

# **Procesorul MIPS, ciclu unic**

## **– versiune pe 16 biți –**

**Student: Marcu Ariana-Mălina**

**Grupa: 30222**

## 1. Instrucțiuni alese suplimentar



### *Instrucțiunea xor (Bitwise Exclusive OR)*

- instrucțiune de tip R
- realizează operația de sau-exclusiv între conținutul a două registre
- xor \$d, \$s, \$t
- 



### *Instrucțiunea slt (Set on Less Than)*

- instrucțiune de tip R
- setează registrul destinație atunci când conținutul primului registru sursă este mai mic decât conținutul celui de-al doilea registru sursă
- slt \$d, \$s, \$t



### *Instrucțiunea bgez (Branch on Greater Than or Equal to Zero)*

- instrucțiune de tip I
- efectuează un salt condiționat la o adresă relativă la adresa instrucțiunii următoare dacă registrul sursă are conținutul mai mare sau egal cu 0
- sintaxa: bgez \$s, offset



### *Instrucțiunea bne (Branch on not equal)*

- instrucțiune de tip I
- efectuează un salt condiționat la o adresă relativă la adresa instrucțiunii următoare dacă două registre sunt diferite
- sintaxa: bne \$s, \$t, offset

## 2. Tabel cu valorile semnalelor de control pentru setul de instrucțiuni selectat

Tip	Instruction	Opcode	Reg Dst	Ext Op	ALU SRC	Branch	Br Gez	Br Ne	Jump	Mem Wr	Mem to Reg	Reg Wr	func	Alu ctrl	Alu Op
R	ADD	000	1	any	0	0	0	0	0	0	0	1	011	011	000
	SUB	000	1	any	0	0	0	0	0	0	0	1	000	000	000
	SLL	000	1	any	0	0	0	0	0	0	0	1	010	010	000
	SRL	000	1	any	0	0	0	0	0	0	0	1	100	100	000
	AND	000	1	any	0	0	0	0	0	0	0	1	110	110	000
	OR	000	1	any	0	0	0	0	0	0	0	1	111	111	000
	SLT	000	1	any	0	0	0	0	0	0	0	1	101	101	000
	XOR	000	1	any	0	0	0	0	0	0	0	1	001	001	000
i	ADDI	001	0	1	1	0	0	0	0	0	0	1	x	011	001
	LW	101	any	1	1	0	0	0	0	0	1	1	x	011	101
	SW	010	any	1	1	0	0	0	0	1	any	0	x	011	010
	BEQ	110	any	1	0	1	0	0	0	0	any	0	x	000	110
	BNE	100	any	1	0	0	0	1	0	0	any	0	x	000	100
	BGEZ	011	any	1	0	0	1	0	0	0	0	0	x	000	011
	JUMP	111	any	any	any	0	any	any	1	0	any	0	x	x	x

### 3. Descrierea în cuvinte, cod C și cod mașină a programului încărcat în memoria ROM

Codul C: cmmdc dintre valoarea obținută ca suma numerelor dintr-un interval și un număr dat pe care îl scriu din memorie.

```
int a=0, x=3, y=7, b;
for (int i=x; i<=y; i++)
    a=a+i;
scanf("%d", &b);
while (a != b)
    if (a>b) a=a-b;
    else b=b-a;
printf("%d", a);
```

ex: [3,7]  $\Rightarrow$  a=25, b=5  $\Rightarrow$  cmmdc = 5

	J			i			R	Hexa	
0	000	000	000	001	0	011	0013	ADD	
1	101	000	010	000	000	0	A100	LW	+
2	101	000	011	000	000	1	A181	LW	+
3	101	000	100	000	001	0	A202	LW	x
4	000	000	000	101	0	011	0053	ADD	
5	110	010	011	000	001	1	C983	BEQ	
6	000	001	010	001	0	011	0513	ADD	
7	001	010	010	000	000	1	2901	ADDI	
8	111	000	000	000	101	1	E005	J	
9	110	100	001	000	011	0	D086	BEQ	
10	000	100	001	101	0	101	10D5	SLT	
11	100	101	000	000	001	0	9402	BNE	
12	000	001	100	001	0	000	0610	SUB	
13	111	000	000	000	100	1	E009	J	
14	000	100	001	100	0	000	10C0	SUB	
15	111	000	000	000	100	1	E009	J	
16	010	000	100	000	001	1	4203	SW	

- buton rus plăcuță - iterăm PC
- cu sw (0) se face jump la x'0000' initial
- cu sw (1) - to set the PC value to the branch address x'0004'
- cu sw (11) - illustrează valoarea PC+1 pe SSD



#### 4. Trasarea execuției programului

$Suma = a + (a+1) + (a+2) + \dots + (b-1)$

$3+4+5+6+7 = 25$  [3x8]

0. add \$1, \$0, \$0 → \$1 suma nr din intervalul [a, b)

1. lui \$2, 0(\$0) → \$2 = a (a se află în memorie la adresa 0(\$0))

2. lui \$3, 1(\$0) → \$3 = b (b se află în memorie la adresa 1(\$0))

3. lui \$4, 2(\$0) → \$4 = nr cu care se va face emmdc dintre el și suma (și se află în memorie la adresa 2(\$0))

4. add \$5, \$0, \$0 → \$5 se folosește pt verificare dacă suma e mai mare decât nr sau invers

5. beg \$2, \$3, 3 → verificare dacă a = b

6. add \$1, \$1, \$2 → adunare element din interval la suma

7. addi \$2, \$2, 1 → incrementare a

8. j 5 → jump la linia 5 (la începutul buclei) până când ajung la finalul intervalului (la b)

9. beg \$4, \$1, 6 → dacă \$4 = \$1, se sare la linia 16 (salvare în memorie)

10. slt \$5, \$4, \$1 → \$5 = 1 dacă \$4 < \$1 (ca să știm cum se face scăderea - din nr mai mare)

11. bne \$5, \$0, 2 → dacă \$5 != 0, se execută ramura 2 (la adresa 14)

12. sub \$1, \$1, \$4 → \$1 = \$1 - \$4

13. j 9 → jump linia 9

14. sub \$4, \$4, \$1 → \$4 = \$4 - \$1

15. j 9 → întoarcere în buclă

16. lui \$4, 3(\$0) → se salvează emmdc (care se află atât în \$4 cât și în \$1, se poate pune oricare) la adresa 3(\$0) în memorie

Din asamblare în cod mașină

0 add \$1, \$0, \$0

1 lui \$2, \$0, addr = a (\$0)

2 lui \$3, \$0, addr = b (\$0)

3 lui \$4, \$0, addr = nr (\$0)

4 add \$5, \$0, \$0

5 begin\_loop: beg \$2, \$3, end\_loop

6 add \$1, \$1, \$2

7 addi \$2, \$2, 1

8 j begin\_loop

9 end\_loop:

10 emmdc: beg \$4, \$1, emmdc\_end

11 slt \$5, \$4, \$1

12 bne \$5, \$0, linia\_14

13 sub \$1, \$1, \$4

14 j emmdc

15 linia\_14: sub \$4, \$4, \$1

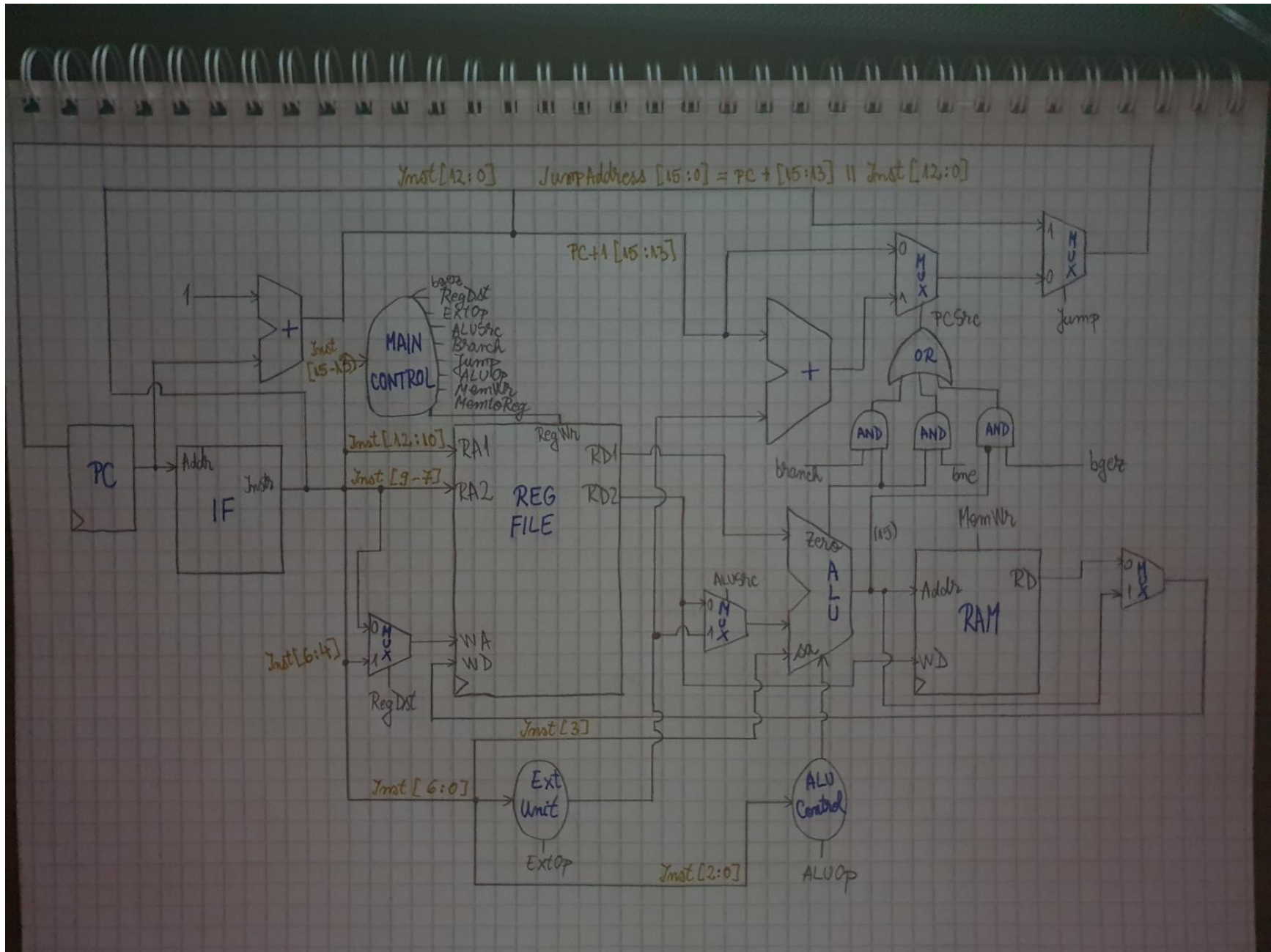
16 j emmdc

17 emmdc\_end: lui \$4, \$0, emmdc = addr (\$0)

9. PC+1 (ca să treacă la adresa următoare dacă nu se îndeplinește condiția)

11. PC+1+2 = 11+1+2 = 14

5. RTL schematic – in PDF-ul atașat + imaginea alaturata



## 6. Trasarea execuției programului de test pentru MIPS16

- sw(7:5) = 000 - se afișează instrucțiunea pe SSD
- sw(7:5) = 001 - se afișează următoare valoare secvențială a PC (PC + 1), pe SSD
- sw(7:5) = 010 - se afișează RD1 pe SSD
- sw(7:5) = 011 - se afișează RD2 pe SSD
- sw(7:5) = 100 - se afișează Ext\_Imm pe SSD
- sw(7:5) = 101 - se afișează ALURes pe SSD
- sw(7:5) = 110 - se afișează MemData pe SSD
- sw(7:5) = 111 - se afișează WD pe SSD.

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.

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(rs)	RD2(rt)	Ext_Imm	ALURes	MemData	WD	BranchAddr	JumpAddr
0	add \$1, \$0, \$0	X"0013"	X"0001"	X"0000"	X"0000"	-	X"0000"	X"0000"	X"0000"		
1	lw \$2, 0(\$0)	X"A100"	X"0002"	X"0000"	X"0000"	X"0000"	X"0000"	X"0000"	X"0000"		
2	lw \$3, 1(\$0)	X"A181"	X"0003"	X"0000"	X"0000"	X"0001"	X"0001"	X"0000"	X"0000"		
3	lw \$4, 2(\$0)	X"A202"	X"0004"	X"0000"	X"0000"	X"0002"	X"0002"	X"0003"	X"0003"		
4	add \$5, \$0, \$0	X"0053"	X"0005"	X"0000"	X"0000"	-	X"0000"	X"0000"	X"0000"		
5	beq \$2, \$3, 3	X"C983"	X"0006"	X"0003"	X"0008"	X"0003"	X"0003"	X"0008"	X"0003"	X"0009"	
6	add \$1, \$1, \$2	X"0513"	X"0007"	X"0000"	X"0003"	-	X"0003"	X"0008"	X"0003"		
7	addi \$2, \$2, 1	X"2901"	X"0008"	X"0003"	X"0004"	X"0001"	X"0003"	X"0008"	X"0003"		
8	j 5	X"E005"	X"0009"	X"0000"	X"0000"	-	-	-	-		X"0005"

9	beq \$4, \$1, 6	X"D086"	X"000A"	X"0005"	X"0019"(25)	X"0006"	X"0005"	X"0000"	X"0005"	X"0010"	
10	slt \$5, \$4, \$1	X"10D5"	X"000B"	X"0001"	X"0005"	-	X"0006"	-	X"0006"		
11	bne \$5, \$0, 2	X"9402"	X"000C"	X"0001"	X"0000"	X"0002"	X"0003"	X"0008"	X"0003"	X"000E"	
12	sub \$1, \$1, \$4	X"0610"	X"000D"	X"0019"	X"0005"	-	X"0014"	-	X"0014"		
13	j 9	X"E009"	X"000E"	X"0000"	X"0000"	-	-	-	-		X"0009"
14	sub \$4, \$4, \$1	X"10C0"	X"000F"	X"0005"	X"0019"	X"0000"	X"0014"	-	X"0014"		
15	j 9	X"E009"	X"0010"	X"0000"	X"0000"	-	-	-	-		X"0009"
16	sw \$4, 3(\$0)	X"4203"	X"0011"	X"0000"	X"0005"(rez	X"0003"	X"0005"	X"0001"	X"0001"		

Programul a fost testat si functioneaza bine totul pana la InstructionDecode inclusiv, insa la testarea finala, de la primul beq, nu mai functioneaza corect, doar partial.