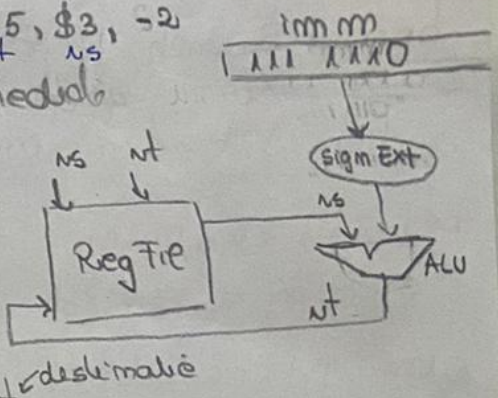
addi 111 011 101 1111110 ⇔ addi $5, $3, -2
-ns- -nt-
addi $t, $s, imm
; Aduno valoarea din registru ns cu valoare imediată
⇒ rezultatul în nt

RF[nt] ← RF[ns] + S_Ext(imm)

PC ← PC + 1



nt = destinație

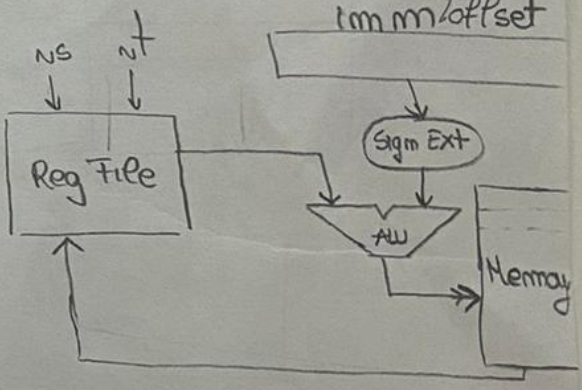
lw 100 011 101 0000010 ⇔ lw $5, 2($3)
ns nt offset

load word - încarcă cuvânt din memorie într-un registru
↳ încarcă în $nt cuvântul aflat în memorie la adresa efectivă ($ns + offset)

RF[nt] ← M[RF[ns] + S_EXT(imm)]

PC ← PC + 1

lw $t, imm($s)



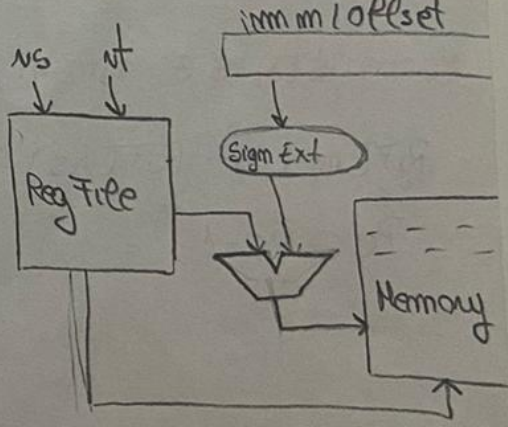
sw

store word - stochează cuvântul din $nt în memorie la adresa efectivă ($ns + offset)

M[RF[ns] + S_EXT(imm)] ← RF[nt]

PC ← PC + 1

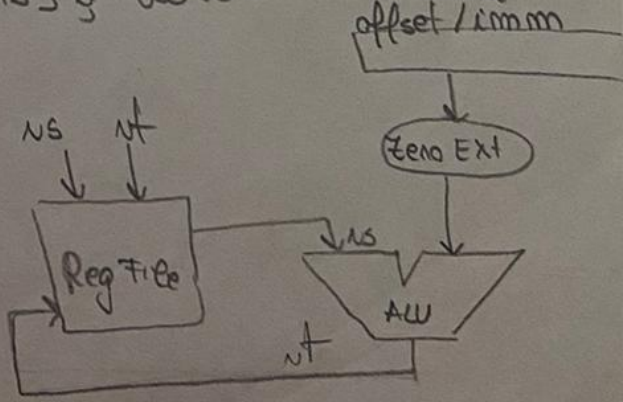
sw $t, imm($s)



Ori $t, $s, imm
⇒ op „SAU" logic între perechi de biți RF[ns] și valoarea imediată

RF[nt] ← RF[ns] OR Z_EXT(imm)

PC ← PC + 1

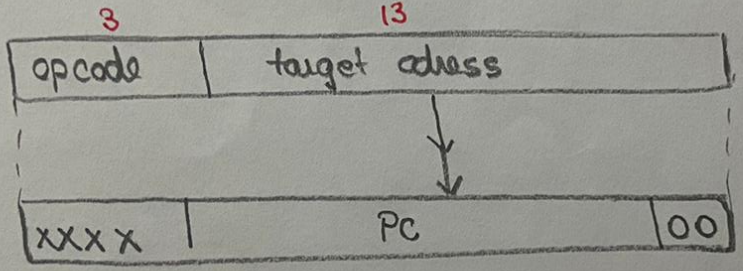


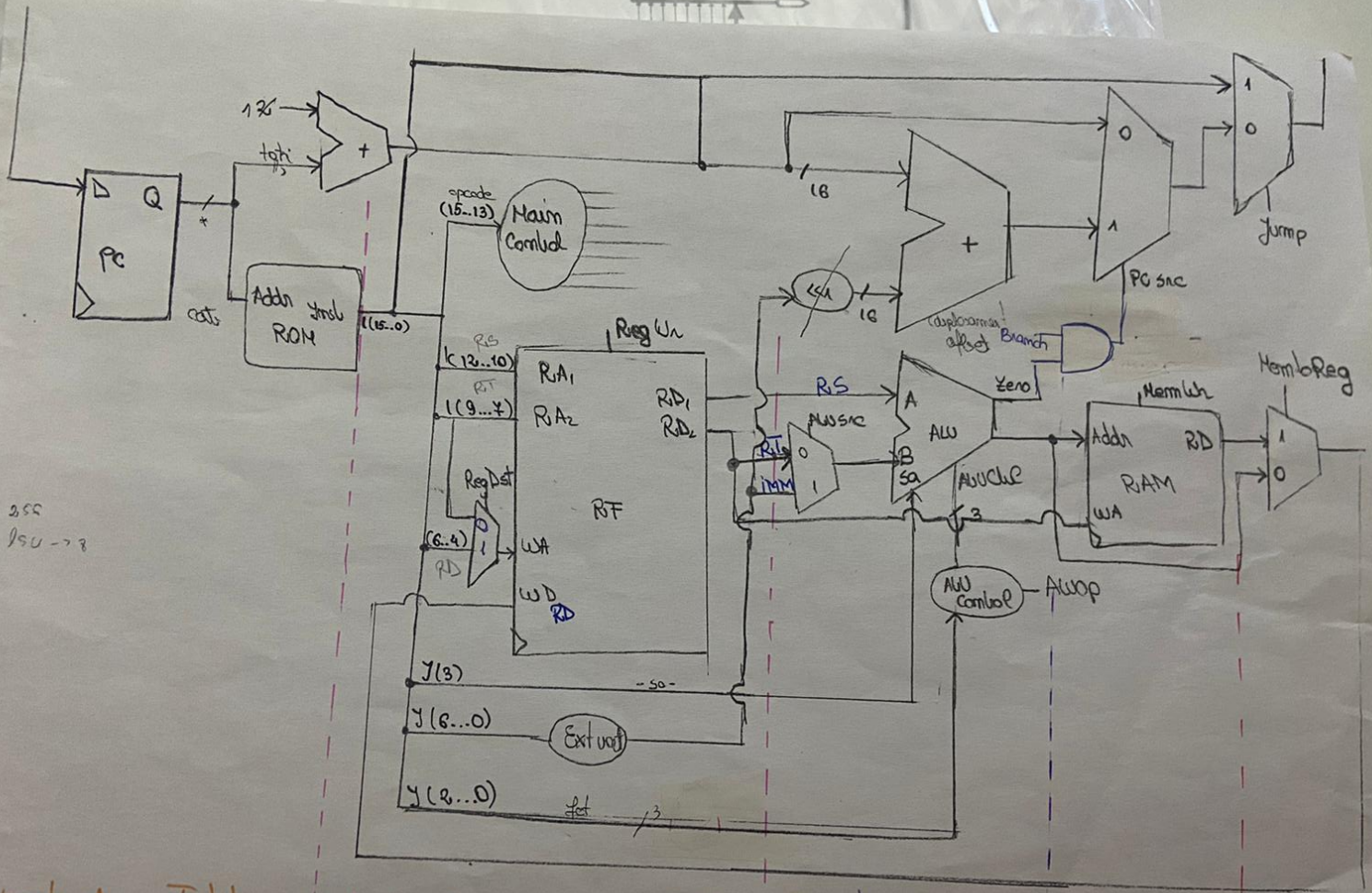
Tip J

→ Instrucțiunea Jump

31:28

j target_adress → PC : [15:13] & target_adress & 00.





IF- Instruction Fetch
- extragerea instrucțiunii

Instruction Decode & Operand Fetch.
- decodificarea instrucțiunii / extragerea operanzilor

Execute- EX
- execute

MEM
memory

Write Back
WB

Trasarea execuției programului de test pentru MIPS16

Valorile se completează în hexazecimal așa cum trebuie să apară pe SSD. Succesiunea pașilor reprezintă ordinea de execuție în timp la apăsarea butonului ENable. **Pasul 0 corespunde stării inițiale a circuitului (PC = 0), iar pasul *N* caracterizează starea după apăsarea de *N* ori a butonului ENable.** Inițial registrele vor avea valoarea 0 (care se atribuie automat în lipsa unei inițializări explicite a RF), iar memoria de date RAM poate fi inițializată cu valori dorite. Tabelul se completează pentru tot programul sau dacă are buclă până la finalul primei iterații. *Buclă = revenirea execuției la o instrucțiune care a mai fost executată anterior.*

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	ADDI ...$1, $0, 9	E089	x"0001"	0000	0000	0009	0009	0000	0009		
1	ADDI $4, $0, 24	E218	0002	0000	0000	0018	0018	0000	0018		
2	ADDI $5, $0, 0	E280	0003	0000	0000	0000	0000	0000	0000		
3	BEQ $1, $4, -3	D083	0004	0018	0009	0003	000F	0000	000F	0007	
4	ADDI $1, $1, 3	E483	0005	0009	0009	0003	000C	0000	000C		
5	ADD $5, $5, $1	14D0	0006	0000	000C	0050	000C	0000	000C		
6	j3	6003	0007	0000	0000	0003	0000	0000	0000		0003
7	BEQ $1, $4, 3	D083	0004	0018	0000	0003	000C	0000	000d	0007	
8	ADDI $1, $1, 3	E483	0005	000C	000C	0003	000F	0000	000F		
9	ADD $5, $5, $1	14D0	0006	000C	000F	0050	001b	0000	001b		
10	j3	6003	0007	0000	0000	0003	0000	0000	0000		0003
11	BEQ $1, $4, 3	D083	0004	0018	000F	0003	0009	0000	000A	0007	
12	ADDI $1, $1, 3	E483	0005	000F	000F	0003	0012	0000	0012		
13	ADD $5, $5, $1	14D0	0006	001b	0012	0050	002d	0000	002d		
14	j3	6003	0007	0000	0000	0003	0000	0000	0000		0003
15	BEQ $1, $4, 3	D083	0004	0018	0012	0003	0006	0000	0007	0007	

https://drive.google.com/file/d/1OgoST1-tEe1cbUdNk_VKr6NHq3zVfs83/view?usp=sharing

Trasarea execuției programului de test pentru MIPS16

Valorile se completează în hexazecimal așa cum trebuie să apară pe SSD. Succesiunea pașilor reprezintă ordinea de execuție în timp la apăsarea butonului *ENable*. **Pasul 0 corespunde stării inițiale a circuitului (PC = 0), iar pasul *N* caracterizează starea după apăsarea de *N* ori a butonului ENable.** Inițial registrele vor avea valoarea 0 (care se atribuie automat în lipsa unei inițializări explicite a RF), iar memoria de date RAM poate fi inițializată cu valori dorite. Tabelul se completează pentru tot programul sau dacă are buclă până la finalul primei iterații. *Buclă = revenirea execuției la o instrucțiune care a mai fost executată anterior.*

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**
16	ADDI .... $1 , $1, 3	E483	0005	0012	0012	0003	0015	0000	0015		
-	ADD $5, $5, $1	14D0	0006	002d	0015	0050	0042	0000	0042		
-	j 3	6003	0007	0000	0000	0003	0000	0000	0000		0003
	BEQ $1, $4, 3	D083	0004	0018	0015	0003	0003	0000	0000	0007	
	ADDi $1, $1, 3	E483	0005	0015	0015	0003	0018	0000	0018		
	ADD $5, $5, $1	14D0	0006	0042	0018	0050	005A	0000	005A		
	j 3	6003	0007	0000	0000	0003	0000	0000	0000		0003
	BEQ $1, $4, 3	D083	0004	0018	0018	0003	0000	0000	0000		
	SW $5, 10($0)	228A	0008	0000	005A	000A	000A	005A	000A		
10											
11											
12											
13											
14											
15											

URL: https://drive.google.com/file/d/1OgoST1-tEe1cbUdNk_VKr6NHq3zVfs83/view?usp=sharing

Trasarea execuției programului de test pentru MIPS16

Valorile se completează în hexazecimal așa cum trebuie să apară pe SSD. Succesiunea pașilor reprezintă ordinea de execuție în timp la apăsarea butonului ENable. **Pasul 0 corespunde stării inițiale a circuitului (PC = 0), iar pasul *N* caracterizează starea după apăsarea de *N* ori a butonului ENable**. Inițial registrele vor avea valoarea 0 (care se atribuie automat în lipsa unei inițializări explicite a RF), iar memoria de date RAM poate fi inițializată cu valori dorite. Tabelul se completează pentru tot programul sau dacă are buclă până la finalul primei iterații. *Buclă = revenirea execuției la o instrucțiune care a mai fost executată anterior.*

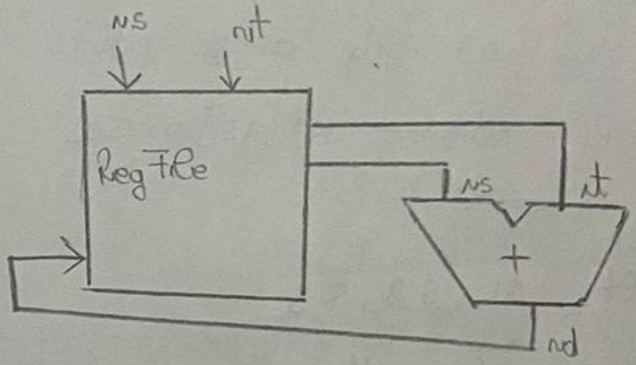
Pas	SW(7:5)	"000"	"001"	"010"	"011"	"100"	"101"	"110"	"111"	De completat numai pentru instrucțiuni de salt	
	Instr (*în asamblare*)	Instr (*hexa*)	PC+1	RD1	RD2	Ext_Imm	ALURes	MemData	WD	BranchAddr	JumpAddr
0	ADDI ...$1, $0, 9	x"E089"	x"0001"	x"0000"	x"0000"	x"0009"	x"0009"	x"0000"	x"0009"		
1	ADDI $4, $0, 24	x"E218"	x"0002"	x"0000"	x"0000"	x"0018"	x"0018"	x"0000"	x"0018"		
2	ADDI $5, $0, 0	x"E280"	x"0003"	x"0000"	x"0000"	x"0000"	x"0000"	x"0000"	x"0000"		
3	BEQ $1, $4, 3	x"D083"	x"0004"	x"0018"	x"0009"	x"0003"	x"0007"	x"0000"	x"0007"	x"0007"	
4	ADDI $1, $1, 3	x"E483"	x"0005"	x"0009"	x"0009"	x"0003"	x"000C"	x"0000"	x"000C"		
5	ADDI $5, $5, $1	x"1480"	x"0006"	x"0000"	x"000C"	x"0050"	x"000C"	x"0000"	x"000C"		
6	j 3	x"6003"	x"0007"	x"0000"	x"0000"	x"0008"	x"0000"	x"0000"	x"0000"		x"0003"
7	SW $5, 10($0)	x"2A8A"	x"0008"	x"0000"	x"005A"	x"000A"	x"000A"	x"005A"	x"000A"		
8	// instrucțiunea SW se exec la finalul programului; după ce se calculează										
9	suma										
10											
11											
12											
13											
14											
15											

URL: https://drive.google.com/file/d/1OgoST1-tEe1cbUdNk_VKr6NHq3zVfs83/view?usp=sharing

add $1, $2, $3 ↔ 000 010 011 001 0 000
(rd, rs, rt)
(rs, rt, rd)

- adună 2 registre și memorează rezultatul în al treilea. (rd)

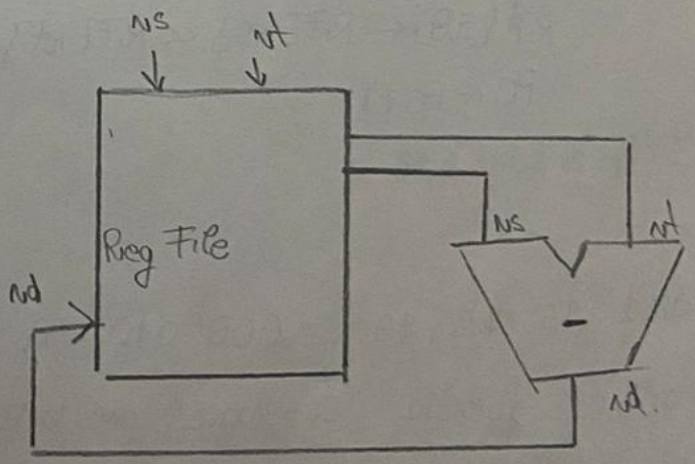
RTL: RF[rd] ← RF[rs] + RF[rt]
PC ← PC+1
add $d, $s, $t



sub $4, $2, $3 ↔ 000 010 011 100 001

- scade al 2-lea registru din primul și memorează rezultatul în $rd.

RF[rd] ← RF[rs] - RF[rt]
PC ← PC+1
sub $d, $s, $t



		7	6	5	4	3	2	1	0	(10:8)		
Instructiune	Opcode	RegDst	ExtOp	AluSRc	Branch	Jump	MemWr	MemtoReg	RegWr	AluOp	funct	AluCtrl
add	000	1	0 (X)	0	0	0	0	0	1	000	000	000 +
sub	000	1	0 (X)	0	0	0	0	0	1	000	001	001 -
sll	000	1	0 (X)	X?	0	0	0	0	1	000	011	010 <
srl	000	1	0 (X)	X?	0	0	0	0	1	000	100	011 >
and	000	1	0 (X)	0	0	0	0	0	1	000	010	100 &
or	000	1	0 (X)	0	0	0	0	0	1	000	101	101 |
xor	000	1	0 (X)	0	0	0	0	0	1	000	110	110 ^
sllv	000	1	0 (X)	0	0	0	0	0	1	000	111	111 >>
~~sra~~												
addi	111	0	1	1	0	0	0	0	1	001 +	-	000 +
lw	100	0	1	1	0	0	0	1	1	001 +	-	000 +
sw	001	0	1	1	0	0	1	0	0	001 +	-	000 +
beq	110	0	1	0	1	0	0	0	0	010 -	-	001 -
ori	101	0	0 (X)	1	0	0	0	0	1	011 OR	-	101 |
xori	010	0	0 (X)	1	0	0	0	0	1	100 XOR	-	110 ^
j	011	0 (X)	0 (X)	0 (X)	0	1	0	0	0	000 (X)	-	

Reg dst < 0 (RT[rt])
1 (RT[rd])

ALU SRC < 0 - RT
1 - immediate

raw imb in B

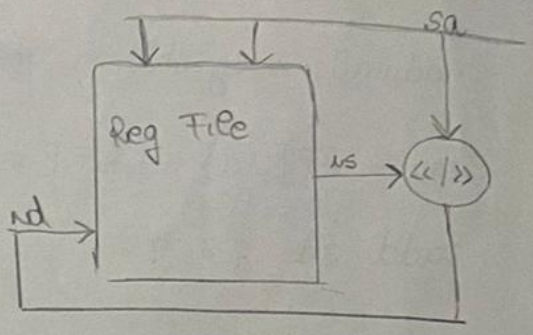
sll $t, $s, 0 — 000 010 000 111 0 011 (sa)

- shifteaza bitii cu sa pozitii la stanga

sll $d, $s, sa

RF[rd] ← RF[rs] << sa

PC ← PC+1



srl $t, $s, 0 — 000 010 000 101 0 100

- shifteaza bitii cu sa pozitii la dreapta

RF[rd] ← RF[rs] >> sa

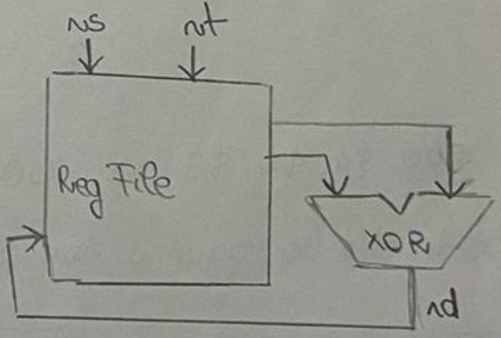
PC ← PC+1

XOR $1, $2, $3 — 000 010 011 001 0 110

RF[rd] ← RF[rs] ^ RF[rt]

PC ← PC+1

XOR $d, $s, $t

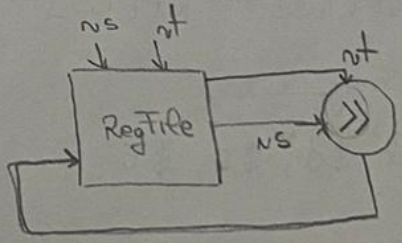


srlv $rd, $rs, $rt

RF[rd] ← RF[rs] << RF[rt]

PC ← PC+1

srlv $d, $s, $t



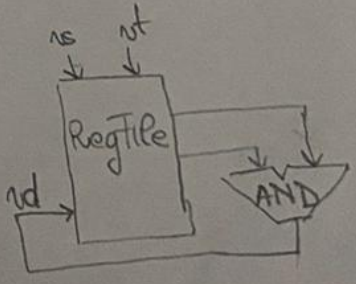
and $5, $2, $3 — 000 010 011 101 0 100

- aplica functia "si-logic" pe biti din reg rs si rt => pune rez in rd

and $d, $s, $t

RF[rd] ← RF[rs] & RF[rt]

PC ← PC+1



or $6, $2, $3 — 000 010 011 110 0 101

- aplica functia "SAU-logic" pe biti din reg rs si rt => pune rez in rd

RF[rd] ← RF[rs] | RF[rt]

PC ← PC+1

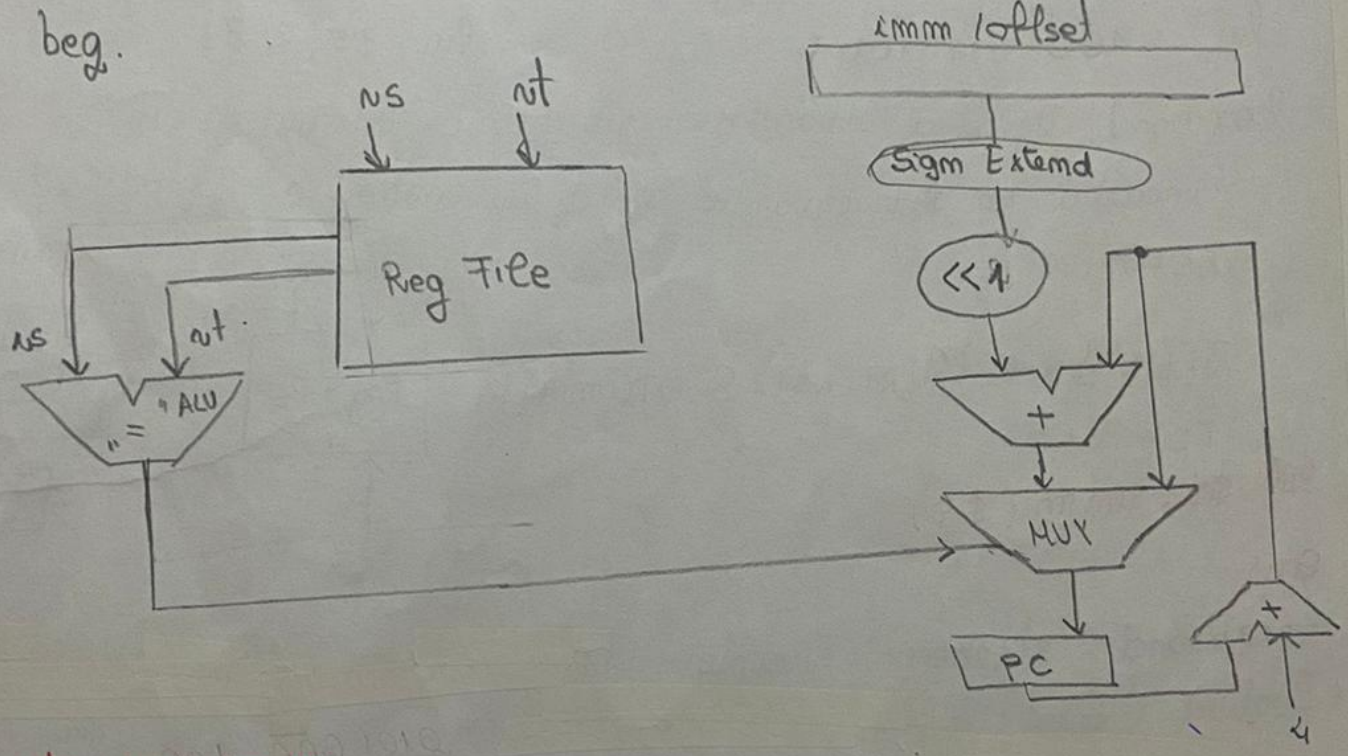
beg - Branch If Equal

110 011 010 0000010 → beg $2, $3, 2
(ns, nt)

beg $nt, $ns, imm → if (RF[ns] == RF[nt])
then PC ← PC + 4 + S_Ext(imm)<<2

beg, $s, $t, imm

- compară 2 regiștri și în cazul de „=" execută un salt peste „imm+1" instrucțiuni față de instrucțiunea care urmează în program după beg.



xoriu $t, $s, imm

RF[nt] ← RF[ns] ^ Z_EXT(imm)

PC ← PC+1.