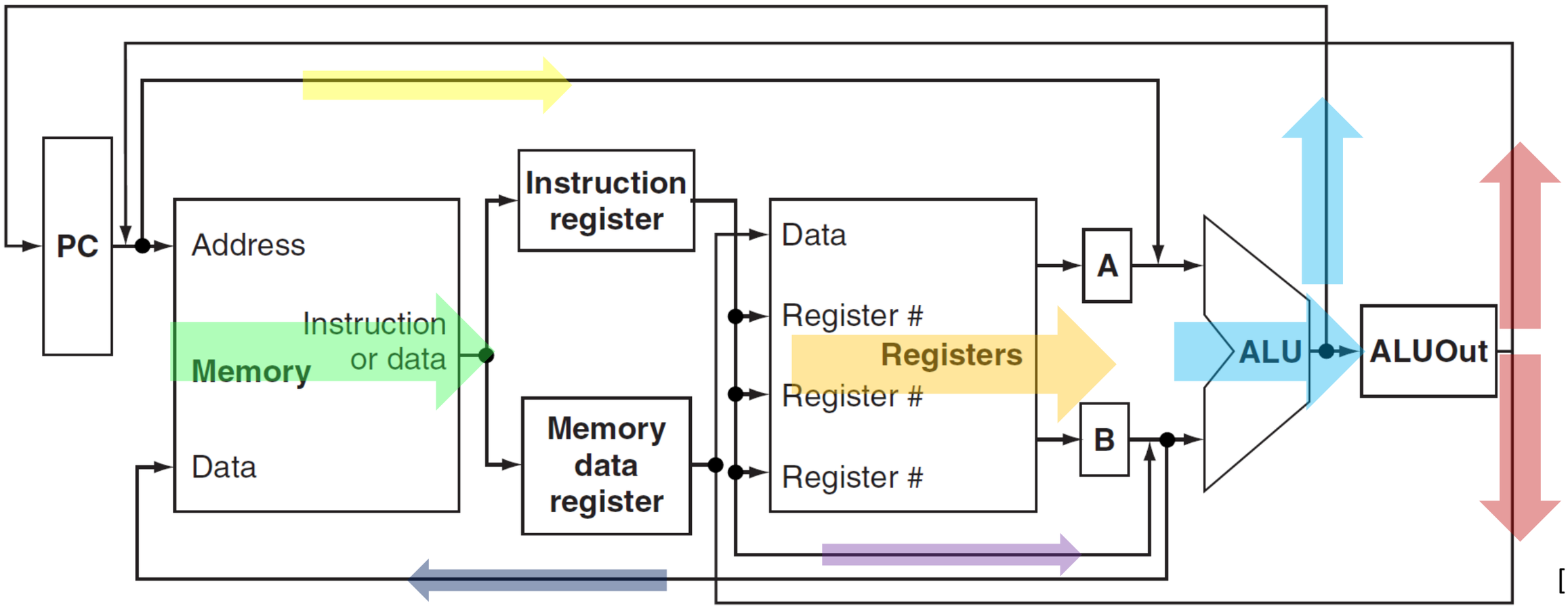


Căile de date MIPS cu ciclu unic, (!) reprezentare de nivel înalt

[1]

Proiectarea MIPS Multi-ciclu – Pas 2

➢ **Pas 2**: Componentele căilor de date – obținute prin partiționare



[1]

Căi de date MIPS Multi-ciclu (Reprezentare de nivel înalt)

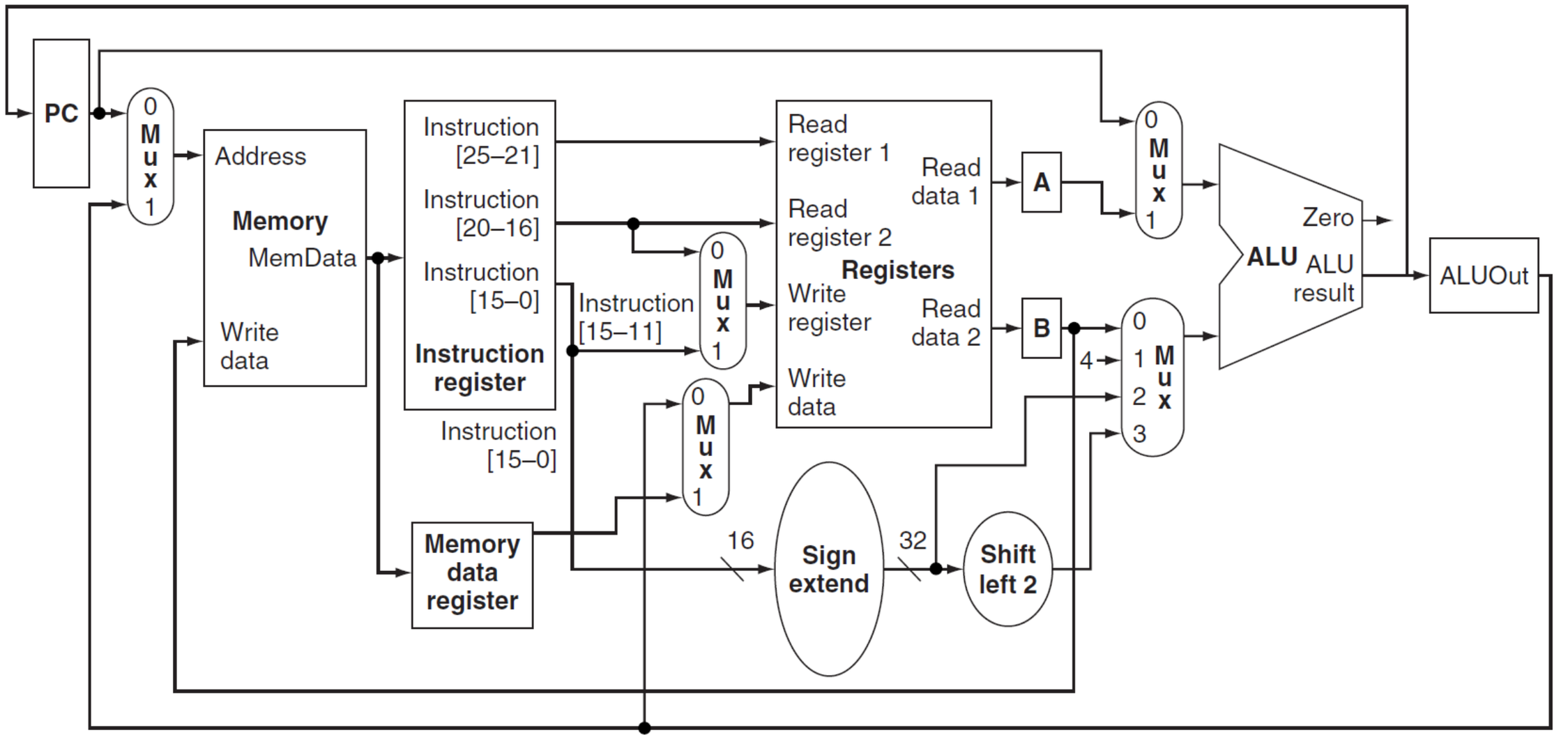
➢ **Au fost adăugate registre invizibile pentru programator, Memoria și ALU sunt refolosite**

IR – Instruction Register; MDR – Memory Data Register; A, B – registre de citire din RF; ALUOut – ALU output register.

Proiectarea MIPS Multi-ciclu – Pas 2/3

[1]

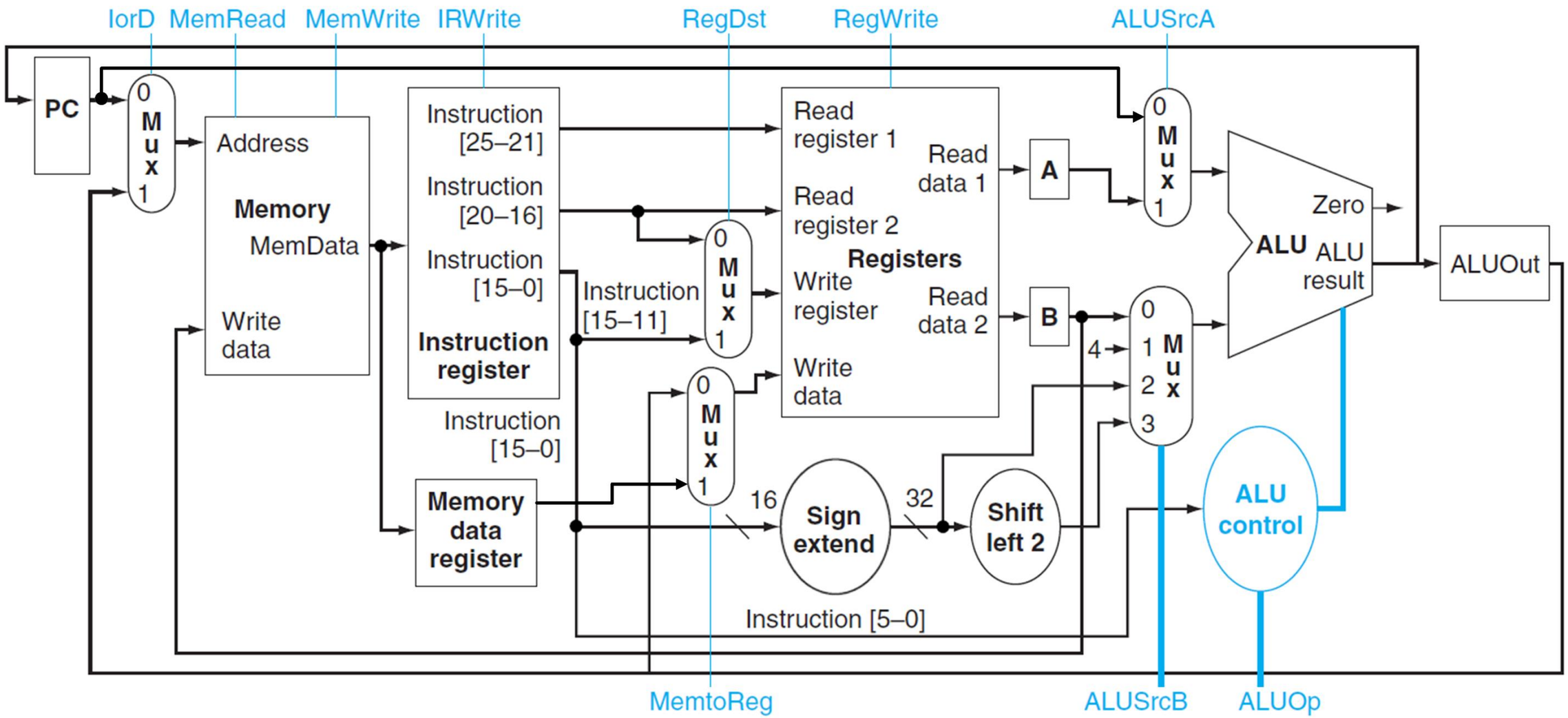
Componentele căilor de date MIPS Multi-ciclu, bazat pe multiplexoare, în detaliu



- Datele folosite de instrucţiunile următoare sunt depozitate în registre vizibile de către programator (ex, Bloc de Registre, PC, sau Memoria). …la fel ca la versiunea ciclu unic

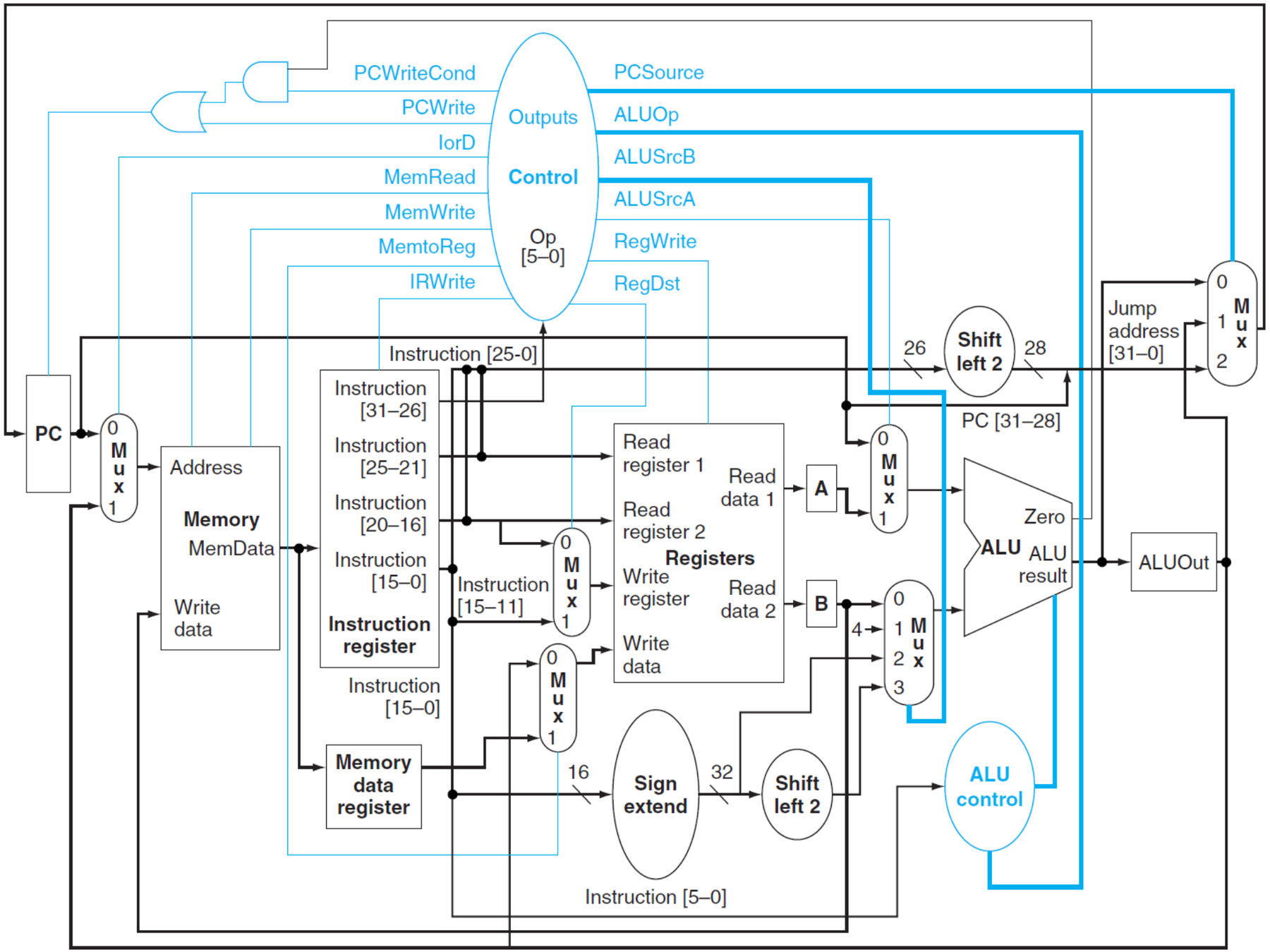
Proiectarea MIPS Multi-ciclu – Pas 3

Pas 3: RTL + Componente → Căi de date MIPS Multi-ciclu, în detaliu, cu semnalele de control



– Instruction Register (IR): IR[25:21] → rs; IR[20:16] → rt; IR[15:11] → rd (notaţii)

Pas 3 - Cale de date MIPS Multi-ciclu, cu Unitatea de Control



Proiectarea MIPS Multi-ciclu – Pas 3

Nume semnal	Efect inactiv (=0)	Efect activ (=1)
RegDst	Adresa de scriere (destinaţie) vine de la câmpul rt (biţii de instrucţiune 20:16)	Adresa de scriere (destinaţie) vine de la câmpul rd (biţii de instrucţiune 15:11)
RegWrite	Nu are	Validează scrierea valorii de la intrarea Write data in registrul adresat de Write register
ALUSrcA	Selectează PC ca operand ALU	Selectează registrul A (R[rs]) ca operand ALU
MemRead	Nu are	Conţinutul celulei de memorie specificata de intrarea Address apare pe ieşirea MemData
MemWrite	Nu are	Conţinutul celulei de memorie specificată de intrarea Address este înlocuit de valoarea de pe intrarea Write data
MemtoReg	Valoarea pentru Write data provine de la registrul ALUOut	Valoarea pentru Write data provine de la Memory data register (MDR)
IorD	Selecteaza PC pentru adresarea memoriei	Selectează registrul ALUOut pentru adresare memoriei
IRWrite	Nu are	Ieşirea memoriei se înregistrează in Instruction Register (IR)
PCWrite	Nu are	Se actualizează PC; sursa este selectata de PC Source
PCWriteCond	Nu are	PC se actualizează daca ieşirea Zero de la ALU este = 1

Semnificația semnalelor de control de 1 bit

Proiectarea MIPS Multi-ciclu – Pas 3

Nume semnal	Valoare(binară)	Efect
ALUOp	00	ALU execută operaţia de adunare*
	01	ALU execută operaţia de scădere
	10	Câmpul function al instrucţiunii determină operaţia ALU (tip R)
	11	ALU execută operația SAU logic
ALUSrcB	00	A doua intrare în ALU este registrul B
	01	A doua intrare în ALU este constanta 4
	10	A doua intrare în ALU este valoarea imm extinsă
	11	A doua intrare în ALU este valoarea imm extinsă, deplasata la stânga 2 biti
PCSource	00	Ieşirea ALU (PC + 4) va actualiza PC
	01	Conţinutul ALUOut (adresa ţintă de ramificare) va actualiza PC
	10	Adresa ţinta pentru Salt (IR[25:0) deplasată la stânga 2 biţi și concatenată cu (PC+4)[31:28] va actualiza PC

Semnificația semnalelor de control de 2 biți

* Obs. addi nu este prezentat explicit în slide-urile care urmează

Proiectarea MIPS Multi-ciclu – Pas 4

- **Pas 4**: Calea de date + RTL Abstract → RTL Concret
  - Pentru calea de date multi-ciclu, scriem transferurile RTL pentru instrucţiunile de bază, pentru definirea valorilor semnalelor de comandă pentru fiecare perioada de ceas
- Instrucțiunile din perspectiva ISA definite prin RTL Abstract
  - Specifică instrucțiunile independent de o implementare concretă
  - Exemplu: instrucțiunile tip R, aritmetice

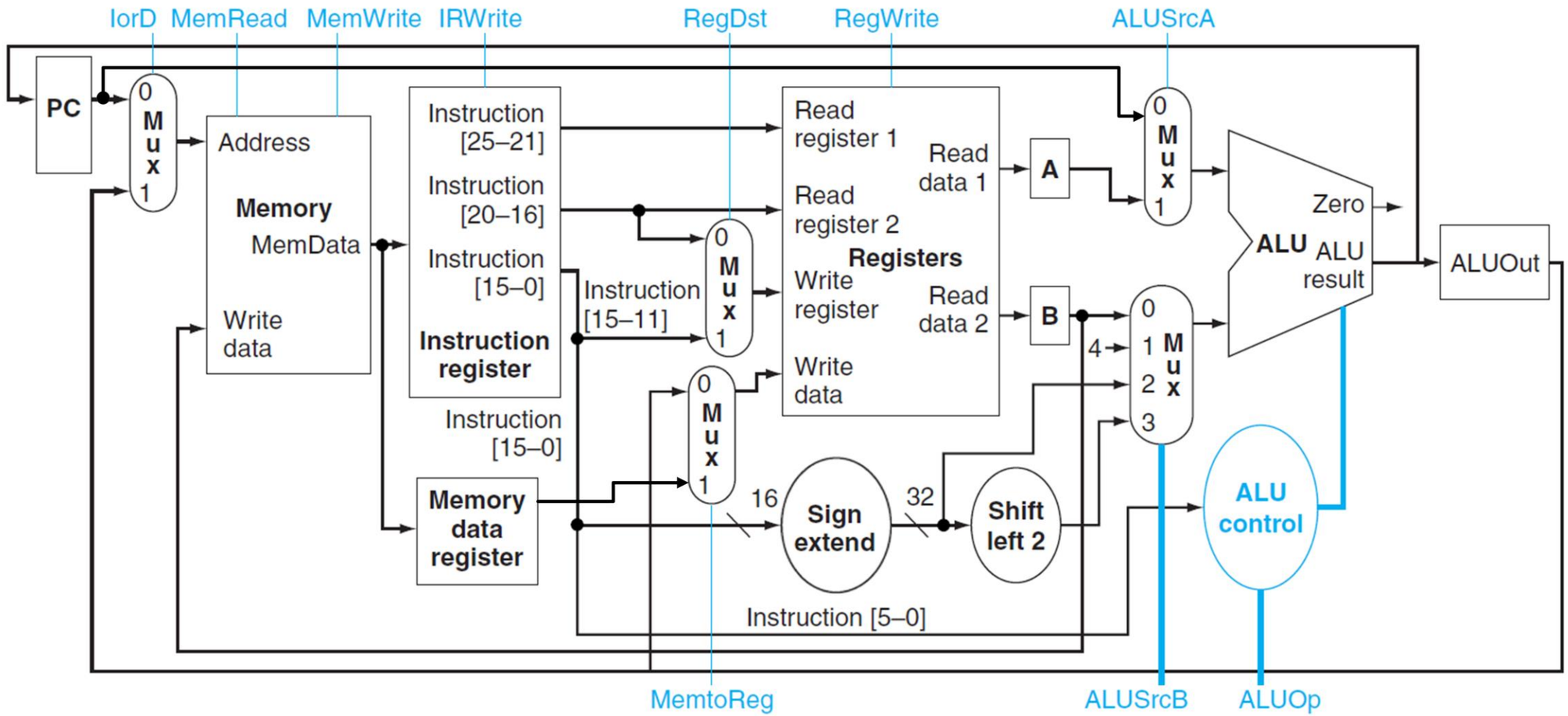
    R[rd] ← R[rs] op R[rt]
- RTL Concret
  - Descrie pașii de execuție a instrucţiunilor (pe fiecare perioadă de ceas)
  - De exemplu: instrucțiune tip R, aritmetică:

    T0 → IR ← M[PC]
    T1 → A ← R[rs], B ← R[rt]
    T2 → ALUOut ← A op B
    T3 → R[rd] ← ALUOut
  - Nu am precizat o parte importantă din instrucțiune!
    - PC ← PC + 4

Obs. Din motive de spațiu (pe diagramă) semnalele PCSrc, PCWrite și PCWriteCond nu se vor prezenta explicit la add, ori, lw, sw. Valorile lor se vor specifica complet spre final (pag 32).

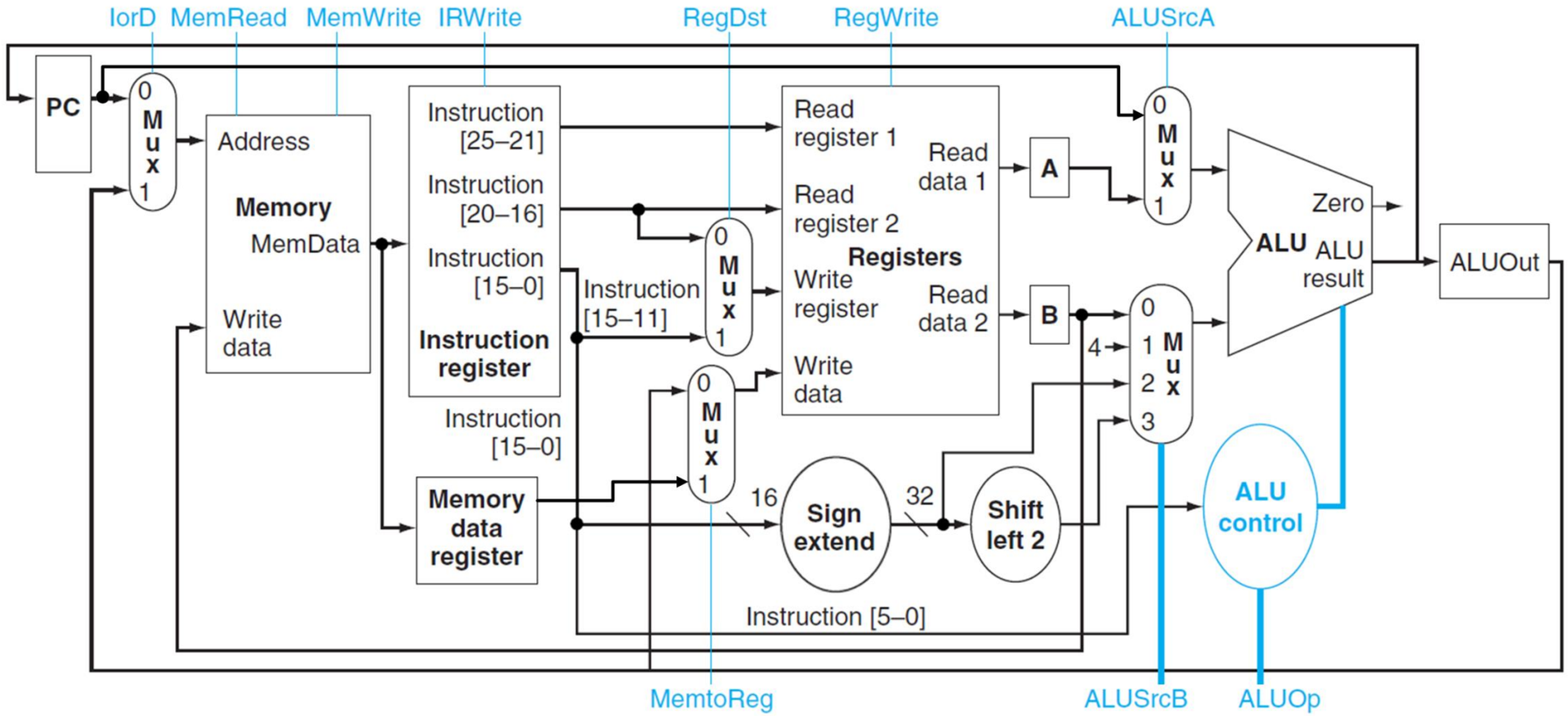
Proiectarea MIPS Multi-ciclu – Pas 4

add $rd,$rs,$rt, **RTL Abstract**: R[rd] ← R[rs] + R[rt], PC ← PC + 4;



Pas (tact)	RTL Concret	Semnale de control (relevante)
T0 →	IR ← M[PC], PC ← PC+4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A ← R[rs], B ← R[rt];	
ADD & T2 →	ALUOut ← A + B;	ALUSrcA, ALUSrcB=0, func
ADD & T3 →	R[rd] ← ALUOut;	RegDst, /MemtoReg,RegWrite

Proiectarea MIPS Multi-ciclu – Pas 4

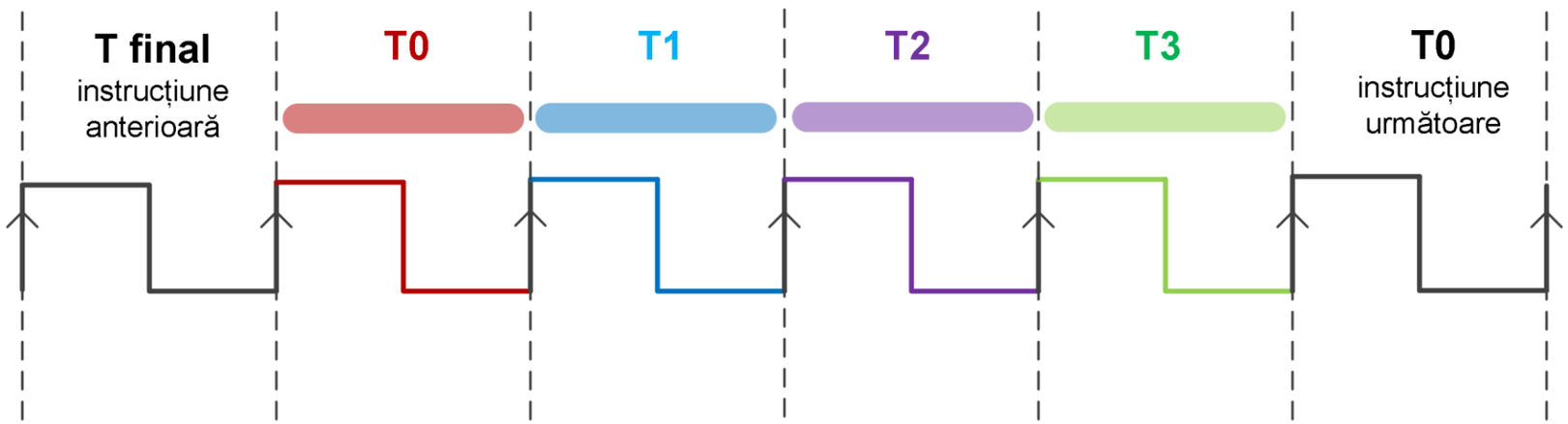


Add - Alternativă la specificarea semnalelor de control (specificare completă)

Pas	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	x	1	0	**func**
T3	x	0	0	0	1	0	1	x	x	x	x

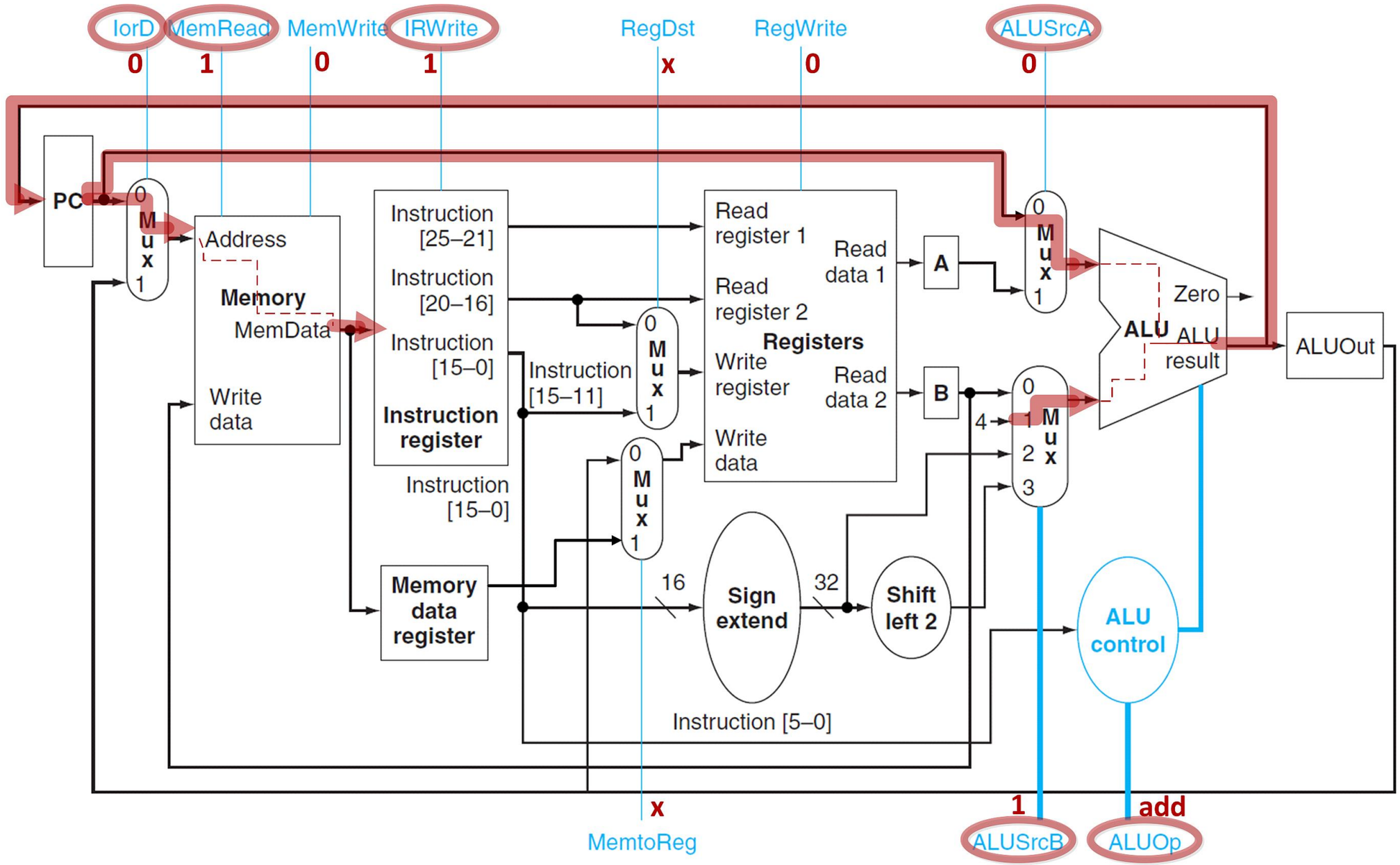
Proiectarea MIPS Multi-ciclu – Pas 4

- **Add:** trasare în detaliu a transferurilor RTL concret pe fiecare pas
- Pentru fiecare transfer RTL concret din pas T_x, scrierea efectivă în destinație se face pe frontul crescător de la finalul lui T_x (valorile noi vor fi disponibile pe ieșirea componentelor sincrone la începutul lui T_{x+1})
- Transferul (cu diferite transformări/selecția unor locații) combinațional din interiorul unităților mari va fi desenat punctat

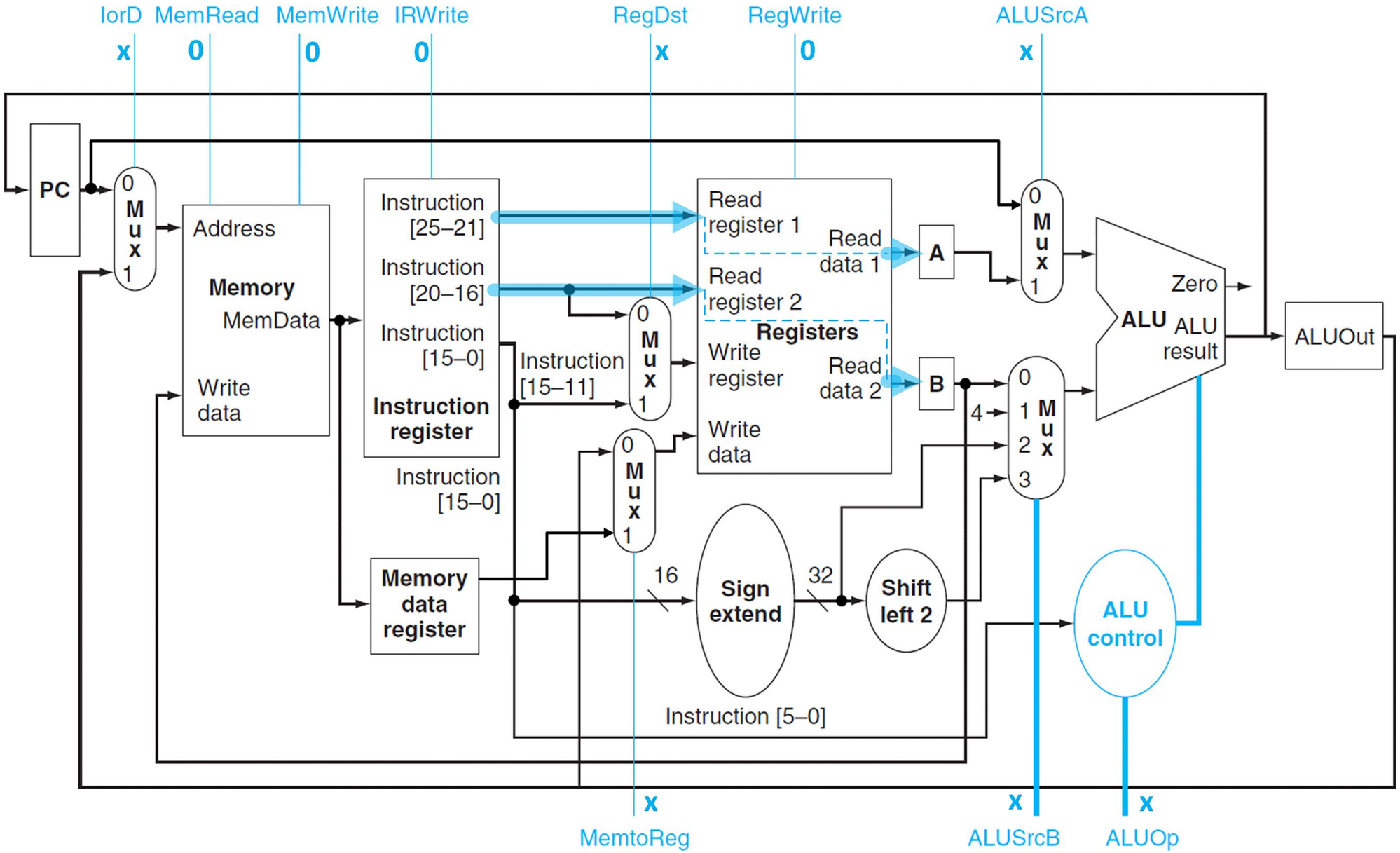


Pas (tact)	RTL Concret	Semnale de control (relevante)
T0 →	IR ← M[PC], PC ← PC+4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A ← R[rs], B ← R[rt];	
ADD & T2 →	**ALUOut ← A + B;**	ALUSrcA, ALUSrcB=0, func
ADD & T3 →	**R[rd] ← ALUOut;**	RegDst, /MemtoReg,RegWrite

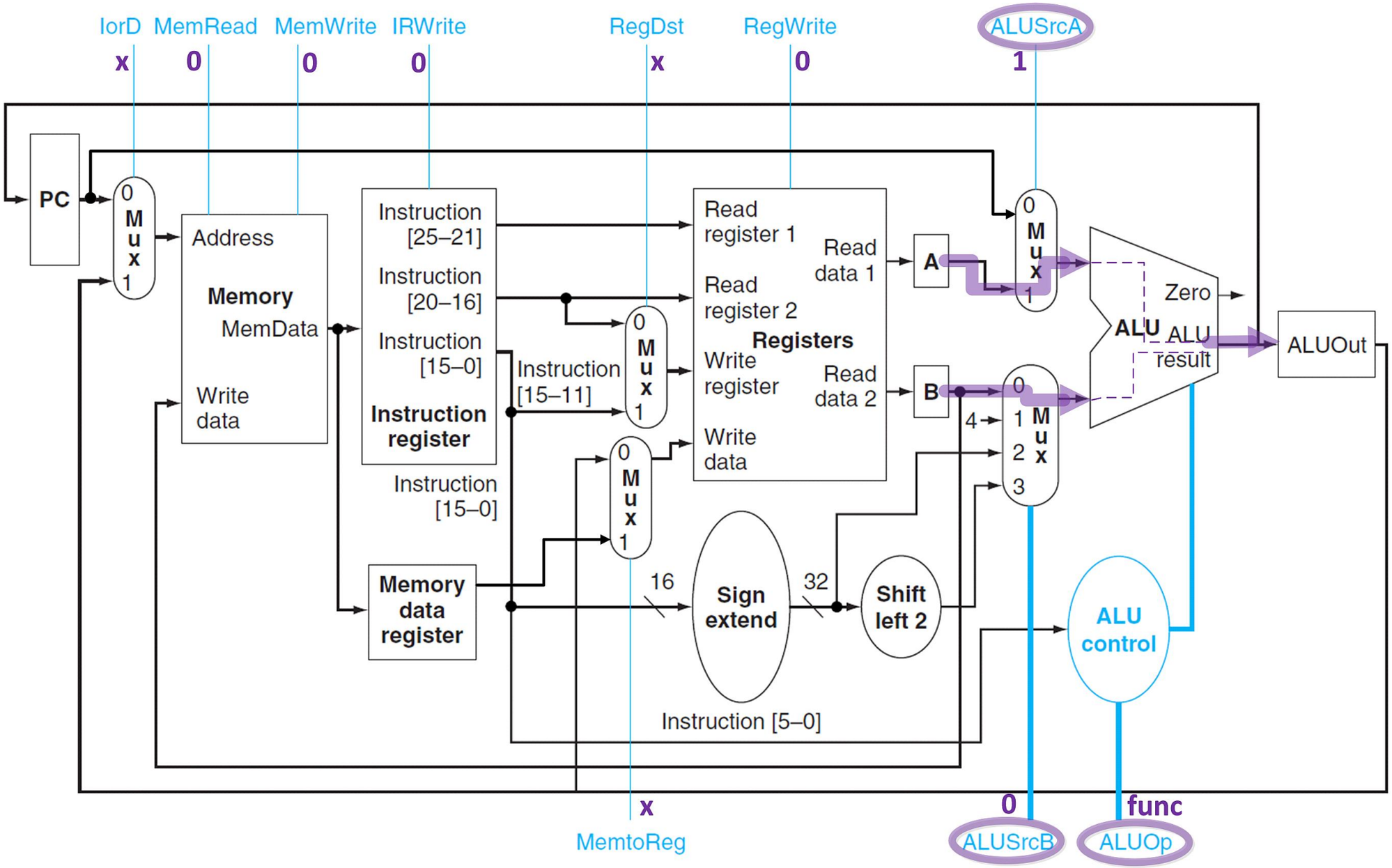
Add: T0 → IR ← M[PC], PC ← PC+4;



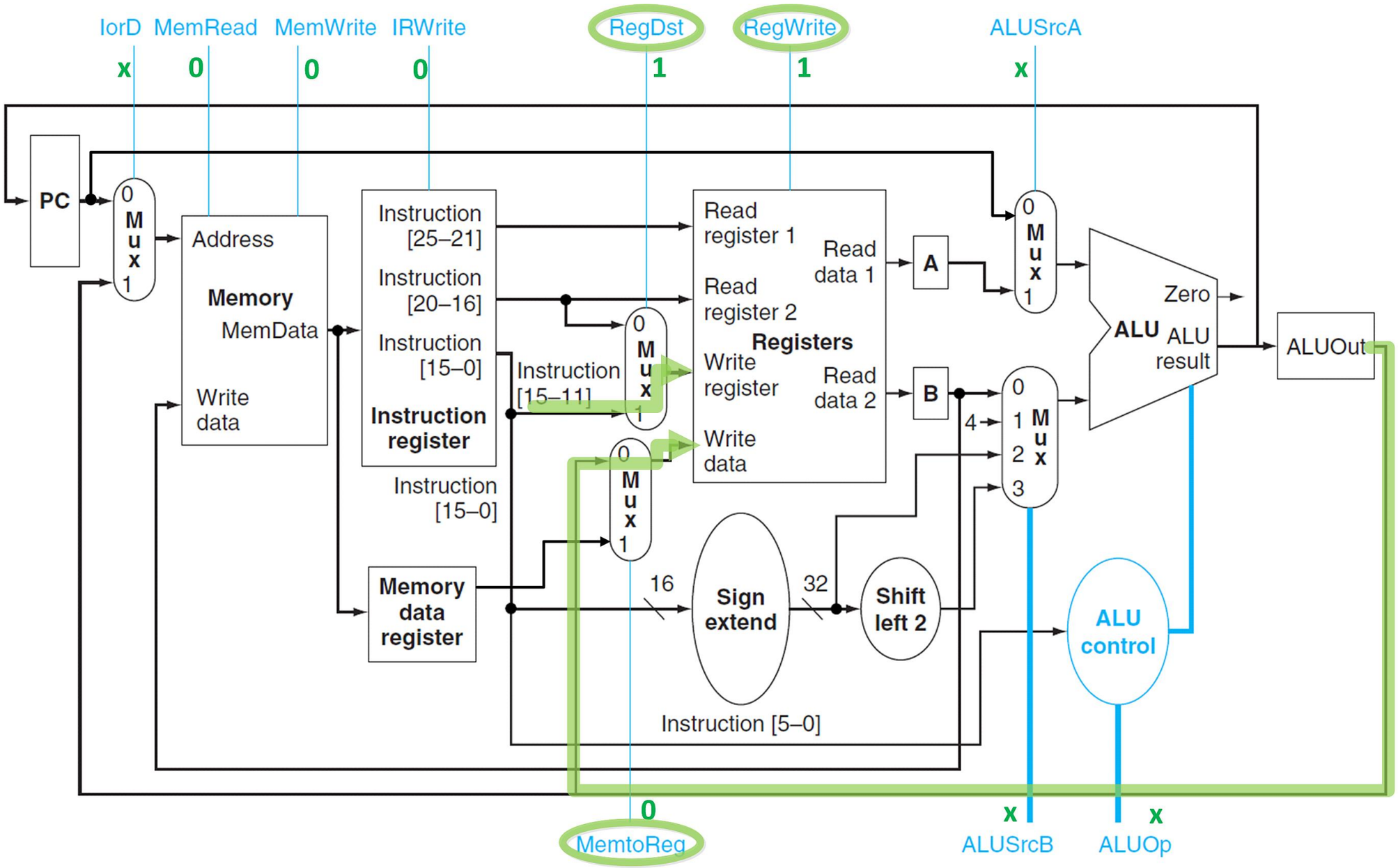
Add: T1 → A ← R[rs], B ← R[rt];



Add: ADD & T2 → ALUOut ← A + B;

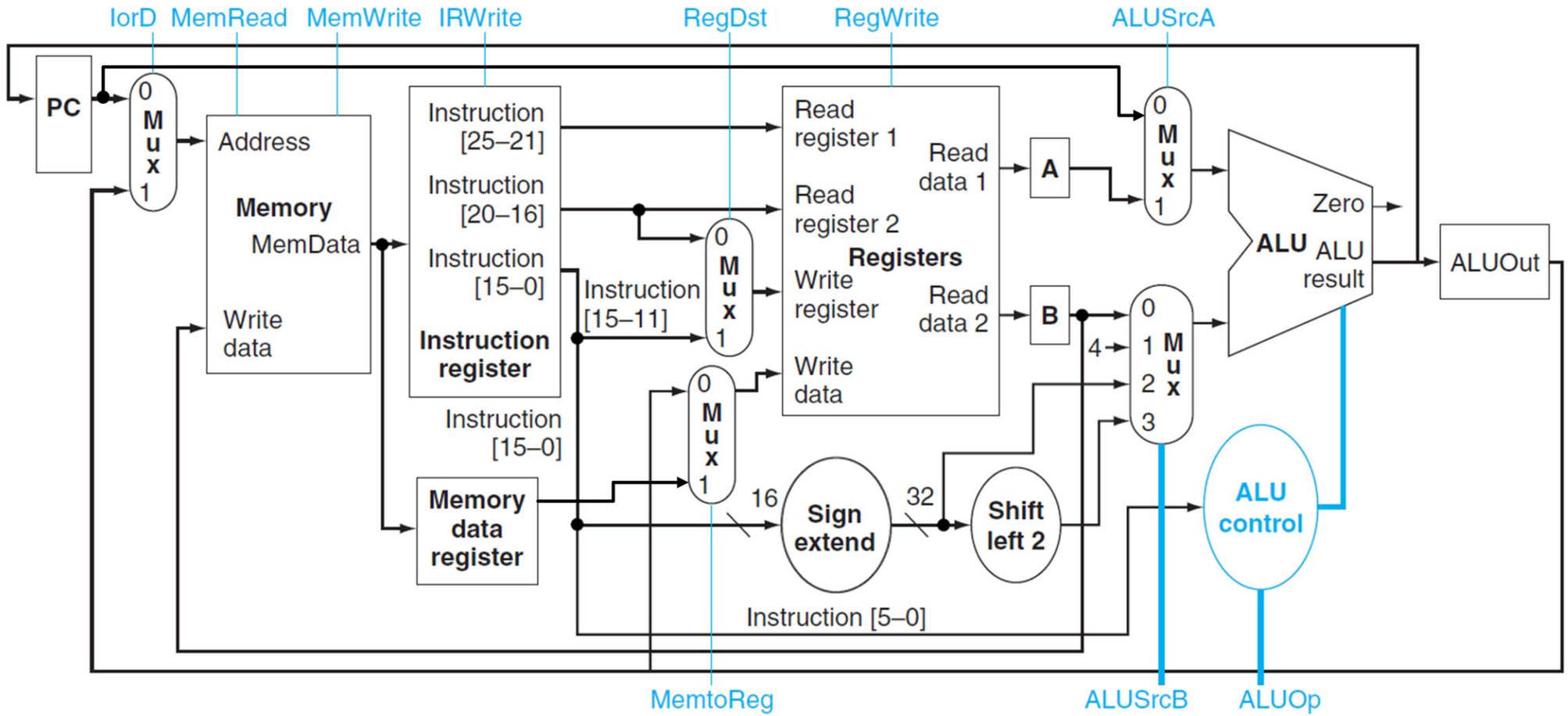


Add: ADD & T3 → R[rd] ← ALUOut;



Proiectarea MIPS Multi-ciclu – Pas 4

ori $rt, $rs, imm16, **RTL Abstract**: R[rt] ← R[rs] | ZeroExt(Imm16), PC ← PC + 4;

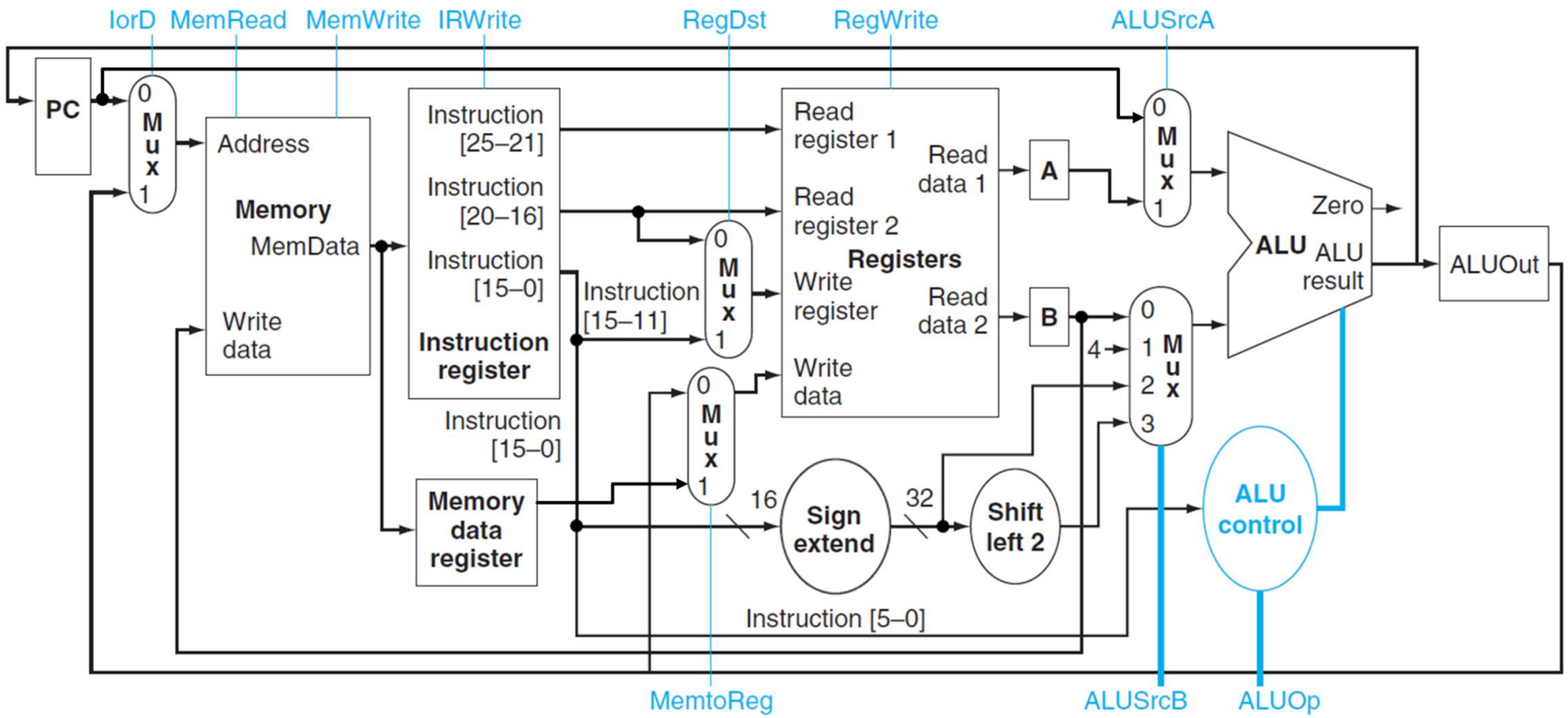


[1]

Pas (tact)	RTL Concret	Semnale de control (relevante)
T0 →	IR ← M[PC], PC ← PC + 4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A ← R[rs], B ← R[rt];	
ORI & T2 →	ALUOut ← A OR *Z*_Ext(Imm16);	/ExtOp, ALUSrcA, ALUSrcB=2, or
ORI & T3 →	R[rt] ← ALUOut;	/RegDst, /MemtoReg, RegWrite

Proiectarea MIPS Multi-ciclu – Pas 4

ori – continuare…



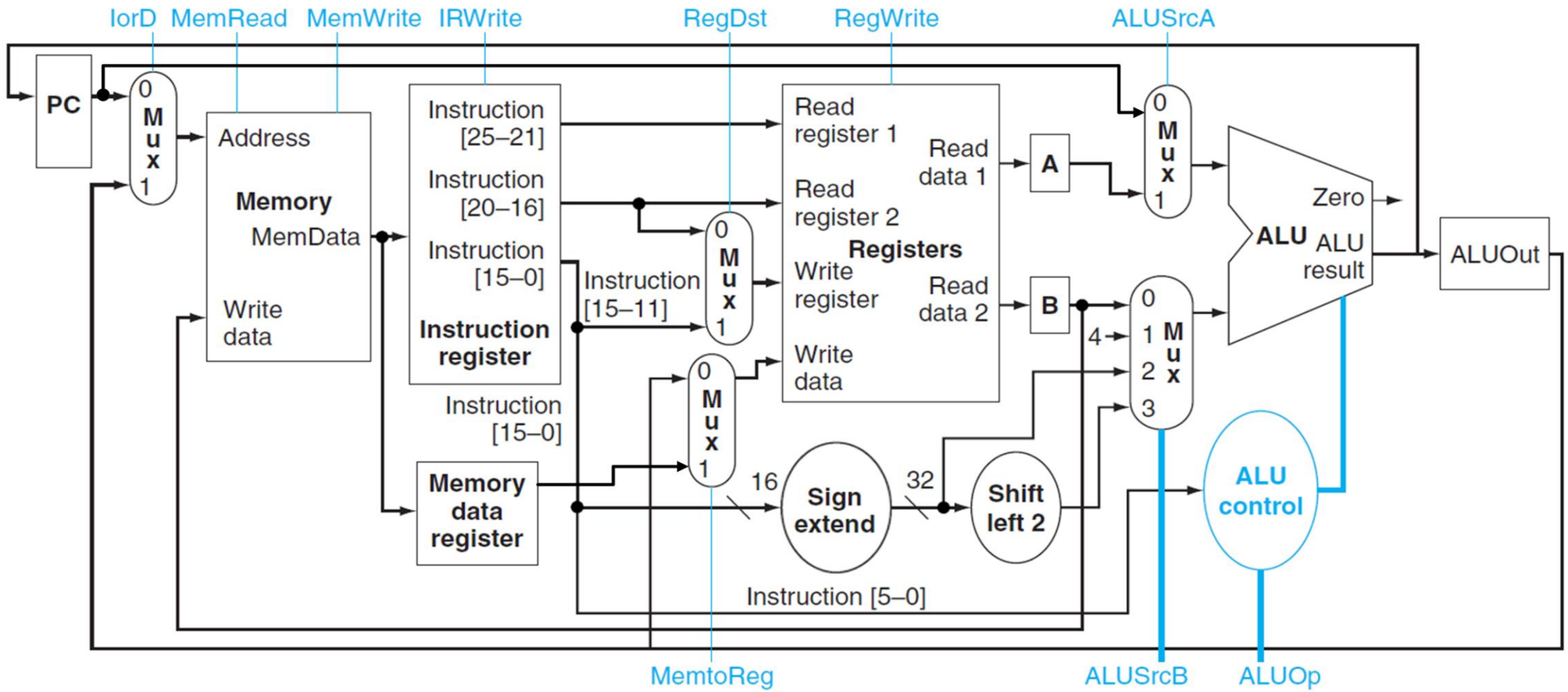
[1]

Semnale de control (specificare completă)

Pas	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	0	1	2	**or**
T3	x	0	0	0	0	0	1	x	x	x	x

Proiectarea MIPS Multi-ciclu – Pas 4

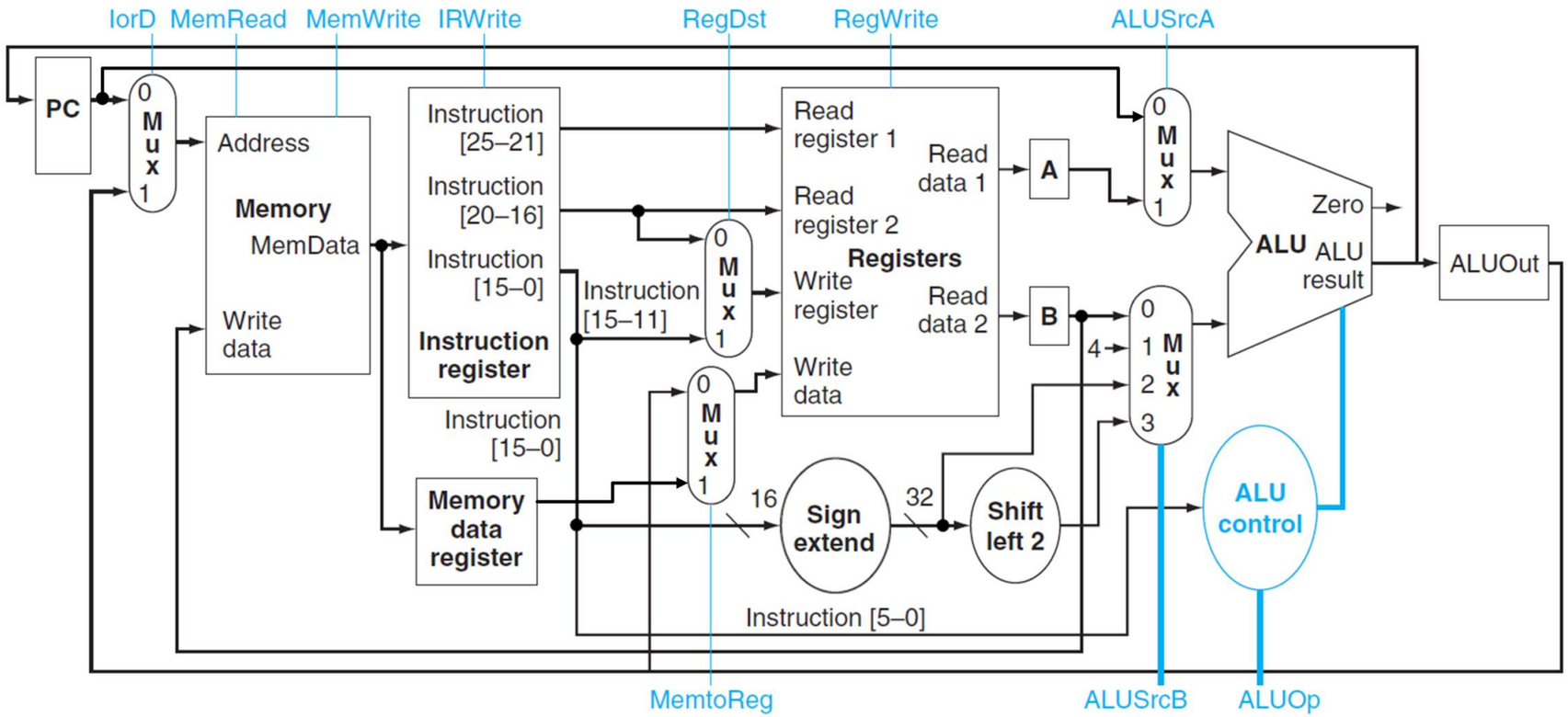
lw $rt, imm($rs), **RTL Abstract**: RF[rt] ← M[RF[rs] + S_Ext(imm)], PC ← PC + 4



[1]

Pas (tact)	RTL Concret	Semnale de control (relevante)
T0 →	IR ← M[PC], PC ← PC + 4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A← R[rs], B ← R[rt];	
LW & T2 →	ALUOut ← A + SignExt(Imm16);	ExtOp, ALUSrcA, ALUSrcB=2,add
LW & T3 →	MDR ← M[ALUOut];	IorD, MemRead
LW & T4 →	R[rt] ← MDR;	MemtoReg, RegWrite

Proiectarea MIPS Multi-ciclu – Pas 4



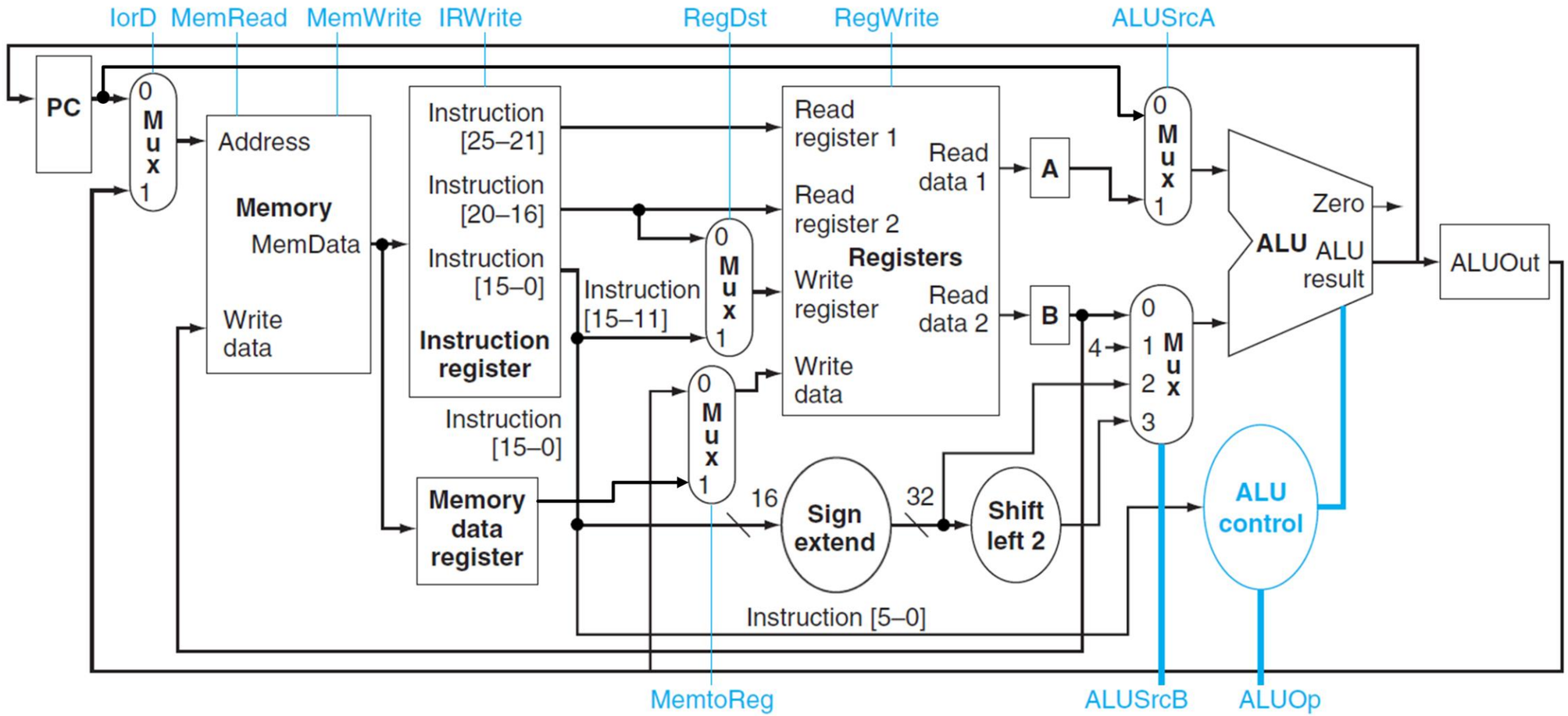
[1]

lw - semnale de control (specificare completă)

Pas	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	X	0	x	0	1	add
T1	x	0	0	0	x	X	0	x	x	x	x
T2	x	0	0	0	x	X	0	1	1	2	add
T3	1	1	0	0	x	X	0	x	x	x	x
T4	x	0	0	0	0	1	1	x	x	x	x

Proiectarea MIPS Multi-ciclu – Pas 4

sw $rt, imm($rs), **RTL Abstract**: M[RF[rs] + S_Ext(imm)] ← RF[rt], PC ← PC + 4

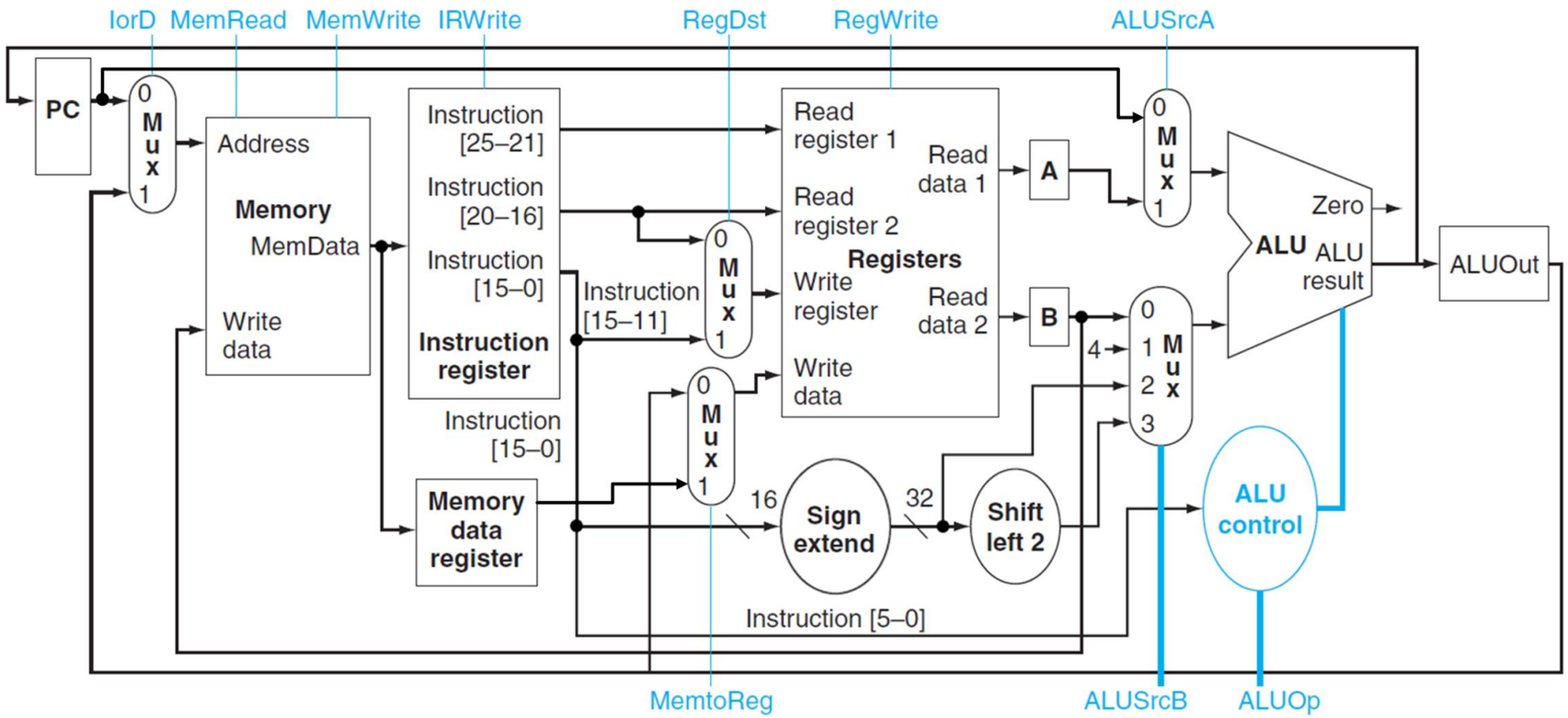


[1]

Pas (tact)	RTL Concret	Semnale de control (relevante)
T0 →	IR ← M[PC], PC ← PC + 4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A← R[rs], B ← R[rt];	
SW & T2 →	ALUOut ← A + SignExt(Imm16);	ExtOp, ALUSrcA, ALUSrcB=2,add
SW & T3 →	M[ALUOut] ← B;	IorD, MemWrite

Proiectarea MIPS Multi-ciclu – Pas 4

sw – continuare…

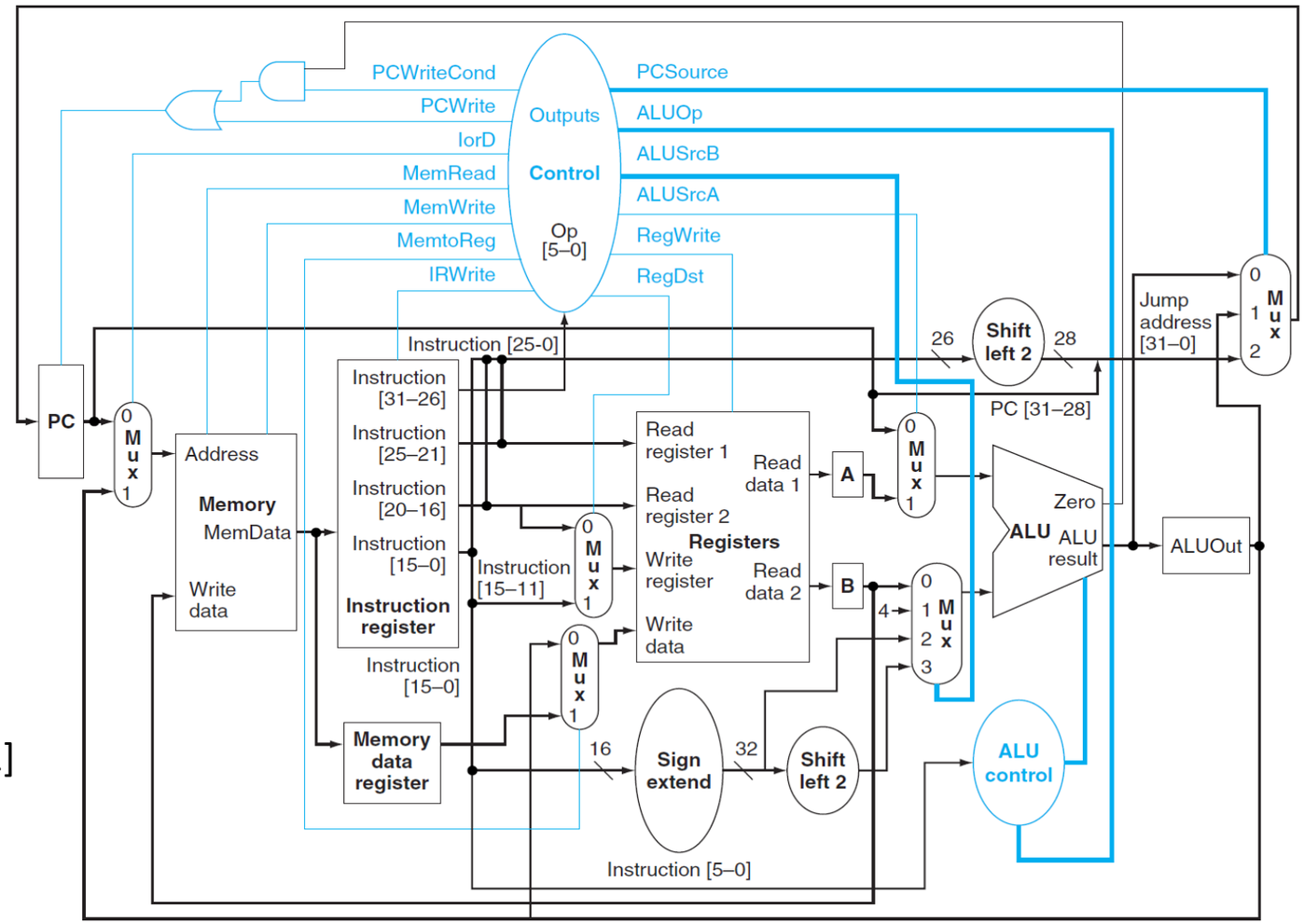


[1]

Semnale de control (specificare completă)

Pas	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	1	1	2	add
T3	1	0	1	0	x	x	0	x	x	x	X

Proiectarea MIPS Multi-ciclu – Pas 4



[1]

beq $rt, $rs, imm

RTL Abstract:

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

PC ← PC + 4 + S_Ext(imm) << 2

else

PC ← PC + 4

Notă (vezi mai jos): T1 si T2 pot fi executate în paralel, în aceeași perioadă de ceas!

T0 →	IR ← M[PC], PC ← PC + 4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A ← R[rs], B ← R[rt];	
BEQ & T2 →	ALUOut ← PC + S_Ext(Imm16)<<2;	ExtOp, /ALUSrcA, ALUSrcB=3,add
BEQ & T3 →	(R[rs] = R[rt]) → PC ← ALUOut;	ALUSrcA, ALUSrcB=0, sub, PCSrc=1, PCWrCd, /PcWr

Proiectarea MIPS Multi-ciclu – Pas 4

Optimizare beq prin unificarea pașilor T1 și T2 pe același ciclu de ceas

T0 →	IR ← M[PC], PC ← PC + 4;	/IorD, MemRead, IRWrite, /ALUSrcA, ALUSrcB=1, add
T1 →	A ← R[rs], B ← R[rt], ALUOut ← PC + S_Ext(Imm16)<<2;	ExtOp, /ALUSrcA, ALUSrcB=3, add
BEQ & T2 →	(R[rs] = R[rt])→ PC ← ALUOut;	sub, PCSrc=1, PCWrCd, /PcWr, ALUSrcA, ALUSrcB=0

Semnale de control în detaliu pentru beq

Pas	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op	PC Src	PC WrCd	PC Wr
T0	0	1	0	1	x	x	0	x	0	1	add	0	0	1
T1	X	0	0	0	x	x	0	1	0	3	add	x	0	0
T2	X	0	0	0	x	x	0	x	1	0	sub	1	1	0

JMP ← temă acasă…deși seamănă, acesta nu e RTL abstract, se face **concret**!

Proiectarea MIPS Multi-ciclu, pași 1 - 4 – Sumar

- Cinci faze de execuţie, implementate pe un număr variabil de pași (3-5 tacți):
  - Obţinerea instrucţiunii (Instruction Fetch)
  - Decodificarea instrucţiunii și citire din Blocul de Registre
  - Execuţie, calculul adresei de memorie, sau terminare instrucțiune de ramificare
  - Acces la memorie sau terminarea instrucțiunii de tip R
  - Scrierea rezultatului în Blocul de Registre (Write-back)
- Toate operațiile din fiecare ciclu Ti sunt efectuate în paralel, nu secvențial!
  - De ex. pe T0 → IR ← M[PC] și PC ← PC+4 se execută simultan!
- Între ciclurile T1 și T2 unitatea de control selectează pasul cu care se continuă conform tipului instrucțiunii.

Proiectarea MIPS Multi-ciclu, pași 1 - 4 – Sumar

Pas / Ciclu	Action for R-type instructions	Action for ORI instruction	Action for memory reference instructions	Action for branches	Action for jumps
T0: Instruction Fetch	IR ← M[PC] PC ← PC + 4				
T1: Instruction decode / register Fetch	A ← RF[IR[25:21]] B ← RF[IR[20:16]] ALUOut ← PC + S_Ext(IR[15:0]) << 2				
T2: Execution, address computation, branch / jump completion	ALUOut ← A op B	ALUOut ← A OR Z_Ext(Imm16)	ALUOut ← A + S_Ext(IR[15:0])	If (A == B) PC ← ALUOut	PC ← PC[31:28]|| IR[25:0] << 2
T3: Memory access or R-type completion	RF[IR[15:11] ← ALUOut	RF[IR[20:16]] ← ALUOut;	LW: MDR ← M[ALUOut] SW: M[ALUOut] ← B		
T4: Memory read completion			LW: RF[IR[20:16]] ← MDR		

Proiectarea MIPS Multi-ciclu – Semnale de Control

T	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op	PC Src	PC WrCd	PC Wr	
T0	0	1	0	1	x	x	0	x	0	1	add	0	0	1	**IF**
T1	x	0	0	0	x	x	0	1	1	3	add	x	0	0	**ID**
T2	x	0	0	0	x	x	0	x	1	0	func	x	0	0	**Ex R-T**
T3	x	0	0	0	1	0	1	x	x	x	x	x	0	0	**Wb R-T**
T2	x	0	0	0	x	x	0	0	1	2	or	x	0	0	**Ex ORI**
T3	x	0	0	0	0	0	1	x	x	x	x	x	0	0	**Wb ORI**
T2	x	0	0	0	x	x	0	1	1	2	add	x	0	0	**Ex LW**
T3	1	1	0	0	x	x	0	x	x	x	x	x	0	0	**M LW**
T4	x	0	0	0	0	1	1	x	x	x	x	x	0	0	**Wb LW**
T2	x	0	0	0	x	x	0	1	1	2	add	x	0	0	**Ex SW**
T3	1	0	1	0	x	x	0	x	x	x	x	x	0	0	**M SW**
T2	x	0	0	0	x	x	0	x	x	x	sub	1	1	0	**Ex Beq**
T2	x	0	0	0	x	x	0	x	x	x	x	2	0	1	**Ex J**

Tabelul 1: Valorile Semnalelor de Control pe fiecare pas / ciclu de ceas

- Fazele de execuție: IF, ID, Ex – Execute, M – Memory, Wb – Write result
- Instructiuni: R - T – tip R, ORI, LW, SW, Beq , J
- ExtOp: 1/0 → 1 – aritmetic, 0 – operații logice

…Proiectarea in detaliu pentru unitatea de control, în mai multe versiuni, în cursul 7

Proiectarea MIPS multi-ciclu - Concluzii

- Transformarea procesorul MIPS cu ciclu unic în MIPS multi-ciclu
  - Căile combinaţionale lungi au fost secționate *echilibrat* prin introducerea de regiștri invizibili pentru programator
  - Componente de bază au fost refolosite (Memorie comună de date și instrucțiuni, ALU), în caz contrar se obținea un procesor de tip pipeline
- Prin refolosirea componentelor => necesitatea mai multor căi spre aceleași destinații
- Controlul comunicației prin căile de date s-a realizat prin folosirea de multiplexoare ⇔ căi de date bazate pe multiplexoare

Probleme

1. Implementarea altor instrucțiuni. Vezi problemele din cursul 4 si cursul de rezolvat probleme!

Bibliografie

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013 si editii mai noi.
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”.