

Important Notes:

* I didn't add branch

* My ALU is same with hw3 , errors still exist

* My sign extender adds 26 times 0 front of the immediate to make it 32 bit

* With .qar I add all files in a zip.

* Some results may be wrong. Because of time problems I cannot make a great error check.

SIGNALS TABLE:

INSTR.	OPCODE	FUNCT	ALU OP	ALU CTR	ALU ACT	Mem Write	Mem Read	Reg Dst	Mem ToReg	Reg Write	ALU SRC
AND	0000	000	111	000	AND	0	0	1	0	1	0
ADD	0000	001	111	001	ADD	0	0	1	0	1	0
SUB	0000	010	111	010	SUB	0	0	1	0	1	0
XOR	0000	011	111	011	XOR	0	0	1	0	1	0
NOR	0000	100	111	100	NOR	0	0	1	0	1	0
OR	0000	101	111	101	OR	0	0	1	0	1	0
ADDI	0001	XXX	001	001	ADD	0	0	0	0	1	1
ANDI	0010	XXX	000	000	AND	0	0	0	0	1	1
ORI	0011	XXX	101	101	OR	0	0	0	0	1	1
NORI	0100	XXX	100	100	NOR	0	0	0	0	1	1
BEQ	0101	XXX	110	110	SLT	0	0	0	0	0	0
BNE	0110	XXX	110	110	SLT	0	0	0	0	0	0
SLTI	0111	XXX	110	110	SLT	0	0	0	0	1	1
LW	1000	XXX	001	001	ADD	0	1	0	1	1	1
SW	1001	XXX	001	001	ADD	1	0	0	0	0	1

ALUOp 0 = Opcode0 + (Opcode2 * Opcode0) + ~(Opcode3 + Opcode2 + Opcode1 + Opcode0)

ALUOp 1 = Opcode2 * (Opcode1 + Opcode0) + ~(Opcode3 + Opcode2 + Opcode1 + Opcode0)

ALUOp 2 = Opcode2 + (Opcode1 * Opcode0) + ~(Opcode3 + Opcode2 + Opcode1 + Opcode0)

MemWrite = Opcode3 * Opcode0

MemRead = Opcode3 * ~Opcode0

RegDst = ~(Opcode3 + Opcode2 + Opcode1 + Opcode0)

MemToReg = Opcode3 * ~Opcode0

RegWrite = ~((Opcode3 xor Opcode2) * (Opcode1 xor Opcode0))

ALUSrc = ~((Opcode1 xor Opcode0) * Opcode2) + ~(Opcode3 + Opcode2 + Opcode1 + Opcode0)

ALUCtr0 = if (ALUOp0 * ALUOp1 * ALUOp2) == 1 { funct0 or 0 } else {ALUOp0 or 0}

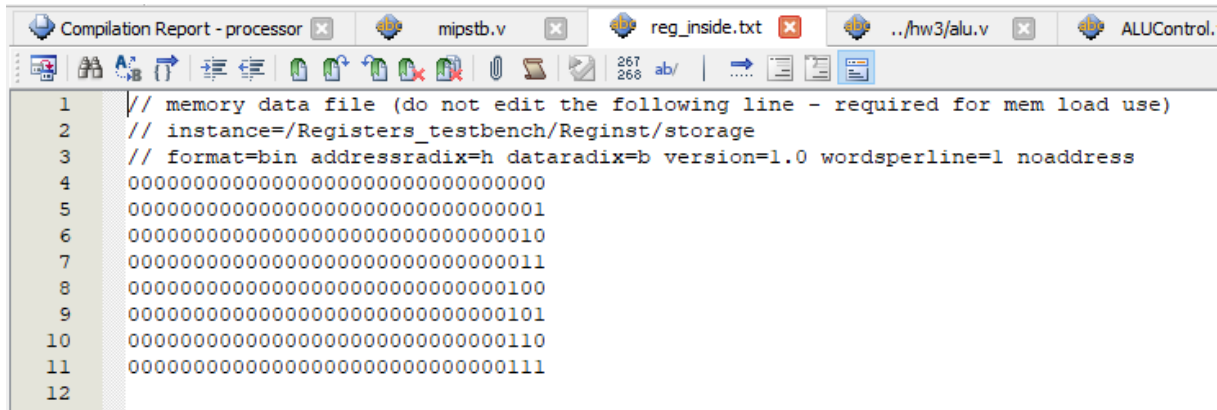
ALUCtr1 = if (ALUOp0 * ALUOp1 * ALUOp2) == 1 { funct1 or 0 } else {ALUOp1 or 0}

ALUCtr2 = if (ALUOp0 * ALUOp1 * ALUOp2) == 1 { funct2 or 0 } else {ALUOp2 or 0}

TEST RESULTS:

mipstb: Testbench of minips.v (complete processor)

Initial values of registers:



```
1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=/Registers_testbench/Reginst/storage
3 // format=bin addressradix=h data radix=b version=1.0 wordsperline=1 noaddress
4 00000000000000000000000000000000
5 00000000000000000000000000000001
6 00000000000000000000000000000010
7 00000000000000000000000000000011
8 00000000000000000000000000000100
9 00000000000000000000000000000101
10 00000000000000000000000000000110
11 00000000000000000000000000000111
12
```

Initial values of memory is same with draft file.

Instruction list:

```
instruction = 32'b0000000001010000; // 000 ve 001 i andle 010 a yaz
#20 instruction = 32'b0000000010011000; // 000 ve 010 i andle 011 e yaz
#20 instruction = 32'b0000000011100001; // 000 + 011 -> 100
#20 instruction = 32'b00000000100101001; // 000 + 100 -> 101
#20 instruction = 32'b0000101000110010; // 101 - 000 -> 110
#20 instruction = 32'b0000110000111010; // 110 - 000 -> 111
#20 instruction = 32'b00000000111001011; // 000 xor 111 -> 001
#20 instruction = 32'b000000001010011; // 000 xor 001 -> 010
#20 instruction = 32'b0000000010011100; // 000 nor 010 -> 011
#20 instruction = 32'b0000000011100100; // 000 nor 011 -> 100
#20 instruction = 32'b00000000100101101; // 000 or 100 -> 101
#20 instruction = 32'b00000000101110101; // 000 or 101 -> 110
#20 instruction = 32'b0001000110111110; // 000 + 111110 -> 110
#20 instruction = 32'b0001000111001110; // 000 + 001110 -> 111
#20 instruction = 32'b0010000001010111; // 000 and 010111 -> 001
#20 instruction = 32'b0010000010011111; // 000 and 011111 -> 010
#20 instruction = 32'b0011000011100000; // 000 or 100000 -> 011
#20 instruction = 32'b0011000100101000; // 000 or 101000 -> 100
#20 instruction = 32'b0100000101110001; // 000 nor 110001 -> 101
#20 instruction = 32'b0100000110111001; // 000 nor 111001 -> 110
#20 instruction = 32'b0111000100101100; // 000 slt 101100 -> 100
#20 instruction = 32'b0111000101110100; // 000 slt 110100 -> 101
#20 instruction = 32'b10000000110111101; // mem[111101] -> 110
#20 instruction = 32'b10000000111001101; // mem [001101] -> 111
#20 instruction = 32'b1001000001010110; // 001 -> mem [010110]
#20 instruction = 32'b1001000010011110; // 010 -> mem [011110]
```

Test results:

```
# time:          0, instruction: 000000001010000, result: 00000000000000000000000000000000
#
# time:         20, instruction: 0000000010011000, result: 00000000000000000000000000000000
#
# time:         40, instruction: 0000000011100001, result: 00000000000000000000000000000000
#
# time:         60, instruction: 00000000100101001, result: 00000000000000000000000000000000
#
# time:         80, instruction: 0000101000110010, result: 00000000000000000000000000000000
#
# time:        100, instruction: 0000110000111010, result: 00000000000000000000000000000000
#
# time:        120, instruction: 0000000111001011, result: 00000000000000000000000000000000
#
# time:        140, instruction: 0000000001010011, result: 00000000000000000000000000000000
#
# time:        160, instruction: 0000000010011100, result: 11111111111111111111111111111111
#
# time:        180, instruction: 0000000011100100, result: 11111111111111111111111111111111
#
# time:        180, instruction: 0000000011100100, result: 00000000000000000000000000000000
#
# time:        180, instruction: 0000000011100100, result: 00000000000000000000000000000000
#
# time:        200, instruction: 0000000100101101, result: 11111111111111111111111111111111
#
# time:        200, instruction: 0000000100101101, result: 00000000000000000000000000000000
#
# time:        200, instruction: 0000000100101101, result: 00000000000000000000000000000000
#
# time:        220, instruction: 0000000101110101, result: 00000000000000000000000000000000
#
# time:        240, instruction: 0001000110111110, result: 000000000000000000000000000011110
#
# time:        260, instruction: 0001000111001110, result: 00000000000000000000000000001110
```

```
# time:        260, instruction: 0001000111001110, result: 00000000000000000000000000001110
#
# time:        280, instruction: 0010000001010111, result: 00000000000000000000000000000000
#
# time:        300, instruction: 0010000010011111, result: 00000000000000000000000000000000
#
# time:        320, instruction: 0011000011100000, result: 0000000000000000000000000000100000
#
# time:        340, instruction: 0011000100101000, result: 0000000000000000000000000000101000
#
# time:        360, instruction: 0100000101110001, result: 11111111111111111111111111001110
#
# time:        380, instruction: 0100000110111001, result: 11111111111111111111111111000110
#
# time:        400, instruction: 0111000100101100, result: 11111111111111111111111111111111
#
# time:        420, instruction: 0111000101110100, result: 11111111111111111111111111111111
#
# time:        440, instruction: 1000000110111101, result: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
# time:        440, instruction: 1000000110111101, result: 000000000000000000000000000011101
#
# time:        440, instruction: 1000000110111101, result: 000000000000000000000000000011101
#
# time:        460, instruction: 1000000111001101, result: 000000000000000000000000000011101
#
# time:        460, instruction: 1000000111001101, result: 000000000000000000000000000011101
#
# time:        460, instruction: 1000000111001101, result: 000000000000000000000000000011101
#
# time:        480, instruction: 1001000001010110, result: 000000000000000000000000000010110
#
# time:        480, instruction: 1001000001010110, result: 000000000000000000000000000010110
#
# time:        500, instruction: 1001000010011110, result: 000000000000000000000000000011110
#
# time:        500, instruction: 1001000010011110, result: 000000000000000000000000000011110
#
```

Final values of registers:

[illegible]

Final values of memory:

It is so long so I didn't add it in this report file. File name is mem_end.txt. You can find it in zip file and rar file.

Tests results of modules:

alucontrol:

ALUControl_testbench/Func	111	000	001	010	011	100	101	110	111	000	001					101	100	111
ALUControl_testbench/ALUOp	001	111						001	000	101	100	110					001	
ALUControl_testbench/ALUCtr	001	000	001	010	011	100	101	001	000	101	100	110					001	

Control unit:

[illegible]

Memory:

[illegible]

Register:

1	/Registers_testbench/RegWrite	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
---	-------------------------------	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Sign Extender:

Timing diagram showing two digital signals over 18 ps. The top signal, `/signExtender_testbench/imm`, is a constant high signal (011100). The bottom signal, `/signExtender_testbench/signExt...`, is a constant low signal (000000). The signals are shown as horizontal lines with binary values at the top and a time axis at the bottom.