Procesor MIPS 32
-ciclu unic-

nume: Rujac Roxana
grupa: 30224

1. Componente

Procesorul implementat rezolva problema găsirii celui mai mic număr impar dintr-un șir de numere. Pentru aceasta am avut nevoie de următoarele componente:
-o unitate de control (UC)
-o memorie (MEM)
-un generator monopols (MPG)
-un afisor pe 7 segmente (SSD)
-o unitate de decodificare a instrucțiunilor (ID)
-o unitate de extragere a instrucțiunilor (IFetch)
-o unitate de execuție a instrucțiunilor (EX)

Toate acestea sunt componente folosite pentru a crea arhitectura completa a procesorului în fișierul test_env.

2. Programul in limbaj de asamblare:

```
0:      xor     $1, $0, $0          --initializarea contorului cu 0
1:      lw      $2, 8($0)           --initializez minimul impar cu cel mai mare numar
2:      lw      $3, 4($0)           --am citit n de la adresa 4 (numarul de numere
stocate)
3:      lw      $4, 0($0)           --citesc adresa de inceput a sirului A de la adresa 0
(pointer spre inceputul sirului)
4:      addi    $5, $0, $0          --folosesc $5 pentru a parcurge sirul ($5 - indexul
sirului, A[index])

--bucla

5:      beq     $1, $3, 10          --verific daca contorul <= n si daca nu sar la finalul
buclei
6:      lw      $6, 12($5)          --luam elementul curent $6 = A[index]
7:      addi    $7, $0, 1           --incarc valoarea 1 in registrul 7
8:      and     $8, $6, $7          --and intre numarul curent si valoarea 1 in $8
9:      beq     $8, $0, 3           --daca rezultatul din $8 e 0 atunci e nr par sarim la
instructiunile de incrementare
10:     slt     $9, $6, $2          --verifica daca numarul actual (impar) e mai mic
decat minimul actual
11:     beq     $9, $0, 1           --daca nr nu este mai mic decat minimul curent
sarim la instructiunile de incrementare
12:     add     $2, $6, $0          --salvam minimul curent in $2
```

```
13:     addi    $1, $1, 1           --incrementam contorul
14:     addi    $5, $5, 4           --actualizam pozitia din sir

--afara

15:     j       5                   --revine la inceputul buclei
16:     sw      $2, 8($0)           --pune in memorie la adresa 8 rezultatul -> minimul impar
```

3. Instructiuni adaugate:

1. XOR

Descriere: SAU-Exclusiv logic între două registre, memorează rezultatul în alt registru
RTL: $d $s ^ $t; PC PC + 4;
Sintaxă asamblare: xor $d, $s, $t
Format: 000000 sssss ttttt ddddd 00000 100110

2. SLT

Descriere: Dacă $s < $t, $d este inițializat cu 1, altfel cu 0
RTL: PC PC + 4; if $s < $t then $d 1 else $d 0;
Sintaxă asamblare: slt $d, $s, $t
Format: 000000 sssss ttttt ddddd 00000 101010

3. BGEZ

Descriere: Salt condiționat dacă un registru este mai mare sau egal cu 0
RTL: if $s ≥ 0 then PC (PC + 4) + (SE(offset) << 2) else PC PC + 4;
Sintaxă asamblare: bgez $s, offset
Format: 000001 sssss 00000 oooooooooooooooo

4. SLTI

Descriere: Dacă $s este mai mic decât un imediat, $t este inițializat cu 1, altfel cu 0
RTL: PC PC + 4; if $s < SE(imm) then $t 1 else $t 0;
Sintaxă asamblare: slti $t, $s, imm
Format: 001010 sssss ttttt iiiiiiiiiiiiiiii

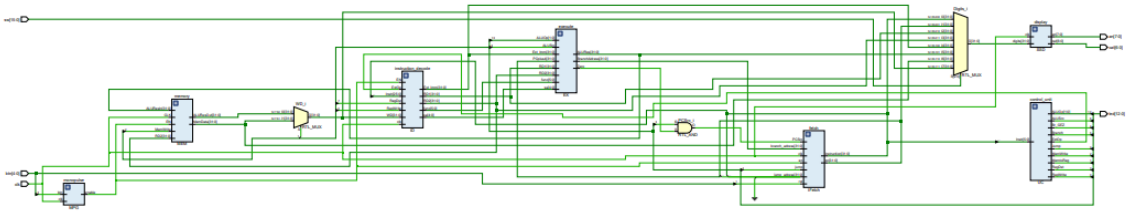
4. Semnalele de control pentru instrucțiuni

Instrucțiune	Opcode *Instr[31-26]*	RegDst	ExtOp	ALUSrc	Branch	BrGEZ	Jump	JmpR (optional)	Mem Write	Memto Reg	Reg Write	ALUOp[1:0]	function *Instr[5-0]*	ALUCtrl[2:0]
ADD	000000	1	x	0	0	0	0		0	0	1	00	000000	000(+)
SUB	000000	1	x	0	0	0	0		0	0	1	00	000001	001(-)
SLL	000000	1	x	0	0	0	0		0	0	1	00	000010	010(<<)
SRL	000000	1	x	0	0	0	0		0	0	1	00	000011	011(>>)
AND	000000	1	x	0	0	0	0		0	0	1	00	000100	100(&)
OR	000000	1	x	0	0	0	0		0	0	1	00	000101	101(|)
XOR	000000	1	x	0	0	0	0		0	0	1	00	000110	110(^)
SLT	000000	1	x	0	0	0	0		0	0	1	00	000111	001(-)
ADDI	000001	0	1	1	0	0	0		0	0	1	01	xxxxxx	000(+)
LW	000010	0	1	1	0	0	0		0	1	1	01	xxxxxx	000(+)
SW	000011	x	1	1	0	0	0		1	x	0	01	xxxxxx	000(+)
BEQ	000100	x	1	0	1	0	0		0	x	0	10	xxxxxx	001(-)
BGEZ	000101	x	1	0	0	1	0		0	x	0	10	xxxxxx	001(-)
SLTI	000110	0	1	1	0	0	0		0	1	1	10	xxxxxx	001(-)
J	000111	x	x	x	x	x	1		0	x	0	xx	xxxxxx	xxx

5. Trasarea execuției programului

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+4	RD1	RD2	Ext_Imm	ALURes	MemData	WD	BranchAddr	JumpAddr
0	xor $1, $0, $0	X"00000806"	X"00000004"	X"0"	X"0"	X"00000806"	X"00000000"	X"0000000C"	X"00000000"	X	X
1	lw $2, 8($0)	X"08020008"	X"00000008"	X"00000000"	X"00000005"	X"00000008"	X"00000008"	X"00000005"	X"00000005"	X	X
2	lw $3, 4($0)	X"08030004"	X"0000000C"	X"00000000"	X"00000008"	X"00000004"	X"00000004"	X"00000008"	X"00000008"	X	X
3	lw $4, 0($0)	X"08040000"	X"00000010"	X"00000000"	X"0000000C"	X"00000000"	X"00000000"	X"0000000C"	X"0000000C"	X	X
4	addi $5, $0, 0	X"04050000"	X"00000014"	X"00000000"	X"00000020"	X"00000000"	X"00000000"	X"0000000C"	X"00000000"	X	X
5	beq $1, $3, 10	X"1023000A"	X"00000018"	X"00000000"	X"00000008"	X"0000000A"	X"FFFFFFF8"	X"00000000"	X"FFFFFFF8"	X"0000000A"	X
6	lw $6, 12($5)	X"08A6000C"	X"0000001C"	X"00000000"	X"0000000A"	X"0000000C"	X"0000000C"	X"00000002"	X"00000002"	X	X
7	addi $7, $0, 1	X"04070001"	X"00000020"	X"00000000"	X"00000001"	X"00000001"	X"00000001"	X"0000000C"	X"00000001"	X	X
8	and $8, $6, $7	X"00C74004"	X"00000024"	X"00000002"	X"00000001"	X"00004004"	X"00000000"	X"0000000C"	X"00000000"	X	X
9	beq $8, $0, 3	X"11000003"	X"00000028"	X"00000000"	X"00000000"	X"00000003"	X"00000000"	X"0000000C"	X"00000000"	X"00000011"	X
10	slt $9, $6, $2	X"00C24807"	X"0000002C"	X"0000000F"	X"00000005"	X"00004807"	X"00000000"	X"0000000C"	X"00000000"	X	X
11	beq $9, $0, 1	X"11200001"	X"00000030"	X"00000000"	X"00000000"	X"00000001"	X"00000000"	X"0000000C"	X"00000000"	X"00000001"	X
12	add $2, $6, $0	X"00C01000"	X"00000034"	X"0000000F"	X"0000000"	X"00001000"	X"0000000F"	X"00000002"	X"0000000F"	X	X
13	addi $1, $1, 1	X"04210001"	X"00000038"	X"00000000"	X"0000000"	X"00000001"	X"00000001"	X"0000000C"	X"00000001"	X	X
14	addi $5, $5, 4	X"04A50004"	X"0000003C"	X"00000000"	X"00000000"	X"00000004"	X"00000004"	X"00000008"	X"00000004"	X	X
15	j 5	X"01C00000"	X"00000040"	X"00000000"	X"00000000"	X"00000005"	X"00000000"	X"0000000C"	X"00000000"	X	X"00000018"
16	sw $2, 8($0)	X"00C02000"	X"00000044"	X"00000000"	X"00000005"	X"00000008"	X"00000008"	X"00000005"	X"00000008"	X	X

6. Schema RTL



7. Evaluare functionalitate

Toate componentele sunt funcționale și comunică cum ar trebui între ele. Programul este testat integral pe placuta, atat componentele separate cat și programul final. În timpul implementării acestui proiect au apărut mai multe probleme care mi-au incetinit progresul: scrierea greșită a instrucțiunilor, faptul ca nu am inteles mereu la momentul potrivit ce scop ar avea unele componente, faptul ca nu am putut testa pe placuta de la laborator de fiecare data cate o componentă separată iar acest lucru s-a dovedit mai dificil de făcut la final.
Cu toate acestea, problemele au fost rezolvate iar programul funcționează cum ar trebui.