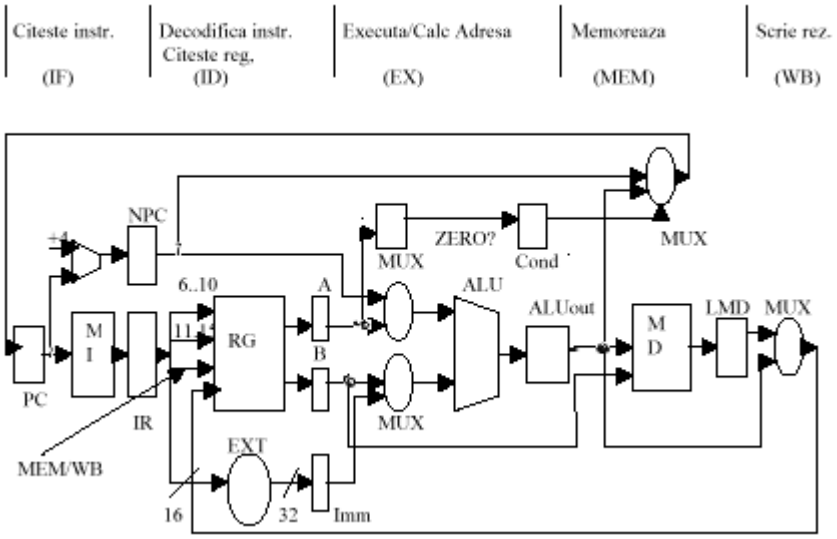
CURS 6. Proiectarea unui procesor care efectueaza o instructiune in mai multe cicluri de ceas

Se considera procesorul DLX in varianta cu doua memorii: pentru Instructiuni (MI) si pentru Date (MD). Instructiunile se efectueaza in mai multe cicluri de ceas.

O analiza a derularii in timp a instructiunilor va evidentia 5 faze posibile:

- Citeste instructiunea (Instruction Fetch – IF)
- Decodifica instructiunea/Citeste registrele generale (Instruction Decode – ID)
- Executa instructiunea/Calculeaza adresa (Instruction Execute – EXE)
- Memoreaza rezultatul in Memoria de date (Store Memory – MEM)
- Scrie rezultatul in registrul general (Write back - WB)

In scopul derularii instructiunilor in mai multe perioade de ceas, au fost introduse o serie de registre temporare, care nu fac parte din arhitectura si care nu sunt vizibile pentru programator.



Registrele temporare introduse sunt urmatoarele:

- NPC (Next Program Counter) pentru stocarea valorii urmatoare a Contorului Programului
- A, B si Imm – registre pentru stocarea operanzilor cititi din Registrele Generale si a operandului imediat dupa ce a fost prelucrat in circuitul de extindere EXT
- ALUout – registru pentru stocarea rezultatului obtinut in Unitatea Aritmetica Logica

- Cond – registru de un bit pentru stocarea rezultatului compararii cu 0 a continutului lui A
- LMD – registru in care se stocheaza data citita din Memoria de Date

Dupa cum se va vedea in derularea unora dintre instructiuni fazele MEM sau WB pot lipsi.

Etapele/Fazele derularii instructiunilor:

1. **Citire Instructiune. (IF**)

   IR ← MI[PC]; NPC ← PC + 4

2. **Decodificare Instructiune/ Citire Registre Generale (ID)**

   A ← RG[$IR_{6..10}$]; B ← RG[$IR_{10..15}$]; Imm ← ((IR_{16})^{16} ## $IR_{16...31}$)

3. **Executie/Calcul Adresa Efectiva. (EX)**

   In functie de codul operatiei UAL executa una dintre urmatoarele 4 operatii:

   3.1. Calcul Adresa Efectiva:

   UALies ← A + Imm

   3.2. Executie Instructiune R-R:

   UALies ← A op B

   3.3. Executie Instructiune R-I:

   UALies ←A op Imm

   3.4. Ramificare:

   UALies ← NPC + Imm; Cond ← (A op 0)

op constituie un operator relational, de exemplu "==", pentru instructiunea BEQZ. Arhitectura "citeste/memoreaza", pentru DLX permite combinarea operatiilor de executie a instructiunii si de calcul al adresei efective intr-un singur ciclu de ceas.

4. Accesul la Memoria de Date/Terminarea Ciclului de Ramificare (MEM)

In acest ciclul sunt active Incarcarile, Stocariile si Ramificarile.

4.1. Incarcari/Stocari – Accese la memorie

LMD ← MD[UALies] sau MD[UALies] ←B

4.2. Ramificare

if(cond) PC← UALies else PC ← NPC

5. Scrie rezultatul (WB).

5.1. Instructiuni UAL: R-R

$RG[IR_{16..20}] \leftarrow$ UALies

5.2. Instructiuni UAL: R-I

$RG[IR_{11..15}] \leftarrow$ UALies

5.3. Instructiune Incarca:

$RG[IR_{11..15}] \leftarrow$ LMD

Valoarea calculata in cadrul fiecarui ciclu este stocata la sfarsitul acestuia intr-un dispozitiv de memorare (memorie, registru general, PC, registre temporare: LMD, Imm, A, B, IR, NPC, ALUout, Cond) pentru a fi utilizata in ciclul urmator al instructiunii curente sau in cadrul instructiunii urmatoare. *Registrele temporare pastreaza valorile intre ciclurile de ceas ale unei instructiuni, in timp ce alte elemente de memorare, care sunt parti vizibile ale starii masinii, pastreaza valorile intre instructiunile succesive.*

Se poate observa ca PC este actualizat in cadrul ciclului MEM, in timp ce Registrul General destinatie este actualizat pe durata ciclului WB.

In aceasta implementare instructiunea de ramificare necesita 4 cicluri, iar celelalte instructiuni se executa in 5 cicluri. Daca procentul de aparitii ale instructiunii de ramificare este 12% se va obtine CPI = 4,88.

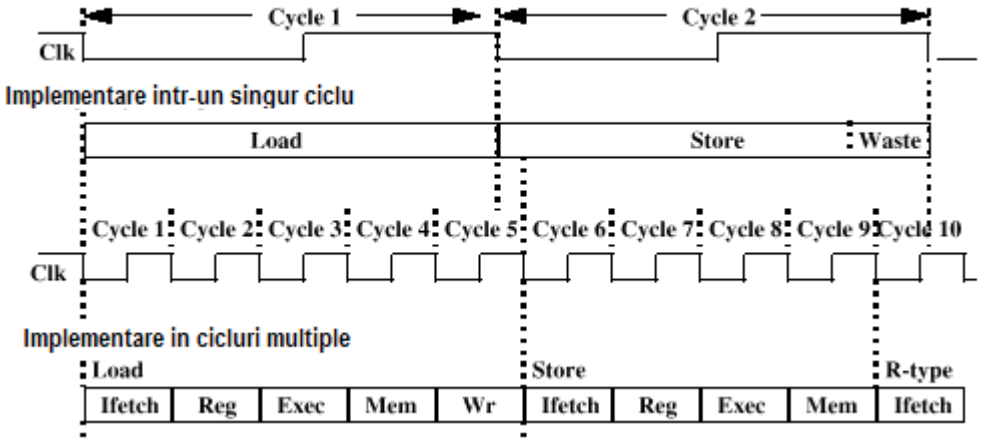
Aceasta implementare nu este optimala nici sub aspectul performantei, nici sub aspectul cantitatii de hardware folosit.

CPI poate fi redus si mai mult prin terminarea instructiunilor UAL in ciclul MEM, intrucat ele sunt inactive in acest ciclu. Instructiunile UAL au o pondere de 44%, ceea ce va asigura un CPI = 4,44

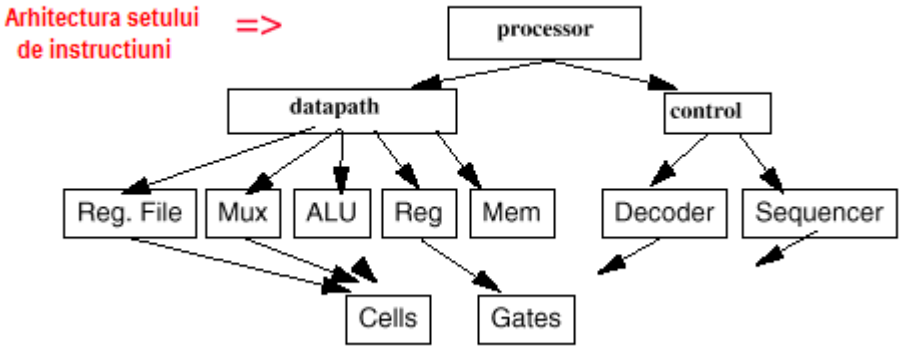
In continuare, incercarile de reducere a CPI vor conduce la aparitia mai multor activitati in cadrul fiecarui ciclu, ceea ce va impune cresterea perioadei ceasului. Se poate realiza un compromis intre CPI si perioada ceasului..

Se pot efectua, de asemenea, si unele reduceri de hardware:

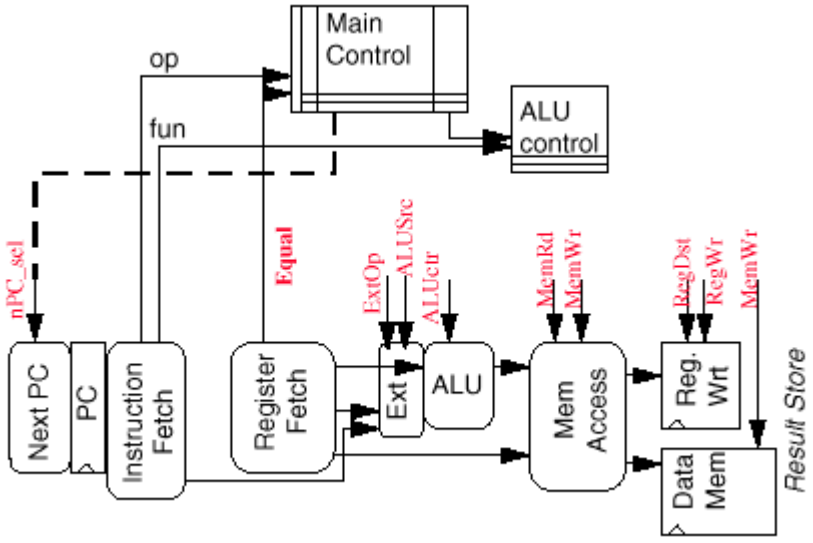
- cele doua UAL-uri se pot reduce la una singura, intrucat nu sunt active in acelasi ciclu,
- se poate utiliza o singura memorie pentru instructiuni si pentru date.



- **Proiectarea unui procesor se poate realiza in mai multe moduri:**
- **Bottom-up:**
  - se asambleaza componentele in cadrul unei tehnologii date, pentru a stabili sincronizarea/temporizarea critica
- **Top-down**
  - se specifica comportamentul componentelor pe baza cerintelor furnizate la nivel ridicat
- **Rafinare iterativa:**
  - se elaboreaza o solutie partiala, care se extinde si se imbunatateste

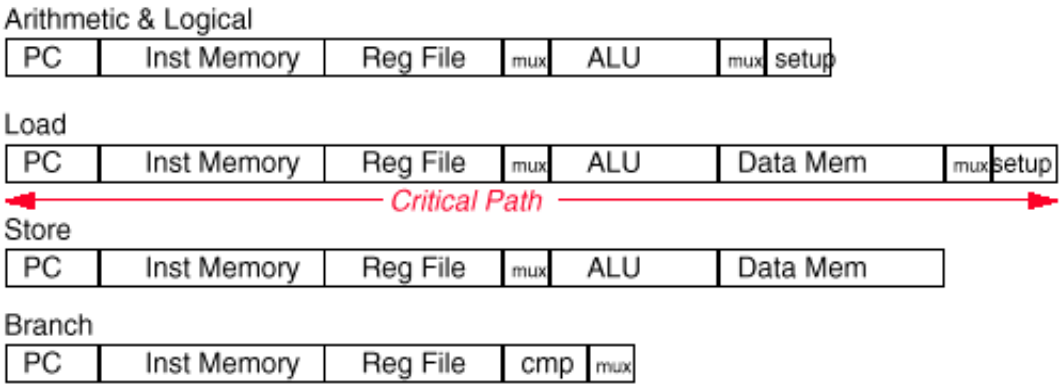


Reprezentarea abstracta a procesorului care opereaza intr-un singur ciclu de ceas:



Se prezinta ca un automat secvential cu un numar finit de stari, starea fiind furnizata de catre PC.

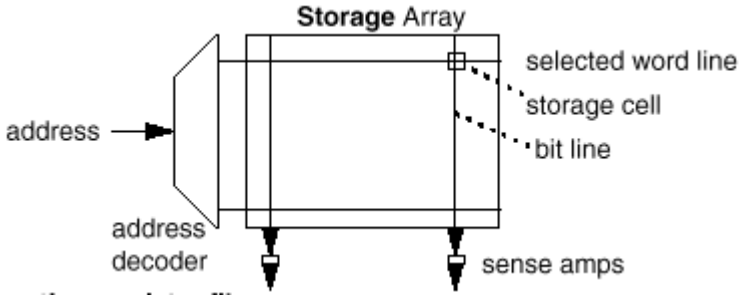
Duratele derularii diferitelor instructiuni in implementarea procesorului, care opereaza intr-un singur ciclu de ceas:



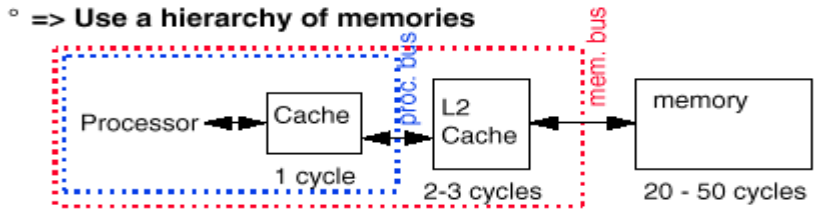
- Ciclul instructiunii este mare.
- Toate instructiunile se vor derula in acelsi interval de timp ca si cea mai lenta.
- Memoria reala prezinta mai multe probleme decat memoria ideala.

Timpul de Acces la Memorie.

- Tehnologia si capacitatea determina timpul de acces: memoriile de capacitate mica au timp mic de acces, memoriile de capacitate mare au timp mare de acces.

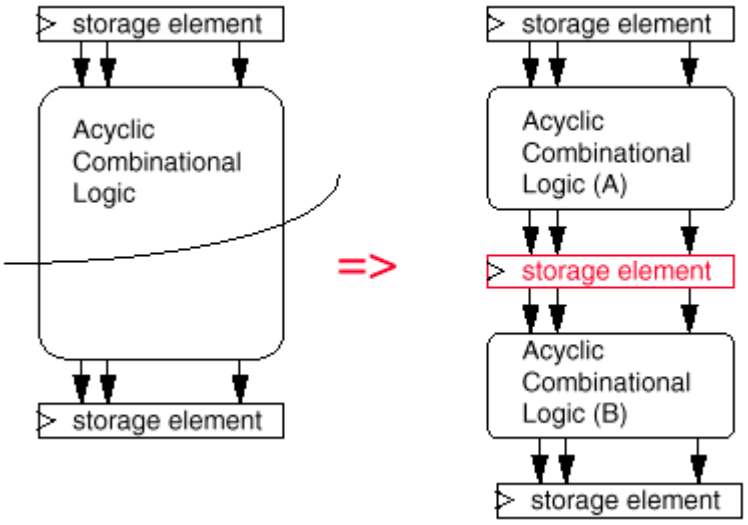


Comparatie intre fisierele/tablourile de registre generale si memorii:



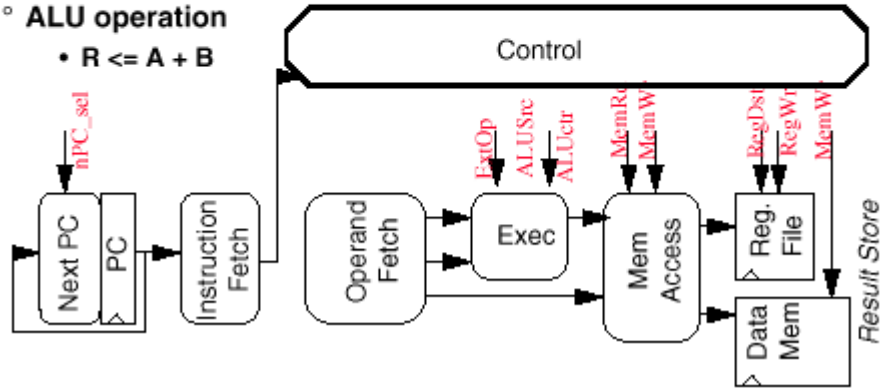
Reducerea duratei ciclului:

- se sectioneaza graful de dependenta combinational si se insereaza registre/latch-uri
- se efectueaza activitatea dintr-un ciclu lung in doua cicluri mai scurte:



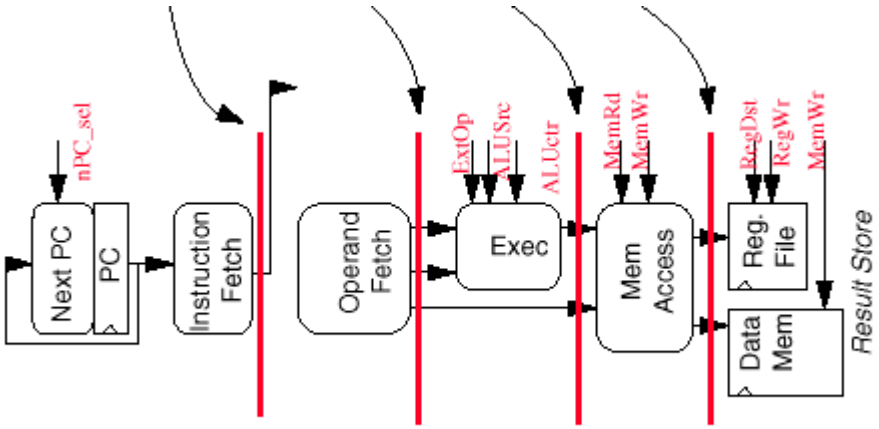
Limitarile principale ale procesorului care opereaza intr-un singur ciclu de ceas:

- Next address logic
  - PC <= branch ? PC + offset : PC + 4
- Instruction Fetch
  - InstructionReg <= Mem[PC]
- Register Access
  - A <= R[rs]
- ALU operation
  - R <= A + B

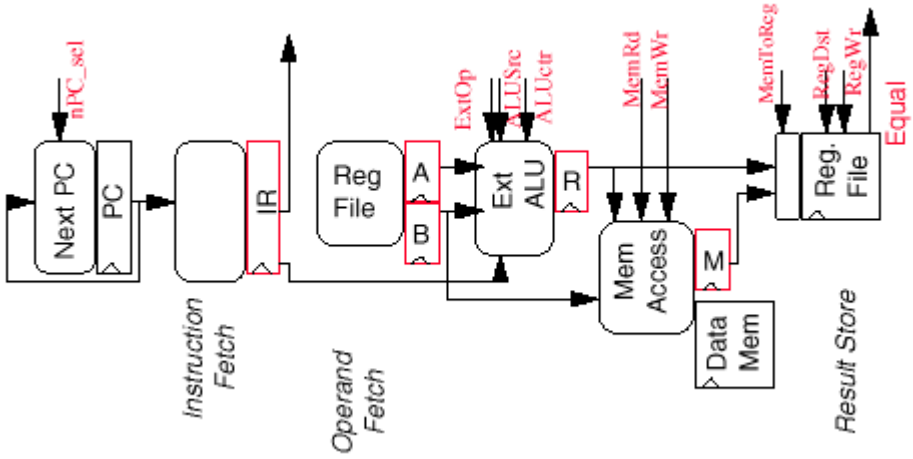


Sectionarea Unitatii de Executie avand CPI = 1

- Se introduc registre intre cei mai mici pasi



Exemplu de Unitate de executie care opereaza in mai multe cicluri de ceas



Care este “drumul critic”?

Proiectarea Procesorului Pas -cu Pas (reluare):

1. ISA => Transferurile Logice intre Registre (RTL)
2. Componentele Unitatii de Executie
3. RTL + Componente => Unitatea de Executie
4. RTL + Componente => Transferuri intre Registre fizice (tratat in continuare)
5. Transferuri intre Registre fizice => Comanda

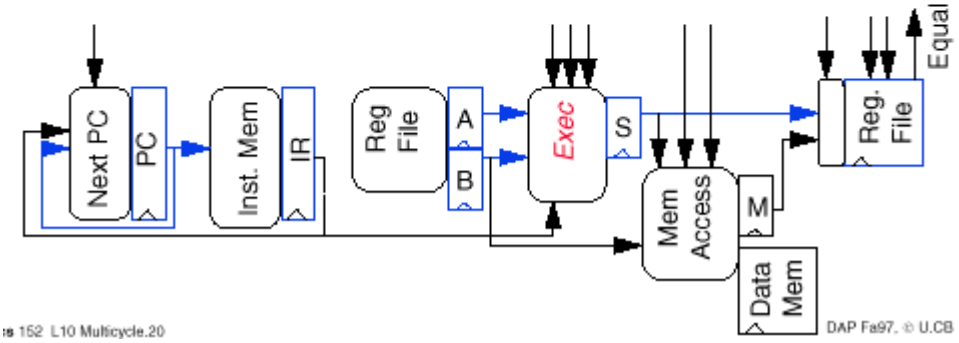
Pasul 4. Tipul R (add,sub,…)

° Logical Register Transfer

inst	Logical Register Transfers
ADDU	R[rd] <– R[rs] + R[rt]; PC <– PC + 4

° Physical Register Transfers

inst	Physical Register Transfers
	IR <– MEM[pc]
ADDU	A<– R[rs]; B <– R[rt]
	S <– A + B
	R[rd] <– S; PC <– PC + 4



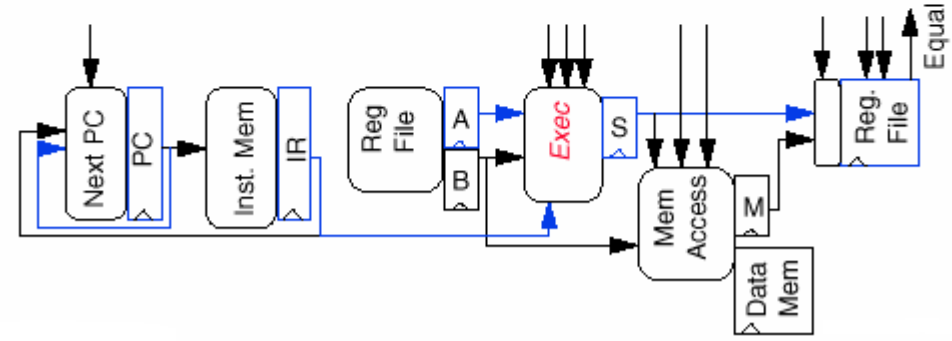
Pasul 4. Tipul I (andI, orI,…)

° Logical Register Transfer

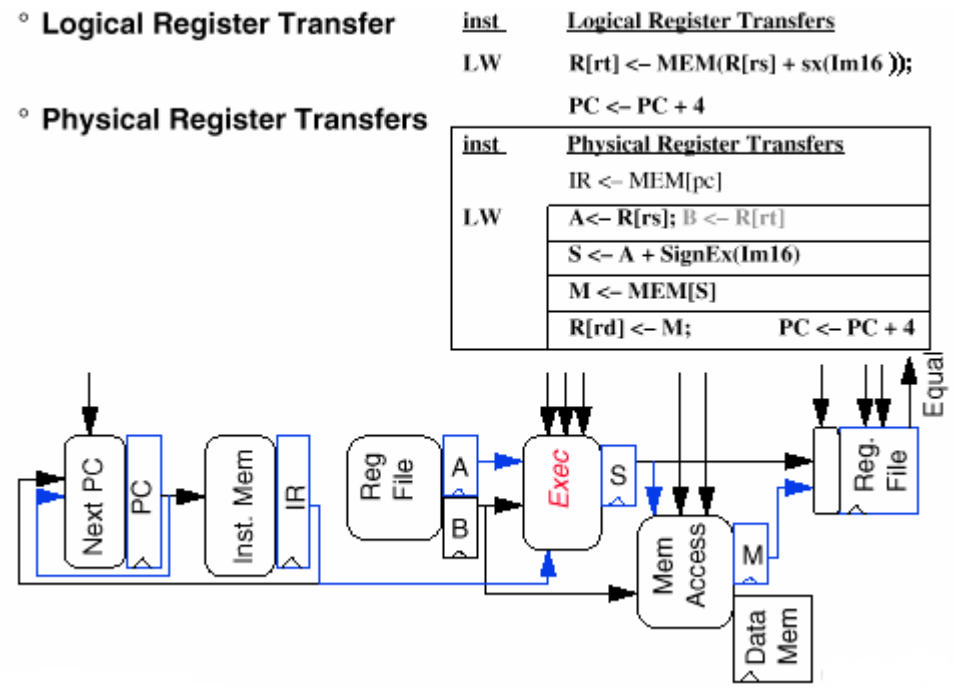
inst	Logical Register Transfers
ORI	R[rt] <– R[rs] OR zx(Im16); PC <– PC + 4

° Physical Register Transfers

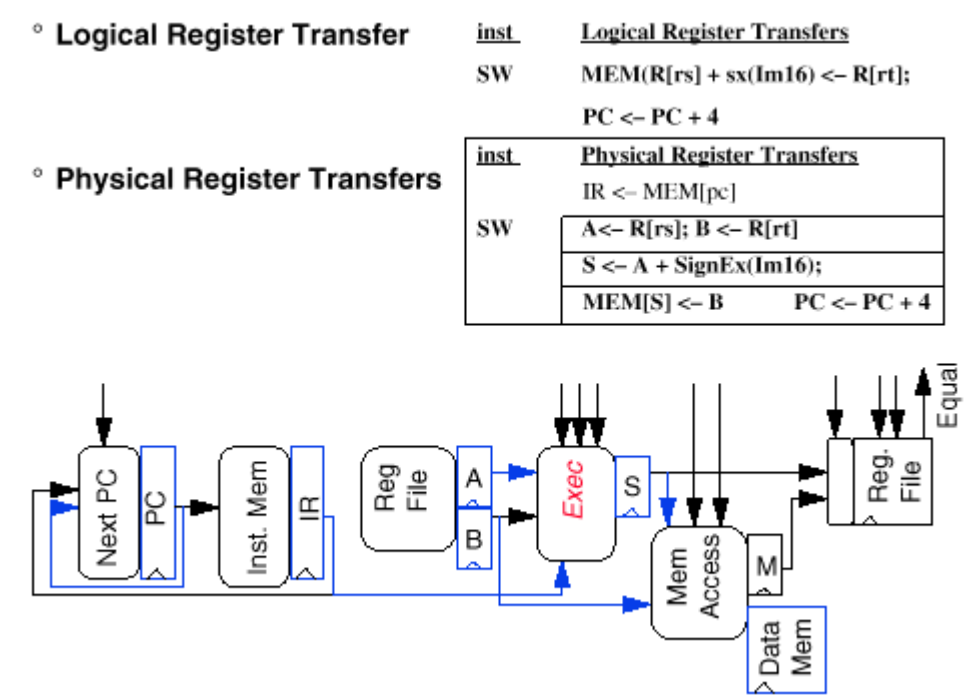
inst	Physical Register Transfers
	IR <– MEM[pc]
ORI	A<– R[rs]; B <– R[rt]
	S <– A *or* ZeroExt(Im16)
	R[rt] <– S; PC <– PC + 4



Pasul 4. Incarca (Load)



Pasul 4. Memoreaza(Store)



Pasul 4. Ramificare (Branch)

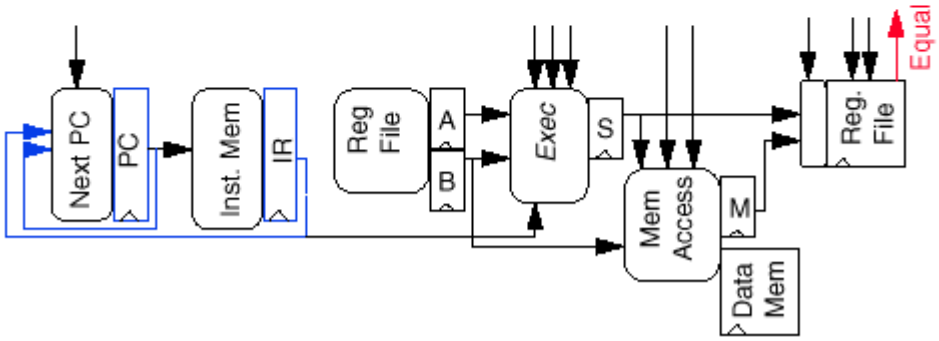
° Logical Register Transfer

inst	Logical Register Transfers
BEQ	if R[rs] == R[rt]
	then PC <= PC + sx(Im16) || 00
	else PC <= PC + 4

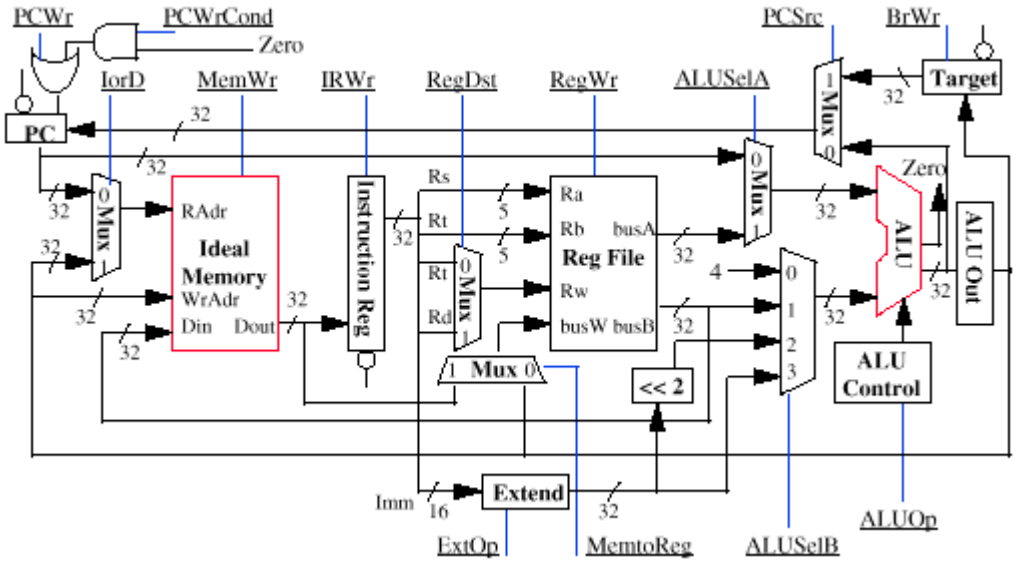
° Physical Register Transfers

inst	Physical Register Transfers
	IR <– MEM[pc]
BEQ|$\overline{Eq}$	PC <– PC + 4

inst	Physical Register Transfers
	IR <– MEM[pc]
BEQ|Eq	PC <– PC + sx(Im16) || 00



O reprezentarea alternativa a Unitatii de Executie

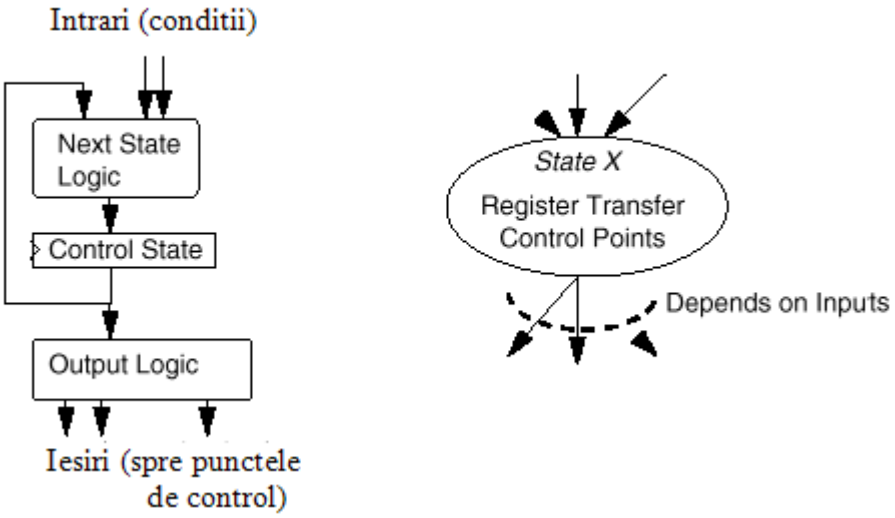


Se efectueaza urmatoarele reduceri de hardware:

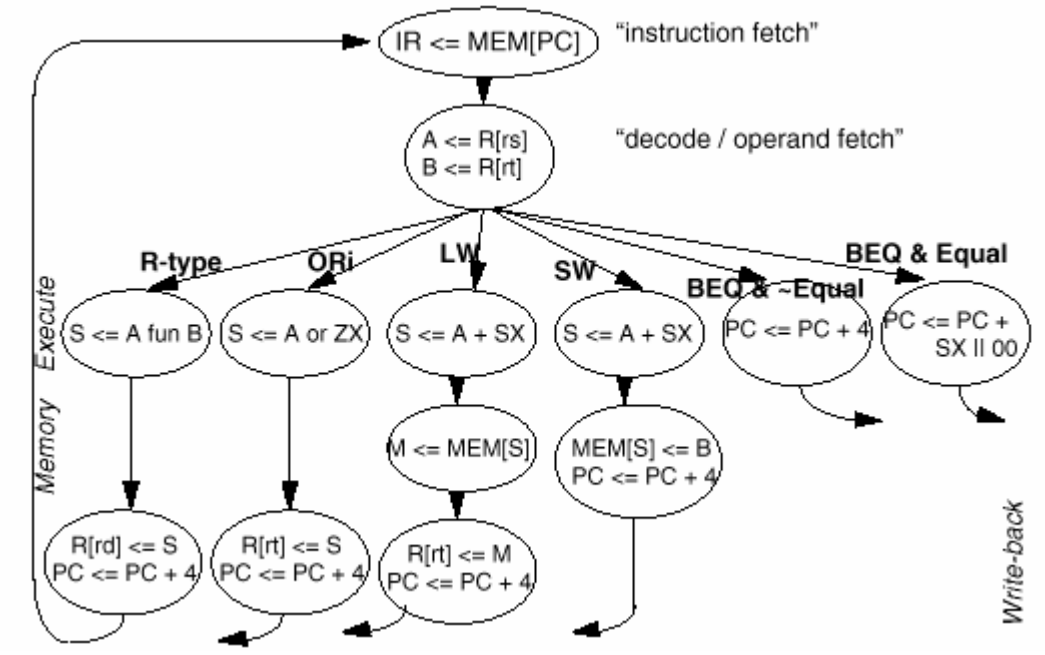
- **o singura memorie**
- **un singur sumator**

Modelul pentru comanda:

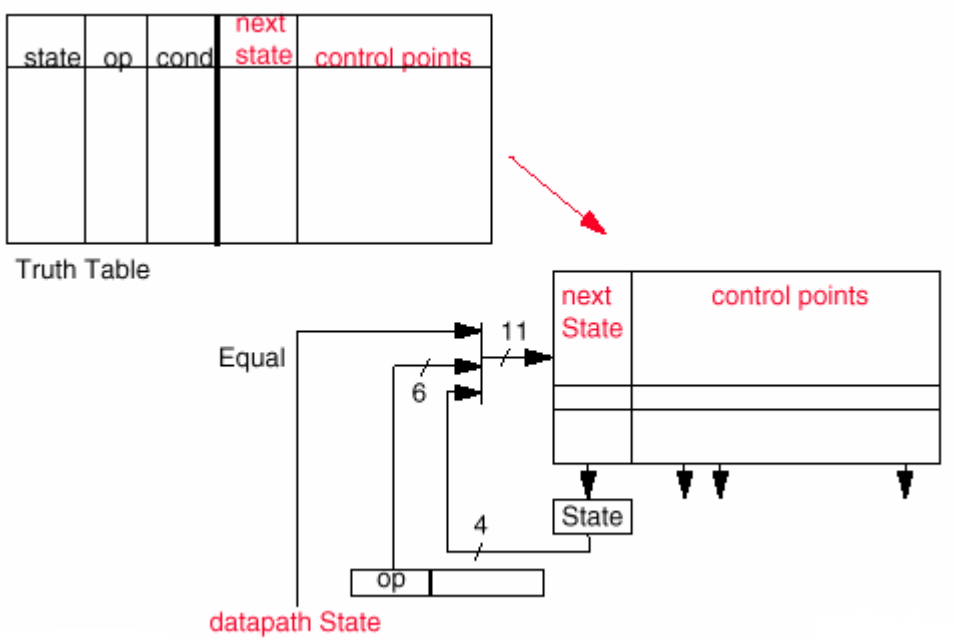
- Starea specifica punctele de comanda pentru Transferurile intre Registre.
- Transferurile se termina in momentul parasirii starii (pe acelasi front negativ).



Pasul 4. Specificatiile comenzii pentru procesorul care opereaza in mai multe cicluri de ceas

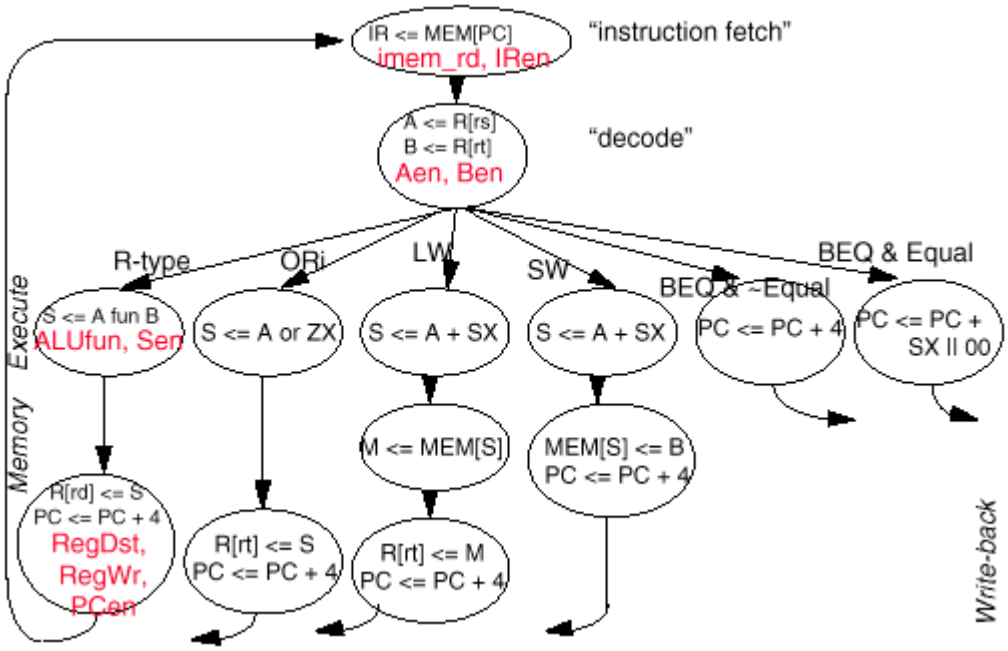


Comanda conventionala. Automatul cu Numar Finit de Stari

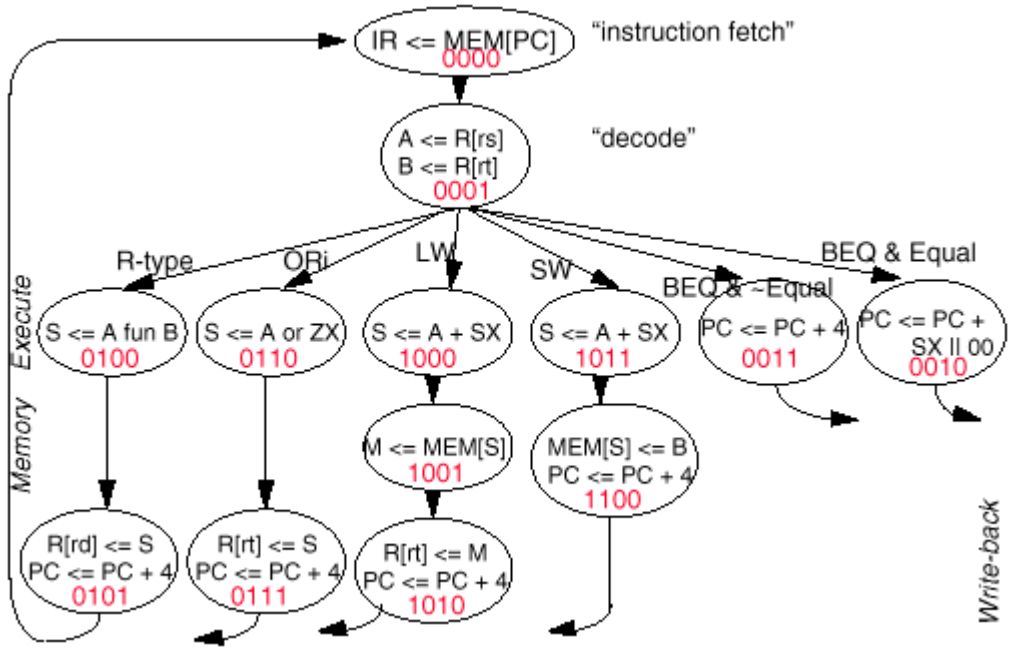


Pasul 5: Unitatea de Executie + Diagrama de Stare => Unitatea de Comanda.

- Translateaza Transferurile intre Registre, in Puncte de Comanda.
- Asigneaza Starile.
- Construieste Unitatea de Comanda.



Asignarea Starilor:



Specificarea detaliata a comenzii:

	State	Op field	Eq	Next	IR	PC en sel	Ops A B	Exec Ex Sr ALU S	Mem R W M	Write-Back M-R Wr Dst
	0000	??????	?	0001	1					
	0001	BEQ	0	0011			1 1	*-all same in Moore machine*		
	0001	BEQ	1	0010			1 1			
	0001	R-type	x	0100			1 1			
	0001	orI	x	0110			1 1			
	0001	LW	x	1000			1 1			
	0001	SW	x	1011			1 1			
	0010	xxxxxx	x	0000		1 1				
	0011	xxxxxx	x	0000		1 0				
R:	0100	xxxxxx	x	0101				0 1 fun 1		
	0101	xxxxxx	x	0000		1 0				0 1 1
ORi:	0110	xxxxxx	x	0111				0 0 or 1		
	0111	xxxxxx	x	0000		1 0				0 1 0
LW:	1000	xxxxxx	x	1001				1 0 add 1		
	1001	xxxxxx	x	1010					1 0 0	
	1010	xxxxxx	x	0000		1 0				1 1 0
SW:	1011	xxxxxx	x	1100				1 0 add 1		
	1100	xxxxxx	x	0000		1 0			0 1	

Evaluarea Performantei

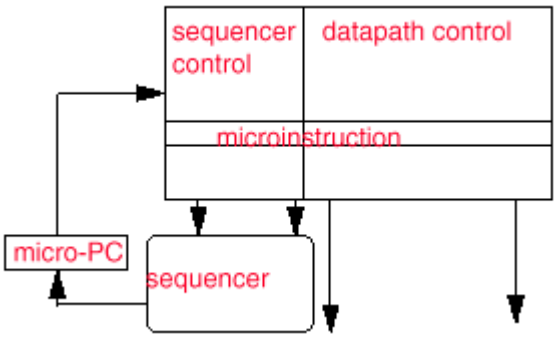
- Care este CPI mediu?
  - Diagrama de stari furnizeaza CPI pentru fiecare tip de instructiune.
  - Sarcina de lucru (workload) furnizeaza frecventa pentru fiecare tip.

Type	CPI_i for type	Frequency	CPI_i x $freql_i$
Arith/Logic	4	40%	1.6
Load	5	30%	1.5
Store	4	10%	0.4
branch	3	20%	0.6
		Average CPI:4.1	

Proiectarea Unitatii de Comanda

- Diagramele de Stare definesc Unitatea de Comanda pentru un Set de Instructiuni ale Procesorului si sunt in cea mai mare masura structurate;
- Se utilizeaza aceasta structura pentru a construi un microsecventiator simplu;
- Controlul se reduce la programarea acestui dispozitiv extrem de simplu.

Microprogramare



Exemplu: Contor – Jump

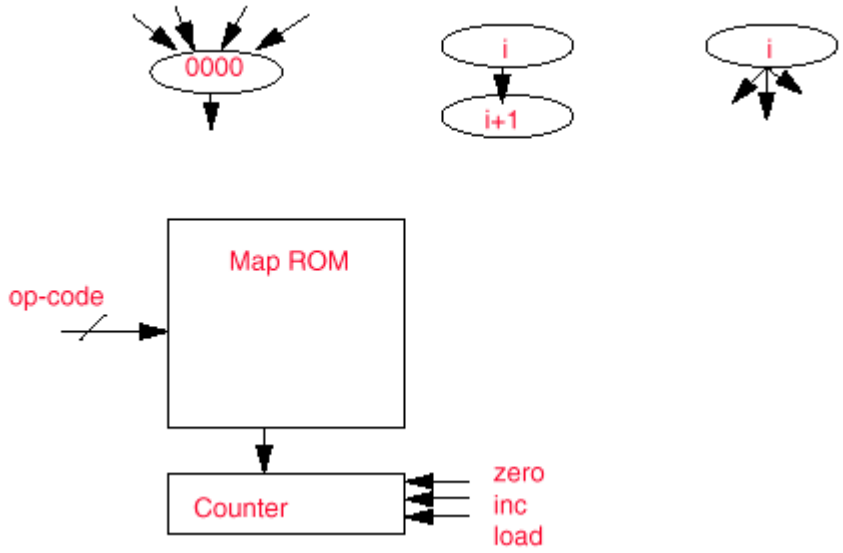
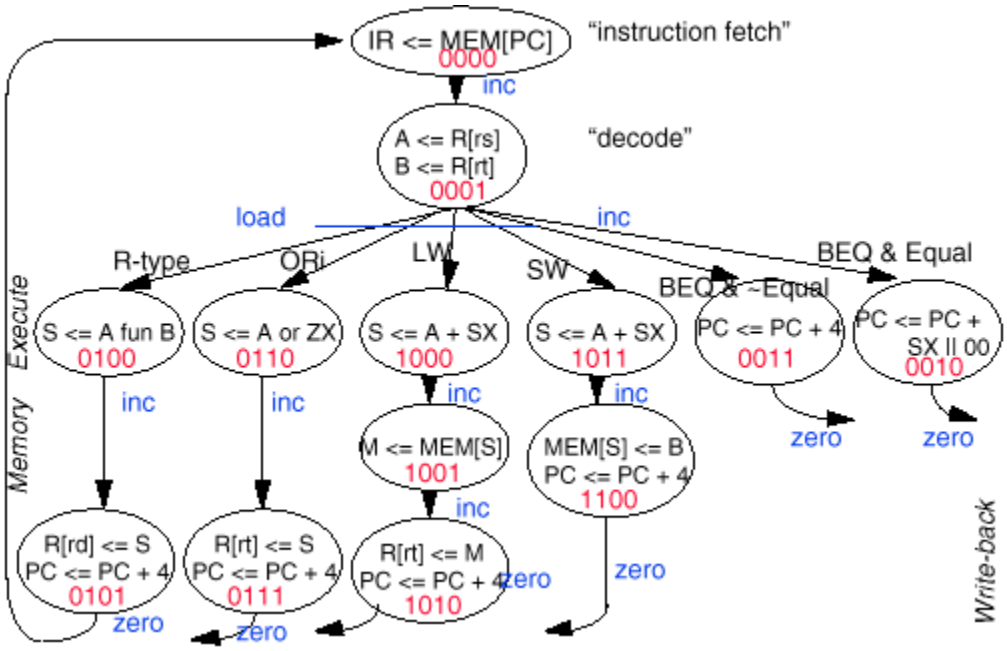
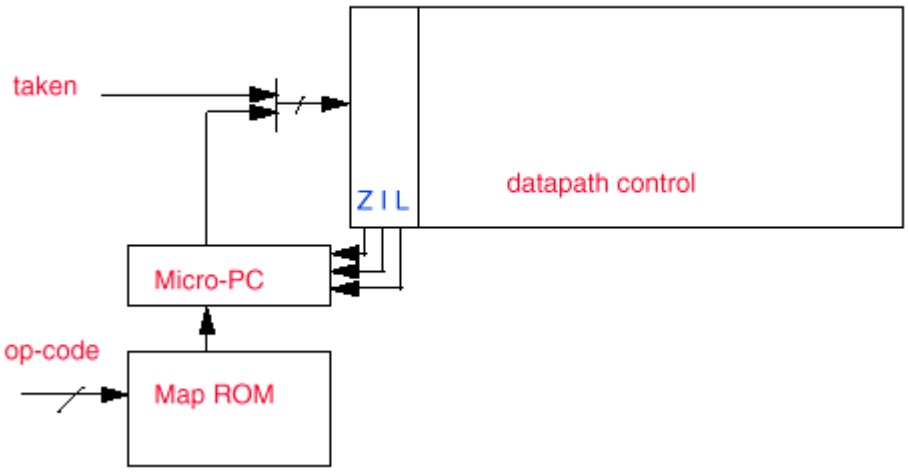


Diagrama de Stari (Contor – Jump)



Microsecventiatorul



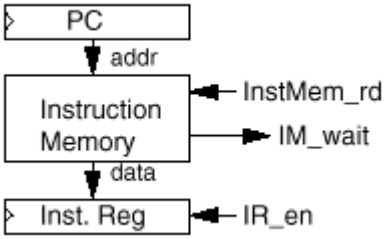
Specificatiile Controlului microprogramat

	μPC	Taken	Next	IR	PC en sel	Ops A B	Exec Ex Sr ALU S	Mem R W M	Write-Back M-R Wr Dst
	0000	?	inc	1					
	0001	0	load						
	0001	1	inc						
	0010	x	zero		1 1				
BEQ	0011	x	zero		1 0				
R:	0100	x	inc				0 1 fun 1		
	0101	x	zero		1 0				0 1 1
ORi:	0110	x	inc				0 0 or 1		
	0111	x	zero		1 0				0 1 0
LW:	1000	x	inc				1 0 add 1		
	1001	x	inc					1 0 0	
	1010	x	zero		1 0				1 1 0
SW:	1011	x	inc				1 0 add 1		
	1100	x	zero		1 0			0 1	

Maparea in ROM

R-type	000000	0100
BEQ	000100	0011
ori	001101	0110
LW	100011	1000
SW	101011	1011

Exemplu: Controlul Memoriei



Controlul are in vedere o memorie reala (non-ideala)

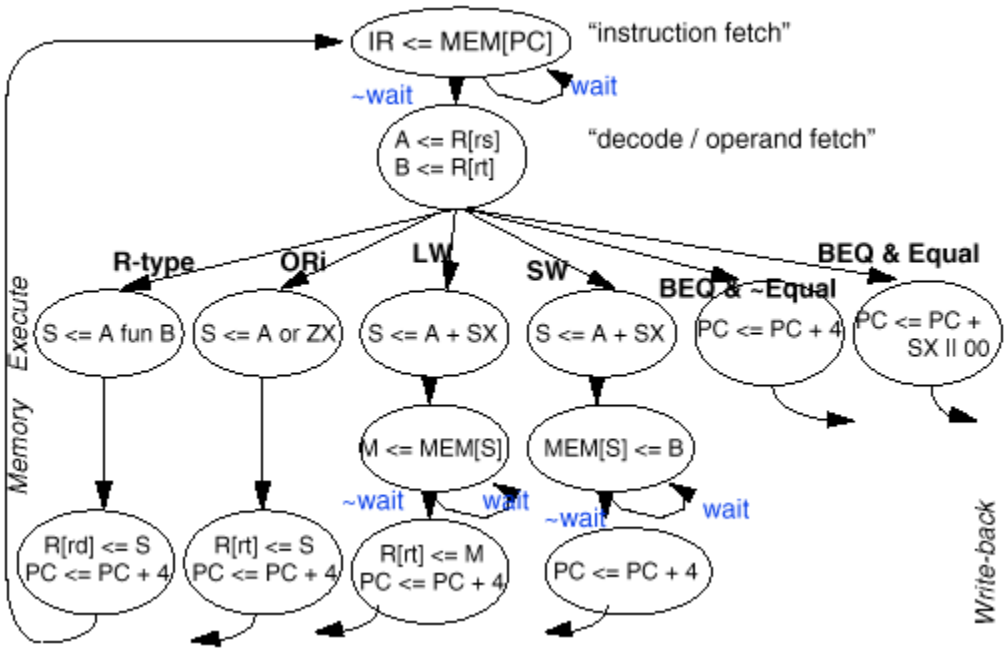
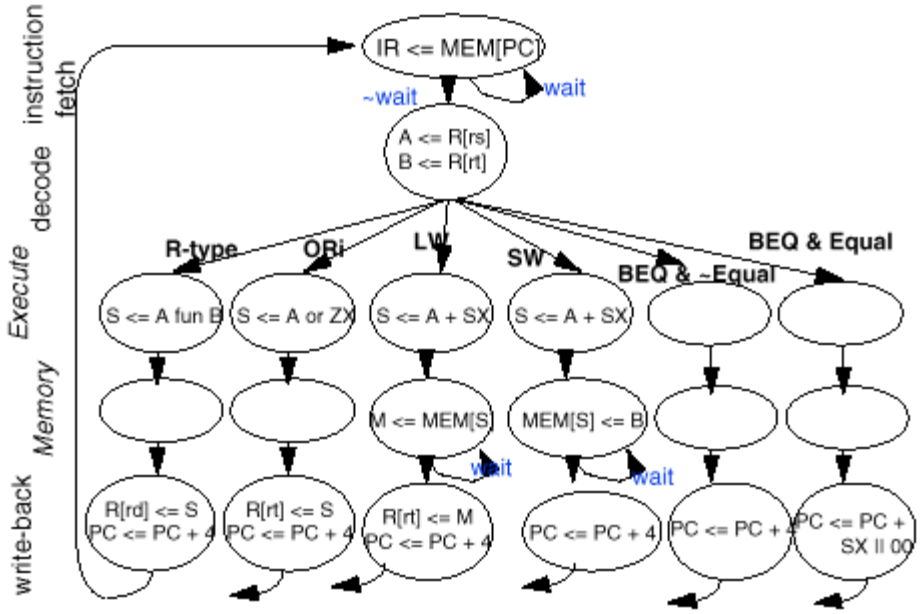
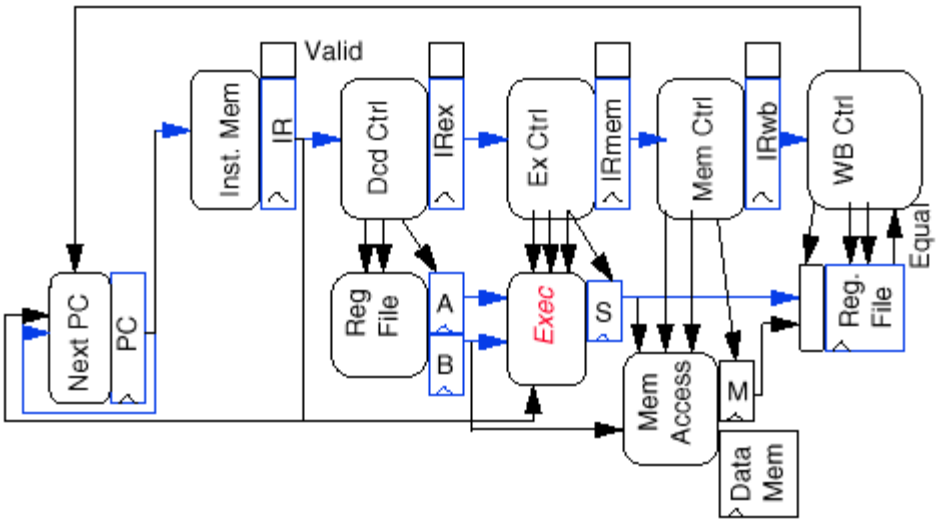


Diagrama de Stari – Comanda Reala



Comanda Timp – Stare

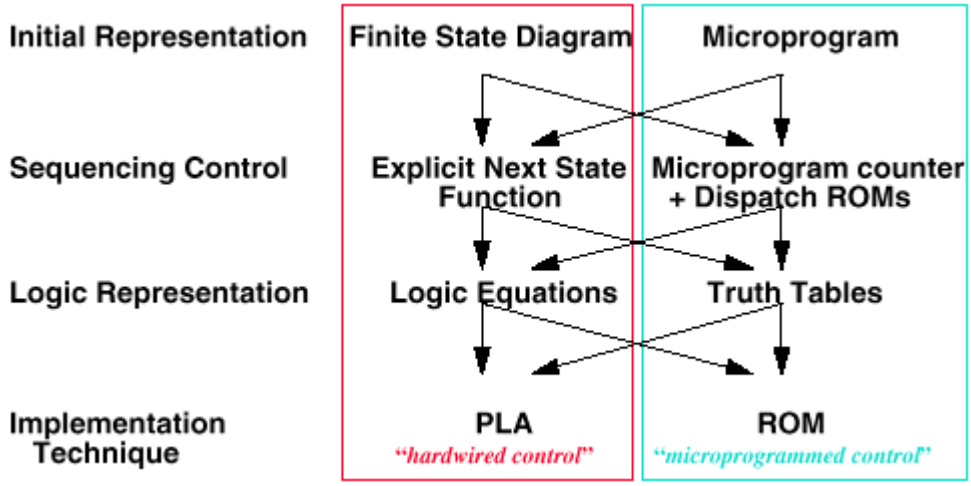
- Decodificare locala si comanda la fiecare etaj



Aspecte de baza privind Comanda.

- Comanda poate fi proiectata folosind una dintre reprezentarile initiale;
- Alegerea controlului secventei si a modului in care este reprezentata logica se pot efectua independent.

Comanda poate fi implementata cu una dintre metodele date, folosind o tehnica de logica structurata.



Sumar:

- **Dezavantajele Procesorului care Opereaza intr-un Singur Ciclu**
  - Durata mare a ciclului:
    - Durata ciclului este mult prea mare pentru majoritatea instructiunilor, cu exceptia instructiunii Incarca.
- **Procesorul care opereaza in mai multe Cicluri de Ceas**
  - Se sparg instructiunile in pasi mai mici;
  - Se executa fiecare pas (in locul intregii instructiuni) intr-un ciclu.
- **Se partitioneaza Unitatea de Executie in segmente de dimensiuni egale pentru a minimiza durata ciclului:**
  - Se accepta circa 10 niveluri logice intre registre/latch-uri
- **Se urmareste aceeasi metoda, in 5 pasi, pentru proiectarea procesorului "real";**
- **Comanda este specificata printr-o diagrama finita de stari;**
- **Diagramele de stari specializate se "transpun" usor intr-un microsecventiator:**

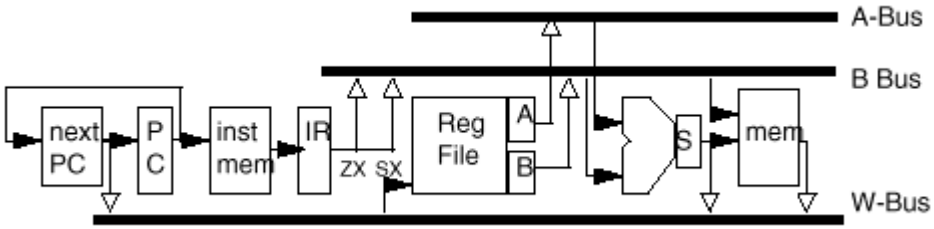
- Campuri simple “increment & branch”
- Campuri de comanda a Unitatii de Executie

- **Proiectarea Comenzii se reduce la Microprogramare**
- **Comanda este mult mai complicata in cazurile urmatoare:**
  - Set Complex de Instructiuni;
  - Unitati de executie cu restrictii (Vezi anexa).

- **Set de Instructiuni Simplu si o Unitate de Executie puternica => Comanda Simpla**
  - Se poate incerca simplificarea hardware-lui;
  - In loc de a urmari cresterea de viteza => mai multe instructiuni simultan.

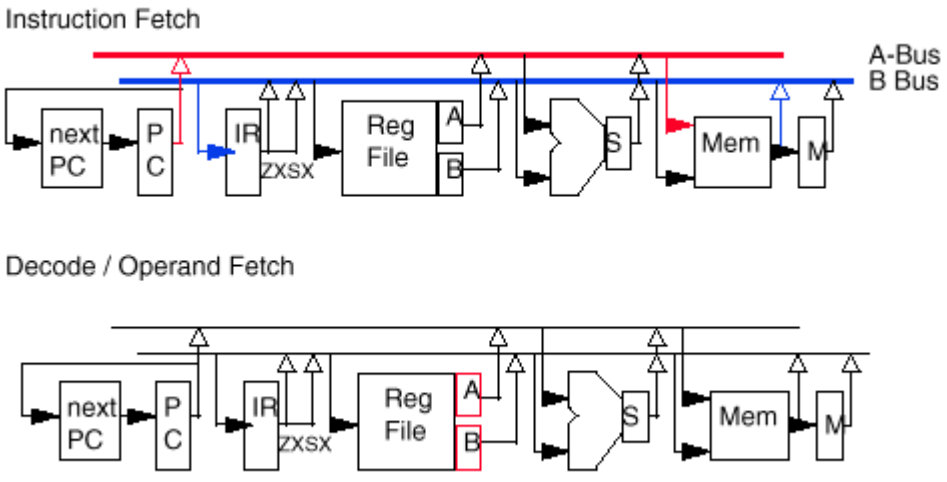
ANEXA

Unitate de Executie in Mai multe Cicluri



- In fiecare ciclu de ceas, fiecare Magistrala poate fi utilizata pentru transferul de la o singura sursa;
- μ-instructiunea poate contine campurile B-Bus si W-Bus

Microarhitectura cu doua magistrale. Comentarii



Ce se poate spune despre urmatoarele situatii:

- O singura magistrala?
- Un singur sumator?
- Registre Generale cu un singur port?