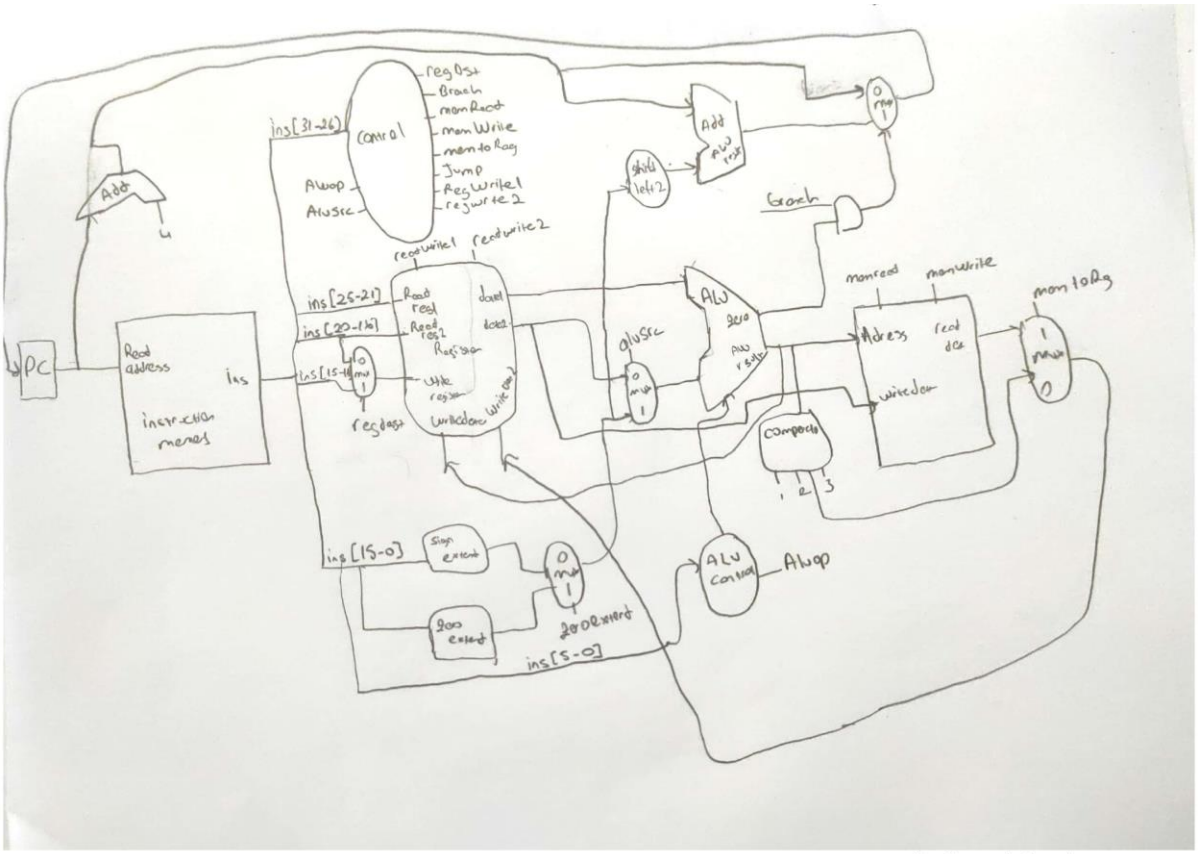
CSE 331/503

COMPUTER ORGANIZATION FALL 2020

HOMEWORK 4 REPORT

OZAN GEÇKİN 1801042103

Single Cycle Datapath Design:



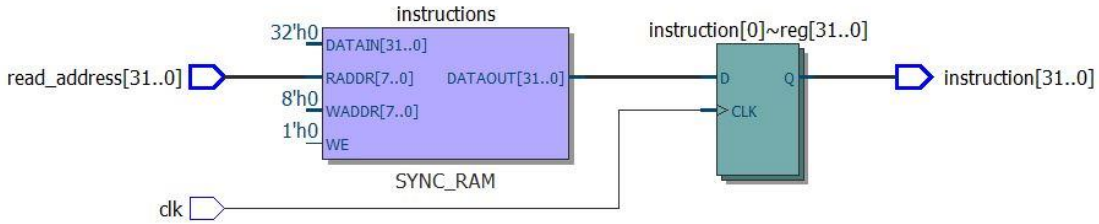
CamScanner ile tarandı

I used following schema. I designed all the modules I use in my datapath.

Modules

Instruction Memory:

I use it to hold my instructions. It receives 32 bit address and clock as input. It gives 32 bit instruction as output.

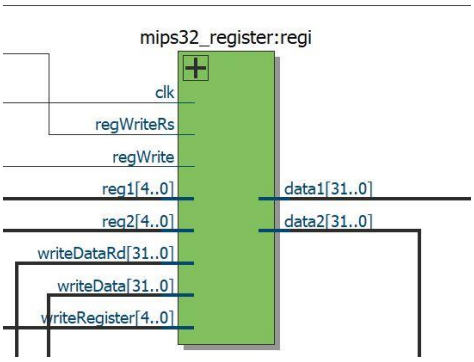


Instruction Memory Test:

```
sim:/instruction_testbench/PC
VSIM 6> step -current
# Inst [         1]-- Opcode=000000 [00], Rs=01000 [08], Rt=00011 [03], Imm=0000100010100000 [08a0]
# Inst [         2]-- Opcode=001010 [0a], Rs=11111 [1f], Rt=00101 [05], Imm=0000010011111111 [04ff]
# Inst [         3]-- Opcode=000100 [04], Rs=00001 [01], Rt=00101 [05], Imm=0000000000000011 [0003]
# Inst [         4]-- Opcode=000000 [00], Rs=01000 [08], Rt=01001 [09], Imm=1111100010100001 [f8a1]
# Inst [         5]-- Opcode=000000 [00], Rs=01000 [08], Rt=01001 [09], Imm=1111100010100101 [f8a5]
# Inst [         6]-- Opcode=000000 [00], Rs=01000 [08], Rt=01001 [09], Imm=1111100010000000 [f880]
# Inst [         7]-- Opcode=100011 [23], Rs=01001 [09], Rt=01000 [08], Imm=0000000000001111 [000f]
# Inst [         8]-- Opcode=000000 [00], Rs=01010 [0a], Rt=01110 [0e], Imm=0111100010000010 [7882]
# Inst [         9]-- Opcode=000000 [00], Rs=11111 [1f], Rt=10000 [10], Imm=1000100000100010 [8822]
# Inst [        10]-- Opcode=000000 [00], Rs=11110 [1e], Rt=10001 [11], Imm=1001000000100011 [9023]
# Inst [        11]-- Opcode=000000 [00], Rs=10010 [12], Rt=11101 [1d], Imm=1010000000101010 [a02a]
# Inst [        12]-- Opcode=000000 [00], Rs=10110 [16], Rt=10101 [15], Imm=1010100000101011 [a82b]
# Inst [        13]-- Opcode=001000 [08], Rs=10101 [15], Rt=10111 [17], Imm=0000000000010010 [0012]
# Inst [        14]-- Opcode=001100 [0c], Rs=10111 [17], Rt=11001 [19], Imm=0000000000011011 [001b]
# Inst [        15]-- Opcode=001001 [09], Rs=11001 [19], Rt=11010 [1a], Imm=0000000000100011 [0023]
# Inst [        16]-- Opcode=001101 [0d], Rs=11010 [1a], Rt=11011 [1b], Imm=0000000000011001 [0019]
# Inst [        17]-- Opcode=001010 [0a], Rs=11101 [1d], Rt=11100 [1c], Imm=0000000000011101 [001d]
# Inst [        18]-- Opcode=100011 [23], Rs=11110 [1e], Rt=11111 [1f], Imm=0000000001000000 [0040]
# Inst [        19]-- Opcode=110000 [30], Rs=11111 [1f], Rt=11110 [1e], Imm=0000000010000000 [0080]
# Inst [        20]-- Opcode=001111 [0f], Rs=01011 [0b], Rt=11101 [1d], Imm=0000000001111111 [007f]
# Inst [        21]-- Opcode=000100 [04], Rs=00011 [03], Rt=00111 [07], Imm=0000000000010110 [0016]
# ** Note: $finish    : C:/altera/13.1/workspace/hw/instruction_testbench.v(13)
#    Time: 420 ps  Iteration: 0  Instance: /instruction_testbench
```

Register Modul:

You redesigned it according to the New R-type instructions. There were 2 data entries. You can see all inputs and outputs in RTL view.

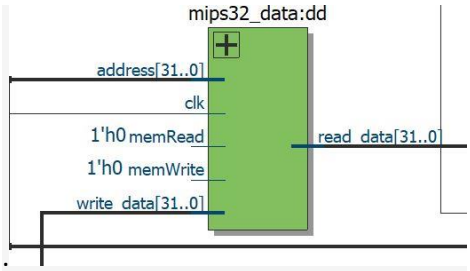


Register Modül Test:

Register Before: Test Code: Register After:

Data Memory :

I use it to write and read data. You can see all inputs and outputs in RTL view.



Data Memory Test:

Data Before:

```
// format=bin addressradix=h dataradix=b version=1
00000000000000000000000000000000
00000000000000000000000000010100
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
```

Data After:

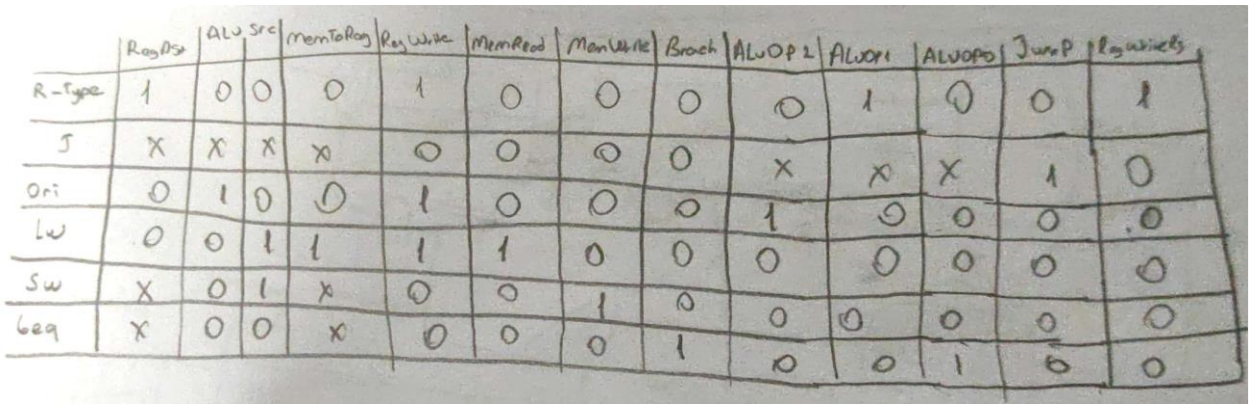
```
// format=bin addressradix=h dataradix=b version=1
11111111111111111111111111111111
00000000000000000000000000010100
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
```

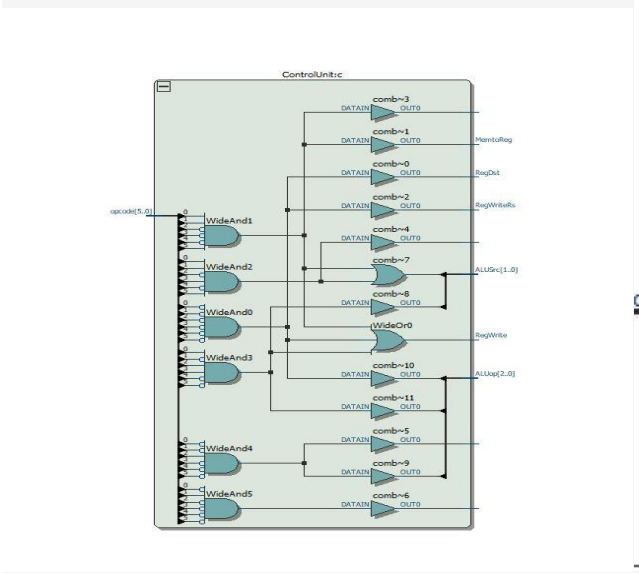
Test Code:

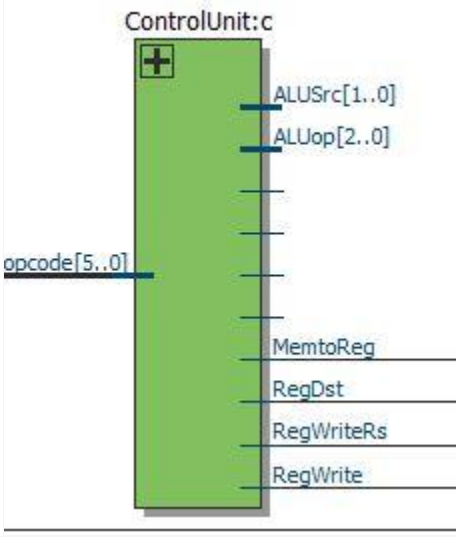
```
sim:/mips32_data_testbench/memRead \
sim:/mips32_data_testbench/clk
VSIM 5> step -current
# Memorye store edilen data: 11111111111111111111111111111111
# signal_mem_write=1, signal_mem_read=1, clk=1, write_data=11111111111111111111111111111111 ,address=00000000000000000000000000000000, read_data=11111111111111111111111111111111

VSIM 6>
```

Control Unit:

I have generated my signals using the Turth table. The control unit takes opcode as input. And it generates the signals required for the instruction to occur.





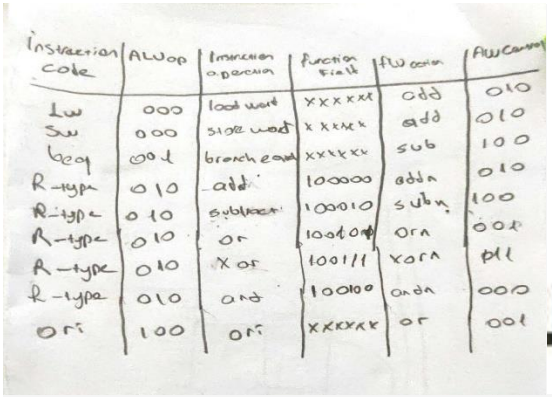


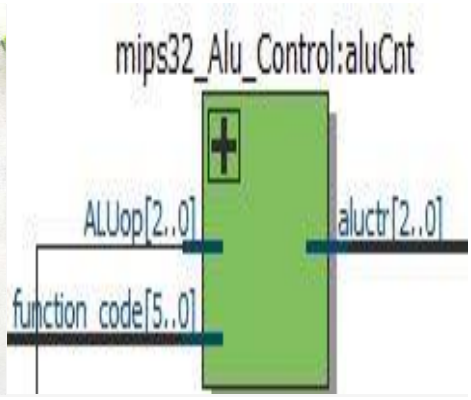
Control Unit Test:

```
sim:/ControlUnit_testbench/RegWrite
VSIM 5> step -current
# opcode=100011, regDst=0,ALUSrc=01,regWire=1,memRead=1,MemToReg=0,memWrite=1,brach=0,aluop=000,jump=0,RegWriteRs=0
# opcode=101011, regDst=0,ALUSrc=01,regWire=0,memRead=0,MemToReg=1,memWrite=0,brach=0,aluop=000,jump=0,RegWriteRs=0
# opcode=000000, regDst=1,ALUSrc=00,regWire=1,memRead=0,MemToReg=0,memWrite=0,brach=0,aluop=010,jump=0,RegWriteRs=1

VSIM 6>
```

ALU Control Unit:

It generates the signal to choose which action the ALU does. It receives the function code and the aluop. It receives the ALU from the control unit.



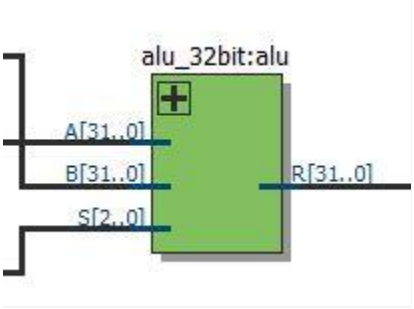


ALU Control Test:

```
VSIM 5> step -current
# aluOp=010,func=100000,aluctr=010
# aluOp=000,func=100000,aluctr=010
# aluOp=001,func=001111,aluctr=100

VSIM 6>
```

ALU Module:

ALU performs operations according to the signal coming from the control and draws results.

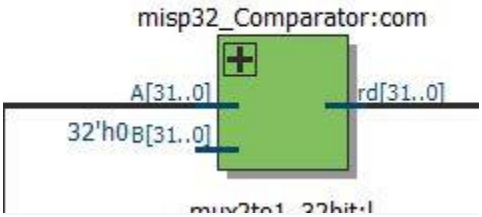


ALU Module Test:

```
sim:/alu32_bit_testbench/sel \
sim:/alu32_bit_testbench/R
VSIM 5> step -current
# A=00000000000000000000000000000101 ,B=00000000000000000000000000000001 ,sel=000, R=00000000000000000000000000000001
# A=00000000000000000000000000000101 ,B=00000000000000000000000000000001 ,sel=001, R=00000000000000000000000000000101
# A=00000000000000000000000000000101 ,B=00000000000000000000000000000001 ,sel=010, R=00000000000000000000000000000110
# A=00000000000000000000000000000101 ,B=00000000000000000000000000000001 ,sel=011, R=00000000000000000000000000000100
# A=00000000000000000000000000000101 ,B=00000000000000000000000000000001 ,sel=100, R=00000000000000000000000000000100

VSIM 6>
```

Comparator Module:

Comparator produces the value by comparing the content of Rs and Rd with 0 according to the result from alu.

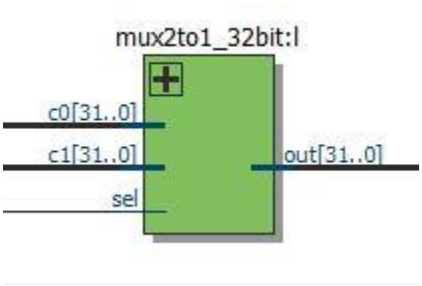


Comparator Module Test:

```
sim:/mips32_Comparator_testbench/rd
VSIM 5> step -current
# A=00000000000000000000000000001010,rd=00000000000000000000000000000010
# A=11111111111111111111111111111101,rd=00000000000000000000000000000011
# A=00000000000000000000000000000000,rd=00000000000000000000000000000001

VSIM 6>
```

Mux 2 to 1 Module:

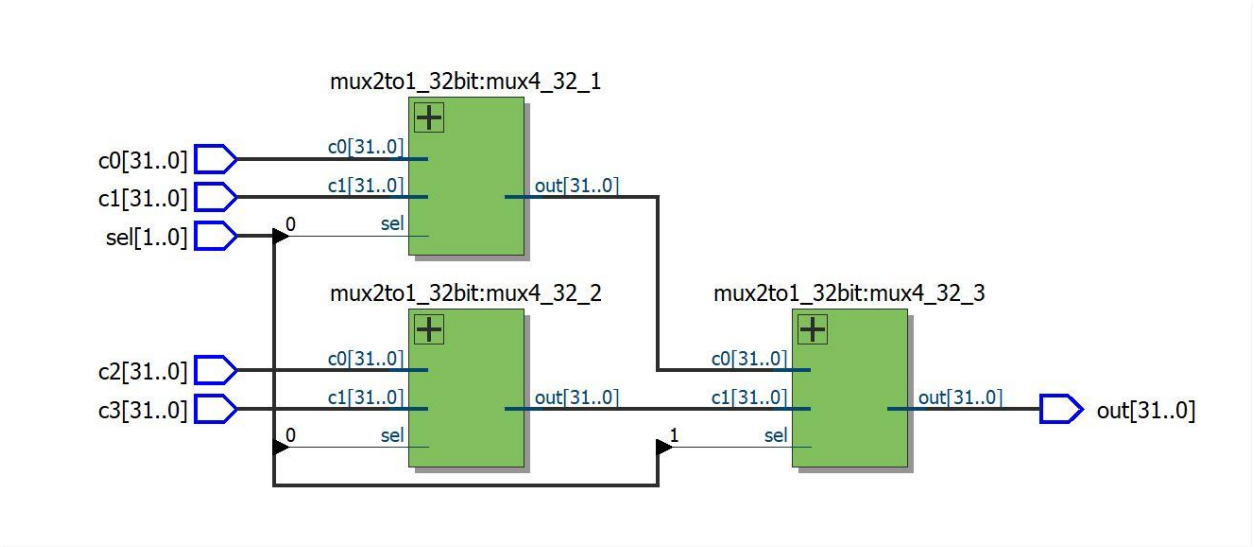
It selects one of the 2 inputs according to the incoming signal.



Mux 2 to 1 Module test:

Mux 4 to 1 Module:

It selects one of the 4 inputs according to the incoming signal. I used Mux 2 to 1 in its production.

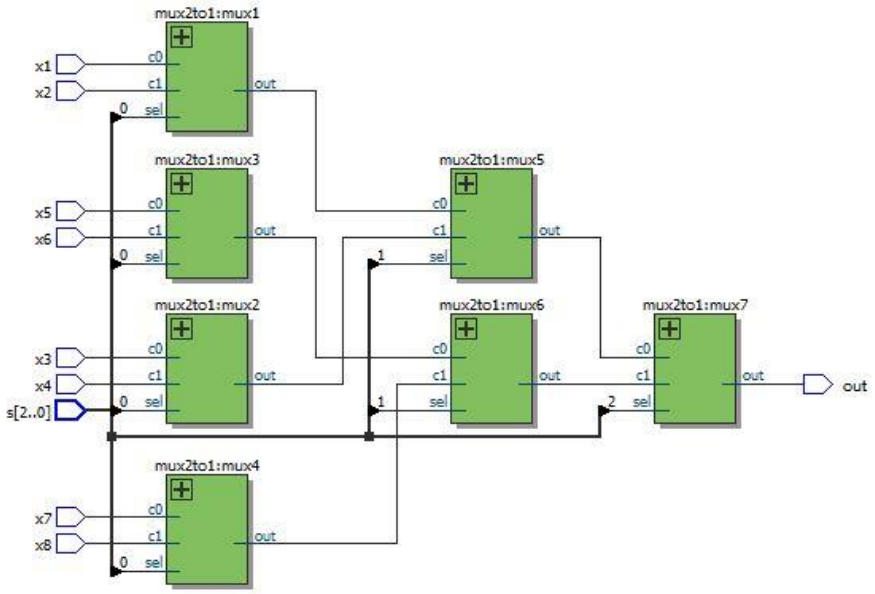


Mux 4 to 1 Module Test:

```
VSIM 5> step -current
#  c0=00000000000000000000000000000000 c1=00000000000000000000000000000001 c2=00000000000000000000000000000010 c3=00000000000000000000000000000011 sel=00 out=00000000000000000000000000000000
#  c0=00000000000000000000000000000000 c1=00000000000000000000000000000001 c2=00000000000000000000000000000010 c3=00000000000000000000000000000011 sel=01 out=00000000000000000000000000000001
#  c0=00000000000000000000000000000000 c1=00000000000000000000000000000001 c2=00000000000000000000000000000010 c3=00000000000000000000000000000011 sel=10 out=00000000000000000000000000000010
#  c0=00000000000000000000000000000000 c1=00000000000000000000000000000001 c2=00000000000000000000000000000010 c3=00000000000000000000000000000011 sel=11 out=00000000000000000000000000000011

VSIM 6>
```

Mux 8 to 1 Module :

It selects one of the 8 inputs according to the incoming signal. I used Mux 2 to 1 in its production.

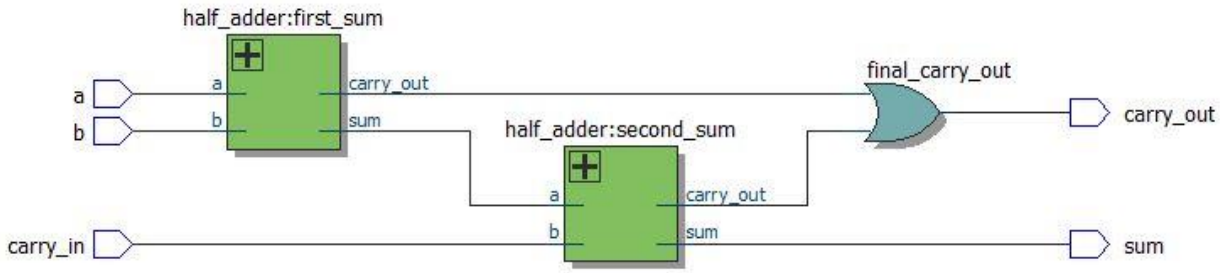


Mux 8 to 1 Module Test:

```
VSIM 5> step -current
#  c1=00000000000000000000000000000000 c2=00000000000000000000000000000001 c3=00000000000000000000000000000010 c4=00000000000000000000000000000011 c5=00000000000000000000000000000001 c6=00000000000000000000000000000001 c7=00000000000000000000000000000001 c8=00000000
00000000000000000000001 sel=000 out=00000000000000000000000000000000
```

Adder Module :

Adds 2 32-bit numbers.

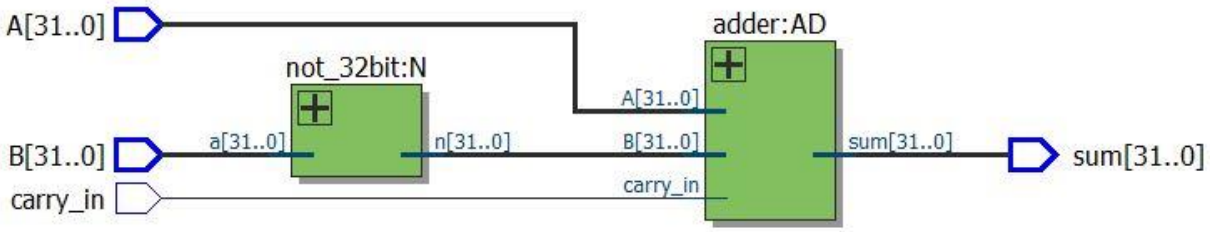


Adder Module Test:

```
VSIM 5> step -current
# A=00000000000000000000000000000100,B=00000000000000000000000000000110,Sum=00000000000000000000000000001010
# A=00000000000000000000000000000101,B=00000000000000000000000000001100,Sum=00000000000000000000000000010001
```

Subtract Module:

Subs 2 32-bit numbers.



Subtract Test Module

```
VSIM 5> step -current
# A=00000000000000000000000000000100,B=00000000000000000000000000000110,Sub=11111111111111111111111111111110
# A=00000000000000000000000000000101,B=00000000000000000000000000001100,Sub=11111111111111111111111111111001

VSIM 6>
```

And Module:

Ands 2 32-bit numbers.

And Module Test:

```
VSIM 5> step -current
# A=00000000000000000000000000000100,B=00000000000000000000000000000110, And=00000000000000000000000000000100
# A=00000000000000000000000000000101,B=00000000000000000000000000001100, And=00000000000000000000000000000100

VSIM 6>
```

Or Module:

Or 2 32-bit numbers.

Or Module Test:

```
VSIM 5> step -current
# A=00000000001000000000000000000100,B=00000000000000000000000000000110,Or=00000000001000000000000000000110
# A=01000000000000000000000000000101,B=00000100001000000000000000001100,Or=01000100001000000000000000001101
```

Xor Module:

Xor 2 32-bit numbers.

Xor Module Test:

```
VSIM 5> step -current
# A=00000000001000000000000000000100,B=00000000000000000000000000000110,XOR=00000000001000000000000000000010
# A=01000000000000000000000000000101,B=00000100001000000000000000001100,XOR=01000100001000000000000000001001

VSIM 6>
```

Not Module:

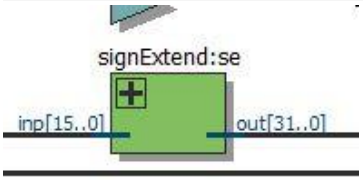
Notes 1 32-bit number.

Not Module Test:

```
sim:/not_32bit_testbench/A
VSIM 5> step -current
# A=00000000000000000000000000000100,not_A=11111111111111111111111111111011
# A=00000000000000000000000000000110,not_A=11111111111111111111111111111001
# A=00000000000000000000000000000101,not_A=11111111111111111111111111111010
# A=00000000000000000000000000001100,not_A=11111111111111111111111111110011

VSIM 6>
```

SignExtend Module:

According to Most Significant bit, it completes the number from 16 bits to 32 bits.



SignExtend Module Test:

```
sim:/signExtend_testbench/out
VSIM 5> step -current
# inp=0000001111111111,out=00000000000000000000001111111111
```

ZeroExtend Module:

It completes a 16-bit number with a leading 0 to 32 bits.

ZeroExtend Module Test:

```
sim:/zeroExtend_testbench/A
VSIM 5> step -current
# A=0101010101000000,ZeroExtend=00000000000000000101010101000000
```

Program Counter :

I designed it for the jump and branch instructions. He gets insturction, jump signal, branch signal, clock. And holding the counter returns the omitted instruction.

NOT: I ran all my modules separately, but I had problems integrating with the datapath. I have shown that all my modules are working by adding their tests to my report. Thank you very much for your understanding and teaching in the whole semester.