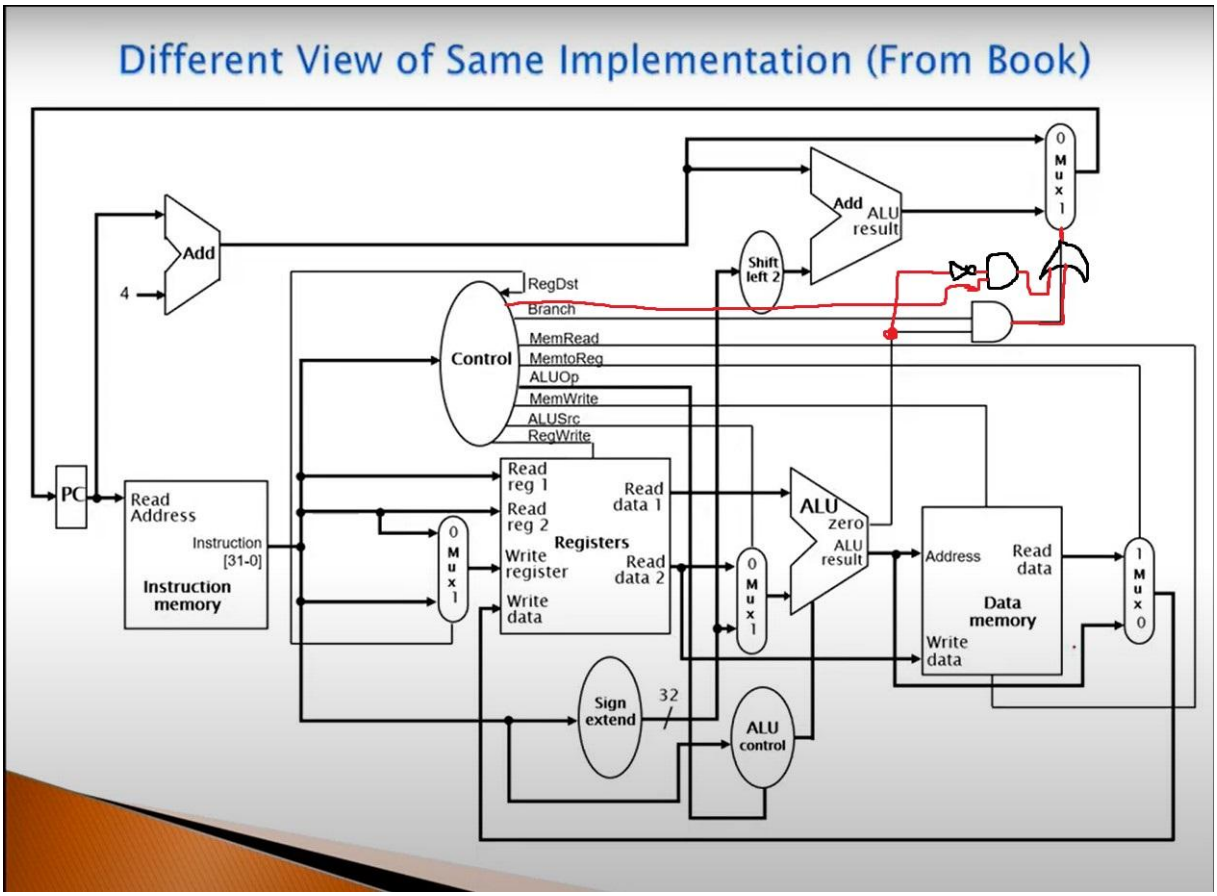
CSE 331 ASSIGNMENT 4 RAPORT

Merve Horuz

Here is datapath of minimips processor:



Missing parts are only bne and bnq operations.

_32bit_adder_.v module

```
Transcript
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_adder_testbench
#
# Top level modules:
#       _32bit_adder_testbench
ModelSim> vsim work._32bit_adder_testbench
# vsim work._32bit_adder_testbench
# Loading work._32bit_adder_testbench
# Loading work._32bit_adder_
# Loading work.full_adder_
# Loading work.half_adder_
VSIM 4> vsim work._32bit_adder_testbench
# vsim work._32bit_adder_testbench
# Loading work._32bit_adder_testbench
# Loading work._32bit_adder_
# Loading work.full_adder_
# Loading work.half_adder_
add wave -position insertpoint  \
sim:/_32bit_adder_testbench/a \
sim:/_32bit_adder_testbench/b \
sim:/_32bit_adder_testbench/carry_in \
sim:/_32bit_adder_testbench/carry_out \
sim:/_32bit_adder_testbench/sum
VSIM 5> step -current
# time= 0, a=10101010101111111111111111111111, b=10100011101001111111111111111111, carry_in=0, carry_out=1, sum=01001110011001111111111111111110
# time=20, a=00000000000000000000111111111111, b=11111111111111111111111110000000, carry_in=0, carry_out=1, sum=00000000000000000000111110111111

VSIM 6>
```

_32bit_sub_.v module

```
#
# Top level modules:
#       _32bit_sub_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_32bit_sub_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_sub_testbench
#
# Top level modules:
#       _32bit_sub_testbench
ModelSim> vsim work._32bit_sub_testbench
# vsim work._32bit_sub_testbench
# Loading work._32bit_sub_testbench
# Loading work._32bit_sub_
# Loading work.full_adder_
# Loading work.half_adder_
add wave -position insertpoint  \
sim:/_32bit_sub_testbench/a \
sim:/_32bit_sub_testbench/b \
sim:/_32bit_sub_testbench/carry_in \
sim:/_32bit_sub_testbench/carry_out \
sim:/_32bit_sub_testbench/sub
VSIM 5> step -current
# time= 0, a=10101010101111111111111111111111, b=10100011101001111111111111111111, carry_in=0, carry_out=1, sub=00000111000110000000000000000000
# time=20, a=00000000000000000000111111111111, b=11111111111111111111111110000000, carry_in=0, carry_out=0, sub=00000000000000000010000001111111
# time=40, a=10100000000000000000000000000001, b=00001100000000000000000000110000, carry_in=0, carry_out=1, sub=10010011111111111111111111010001

VSIM 6>
```

If first number less then second number, set result to 1.

_32bit_slt_.v module

```
#
#          Region: /_32bit_slt_testbench/io/s32/ao31
add wave -position insertpoint  \
sim:/_32bit_slt_testbench/a \
sim:/_32bit_slt_testbench/b \
sim:/_32bit_slt_testbench/result
VSIM 5> step -current
# time= 0, a=10101010101111111111111111111111, b=00100011101001111111111111111111, result=1
# time=20, a=00000000000000000000111111111111, b=11111111111111111111111110000000, result=0
# time=40, a=10100000000000000000000000000001, b=00001100000000000000000000110000, result=1
# time=60, a=01100000000000000000000000000001, b=00001100000000000000000000110000, result=0

VSIM 6>
```

_32bit_xor_ module

```
Transcript
# vlog -vlog01compat -work work +incdir+C:/altera/13.1/workspace/hw3 {C:/altera/13.1/workspace/hw3/_32bit_xor_.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_xor_
#
# Top level modules:
#       _32bit_xor_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_32bit_xor_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_xor_testbench
#
# Top level modules:
#       _32bit_xor_testbench
ModelSim> vsim work._32bit_xor_testbench
# vsim work._32bit_xor_testbench
# Loading work._32bit_xor_testbench
# Loading work._32bit_xor_
# Loading work.xor_
add wave -position insertpoint  \
sim:/_32bit_xor_testbench/a \
sim:/_32bit_xor_testbench/b \
sim:/_32bit_xor_testbench/result
VSIM 5> step -current
# time =  0, a=00100001001010000010011111000110, b=00101111000000000000011111000110, result=00001110001010000010000000000000
# time = 20, a=00000000000000000000010111000110, b=00101111001010000000011111000110, result=00101111001010000000001000000000
# time = 40, a=00000000000000000000000010000110, b=00000010000000010000011111000110, result=00000010000000010000011101000000

VSIM 6>
```

_32bit_nor_.v module

```
Transcript
# vlog -vlog01compat -work work +incdir+C:/altera/13.1/workspace/hw3 {C:/altera/13.1/workspace/hw3/_32bit_nor_.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_nor_
#
# Top level modules:
#       _32bit_nor_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_32bit_nor_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_nor_testbench
#
# Top level modules:
#       _32bit_nor_testbench
ModelSim> vsim work._32bit_nor_testbench
# vsim work._32bit_nor_testbench
# Loading work._32bit_nor_testbench
# Loading work._32bit_nor_
# Loading work.nor_
add wave -position insertpoint  \
sim:/_32bit_nor_testbench/a \
sim:/_32bit_nor_testbench/b \
sim:/_32bit_nor_testbench/result
VSIM 5> step -current
# time =  0, a=00100001001010000010011111000110, b=00101111000000000000011111000110, result=11010000110101111101100000111001
# time = 20, a=00000000000000000000000010111000110, b=00101111001010000000011111000110, result=11010000110101111111100000111001
# time = 40, a=00000000000000000000000010000110, b=00000010000000010000011111000110, result=11111101111111101111100000111001

VSIM 6>
```

I started to design 32 bits multiplexer from 2x1 mux.

_2_1_mux_.v module

```
Transcript
#          Updated modelsim.ini.
#
# vlog -vlog01compat -work work +incdir+C:/altera/13.1/workspace/hw3 {C:/altera/13.1/workspace/hw3/_2_1_mux_.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _2_1_mux_
#
# Top level modules:
#       _2_1_mux_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_2_1_mux_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _2_1_mux_testbench
#
# Top level modules:
#       _2_1_mux_testbench
ModelSim> vsim work._2_1_mux_testbench
# vsim work._2_1_mux_testbench
# Loading work._2_1_mux_testbench
# Loading work._2_1_mux_
add wave -position insertpoint  \
sim:/_2_1_mux_testbench/a \
sim:/_2_1_mux_testbench/result \
sim:/_2_1_mux_testbench/s
VSIM 5> step -current
# time =  0, a[0]=1, a[1]=0, s=0, result=1
# time = 20, a[0]=1, a[1]=0, s=1, result=0

VSIM 6>
```

_4_1_mux_.v module

```
Transcript
#       _2_1_mux_
# vlog -vlog01compat -work work +incdir+C:/altera/13.1/workspace/hw3 {C:/altera/13.1/workspace/hw3/_4_1_mux_.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _4_1_mux_
#
# Top level modules:
#       _4_1_mux_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_4_1_mux_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _4_1_mux_testbench
#
# Top level modules:
#       _4_1_mux_testbench
ModelSim> vsim work._4_1_mux_testbench
# vsim work._4_1_mux_testbench
# Loading work._4_1_mux_testbench
# Loading work._4_1_mux_
# Loading work._2_1_mux_
add wave -position insertpoint  \
sim:/_4_1_mux_testbench/a \
sim:/_4_1_mux_testbench/result \
sim:/_4_1_mux_testbench/s
VSIM 5> step -current
# time =  0, a=0001, s=01, result=0
# time = 20, a=1111, s=10, result=1

VSIM 6>
```

_8_1_mux_.v module

```
# -- Compiling module _8_1_mux_
#
# Top level modules:
#       _8_1_mux_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_8_1_mux_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _8_1_mux_testbench
# ** Warning: C:/altera/13.1/workspace/hw3/_8_1_mux_testbench.v(14): (vlog-2600) [RDGN] - Redundant digits in numeric literal.
#
#
# Top level modules:
#       _8_1_mux_testbench
ModelSim> vsim work._8_1_mux_testbench
# vsim work._8_1_mux_testbench
# Loading work._8_1_mux_testbench
# Loading work._8_1_mux_
# Loading work._4_1_mux_
# Loading work._2_1_mux_
add wave -position insertpoint  \
sim:/_8_1_mux_testbench/a \
sim:/_8_1_mux_testbench/result \
sim:/_8_1_mux_testbench/s
VSIM 5> step -current
# time =  0, a=00011001, s=011, result=1
# time = 20, a=11110001, s=000, result=1

VSIM 6>
```

_32_1_mux_.v module

```
Transcript
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32_1_mux_
#
# Top level modules:
#       _32_1_mux_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_32_1_mux_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32_1_mux_testbench
#
# Top level modules:
#       _32_1_mux_testbench
ModelSim> vsim work._32_1_mux_testbench
# vsim work._32_1_mux_testbench
# Loading work._32_1_mux_testbench
# Loading work._32_1_mux_
# Loading work._8_1_mux_
# Loading work._4_1_mux_
# Loading work._2_1_mux_
add wave -position insertpoint  \
sim:/_32_1_mux_testbench/a \
sim:/_32_1_mux_testbench/result \
sim:/_32_1_mux_testbench/s
VSIM 5> step -current
# time =  0, a=00000000011110101001001000011001, s=00111, result=0
# time = 20, a=00000000011110101001001000011001, s=00101, result=0

VSIM 6>
```

32 bit and, 32 bit or modules were designed by using 1 bit and/or.

_32bit_and_.v module

```
Transcript
#
# vlog -vlog01compat -work work +incdir+C:/altera/13.1/workspace/hw3 {C:/altera/13.1/workspace/hw3/_32bit_and_.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_and_
#
# Top level modules:
#       _32bit_and_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_32bit_and_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_and_testbench
#
# Top level modules:
#       _32bit_and_testbench
ModelSim> vsim work._32bit_and_testbench
# vsim work._32bit_and_testbench
# Loading work._32bit_and_testbench
# Loading work._32bit_and_
add wave -position insertpoint  \
sim:/_32bit_and_testbench/a \
sim:/_32bit_and_testbench/b \
sim:/_32bit_and_testbench/result
VSIM 5> step -current
# time =  0, a=00100001001010000010011111000110, b=00101111000000000000011111000110, result=00100001000000000000011111000110
# time = 20, a=00000000000000000000010111000110, b=00101111001010000000011111000110, result=00000000000000000000010111000110
# time = 40, a=00000000000000000000000010000110, b=00000010000000010000011111000110, result=00000000000000000000000010000110

VSIM 6>
```

_32bit_or_.v module

```
Transcript
#
# vlog -vlog01compat -work work +incdir+C:/altera/13.1/workspace/hw3 {C:/altera/13.1/workspace/hw3/_32bit_or_.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_or_
#
# Top level modules:
#       _32bit_or_
#
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_32bit_or_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _32bit_or_testbench
#
# Top level modules:
#       _32bit_or_testbench
ModelSim> vsim work._32bit_or_testbench
# vsim work._32bit_or_testbench
# Loading work._32bit_or_testbench
# Loading work._32bit_or_
add wave -position insertpoint  \
sim:/_32bit_or_testbench/a \
sim:/_32bit_or_testbench/b \
sim:/_32bit_or_testbench/result
VSIM 5> step -current
# time =  0, a=00100001001010000010011111000110, b=00101111000000000000011111000110, result=00101111001010000010011111000110
# time = 20, a=00000000000000000000010111000110, b=00101111001010000000011111000110, result=00101111001010000000011111000110
# time = 40, a=00000000000000000000000010000110, b=00000010000000010000011111000110, result=00000010000000010000011111000110

VSIM 6>
```

Here is my alu control truth table and equations for ALUctr.

instruction	$P_3P_2P_1P_0$ ALUop	$F_2F_1F_0$ func	$C_2C_1C_0$ Aluselect
AND	0000	000	110
ADD	0000	001	000
SUB	0000	010	010
XOR	0000	011	001
NOR	0000	100	101
OR	0000	101	111
ADDI	0001	XXX	000
ANDI	0010	XXX	110
ORI	0011	XXX	111
NORI	0100	XXX	101
BEQ	0101	XXX	011
BNE	0110	XXX	011
SLTI	0111	XXX	100
LW	1000	XXX	011
SW	1001	XXX	011

$$C_2 = P_3'P_2'P_1'P_0'F_1'F_2'F_0' + F_2P_3'P_2'P_1'P_0' + P_3'P_2'P_1 + P_3'P_1P_0 + P_3'P_2P_1'P_0'$$

$$C_1 = P_3'P_2'P_1'P_0'(F_2'F_1'F_0' + F_2'F_1F_0' + F_2F_1'F_0) + P_3'P_1P_0' + P_3'P_2'P_1 + P_3P_2'P_1' + P_3'P_2P_1'P_0$$

$$C_0 = P_3'P_2'P_1'P_0'(F_2 + F_1F_0) + P_3'P_2P_1' + \cancel{P_3P_2'}P_1' + P_3'P_2'P_1P_0 + P_3'P_2P_1P_0'$$

_alu_control_.v module :

```
Transcript
# -- Compiling module _alu_control_testbench
#
# Top level modules:
#        _alu_control_testbench
ModelSim> vsim work._alu_control_testbench
# vsim work._alu_control_testbench
# Loading work._alu_control_testbench
# Loading work._alu_control
add wave -position insertpoint  \
sim:/_alu_control_testbench/alu_op \
sim:/_alu_control_testbench/func \
sim:/_alu_control_testbench/alu_ctr
VSIM 5> step -current
# time =  0, alu_op=0000, func=000, aluctr=110
# time = 20, alu_op=0000, func=001, aluctr=000
# time = 40, alu_op=0000, func=010, aluctr=010
# time = 60, alu_op=0000, func=011, aluctr=001
# time = 80, alu_op=0000, func=100, aluctr=101
# time = 100, alu_op=0000, func=101, aluctr=111
# time = 120, alu_op=0001, func=000, aluctr=000
# time = 140, alu_op=0010, func=000, aluctr=110
# time = 160, alu_op=0011, func=000, aluctr=111
# time = 180, alu_op=0100, func=000, aluctr=101
# time = 200, alu_op=0101, func=000, aluctr=011
# time = 220, alu_op=0110, func=000, aluctr=011
# time = 240, alu_op=0111, func=000, aluctr=100
# time = 260, alu_op=1000, func=000, aluctr=011
# time = 280, alu_op=1001, func=000, aluctr=011

VSIM 6>
```

In this module, operation is determine according to opcode coming from ALUcontrol.

alu32.v module

```
Transcript
Loading work._32bit_sub_
Loading work.full_adder_
Loading work.half_adder_
Loading work._32bit_xor_
Loading work.xor_
Loading work._32bit_adder_
Loading work._8_8_1_mux_
Loading work._4_1_mux_
Loading work._2_1_mux_
** Warning: (vsim-3015) C:/altera/13.1/workspace/hw3/_32bit_slt_.v(9): [PCDPC] - Port size (1 or 1) does not match connection size (32) for port 'carr
32bit_sub_.v(1).

        Region: /_alu32_testbench/io/slt0/s32
dd wave -position insertpoint  \
im:/_alu32_testbench/a \
im:/_alu32_testbench/b \
im:/_alu32_testbench/op_code \
im:/_alu32_testbench/out1
SIM 5> step -current
time =  0, a=00100001001010000010011111000110, b=00101111000000000000011111000110, op_code=000, result=01010000001010000010111110001100
time = 20, a=00000000000000000000010111000110, b=00101111001010000000011111000110, op_code=001, result=00101111001010000000000001000000000
time = 40, a=00000000000000000000000000010000110, b=00000010000000001000001111111000110, op_code=010, result=11111101111111101111100011000000
time = 60, a=00000000000000000000000000010000110, b=00000010000000001000001111111000110, op_code=100, result=00000000000000000000000000000000001
time = 80, a=00000000000000000000000000010000110, b=00000010000000001000001111111000110, op_code=101, result=11111101111111101111100000111001
time = 100, a=00000000000000000000000000010000110, b=00000010000000001000001111111000110, op_code=110, result=00000000000000000000000000010000110
time = 120, a=00000000000000000000000000010000110, b=00000010000000001000001111111000110, op_code=111, result=00000010000000001000001111111000110

SIM 6>
```

In _alu32_.v module, each operation is calculated. And then output is generated as result according to opcode. So we have to use a 8x1 multiplexer that takes eight parameters.

_8_8_1_mux_.v module

```
Transcript
vlog -reportprogress 300 -work work C:/altera/13.1/workspace/hw3/_8_8_1_mux_testbench.v
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module _8_8_1_mux_testbench
#
# Top level modules:
#        _8_8_1_mux_testbench
ModelSim> vsim work._8_8_1_mux_testbench
# vsim work._8_8_1_mux_testbench
# Loading work._8_8_1_mux_testbench
# Loading work._8_8_1_mux_
# Loading work._4_1_mux_
# Loading work._2_1_mux_
add wave -position insertpoint  \
sim:/_8_8_1_mux_testbench/result \
sim:/_8_8_1_mux_testbench/op_code \
sim:/_8_8_1_mux_testbench/a7 \
sim:/_8_8_1_mux_testbench/a6 \
sim:/_8_8_1_mux_testbench/a5 \
sim:/_8_8_1_mux_testbench/a4 \
sim:/_8_8_1_mux_testbench/a3 \
sim:/_8_8_1_mux_testbench/a2 \
sim:/_8_8_1_mux_testbench/a1 \
sim:/_8_8_1_mux_testbench/a0
VSIM 5> step -current
# time =  0, a0=1, a1=1, a2=0, a3=0, a4=1, a5=1, a6=0, a7=1, op_code=000, result=1
# time = 20, a0=1, a1=1, a2=0, a3=0, a4=1, a5=1, a6=0, a7=1, op_code=110, result=0

VSIM 6>
```

Here is truth table of control unit.

	0000 R-type	0001 addi	0010 andi	0011 ori	0100 nori	0111 slti	1000 lw	1001 sw	0101 beq	0110 bne
RegDst	1	0	0	0	0	0	0	x	x	x
ALUsrc	0	1	1	1	1	1	1	1	0	0
Memtoreg	0	0	0	0	0	0	1	x	x	x
Regwrite	1	1	1	1	1	1	1	0	0	0
MemRead	0	0	0	0	0	0	1	0	0	0
MemWrite	0	0	0	0	0	0	0	1	0	0
Branch	0	0	0	0	0	0	0	0	1	0
ALUop3	0	0	0	0	0	0	1	1	0	0
ALUop2	0	0	0	0	1	1	0	0	1	1
ALUop1	0	0	1	1	0	1	0	0	0	1
ALUop0	0	1	0	1	0	1	0	1	1	0
Branchnot	0	0	0	0	0	0	0	0	0	1

control.v module:

```
Transcript
ModelSim> vsim work._control_testbench
# vsim work._control_testbench
# Loading work._control_testbench
# Loading work._control_
add wave -position insertpoint  \
sim:/_control_testbench/RegWrite \
sim:/_control_testbench/RegDst \
sim:/_control_testbench/Opcode \
sim:/_control_testbench/MemtoReg \
sim:/_control_testbench/MemWrite \
sim:/_control_testbench/MemRead \
sim:/_control_testbench/BranchNot \
sim:/_control_testbench/Branch \
sim:/_control_testbench/ALUSrc \
sim:/_control_testbench/ALUOp
VSIM 5> step -current
# time =  0, RegDst=1, ALUSrc=0, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, BranchNot=0, ALUOp=0000, Opcode=0000
# time = 20, RegDst=0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, BranchNot=0, ALUOp=0001, Opcode=0001
# time = 40, RegDst=0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, BranchNot=0, ALUOp=0010, Opcode=0010
# time = 60, RegDst=0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, BranchNot=0, ALUOp=0011, Opcode=0011
# time = 80, RegDst=0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, BranchNot=0, ALUOp=0100, Opcode=0100
# time = 100, RegDst=0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, BranchNot=0, ALUOp=0111, Opcode=0111
# time = 120, RegDst=0, ALUSrc=1, MemtoReg=1, RegWrite=1, MemRead=1, MemWrite=0, Branch=0, BranchNot=0, ALUOp=1000, Opcode=1000
# time = 140, RegDst=0, ALUSrc=1, MemtoReg=0, RegWrite=0, MemRead=0, MemWrite=1, Branch=0, BranchNot=0, ALUOp=1001, Opcode=1001
# time = 160, RegDst=0, ALUSrc=0, MemtoReg=0, RegWrite=0, MemRead=0, MemWrite=0, Branch=1, BranchNot=0, ALUOp=0101, Opcode=0101
# time = 180, RegDst=0, ALUSrc=0, MemtoReg=0, RegWrite=0, MemRead=0, MemWrite=0, Branch=0, BranchNot=1, ALUOp=0110, Opcode=0110

VSIM 6>
```

_instruction_memory_.v module

```
sim:/_instruction_memory_testbench/read_address \
sim:/_instruction_memory_testbench/_clock \
sim:/_instruction_memory_testbench/instruction
VSIM 5> step -current
# time =  0, read_address=000000, instruction=0000011010001000
# time = 100, read_address=000001, instruction=0000011010001000
# time = 200, read_address=000010, instruction=0001000100000100
# time = 300, read_address=000011, instruction=0000010010011001
# time = 400, read_address=000100, instruction=0010000100000001
# time = 500, read_address=000101, instruction=0000001000010010
# time = 600, read_address=000110, instruction=0011000001000100
# time = 700, read_address=000111, instruction=0000010010010011
# time = 800, read_address=001000, instruction=0100000100000010
# time = 900, read_address=001001, instruction=0000101001010100
# time = 1000, read_address=001010, instruction=0111010001000001
# time = 1100, read_address=001011, instruction=0000100101010101
# time = 1200, read_address=001100, instruction=1000100000000001
# time = 1300, read_address=001101, instruction=1001010101000010
# time = 1400, read_address=001110, instruction=0000101010001000
# time = 1500, read_address=001111, instruction=0001000101100100
# time = 1600, read_address=010000, instruction=0000000110011001
# time = 1700, read_address=010001, instruction=0010001100000001
# time = 1800, read_address=010010, instruction=0000111000010010
# time = 1900, read_address=010011, instruction=0011001001000100
# time = 2000, read_address=010100, instruction=0000010010110011
# time = 2100, read_address=010101, instruction=0100001100000010
# time = 2200, read_address=010110, instruction=0000100001010100
# time = 2300, read_address=010111, instruction=0111010011000001
# time = 2400, read_address=011000, instruction=0000100111010101
# time = 2500, read_address=011001, instruction=1000101000000001
# time = 2600, read_address=011010, instruction=1001010001000010
```

If number is negative, it is extended with 1 to 32 bits binary number.

Else, extended with 0.

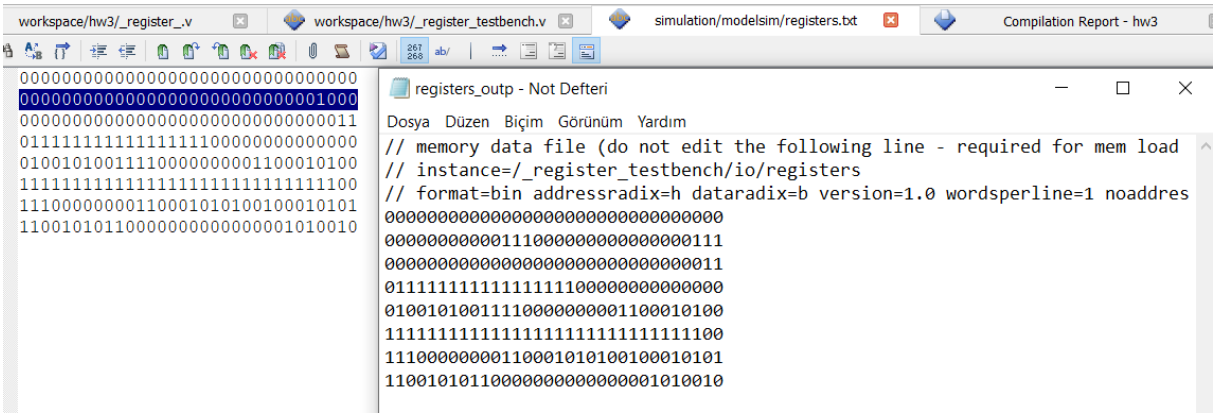
_sign_extend_.v module

```
#
# Top level modules:
#        _sign_extend_testbench
ModelSim> vsim work._sign_extend_testbench
# vsim work._sign_extend_testbench
# Loading work._sign_extend_testbench
# Loading work._sign_extend_
add wave -position insertpoint  \
sim:/_sign_extend_testbench/immediate \
sim:/_sign_extend_testbench/extended
VSIM 5> step -current
# time =  0, immediate=100001, extended=11111111111111111111111111100001
# time = 20, immediate=001110, extended=00000000000000000000000000001110

VSIM 6>
```

register.v module

```
# Top level modules:
#       _register_testbench
ModelSim> vsim work._register_testbench
# vsim work._register_testbench
# Loading work._register_testbench
# Loading work._register_
add wave -position insertpoint  \
sim:/_register_testbench/write_reg \
sim:/_register_testbench/read_reg1 \
sim:/_register_testbench/read_reg2 \
sim:/_register_testbench/write_data \
sim:/_register_testbench/reg_write \
sim:/_register_testbench/_clock \
sim:/_register_testbench/read_data1 \
sim:/_register_testbench/read_data2
VSIM 6> step -current
# time =  0, read_data1=00000000000000000000000000000011, read_data2=01111111111111111100000000000000, read_reg1=010, read_reg2=011, write_reg=001, write_data=00000000000111000000000000000111, reg_write=1
# time = 200, read_data1=00000000000000000000000000000011, read_data2=01111111111111111100000000000000, read_reg1=010, read_reg2=011, write_reg=001, write_data=00000000000111000000000000000111, reg_write=0

VSIM 7>
```

In the below, left one is before of register.txt; right one is after register.txt. Because of reg_write is 1 in _register_testbench.v file, data is written to 1th index.



_data_memory_.v module

```
sim:/_data_memory_testbench/write_data \
sim:/_data_memory_testbench/mem_read \
sim:/_data_memory_testbench/mem_write \
sim:/_data_memory_testbench/_clock \
sim:/_data_memory_testbench/read_data
VSIM 5> step -current
# time =  0, read_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, address=00000000000000000000000000000111, write_data=00000000000111000000000000000111, mem_read=0, mem_write=1
# time = 200, read_data=00000000000111000000000000000111, address=00000000000000000000000000000111, write_data=00000000000111000000000000000111, mem_read=1, mem_write=0

VSIM 6>
```

Before write to memory:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/_mini_mips_testbench_/io/io8/cells
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000010100
00000000000000000000000000000010
00000000000000000000000000000011
11111111111111111111111111111100
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
```

After writing to memory:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/_data_memory_testbench/io/cells
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000010100
00000000000000000000000000000010
00000000000000000000000000000011
11111111111111111111111111111100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000011100000000000000111
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
```

A multiplexer is designed that decides whether the result calculated by the ALU or the value read from the memory will be written to the register and takes two 32 bits numbers.

_2_1_mux_32bit_.v module

```
# Top level modules:
#        _2_1_mux_32bit_testbench
ModelSim> vsim work._2_1_mux_32bit_testbench
# vsim work._2_1_mux_32bit_testbench
# Loading work._2_1_mux_32bit_testbench
# Loading work._2_1_mux_32bit_
# Loading work._2_1_2I_mux_
add wave -position insertpoint  \
sim:/_2_1_mux_32bit_testbench/a \
sim:/_2_1_mux_32bit_testbench/b \
sim:/_2_1_mux_32bit_testbench/s \
sim:/_2_1_mux_32bit_testbench/outp
VSIM 5> step -current
# time =  0, a=00000000000111000001000000000100, b=00000001000000000000000000000001, s=0, outp=00000000000111000001000000000100
# time = 20, a=00000000000111000001000000000100, b=00000001000000000000000000000001, s=1, outp=00000001000000000000000000000001

VSIM 6>
```

The 2x1 multiplexer is designed that takes two seperate numbers(a, b) instead of an array(a[1:0]).

_2_1_2I_mux_.v module

```
# Top level modules:
#        _2_1_2I_mux_testbench
ModelSim> vsim work._2_1_2I_mux_testbench
# vsim work._2_1_2I_mux_testbench
# Loading work._2_1_2I_mux_testbench
# Loading work._2_1_2I_mux_
add wave -position insertpoint  \
sim:/_2_1_2I_mux_testbench/a \
sim:/_2_1_2I_mux_testbench/b \
sim:/_2_1_2I_mux_testbench/s \
sim:/_2_1_2I_mux_testbench/out1
VSIM 5> step -current
# time =  0, a=0, b=1, s=0, out1=0
# time = 20, a=0, b=1, s=1, out1=1

VSIM 6>
```

Here is main module.

mini_mips.v module

```
add wave -position insertpoint  \
sim:/_mini_mips_testbench_/_clock \
sim:/_mini_mips_testbench_/read_address
VSIM 5> step -current
# Opcode=0000, Rs=011, Rt=010, immediate=001000
# read_data1=01111111111111111100000000000000, read_data2=00000000000000000000000000000011, Result=00000000000000000000000000000000
# Opcode=0001, Rs=000, Rt=100, immediate=000100
# read_data1=00000000000000000000000000000000, read_data2=01001010011110000000001100010100, Result=00000000000000000000000000000100
# Opcode=0000, Rs=010, Rt=010, immediate=011001
# read_data1=00000000000000000000000000000011, read_data2=00000000000000000000000000000011, Result=00000000000000000000000000000110
# Opcode=0010, Rs=000, Rt=100, immediate=000001
# read_data1=00000000000000000000000000000000, read_data2=00000000000000000000000000000100, Result=00000000000000000000000000000000
# Opcode=0000, Rs=001, Rt=000, immediate=010010
# read_data1=00000000000000000000000000000000, read_data2=00000000000000000000000000000000, Result=00000000000000000000000000000000
# Opcode=0011, Rs=000, Rt=001, immediate=000100
# read_data1=00000000000000000000000000000000, read_data2=00000000000000000000000000000000, Result=00000000000000000000000000000100
# Opcode=0000, Rs=010, Rt=010, immediate=010011
# read_data1=00000000000000000000000000000000, read_data2=00000000000000000000000000000000, Result=00000000000000000000000000000000
# Opcode=0100, Rs=000, Rt=100, immediate=000010
# read_data1=00000000000000000000000000000000, read_data2=00000000000000000000000000000000, Result=11111111111111111111111111111101
# Opcode=0000, Rs=101, Rt=001, immediate=010100
# read_data1=11111111111111111111111111111100, read_data2=00000000000000000000000000000100, Result=00000000000000000000000000000011
# Opcode=0111, Rs=010, Rt=001, immediate=000001
# read_data1=00000000000000000000000000000011, read_data2=00000000000000000000000000000100, Result=00000000000000000000000000000000
# Opcode=0000, Rs=100, Rt=101, immediate=010101
# read_data1=11111111111111111111111111111101, read_data2=11111111111111111111111111111100, Result=11111111111111111111111111111101
# Opcode=1000, Rs=100, Rt=000, immediate=000001
# read_data1=11111111111111111111111111111101, read_data2=00000000000000000000000000000000, Result=11111111111111111111111111111110
# Opcode=1001, Rs=010, Rt=101, immediate=000010
# read_data1=11111111111111111111111111111101, read_data2=11111111111111111111111111111100, Result=11111111111111111111111111111111
# Opcode=0000, Rs=101, Rt=010, immediate=001000
# read_data1=11111111111111111111111111111100, read_data2=11111111111111111111111111111101, Result=11111111111111111111111111111100
# Opcode=0001, Rs=000, Rt=101, immediate=100100
# read_data1=00000000000000000000000000011110, read_data2=11111111111111111111111111111100, Result=00000000000000000000000000000010
# Opcode=0000, Rs=000, Rt=110, immediate=011001
# read_data1=00000000000000000000000000011110, read_data2=11100000000110001010100100010101, Result=11100000000110001010100100110011
# Opcode=0010, Rs=001, Rt=100, immediate=000001
# read_data1=11111111111111111111111111111101, read_data2=11111111111111111111111111111101, Result=00000000000000000000000000000000
```

```
Transcript
# Opcode=0000, Rs=101, Rt=001, immediate=010100
# read_data1=11111111111111111111111111111100, read_data2=00000000000000000000000000000100, Result=00000000000000000000000000000011
# Opcode=0111, Rs=010, Rt=001, immediate=000001
# read_data1=00000000000000000000000000000011, read_data2=00000000000000000000000000000100, Result=00000000000000000000000000000000
# Opcode=0000, Rs=100, Rt=101, immediate=010101
# read_data1=11111111111111111111111111111101, read_data2=11111111111111111111111111111100, Result=11111111111111111111111111111101
# Opcode=1000, Rs=100, Rt=000, immediate=000001
# read_data1=11111111111111111111111111111101, read_data2=00000000000000000000000000000000, Result=11111111111111111111111111111110
# Opcode=1001, Rs=010, Rt=101, immediate=000010
# read_data1=11111111111111111111111111111101, read_data2=11111111111111111111111111111100, Result=11111111111111111111111111111111
# Opcode=0000, Rs=101, Rt=010, immediate=001000
# read_data1=11111111111111111111111111111100, read_data2=11111111111111111111111111111101, Result=11111111111111111111111111111100
# Opcode=0001, Rs=000, Rt=101, immediate=100100
# read_data1=00000000000000000000000000011110, read_data2=11111111111111111111111111111100, Result=00000000000000000000000000000010
# Opcode=0000, Rs=000, Rt=110, immediate=011001
# read_data1=00000000000000000000000000011110, read_data2=11100000000110001010100100010101, Result=11100000000110001010100100110011
# Opcode=0010, Rs=001, Rt=100, immediate=000001
# read_data1=11111111111111111111111111111100, read_data2=11111111111111111111111111111101, Result=00000000000000000000000000000000
# Opcode=0000, Rs=111, Rt=000, immediate=010010
# read_data1=11001010110000000000000001010010, read_data2=00000000000000000000000000011110, Result=11001010110000000000000000110100
# Opcode=0011, Rs=001, Rt=001, immediate=000100
# read_data1=11111111111111111111111111111100, read_data2=11111111111111111111111111111100, Result=11111111111111111111111111111100
# Opcode=0000, Rs=010, Rt=010, immediate=110011
# read_data1=11001010110000000000000000110100, read_data2=11001010110000000000000000110100, Result=00000000000000000000000000000000
# Opcode=0100, Rs=001, Rt=100, immediate=000010
# read_data1=11111111111111111111111111111100, read_data2=00000000000000000000000000000000, Result=00000000000000000000000000000001
# Opcode=0000, Rs=100, Rt=001, immediate=010100
# read_data1=00000000000000000000000000000001, read_data2=11111111111111111111111111111100, Result=00000000000000000000000000000010
# Opcode=0111, Rs=010, Rt=011, immediate=000001
# read_data1=00000000000000000000000000000010, read_data2=11100000000110001010100100110011, Result=00000000000000000000000000000000
# Opcode=0000, Rs=100, Rt=111, immediate=010101
# read_data1=00000000000000000000000000000001, read_data2=11001010110000000000000001010010, Result=11001010110000000000000001010011
# Opcode=1000, Rs=101, Rt=000, immediate=000001
# read_data1=00000000000000000000000000000010, read_data2=00000000000000000000000000011110, Result=00000000000000000000000000000011
# Opcode=1001, Rs=010, Rt=001, immediate=000010
# read_data1=11001010110000000000000001010011, read_data2=11111111111111111111111111111100, Result=11001010110000000000000001010101

VSIM 6>
```

Before registers.txt:

```
00000000000000000000000000000000
00000000000000000000000000001000
00000000000000000000000000000011
01111111111111111110000000000000
01001010011110000000001100010100
11111111111111111111111111111100
11100000000110001010100100010101
11001010110000000000000001010010
```

After testbench of mini_mips module is run, state of registers.txt.

registers_outp - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
// memory data file (do not edit the following line - required for mem load
// instance=/_mini_mips_testbench_/io/io11/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddres
00000000000000000000000000011110
11111111111111111111111111111100
11001010110000000000000001010011
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000000
11001010110000000000000001010010
```