CN2_4.

Proiectarea unui procesor care opereaza intr-un singur ciclu de ceas.

Performantele unui procesor sunt sunt impuse de catre urmatorii factori:

- numarul de instructiuni din programul care se executa (n);
- perioada/ciclul ceasului (T);
- numarul mediu de cicluri de ceas pe instructiune (CPI).

Proiectarea procesorului (unitatea de executie si unitatea de comanda) va determina:

- perioada ceasului;
- numarul de cicluri de ceas pe instructiune.

Subiectul cursului curent:

Proiectarea unui procesor care executa o instructiune intr-un singur ciclu de ceas.

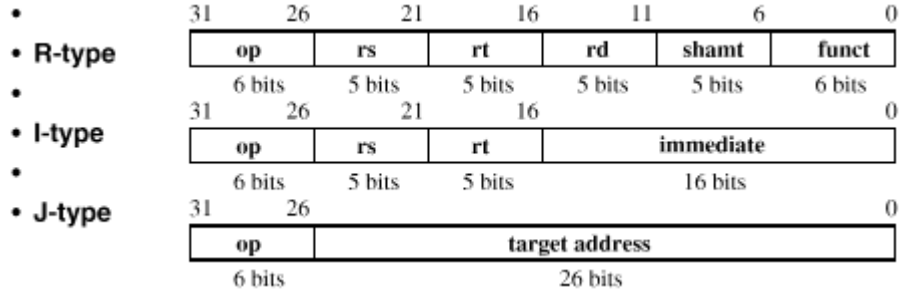
- Avantaj: Un singur ciclu de ceas pe instructiune;
- Dezavantaj: Durata mare a ciclului de ceas.

Etapele proiectarii:

1. Se examineaza setul de instructiuni din care rezulta cerintele pentru unitatea de executie:
   - semnificatia fiecarei instructiuni este data de transferurile intre registre;
   - unitatea de executie trebuie sa includa elementele de memorare corespunzatoare: registrele necesare setului de instructiuni (eventual mai multe);
   - unitatea de excutie trebuie sa asigure fiecare transfer intre registre;
2. Se selecteaza componentele unitatii de executie si se stabileste metodologia de sincronizare (aplicare a ceasului).
3. Se asambleaza unitatea de executie corespunzator specificatiilor.
4. Se analizeaza implementarea fiecarei instructiuni pentru determinarea semnalelor si a punctelor de comanda, care afecteaza implementarea fiecarui transfer intre registre.
5. Se asambleaza logica de comanda.

Formatele instructiunilor procesorului MIPS (Microprocessor without Interlocking Pipeline Stages):

Toate instructiunile MIPS au 32 de biti. Sunt prezente trei formate diferite:

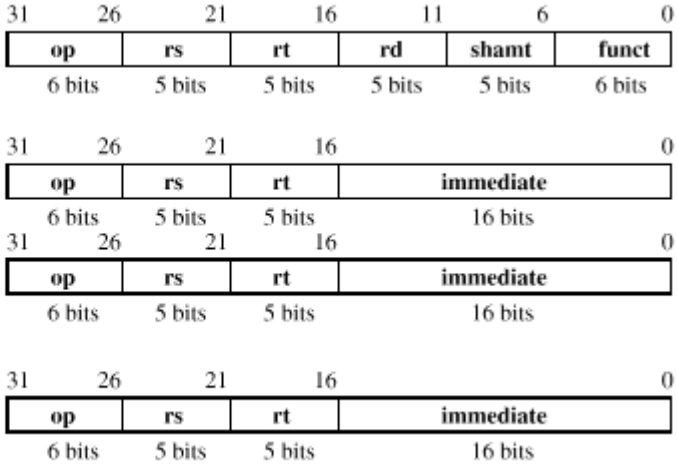


Campurile de operatie sunt urmatoarele:

- op: codul de operatie al instructiunii;
- rs, rt, rd: adresele registrelor surse si destinatie
- shamt: cantitatea/numarul de biti cu care se efectueaza deplasarea;
- funct: selecteaza varianta se operatie specificata de catre op;
- adresa/imediat: deplasarea adresei(offset)/valoare imediata;
- adresa tinta/target address: deplasarea pentru adresa tinta de salt.

Se va studia urmatorul subset de instructiuni MIPS:

- **ADD and SUB**
  - **addU rd, rs, rt**
  - **subU rd, rs, rt**
- **OR Immediate:**
  - **ori rt, rs, imm16**
- **LOAD and STORE Word**
  - **lw rt, rs, imm16**
  - **sw rt, rs, imm16**
- **BRANCH:**
  - **beq rs, rt, imm16**



Transferurile Logice intre Registre (TLR)

- TLR specifica semnificatia instructiunilor.
- Toate incep cu citirea instructiunii.

op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADDU	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUBU	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then	PC <– PC + sign_ext(Imm16)] || 00
	else	PC <– PC + 4

Pasul 1. Cerintele setului de instructiuni:

Memoria:

- instructiuni si date

Registre:

- capacitate 32 cuvinte x 32 biti
- citeste rs, rt;
- scrie rd, rt.

Contor program (PC)

Circuit de extindere a semnului (Extender)

Aduna/Scade Registru sau Imediat (offset-ul) extins

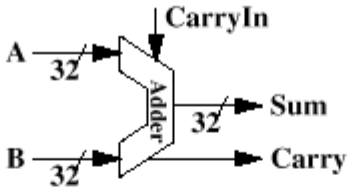
Aduna 4 sau Imediat extins la PC

Pasul 2. Componentele Unitatii de Executie:

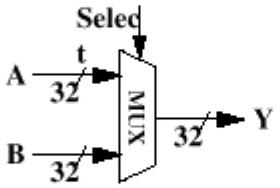
- Elemente combinationale
- Elemente de memorare:
    - Metodologia de sincronizare

Elementele combinationale: blocurile constructive de baza.

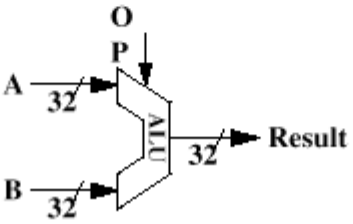
- **Adder**



- **MUX**



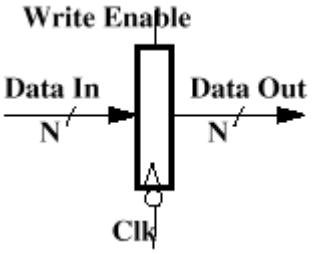
- **ALU**



Elementele de memorare: blocurile constructive de baza

Registrul

- asemanator cu bistabilului D cu exceptia ca:



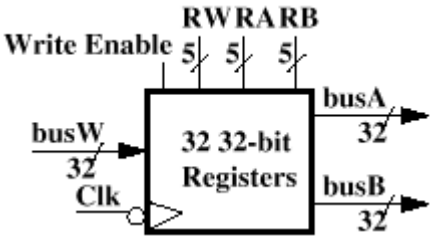
    - are intrare si iesire de cate N biti.
- intrare de activare (write enable)

- Activare scriere (write enable):
    - nivel coborat (0): Iesirea de Date (Data Out) nu se va modifica
    - nivel ridicat (1): Iesirea de Date va lua valoarea Intrarii de Date.

Registrele Generale:

Constau in 32 de registre biport (iesire):

- Doua magistrale de iesire de cate 32 de biti: busA si busB
- O magistrala de intrare de 32 de biti: busW



Registrul este slectat prin:

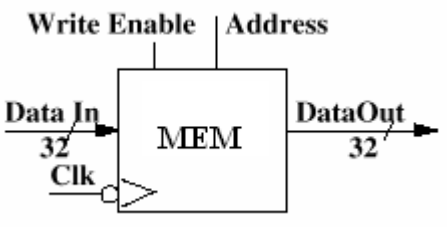
- RA (numar), care specifica registrul general al carui continut se plaseaza pe busA
- RB (numar), care specifica registrul general al carui continut se plaseaza pe busB
- RW (numar), care specifica registrul general al carui continut va fi modificat prin fortarea continutului magistralei busW, cand Write Enable este pe nivel ridicat

Intrarea de ceas CLK

- Intrarea CLK este efectiva numai in operatiile de scriere
- Pe durata operatiilor de citire se comporta ca un bloc combinational:
- RA sau/si RB valid/valide implica busA sau/si busB valid/valide dupa “timpul de acces”.

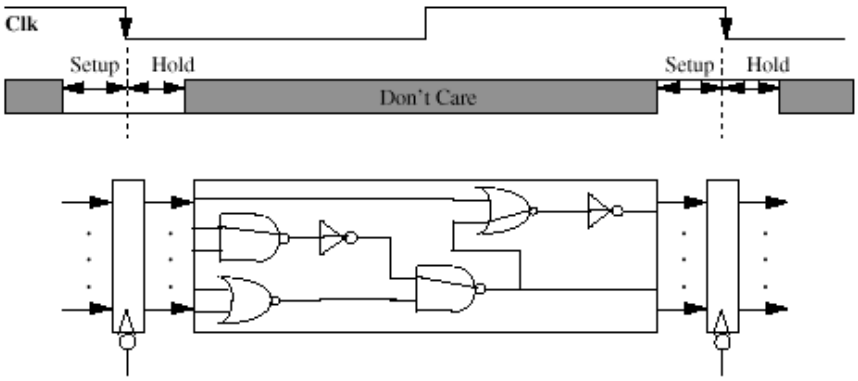
Memoria Ideala

- O magistrala de intrare: Data In
- O magistrala de Iesire: Data Out



- Cuvantul din memorie este selectat astfel:
  - Adresa (Address) selecteaza cuvantul al carui continut va fi fortat pe Data Out
  - Se forteaza Write Enable = 1: adresa va selecta cuvantul din memorie care va fi modificat de Data In

- Intrarea de Ceas (CLK)
  - Intrarea CLK este efectiva numai in operatiile de scriere
  - Pe durata operatiilor de citire se comporta ca un bloc combinational:
    - Adresa valida implica Data Out valid dupa "timpul de acces"



Metodologia de sincronizare (clocking)

- Toate elementele de memorare sunt controlate pe acelasi front al ceasului
- Durata Ciclului: = CLK → Q + Intarzierea pe Calea cea mai Lunga + Timp Stabilire (Setup Time) + Alunecarea Ceasului (Clock Skew)
- (CLK → Q + Intarzierea pe Calea cea mai Scurta – Alunecarea Ceasului) > Timpul de Retinere (Hold Time)

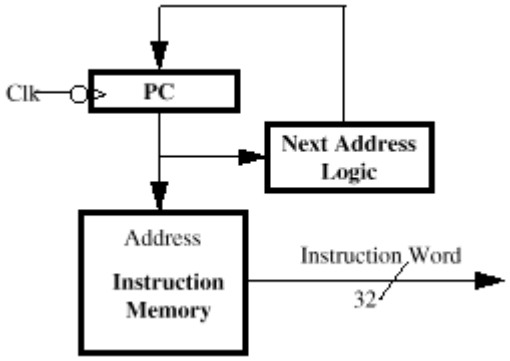
Pasul 3.

- Cerintele Transferurilor intre Registre impun alcatuirea Unitatii de Executie
- Citirea Instructiunii
- Citirea Operanzilor si Executarea Instructiunii

3.A: Unitatea de Citire a Instructiunii

Operatiile RTL:

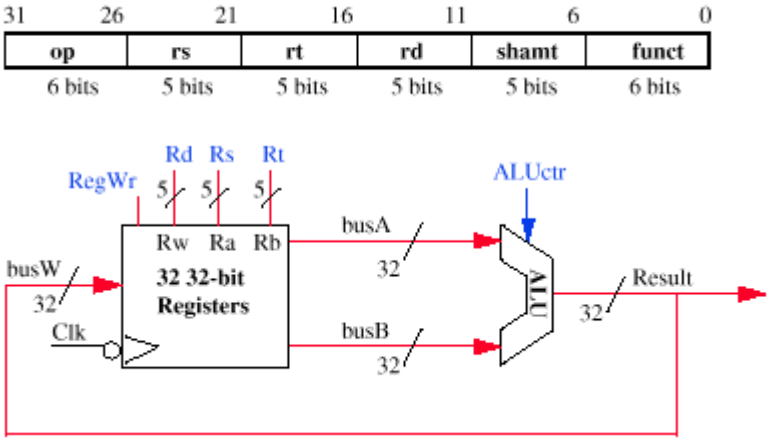
- Citeste instructiunea din IM[PC] din Memoria de Instructiuni
- Actualizeaza Contorul Programului PC:
  - Cod Secvential: PC ← PC + 4
  - Ramificare sau Salt: PC ← " adresa tinta"



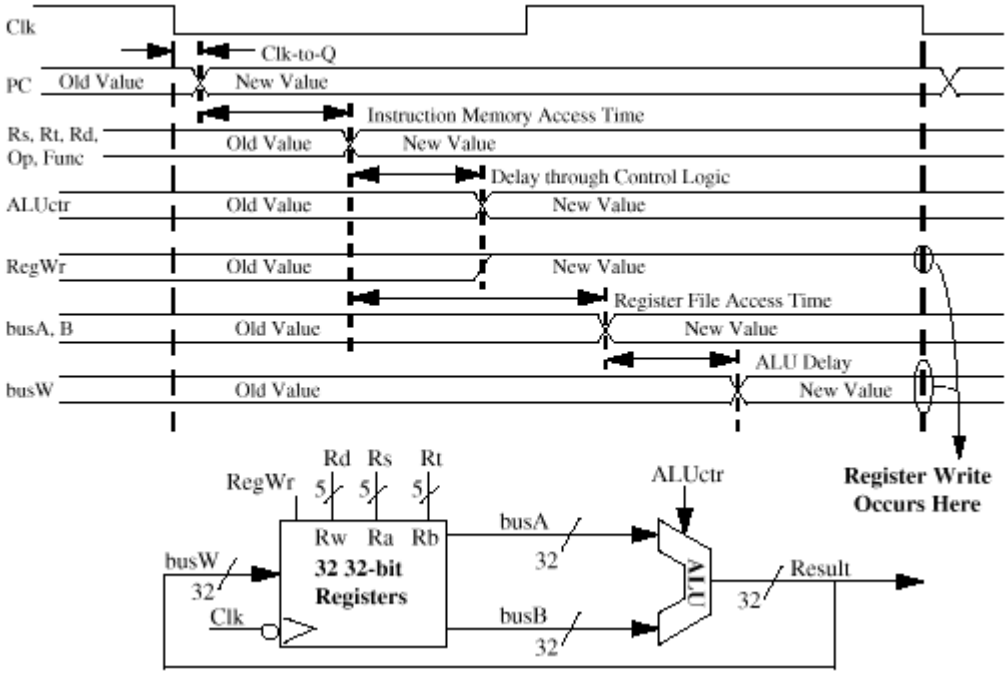
3.B: Aduna/Scade

R[rd] ← R[rs] op R[rt] Exemplu: addU rd, rs, rt

- Ra, Rb si Rw sunt furnizate de campurile rs, rt si rd ale instructiunii
- ALUctr si RegWr: logica de control dupa decodificarea instructiunii:

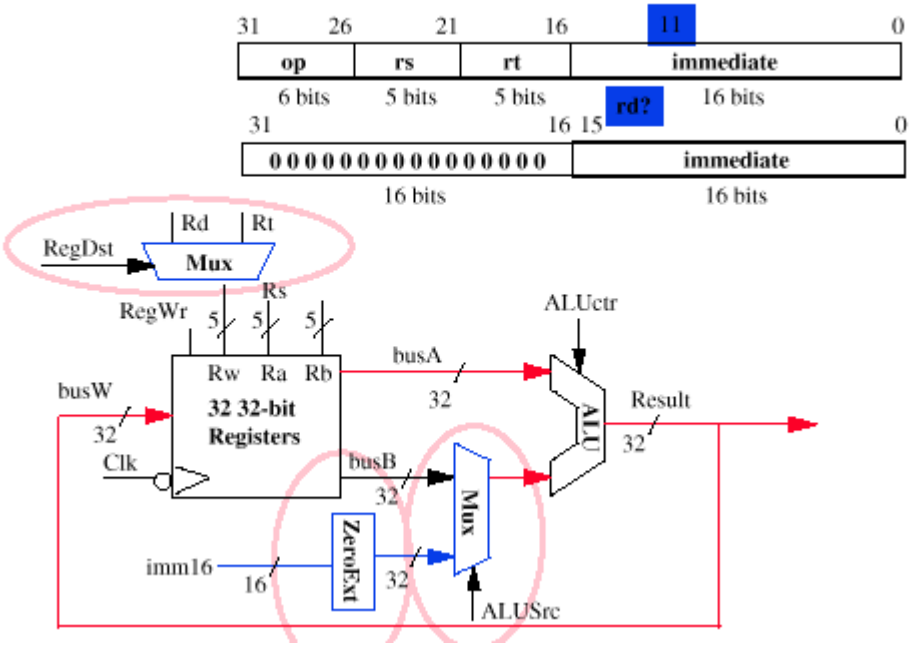


Sincronizarea Registru - Registru

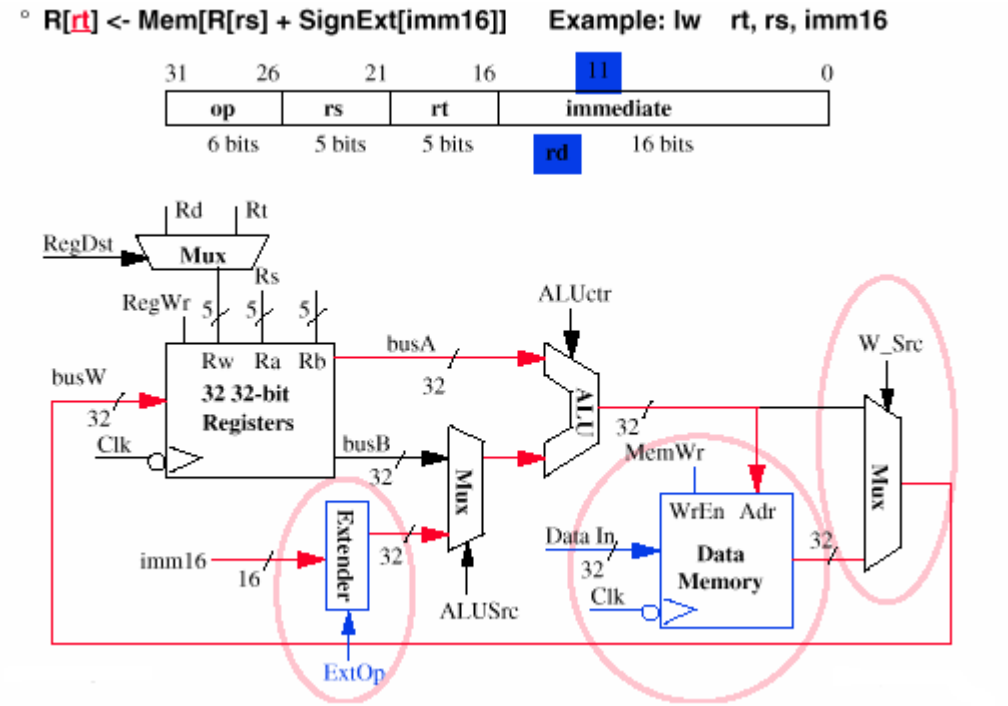


3.C: Operatii logice cu operand Imediat

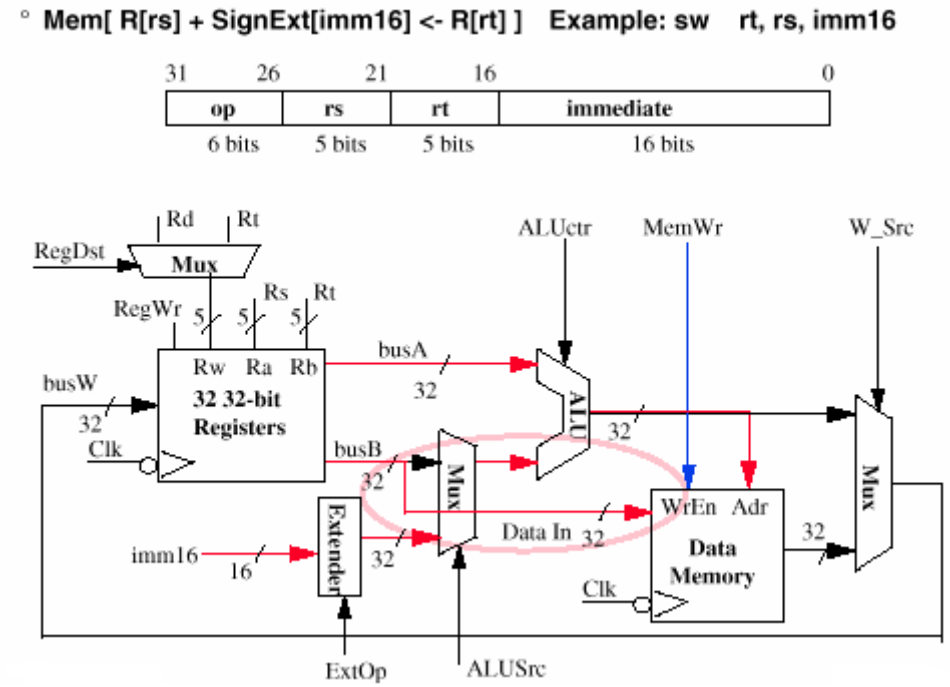
° R[rt] <- R[rs] op ZeroExt[imm16]]



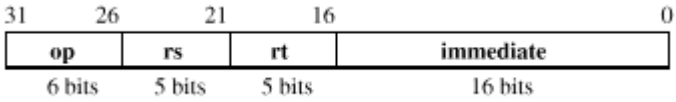
3.D: Operatii de Incarcare Load



3.E: Operatii de Stocare: Store



3.F: Instructiunea de Ramificare Branch.



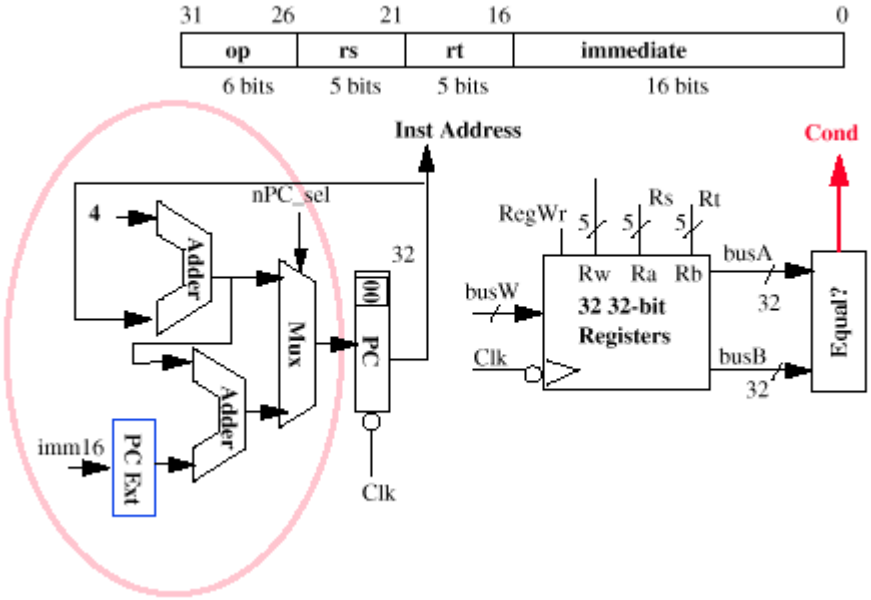
beq rs, rt, imm16

- IM[PC] Citeste Instructiunea din Memoria de Instructiuni
- Egal ← R[rs] = = R[rt] Calculeaza Conditia de Ramificare
- if(COND = 0) Calculeaza adresa instructiunii urmatoare
    - PC ← PC + 4 + (ExtSemn(imm16) x 4)
- else
    - PC ← PC + 4

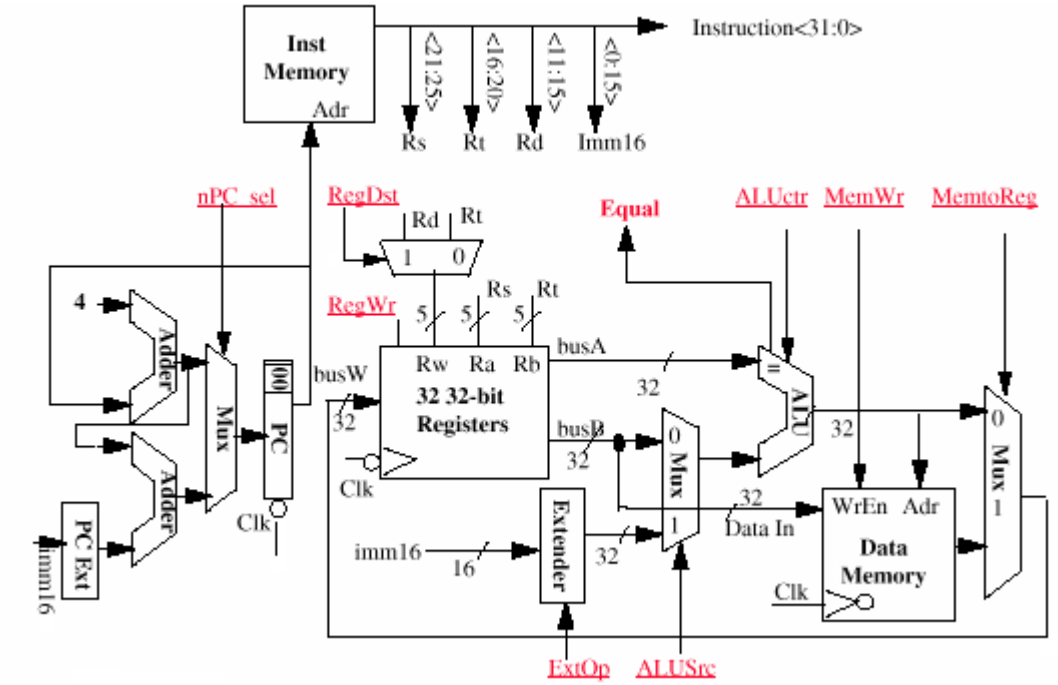
Unitatea de Executie pentru Operatiile de Ramificare:

beq rs, rt, imm16

Unitatea de Executie genereaza conditia (Egal)



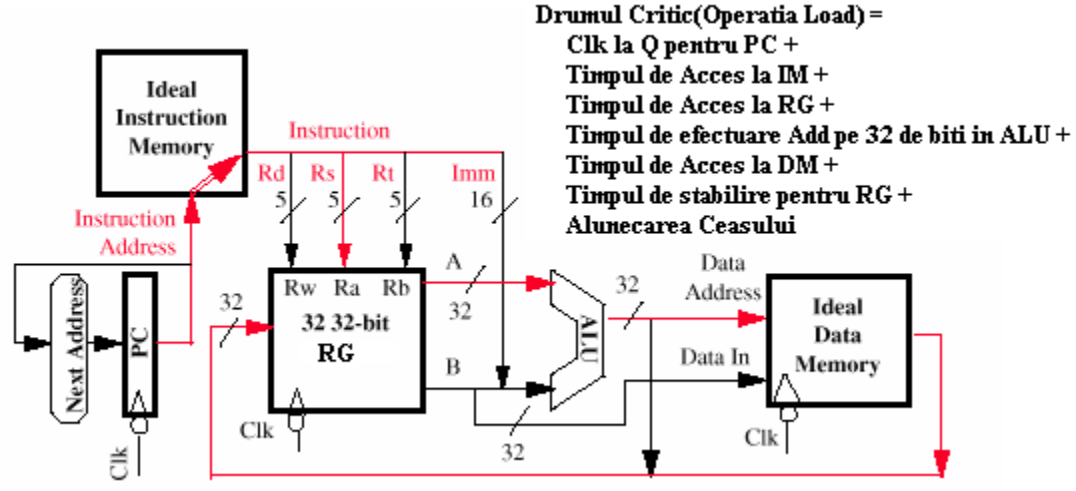
Unitatea de Executie care Opereaza intr-un Singur Ciclu de Ceas



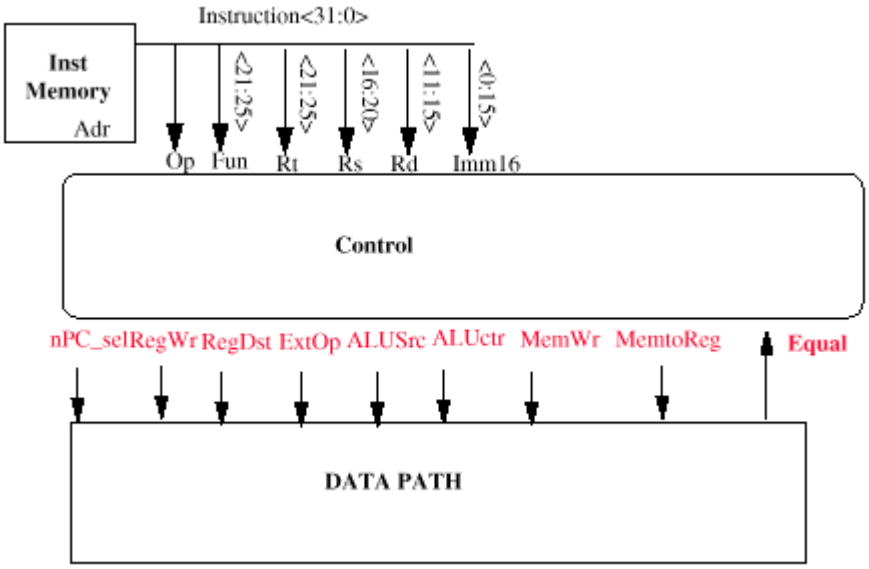
Unitatea de Executie – O abordare abstracta

Registrele generale (fisiere) si memoria idealizata:

- Intrarea CLK este activa numai pe durata operatiei de scriere
- Pe durata operatiei de citire se comporta ca logica combinationala:
    - Adresa valida conduce la Iesire valida dupa "timpul de acces"

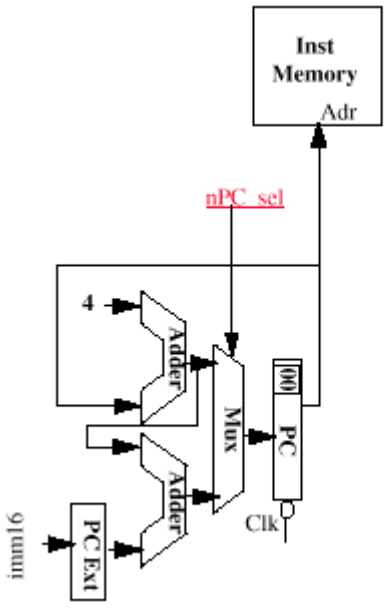


Pasul 4. Unitatea de Executie data: RTL => Comanda



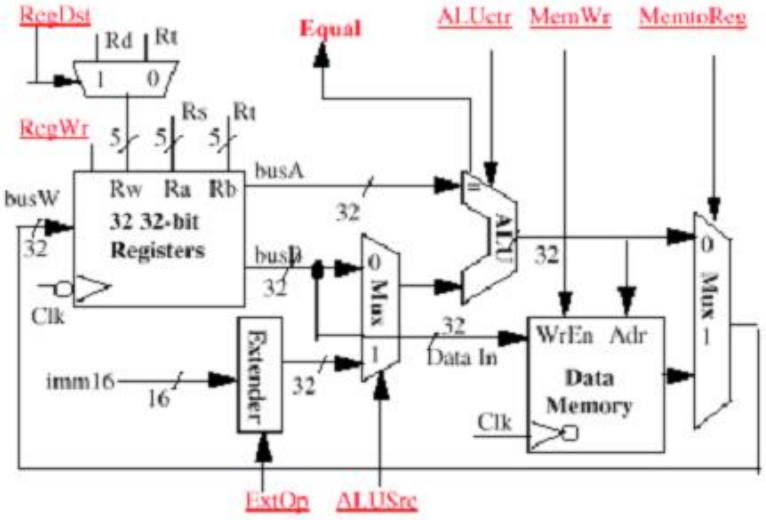
Semnificatiile Semnalelor de Comanda

- Rs, Rt si Imed16 sunt cablate in Unitatea de Executie
- nPC_sel: 0 => PC ← PC + 4; 1 => PC ← PC + 4 + ExtSemn(Imed16) || 00



Semnificatia Semnalelor de Comanda

- ExtOp: "zero", "sign"
- ALUsrc: 0 => regB; 1 => immed
- ALUctr: "add", "sub", "or"
- MemWr: write memory
- MemtoReg: 1 => Mem
- RegDst: 0 => "rt"; 1 => "rd"
- RegWr: write dest register



inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
ALUsrc = RegB, ALUctr = "add", RegDst = rd, RegWr, nPC_sel = "+4"		
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__		
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__		
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__		
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__		
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4	
ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__		

Semnalele de Comanda

Pasul 5. Logica pentru fiecare semnal de comanda.

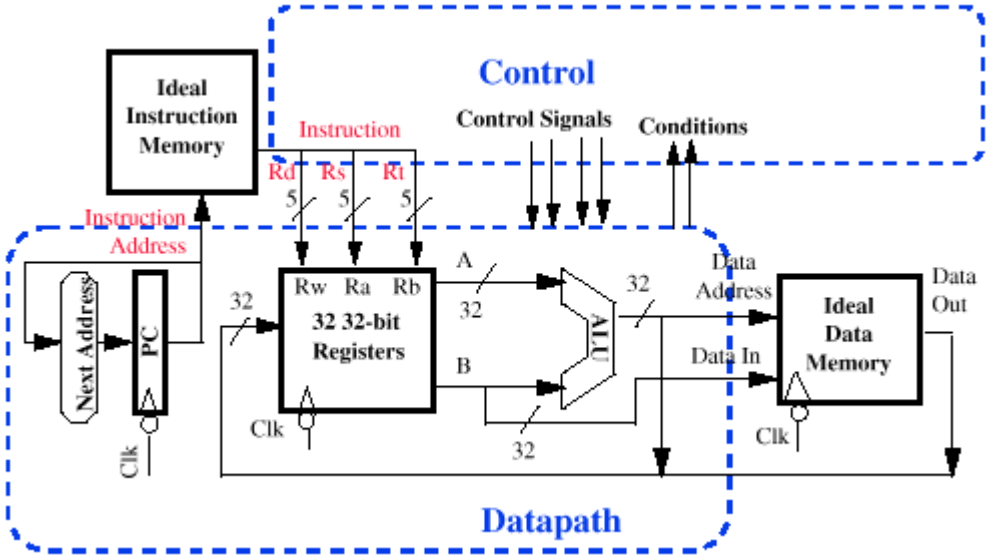
- nPC_sel <= if (OP == BEQ) then EQUAL else 0
- ALUsrc <= if (OP == "000000") then "regB" else "immed"
- ALUctr <= if (OP == "000000") then funct
  elseif (OP == ORi) then "OR"
  elseif (OP == BEQ) then "sub"
  else "add"
- ExtOp <= _____
- MemWr <= _____
- MemtoReg <= _____
- RegWr: <=_____
- RegDst: <= _____

Logica pentru fiecare semnal de comanda

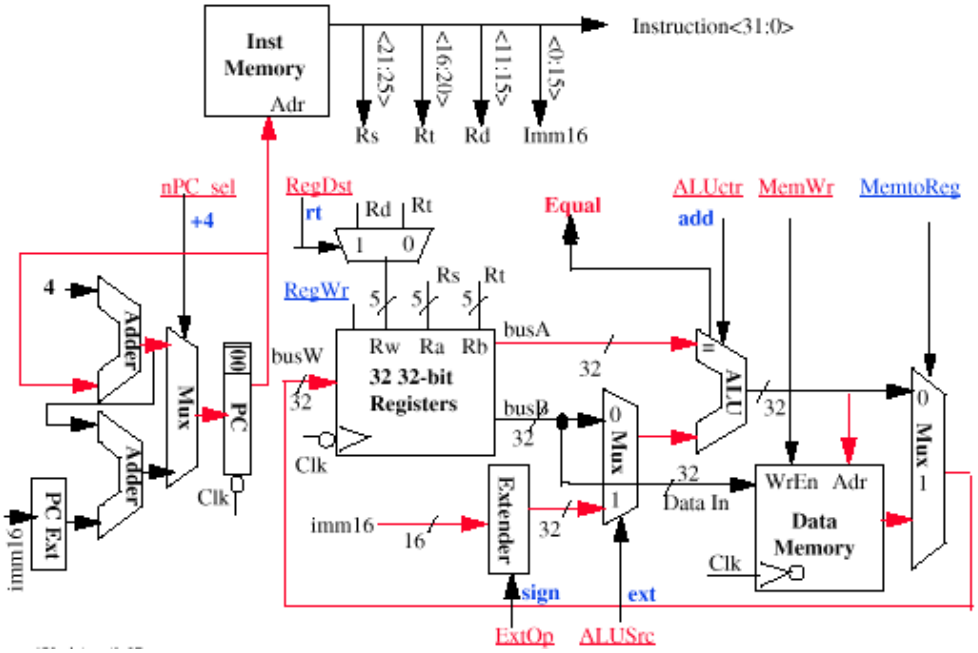
- nPC_sel <= if (OP == BEQ) then EQUAL else 0
- ALUsrc <= if (OP == "000000") then "regB" else "immed"
- ALUctr <= if (OP == "000000") then funct
  elseif (OP == ORi) then "OR"
  elseif (OP == BEQ) then "sub"
  else "add"
- ExtOp <= if (OP == ORi) then "zero" else "sign"
- MemWr <= (OP == Store)
- MemtoReg <= (OP == Load)
- RegWr: <= if ((OP == Store) || (OP == BEQ)) then 0 else 1
- RegDst: <= if ((OP == Load) || (OP == ORi)) then 0 else 1

Exemplu: Instructiunea Incarca/Load

Reprezentarea Abstracta a Implementarii



Structura logica in raport cu structura fizica



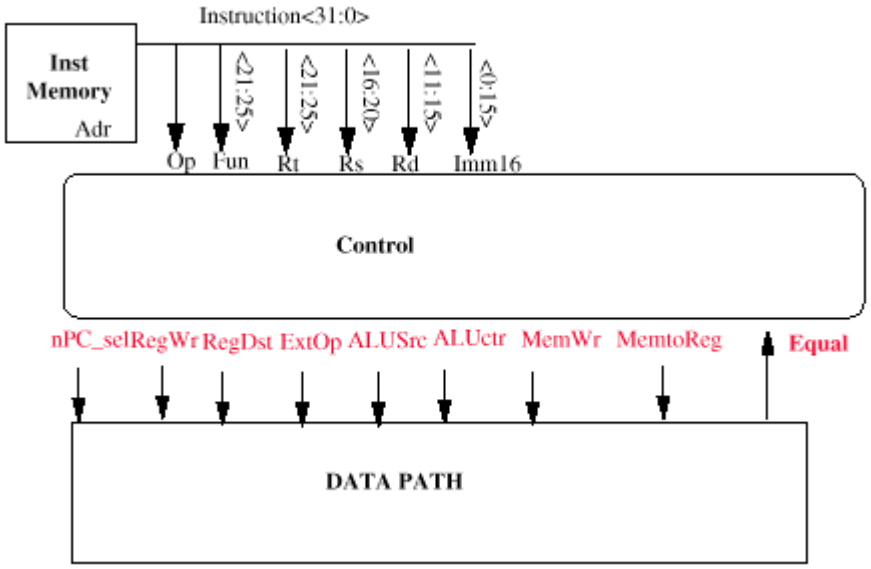
Structura MIPS se caracterizeaza prin urmatoarele elemente:

- Instructiunile au lungimea fixa
- Registrele surse sunt pozitionate in aceleasi campuri
- “Imediat” are aceeasi dimensiune si aceeasi pozitionare
- Operatiile vizeaza operanzi din registre sau campul imediat

Unitatea de Executie intr-un Singur Ciclu => CPI = 1, Durata Ciclului => mare

Proiectarea Unitatii de Comanda:

Pasul 5. Unitatea de Executie Data => Unitatea de Comanda

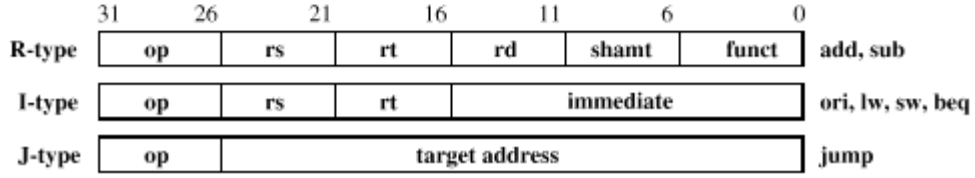


inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = "add", RegDst = rd, RegWr, nPC_sel = "+4"	
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = "sub", RegDst = rd, RegWr, nPC_sel = "+4"	
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
	ALUsrc = Im, Extop = "Z", ALUctr = "or", RegDst = rt, RegWr, nPC_sel = "+4"	
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
	ALUsrc = Im, Extop = "Sn", ALUctr = "add", MemtoReg, RegDst = rt, RegWr,	nPC_sel = "+4"
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
	ALUsrc = Im, Extop = "Sn", ALUctr = "add", MemWr, nPC_sel = "+4"	
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4	
	nPC_sel = "Br", ALUctr = "sub"	

Sumarul Semnalelor de Comanda

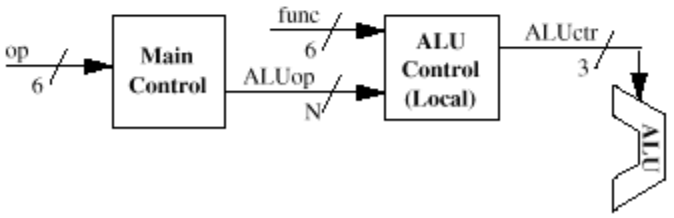
Tabela Semnalelor de Comanda (Sumar)

See Appendix A	func	10 0000	10 0010	We Don't Care :-)				
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
		add	sub	ori	lw	sw	beq	jump
RegDst		1	1	0	0	x	x	x
ALUSrc		0	0	1	1	1	0	x
MemtoReg		0	0	0	1	x	x	x
RegWrite		1	1	1	1	0	0	0
MemWrite		0	0	0	0	1	0	0
nPCsel		0	0	0	0	0	1	0
Jump		0	0	0	0	0	0	1
ExtOp		x	x	0	1	1	x	x
ALUctr<2:0>		Add	Subtract	Or	Add	Add	Subtract	xxx



Conceptul Decodificarii Locale

op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	"R-type"	Or	Add	Add	Subtract	xxx



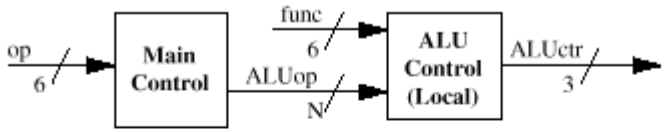
Codificarea Operatiilor UALop - ALUop

Implementarea Setului complet de Instructiuni MIPS, necesita ca UALop sa aibe 3 biti pentru a reprezenta:

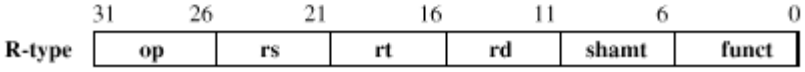
- (1) Instructiunile de "tip R"
- Instructiunile de "tip I", care asigura ca UAL sa execute:
  - (2) SAU - Or,
  - (3) Add,
  - (4) Sub si
  - (5) And (Exemplul andi)

	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
ALUop (Symbolic)	"R-type"	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx

Decodificarea Campului “func”



	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx



funct<5:0>	Instruction Operation
10 0000	add
10 0010	subtract
10 0100	and
10 0101	or
10 1010	set-on-less-than

ALUctr<2:0>	ALU Operation
000	Add
001	Subtract
010	And
110	Or
111	Set-on-less-than

Tabela de adevar pentru ALUctr

ALUop (Symbolic)	R-type	ori	lw	sw	beq
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01

funct<3:0>	Instruction Op.
0000	add
0010	subtract
0100	and
0101	or
1010	set-on-less-than

ALUop bit<2>	ALUop bit<1>	ALUop bit<0>	func bit<3>	func bit<2>	func bit<1>	func bit<0>	ALU Operation	ALUctr bit<2>	ALUctr bit<1>	ALUctr bit<0>
0	0	0	x	x	x	x	Add	0	1	0
0	x	1	x	x	x	x	Subtract	1	1	0
0	1	x	x	x	x	x	Or	0	0	1
1	x	x	0	0	0	0	Add	0	1	0
1	x	x	0	0	1	0	Subtract	1	1	0
1	x	x	0	1	0	0	And	0	0	0
1	x	x	0	1	0	1	Or	0	0	1
1	x	x	1	0	1	0	Set on <	1	1	1

Ecuatiile Logice pentru ALUctr<2>

ALUop			func				
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	ALUctr<2>
0	x	1	x	x	x	x	1
1	x	x	0	0	1	0	1
1	x	x	1	0	1	0	1

This makes func<3> a don't care

° ALUctr<2> = !ALUop<2> & ALUop<0> +
ALUop<2> & !func<2> & func<1> & !func<0>

Ecuatiile Logice pentru ALUctr<1>

ALUop			func				
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	ALUctr<1>
0	0	0	x	x	x	x	1
0	x	1	x	x	x	x	1
1	x	x	0	0	0	0	1
1	x	x	0	0	1	0	1
1	x	x	1	0	1	0	1

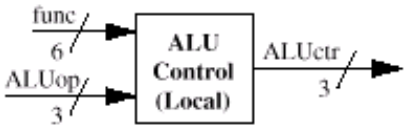
° ALUctr<1> = !ALUop<2> & !ALUop<1> +
ALUop<2> & !func<2> & !func<0>

Ecuatiile Logice pentru ALUctr<0>

ALUop			func				
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	ALUctr<0>
0	1	x	x	x	x	x	1
1	x	x	0	1	0	1	1
1	x	x	1	0	1	0	1

° ALUctr<0> = !ALUop<2> & ALUop<1>
+ ALUop<2> & !func<3> & func<2> & !func<1> & func<0>
+ ALUop<2> & func<3> & !func<2> & func<1> & !func<0>

Blocul de comanda pentru UAL



- ALUctr<2> = !ALUop<2> & ALUop<0> +
  ALUop<2> & !func<2> & func<1> & !func<0>
- ALUctr<1> = !ALUop<2> & !ALUop:<1> +
  ALUop<2> & !func<2> & !func<0>
- ALUctr<0> = !ALUop<2> & ALUop:<1>
  + ALUop<2> & !func<3> & func<2> & !func<1> & func<0>
  + ALUop<2> & func<3> & !func<2> & func<1> & !func<0>

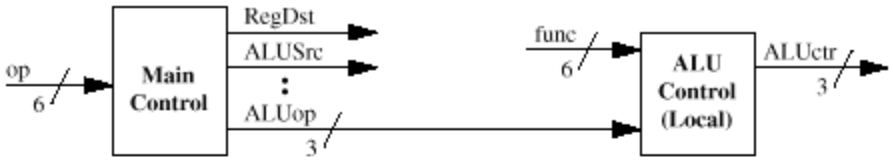
Pasul 5 : Logica pentru fiecare semnal de comanda

- nPC_sel <= if (OP == BEQ) then EQUAL else 0
- ALUsrc <= if (OP == "Rtype") then "regB" else "immed"
- ALUctr <= if (OP == "Rtype") then funct
  elseif (OP == ORi) then "OR"
  elseif (OP == BEQ) then "sub"
  else "add"
- ExtOp <= _____
- MemWr <= _____
- MemtoReg <= _____
- RegWr: <=_____
- RegDst: <= _____

Pasul 5. Logica pentru fiecare semnal de comanda

- **nPC_sel** <= if (OP == BEQ) then EQUAL else 0
- **ALUsrc** <= if (OP == "Rtype") then "regB" else "immed"
- **ALUctr** <= if (OP == "Rtype") then funct
  elseif (OP == ORi) then "OR"
  elseif (OP == BEQ) then "sub"
  else "add"
- **ExtOp** <= if (OP == ORi) then "zero" else "sign"
- **MemWr** <= (OP == Store)
- **MemtoReg** <= (OP == Load)
- **RegWr:** <= if ((OP == Store) || (OP == BEQ)) then 0 else 1
- **RegDst:** <= if ((OP == Load) || (OP == ORi)) then 0 else 1

Logica pentru Comanda Principala



op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop (Symbolic)	"R-type"	Or	Add	Add	Subtract	xxx
ALUop <2>	1	0	0	0	0	x
ALUop <1>	0	1	0	0	0	x
ALUop <0>	0	0	0	0	1	x

Tabela de Adevar pentru RegWrite

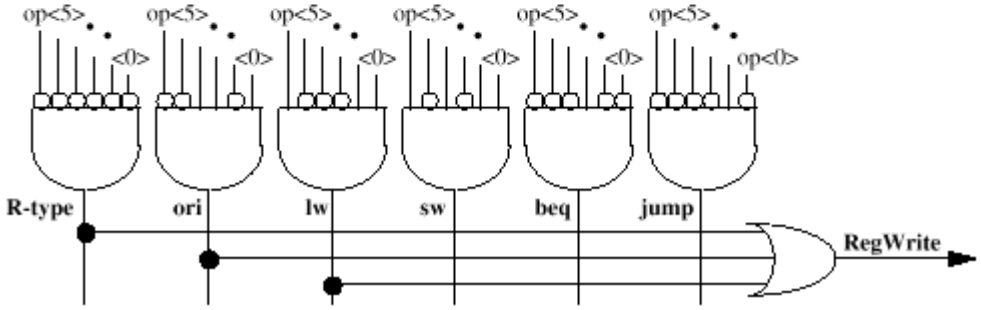
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegWrite	1	1	1	0	0	0

° **RegWrite = R-type + ori + lw**

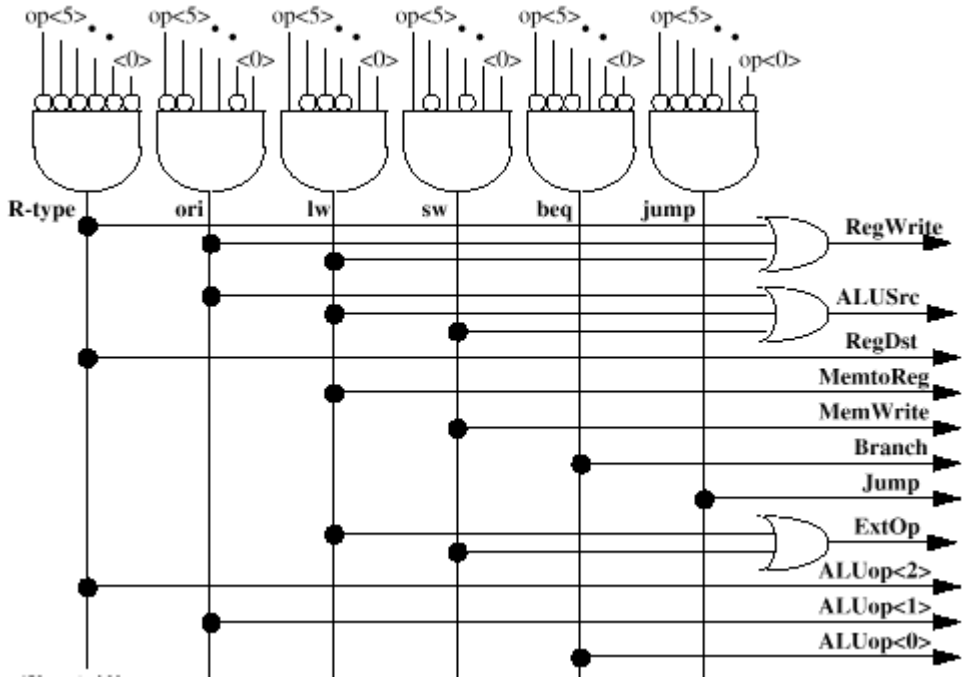
= !op<5> & !op<4> & !op<3> & !op<2> & !op<1> & !op<0> **(R-type)**

+ !op<5> & !op<4> & op<3> & op<2> & !op<1> & op<0> **(ori)**

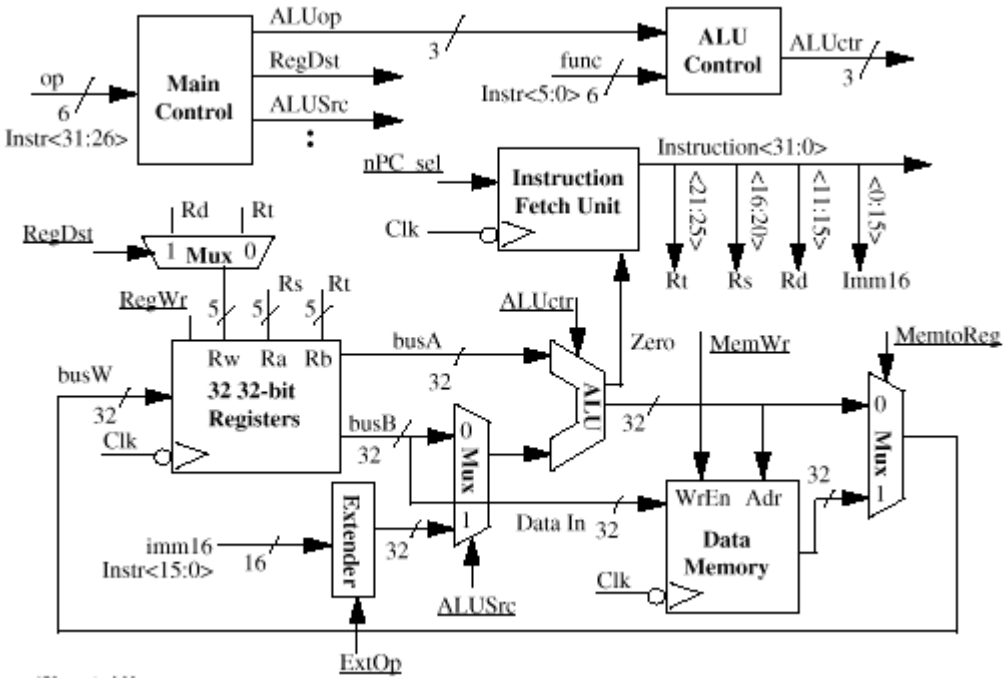
+ op<5> & !op<4> & !op<3> & !op<2> & op<1> & op<0> **(lw)**



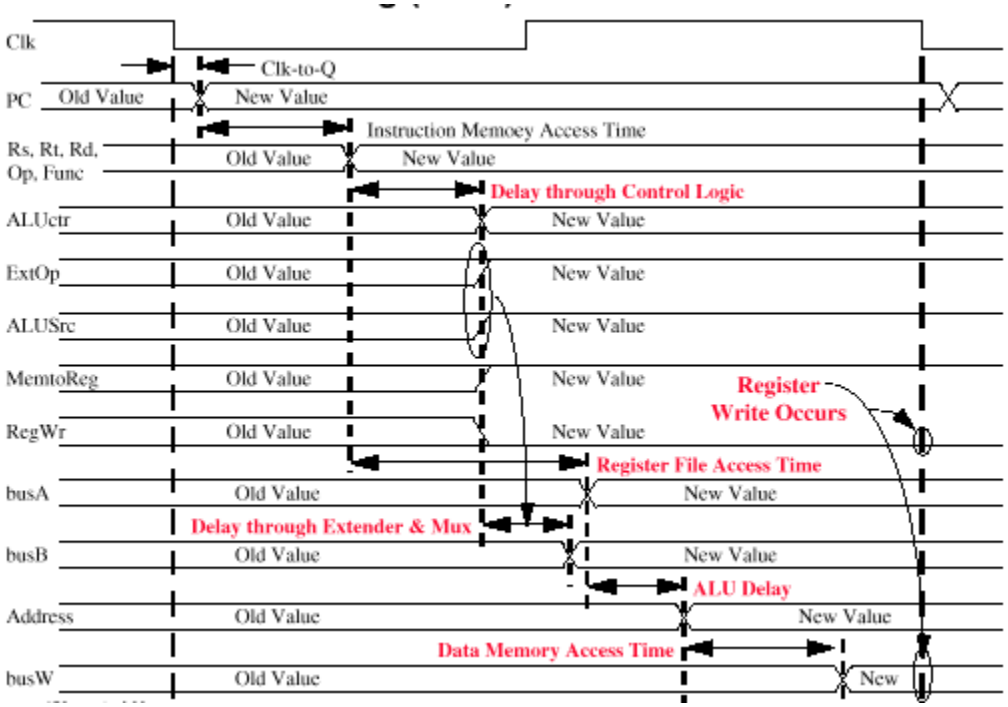
Implementarea PLA a Comenzii principale



Procesorul care opereaza intr-un singur ciclu de ceas - Ansamblu



Cazul cel mai defavorabil pentru sincronizare Incarca - Load



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

Ciclul mare de lucru:

- Ciclul de lucru trebuie sa fie suficient de mare pentru instructiunea:

  Incarca – Load:

  CPceas → Q +

  Timpul de Acces la Memoria de Instructiuni +

  Timpul de Acces la Registrele Generale (Fisierul de Registre R) +

  Intarzierea in UAL (Calculul Adresei) +

  Timpul de Acces la Memoria de date +

  Timpul de Stabilire pentru Registrele Generale +

  Alunecarea Ceasului.

- Durata ciclului pentru instructiunea Incarca – Load este mult mai mare decat cea necesara pentru alte instructiuni