GTU

DEPARTMENT OF

COMPUTER ENGINEERING

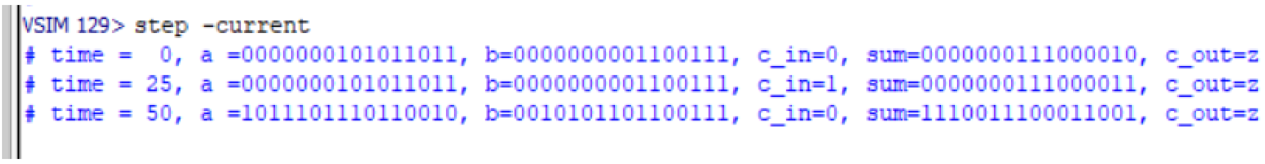
CSE 331
Computer Organization
Autumn 2022/2023

HOMEWORK 3 REPORT

SÜLEYMAN GÖLBOL

1801042656

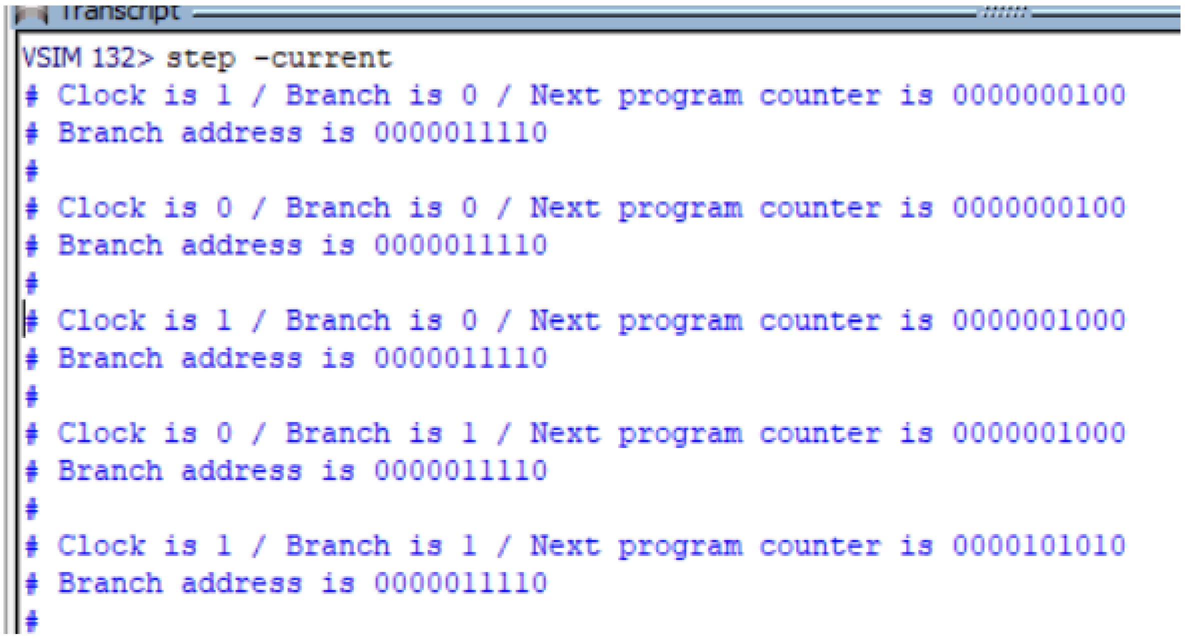
Adder Testbench



```
VSIM 129> step -current
# time =  0, a =0000000101011011, b=0000000001100111, c_in=0, sum=0000000111000010, c_out=z
# time = 25, a =0000000101011011, b=0000000001100111, c_in=1, sum=0000000111000011, c_out=z
# time = 50, a =1011101110110010, b=0010101101100111, c_in=0, sum=1110011100011001, c_out=z
```

Other 16 bit modules works like that.

Next_program_counter Component

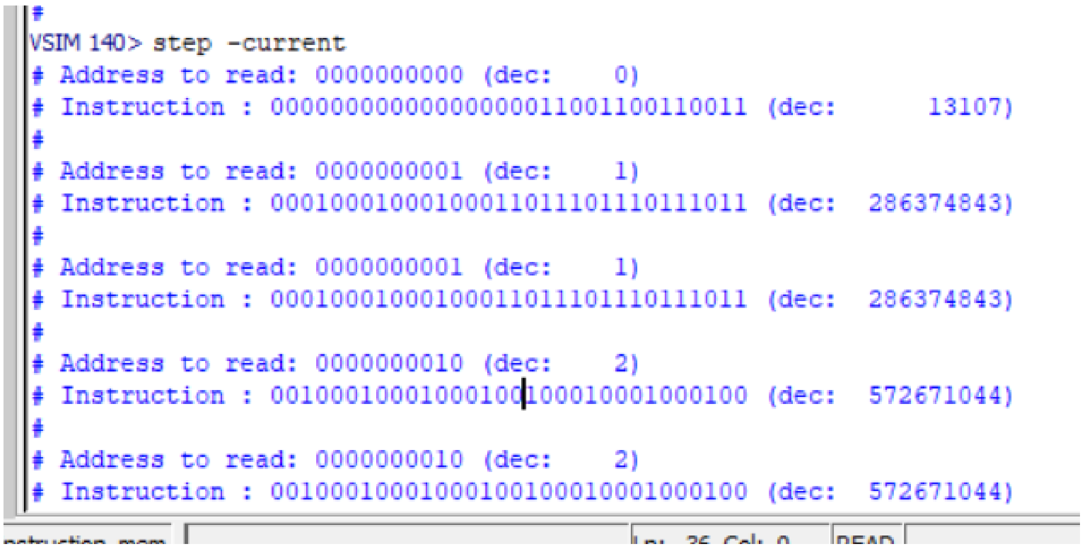


```
VSIM 132> step -current
# Clock is 1 / Branch is 0 / Next program counter is 0000000100
# Branch address is 0000011110
#
# Clock is 0 / Branch is 0 / Next program counter is 0000000100
# Branch address is 0000011110
#
# Clock is 1 / Branch is 0 / Next program counter is 0000001000
# Branch address is 0000011110
#
# Clock is 0 / Branch is 1 / Next program counter is 0000001000
# Branch address is 0000011110
#
# Clock is 1 / Branch is 1 / Next program counter is 0000101010
# Branch address is 0000011110
#
```

When the clock is 1, it increments the program counter by 4. If branch is equal to 1 it jumps.

Note: I changed to increment counter by 1 so that file operations works correctly.

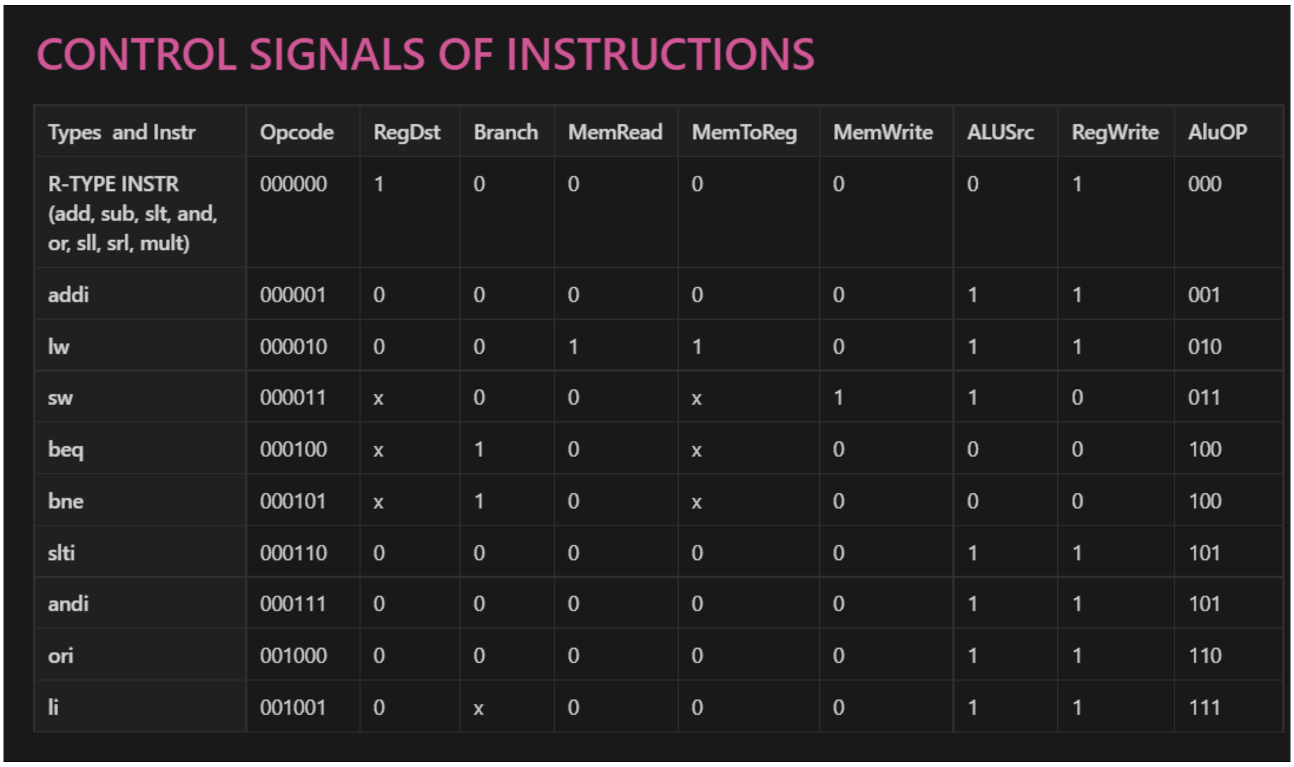
INSTRUCTION_MEM Component



```
#
VSIM 140> step -current
# Address to read: 0000000000 (dec:    0)
# Instruction : 00000000000000000011001100110011 (dec:     13107)
#
# Address to read: 0000000001 (dec:    1)
# Instruction : 00010001000100011011101110111011 (dec:  286374843)
#
# Address to read: 0000000001 (dec:    1)
# Instruction : 00010001000100011011101110111011 (dec:  286374843)
#
# Address to read: 0000000010 (dec:    2)
# Instruction : 00100010001000100100010001000100 (dec:  572671044)
#
# Address to read: 0000000010 (dec:    2)
# Instruction : 00100010001000100100010001000100 (dec:  572671044)
```

I put some value in instruction mem. When reads it prints the instruction successfully. (Input: address to read, output: instruction)

CONTROL COMPONENT

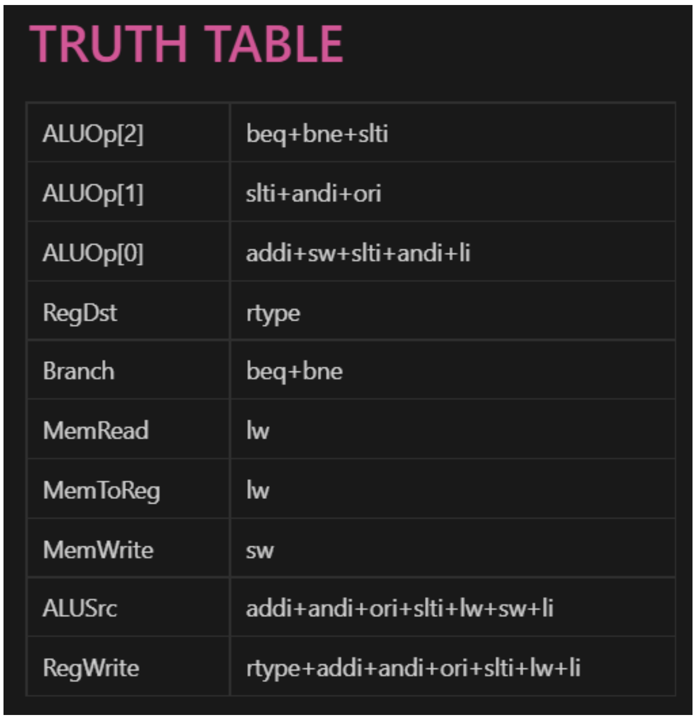


CONTROL SIGNALS OF INSTRUCTIONS

Types and Instr	Opcode	RegDst	Branch	MemRead	MemToReg	MemWrite	ALUSrc	RegWrite	AluOP
R-TYPE INSTR (add, sub, slt, and, or, sll, srl, mult)	000000	1	0	0	0	0	0	1	000
addi	000001	0	0	0	0	0	1	1	001
lw	000010	0	0	1	1	0	1	1	010
sw	000011	x	0	0	x	1	1	0	011
beq	000100	x	1	0	x	0	0	0	100
bne	000101	x	1	0	x	0	0	0	100
slti	000110	0	0	0	0	0	1	1	101
andi	000111	0	0	0	0	0	1	1	101
ori	001000	0	0	0	0	0	1	1	110
li	001001	0	x	0	0	0	1	1	111

These are the control signals for instructions.

Note: Unfortunately, I couldn’t create instructions for j type since I couldn’t handle new signals for jumps.

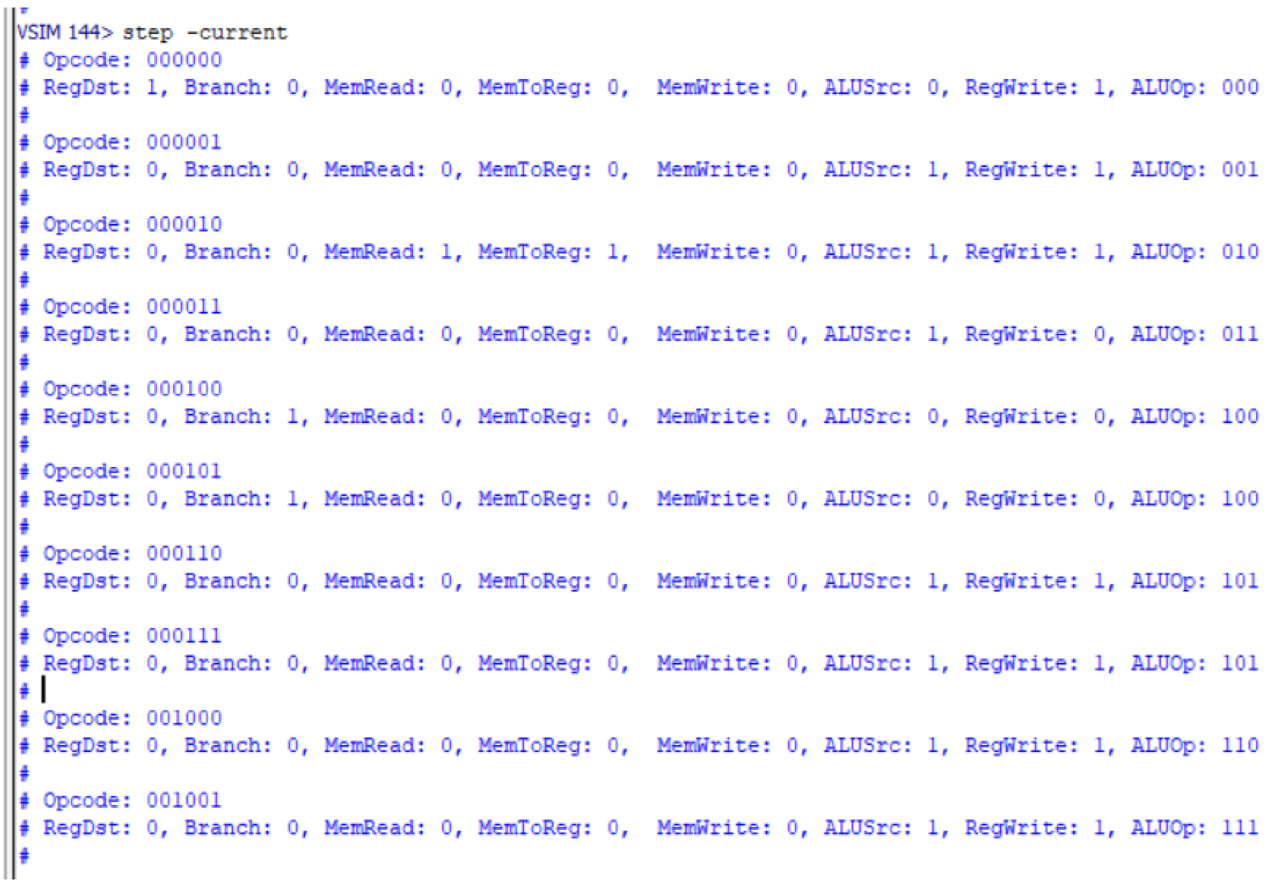


TRUTH TABLE

ALUOp[2]	beq+bne+slti
ALUOp[1]	slti+andi+ori
ALUOp[0]	addi+sw+slti+andi+li
RegDst	rtype
Branch	beq+bne
MemRead	lw
MemToReg	lw
MemWrite	sw
ALUSrc	addi+andi+ori+slti+lw+sw+li
RegWrite	rtype+addi+andi+ori+slti+lw+li

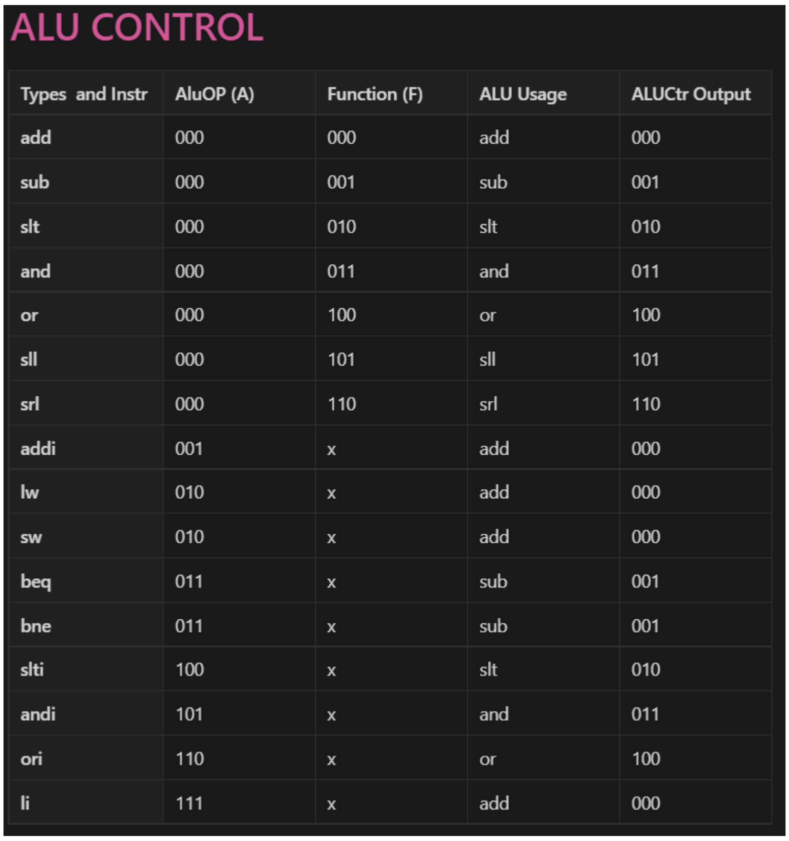
I derived this truth table for signal outputs.

This is the testbench result for control. Which opcode belongs to which instruction writes in above table.



```
VSIM 144> step -current
# Opcode: 000000
# RegDst: 1, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 0, RegWrite: 1, ALUOp: 000
#
# Opcode: 000001
# RegDst: 0, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 001
#
# Opcode: 000010
# RegDst: 0, Branch: 0, MemRead: 1, MemToReg: 1,  MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 010
#
# Opcode: 000011
# RegDst: 0, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 1, RegWrite: 0, ALUOp: 011
#
# Opcode: 000100
# RegDst: 0, Branch: 1, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 0, RegWrite: 0, ALUOp: 100
#
# Opcode: 000101
# RegDst: 0, Branch: 1, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 0, RegWrite: 0, ALUOp: 100
#
# Opcode: 000110
# RegDst: 0, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 101
#
# Opcode: 000111
# RegDst: 0, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 101
#
# Opcode: 001000
# RegDst: 0, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 110
#
# Opcode: 001001
# RegDst: 0, Branch: 0, MemRead: 0, MemToReg: 0,  MemWrite: 0, ALUSrc: 1, RegWrite: 1, ALUOp: 111
#
```

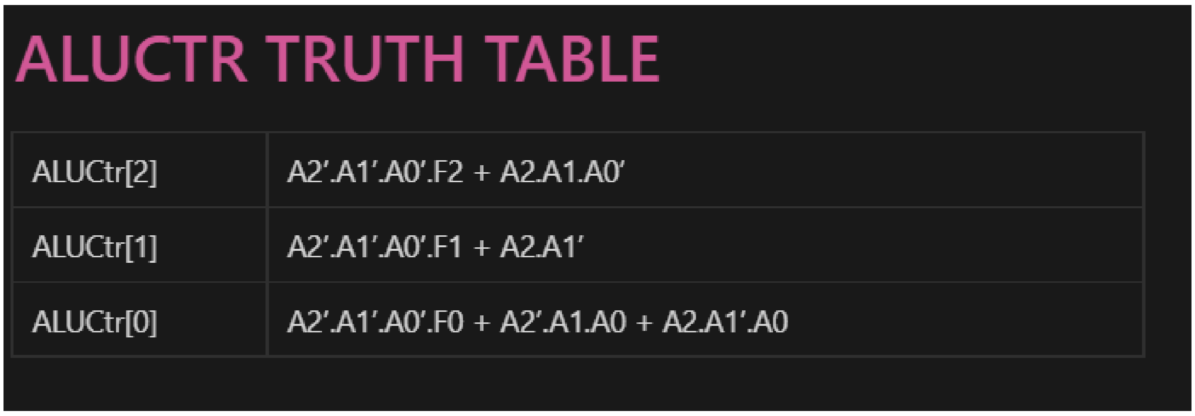
AluControl Component



ALU CONTROL

Types and Instr	AluOP (A)	Function (F)	ALU Usage	ALUCtr Output
add	000	000	add	000
sub	000	001	sub	001
slt	000	010	slt	010
and	000	011	and	011
or	000	100	or	100
sll	000	101	sll	101
srl	000	110	srl	110
addi	001	x	add	000
lw	010	x	add	000
sw	010	x	add	000
beq	011	x	sub	001
bne	011	x	sub	001
slti	100	x	slt	010
andi	101	x	and	011
ori	110	x	or	100
li	111	x	add	000

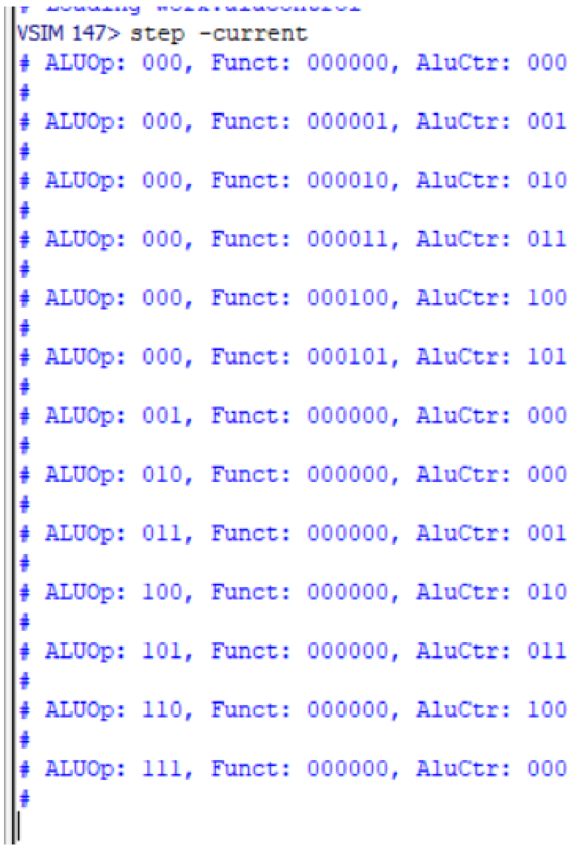
The truth table of AluControl is:



ALUCTR TRUTH TABLE

ALUCtr[2]	A2'.A1'.A0'.F2 + A2.A1.A0'
ALUCtr[1]	A2'.A1'.A0'.F1 + A2.A1'
ALUCtr[0]	A2'.A1'.A0'.F0 + A2'.A1.A0 + A2.A1'.A0

The testbench is like that. It works as exactly like my table.

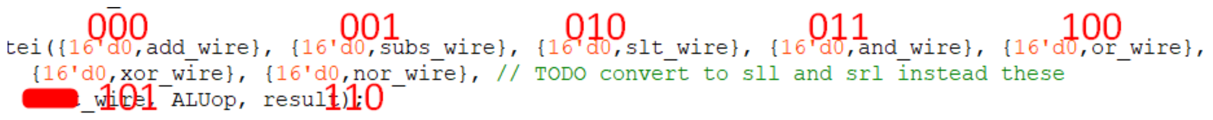


```
VSIM 147> step -current
# ALUOp: 000, Funct: 000000, AluCtr: 000
#
# ALUOp: 000, Funct: 000001, AluCtr: 001
#
# ALUOp: 000, Funct: 000010, AluCtr: 010
#
# ALUOp: 000, Funct: 000011, AluCtr: 011
#
# ALUOp: 000, Funct: 000100, AluCtr: 100
#
# ALUOp: 000, Funct: 000101, AluCtr: 101
#
# ALUOp: 001, Funct: 000000, AluCtr: 000
#
# ALUOp: 010, Funct: 000000, AluCtr: 000
#
# ALUOp: 011, Funct: 000000, AluCtr: 001
#
# ALUOp: 100, Funct: 000000, AluCtr: 010
#
# ALUOp: 101, Funct: 000000, AluCtr: 011
#
# ALUOp: 110, Funct: 000000, AluCtr: 100
#
# ALUOp: 111, Funct: 000000, AluCtr: 000
#
```

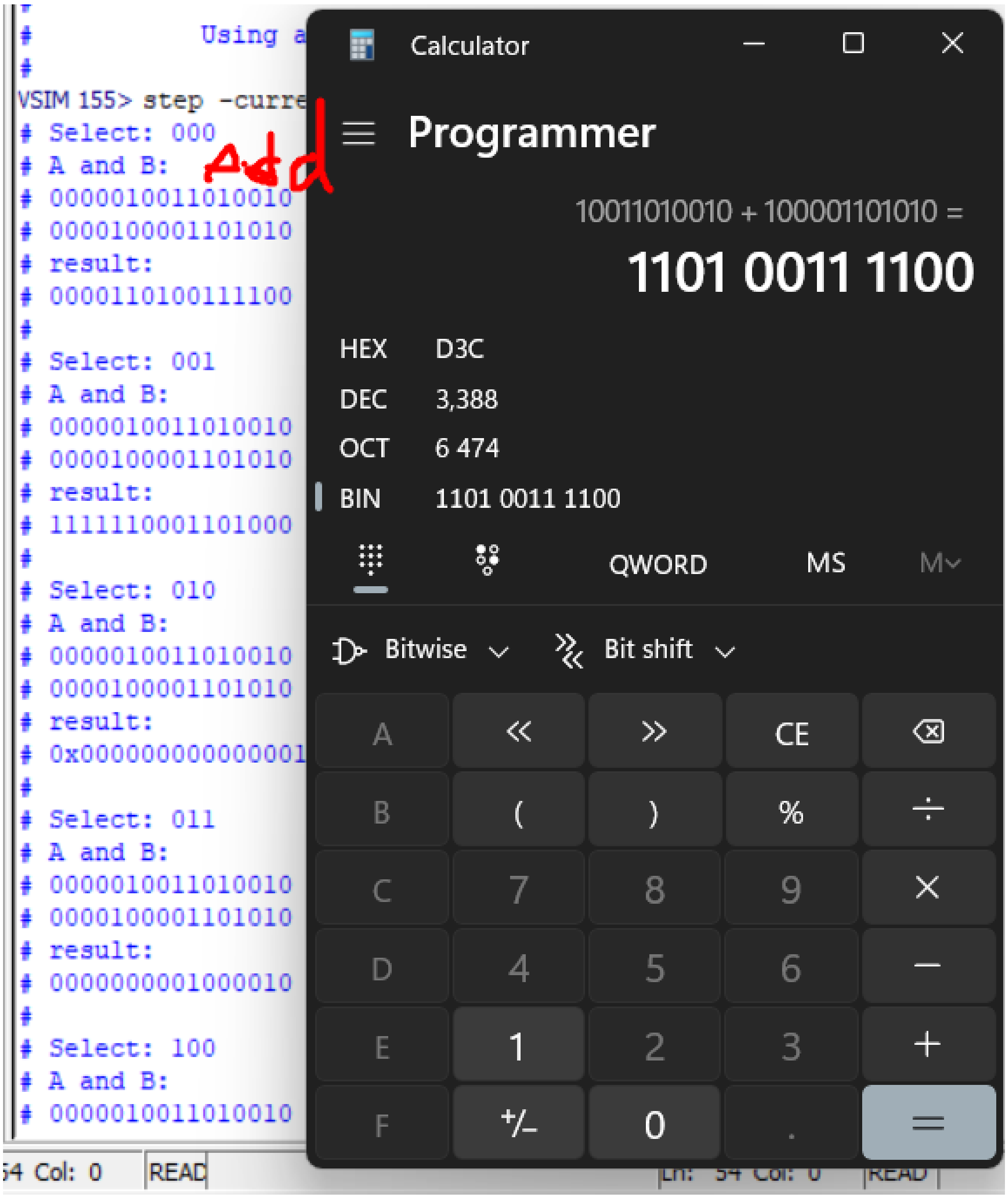
Alu16 Component

This part is similar to Homework 2. I changed the 32 bits to 16 bits. It Takes 2 number a and b, makes the operation and returns result.

SELECT INPUTS:



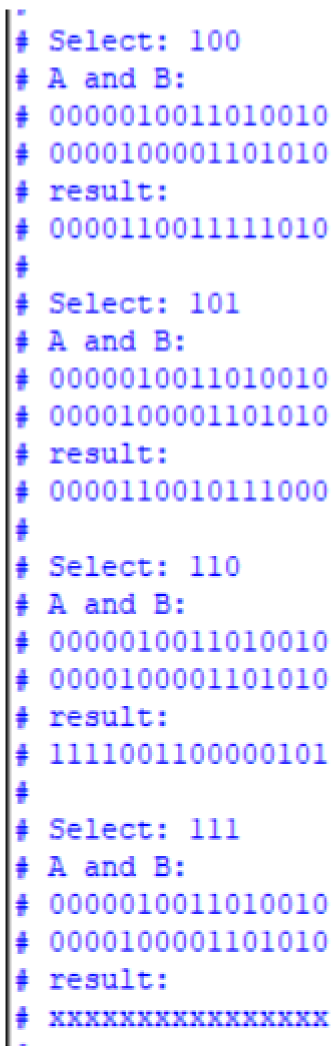
```
tei({16'd0,add_wire}, {16'd0,subs_wire}, {16'd0,slt_wire}, {16'd0,and_wire}, {16'd0,or_wire},
  {16'd0,xor_wire}, {16'd0,nor_wire}, // TODO convert to sll and srl instead these
 [redacted]_wire, ALUop, result);
```



```
#
#              Using a
#
VSIM 155> step -curre
# Select: 000
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 0000110100111100
#
# Select: 001
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 1111110001101000
#
# Select: 010
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 0x000000000000001
#
# Select: 011
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 0000000001000010
#
# Select: 100
# A and B:
# 0000010011010010
```

Add

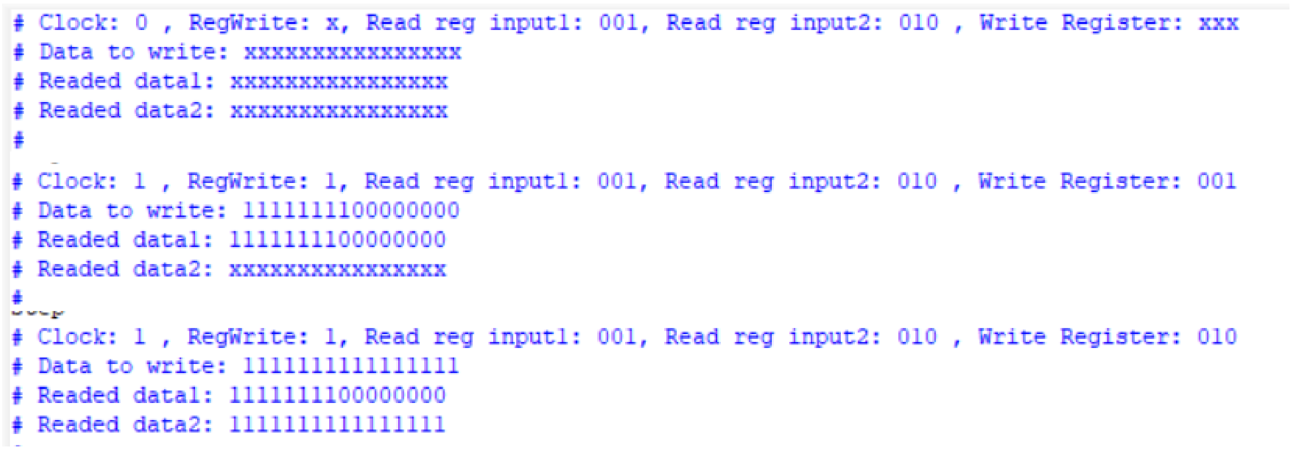
```
Calculator
Programmer
10011010010 + 100001101010 =
1101 0011 1100
HEX  D3C
DEC  3,388
OCT  6 474
BIN  1101 0011 1100
```



```
# Select: 100
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 0000110011111010
#
# Select: 101
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 0000110010111000
#
# Select: 110
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# 1111001100000101
#
# Select: 111
# A and B:
# 0000010011010010
# 0000100001101010
# result:
# xxxxxxxxxxxxxxxx
```

I didn't add mult as 16 bits so it shows x.

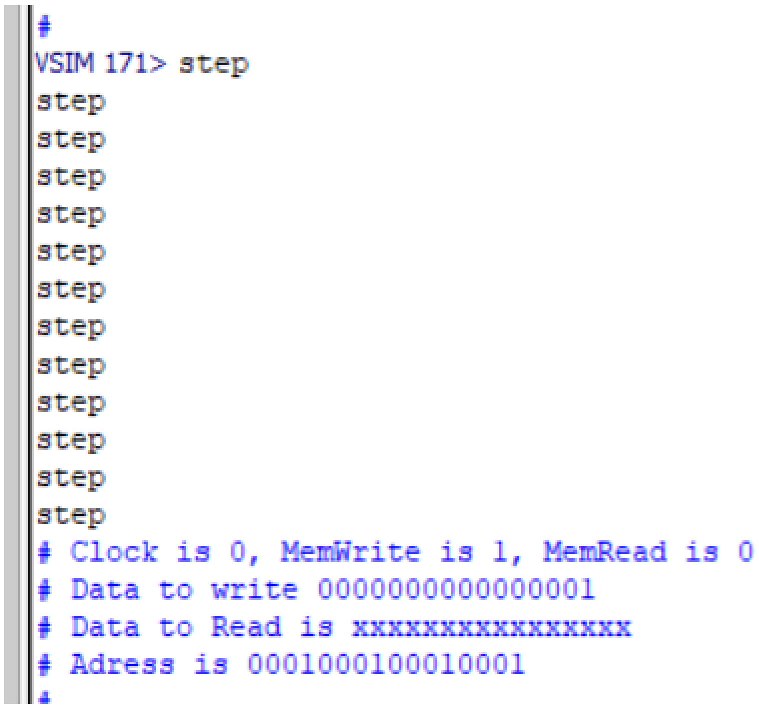
Register Component



```
# Clock: 0 , RegWrite: x, Read reg input1: 001, Read reg input2: 010 , Write Register: xxx
# Data to write: xxxxxxxxxxxxxxxx
# Readed data1: xxxxxxxxxxxxxxxx
# Readed data2: xxxxxxxxxxxxxxxx
#
# Clock: 1 , RegWrite: 1, Read reg input1: 001, Read reg input2: 010 , Write Register: 001
# Data to write: 1111111100000000
# Readed data1: 1111111100000000
# Readed data2: xxxxxxxxxxxxxxxx
#
# Clock: 1 , RegWrite: 1, Read reg input1: 001, Read reg input2: 010 , Write Register: 010
# Data to write: 1111111111111111
# Readed data1: 1111111100000000
# Readed data2: 1111111111111111
```

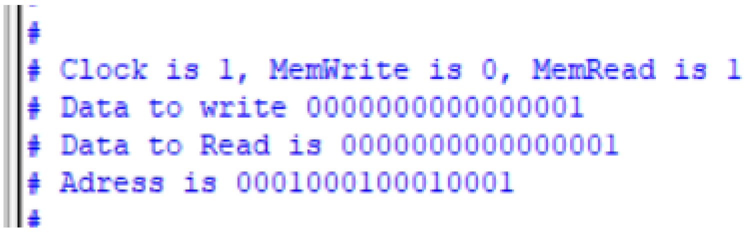
Registers testbench results are like that.

Data Memory Component



```
#
VSIM 171> step
step
step
step
step
step
step
step
step
step
step
step
step
# Clock is 0, MemWrite is 1, MemRead is 0
# Data to write 0000000000000001
# Data to Read is xxxxxxxxxxxxxxxx
# Adress is 0001000100010001
```

After setting memread to 1 and memwrite to 0,



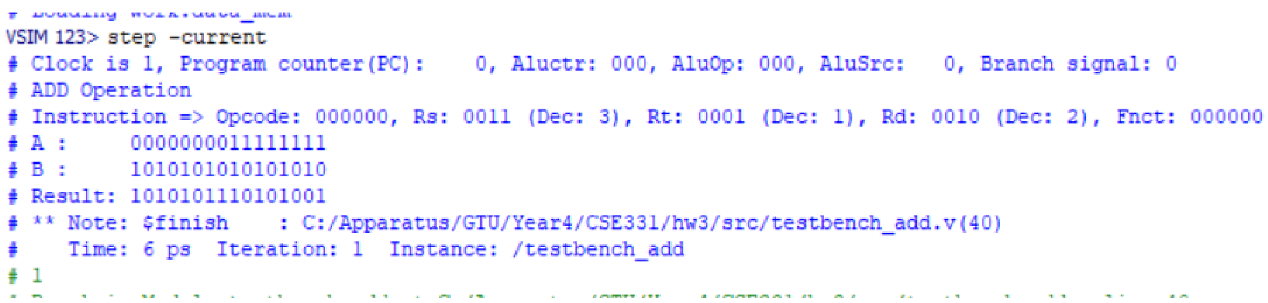
```
#
# Clock is 1, MemWrite is 0, MemRead is 1
# Data to write 0000000000000001
# Data to Read is 0000000000000001
# Adress is 0001000100010001
#
```

Note: I have 3 files for register, instructions and data memory. They need to be modelsim folder so that works correctly.

Mips16bits Module

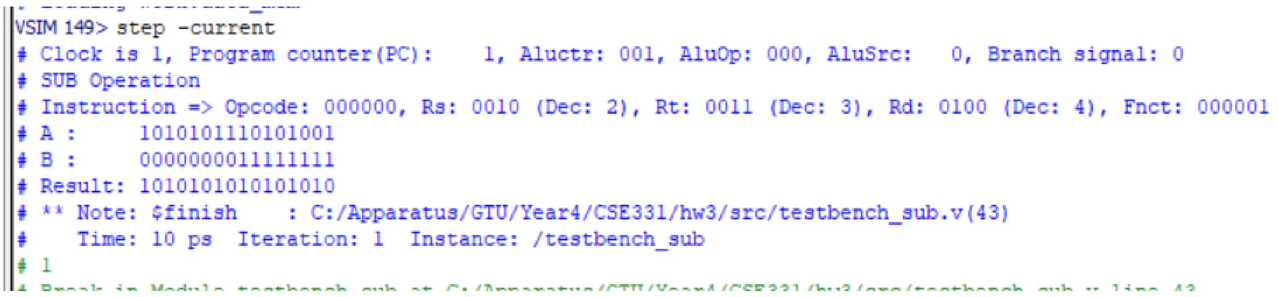
S

Add Testbench



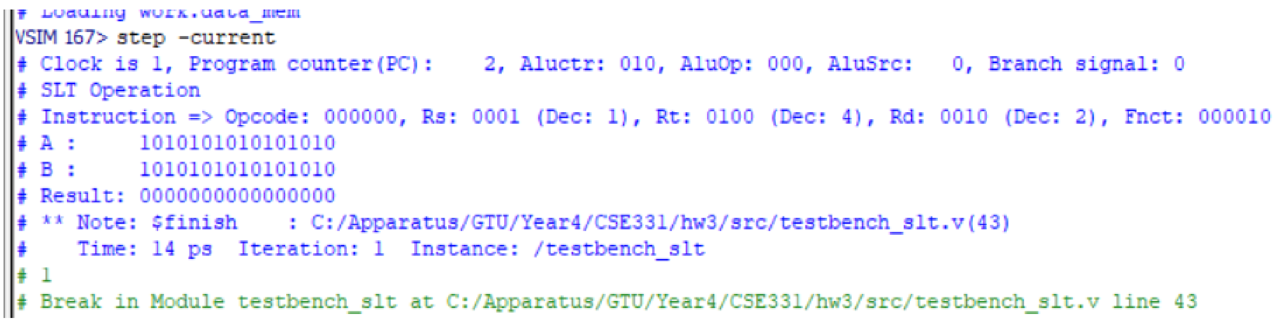
```
VSIM 123> step -current
# Clock is 1, Program counter(PC):     0, Aluctr: 000, AluOp: 000, AluSrc:    0, Branch signal: 0
# ADD Operation
# Instruction => Opcode: 000000, Rs: 0011 (Dec: 3), Rt: 0001 (Dec: 1), Rd: 0010 (Dec: 2), Fnct: 000000
# A :      0000000011111111
# B :      1010101010101010
# Result: 1010101110101001
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_add.v(40)
#    Time: 6 ps  Iteration: 1  Instance: /testbench_add
# 1
```

Sub Testbench



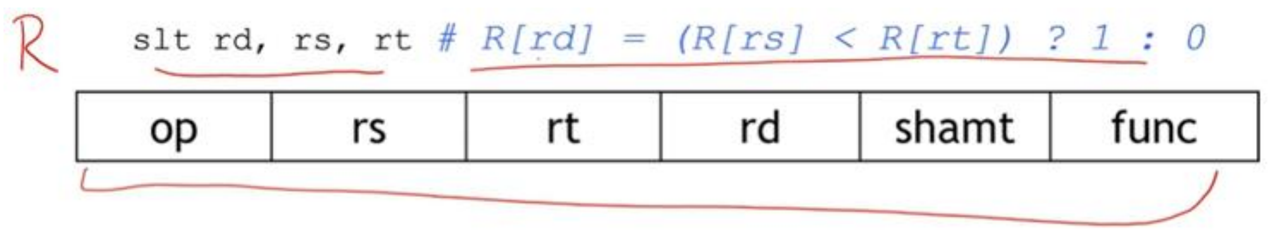
```
# Loading work.data_mem
VSIM 149> step -current
# Clock is 1, Program counter(PC):    1, Aluctr: 001, AluOp: 000, AluSrc:   0, Branch signal: 0
# SUB Operation
# Instruction => Opcode: 000000, Rs: 0010 (Dec: 2), Rt: 0011 (Dec: 3), Rd: 0100 (Dec: 4), Fnct: 000001
# A :     1010101110101001
# B :     0000000011111111
# Result: 1010101010101010
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_sub.v(43)
#    Time: 10 ps  Iteration: 1  Instance: /testbench_sub
# 1
# Break in Module testbench_sub at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_sub.v line 43
```

Slt Testbench

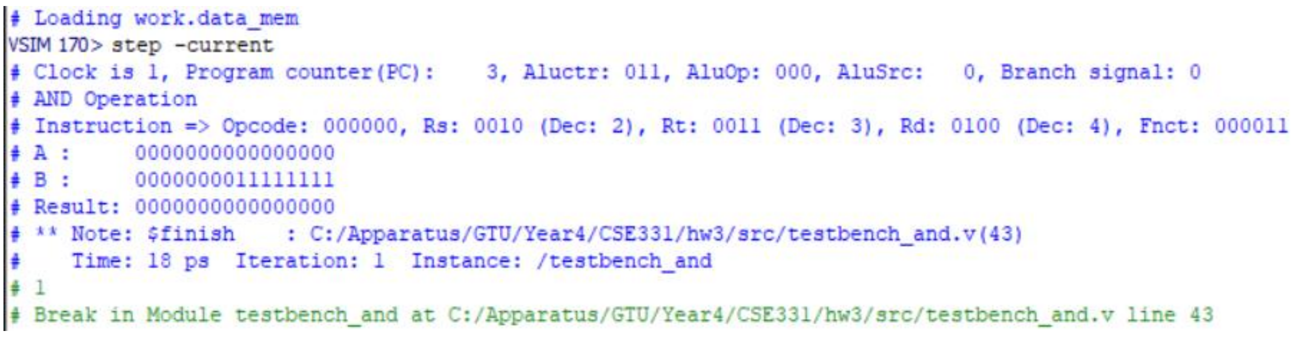


```
# Loading work.data_mem
VSIM 167> step -current
# Clock is 1, Program counter(PC):    2, Aluctr: 010, AluOp: 000, AluSrc:   0, Branch signal: 0
# SLT Operation
# Instruction => Opcode: 000000, Rs: 0001 (Dec: 1), Rt: 0100 (Dec: 4), Rd: 0010 (Dec: 2), Fnct: 000010
# A :     1010101010101010
# B :     1010101010101010
# Result: 0000000000000000
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_slt.v(43)
#    Time: 14 ps  Iteration: 1  Instance: /testbench_slt
# 1
# Break in Module testbench_slt at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_slt.v line 43
```

It has problem with printing Rt value but works correctly except than that.

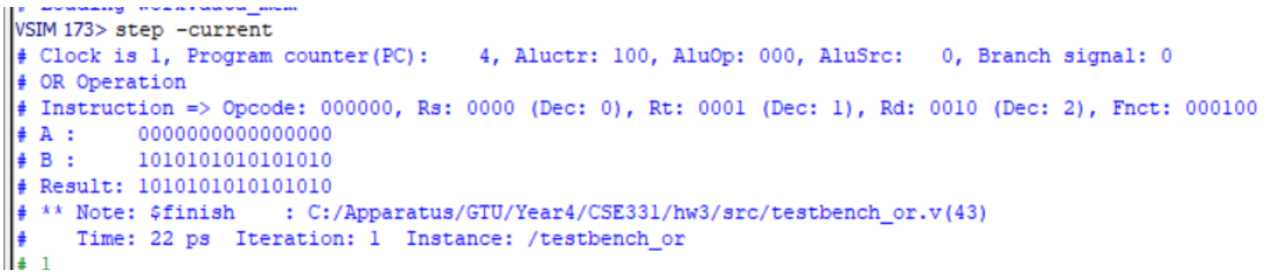


And Testbench



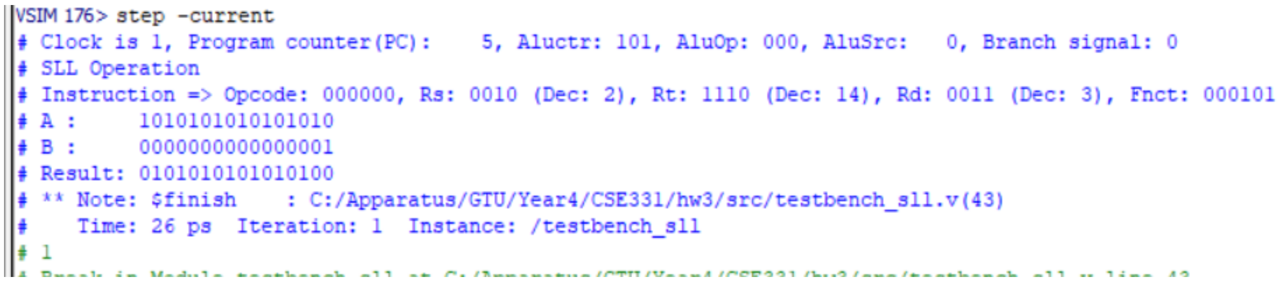
```
# Loading work.data_mem
VSIM 170> step -current
# Clock is 1, Program counter(PC):    3, Aluctr: 011, AluOp: 000, AluSrc:   0, Branch signal: 0
# AND Operation
# Instruction => Opcode: 000000, Rs: 0010 (Dec: 2), Rt: 0011 (Dec: 3), Rd: 0100 (Dec: 4), Fnct: 000011
# A :     0000000000000000
# B :     0000000011111111
# Result: 0000000000000000
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_and.v(43)
#    Time: 18 ps  Iteration: 1  Instance: /testbench_and
# 1
# Break in Module testbench_and at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_and.v line 43
```

Or Testbench



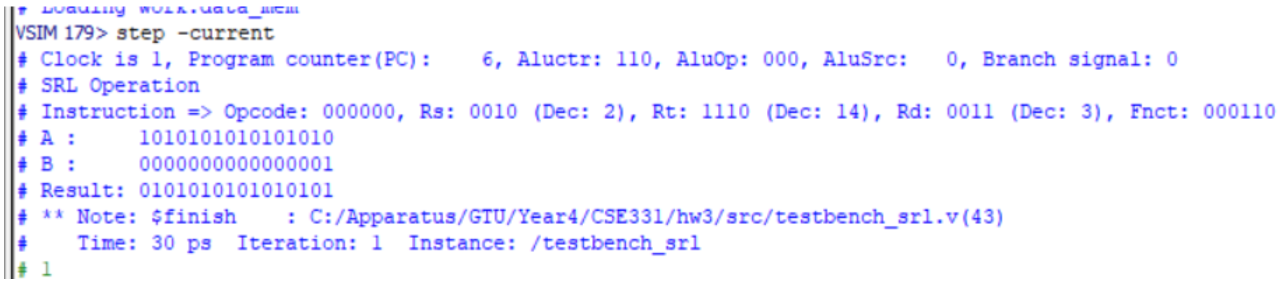
```
# Loading work.data_mem
VSIM 173> step -current
# Clock is 1, Program counter(PC):    4, Aluctr: 100, AluOp: 000, AluSrc:   0, Branch signal: 0
# OR Operation
# Instruction => Opcode: 000000, Rs: 0000 (Dec: 0), Rt: 0001 (Dec: 1), Rd: 0010 (Dec: 2), Fnct: 000100
# A :     0000000000000000
# B :     1010101010101010
# Result: 1010101010101010
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_or.v(43)
#    Time: 22 ps  Iteration: 1  Instance: /testbench_or
# 1
```

Sll Testbench



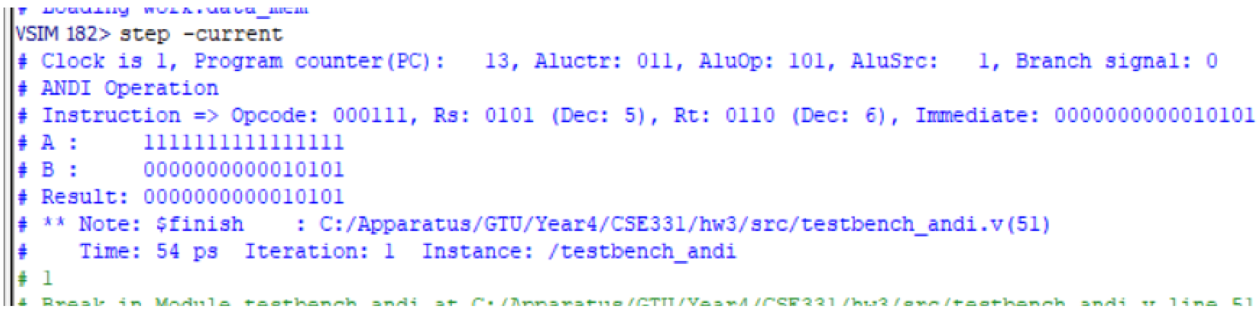
```
VSIM 176> step -current
# Clock is 1, Program counter(PC):    5, Aluctr: 101, AluOp: 000, AluSrc:   0, Branch signal: 0
# SLL Operation
# Instruction => Opcode: 000000, Rs: 0010 (Dec: 2), Rt: 1110 (Dec: 14), Rd: 0011 (Dec: 3), Fnct: 000101
# A :     1010101010101010
# B :     0000000000000001
# Result: 0101010101010100
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_sll.v(43)
#    Time: 26 ps  Iteration: 1  Instance: /testbench_sll
# 1
# Break in Module testbench_sll at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_sll.v line 43
```

Srl Testbench



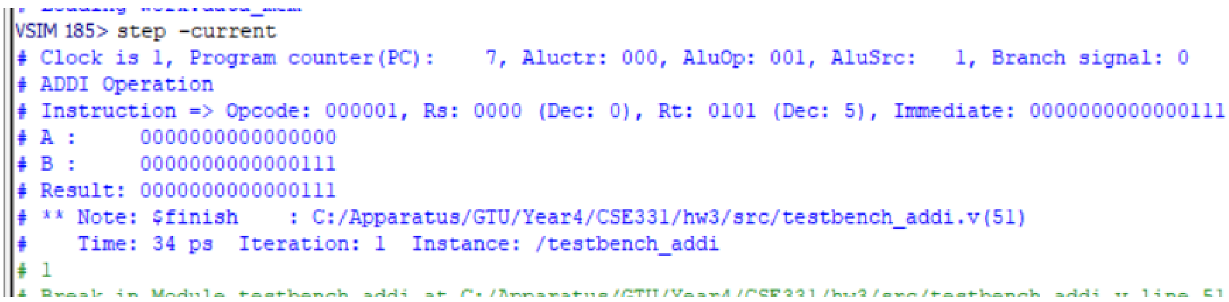
```
# Loading work.data_mem
VSIM 179> step -current
# Clock is 1, Program counter(PC):    6, Aluctr: 110, AluOp: 000, AluSrc:   0, Branch signal: 0
# SRL Operation
# Instruction => Opcode: 000000, Rs: 0010 (Dec: 2), Rt: 1110 (Dec: 14), Rd: 0011 (Dec: 3), Fnct: 000110
# A :     1010101010101010
# B :     0000000000000001
# Result: 0101010101010101
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_srl.v(43)
#    Time: 30 ps  Iteration: 1  Instance: /testbench_srl
# 1
```

Andi Testbench



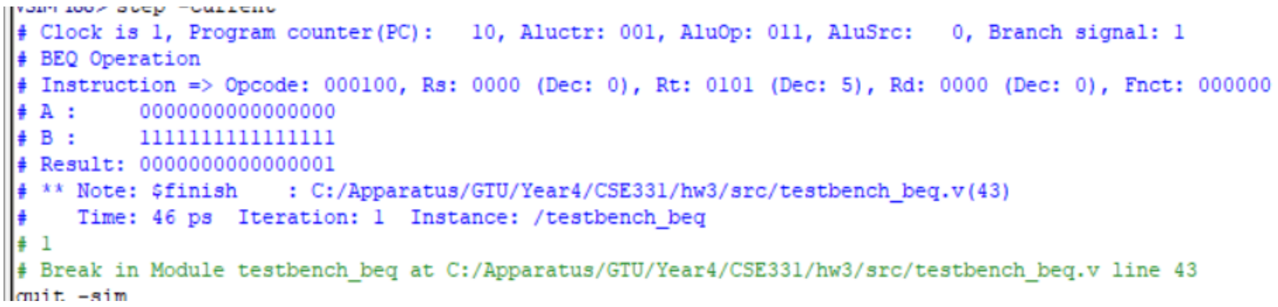
```
# Loading work.data_mem
VSIM 182> step -current
# Clock is 1, Program counter(PC):   13, Aluctr: 011, AluOp: 101, AluSrc:   1, Branch signal: 0
# ANDI Operation
# Instruction => Opcode: 000111, Rs: 0101 (Dec: 5), Rt: 0110 (Dec: 6), Immediate: 0000000000010101
# A :     1111111111111111
# B :     0000000000010101
# Result: 0000000000010101
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_andi.v(51)
#    Time: 54 ps  Iteration: 1  Instance: /testbench_andi
# 1
# Break in Module testbench_andi at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_andi.v line 51
```

Addi Testbench



```
# Loading work.data_mem
VSIM 185> step -current
# Clock is 1, Program counter(PC):    7, Aluctr: 000, AluOp: 001, AluSrc:   1, Branch signal: 0
# ADDI Operation
# Instruction => Opcode: 000001, Rs: 0000 (Dec: 0), Rt: 0101 (Dec: 5), Immediate: 0000000000000111
# A :     0000000000000000
# B :     0000000000000111
# Result: 0000000000000111
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_addi.v(51)
#    Time: 34 ps  Iteration: 1  Instance: /testbench_addi
# 1
# Break in Module testbench_addi at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_addi.v line 51
```

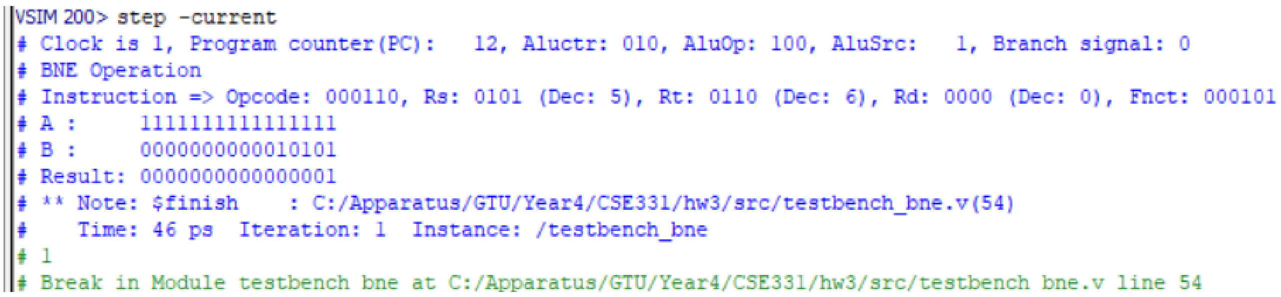
Beq Testbench



```
# Clock is 1, Program counter(PC):   10, Aluctr: 001, AluOp: 011, AluSrc:   0, Branch signal: 1
# BEQ Operation
# Instruction => Opcode: 000100, Rs: 0000 (Dec: 0), Rt: 0101 (Dec: 5), Rd: 0000 (Dec: 0), Fnct: 000000
# A :      0000000000000000
# B :      1111111111111111
# Result:  0000000000000001
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_beq.v(43)
#    Time: 46 ps  Iteration: 1  Instance: /testbench_beq
# 1
# Break in Module testbench_beq at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_beq.v line 43
quit -sim
```

It works correctly. To understand that use testbench_mips16bits module because there you can see it changes next program counter.

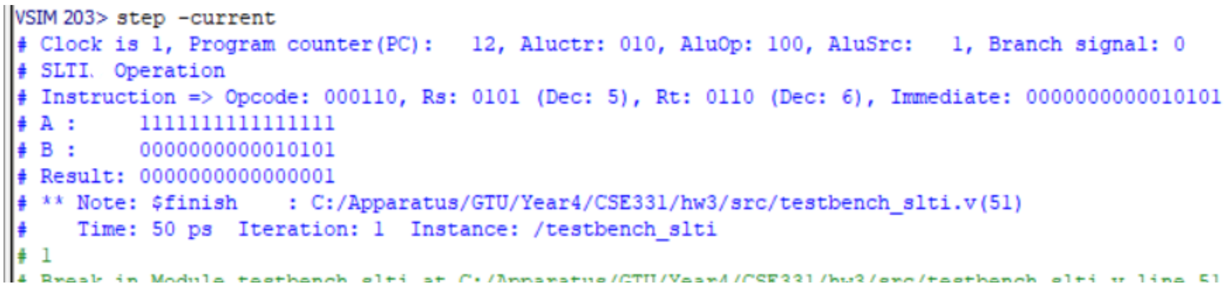
Bne Testbench



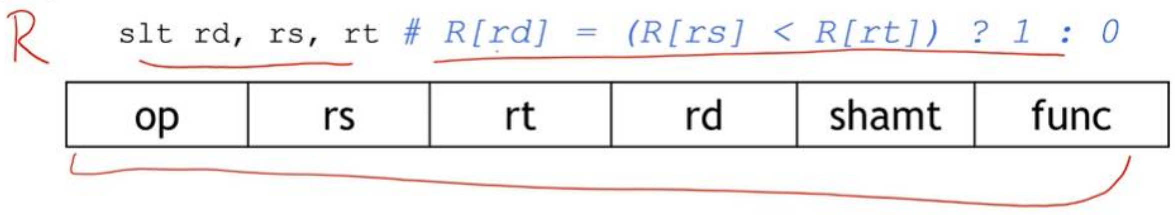
```
VSIM 200> step -current
# Clock is 1, Program counter(PC):   12, Aluctr: 010, AluOp: 100, AluSrc:   1, Branch signal: 0
# BNE Operation
# Instruction => Opcode: 000110, Rs: 0101 (Dec: 5), Rt: 0110 (Dec: 6), Rd: 0000 (Dec: 0), Fnct: 000101
# A :      1111111111111111
# B :      0000000000010101
# Result:  0000000000000001
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_bne.v(54)
#    Time: 46 ps  Iteration: 1  Instance: /testbench_bne
# 1
# Break in Module testbench bne at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench bne.v line 54
```

To understand that use testbench_mips16bits module because there you can see it changes next program counter.

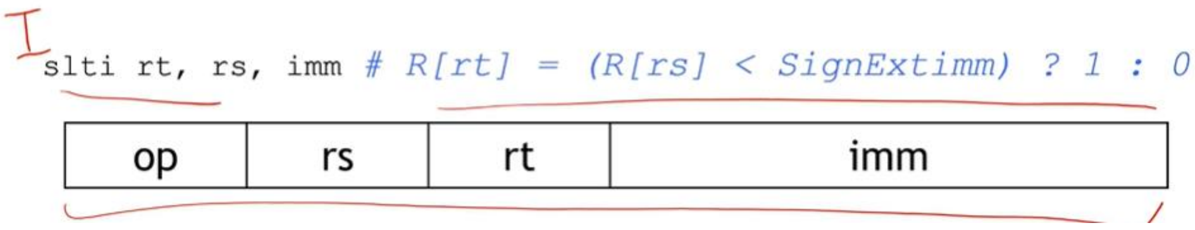
Slti Testbench



```
VSIM 203> step -current
# Clock is 1, Program counter(PC):   12, Aluctr: 010, AluOp: 100, AluSrc:   1, Branch signal: 0
# SLTI. Operation
# Instruction => Opcode: 000110, Rs: 0101 (Dec: 5), Rt: 0110 (Dec: 6), Immediate: 0000000000010101
# A :      1111111111111111
# B :      0000000000010101
# Result:  0000000000000001
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_slti.v(51)
#    Time: 50 ps  Iteration: 1  Instance: /testbench_slti
# 1
# Break in Module testbench slti at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench slti.v line 51
```

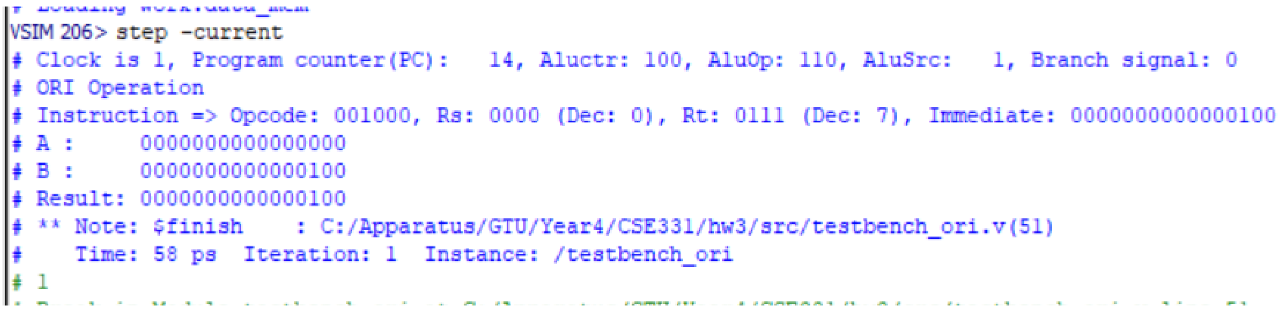


```
slt rd, rs, rt # R[rd] = (R[rs] < R[rt]) ? 1 : 0
```



```
slti rt, rs, imm # R[rt] = (R[rs] < SignExtimm) ? 1 : 0
```

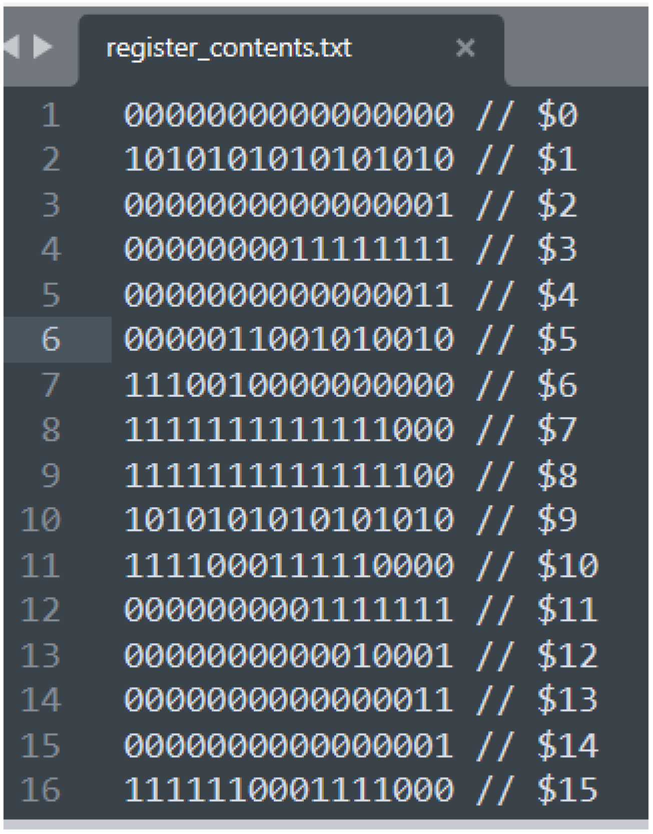
Ori Testbench



```
VSIM 206> step -current
# Clock is 1, Program counter(PC):   14, Aluctr: 100, AluOp: 110, AluSrc:    1, Branch signal: 0
# ORI Operation
# Instruction => Opcode: 001000, Rs: 0000 (Dec: 0), Rt: 0111 (Dec: 7), Immediate: 0000000000000100
# A :      0000000000000000
# B :      0000000000000100
# Result: 0000000000000100
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_ori.v(51)
#    Time: 58 ps  Iteration: 1  Instance: /testbench_ori
# 1
```

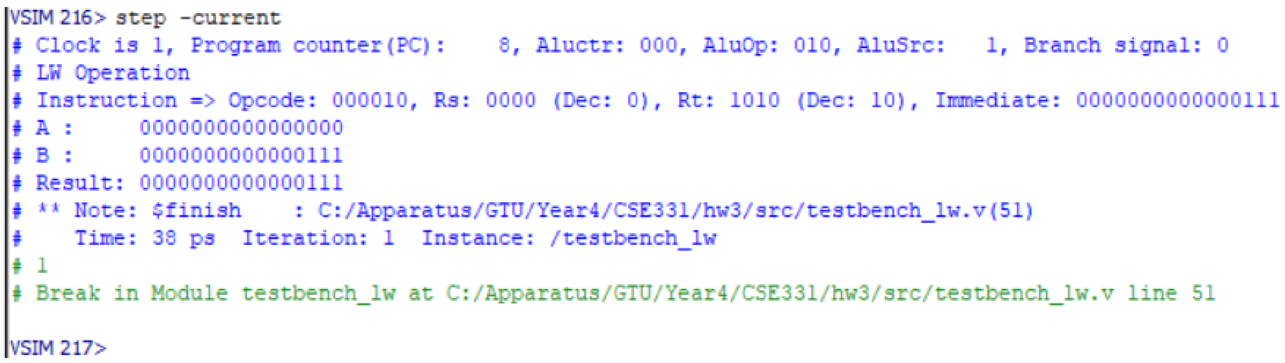
Lw Testbench

Before test, the registers:



```
0000000000000000 // $0
1010101010101010 // $1
0000000000000001 // $2
0000000011111111 // $3
0000000000000011 // $4
0000011001010010 // $5
1110010000000000 // $6
1111111111111000 // $7
1111111111111100 // $8
1010101010101010 // $9
1111000111110000 // $10
0000000001111111 // $11
0000000000010001 // $12
0000000000000011 // $13
0000000000000001 // $14
1111110001111000 // $15
```

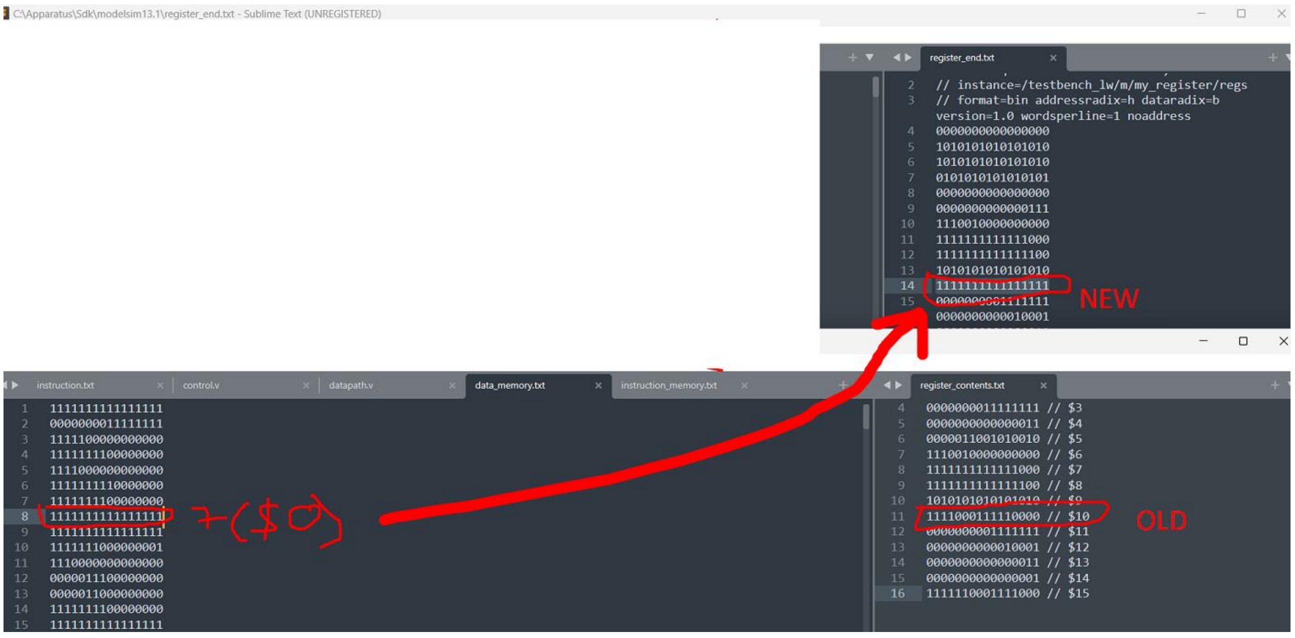
Test



```
VSIM 216> step -current
# Clock is 1, Program counter(PC):    8, Aluctr: 000, AluOp: 010, AluSrc:    1, Branch signal: 0
# LW Operation
# Instruction => Opcode: 000010, Rs: 0000 (Dec: 0), Rt: 1010 (Dec: 10), Immediate: 0000000000000111
# A :      0000000000000000
# B :      0000000000000111
# Result: 0000000000000111
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_lw.v(51)
#    Time: 38 ps  Iteration: 1  Instance: /testbench_lw
# 1
# Break in Module testbench_lw at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_lw.v line 51

VSIM 217>
```

After test, the registers



C:\Apparatus\Sdk\modelsim13.1\register_end.txt - Sublime Text (UNREGISTERED)

register_end.txt

```
// instance=/testbench_lw/m/my_register/regs
// format=bin addressradix=h dataradix=b
version=1.0 wordsperline=1 noaddress
0000000000000000
1010101010101010
1010101010101010
0101010101010101
0000000000000000
0000000000000111
1110010000000000
1111111111111000
1111111111111100
1010101010101010
1111111111111111
0000000001111111
0000000000010001
```

instance.txt · control.v · datapath.v · data_memory.txt · instruction_memory.txt

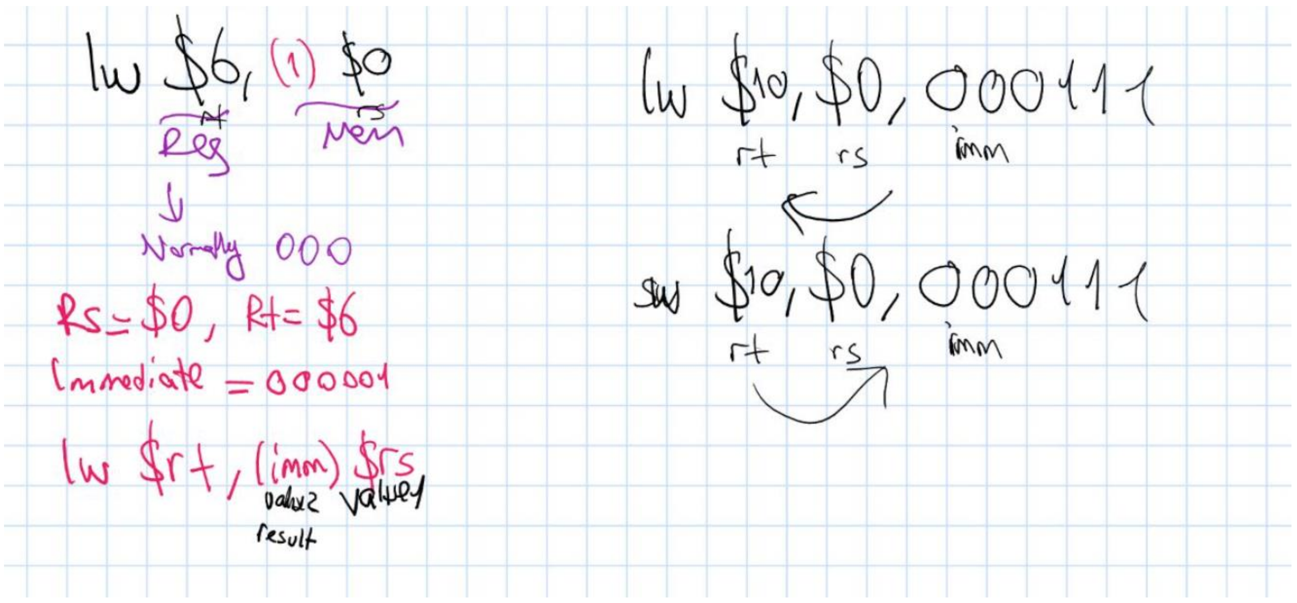
data_memory.txt

```
1111111111111111
0000000011111111
1111100000000000
1111111100000000
1111000000000000
1111111110000000
1111111100000000
1111111111111111
1111111111111111
1111111000000001
1110000000000000
0000011100000000
0000011000000000
1111111100000000
1111111111111111
```

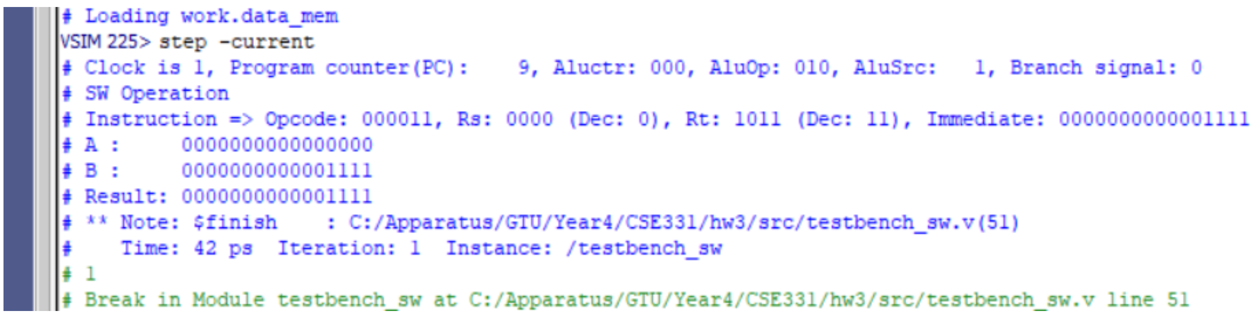
register_contents.txt

```
0000000011111111 // $3
0000000000000011 // $4
0000011001010010 // $5
1110010000000000 // $6
1111111111111000 // $7
1111111111111100 // $8
1010101010101010 // $9
1111000111110000 // $10
0000000001111111 // $11
0000000000010001 // $12
0000000000000011 // $13
0000000000000001 // $14
1111110001111000 // $15
```

Works fine.



Sw Testbench



```
# Loading work.data_mem
VSIM 225> step -current
# Clock is 1, Program counter(PC):     9, Aluctr: 000, AluOp: 010, AluSrc:   1, Branch signal: 0
# SW Operation
# Instruction => Opcode: 000011, Rs: 0000 (Dec: 0), Rt: 1011 (Dec: 11), Immediate: 0000000000001111
# A :      0000000000000000
# B :      0000000000001111
# Result: 0000000000001111
# ** Note: $finish    : C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_sw.v(51)
#    Time: 42 ps  Iteration: 1  Instance: /testbench_sw
# 1
# Break in Module testbench_sw at C:/Apparatus/GTU/Year4/CSE331/hw3/src/testbench_sw.v line 51
```

Unfortunately I couldn’t finish this.

Li

Since time limitation, I couldn’t implement load immediate exactly.

J Type Instructions

I couldn’t implement j type instructions.