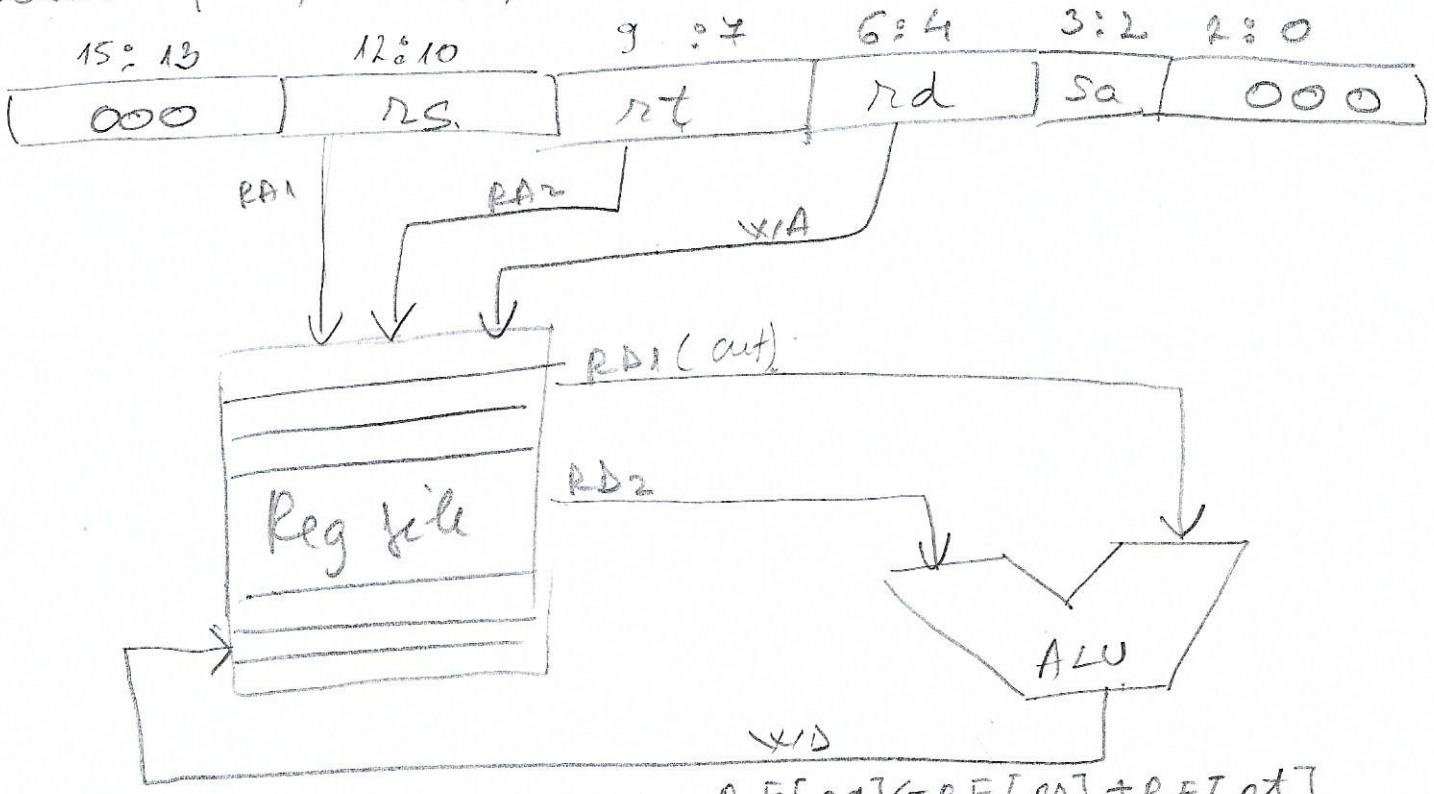
R

→add $rd, $rs, $rt RTL – RF[rd] ← RF[rs] + RF[rt]



→addu 000_rs_rt_rd_111 RF[rd] ← RF[rs] + RF[rt]

000_rs_rt_rd_001

→sub $rd, $rs, $rt RF[rd] ← RF[rs] - RF[rt]

→and 000_rs_rt_rd_100 RF[rd] ← RF[rs] & RF[rt]

→or 000_rs_rt_rd_101 RF[rd] ← RF[rs] | RF[rt]

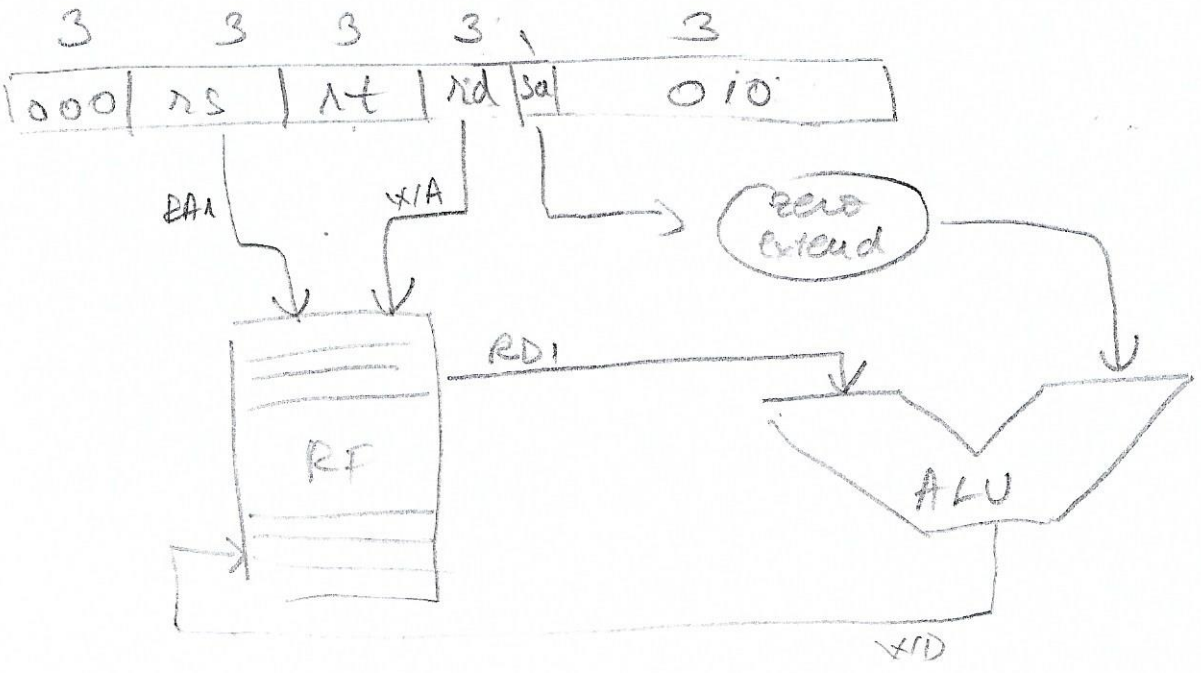
→xor 000_rs_rt_rd_110 RF[rd] ← RF[rs] ^ RF[rt]

PC ← PC + 1

shift left logical cantitatea de deplasat

→sll $rd, $rs, sa RF[rd] ← RF[rs] << sa

000_rs_rt_rd_sa_010 PC ← PC + 1



→ Slr - shift right logical

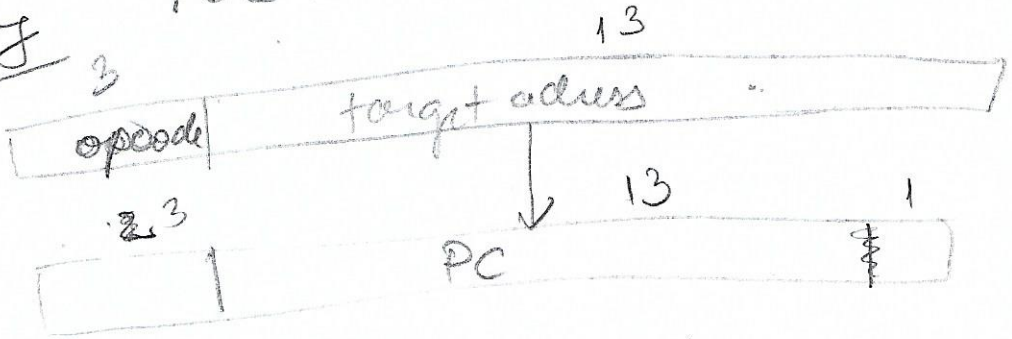
$rd, $rs, sa RI[rd] ← RF[rs] >> sa

| 011 | rs | rt | rd | sa | function |

salvarea ca la sll

PC ← PC[31:28] & target_address && 00

J



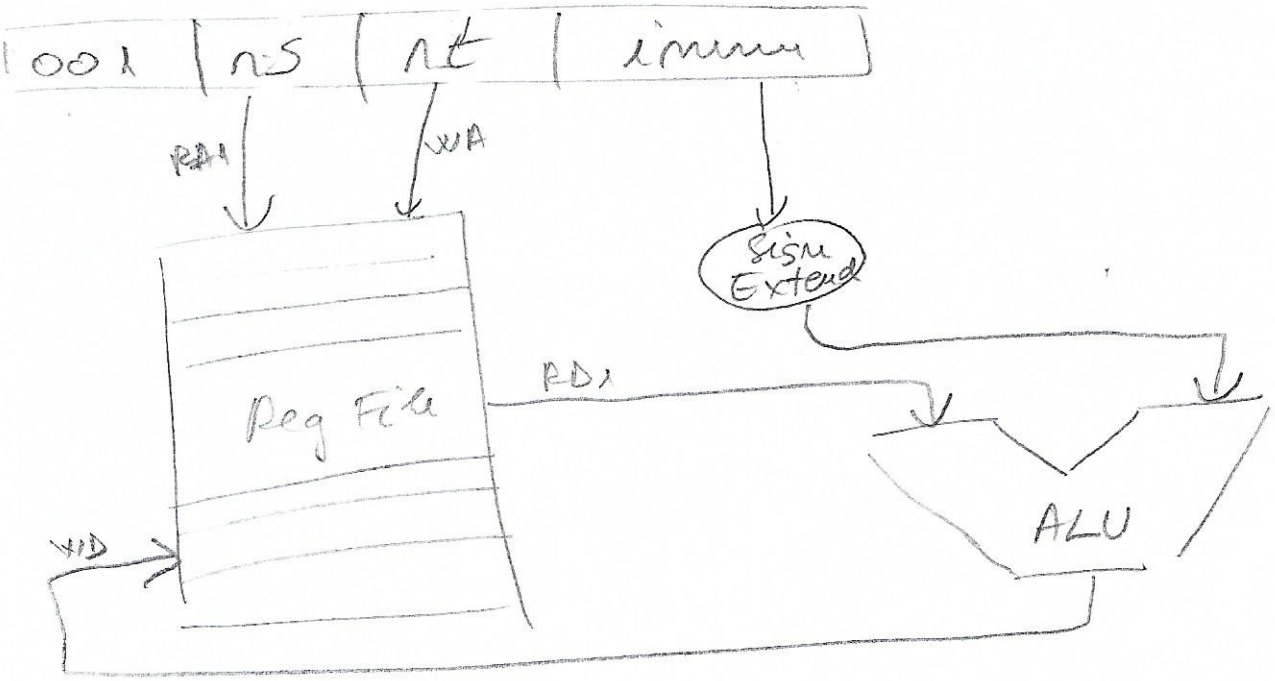
(PC ← (PC+1) & 0xf0000000) | (target << 2)

→ addi $rt, $rs, imm

3	3	3	7
001	rs	rt	immediate

RF[rt] ← RF[rs] + S_Ext(imm)

PC ← PC + 1



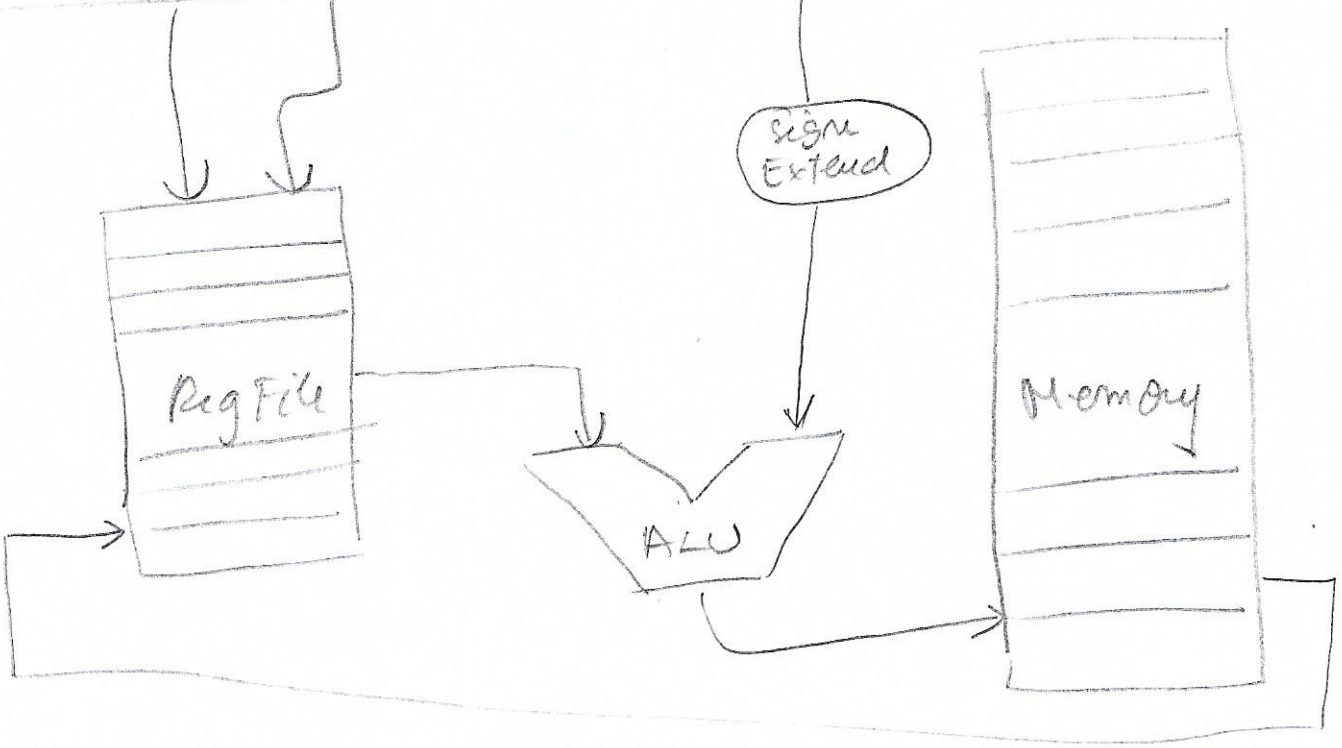
Load word ⊕ ALU

→ lw . $rt, imm($rs)

RF[rt] ← M[RF[rs] + S_Ext(imm)]

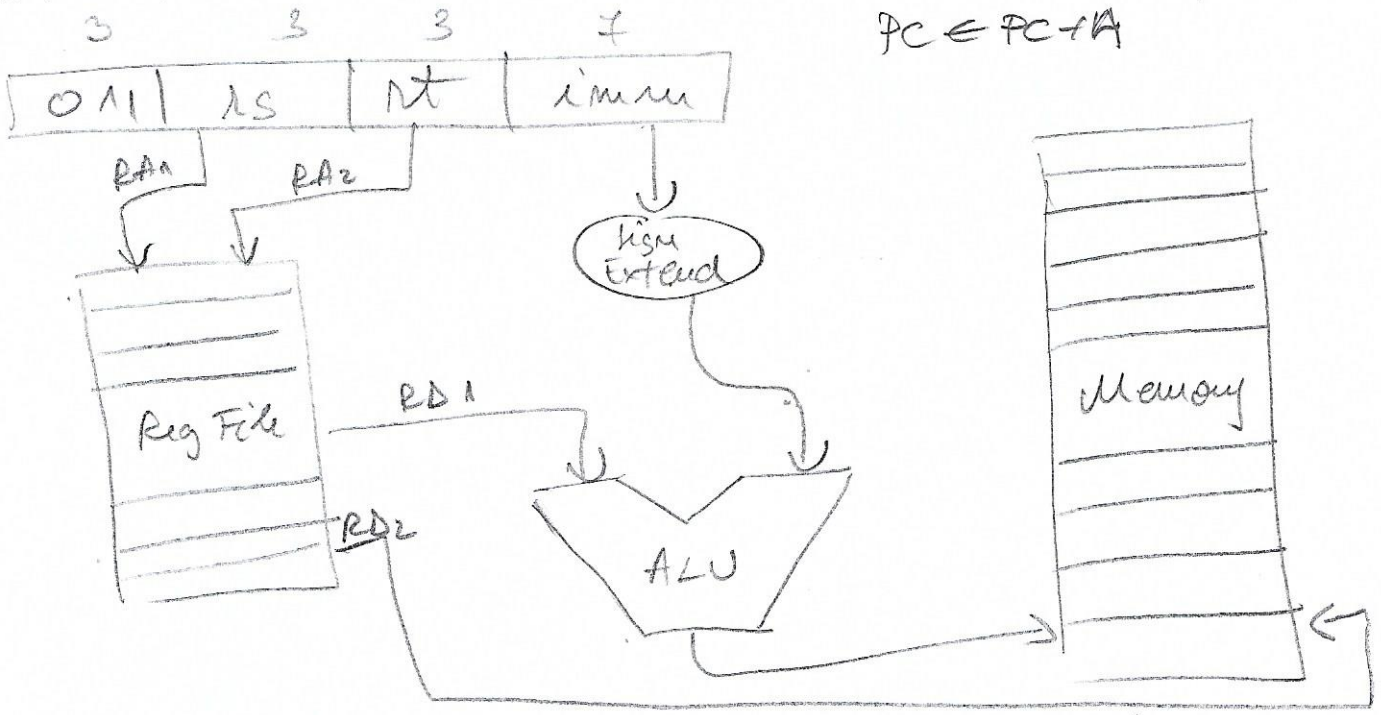
PC ← PC + 1

010	rs	rt	immediate



SW $rt, imm($rs) M[RF[rs]+S_Ext(imm)] ← RF[rt]

PC ← PC+4



(+) ALU

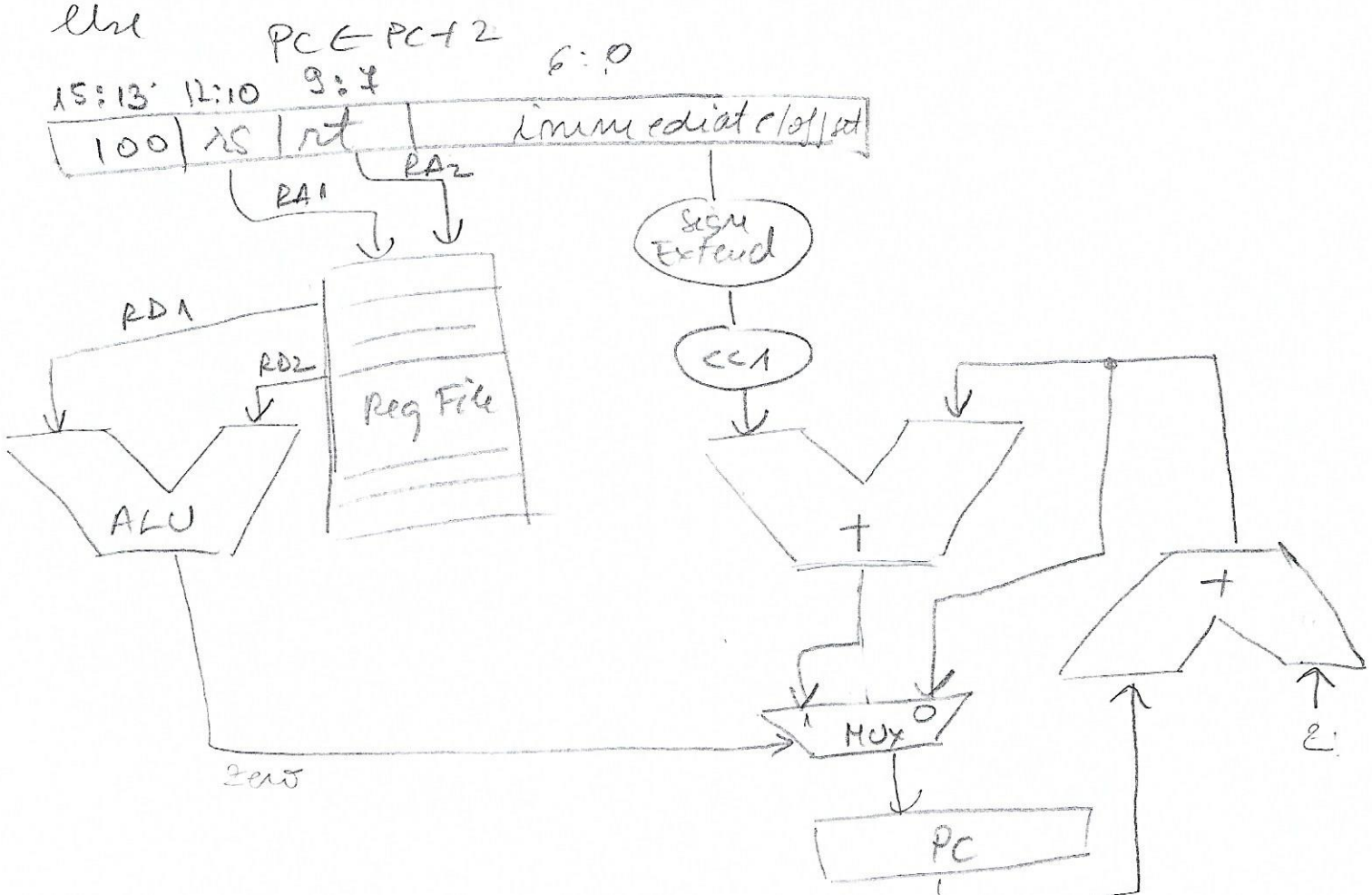
Branch if Equal

→ beq $rt, $rs, imm

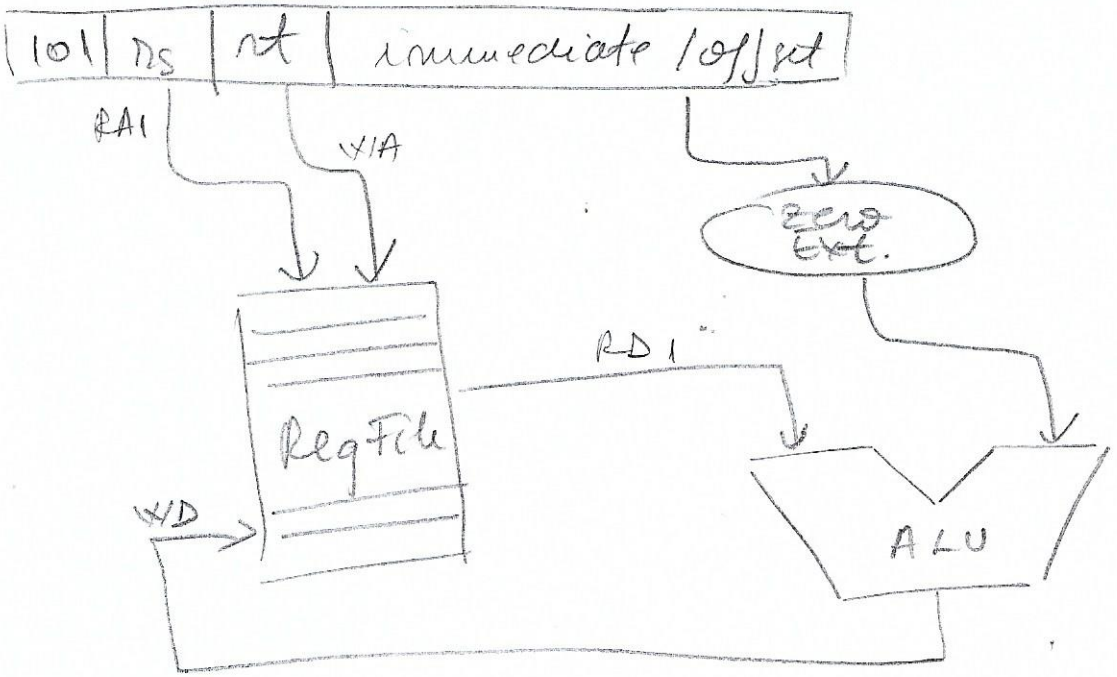
if (RF[rs] == RF[rt]) then

PC ← PC + 2 + S_Ext(imm) << 1

else PC ← PC + 2



→ andi $rt, $rs, imm RF[rt] ← RF[rs] & Z_Ext(imm)



→ ori $rt, $rs, imm RF[rt] ← RF[rs] | Z_Ext(imm)

| 110 | rs | rt | immediate / offset |

uguale a andi

R: | opcode | rs | rt | rd | sa | function |

000 - rs - rt - rd - 0 000 - add

0 001 - sub

1 010 - sll

1 011 - srl

0 100 - and

0 101 - or

0 110 - xor

0 111 - addu

000

000

000

000

000

000

000

I | opcode | rs | rt | address/immediate |

3 3 3 4

001 - addi

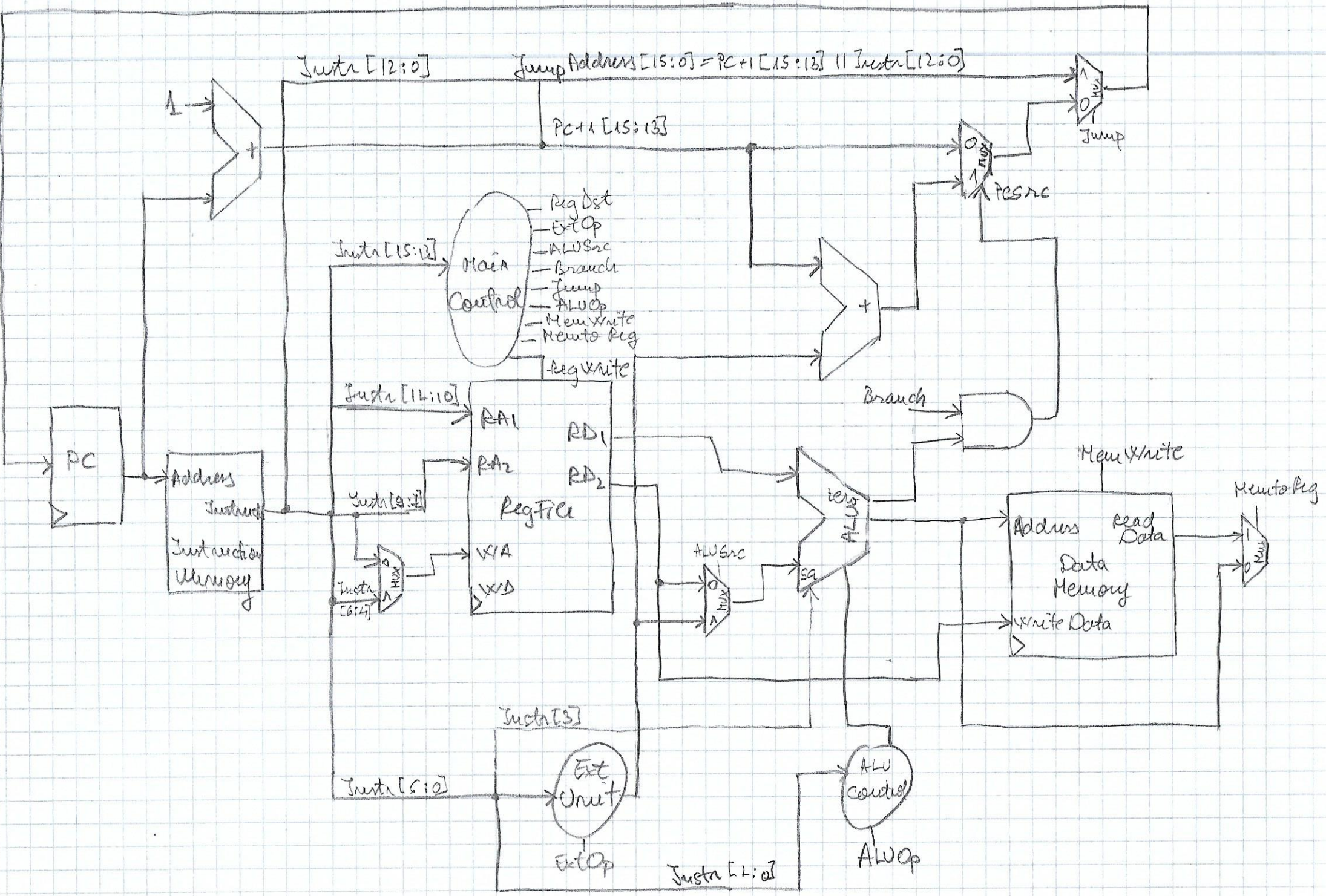
010 - lw

011 - sw

100 - beq

101 - andi

110 - ori


Instr [12:0]
Jump Address [15:0] = PC+1 [15:13] || Instr [12:0]
1
+
PC+1 [15:13]
M
u
x
0
1
Jump
PCSrc
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
Main Control
Instr [15:13]
RegWrite
+
Instr [11:0]
Branch
PC
Address
Instruction
Instruction Memory
RA1
RA2
WA
WD
RD1
RD2
RegFile
Instr [9:7]
Instr [6:4]
ALUSrc
zero
ALU
SA
MemWrite
MemtoReg
Address
Read Data
Data Memory
Write Data
Instr [3]
Instr [6:0]
Ext Unit
ExtOp
ALU control
ALUOp
Instr [1:0]

```
0.  add  $1, $0, $0   // i = 0
1.  addi $4, $0, 7
2.  add  $2, $0, $0   // init index loc. mem
3.  add  $5, $0, $0   // sum = 0
4.  addi $6, $0, 1
5.  beq  $1, $4, end_loop (9)
6.  lw   $3, A_addr($2)
7.  and  $6, $6, $3
8.  beq  $6, $0, daca e par ie adun (2)   // in $6 am 1
9.  addi $2, $2, 1
10. addi $1, $1, 1
11. j begin loop (5)
12. add  $5, $5, $6
13. addi $6, $0, 1
14. addi $2, $2, 1
15. addi $1, $1, 1
16. j begin loop (5)
17. sw   $5, sum_addr($7)
```

$1 = i
$5 = sum
$4 = M. elem
$3 = el. curent
$6 sum 1

elem. curent $1

daca e impar trec la urmat. element doar

daca e par il adun

end loop

Codul în C

```
int a[7] = {2,3,4,5,6,7,8};
int sum = 0;
for (int i = 0; i < 7; i++)
{
    if (a[i] % 2 == 0)
    {
        sum = sum + a[i];
    }
}
```

Semnale control MIPS16 pentru Anexa 5

<?> ∈ {_gez, _ne, _gtz}

Tipuri de operații care se pun în paranteză la ALUOp si ALUCtrl: {(+), (-), (&), (|), (^), (<<l), (<<lv), (>>l), (>>a), (<)}, & - AND, | - OR, ^ - XOR, l - logic, a - aritmetic, v - cu variabilă

Instrucțiune	Opcode *Instr(15-13)*	RegDst	ExtOp	ALUSrc	Branch	<Br?> (optional)	Jump	JmpR (optional)	Mem Write	Memto Reg	Reg Write	ALUOp (1:0)	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	1	0	0		0		0	0	1	000(<<)	010	010
SRL	000	1	1	0	0		0		0	0	1	000(>>)	011	011
AND	000	1	0	0	0		0		0	0	1	000(AND)	100	100
OR	000	1	0	0	0		0		0	0	1	000(OR)	101	101
XOR	000	1	0	0	0		0		0	0	1	000(XOR)	110	110
ADDU	000	1	0	0	0		0		0	0	1	000(ADDU)	111	111
ADDI	001	0	1	1	0		0		0	0	1	001(+)	-	000
LW	010	0	1	1	0		0		1	1	1	001(+)	-	000
SW	011	0	1	1	0		0		1	0	0	001(+)	-	000
BEQ	100	0	1	0	1		0		0	0	0	010(-)	-	001
ANDI	101	0	1	1	0		0		0	0	1	101	-	100
ORI	110	0	1	1	0		0		0	0	1	110	-	101
J	111	-	-	-	0		1		-	0	0	-	-	-

```
B"000_000_000_001_0_000",  --add $1,$0,&0    X 10
B"001_000_100_0000111",    --addi $4,$0,7    X 2207
B"000_000_000_010_0_000",  --add $2,$0,&0    X 20
B"000_000_000_101_0_000",  --add $5,$0,&0    X 50
B"001_000_110_0000001",    --addi &6,&0,1    X 2301

B"100_001_100_0001011",     --beq $1,$4,11         X 860b
B"010_010_011_0000000",     --lw &3,A_addr($2)     X 4980
B"000_011_110_110_0_100",   -- and $6,$6,$3        X 0F64

B"100_110_000_0000011",    --beq $6,$0,3          X 9803
B"001_010_010_0000001",    --addi $2,$2,1         X 2901
B"001_001_001_0000001",    --addi $1,$1,1         X 2481
B"111_0000000000101",       --J 5                 X E005

B"000_110_101_101_0_000",  --add $5,$5,&6         X 1AD0
B"001_000_110_0000001",    --addi $6,$0,1         X 2301
B"001_010_010_0000001",    --addi $2,$2,1         X 2901
B"001_001_001_0000001",    --addi $1,$1,1         X 2481
B"111_0000000000101",       --J 5                 X E005

B"011_111_101_0000000",   --sw $5,sum_addr($7)    X 7E80
```

Trasarea execuției programului de test pentru MIPS16

Pas	SW(7:5)	"000"	"001"	"010"	"011"	"100"	"101"	"110"	"111"	De completat numai pentru instrucțiuni de salt	
	Instr (*în asamblare*)	Instr (*hexa*)	PC+1	RD1	RD2	Ext_Imm	ALURes	MemData	WD	BranchAddr	JumpAddr
0	add $1,$0,$0	10	1	0	0	10	0	2	0		
1	addi $4,$0,7	2207	2	0	7	7	7	0	7		
2	add $2,$0,$0	20	3	0	0	20	0	2	0		
3	add $5,$0,$0	50	4	0	0	50	0	2	0		
4	addi $6,$0,1	2301	5	0	1	1	1	3	1		
5	beq $1,$4,11	860b	6	0	7	9	FFF9	0	FFF9		
6	lw $3,A-addr($2)	4980	7	0	2	0	0	2	2		
7	and $6,$6,$3	0F64	8	2	1	64	0	2	0		
8	beq $6,$0,3	9803	9	0	0	2	0	2	0		
9	add $5,$5,$6	1Ad0	d	0	0	50	0	2	0		
10	addi $6,$0,1	2301	E	0	0	1	1	3	1		
11	addi $2,$2,1	2901	F	0	0	1	1	3	1		
12	addi $1,$1,1	2481	10	0	0	1	1	3	1		
13	J 5	E005	11	0	0	5	0	2	0		
14	beq $1,$4,11	860b	6	1	7	6	FFFA	0	FFFA		
15	lw $3,A-addr($2)	4980	7	1	2	0	1	3	3		
	and $6,$6,$3	0F64	8	3	1	64	1	3	1		