- CSE 331 -

Computer Organization

HW4 Report

(Esra Eryılmaz 171044046)

Different version of 32-bit MIPS processor using Altera Quartus II with Verilog :

- You said that; You should send whatever you did and defend yourself. So I send whatever I did. 🙃

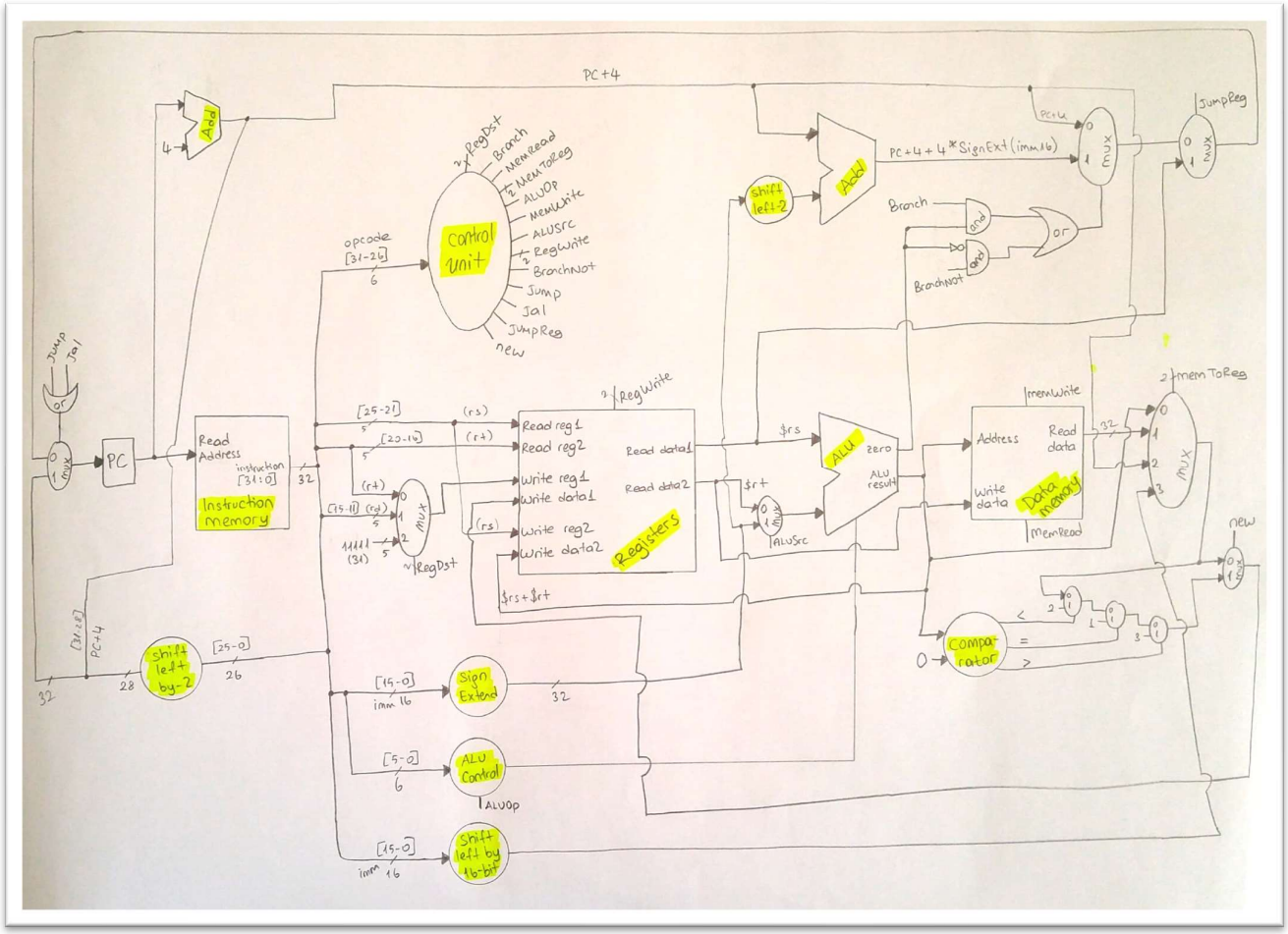
- First of all it is not a complete homework.

- Actually I designed datapath and I did truth tables and boolean expressions...

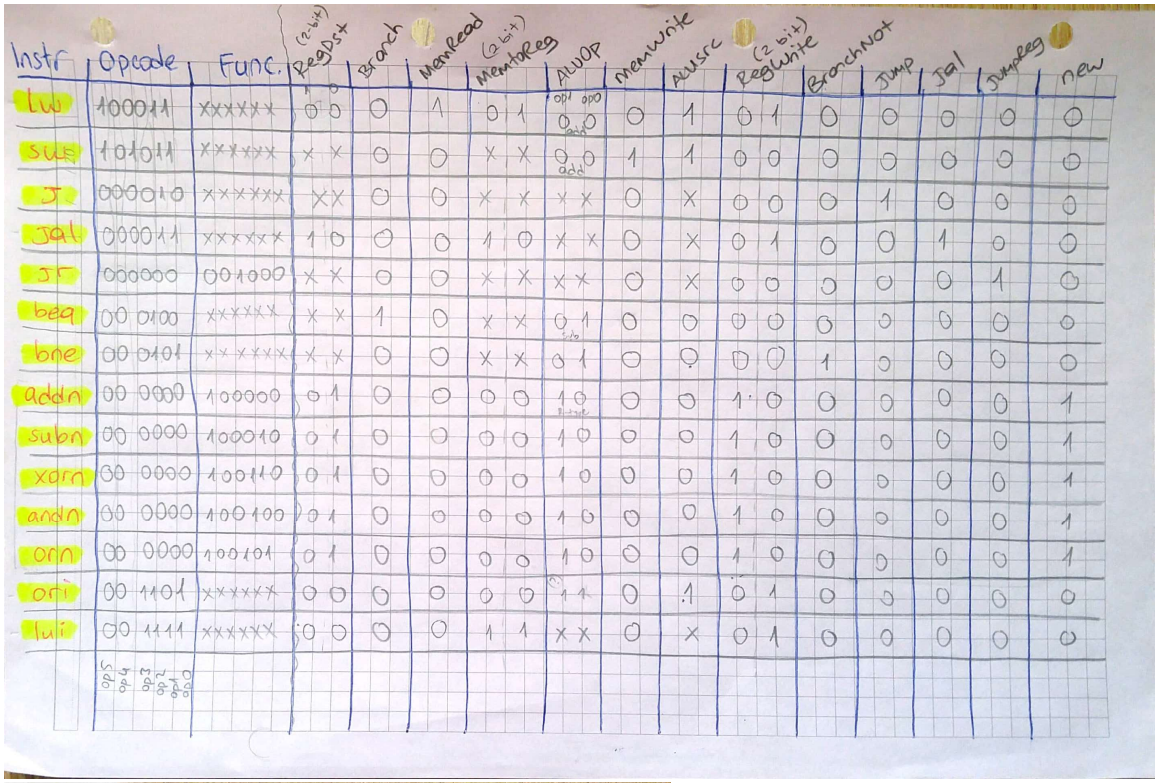
- But in Verilog I couldn't manage the modules especially which needs clock. I couldn't figure it out. I could not put the pieces together.
  So instructions does not work.

- I designed sub modules and I put screenshots of the tests of these sub modules.

➢ Datapath Design :



➢ Truth Tables and Boolean expressions :



Instr	Opcode	Func.	RegDst (2 bit)	Branch	MemRead	MemtoReg (2 bit)	ALUOP	MemWrite	ALUSrc	RegWrite (2 bit)	BranchNot	Jump	Jal	JumpReg	new
lw	100011	xxxxxx	0 0	0	1	0 1	0 0	0	1	0 1	0	0	0	0	0
sw	101011	xxxxxx	x x	0	0	x x	0 0	1	1	0 0	0	0	0	0	0
J	000010	xxxxxx	xx	0	0	x x	x x	0	x	0 0	0	1	0	0	0
jal	000011	xxxxxx	1 0	0	0	1 0	x x	0	x	0 1	0	0	1	0	0
JR	000000	001000	x x	0	0	x x	x x	0	x	0 0	0	0	0	1	0
beq	00 0100	xxxxxx	x x	1	0	x x	0 1	0	0	0 0	0	0	0	0	0
bne	00 0101	xxxxxx	x x	0	0	x x	0 1	0	0	0 0	1	0	0	0	0
addn	00 0000	100000	0 1	0	0	0 0	1 0	0	0	1 0	0	0	0	0	1
subn	00 0000	100010	0 1	0	0	0 0	1 0	0	0	1 0	0	0	0	0	1
xorn	00 0000	100110	0 1	0	0	0 0	1 0	0	0	1 0	0	0	0	0	1
andn	00 0000	100100	0 1	0	0	0 0	1 0	0	0	1 0	0	0	0	0	1
orn	00 0000	100101	0 1	0	0	0 0	1 0	0	0	1 0	0	0	0	0	1
ori	00 1101	xxxxxx	0 0	0	0	0 0	1 1	0	1	0 1	0	0	0	0	0
lui	00 1111	xxxxxx	0 0	0	0	1 1	x x	0	x	0 1	0	0	0	0	0

$RegDst[1] = op5' \cdot op4' \cdot op2' \cdot op1 \cdot op0$

$RegDst[0] = op0'$

$Branch = op3' \cdot op2 \cdot op0$

$MemRead = op5 \cdot op3'$

$MemToReg[1] = op5' \cdot op1 \cdot op0$

$MemToReg[0] = op5 + op2 \cdot op1 \cdot op0$

$ALUOp[1] = op2' op1' op0' + op3 \cdot op2 \cdot op1' \cdot op0$

$ALUOp[0] = op3' op2 + op1' \cdot op0$

$MemWrite = op3 \cdot op2' \cdot op1$

$ALUSrc = op5 + op3\ op2\ op1'$

$RegWrite[1] = op2' op1' op0'$

$RegWrite[0] = op5' op1\ op0 + op5 \cdot op3' + op3\ op2$

$BranchNot = op3' op2\ op0$

$Jump = op2' op1\ op0'$

$Jal = op5' \cdot op2'\ op1\ op0$

$JumpReg = op2' op1' op0'$

$new = op2' op1' op0'$

ALU Control Truth Table

	Inputs								Output			
Instruc.	ALUOp		Function Field						ALU control			
	ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	ALUsel2	ALUsel1	ALUsel0	
lw	0	0	x	x	x	x	x	x	0	1	0	add
sw	0	0	x	x	x	x	x	x	0	1	0	add
j	x	x	x	x	x	x	x	x	x	x	x	–
jal	x	x	x	x	x	x	x	x	x	x	x	–
jr	x	x	0	0	1	0	0	0	x	x	x	–
beq	0	1	x	x	x	x	x	x	1	0	0	sub
bne	0	1	x	x	x	x	x	x	1	0	0	sub
addn	1	0	1	0	0	0	0	0	0	1	0	add
subn	1	0	1	0	0	0	1	0	1	0	0	sub
xorn	1	0	1	0	0	1	1	0	0	1	1	xor
andn	1	0	1	0	0	1	0	0	0	0	0	and
orn	1	0	1	0	0	1	0	1	0	0	1	or
ori	1	1	x	x	x	x	x	x	0	0	1	or
lui	x	x	x	x	x	x	x	x	x	x	x	–

$ALUsel[2] = ALUOp1' \cdot ALUOp0 + ALUOp1' \cdot ALUOp0 + ALUOp1 \cdot ALUOp0' \cdot F5 \cdot F4' \cdot F3' \cdot F2' \cdot F1 \cdot F0'$

$ALUsel[1] = ALUop1' \cdot ALUop0' + ALUOp1 \cdot ALUOp0' \cdot F5 \cdot F4' \cdot F3' \cdot F2' F1' F0' + ALUop1 \cdot ALUop0' \cdot F5 \cdot F4' \cdot F3' \cdot F2 \cdot F1 \cdot F0'$

$ALUsel[0] = ALUOp1 \cdot ALUop0' \cdot F5 \cdot F4' \cdot F3' \cdot F2 \cdot F1 \cdot F0' + ALUop1 \cdot ALUOp0' \cdot F5 \cdot F4' \cdot F3' \cdot F2 \cdot F1' \cdot F0 + ALUOp1 \cdot ALUOp0$

➢ Verilog Modules and their tests:

and_32bit

```
# Loading work.and_32bit_testbench
# Loading work.and_32bit
VSIM 25> step -current
# time= 0, a=00000000000000000000000000000011, b=00000000000000000000000000000001, output=00000000000000000000000000000001
# time=20, a=00000000000000000000000000000010, b=00100000000000000000000000000001, output=00000000000000000000000000000000
# time=40, a=11111111111111111111111111111111, b=11111111111111111111111111111111, output=11111111111111111111111111111111
```

full_adder_1bit

```
# Loading work.full_adder_1bit_testbench
# Loading work.full_adder_1bit
VSIM 27> step -current
# time= 0, a=0, b=0, c_in=0, sum=0, c_out=0
# time=20, a=0, b=0, c_in=1, sum=1, c_out=0
# time=40, a=0, b=1, c_in=0, sum=1, c_out=0
# time=60, a=0, b=1, c_in=1, sum=0, c_out=1
# time=80, a=1, b=0, c_in=0, sum=1, c_out=0
# time=100, a=1, b=0, c_in=1, sum=0, c_out=1
# time=120, a=1, b=1, c_in=0, sum=0, c_out=1
# time=140, a=1, b=1, c_in=1, sum=1, c_out=1
```

full_adder_32bit

```
# Loading work.full_adder_32bit_testbench
# Loading work.full_adder_32bit
# Loading work.full_adder_1bit
VSIM 29> step -current
# time= 0, a=00000000000000000000000000000011, b=00000000000000000000000000000001, c_in=1, sum=00000000000000000000000000000010, c_out=1
# time=20, a=00000000000000000000000000000010, b=00100000000000000000000000000001, c_in=0, sum=00100000000000000000000000000011, c_out=0
# time=40, a=11111111111111111111111111111111, b=11111111111111111111111111111111, c_in=0, sum=11111111111111111111111111111110, c_out=1
```

mux_2x1

```
#         Region: /mux_2x1_testbench/mymux
VSIM 31> step -current
# time= 0, i0=00000000000000000000000000000011, i1=00000000000000000000000000000001, a0=1, output=zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1
# time=20, i0=00000000000000000000000000000010, i1=00100000000000000000000000000001, a0=0, output=zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz0
# time=40, i0=11111111111111111111111111111111, i1=11111111111111111111111111111111, a0=0, output=zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1
# time=60, i0=01111111111111111111111111111111, i1=00000000000000000000000000000001, a0=1, output=zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1
# time=80, i0=00000000000000000000000000111111, i1=00000000000000000000000000111011, a0=0, output=zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz1
```

mux_4x1

```
# Loading work.mux_4x1
VSIM 33> step -current
# time= 0, i0=11100000000001110000000000000011, i1=00000000000000000000000000000001, i2=11100000000000000000000000000011, i3=00000000001110000000000000000111 ,s=00, out=11100000000001110000000000000011
# time=20, i0=11100000000001110000000000000011, i1=00000000000000000000000000000001, i2=11100000000000000000000000000011, i3=00000000001110000000000000000111 ,s=01, out=00000000000000000000000000000001
# time=40, i0=11100000000001110000000000000011, i1=00000000000000000000000000000001, i2=11100000000000000000000000000011, i3=00000000001110000000000000000111 ,s=10, out=11100000000000000000000000000011
# time=60, i0=11100000000001110000000000000011, i1=00000000000000000000000000000001, i2=11100000000000000000000000000011, i3=00000000001110000000000000000111 ,s=11, out=00000000011100000000000000000111
```

or_32bit

```
# Loading work.or_32bit_testbench
# Loading work.or_32bit
VSIM 35> step -current
# time= 0, a=00000000000000000000000000000011, b=00000000000000000000000000000001, output=00000000000000000000000000000011
# time=20, a=00000000000000000000000000000010, b=00100000000000000000000000000001, output=00100000000000000000000000000011
# time=40, a=11111111111111111111111111111111, b=11111111111111111111111111111111, output=11111111111111111111111111111111
```

sign_extend

```
# Loading work.sign_extend
VSIM 37> step -current
# imm16=1000000101010101, sign_extended32=11111111111111111000000101010101
# imm16=0000000000001111, sign_extended32=00000000000000000000000000001111
```

xor_32bit

```
# Loading work.xor_32bit
VSIM 39> step -current
# time= 0, a=00000000000000000000000000000011, b=00000000000000000000000000000001, output=00000000000000000000000000000010
# time=20, a=00000000000000000000000000000010, b=00100000000000000000000000000001, output=00100000000000000000000000000011
# time=40, a=11111111111111111111111111111111, b=11111111111111111111111111111111, output=00000000000000000000000000000000
```