Proiectarea procesorului MIPS care operează în mai multe cicluri de ceas

– Curs 9–

Într-o implementare cu mai multe cicluri, fiecare pas al execuției va necesita o perioadă de ceas.

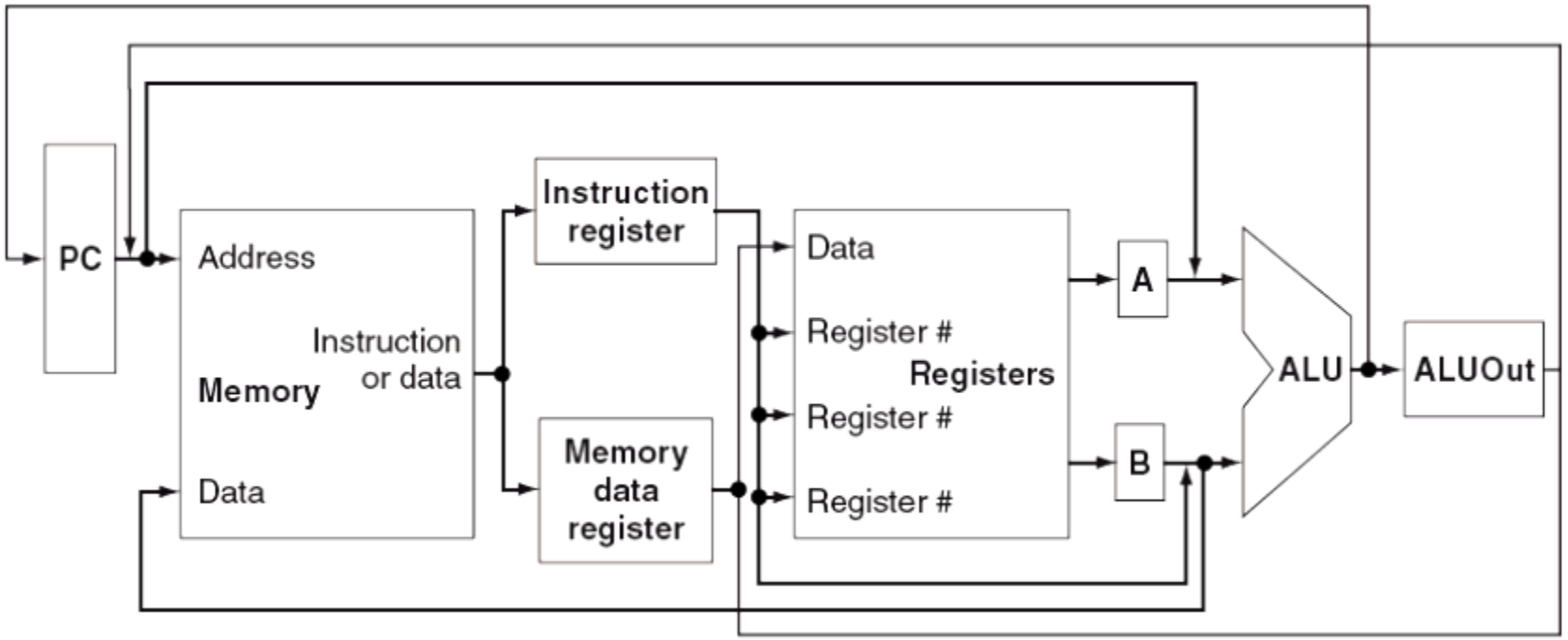
Acest tip de implementare permite unei unități funcționale să fie utilizată mai mult decât o singură dată pe instrucțiune, atâta timp cât este utilizată în cicluri de ceas diferite.

Această utilizare multiplă ⟶ reduce cantitatea de hardware.

Avantajele majore ale implementării cu mai multe cicluri de ceas:

1. Posibilitatea de a folosi aceleași unități funcționale de mai multe ori pe durata execuției unei singure instrucțiuni.
2. Posibilitatea de a permite instrucțiunilor să folosească un număr diferit de cicluri de timp.

Calea de date cu mai multe cicluri (versiune abstractă)



Diferențe față de versiunea cu un singur ciclu:

1. O singură unitate de memorie, atât pentru instrucțiuni cât și pentru date.
2. Un singur UAL
3. Sunt adăugate unul sau mai multe registre după fiecare unitate funcțională majoră pentru a păstra ieșirea acelei unități până când valoarea va fi folosită într-un ciclu ulterior de ceas

Elementele principale:

- O unitate de memorie partajată
- Un UAL partajat de instrucțiuni
- Conexiuni între unitățile folosite în comun.

Lărgirea multiplexoarelor și folosirea registrelor suplimentare
(IR – reg. de instrucțiuni; MDR – reg. datelor de memorie, A, B, ALUOut)

- La finalul unui ciclu de ceas toate datele care vor fi folosite în ciclurile următoare trebuie memorate în elemente de stare.
- Datele folosite de instrucțiunile următoare într-un ciclu ulterior de ceas vor fi memorate în elementele de stare vizibile programatorului: registrele generale, PC-ul sau memoria.
- Datele folosite de aceeași instrucțiune într-un ciclu ulterior de ceas trebuie memorate în registrele suplimentare .

Poziția reg. suplimentare sunt determinate de doi factori:

1. Ce unități combinaționale sunt cuprinse într-un ciclu de ceas
2. Care sunt datele necesare în ciclurile ulterioare ce implementează instrucțiunea

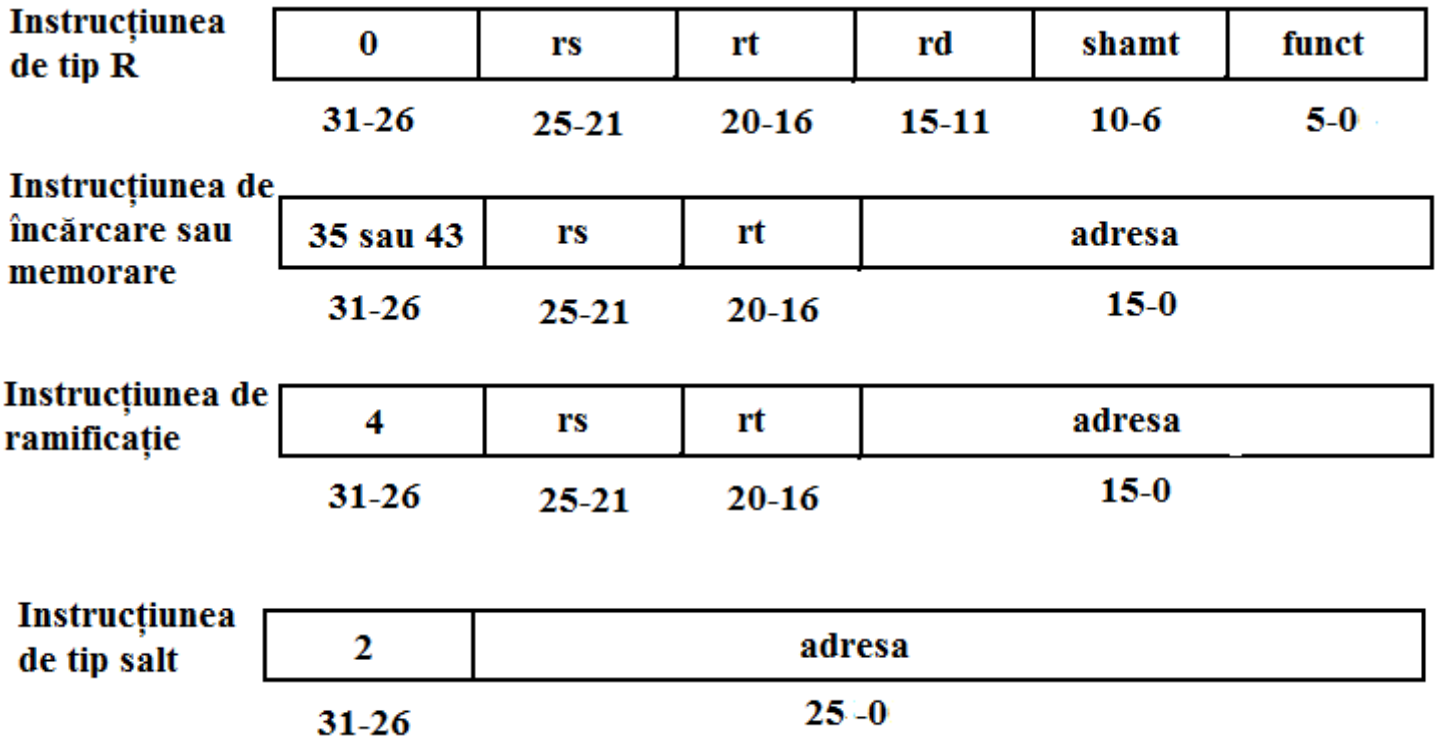
Proiectarea prezentată presupune că durata ciclului de ceas poate deservi cel mult:

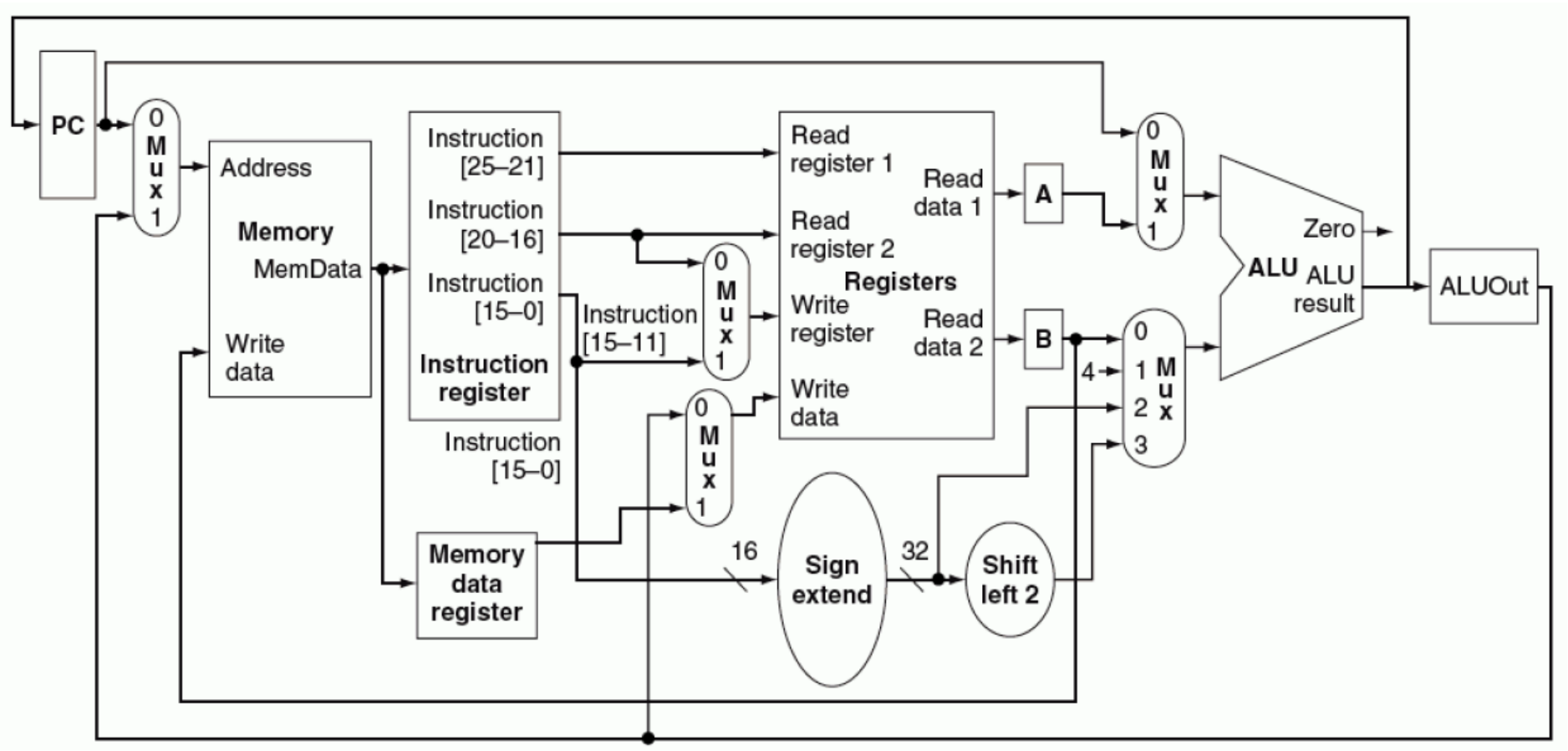
- un acces la memorie
- un acces la fișierul de registre – 2 citiri și o scriere
- o operație UAL

Astfel, orice dată produsă de aceste unități funcționale, trebuie salvată într-un registru suplimentar ca sa fie folosită într-un ciclu ulterior:

• IR salvează ieșirea memoriei de instrucțiuni și MDR salvează ieșirea datelor. (reg. separate pt. că ambele valori trebuie folosite în timpul aceluiași ciclu de ceas)
•A și B păstrează valorile operanzilor citiți din fișierul de registre.
•ALUOut păstrează ieșirea din UAL.

Remember!!! (din cursul trecut)





Folosim mai multe unități funcționale în comun, deci trebuie adăugate multiplexoare sau extindem unele deja existente.

Exp: Avem o singură memorie atât pentru date cât și pentru instrucțiuni – PC și OpUAL

Este folosit un singur UAL în loc de 3 ca în cazul implementării cu un singur ciclu de ceas. Așadar se fac următoarele schimbări:

• un multiplexor suplimentar adăugat primei intrări în UAL (decide între registrele A și PC)

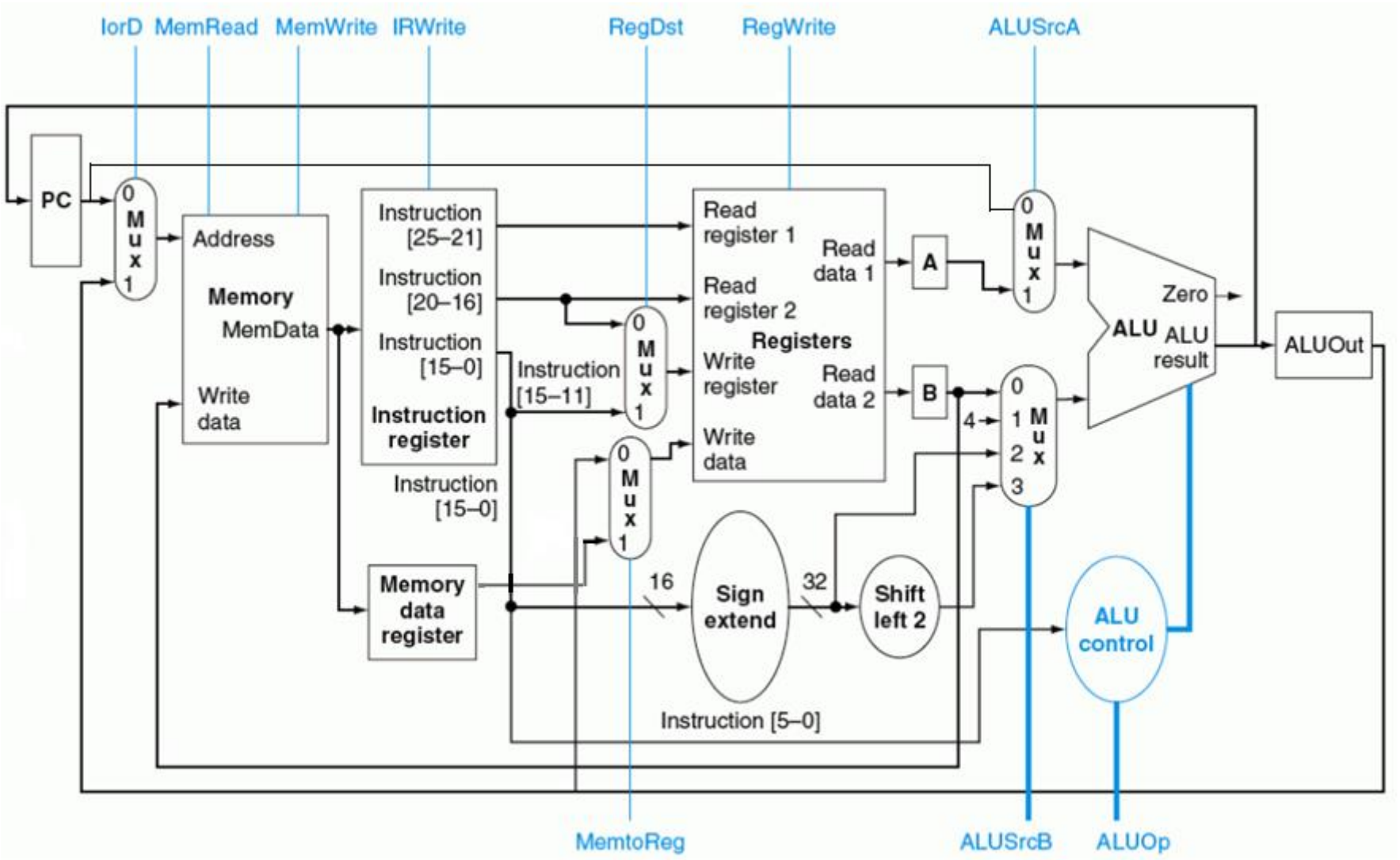
• multiplexorul celei de-a doua intrări în UAL din MUX2 se transformă în MUX4, cu intrările 4 (pt. incrementarea PC-ului cu 4) și câmpul deplasării cu semnul extins și mutat la stânga cu două poziții (folosit pt. calcularea adresei de ramificație).

Astfel s-a renunțat la 2 sumatoare și o unitate de memorie.

S-A OBȚINUT O REDUCERE SEMNIFICATIVĂ A COSTULUI HARDWARE-ULUI

Obs: *Încă nu s-au implementat ramificațiile și salturile!*

Calea de date și liniile de control



Obs: *A și B nu au nevoi de semnal de scriere pt. că datele lor vor fi citite în ciclul care urmează imediat după ce au fost scrise.*

Având în vedere salturile precum și ramificațiile există 3 surse posibile pentru PC:

1. ieșirea UAL care reprezintă valoarea PC + 4, în timpul extragerii instrucțiunii. Această valoare trebuie memorată direct în PC.
2. registrul ALUOut, unde este memorată adresa obiectiv pentru ramificație, după ce este calculată.
3. cei 26 de biți inferiori ai lui IR deplasați spre stânga cu 2 poziții și concatenați cu cei 4 biți superiori ai PC-ului incrementat – care este sursa pentru instrucțiunea de salt

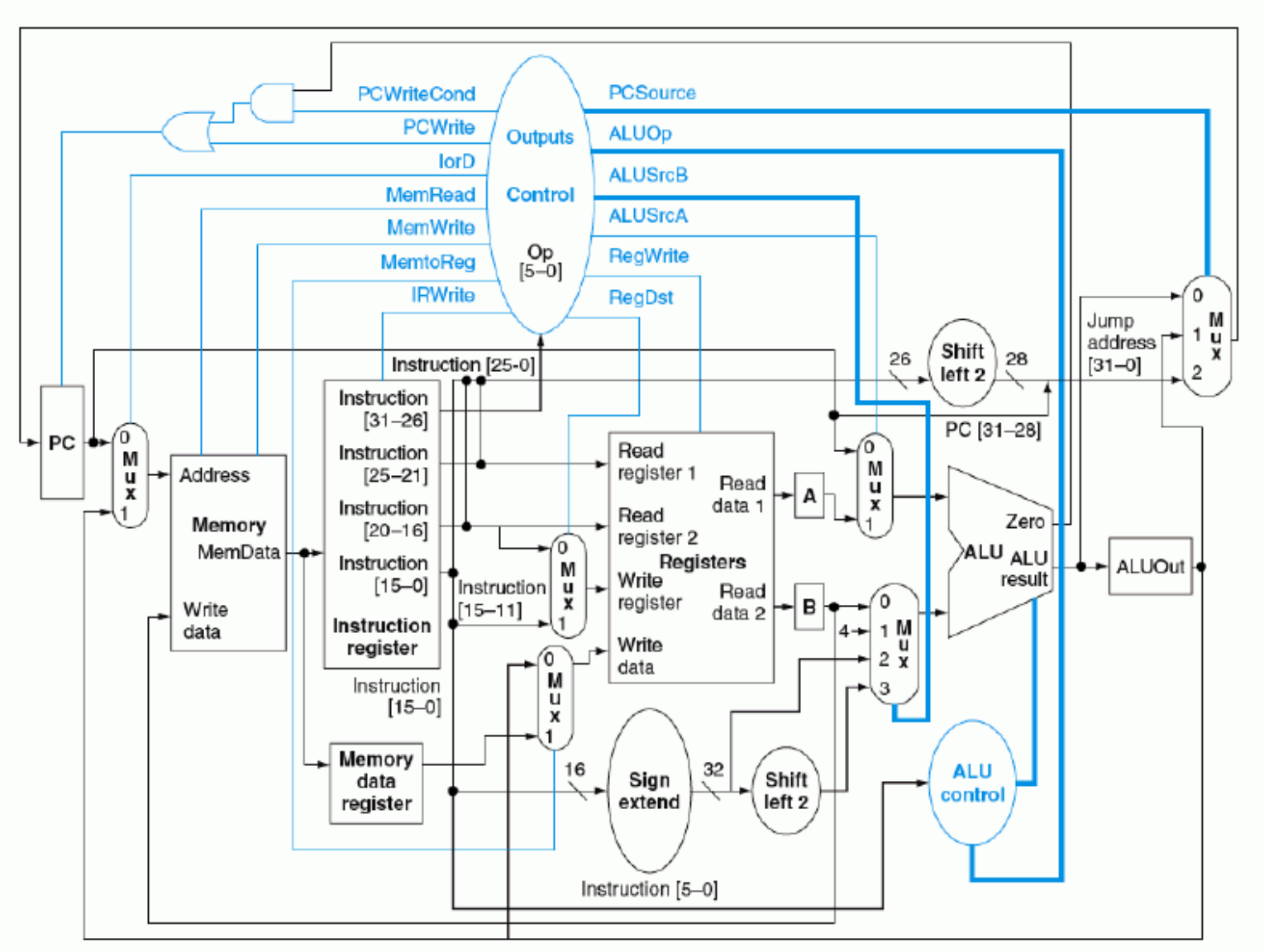
PC-ul va trebui scris atât necondiționat cât și condiționat !!!

În timpul unei incrementări normale sau al salturilor.

Dacă instrucțiunea este de ramificare condițională, PC-ul incrementat este înlocuit cu valoarea din ALUOut doar dacă cele două registre desemnate sunt egale.

Controlul necesită două semnale de scriere pentru PC: **PCWrite** și **PCWriteCond**

Calea de date completă și liniile de control necesare pentru implementarea execuției cu mai multe cicluri.



Pentru reducerea numărului de semnale care interconectează unitățile funcționale, se pot folosi **magistrale partajate**.

O magistrală partajată este un grup de linii care conectează diferite unități.

Exemplu: există 6 surse care sosesc la UAL, însă doar două dintre ele sunt necesare la un anumit moment. Astfel pot fi folosite doar două magistrale partajate. Trebuie să ne asigurăm că doar una dintre surse are controlul magistralei la un moment dat.

Acțiunea semnalelor de control de 1 bit

RegDst 0 - nr de destinație din fișierul de registre pentru *Write Register*
vine din câmpul *rt*
1 - vine din câmpul *rd*

RegWrite 0 - nici un efect
1 - reg universal selectat de *WriteRegister* este scris cu valoarea intrării *WriteData*

ALUSrcA 0 - dacă primul operand al UAL este PC
1 - dacă primul operand UAL vine din registrul A

MemRead 0 nici un efect
1 conținutul locației de memorie specificată de intrarea ADRESĂ e aplicat la ieșirea *MemData*

MemWrite 0 nici un efect
1 conținutul memoriei din locația specificată de intrarea ADRESĂ e înlocuit cu valoarea intrării *WriteData*

MemtoReg 0 - dacă valoarea pentru intrarea WriteData a fișierului de registre vine de la ALUOut

1 - dacă valoarea pentru intrarea WriteData a fișierului de registre vine de la Registrul datelor de memorie

IorD 0 - dacă adresa pentru unitatea de memorie este furnizată de PC

1 - dacă adresa pentru unitatea de memorie este furnizată de ALUOut

IRWrite 0 - nici un efect

1 - conținutul memoriei este scris in IR

PCWrite 0 - nici un efect

1 - PC este scris; sursa controlată de PCSource

PCWriteCond 0 - nici un efect

1 - PC este scris dacă ieșirea UAL-Zero este activă

Acțiunea semnalelor de control de 2 biți

ALUOp 00 - UAL efectuează o operație de adunare
01 - UAL efectuează de scădere
10 - Câmpul funct al instrucțiunii determină operația UAL

ALUSrcB 00 - a doua intrare pentru UAL vine de la registrul B
01 - a doua intrare pentru UAL este constanta 4
10 - a doua intrare pentru UAL sunt cei 16 biți inferiori ai IR, cu semnul extins
11 - a doua intrare pentru UAL sunt cei 16 biți inferiori ai IR cu semnul extins deplasați stânga cu 2 biți

PCSource 00 - Ieșirea UAL (PC+4) este trimisă la PC pentru scriere
01 – valoarea din registrul ALUOut este trimisă la PC pentru scriere
10 - adresa obiectiv pentru salt IR(25-0) deplasată stânga cu 2 biți și concatenată cu PC+4(31-28) este trimisă la PC pentru scriere

Descompunerea execuției instrucțiunii în cicluri de ceas

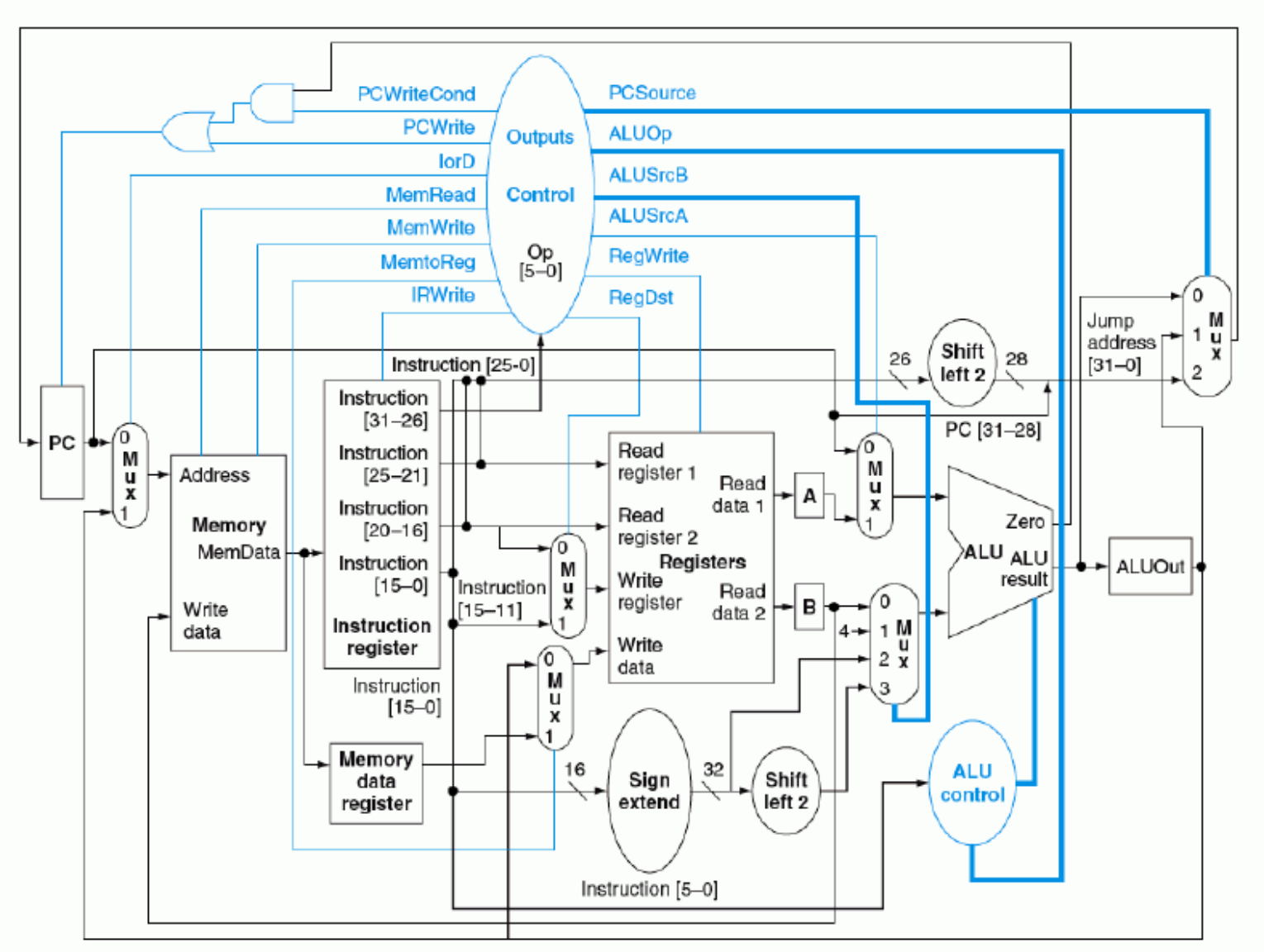
Scopul descompunerii execuției în cicluri de ceas trebuie să fie echilibrarea sarcinilor efectuate în fiecare ciclu astfel încât să se minimizeze durata ciclului de ceas.

Ex.: fiecare pas va fi limitat să conțină cel mult o operație UAL sau un acces la fișierul de registre sau un acces la memorie.

Cu această restricție, durata ciclului de ceas poate să fie tot atât de scurtă ca cea mai lungă dintre aceste operații.

Toate operațiile dintr-un pas se desfășoară în paralel pe durata unei perioade de ceas, în timp ce pașii succesivi operează în serie, în diferite cicluri de ceas. Limitarea la o operație UAL, un acces la memorie și un acces la fișierul de registre determină ce poate să aibă loc într-o perioadă de ceas.

Calea de date completă și liniile de control necesare pentru implementarea execuției cu mai multe cicluri.



1. Pasul de extragere a instrucțiunii

- Extrage instrucțiunea din memorie și calculează adresa următoarei instrucțiuni în secvență.

IR = Memorie(PC);
PC = PC+4

- Se trimite PC-ul ca adresă memoriei, se efectuează citirea instrucțiunii și scrierea ei în IR unde aceasta va fi păstrată. Incrementăm PC-ul cu 4. Trebuie să activăm semnalele de control *MemRead* și *IRWrite* și se stabilește *IorD* pe 0 pentru a selecta PC-ul ca sursă pentru adresă.

- În acest pas se face PC+4 ceea ce presupune poziționarea semnalului *ALUSrcA* pe 0 și a semnalului *ALUSrcB* pe 01 și *ALUOp* pe 00. La final adresa incrementată a instrucțiunii se memorează la loc în PC, ceea ce presupune stabilirea semnalului *PCWrite*.

- Incrementarea PC-ului precum și accesul la instrucțiunea din memorie pot să se desfășoare în paralel. Noua valoare a PC-ului nu este vizibilă până la următorul ciclu de ceas.

2. Decodificarea instrucțiunii și extragerea registrelor

- În acest pas (ca și în precedentul) nu se cunoaște încă instrucțiunea. Se va calcula prin intermediul UAL adresa obiectiv pentru ramificație, valoare ce va fi ignorată dacă operația nu este una de ramificație. Totuși execuția în avans are avantajul că numărul ciclilor de ceas de execuție descrește dacă trebuie executată instrucțiunea anticipată.

- Se accesează fișierul de registre pentru citirea registrelor *rs* și *rt* care sunt memorate în A și B. Trebuie să calculăm adresa obiectiv pentru ramificație și care va fi memorată în *ALUOut*. Deci semnalul *ALUSrcA* trebuie să fie stabilit pe 0, *ALUSrcB* pe valoarea 11, iar *ALUOp* pe 00.

- Accesul fișierului de registre și calcularea adresei obiectiv pentru ramificație se desfășoară în paralel. După acest ciclu de ceas, determinarea acțiunii care trebuie urmată poate depinde de conținutul instrucțiunii.

A = Reg(IR(25-21))
B = Reg (IR(20-16))
ALUOut = PC + (semn-extins(IR(15-0)) << 2);

3. Execuția, calculul adresei de memorie sau terminarea ramificației

Acesta este primul ciclu în care operarea căii de date este determinată de clasa instrucțiunii. În toate cazurile, UAL operează asupra operanzilor pregătiți în pașii anteriori.

- **Referință la memorie** – UAL adună operanzii pentru a forma adresa de memorie. Aceasta necesită stabilirea semnalelor *ALUSrcA* =1 și *ALUSrcB*=10. Semnalele *ALUOp* trebuie să fie stabilite pe 00 (determinând ALU să adune).

  IeșireUAL = A + semn-extins (IR(15-0))

- **Intrucțiune de tip R** – UAL efectuează operația specificată de codul funcțiunii asupra celor două valori citite din fișierul de registre în ciclul precedent. *ALUSrcA* =1 și *ALUSrcB*=00. Semnalele *ALUOp* trebuie să fie stabilite pe 10 deoarece determinarea semnalelor de control pentru UAL se face prin intermediul câmpului funct.

  IeșireUAL = A op B

- **Ramificație** – UAL este utilizat pentru efectuarea comparației dintre 2 registre citite anterior. Semnalul ZERO este folosit pentru a specifica dacă se va efectua ramificația sau nu. *ALUSrcA*=1 și *ALUSrcB*=00. Semnalele *ALUOp* trebuie să fie stabilite pe 01. Semnalul *PCWriteCond* va trebui să fie activat pentru a actualiza PC, dacă ieșirea *UAL-Zero* este activată. Prin stabilirea semnalului *PCSource* pe 01, valoarea scrisă în PC va veni din registrul ALUOut care păstrează adresa obiectiv a ramificației, calculată în ciclul precedent. Ultima valoare scrisă în PC va fi cea folosită pentru extragerea următoarei instrucțiuni.

  dacă (A==B) atunci PC=ALUOut

- **Salt** – PC este înlocuit cu valoarea adresei de salt. Semnalul *PCSource* este stabilit pentru a dirija adresa de salt la PC, iar *PCWrite* este activat pentru a scrie adresa de salt în PC.

  PC = PC(31-28) || IR(25-0) << 2

4. Pasul de acces la memorie sau de terminare a instrucțiunii de tip R

- O instrucțiune de încărcare/memorare accesează memoria, iar o instrucțiune aritmetică-logică își scrie rezultatul.
- Valoarea citită din memorie este scrisă în MDR de unde va fi folosită în ciclul următor.

- Referință de memorie:
  MDR = Memorie (ALUOut) sau **Memorie (ALUOut) = B**

Adresa folosită este cea memorată în ALUOut. Pentru memorare, operandul sursă este salvat în B. Semnalul *MemRead* trebuie activat pentru încărcare iar *MemWrite* pentru memorare. Pentru încărcare – semnalul *IorD*=1 pentru a forța ca adresa de memorie să vină de la UAL și nu de la PC. MDR se scrie la fiecare ciclu de ceas deci nu este necesară activarea nici unui semnal de control

- Instrucțiunea aritmetică-logică: Memorăm în registrul Rezultat conținutul registrului ALUOut, care corespunde cu ieșirea operării UAL-ului din ciclul precedent. Semnalul *RegDst* trebuie forțat pe 1 (folosim câmpul *rd* - biții 15-11) la selectarea intrării fișierului de registre. Semnalul *RegWrite*=1, iar *MemtoReg*=0.

Reg(IR(15-11)) = ALUOut

5. Pasul de terminare a citirii memoriei

- Se scrie valoarea citită în memorie.

Reg(IR(20-16)) = MDR

- Se scrie în fișierul de registre data încărcată, care în ciclul precedent a fost memorată în MDR. Pentru aceasta se stabilește *MemtoReg*=1 (pentru scrierea rezultatului în memorie), *RegWrite*=1 și se pune *RegDst*=0 deoarece dorim să alegem câmpul *rt* (20-16).