CSE 331/503 Computer Organization
Homework 4 Report

Berk PEKGÖZ
171044041

This report file is not for the mips32.v in general. Because there is no such circuit in this project. I couldn't complete the project because of the lackness of time. Only main modules (ALU, Control Unit, ALU Control Unit, Registers, Data Memory and Instruction Memory) and their testbenches implemented. This report shows each test for them.

Top level entity set as Control Module in this project. Each module can be tested with setting it as top-level entity.

There is a file named tut01 because I started on the project given by T.A.

ALU Module

To create a 32-bit ALU I need these sub-units:

- 2x1 Multiplexer (for 4x1 Mux)
- 4x1 Multiplexer (for 1-bit ALU)
- 1-bit ALU

2x1 Multiplexer (mux2x1.v) Test

mux2x1_testbench.v for this test.

```
# time =  0, in1=0, in0=0, slct=0, result=0
# time = 20, in1=0, in0=0, slct=1, result=0
# time = 40, in1=0, in0=1, slct=0, result=1
# time = 60, in1=0, in0=1, slct=1, result=0
# time = 80, in1=1, in0=0, slct=0, result=0
# time = 100, in1=1, in0=0, slct=1, result=1
# time = 120, in1=1, in0=1, slct=0, result=1
# time = 140, in1=1, in0=1, slct=1, result=1
```

For this test:

- In1 > Input 1
- In0 > Input 0

- slct > Select bit

4x1 Multiplexer (mux4x1.v) Test

mux4x1_testbench.v for this test.

```
# time =  0, in3=1, in2=0, in1=0, in0=0, slct=11, result=1
# time = 20, in3=0, in2=1, in1=0, in0=0, slct=10, result=1
# time = 40, in3=0, in2=0, in1=1, in0=0, slct=01, result=1
# time = 60, in3=0, in2=0, in1=0, in0=1, slct=00, result=1
# time = 80, in3=0, in2=1, in1=1, in0=1, slct=11, result=0
# time = 100, in3=1, in2=0, in1=1, in0=1, slct=10, result=0
# time = 120, in3=1, in2=1, in1=0, in0=1, slct=01, result=0
# time = 140, in3=1, in2=1, in1=1, in0=0, slct=00, result=0
```

For this test:

- In3 > Input 11
- In2 > Input 10
- In1 > Input 01
- In0 > Input 00
- slct > Select bits

Goal of this test is show that result gives 1 when the only 1 is selected input while others 0 and result gives 0 when the only 0 is selected input while others 1.

1 Bit ALU (alu_1_bit.v) Test

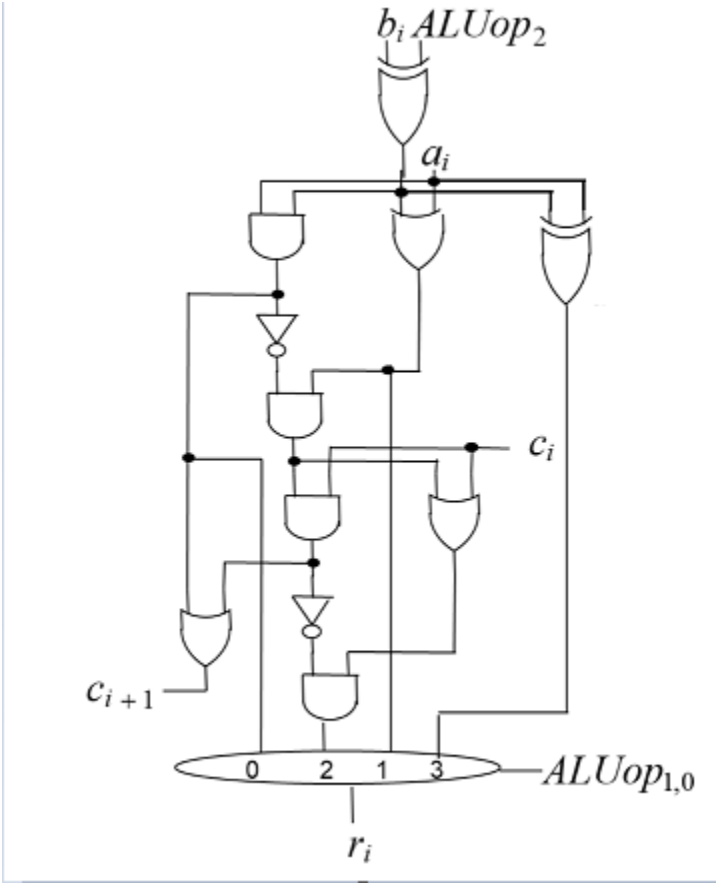
alu_1_bit_testbench.v for this test.

Op codes for this ALU:

- 000 – AND
- 001 – OR
- 010 – ADD
- 011 – XOR
- 110 – SUB

Design of ALU:

This design is very similar to the one which is shown in lecture. Only difference is extra XOR gate instead of Less input.



AND Test (000):
c_in does not affect the result on this operation.
c_out value is not important on this operation.

```
# time =  0, a=1, b=0, c_in=0, alu_code=000, result=0, c_out=0
#
# time = 20, a=0, b=1, c_in=0, alu_code=000, result=0, c_out=0
#
# time = 40, a=1, b=1, c_in=0, alu_code=000, result=1, c_out=1
#
# time = 60, a=0, b=0, c_in=0, alu_code=000, result=0, c_out=0
```

OR Test (001):
c_in does not affect the result on this operation.
c_out value is not important on this operation.

```
# time =  0, a=1, b=0, c_in=0, alu_code=001, result=1, c_out=0
#
# time = 20, a=0, b=1, c_in=0, alu_code=001, result=1, c_out=0
#
# time = 40, a=1, b=1, c_in=0, alu_code=001, result=1, c_out=1
#
# time = 60, a=0, b=0, c_in=0, alu_code=001, result=0, c_out=0
```

XOR Test (011):
c_in does not affect the result on this operation.
c_out value is not important on this operation.

```
# time = 80, a=1, b=0, c_in=0, alu_code=011, result=1, c_out=0
#
# time = 100, a=0, b=1, c_in=0, alu_code=011, result=1, c_out=0
#
# time = 120, a=1, b=1, c_in=0, alu_code=011, result=0, c_out=1
#
# time = 140, a=0, b=0, c_in=0, alu_code=011, result=0, c_out=0
#
```

ADD Test (010):
c_in does affect the result on this operation.
c_out value is important on this operation.

```
# time = 160, a=1, b=0, c_in=1, alu_code=010, result=0, c_out=1
#
# time = 180, a=0, b=1, c_in=1, alu_code=010, result=0, c_out=1
#
# time = 200, a=1, b=1, c_in=1, alu_code=010, result=1, c_out=1
#
# time = 220, a=0, b=0, c_in=1, alu_code=010, result=1, c_out=0
```

SUB Test (110):
c_in does affect the result on this operation.
c_out value is important on this operation.
Actual result will be correct with 32 x 1-bit combined.

```
# time = 240, a=1, b=0, c_in=1, alu_code=110, result=1, c_out=1
#
# time = 260, a=0, b=1, c_in=1, alu_code=110, result=1, c_out=0
#
# time = 280, a=1, b=1, c_in=1, alu_code=110, result=0, c_out=1
#
# time = 300, a=0, b=0, c_in=1, alu_code=110, result=0, c_out=1
#
```

32 Bit ALU (alu_32_bit.v) Test

alu_32_bit_testbench.v for this test.

Op codes for this ALU:

- 000 – AND
- 001 – OR
- 010 – ADD
- 011 – XOR
- 110 – SUB

Design of ALU:

This design is very similar to the one which is shown in lecture. Combines 32 x 1_bit alu and gives output. Overflow and Zero bits are active.

AND Test (000):

Variable = decimal version / binary version (show both versions)
Decimal values are not important for this operation.

```
# a = -1282297788 / 10110011100100011011000001000100 , b =   753120834 / 00101100111000111011011001000010,
#                OPCODE = 000, result =   545370176 / 00100000100000011011000001000000, OF=0, Zero=0
#
```

```
# a = -1431655766 / 10101010101010101010101010101010 , b =  1431655765 / 01010101010101010101010101010101,
#                OPCODE = 000, result =           0 / 00000000000000000000000000000000, OF=0, Zero=1
#
```

OR Test (001):
Variable = decimal version / binary version (show both versions)
Decimal values are not important for this operation.

```
a = -1282297788 / 10110011100100011011000001000100 , b =   753120834 / 00101100111000111011011001000010,
               OPCODE = 001, result = -1074547130 / 10111111111100111011011001000110, OF=0, Zero=0
```

```
a = -1431655766 / 10101010101010101010101010101010 , b =  1431655765 / 01010101010101010101010101010101,
               OPCODE = 001, result =          -1 / 11111111111111111111111111111111, OF=0, Zero=0
```

XOR Test (011):
Variable = decimal version / binary version (show both versions)
Decimal values are not important for this operation.

```
# a =    -1048576 / 11111111111100000000000000000000 , b =    16777215 / 00000000111111111111111111111111,
#                OPCODE = 011, result =   -15728641 / 11111111000011111111111111111111, OF=0, Zero=0
#
```

```
# a = -1431655766 / 10101010101010101010101010101010 , b =  1431655765 / 01010101010101010101010101010101,
#                OPCODE = 011, result =          -1 / 11111111111111111111111111111111, OF=0, Zero=0
#
```

ADD Test (010):
Variable = decimal version / binary version (show both versions)
Second test is for Overflow.

```
# a =         108 / 00000000000000000000000001101100 , b =         322 / 00000000000000000000000101000010,
#                 OPCODE = 010, result =         430 / 00000000000000000000000110101110, OF=0, Zero=0
#
# a =  2147483647 / 01111111111111111111111111111111 , b =  2147483647 / 01111111111111111111111111111111,
#                 OPCODE = 010, result =          -2 / 11111111111111111111111111111110, OF=1, Zero=0
#
```

SUB Test (110):
Variable = decimal version / binary version (show both versions)
First test is for Zero bit.

```
# a =          56 / 00000000000000000000000000111000 , b =          56 / 00000000000000000000000000111000,
#                 OPCODE = 110, result =           0 / 00000000000000000000000000000000, OF=0, Zero=1
#
# a =        1000 / 00000000000000000000001111101000 , b =         654 / 00000000000000000000001010001110,
#                 OPCODE = 110, result =         346 / 00000000000000000000000101011010, OF=0, Zero=0
#
```

Control Module

control_unit_testbench.v for this test.

This module takes **OP Code** and **Function Code** (for jr instruction) as input and gives these signals as output:

- **MemRead**: A signal goes to Data Memory to allow reading data.
- **MemToReg**: A signal goes to a mux which choose between Data Memory output and ALU output.
- **ALUSrc**: A signal goes to a mux which choose ALU source between Read Data 2(register module output) and sign extended Instruction [15-0].
- **RegDst:** A signal goes to a mux which chose Write Register 1(register module input) source between RT RD.
- **RegWrite**: A signal goes to register module to allow writing register for Write register 1 input.
- **RegWrite2**: A signal goes to register module to allow writing register for Write register 1 input.
- **MemWrite**: A signal goes to Data Memory Module to allow writing data.
- **Jump**: A signal goes to a mux to choose next program counter.
- **Jumpl**: A signal goes to a mux to choose next program counter.
- **Branch**: A signal to be used in operations of branch equal instruction.
- **Branchnot**: A signal to be used in operations of branch not equal instruction.
- **Jumpreg**: A signal to be used in operations of jr instruction.

- **ALUOp**: This is a 2-bit output which goes to **ALU Control Module**. Details are on ALU Control Module Part. **This signal does not go to ALU.

Each supported instruction and its control unit output listed below:

- **Lw (100011)**: memread, memtoreg, ALUSrc, RegWrite, AluOP(01)
- **Sw (101011)**: memwrite, ALUSrc, AluOP(01)
- **J (000010):** jump
- **Jal (000011)**: jumpnl, regwrite
- **Beq (000100):** branch, AluOP (10)
- **Bne (000101):** branchn, AluOP (10)
- **Ori (001101)**: ALUSrc, Regwrite, AluOP (11)
- **Lui (001111)**: memread, memtoreg, Regwrite
- **Addn/Subn/Xorn/Andn/Orn(000000/fun code)**: regwrite, regwrite2, regDst, AluOP(00)
- **Jr (000000/001000)**: Jumpreg

LW Test (100011):

FuncCode is not important for this operation.

```
# time =  0, instCode=100011, funcCode=101010, memRead=1, memToReg=1,
# ALUSrc=1, RegDst=0, RegWrite=1, RegWrite2=0, memWrite=0,
# jump=0, jumpl=0, branch=0, branchnot=0, jumpreg=0, ALUOp=01
#
#
```

SW Test (101011):
FuncCode is not important for this operation.

```
# time = 20, instCode=101011, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=1, RegDst=0, RegWrite=0, RegWrite2=0, memWrite=1,
# jump=0, jumpl=0, branch=0, branchnot=0, jumpreg=0, ALUOp=01
#
```

J Test (000010):
FuncCode is not important for this operation.

```
# time = 40, instCode=000010, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=0, RegDst=0, RegWrite=0, RegWrite2=0, memWrite=0,
# jump=1, jumpl=0, branch=0, branchnot=0, jumpreg=0, ALUOp=00
#
```

JAL Test (000011):
FuncCode is not important for this operation.

```
# time = 60, instCode=000011, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=0, RegDst=0, RegWrite=1, RegWrite2=0, memWrite=0,
# jump=0, jumpl=1, branch=0, branchnot=0, jumpreg=0, ALUOp=00
#
```

BEQ Test (000100):
FuncCode is not important for this operation.

```
# time = 80, instCode=000100, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=0, RegDst=0, RegWrite=0, RegWrite2=0, memWrite=0,
# jump=0, jumpl=0, branch=1, branchnot=0, jumpreg=0, ALUOp=10
#
```

BNE Test (000101):
FuncCode is not important for this operation.

```
# time = 100, instCode=000101, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=0, RegDst=0, RegWrite=0, RegWrite2=0, memWrite=0,
# jump=0, jumpl=0, branch=0, branchnot=1, jumpreg=0, ALUOp=10
#
```

ORI Test (001101):
FuncCode is not important for this operation.

```
# time = 120, instCode=001101, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=1, RegDst=0, RegWrite=1, RegWrite2=0, memWrite=0,
# jump=0, jumpl=0, branch=0, branchnot=0, jumpreg=0, ALUOp=11
#
```

LUI Test (001111):
FuncCode is not important for this operation.

```
# time = 140, instCode=001111, funcCode=101010, memRead=1, memToReg=1,
# ALUSrc=0, RegDst=0, RegWrite=1, RegWrite2=0, memWrite=0,
# jump=0, jumpl=0, branch=0, branchnot=0, jumpreg=0, ALUOp=00
#
#
```

JR Test (000000/001000):

```
# time = 160, instCode=000000, funcCode=001000, memRead=0, memToReg=0,
# ALUSrc=0, RegDst=0, RegWrite=0, RegWrite2=0, memWrite=0,
# jump=0, jumpl=0, branch=0, branchnot=0, jumpreg=1, ALUOp=00
#
```

Addn/Subn/Xorn/Andn/Orn Test (000000/fun code):
FuncCode is not important for this operation. It will be decided on ALU Control Module.

```
# time = 180, instCode=000000, funcCode=101010, memRead=0, memToReg=0,
# ALUSrc=0, RegDst=1, RegWrite=1, RegWrite2=1, memWrite=0,
# jump=0, jumpl=0, branch=0, branchnot=0, jumpreg=0, ALUOp=00
#
```

ALU Control Module

ALU_control_unit_testbench.v for this test.

This module takes **ALUOp(itype)** from Control Unit and **Function Code(funcCode)** (for r-type instructions) as input and gives ALU_OP_CODE as output.

ALU_OP_CODE:

- 000 – AND
- 001 – OR
- 010 – ADD
- 011 – XOR
- 110 – SUB

FuncCode is not important if the itype is not 00 (Like lw, sw... instructions).

AND Test (000):

```
# time = 120, itype=00, funcCode=100100, ALU_OP_Code=000
#
```

OR Test (001):

```
#
# time = 40, itype=11, funcCode=100110, ALU_OP_Code=001
#
#
# time = 140, itype=00, funcCode=100101, ALU_OP_Code=001
#
```

ADD Test (010):

```
# time =  0, itype=01, funcCode=100110, ALU_OP_Code=010
#
#
# time = 60, itype=00, funcCode=100000, ALU_OP_Code=010
#
```

XOR Test (011):

```
#
# time = 100, itype=00, funcCode=100110, ALU_OP_Code=011
#
```

SUB Test (110):

```
# time = 20, itype=10, funcCode=100110, ALU_OP_Code=110
#
#
# time = 80, itype=00, funcCode=100010, ALU_OP_Code=110
#
```

Registers Module

registers_testbench.v for this test.

This module takes these as input:

- Readreg_1 is address of the first register to read.
- Readreg_2 is address of the second register to read.
- Writereg_1 is address of the first register to write.
- Writereg_2 is address of the second register to write.
- Writedata_1 is a 32-bit data to write.
- Writedata_2 is a 32-bit data to write.

- Regwrite_1 is a signal which comes from control unit to allow writereg_1.
- Regwrite_2 is a signal which comes from control unit to allow writereg_2.
- CLK is clock.

Register_Data.mem file before the test. - First 8 register -

```
1  00000000011010101100000000011111   Read 1 - Test 1
2  00011111111111111011000000001111   Read 2 - Test 1
3  01111110000000000011111100000111   Read 1 - Test 2
4  11111111111111111111000000000011   Read 2 - Test 2
5  00000000000000000000000000000000
6  00000000000000000000000000000000
7  00000000000000000000000000000000
8  00000000000000000000000000000000
```

Test:

*Tests marked with red are dummy tests to set CLK as 0 to get positive edge for next test.

**Test 3 is stand for proving that data won't be written when Regwrite_1 and Regwrite_2 are 0.

```
# time = 40, Readreg_1=00000, readData_1=00000000011010101100000000011111,
# Readreg_2=00001, readData_2=00011111111111111011000000001111, Writereg_1=00100, Writereg_2=00101,
# Writedata_1=00000000000000000000000000001000, Writedata_2=00000000000000000000000000000100,
# Regwrite_1=1, Regwrite_2=1, CLK=1 TEST 1
#
#
# time = 80, Readreg_1=00000, readData_1=00000000011010101100000000011111,
# Readreg_2=00001, readData_2=00011111111111111011000000001111, Writereg_1=00100, Writereg_2=00101,
# Writedata_1=00000000000000000000000000001000, Writedata_2=00000000000000000000000000000100,
# Regwrite_1=1, Regwrite_2=1, CLK=0
#
#
# time = 100, Readreg_1=00010, readData_1=01111110000000000011111100000111,
# Readreg_2=00011, readData_2=11111111111111111111000000000011, Writereg_1=00110, Writereg_2=00111,
# Writedata_1=00000000000000000000000000010000, Writedata_2=00000000000000000000000000110000,
# Regwrite_1=1, Regwrite_2=1, CLK=1 TEST 2
#
#
# time = 120, Readreg_1=00010, readData_1=01111110000000000011111100000111,
# Readreg_2=00011, readData_2=11111111111111111111000000000011, Writereg_1=00110, Writereg_2=00111,
# Writedata_1=00000000000000000000000000010000, Writedata_2=00000000000000000000000000110000,
# Regwrite_1=1, Regwrite_2=1, CLK=0
#
#
# time = 140, Readreg_1=00000, readData_1=00000000011010101100000000011111,
# Readreg_2=00000, readData_2=00000000011010101100000000011111, Writereg_1=00110, Writereg_2=00111,
# Writedata_1=00000000000000000000001111100111, Writedata_2=00000000000000000000001111100111,
# Regwrite_1=0, Regwrite_2=0, CLK=1 TEST 3
#
```

Register_Data_Out.mem after the test:

```
// memory data file (do not edit the following line - required for
// instance=/registers_testbench/test/regs
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=
00000000011010101100000000011111
00011111111111111011000000001111
01111110000000000011111100000111
11111111111111111111000000000011
00000000000000000000000000001000
00000000000000000000000000000100
00000000000000000000000000010000
00000000000000000000000000110000
00000000000000000000000000000000
00000000000000000000000000000000
```

TEST 1 - Write data 1
TEST 1 - Write data 2
TEST 2 - Write data 1
TEST 2 - Write data 2

Data Memory Module

data_memory_testbench.v for this test.

(*) Size of data memory set to 128 byte(32 word). Because otherwise compile time was taking very long time. (A friend asked to T.A via e-mail.) But still module behave as 18bit addressed.

This module takes these as input:

- Address is address of the word at memory (18 bit).
- Writedata is 32 bit word for writng to memory.
- Memwrite is a signal to allow memory writing.
- Memread is a signal to allow memory reading.
- CLK is clock.

Data_Memory.mem file before the test. - First 8 word / 32 byte-

```
1   00000000
2   00000001
3   00110011
4   10010101
5   00000000
6   00000001
7   00101000
8   10101010
9   00000000
10  00000000
11  00000000
12  00000000
13  00000000
14  00000000
15  00000000
16  00000000
17  00000000
18  00000000
19  00000000
20  00000000
21  00000000
22  00000000
23  00000000
24  00000000
25  00000000
26  00000000
27  00000000
28  00000000
29  00000000
30  00000000
31  00000000
32  00000000
33  00000000
```

Will be read at

Will be read at TEST 2

Will be written at TEST 3

Will be written at TEST 4

Test:

*Test1 and Test2 for showing read data is working.

**Test 3 and Test 4 for showing write data is working. Read data outputs didn't change because readdata input was 0 at Test 3 and Test 4.

```
#
# time = 40, address= 0 / 000000000000000000, readData=00000000000000001001100111001010 1, writedata= 00000000000000000000011111111111, memread= 1, memwrite= 0
#
# TEST 1
#
# time = 80, address= 4 / 000000000000000100, readData=00000000000000001001010001010101 0, writedata= 00000000000000000000011111111111, memread= 1, memwrite= 0
# TEST 2
#
# time = 120, address= 8 / 000000000000001000, readData=00000000000000001001010001010101 0, writedata= 00000000000000000000011111111111, memread= 0, memwrite= 1
#TEST 3
#
# time = 160, address= 12 / 000000000000001100, readData=00000000000000001001010001010101 0, writedata= 00000000000000000000001010111010, memread= 0, memwrite= 1
#TEST 4
#
```

Data_Memory_Out.mem after the test:

```
// memory data file (do no
// instance=/data_memory_t
// format=bin addressradix
00000000
00000001
00110011
10010101
00000000
00000001
00101000
10101010
00000000
00000000
00000111
11111111
00000000
00000000
00000010
10111010
00000000
00000000
00000000
```

Written at Test 3

Written at Test 4

Instruction Memory Module

instruction_memory_testbench.v for this test.

(*) Size of data memory set to 16 word. Because otherwise compile time was taking very long time. (A friend asked to T.A via e-mail.) This module takes these as input:

- progC is program counter.
- CLK is clock.

instructions.mem file before the test.

```
00000000011010101100000000011111
00011111111111111011000000001111
01111110000000000011111100000111
11111111111111111111000000000011
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Will be read

Will be read at test 2

```
# time = 40, PC=    0, instruction=00000000011010101100000000011111
#
#
# time = 80, PC=    1, instruction=00011111111111111011000000001111
#
#
```