Arhitectura Calculatoarelor

Curs 7: Proiectarea MIPS multi-ciclu 2 - Unitatea de control

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

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

Proiectarea MIPS multi-ciclu

Proiectarea procesorului pas cu pas → MIPS multi-ciclu (Multi cycle), calea de date construită în cursul 6 (pași 1-4):

- 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**

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 Regiștri
  - Execuţie, calculul adresei de memorie, sau terminare instrucțiune de ramificare
  - Acces la memorie sau terminarea instrucțiunii de tip R
  - Scrierea rezultatului în Blocul de Regiștri (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 – 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

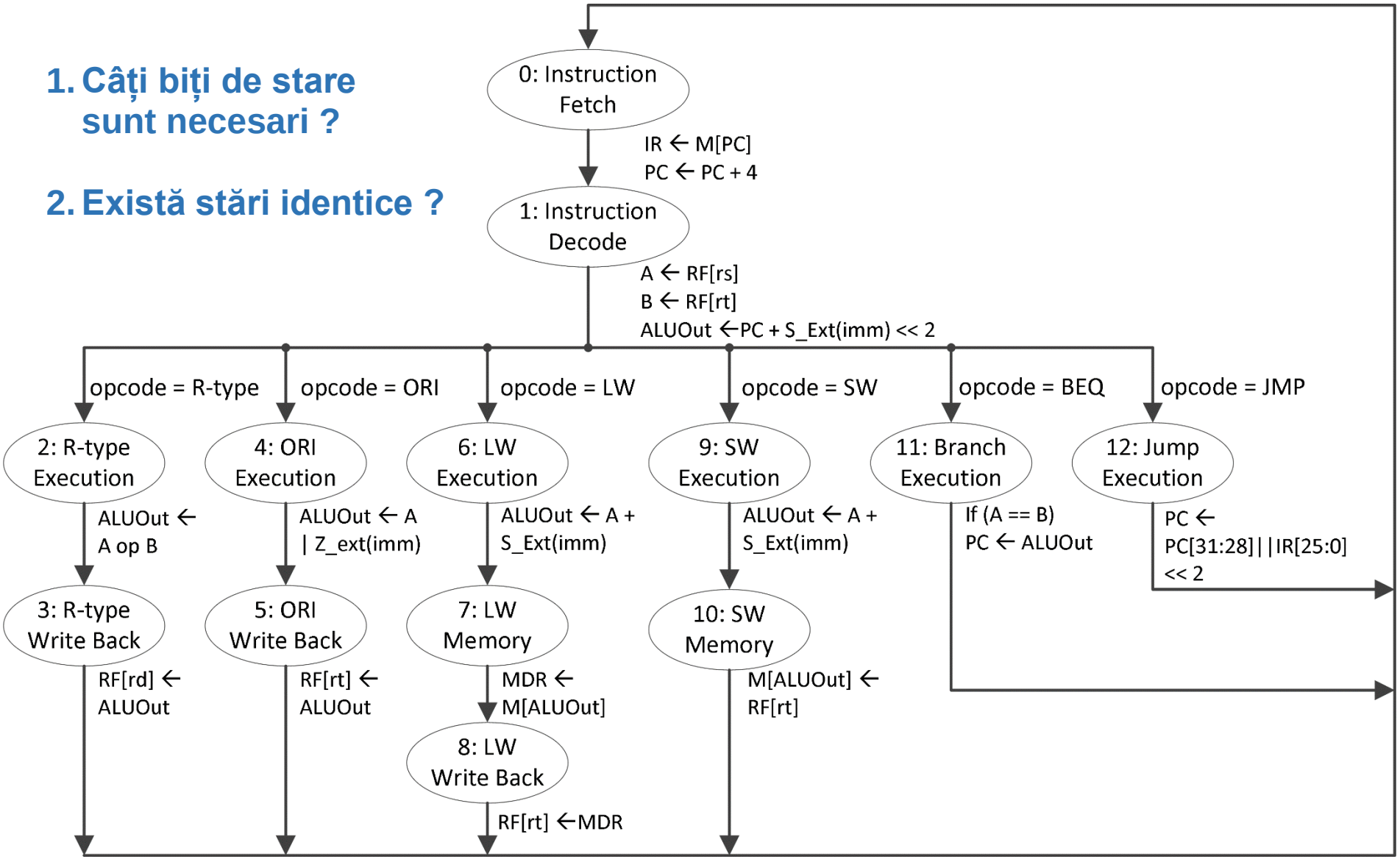
- Fazele de execuție: IF, ID, Ex – Execute, M – Memory, Wb – Write result
- Instructiuni: R - T – tip R, ORI, LW, SW, Beq, J
- ExtOp: 1/0 → 1 – aritmetic, 0 – operații logice

Proiectarea MIPS Multi-ciclu – Pas 5

Mașină cu un număr finit de stări pentru setul de instrucțiuni de bază: FSM1

- FSM1 se bazează pe tabelul 1, cu semnalele de control definite pe pași

1. **Câți biți de stare sunt necesari ?**
2. **Există stări identice ?**

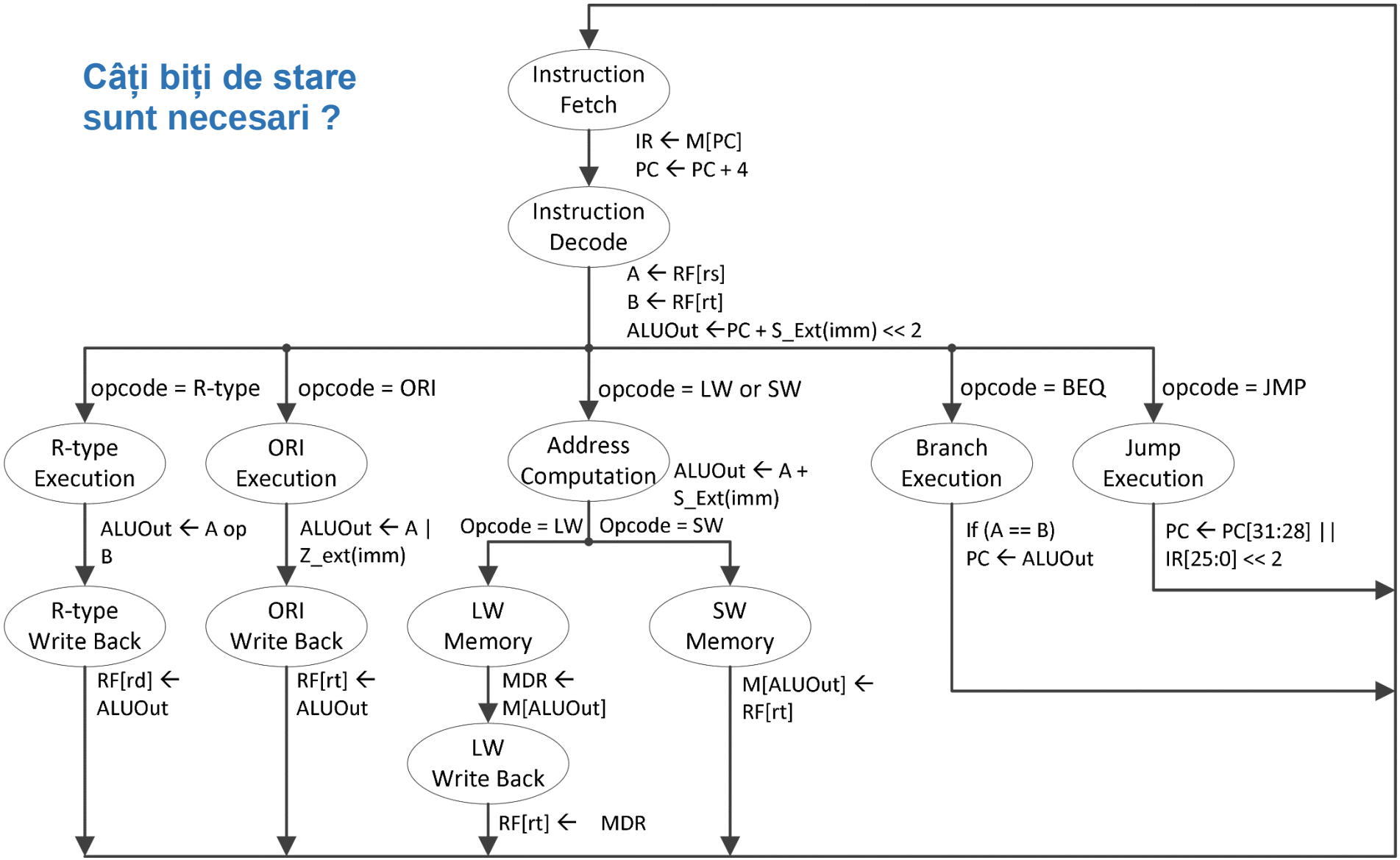


Proiectarea MIPS Multi-ciclu – Pas 5

Mașină cu un număr finit de stări pentru setul de instrucțiuni de bază: FSM2

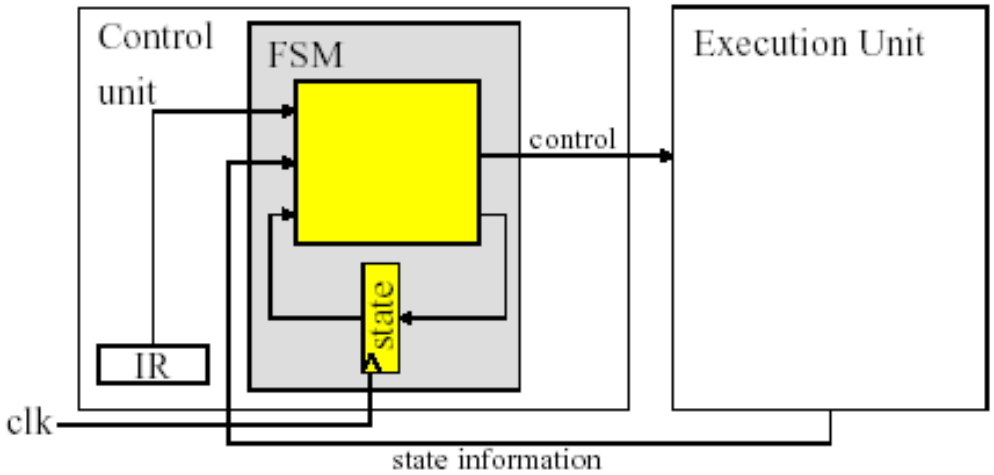
- Optimizare legată de unificare fazei comune de execuție de la LW/SW

Câți biți de stare sunt necesari ?

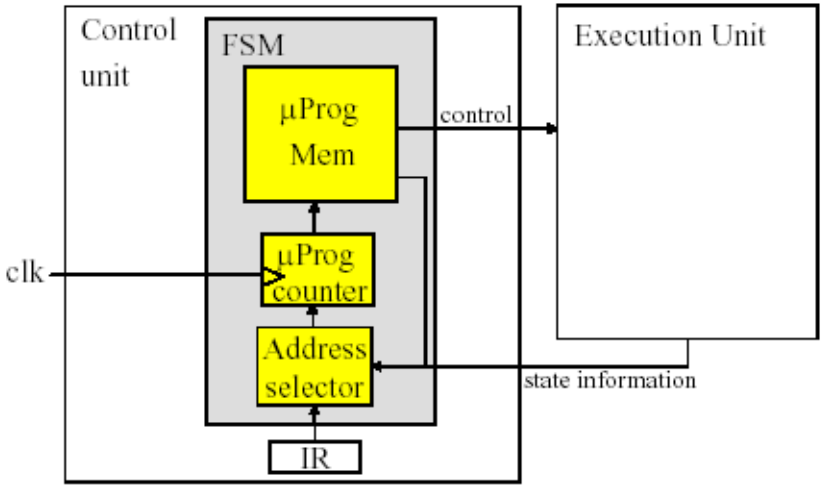


Proiectarea MIPS Multi-ciclu – Pas 5

- **Implementarea Unității de Control:**
  - Cablată (Hardwired)
  - Micro-programată
- **Cablată vs. Micro-programată**
  - Unitățile cablate sunt mai rapide
  - Proiectarea unităților cablate mari este complexă
  - Erorile cablate nu pot fi reparate la utilizator
  - Emularea pentru control micro-programat este simplă



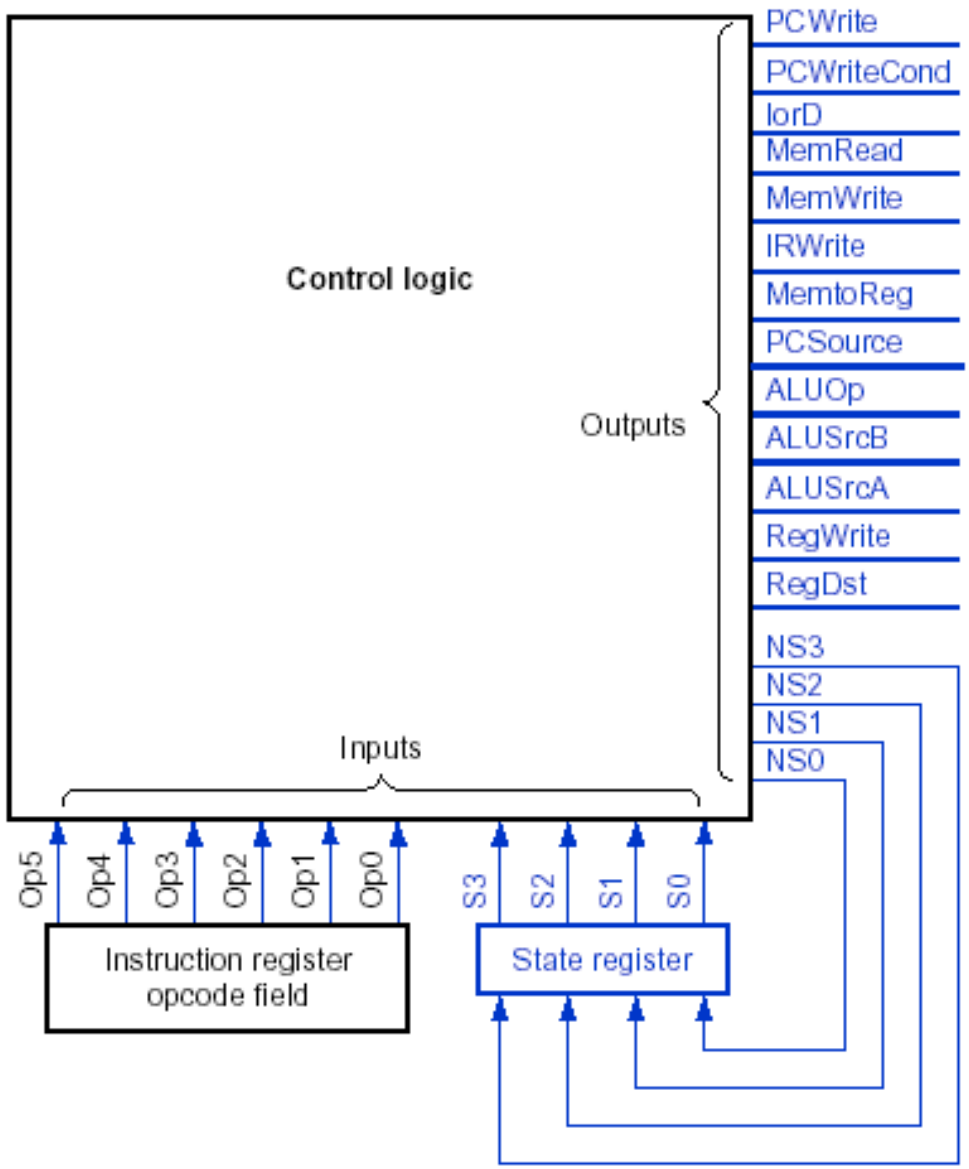
Unitate de control cablată



Unitate de control micro-programată

Proiectarea MIPS Multi-ciclu – Pas 5, Control cablat

Implementarea cablată a unității de control pentru MIPS Multi-ciclu



[2]

Unitate de control formată din:

- Logica de control
- Registrul de Stare pentru memorarea stării

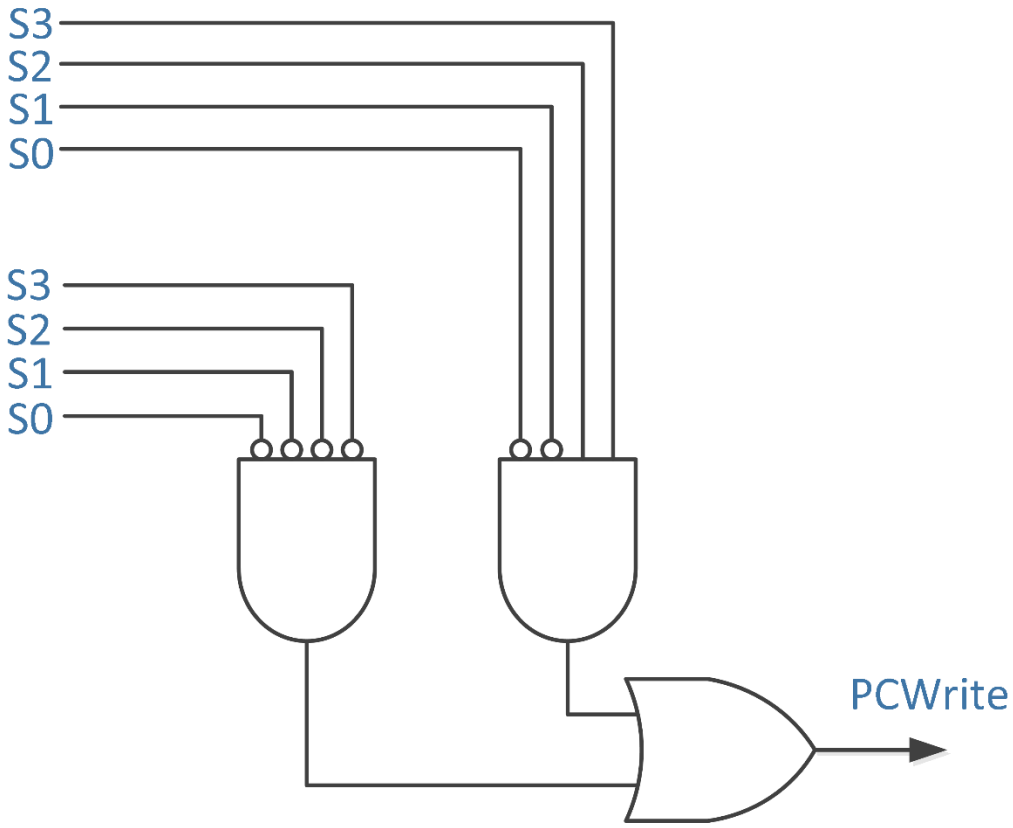
Ca implementări posibile:

- pentru logica de control se folosește PLA
- metoda cu un bistabil D pe stare (one flip-flop per state) pentru implementarea FSM
- numărător de secvențiere / salt + decodificator (nu se va detalia!)

Proiectarea MIPS Multi-ciclu – Pas 5, Control cablat

Implementare bazată pe PLA (Programmable Logic Array) pentru logica de control

- Se formează o rețea de porți SI ale căror ieșiri sunt conectate la porți SAU
- Fiecare semnal de control este setat în funcție de starea curentă
- Starea curentă + câmpul opcode vor da starea următoare



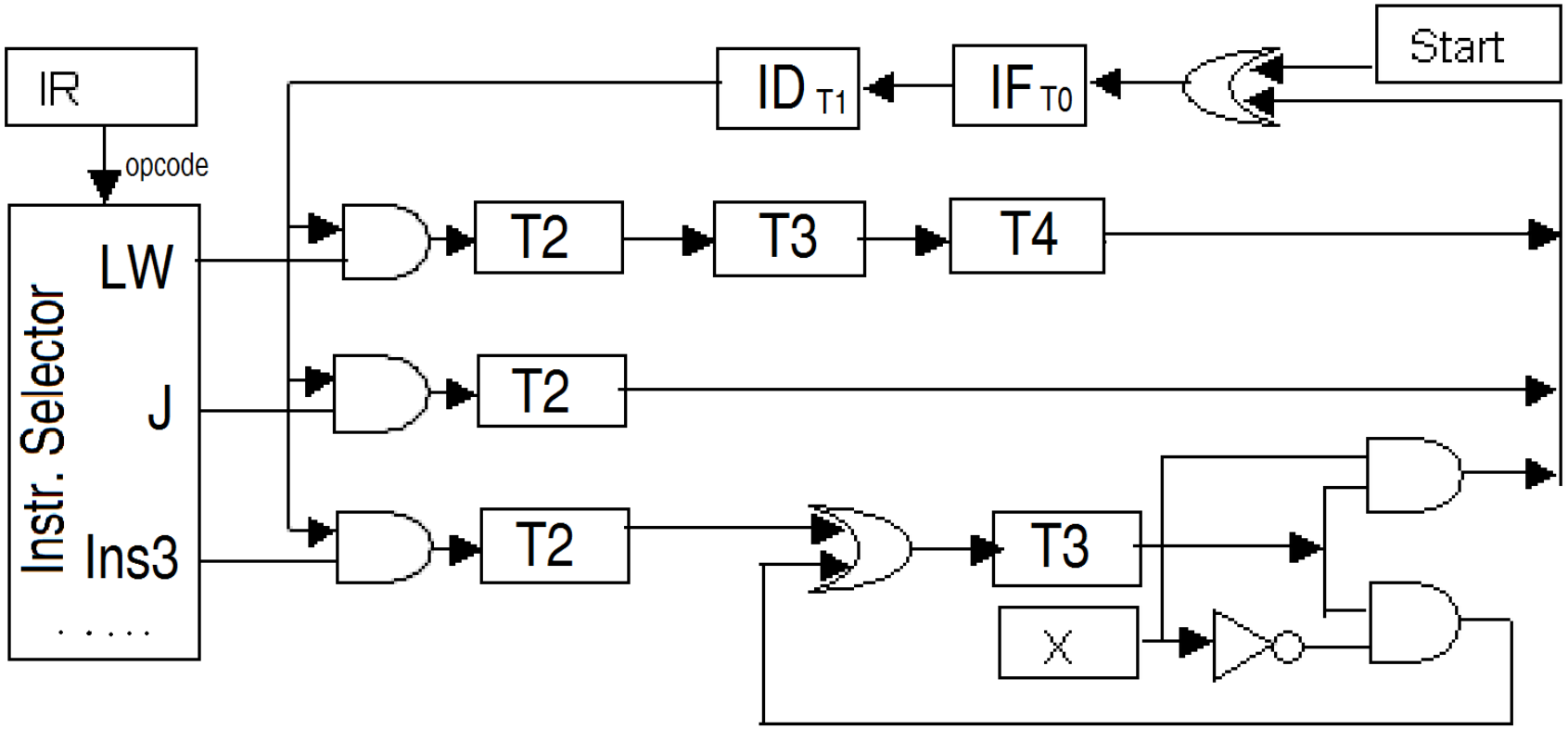
Exemplu: semnalul PCWrite

- vezi tabelul 1, respectiv FSM1, stări pe 4 biți, numerotate de sus în jos / stânga la dreapta
- PCWrite trebuie activat în starea *Instruction Fetch* (codificată 0=”0000”), respectiv în *Jump Execution* (codificată cu 12 = ”1100”)

Proiectarea MIPS Multi-ciclu – Pas 5, Control cablat

Implementare cu un bistabil per stare (One flip-flop per state / One-Hot encoding)

-exemplu cu LW, J si o instrucțiune generica (Ins3, instructiune care necesita etapa repetitiva in functie de un semnal de stare), se extinde similar si pentru restul instructiunilor



- Start: Un semnal sincronizat de durata unei perioade de ceas (1 pentru pornirea executiei)
- X: Flag de condiţie pentru instrucțiuni care necesită repetiție (ex. deplasare)
- Iesirile din fiecare bistabil se leaga la semnalele de control care trebuie sa fie 1 pe faza respectiva
- Mai uşor de extins sau modificat

Proiectarea MIPS Multi-ciclu – Pas 5, Control cablat

- Exemplu de extindere cu instrucțiuni care durează un număr variabil de cicluri de ceas

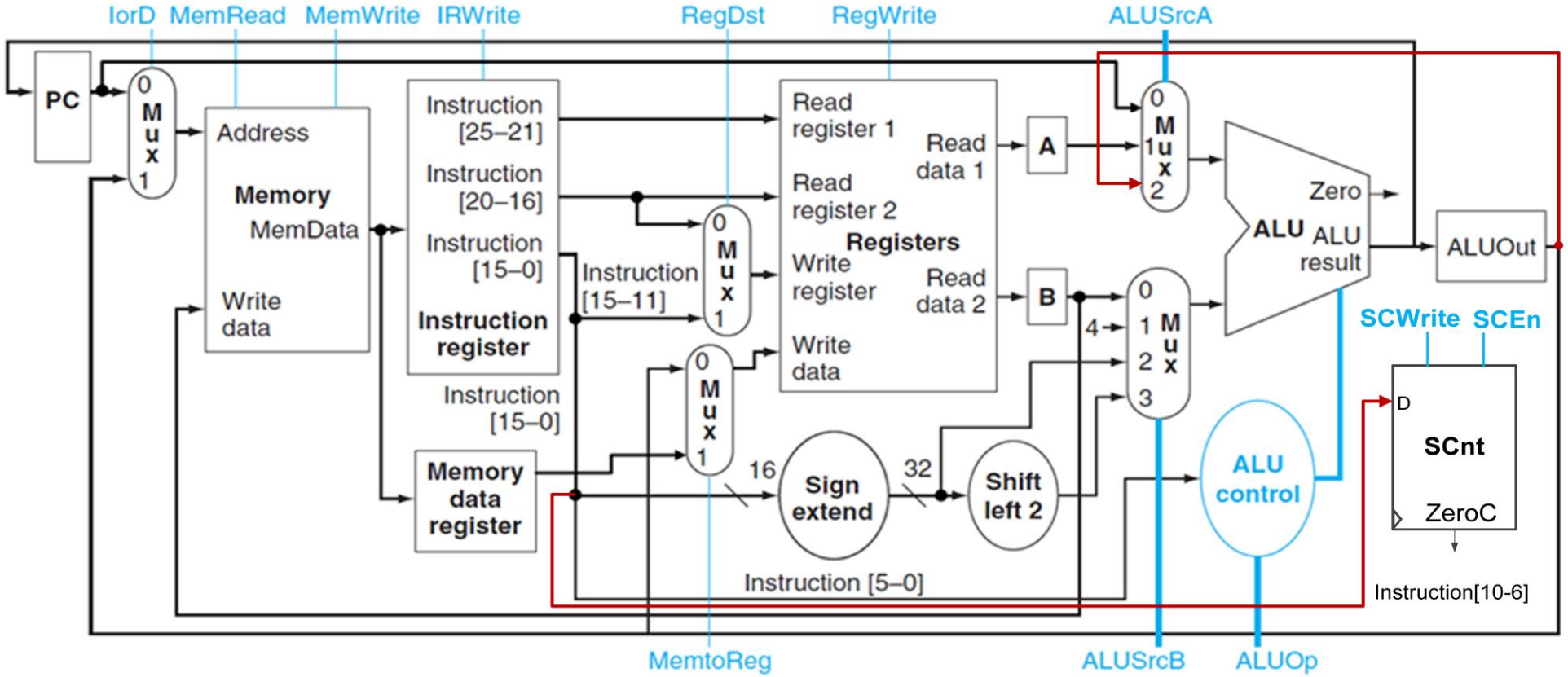
Tip R: sll $rd, $rs, sa RTL Abstract: RF[rd] ← RF[rs] << sa

- Presupunem ca ALU face doar deplasări de 1 bit
- Trebuie extinsă calea de date încât să permită:
  - Un mecanism de numărare a ciclurilor de ceas: numărător dedicat **SCnt** cu încărcare (controlat prin SCWrite, se încarcă valoarea *sa* din instrucțiune), activare a numărării (descrescător, controlat prin SCen), semnal de stare ZeroC pe 1 bit care semnalează când ajunge la 0
  - Legătură de la ALUOut la una dintre intrările ALU, pentru a repeta deplasarea de 1 bit de câte ori este necesar

- Unitatea de control tratează această instrucțiune de tip R tinand cont de ZeroC (nu se va detalia în curs)

Proiectarea MIPS Multi-ciclu – Pas 5, Control cablat

Exemplu de extindere cu Tip R: sll $rd, $rs, sa



[1]

Pas (tact)	RTL Concret	Semnale de control (relevante)
T0 →	IR ← M[PC], PC ← PC + 4;	/IorD, MemRead, IRWrite, ALUSrcA=0, ALUSrcB=1, add
T1 →	A← R[rs], B ← R[rt]; SCnt← sa	SCWrite
T2 & ZeroC=0 →	**ALUOut ← A << 1; SCnt← SCnt – 1**	**ALUSrcA=1, AluOp=TipR, function=sll, SCEn**
T2 & ZeroC=1 →	**ALUOut ← A;**	**ALUSrcA=1, AluOp=PassA**
T3 & (ZeroC=0) →	**ALUOut ← ALUOut << 1; SCnt← SCnt – 1; repeat (load T3)**	**ALUSrcA=2, AluOp=TipR, function=sll, SCEn**
T3 & (ZeroC=1) →	**R[rd] ← ALUOut; load T0**	**RegDst, /MemtoReg,RegWrite**

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat

Dezavantajele folosirii unităților de control cablate

- Complexitatea cablării / FSM-ului depinde de complexitatea căilor de date, respectiv a logicii de control
- Procesoarele reale sunt complexe: sute de instrucțiuni cu 1 – 20 cicluri de ceas

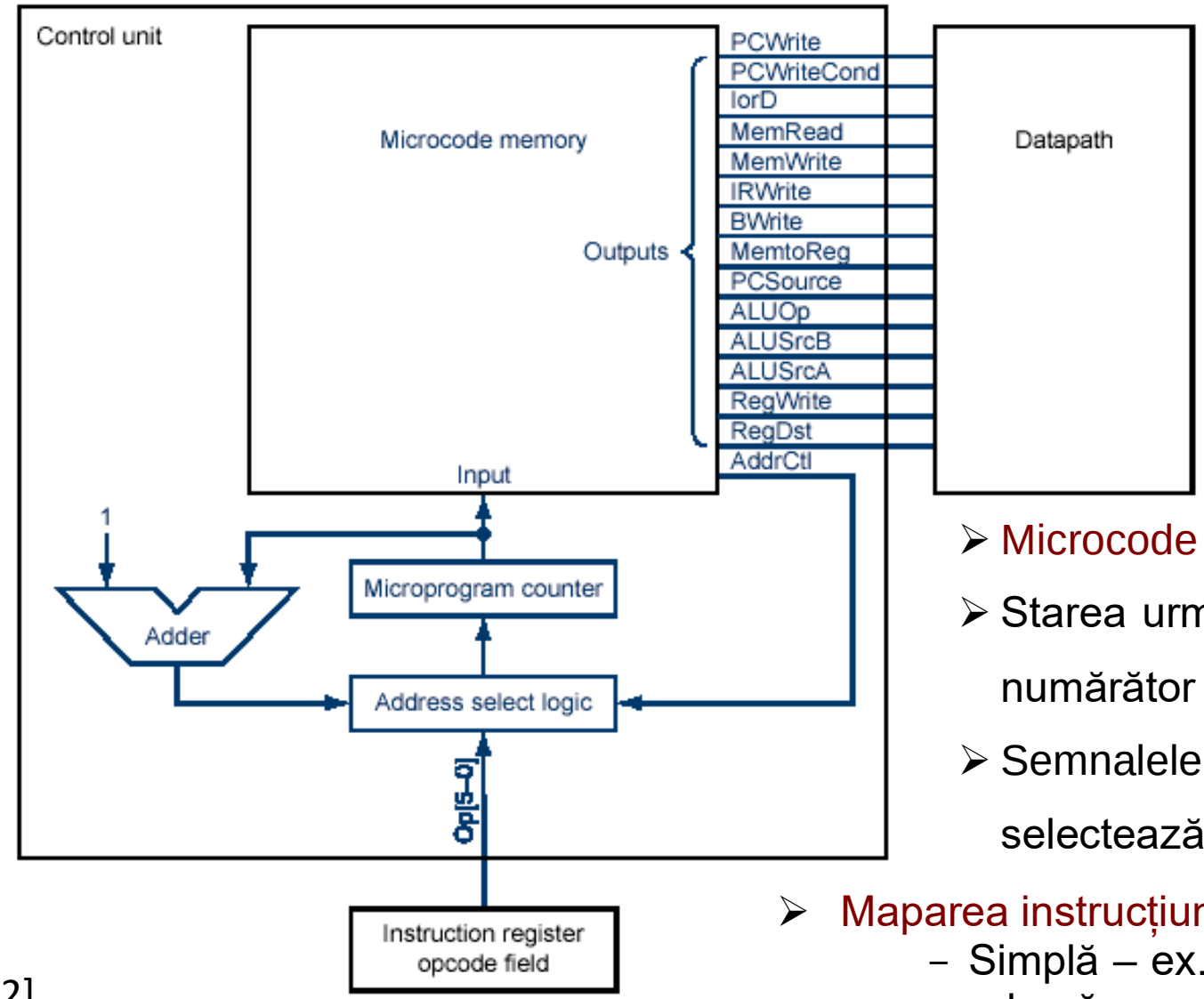
  → Se folosește microprogramare…

Unitatea de control micro-programată

- Potrivită pentru sute de instrucțiuni (opcode-uri), moduri, cicluri, etc.
- Semnalele se specifică simbolic prin micro-instrucțiuni
- Moduri de secvențiere posibile, necesare într-o unitate de control micro-programată:
  - Incrementarea registrului de adresă a micro-instrucțiunii
  - Ramificări condiționate și necondiționate
  - Maparea codului de operație al instrucțiunii la adresa secvenței corespunzătoare de micro-instrucţiuni în memoria unității de control
  - Mijloace pentru apel / revenire subrutină de micro-instrucțiuni

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat

Unitatea de control micro-programată, diagrama bloc



[2]

- Microcode = Micro-instrucțiuni
- Starea următoare se obține folosind un numărător (cel puțin în unele stări)
- Semnalele AddrCtl determină cum se selectează starea (adresa) următoare
- Maparea instrucțiunilor la μ-instrucțiuni
  - Simplă – ex. concatenare între opcode și adresă
  - Complexă – ROM sau PLA

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat

Variante de Microprogramare:

- Microcod "Orizontal" – *urmează un studiu de caz pe paginile următoare*
  - Fiecare semnal de comandă este reprezentat de 1 bit în μ-instrucțiune
  - Operații paralele multiple per μ-instrucțiune ⇔ număr de pași redus per instrucțiune
  - Micro-cod orizontal înseamnă μ-instrucțiuni mai late
  - Codificare simplă → mai mulți biți

- Alte variante : Microcod "Vertical", codificare „Nano"

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat

Microprogramare - Pro și Contra

- Ușor de proiectat
- Flexibilitate
  - Uşor de adaptat la schimbări de organizare, viteză, tehnologie
  - Se pot opera schimbări în fazele de proiectare avansate, sau chiar în exploatare
- Se pot implementa seturi de instrucțiuni foarte complexe (doar cu memorie de microcod mai mare)
- Generalitate
  - Se pot implementa seturi de instrucțiuni multiple pe aceeași mașina
  - Se poate adapta setul de instrucțiuni la cerințele aplicației
- Compatibilitate
  - Mai multe organizări, același set de instrucțiuni
- Mai lent decât implementarea cablată

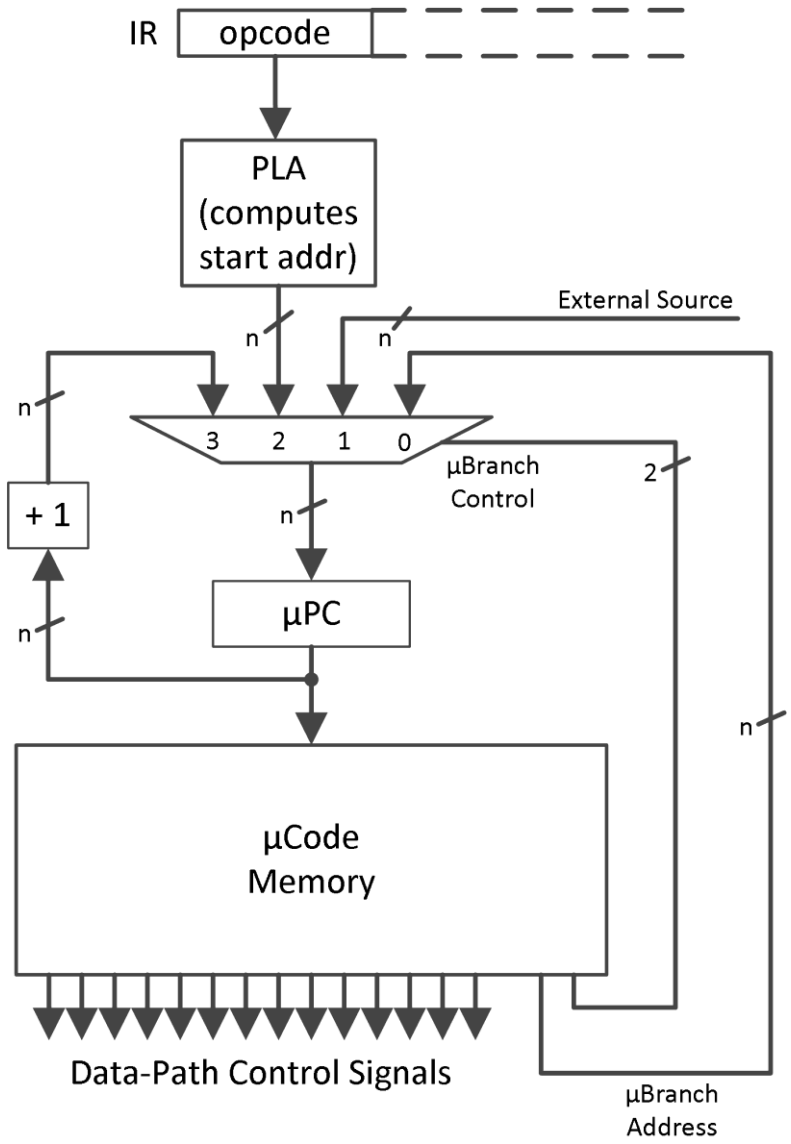
Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat

Proiectarea unui set de μ-instrucţiuni

1. Se pornește de la lista semnalelor de control
2. Se grupează semnalele în „câmpuri” cu o anumită semnificație (vs. random)
3. Se ordonează câmpurile după o anumită logică (ex, semnale pentru operaţii ALU & operanzi ALU, scriere memorie etc. la început, iar la sfârșit câmpul pentru secvențierea μ-instrucțiunilor)
4. Se creează o descriere simbolică pentru formatul micro-instrucțiunilor, care specifică numele câmpurilor și influenta lor asupra valorilor semnalelor de control
5. Pentru minimizarea lățimii, micro-operaţiile care nu vor fi folosite simultan se pot codifica în același câmp (în acest caz se obține microprogram vertical)

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat, V1

Exemplu: Unitate de control cu micro-program orizontal V1 pentru FSM1



Unitate de control micro-programată, schemă bloc (clasică)

Câmpurile μ-instrucțiunii

Data-Path Control Signals	μBranch Address	μBranch Control

- Semnalele de control sunt cele definite în tabelul 1
- μBranch Address: adresă de micro-ramificare, pe 4 biți (?!…sunt 13 stări / adrese)
- μBranch Control: semnalul de control, 2 biți, pentru secvențiere (selecția următoarei m-instrucțiuni, prin mux)
  - 00: adresa micro-ramificare
  - 01: adresa de sursa externă (nu se folosește concret în acest exemplu)
  - 10: adresa primei μ-instrucțiuni pentru codul de operaţie din IR
  - 11: adresa de micro-instrucţiune următoare (secvenţial)

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat, V1

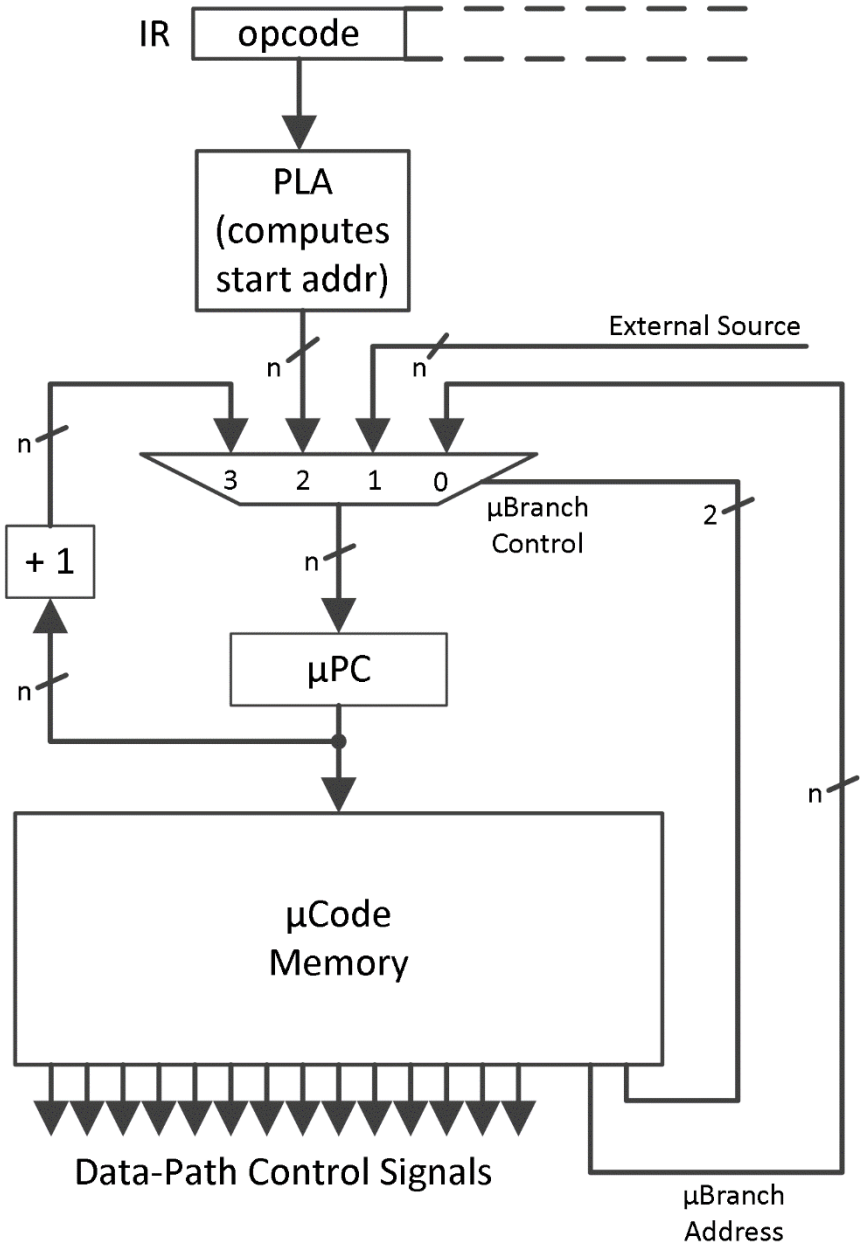
Addr.	IorD	MemRead	Mem Write	IRWrite	RegDst	Mem toReg	Reg Write	ExtOp	ALUSrcA	ALUSrcB	ALUOp	PCSrc	PCWrCd	PCWr	μBranch Address	μBranch Control	Execution Phase
No.	1b	1b	1b	1b	1b	1b	1b	1b	1b	2b	2b	2b	1b	1b	4b	2b	
00	0	1	0	1	x	x	0	x	0	1	add	0	0	1	x	3	IF
01	x	0	0	0	x	x	0	1	1	3	add	x	0	0	x	2	ID
02	x	0	0	0	x	x	0	x	1	0	func	x	0	0	x	3	Ex R-T
03	x	0	0	0	1	0	1	x	x	x	x	x	0	0	0000	0	Wb R-T
04	x	0	0	0	x	x	0	0	1	2	or	x	0	0	x	3	Ex ORI
05	x	0	0	0	0	0	1	x	x	x	x	x	0	0	0000	0	Wb ORI
06	x	0	0	0	x	x	0	1	1	2	add	x	0	0	x	3	Ex LW
07	1	1	0	0	x	x	0	x	x	x	x	x	0	0	x	3	M LW
08	x	0	0	0	0	1	1	x	x	x	x	x	0	0	0000	0	Wb LW
09	x	0	0	0	x	x	0	1	1	2	add	x	0	0	x	3	Ex SW
10	1	0	1	0	x	x	0	x	x	x	x	x	0	0	0000	0	M SW
11	X	0	0	0	x	x	0	x	x	x	sub	1	1	0	0000	0	Ex Br
12	x	0	0	0	x	x	0	x	x	x	x	2	0	1	0000	0	Ex J

Conținutul memoriei de μCod pentru instrucţiunile selectate (FSM1), dimensiune 13x23 biti

- μ-instrucţiuni orizontale, lungime: 23 biţi; adresele corespund cu ordinea din Tabelul 1
- Pentru acest exemplu, câmpul μBranch Address este folosit doar pentru generarea μ–adresei pentru IF (la terminarea instrucţiunii curente)
- Conectând la External Source μ-adresa pentru IF (0), se reduce lungimea μ-instrucţiunii.

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat, V1

*Discutată anterior – versiunea **V1***



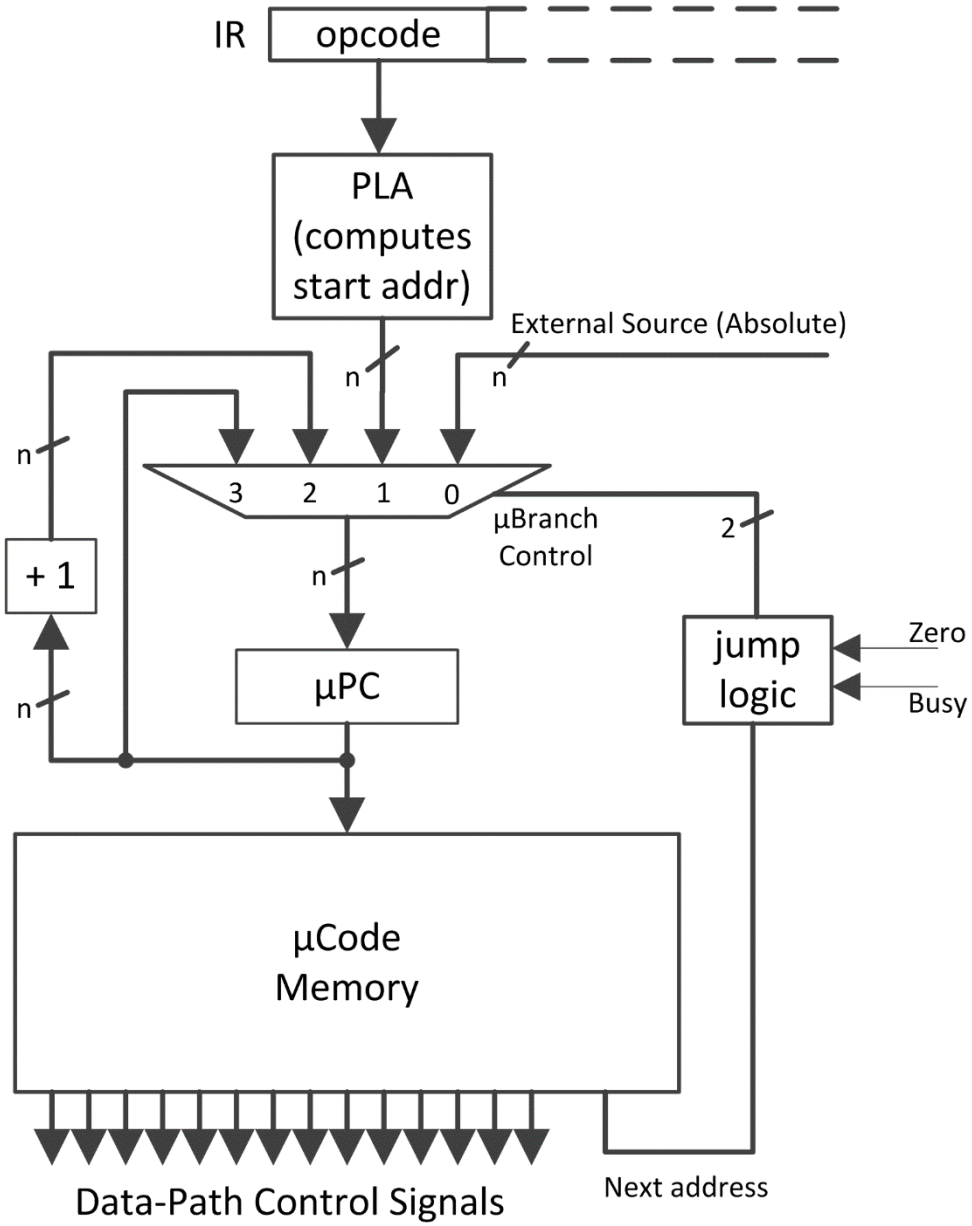
Intrări în MUX-ul de secvențiere:

- μPC+1,
- Adresa de start pentru execuția instrucțiunii curente,
- External Source,
- μBranch Address.

Dezavantaj: logică de selecție necondiționată de semnale de stare pentru MUX-ul de μadresă => nu se pot implementa instrucțiuni mai complexe (ex. SLL)

O variație posibilă: Intrări diferite în MUX-ul de secvențiere și logică de selecţie **condiționată** pentru MUX

Proiectarea MIPS Multi-ciclu – Pas 5, μ-programat, V2



Unitate de control micro-programată **V2**

Diferenţe faţă de V1:

- intrările μPC și μPC+1 la MUX
- în formatul μinstrucțiunii câmpurile μBranch Address și Control sunt înlocuite cu Next address, pe mai puțini biți
- External - valoare absolută indicând adresa IF în memoria de μCod
- secventierea dată de blocul jump logic în funcție de semnalele de stare din căile de date și de Next address

Exemple de μSalturi		**μPC devine**
next (următor)	→	μPC+1
spin (aşteptare)	→	if (busy) then μPC else μPC+1
fetch (start IF)	→	absolute
dispatch (execuție)	→	start addr (PLA)

Proiectarea MIPS Multi-ciclu – Excepții

- **Definiţie**: Excepțiile sunt evenimente care necesită întreruperea execuției fluxului curent de instrucțiuni și transferul controlului spre rutine care să trateze evenimentul
- **Tipuri**:
  - Excepţie [depășire] → generat în procesor
  - Întrerupere [I/O] → asociat cu evenimente externe
- **Excepții MIPS**: instrucțiuni nedefinite sau depăşire aritmetică.
- **Detectarea excepţiei** – Cum se constată o excepţie?
- **Tratarea excepției** – Ce trebuie făcut?

Detectarea Excepţiei la MIPS

- Instrucțiune nedefinită:
  - Adăugam starea IllegalOp la FSM
  - Fiecare instrucțiune necunoscută determină tranziție la starea IllegalOp
- Depășire aritmetică:
  - ALU are logica de detectare a depășirii → definim starea Overflow pentru tratarea depășirii

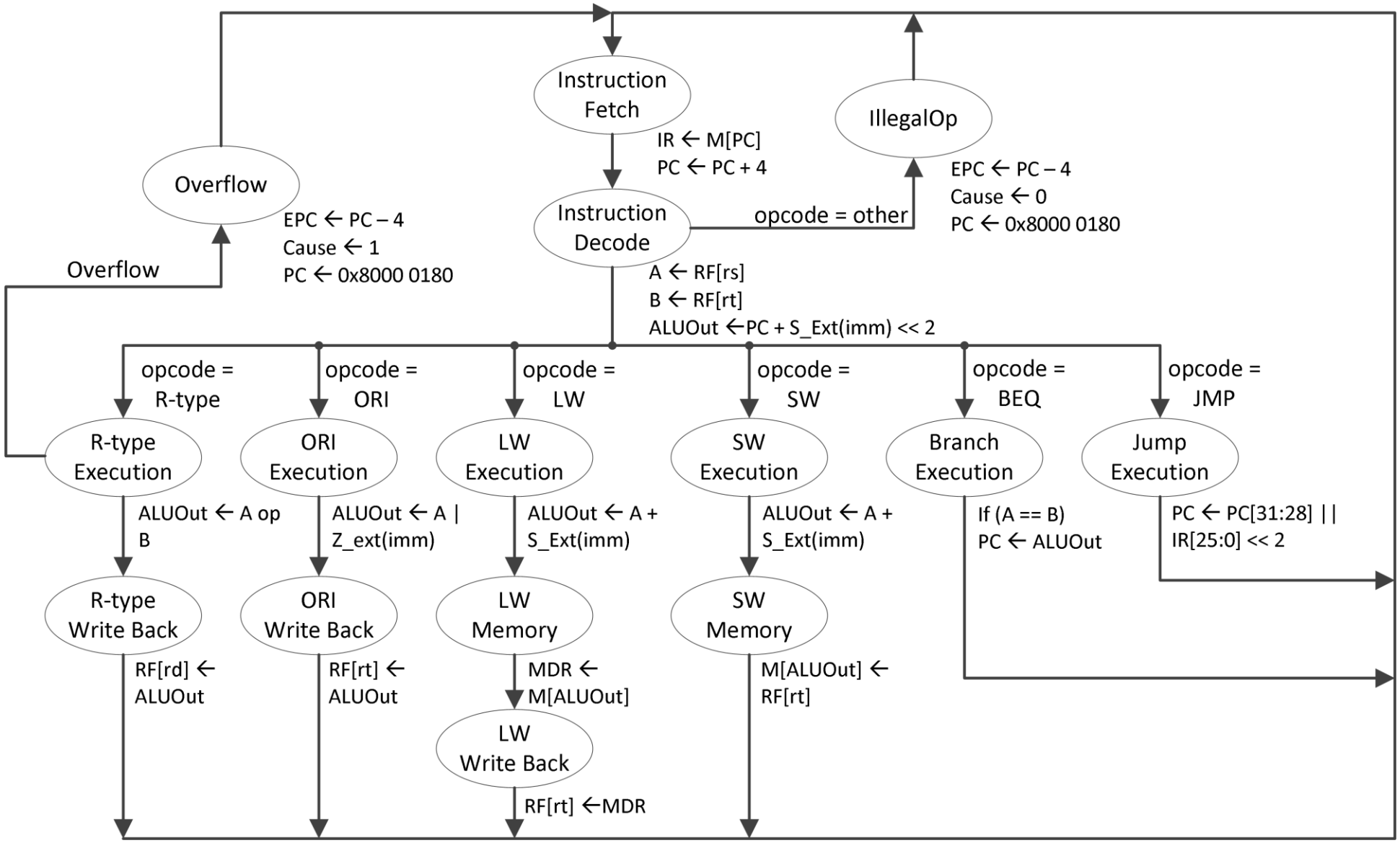
Proiectarea MIPS Multi-ciclu – Excepții

Tratarea excepţiei

- Două tehnici: Exception Program Counter (EPC)/Cause și Întreruperi Vectoriale
- Întreruperi vectoriale:
  - Fiecare excepție are asociată o adresă distinctă A_E
  - Excepție detectată → A_E corespunzătoare se scrie in PC
- EPC / Cause: MIPS
  - Registrul EPC (32 biți): memorează adresa instrucțiunii cauzatoare (în curs de execuţie): EPC ← (PC + 4 – 4) = PC (se folosește ALU pentru scădere cu 4)
  - Registrul Cause (32 biți) memorează un cod corespunzător cu tipul excepției
    - Instrucțiune nedefinită (ilegală): Cause ← 0
    - Depășire aritmetică: Cause ← 1
  - Excepție detectată
    - → Adresa instrucțiunii se salvează în EPC
    - → Codul excepției se salvează în Cause
- Rutina de tratare a excepției acționează conform Cause, apoi încearcă să repornească execuția de la instrucțiunea indicată de EPC

Proiectarea MIPS Multi-ciclu – Excepții

FSM1 se extinde cu 2 stări pentru tratarea Excepțiilor



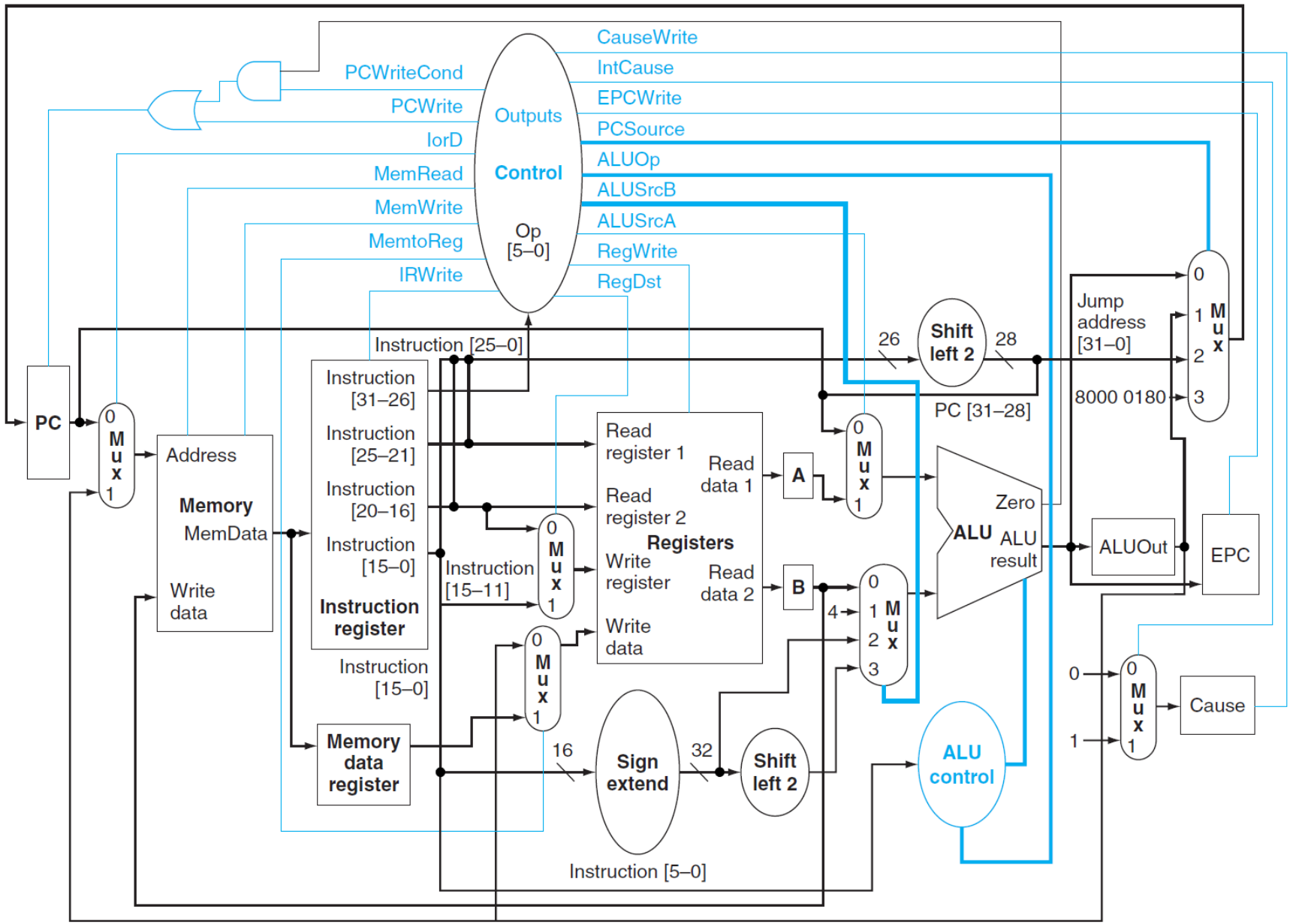
Proiectarea MIPS Multi-ciclu – Excepții

Modificările căilor de date MIPS pentru Excepții

- Regiștri noi – EPC si Cause (32 biți)
- Semnale de control noi pentru validarea scrierii – EpcWrite, CauseWrite
- Semnal de control nou / Mux înainte de Cause: IntCause
  - 0 pentru instrucțiuni nedefinite
  - 1 pentru depășire
- Intrare nouă pentru MUX-ul de la intrarea PC, PCsource = 11_2
  - Intrări PC:
    - PC+4,
    - Branch target address,
    - Jump address
    - Intrarea adițională: adresa rutinei de tratare $A_E = 8000\ 0180_{16}$ (pentru MIPS)

Notă: Detecţia de depăşire în ALU se presupune deja implementată

Proiectarea MIPS Multi-ciclu – Excepții



[1]

Căi de date / control MIPS Multi-ciclu cu tratarea excepțiilor

Probleme

1. Implementarea altor instrucțiuni. Vezi problemele din cursul 4!
2. Proiectați (prin evidențierea modificărilor / completărilor la FSM / microcod) unitatea de control (diferitele variante) pentru a suporta noua (noile) instrucțiuni.

Bibliografie

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”.