```
0  B"001_111_000_0000010",   -- addi $7,$0,2
1  B"001_000_001_0000111",   -- addi $0,$1,7
2  B"001_010_001_0000000",   --addi $2,$1,0
3  B"001_011_001_0000001",   --addi $3,$1,1
4  B"001_100_001_0000000",   --addi $4,$1,0
5  B"001_101_001_0000001",   --addi $5,$1,1
6  B"011_100_010_0000000",   --sw $2,0($4)
7  B"011_101_011_0000000",   --sw $3,0($5)
8  B"010_100_010_0000000",   --lw $2,0($4)
9  B"010_101_011_0000000",   --lw $3,0($5)
10 B"000_010_011_110_0_000", --add $6,$2,$3
11 B"001_111_001_0000010",   -- addi $7,$0,1
12 B"100_111_000_0010000",   -- beq $7,$0,16
13 B"000_001_011_010_0_000", --add $2,$1,$3
14 B"000_001_110_011_0_000", --add $3,$1,$6
15 B"111_0000000001010",     --j 10
16 B"011_000_110_0000000",   -- sw $6,0($0)
```

0 RD1 = 0, Ext_Imm = 2, ALURes = 2, în $7 se încarcă val 2
1 RD1 = 0, Ext_Imm = 7, ALURes = 7, $0 va avea val 7
2 RD1 = 0, Ext_Imm = 0, ALURes = 0, $2 va avea val 0
3 RD1 = 0, Ext_Imm = 1, ALURes = 1, $3 va avea val 1
4 RD1 = 0, Ext_Imm = 0, ALURes = 0, $4 va avea val 0
5 RD1 = 0, Ext_Imm = 1, ALURes = 1, $5 va avea val 1
6 Ext_Imm = 0, MEM[$4] = $2, MemData = 0
7 Ext_Imm = 0, MEM[$5] = $3, MemData = 1
8 Ext_Imm = 0, $2 = MEM[$4], MemData = 0
9 Ext_Imm = 0, $3 = MEM[$5], MemData = 1
10 RD1 = 0, RD2 = 1, ALURes = 1, $6 va avea val 1 (0+1)
11 RD1 = 1, Ext_Imm = 1, $7 va avea valoarea 3, ALURes = 3
12 Ext_Imm = 16
13 RD1 = 0, RD2 = 1, ALURes = 1, $2 va avea valoarea 1 (0+1)
14 RD1 = 0, RD2 = 1, ALURes = 1, $3 va avea valoarea 1 (0+1)
15
16 Ext_Imm = 0, ALURes = 7, $0 va avea val 8

Instrucțiunile alese le-am încadrat în chenar roșu pe următoarele pagini, totodată le-am evidențiat în tabelul „Semnale control MIPS16".

* La fiecare iterație a buclei, la linia 10 RD1 și RD2 se va modifica datorită liniei 13, respectiv 14, ca ALURes să rețină mereu suma dintre cei doi regiștri $2 și $3.

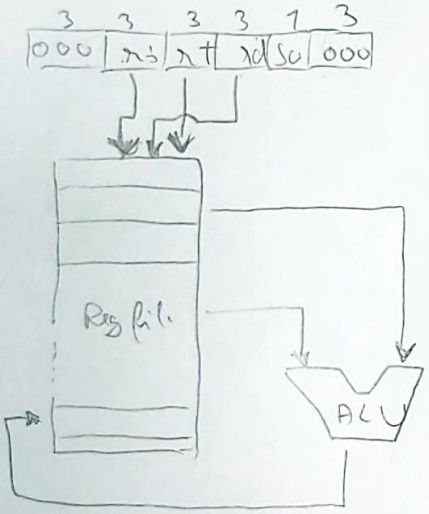
- Logical OR unsigned constant: RF[rt] ← RF[rs] | Zext(imm)

ori $1, $2, 28

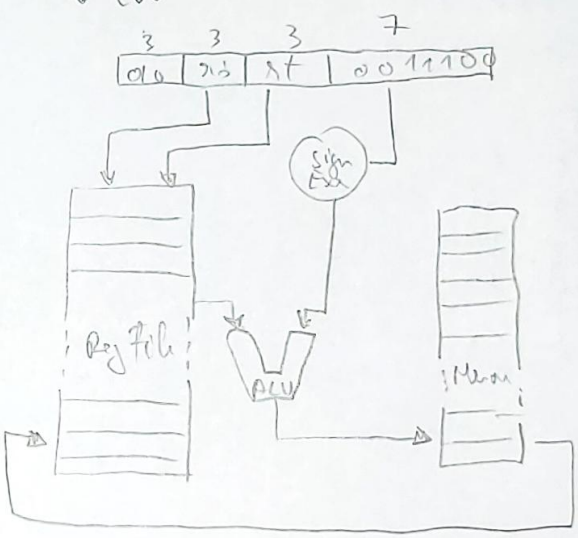
B"110_010_001_0011100"

## Diagrame de procesare
(pe exemplul celor de mine)

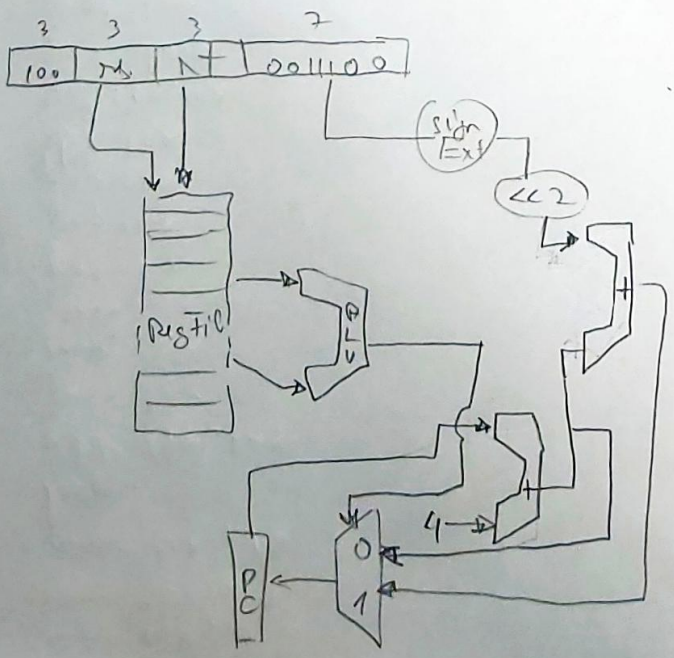
- add



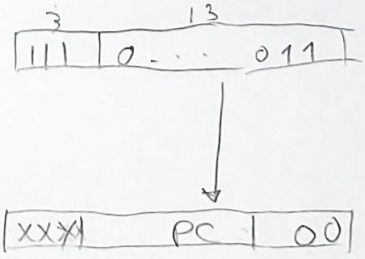

- lw




- beq




- j



- Logical XOR : RF[rd] ← RF[rs] ^ RF[rt]

xor $2, $4, $7

B"000_100_111_010_0_111"

- Set on less than : if (RF[rs] < RF[rt]) then RF[rd] ← 1
  else RF[rd] ← 0

slt $4, $2, $3

B"000_010_011_100_0_000"

## Instructiuni de tip I

- Add Immediate : RF[rt] ← RF[rs] + S_Ext(imm)

addi $3, $4, 28

B"001_100_011_0011100"

- Load Word : RF[rt] ← M[RF[rs] + S_Ext(imm)]

lw $3, 28($5)

B"010_101_011_0011100"

- Store Word : M[RF[rt] + S_ext(imm)] ← RF[rs]

sw $3, 28($5)

B"011_101_011_0011100"

- Branch on Equal : if (RF[rs] = RF[rt]) then PC ← PC + 4 + S_Ext(imm) << 2

beq $3, $4, 28

B"100_100_011_0011100"

- Logical AND unsigned constant : RF[rt] ← RF[rs] & Z_ext(imm)

andi $3, $4, 28

B"101_100_011_0011100"

## Semnale control MIPS16 pentru Anexa 5

| Instrucțiune | Opcode *Instr(15-13)* | RegDst | ExtOp | ALUSrc | Branch | <Br?> (optional) | Jump | JmpR (optional) | Mem Write | Memto Reg | Reg Write | ALUOp (2:0) | func *Instr(2-0)* | ALUCtrl (2:0) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| add | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 000 | 000 |
| sub | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 001 | 001 |
| sll | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 010 | 010 |
| srl | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 011 | 011 |
| and | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 100 | 100 |
| or | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 101 | 101 |
| xor | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 110 | 110 |
| slt | 000 | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 000 | 111 | 111 |
| | | | | | | | | | | | | | | |
| addi | 001 | 0 | 1 | 1 | 0 | | 0 | | 0 | 0 | 1 | 000 | 001 | 000 |
| lw | 010 | 0 | 1 | 1 | 0 | | 0 | | 0 | 1 | 1 | 000 | 001 | 000 |
| sw | 011 | 0 | 1 | 1 | 0 | | 0 | | 1 | 0 | 0 | 000 | 001 | 000 |
| beq | 100 | 0 | 1 | 0 | 1 | | 0 | | 0 | 0 | 0 | 010 | 110 | 001 |
| andi | 101 | 0 | 1 | 1 | 0 | | 0 | | 0 | 0 | 1 | 101 | 101 | 100 |
| ori | 110 | 0 | 1 | 1 | 0 | | 0 | | 0 | 0 | 1 | 110 | 110 | 101 |
| j | 111 | 0 | 0 | 0 | 0 | | 1 | | 0 | 0 | 0 | 111 | 111 | 111 |

*Acele care nu conta valoarea pe un bit am pus-o ca 0, desi o puteam pune ca X