CSE 331 - Spring 2021

Homework 4 Report

Coşkun Hasan ŞALTU
1801042631


```
initial begin
    $readmemb("C:/altera/13.1/mips32/simulation/modelsim/instructions.mem", _instruction_memory);

end
```

To read file you need to add file path!!!

```
a > 13.1 > mips32 > simulation > modelsim > instructions.mem
0000000000000000 //nop inst
0000001010100001 // add $4,$1,$2
0001100111000101 // addi $7,$4,5
0001000110000100 // addi $6,$zero,4
0011000101111111 // ori $5,$zero,1
0101100011111100 // beq $3,$4,-4
0000000000000000 //nop inst
```

While reading the instructions, it is necessary to add one nop instruction at the beginning and one at the end!!!
-This is the shortcoming of the project.

The project can perform all the operations given in the pdf.(r_type,addi,ori,beq,bneq,slti....)

1. **Explanation of Moduls and Project**

and4: This module gets two 4-bit input and give the answer of and operation of two input.

and32: This module gets two 32-bit input and give the answer of and operation of two input.This module use **and4** module 8 times.

or4: This module gets two 4-bit input and give the answer of or operation of two input.

or32: This module gets two 32-bit input and give the answer of or operation of two input.This module use **or4** module 8 times.

xor4: This module gets two 4-bit input and give the answer of xor operation of two input.

xor32: This module gets two 32-bit input and give the answer of xor operation of two input.This module
use x**or4** module 8 times.

nor4: This module gets two 4-bit input and give the answer of nor operation of two input.

nor32: This module gets two 32-bit input and give the answer of nor operation of two input.This module use n**or4** module 8 times.

not4: This module gets two 4-bit input and give the answer of not operation of two input.

not32: This module gets two 32-bit input and give the answer of not operation of two input.This module use n**ot4** module 8 times.

and32_4_numbers: This module gets 4 32-bit input and give the answer of and operation of four input.This module use **and32** module 3 times.

or32_8_numbers: This module gets 8 32-bit input and give the answer of or operation of eight input.This module use **or32** module 7 times.

full_adder: This module gets two 1-bit input and give the answer of add operation of two input.

_4bit_adder: This module gets two 4-bit input and give the answer of add operation of two input.This module use **full_adder** module 4 times.

_32bit_adder: This module gets two 32-bit input and give the answer of add operation of two input.This module use **_4bit_adder** module 8 times.

_32bit_subtracter: This module gets two 32-bit input and give the answer of substract operation of two input.This module gets negative of second number using **xor32** module and **_32bit_adder** module.After that,it add two numbers.

slt_32: This module gets two 32-bit input and give the answer of set less than operation of two input.This module subtract second input from first input using **_32bit_subtracter**.Than,it decide that's operation using most significant bit of substraction operation.

number_1_to_32: This module gets two 1-bit input and it gives 32-bit version of this number.

m81: This module gets eight 32-bit input and 3-bit select input.It chooses true input using select inputs.Firstly,this module find 32-bit version of select inputs using **number_1_to_32**.Secondly,it find not of this select inputs using **not_32.**Finally it decide answer using **and32_4_numbers** and **or32_8_numbers**.

m21: This module gets two 32-bit input and 1-bit select input.It chooses true input using select inputs.

alu32: This module is main module.It gets two 32-bit number,one carry_in,3-bit select and it gives one 32-bir output and carry_out.This module calculate ADD,XOR,SUB,MULT,SLT,NOR,AND,OR operation respectively.Finally,this decides true output using **m81** module.

Test of alu32

```
VSIM 5> step -out -current
# time =  0, a =00000000000000000000000000001001, b=00000000000000000000000000000001, select = 000, out=00000000000000000000000000001010
# time = 20, a =00000000000000000000000000000101, b=00000000000000000000000000000100, select = 001, out=00000000000000000000000000000001
# time = 40, a =00000000000000000000000000001001, b=00000000000000000000000000000001, select = 010, out=00000000000000000000000000001000
# time = 60, a =00000000000000000000000000000111, b=00000000000000000000000000000101, select = 011, out=00000000000000000000000000100011
# time = 80, a =00000000000000000000000000001001, b=00000000000000000000000000000101, select = 100, out=00000000000000000000000000000000
# time = 100, a =00000000000000000000000000000101, b=00000000000000000000000000000100, select = 101, out=11111111111111111111111111111010
# time = 120, a =00000000000000000000000000001000, b=00000000000000000000000000001001, select = 110, out=00000000000000000000000000001000
# time = 140, a =00000000000000000000000000000101, b=00000000000000000000000000000001, select = 111, out=00000000000000000000000000000101
```

Second Test

```
VSIM 5> step -out -current
# time =  0, a =00000000000000000000000000001101, b=00000000000000000000000000001001, select = 000, out=00000000000000000000000000010110
# time = 20, a =00000000000000000000000000000101, b=00000000000000000000000000000100, select = 001, out=00000000000000000000000000000001
# time = 40, a =00000000000000000000000000001001, b=00000000000000000000000000001101, select = 010, out=11111111111111111111111111111100
# time = 60, a =00000000000000000000000000001111, b=00000000000000000000000000000101, select = 011, out=00000000000000000000000001001011
# time = 80, a =00000000000000000000000000001001, b=00000000000000000000000000000101, select = 100, out=00000000000000000000000000000000
# time = 100, a =00000000000000000000000000000101, b=00000000000000000000000000000101, select = 101, out=11111111111111111111111111111010
# time = 120, a =00000000000000000000000000001000, b=00000000000000000000000000001001, select = 110, out=00000000000000000000000000001000
# time = 140, a =00000000000000000000000000000101, b=00000000000000000000000000001001, select = 111, out=00000000000000000000000000000101
```

control_unit: This module gets 1 4-bit opcode input.It gives signals which are needed. (RegW,RegDst,Memtoreg,Branch,MR,MW,AluSrc and 3-bit aluop)

Test of control unit

```
# Time= 0,Opcode=0000,RegW=1,ALUSrc=0,RegDest=1,MemtoReg=0, MR=0, MW=0, Branch:0,ALUOp:100
# Time=100,Opcode=0001,RegW=1,ALUSrc=1,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:0,ALUOp:000
# Time=200,Opcode=0010,RegW=1,ALUSrc=1,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:0,ALUOp:001
# Time=300,Opcode=0011,RegW=1,ALUSrc=1,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:0,ALUOp:111
# Time=400,Opcode=0100,RegW=1,ALUSrc=1,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:0,ALUOp:011
# Time=500,Opcode=0101,RegW=0,ALUSrc=0,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:1,ALUOp:010
# Time=600,Opcode=0110,RegW=0,ALUSrc=0,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:1,ALUOp:010
# Time=700,Opcode=0111,RegW=1,ALUSrc=1,RegDest=0,MemtoReg=0, MR=0, MW=0, Branch:0,ALUOp:101
# Time=800,Opcode=1000,RegW=1,ALUSrc=1,RegDest=0,MemtoReg=1, MR=1, MW=0, Branch:0,ALUOp:000
# Time=900,Opcode=1001,RegW=0,ALUSrc=1,RegDest=0,MemtoReg=0, MR=0, MW=1, Branch:0,ALUOp:000
```

alu_control: This module gets one 3-bit func and one 3-bit aluop.It gives select input of alu.Alu32 takes select input from this module.

Test of alu control

```
# Time= 0,ALUOp:000,Func:xxx,Select:000
# Time=100,ALUOp:001,Func:xxx,Select:110
# Time=200,ALUOp:111,Func:xxx,Select:111
# Time=300,ALUOp:011,Func:xxx,Select:101
# Time=400,ALUOp:010,Func:xxx,Select:010
# Time=600,ALUOp:101,Func:xxx,Select:100
# Time=700,ALUOp:000,Func:xxx,Select:000
# Time=900,ALUOp:100,Func:000,Select:110
# Time=1000,ALUOp:100,Func:001,Select:000
# Time=1100,ALUOp:100,Func:010,Select:010
# Time=1200,ALUOp:100,Func:011,Select:001
# Time=1300,ALUOp:100,Func:100,Select:101
# Time=1400,ALUOp:100,Func:101,Select:111
```

sign_ext: This module takes 6-bit immediate bit and gives 32-bit version of immediates bit.

Test of sign_ext

```
# Time= 0,IMM:010101,Out:00000000000000000000000000010101
# Time=100,IMM:101010,Out:11111111111111111111111111101010
```

register: This module takes two 3-bit register address which read.It reads content of register from file using these register address.It takes 3-bit another register addres which wroten.It takes 32-bit writedata and one 1-bit signal.If the signal is 1,it write the 32-bit data to register which taken adress.

```
# Time= 0,ReadReg1:001,ReadReg2:011,WriteReg:010,WriteData:00000000000000000000000000001111,RegWrite:0,ReadData1:00000000000000000000000000000001,ReadData2:00000000000000000000000000000011
```

data_memory: This module takes two 1-bit signal,mw and mr.This module takes one 32-bit addres and if mr signal is 1,it gives 32-bit data from memory using this addres.Also,it takes 32-bit writedata.If mw signal is 1,it write 32-bit data to memory.

Test data memory

```
VSIM 5> step -out -current
# Time= 0,_adress:00000000000000000000000000000010,_writeData:00000000000000000000000000000110,_mr:1,mw:0,_readData:00000000000000000000000000010010
```

instruction_memory: This module takes 32-bit address(program counter) and it gives 16-bit instruction from instructions.mem file.

Test instruction memory

```
VSIM 9> step -out -current
# Time= 0,ReadData:00000000000000000000000000000000,_inst:0000000000000000
```

mips_32: This module is main module of project.It divide the instruction and send to needed components.

Test All instructions

```
0000000000000000 //nop inst
0000001010100001 // add $4,$1,$2
0000011010110000 // and $6,$3,$2
0000001010111010 // sub $7,$1,$2
0000111010111011 // xor $7,$7,$2
0000011010110100 // nor $6,$1,$2
0000011010110101 // or $6,$1,$2
0001000110000100 // addi $6,$zero,4
0010000110000100 // andi $6,$zero,4
0011001101111111 // ori $5,$1,1
0100000101000000 // nori $5,$zero,0
0101100011111100 // beq $3,$4,-4
0110110011000101 // bneq $6,$4,5
0111100011111100 // slti $3,$4,-4
1000100011111100 // lw $3,$4,-4
1001100011111100 // sw $3,$4,-4
0000000000000000 //nop inst
```

0000001010100001 // add $4,$1,$2

```
clock :0,Instruction: 0000001010100001
cPC:00000000000000000000000000000001,Opcode:0000,AluOP:100,Select:000,Rs:001,Rt:010,Rd:100,Func:001
ALUResult:00000000000000000000000000000011,ReadData1:00000000000000000000000000000001,ReadData2:00000000000000000000000000000010,Imm:11111111111111111111111111100001
RegWrite:1,RegDst:1,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:0
```

0000011010110000 // and $6,$3,$2

```
clock :0,Instruction: 0000011010110000
cPC:00000000000000000000000000000010,Opcode:0000,AluOP:100,Select:110,Rs:011,Rt:010,Rd:110,Func:000
ALUResult:00000000000000000000000000000010,ReadData1:00000000000000000000000000000011,ReadData2:00000000000000000000000000000010,Imm:11111111111111111111111111110000
RegWrite:1,RegDst:1,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:0
```

0000001010111010 // sub $7,$1,$2

```
clock :0,Instruction: 0000001010111010
cPC:00000000000000000000000000000011,Opcode:0000,AluOP:100,Select:010,Rs:001,Rt:010,Rd:111,Func:010
ALUResult:11111111111111111111111111111111,ReadData1:00000000000000000000000000000001,ReadData2:00000000000000000000000000000010,Imm:11111111111111111111111111111010
RegWrite:1,RegDst:1,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:0
```

0000111010111011 // xor $7,$7,$2

```
clock :0,Instruction: 0000111010111011
cPC:00000000000000000000000000000100,Opcode:0000,AluOP:100,Select:001,Rs:111,Rt:010,Rd:111,Func:011
ALUResult:11111111111111111111111111111101,ReadData1:11111111111111111111111111111111,ReadData2:00000000000000000000000000000010,Imm:11111111111111111111111111111011
RegWrite:1,RegDst:1,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:0
```

0000011010110100 // nor $6,$1,$2

```
clock :0,Instruction: 0000011010110100
cPC:00000000000000000000000000000101,Opcode:0000,AluOP:100,Select:101,Rs:011,Rt:010,Rd:110,Func:100
ALUResult:11111111111111111111111111111100,ReadData1:00000000000000000000000000000011,ReadData2:00000000000000000000000000000010,Imm:11111111111111111111111111110100
RegWrite:1,RegDst:1,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:0
```

0000011010110101 // or $6,$1,$2

```
clock :0,Instruction: 0000011010110101
cPC:00000000000000000000000000000110,Opcode:0000,AluOP:100,Select:111,Rs:011,Rt:010,Rd:110,Func:101
ALUResult:00000000000000000000000000000011,ReadData1:00000000000000000000000000000011,ReadData2:00000000000000000000000000000010,Imm:11111111111111111111111111110101
RegWrite:1,RegDst:1,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:0
```

0001000110000100 // addi $6,$zero,4

```
clock :0,Instruction: 0001000110000100
cPC:00000000000000000000000000000111,Opcode:0001,AluOP:000,Select:000,Rs:000,Rt:110,Rd:000,Func:100
ALUResult:00000000000000000000000000000100,ReadData1:00000000000000000000000000000000,ReadData2:00000000000000000000000000000011,Imm:00000000000000000000000000000100
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

0010000110000100 // andi $6,$zero,4

```
clock :0,Instruction: 0010000110000100
cPC:00000000000000000000000000001000,Opcode:0010,AluOP:001,Select:110,Rs:000,Rt:110,Rd:000,Func:100
ALUResult:00000000000000000000000000000000,ReadData1:00000000000000000000000000000000,ReadData2:00000000000000000000000000000100,Imm:00000000000000000000000000000100
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

0011001101111111 // ori $5,$1,1

```
clock :0,Instruction: 0011001101111111
cPC:00000000000000000000000000001001,Opcode:0011,AluOP:111,Select:111,Rs:001,Rt:101,Rd:111,Func:111
ALUResult:11111111111111111111111111111111,ReadData1:00000000000000000000000000000001,ReadData2:00000000000000000000000000000101,Imm:11111111111111111111111111111111
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

0100000101000000 // nori $5,$zero,0

```
clock :0,Instruction: 0100000101000000
cPC:00000000000000000000000000001010,Opcode:0100,AluOP:011,Select:101,Rs:000,Rt:101,Rd:000,Func:000
ALUResult:11111111111111111111111111111111,ReadData1:00000000000000000000000000000000,ReadData2:11111111111111111111111111111111,Imm:00000000000000000000000000000000
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

0101100011111100 // beq $3,$4,-4

```
clock :0,Instruction: 0101100011111100
cPC:00000000000000000000000000001011,Opcode:0101,AluOP:010,Select:010,Rs:100,Rt:011,Rd:111,Func:100
ALUResult:00000000000000000000000000000000,ReadData1:00000000000000000000000000000011,ReadData2:00000000000000000000000000000011,Imm:11111111111111111111111111111100
RegWrite:0,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:1,AluSrc:0
```

0110110011000101 // bneq $6,$4,5

```
clock :0,Instruction: 0001000110000100
cPC:00000000000000000000000000000111,Opcode:0001,AluOP:000,Select:000,Rs:000,Rt:110,Rd:000,Func:100
ALUResult:00000000000000000000000000000100,ReadData1:00000000000000000000000000000000,ReadData2:00000000000000000000000000000000,Imm:00000000000000000000000000000100
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

0111100011111100 // slti $3,$4,-4

```
clock :0,Instruction: 0010000110000100
cPC:00000000000000000000000000001000,Opcode:0010,AluOP:001,Select:110,Rs:000,Rt:110,Rd:000,Func:100
ALUResult:00000000000000000000000000000000,ReadData1:00000000000000000000000000000000,ReadData2:00000000000000000000000000000100,Imm:00000000000000000000000000000100
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

1000100011111100 // lw $3,$4,-4

```
clock :0,Instruction: 0011001101111111
cPC:00000000000000000000000000001001,Opcode:0011,AluOP:111,Select:111,Rs:001,Rt:101,Rd:111,Func:111
ALUResult:11111111111111111111111111111111,ReadData1:00000000000000000000000000000001,ReadData2:11111111111111111111111111111111,Imm:11111111111111111111111111111111
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

1001100011111100 // sw $3,$4,-4

```
clock :0,Instruction: 0100000101000000
cPC:00000000000000000000000000001010,Opcode:0100,AluOP:011,Select:101,Rs:000,Rt:101,Rd:000,Func:000
ALUResult:11111111111111111111111111111111,ReadData1:00000000000000000000000000000000,ReadData2:11111111111111111111111111111111,Imm:00000000000000000000000000000000
RegWrite:1,RegDst:0,MemtoReg:0,MR:0,MW:0,Branc:0,AluSrc:1
```

Truth Table of ALU Control

Inst	$A_2A_1A_0$ AluOp	$F_2F_1F_0$ Func	Alu Select
ADDI	000	xxxx	000
ANDI	001	xxxx	110
ORI	111	xxxx	111
NORI	011	xxxx	101
BEQ	010	xxxx	010
BNE	010	xxxx	010
SLTI	101	xxxx	100
LW	000	xxxx	000
SW	000	xxxx	000
R AND	100	000	110
R ADD	100	001	000
R SUB	100	010	010
R XOR	100	011	001
R NOR	100	100	101
R OR	100	101	111

I

$C_2 = A_2A_1'A_0'F_2F_1' + A_2A_1'A_0'F_2'F_1'F_0' + F_1F_0 + F_1'F_0$

$C_1 = A_2A_1'A_0'F_2'F_0' + A_2A_1'A_0'F_2F_1'F_0 + F_2'F_1F_0' + F_2'F_1'F_0 + F_2F_1F_0$

$C_0 = A_2A_1'A_0'F_2F_1' + A_2A_1'A_0'F_2'F_1F_0 + F_1F_0$