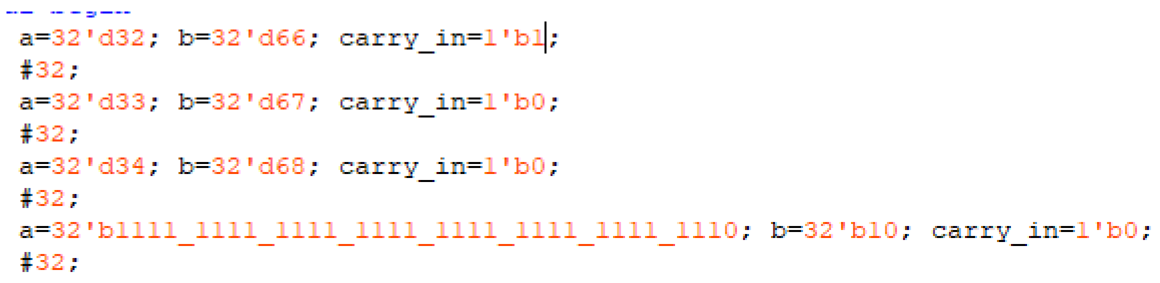
CSE 331 Computer Organization HW4 Report

Yakup Talha Yolcu

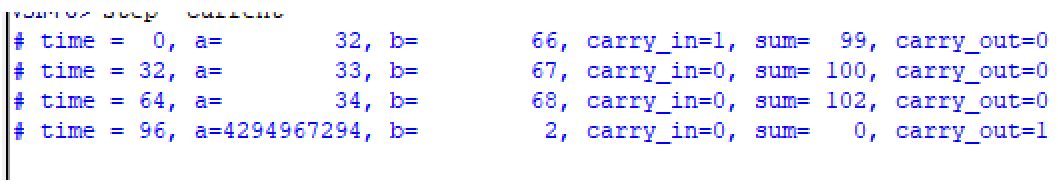
1801042609

1. 32 bit full adder test



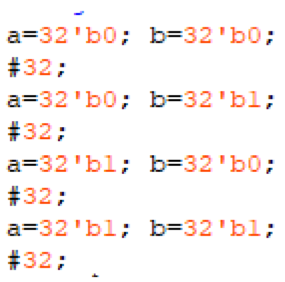
```
 a=32'd32; b=32'd66; carry_in=1'b1;
 #32;
 a=32'd33; b=32'd67; carry_in=1'b0;
 #32;
 a=32'd34; b=32'd68; carry_in=1'b0;
 #32;
 a=32'b1111_1111_1111_1111_1111_1111_1111_1110; b=32'b10; carry_in=1'b0;
 #32;
```

First test, 32 + 66 + 1 = 99 => a=32,b=66,carry in = 1
Second test, 33 + 67 = 100 there is no carry in
Third test 34 + 68 = 102, there is no carry in
Fourth test a=2^32-2 b= 2, I sum them up to get carry out and I get carry out.



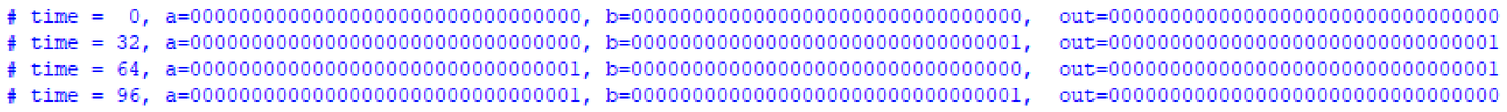
```
# time =  0, a=        32, b=        66, carry_in=1, sum=  99, carry_out=0
# time = 32, a=        33, b=        67, carry_in=0, sum= 100, carry_out=0
# time = 64, a=        34, b=        68, carry_in=0, sum= 102, carry_out=0
# time = 96, a=4294967294, b=         2, carry_in=0, sum=   0, carry_out=1
```

2. 32 bit xor test



```
a=32'b0; b=32'b0;
#32;
a=32'b0; b=32'b1;
#32;
a=32'b1; b=32'b0;
#32;
a=32'b1; b=32'b1;
#32;
```

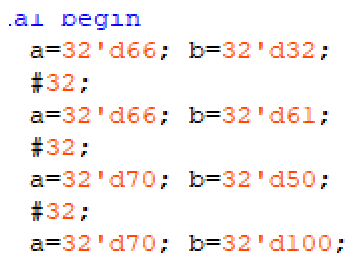
I just xor the 0 and 1 as 1 bits



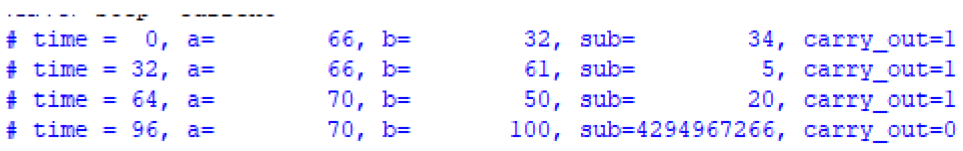
```
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=00000000000000000000000000000001
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=00000000000000000000000000000000
```

3. 32 bit full subtractor test

66-32=34 66-61=5 70-50=20 70-100=-30 as decimal

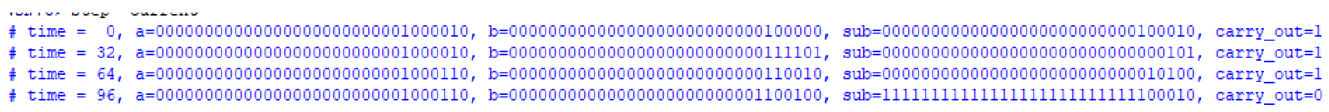


```
.al begin
 a=32'd66; b=32'd32;
 #32;
 a=32'd66; b=32'd61;
 #32;
 a=32'd70; b=32'd50;
 #32;
 a=32'd70; b=32'd100;
```



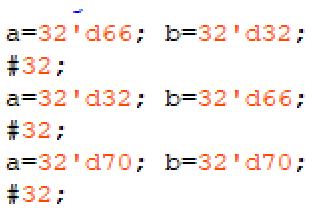
```
# time =  0, a=        66, b=        32, sub=        34, carry_out=1
# time = 32, a=        66, b=        61, sub=         5, carry_out=1
# time = 64, a=        70, b=        50, sub=        20, carry_out=1
# time = 96, a=        70, b=       100, sub=4294967266, carry_out=0
```

Same result as binary format, 11111111111111111111111111100010 is binary of -30 in signed 2's complement (70-100 case)



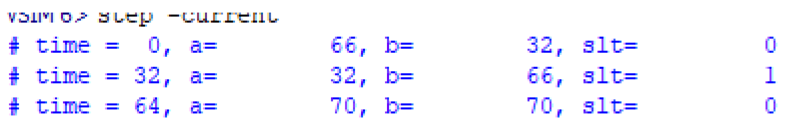
```
# time =  0, a=00000000000000000000000001000010, b=00000000000000000000000000100000, sub=00000000000000000000000000100010, carry_out=1
# time = 32, a=00000000000000000000000001000010, b=00000000000000000000000000111101, sub=00000000000000000000000000000101, carry_out=1
# time = 64, a=00000000000000000000000001000110, b=00000000000000000000000000110010, sub=00000000000000000000000000010100, carry_out=1
# time = 96, a=00000000000000000000000001000110, b=00000000000000000000000001100100, sub=11111111111111111111111111100010, carry_out=0
```

4. Set less than test

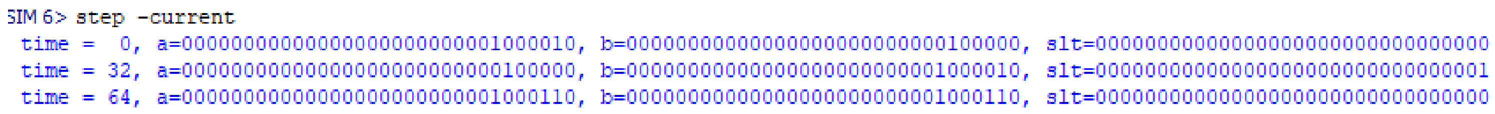


```
a=32'd66; b=32'd32;
#32;
a=32'd32; b=32'd66;
#32;
a=32'd70; b=32'd70;
#32;
```

Outputs as binary and decimal format

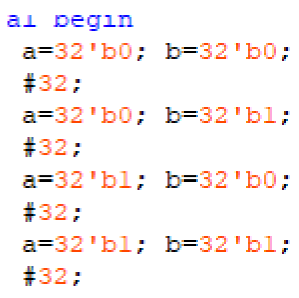


```
VSIM 6> step -current
# time =  0, a=        66, b=        32, slt=         0
# time = 32, a=        32, b=        66, slt=         1
# time = 64, a=        70, b=        70, slt=         0
```



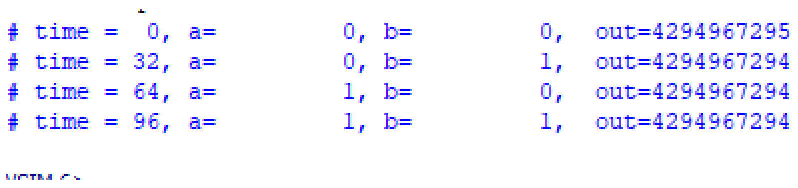
```
SIM 6> step -current
 time =  0, a=00000000000000000000000001000010, b=00000000000000000000000000100000, slt=00000000000000000000000000000000
 time = 32, a=00000000000000000000000000100000, b=00000000000000000000000001000010, slt=00000000000000000000000000000001
 time = 64, a=00000000000000000000000001000110, b=00000000000000000000000001000110, slt=00000000000000000000000000000000
```

5. Nor 32 bit test

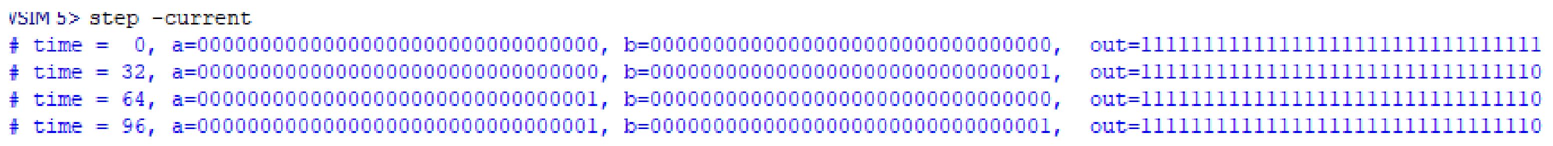


```
al begin
 a=32'b0; b=32'b0;
 #32;
 a=32'b0; b=32'b1;
 #32;
 a=32'b1; b=32'b0;
 #32;
 a=32'b1; b=32'b1;
 #32;
```

Results as binary and decimal format

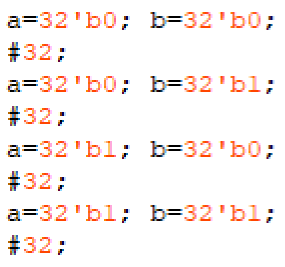


```
# time =  0, a=         0, b=         0,  out=4294967295
# time = 32, a=         0, b=         1,  out=4294967294
# time = 64, a=         1, b=         0,  out=4294967294
# time = 96, a=         1, b=         1,  out=4294967294
```

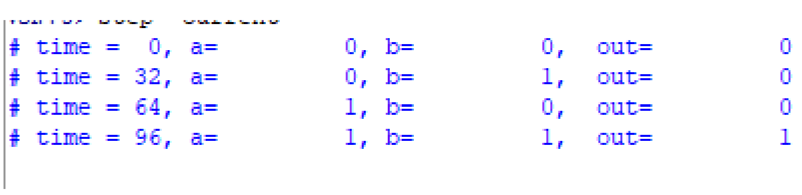


```
VSIM 5> step -current
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=11111111111111111111111111111111
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=11111111111111111111111111111110
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=11111111111111111111111111111110
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=11111111111111111111111111111110
```

6. 32 bit and test

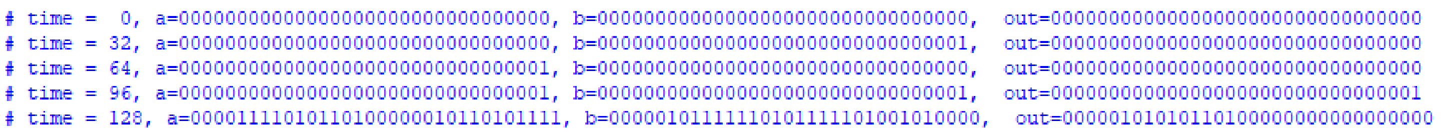


```
a=32'b0; b=32'b0;
#32;
a=32'b0; b=32'b1;
#32;
a=32'b1; b=32'b0;
#32;
a=32'b1; b=32'b1;
#32;
```



```
# time =  0, a=         0, b=         0,  out=         0
# time = 32, a=         0, b=         1,  out=         0
# time = 64, a=         1, b=         0,  out=         0
# time = 96, a=         1, b=         1,  out=         1
```

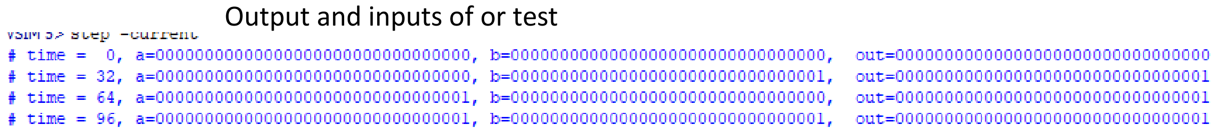
Outputs as binary format



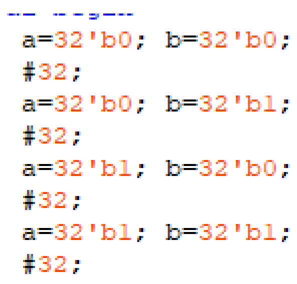
```
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=00000000000000000000000000000000
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
# time = 128, a=00001111010110100000010110101111, b=00001011111101011111010010100000,  out=00001010101101000000000000000000
```

7. 32 bit or test

Output and inputs of or test

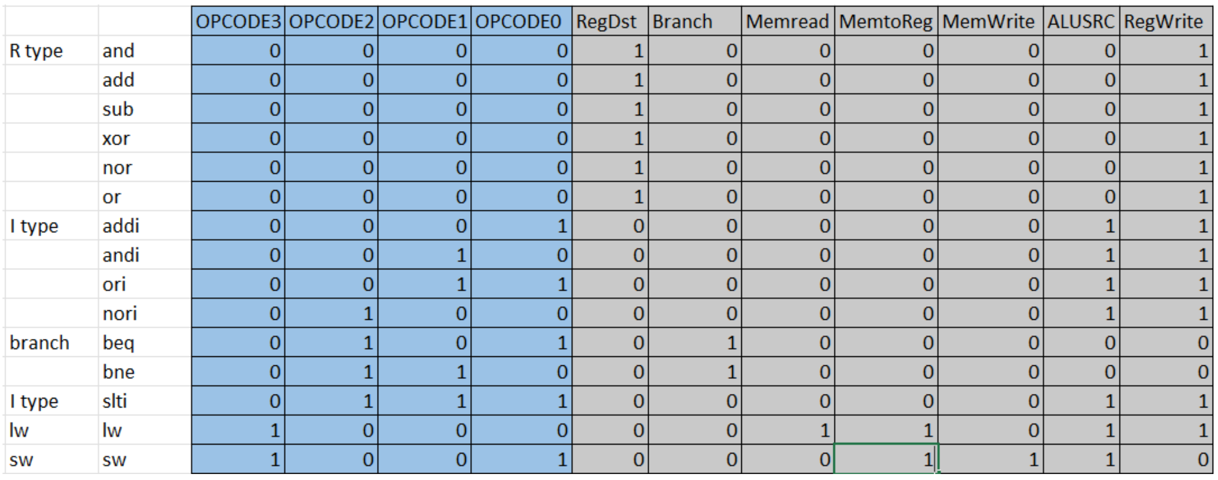


```
VSIM 5> step -current
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=00000000000000000000000000000001
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
```



```
a=32'b0; b=32'b0;
#32;
a=32'b0; b=32'b1;
#32;
a=32'b1; b=32'b0;
#32;
a=32'b1; b=32'b1;
#32;
```

8. Control Unit Signals and Outputs



		OPCODE3	OPCODE2	OPCODE1	OPCODE0	RegDst	Branch	Memread	MemtoReg	MemWrite	ALUSRC	RegWrite
R type	and	0	0	0	0	1	0	0	0	0	0	1
	add	0	0	0	0	1	0	0	0	0	0	1
	sub	0	0	0	0	1	0	0	0	0	0	1
	xor	0	0	0	0	1	0	0	0	0	0	1
	nor	0	0	0	0	1	0	0	0	0	0	1
	or	0	0	0	0	1	0	0	0	0	0	1
I type	addi	0	0	0	1	0	0	0	0	0	1	1
	andi	0	0	1	0	0	0	0	0	0	1	1
	ori	0	0	1	1	0	0	0	0	0	1	1
	nori	0	1	0	0	0	0	0	0	0	1	1
branch	beq	0	1	0	1	0	1	0	0	0	0	0
	bne	0	1	1	0	0	1	0	0	0	0	0
I type	slti	0	1	1	1	0	0	0	0	0	1	1
lw	lw	1	0	0	0	0	0	1	1	0	1	1
sw	sw	1	0	0	1	0	0	0	1	1	1	0

A=OPCODE3

B=OPCODE2

C=OPCODE1

D=OPCODE0

regdst= A'B'C'D'

branch=A'BC'D+ A'BCD'

memread=AB'C'D'

memtoreg=AB'C'

ALUOP2=A'CD + A'BD'

ALUOP1=A'BD + A'BC + A'B'C'D'

ALUOP0=B'C' + A'CD

memwrite=AB'C'D

ALUSRC = A'B'D + A'B'C + A'CD + AB'C' + A'BC'D'

regwrite=A'B' + A'C'D' + B'C'D' + A'CD

9. ALUOP Signals and Equations

		OPCODE3	OPCODE2	OPCODE1	OPCODE0	ALUOP2	ALUOP1	ALUOP0
R type	and	0	0	0	0	0	1	1
	add	0	0	0	0	0	1	1
	sub	0	0	0	0	0	1	1
	xor	0	0	0	0	0	1	1
	nor	0	0	0	0	0	1	1
	or	0	0	0	0	0	1	1
I type	addi	0	0	0	1	0	0	1
	andi	0	0	1	0	0	0	0
	ori	0	0	1	1	1	0	1
	nori	0	1	0	0	1	0	0
branch	beq	0	1	0	1	0	1	0
	bne	0	1	1	0	1	1	0
I type	slti	0	1	1	1	1	1	1
lw	lw	1	0	0	0	0	0	1
sw	sw	1	0	0	1	0	0	1

ALUOP2=A'CD + A'BD'

ALUOP1=A'BD + A'BC + A'B'C'D'

ALUOP0=B'C' + A'CD

A=OPCODE3

B=OPCODE2

C=OPCODE1

D=OPCODE0

I gave same ALUOP code to all R type instructions because I am gonna detect the ALU operation from their function code.

Lw and sw instructions uses the adding immediate alu operation.

10. ALUCTRL Table and Equations

		ALUOP2	ALUOP1	ALUOP0	FUNC2	FUN1	FUNC0	ALUCTRL2	ALUCTRL1	ALUCTRL0
R type	and	0	1	1	0	0	0	0	0	0
	add	0	1	1	0	0	1	0	0	1
	sub	0	1	1	0	1	0	0	1	0
	xor	0	1	1	0	1	1	0	1	1
	nor	0	1	1	1	0	0	1	0	0
	or	0	1	1	1	0	1	1	0	1
I type	addi	0	0	1	x	x	x	0	0	1
	andi	0	0	0	x	x	x	0	0	0
	ori	1	0	1	x	x	x	1	0	1
	nori	1	0	0	x	x	x	1	0	0
branch	beq	0	1	0	x	x	x	0	1	0
	bne	1	1	0	x	x	x	1	1	0
I type	slti	1	1	1	x	x	x	1	1	1
lw	lw	0	0	1	x	x	x	0	0	1
sw	sw	0	0	1	x	x	x	0	0	1

For R types I checked the function fields. Because of their ALUOP's are same, their Function fields directly goes to ALUCTRL output

For I types and branch,lw,sw we don't use function field. ALUOP directly goes to ALUCTRL output.

ALUCTRL signals goes to ALU select

ALUCTRL2=A + BCDE'

ALUCTRL1= BC' + AB + BD'E

ALUCTRL0=B'C + AC + CD'F + CE'F

0	0	0	and
0	0	1	add
0	1	0	sub
0	0	1	xor
1	0	0	nor
1	1	1	or
1	1	0	bneq
1	0	1	slt

A=ALUOP2

B=ALUOP1

C=ALUOP0

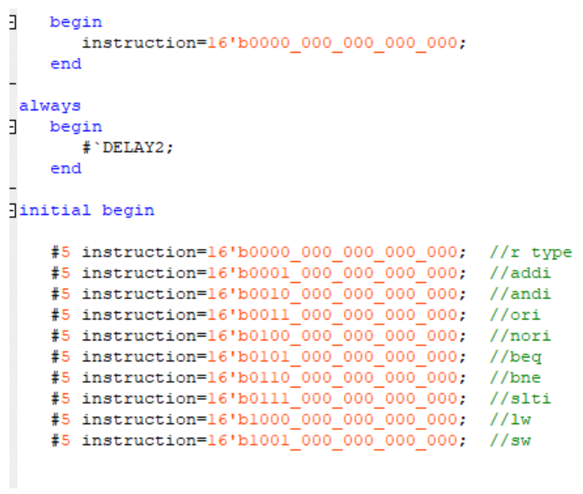
D=FUNC2

E=FUNC1

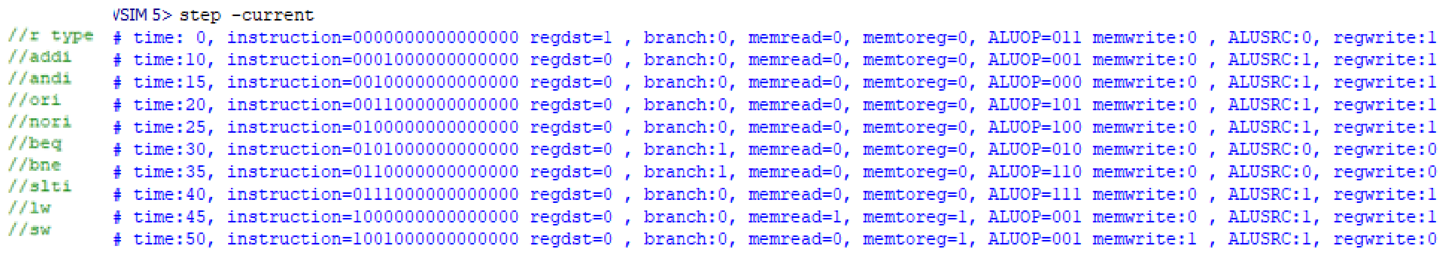
F=FUNC0

11. Control unit Testbench

All R type instructions has the same Control signals. Even the ALUOP

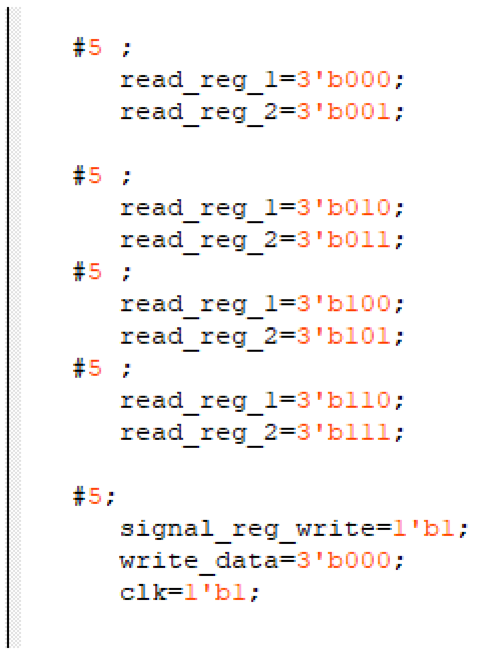


```
    begin
       instruction=16'b0000_000_000_000_000;
    end

 always
    begin
       #`DELAY2;
    end

initial begin

    #5 instruction=16'b0000_000_000_000_000;  //r type
    #5 instruction=16'b0001_000_000_000_000;  //addi
    #5 instruction=16'b0010_000_000_000_000;  //andi
    #5 instruction=16'b0011_000_000_000_000;  //ori
    #5 instruction=16'b0100_000_000_000_000;  //nori
    #5 instruction=16'b0101_000_000_000_000;  //beq
    #5 instruction=16'b0110_000_000_000_000;  //bne
    #5 instruction=16'b0111_000_000_000_000;  //slti
    #5 instruction=16'b1000_000_000_000_000;  //lw
    #5 instruction=16'b1001_000_000_000_000;  //sw
```



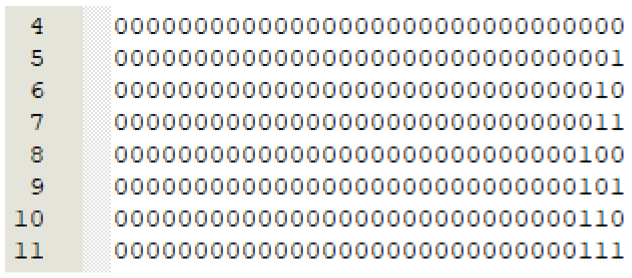
```
          VSIM 5> step -current
//r type  # time: 0, instruction=0000000000000000 regdst=1 , branch:0, memread=0, memtoreg=0, ALUOP=011 memwrite:0 , ALUSRC:0, regwrite:1
//addi    # time:10, instruction=0001000000000000 regdst=0 , branch:0, memread=0, memtoreg=0, ALUOP=001 memwrite:0 , ALUSRC:1, regwrite:1
//andi    # time:15, instruction=0010000000000000 regdst=0 , branch:0, memread=0, memtoreg=0, ALUOP=000 memwrite:0 , ALUSRC:1, regwrite:1
//ori     # time:20, instruction=0011000000000000 regdst=0 , branch:0, memread=0, memtoreg=0, ALUOP=101 memwrite:0 , ALUSRC:1, regwrite:1
//nori    # time:25, instruction=0100000000000000 regdst=0 , branch:0, memread=0, memtoreg=0, ALUOP=100 memwrite:0 , ALUSRC:1, regwrite:1
//beq     # time:30, instruction=0101000000000000 regdst=0 , branch:1, memread=0, memtoreg=0, ALUOP=010 memwrite:0 , ALUSRC:0, regwrite:0
//bne     # time:35, instruction=0110000000000000 regdst=0 , branch:1, memread=0, memtoreg=0, ALUOP=110 memwrite:0 , ALUSRC:0, regwrite:0
//slti    # time:40, instruction=0111000000000000 regdst=0 , branch:0, memread=0, memtoreg=0, ALUOP=111 memwrite:0 , ALUSRC:1, regwrite:1
//lw      # time:45, instruction=1000000000000000 regdst=0 , branch:0, memread=1, memtoreg=1, ALUOP=001 memwrite:0 , ALUSRC:1, regwrite:1
//sw      # time:50, instruction=1001000000000000 regdst=0 , branch:0, memread=0, memtoreg=1, ALUOP=001 memwrite:1 , ALUSRC:1, regwrite:0
```

12. Register unit testbench



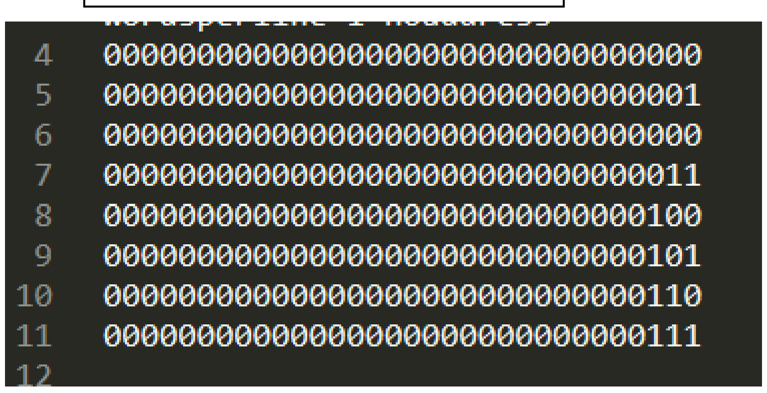
```
    #5 ;
       read_reg_1=3'b000;
       read_reg_2=3'b001;

    #5 ;
       read_reg_1=3'b010;
       read_reg_2=3'b011;
    #5 ;
       read_reg_1=3'b100;
       read_reg_2=3'b101;
    #5 ;
       read_reg_1=3'b110;
       read_reg_2=3'b111;

    #5;
       signal_reg_write=1'b1;
       write_data=3'b000;
       clk=1'b1;
```

Initial register file

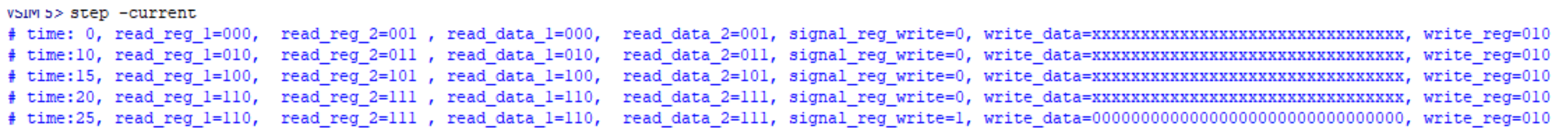


```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
```

Written register file

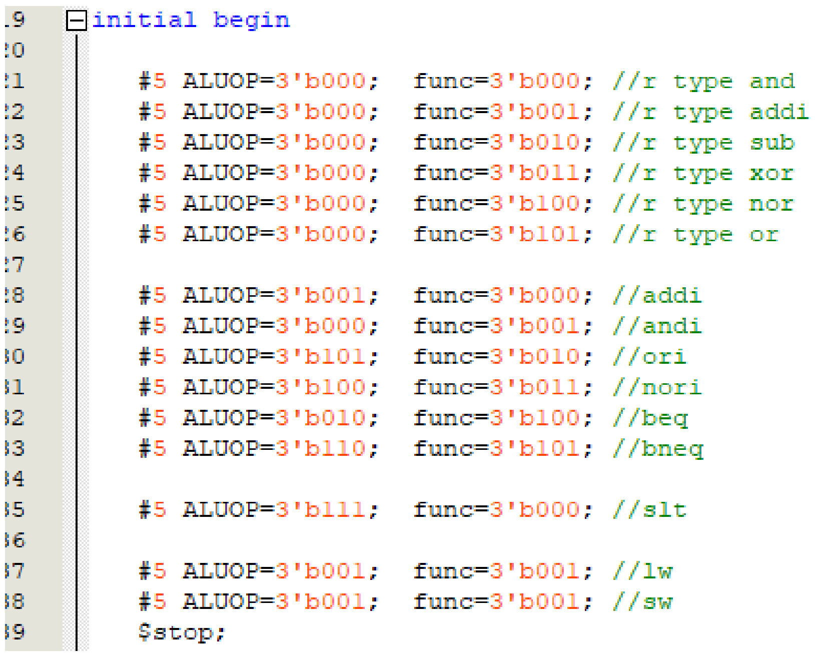


```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
```

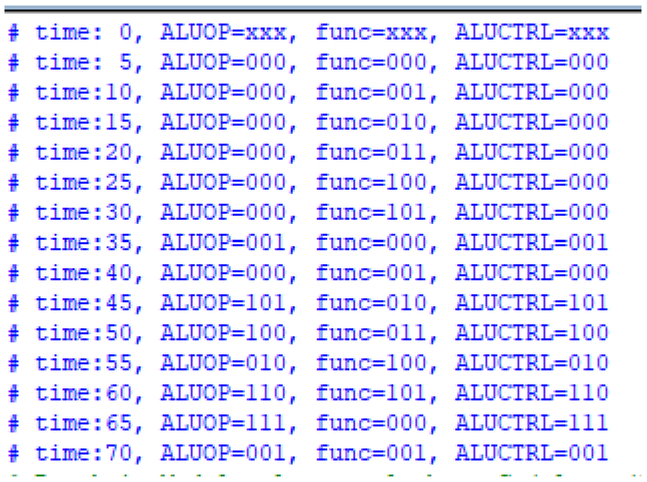


```
VSIM 5> step -current
# time: 0, read_reg_1=000,  read_reg_2=001 , read_data_1=000,  read_data_2=001, signal_reg_write=0, write_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, write_reg=010
# time:10, read_reg_1=010,  read_reg_2=011 , read_data_1=010,  read_data_2=011, signal_reg_write=0, write_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, write_reg=010
# time:15, read_reg_1=100,  read_reg_2=101 , read_data_1=100,  read_data_2=101, signal_reg_write=0, write_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, write_reg=010
# time:20, read_reg_1=110,  read_reg_2=111 , read_data_1=110,  read_data_2=111, signal_reg_write=0, write_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, write_reg=010
# time:25, read_reg_1=110,  read_reg_2=111 , read_data_1=110,  read_data_2=111, signal_reg_write=1, write_data=00000000000000000000000000000000, write_reg=010
```

13. ALU Control Unit Testbench

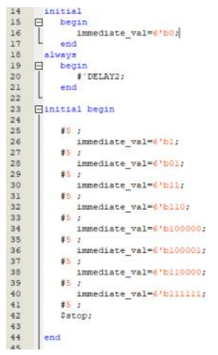


```
initial begin

    #5 ALUOP=3'b000;  func=3'b000; //r type and
    #5 ALUOP=3'b000;  func=3'b001; //r type addi
    #5 ALUOP=3'b000;  func=3'b010; //r type sub
    #5 ALUOP=3'b000;  func=3'b011; //r type xor
    #5 ALUOP=3'b000;  func=3'b100; //r type nor
    #5 ALUOP=3'b000;  func=3'b101; //r type or

    #5 ALUOP=3'b001;  func=3'b000; //addi
    #5 ALUOP=3'b000;  func=3'b001; //andi
    #5 ALUOP=3'b101;  func=3'b010; //ori
    #5 ALUOP=3'b100;  func=3'b011; //nori
    #5 ALUOP=3'b010;  func=3'b100; //beq
    #5 ALUOP=3'b110;  func=3'b101; //bneq

    #5 ALUOP=3'b111;  func=3'b000; //slt

    #5 ALUOP=3'b001;  func=3'b001; //lw
    #5 ALUOP=3'b001;  func=3'b001; //sw
    $stop;
```

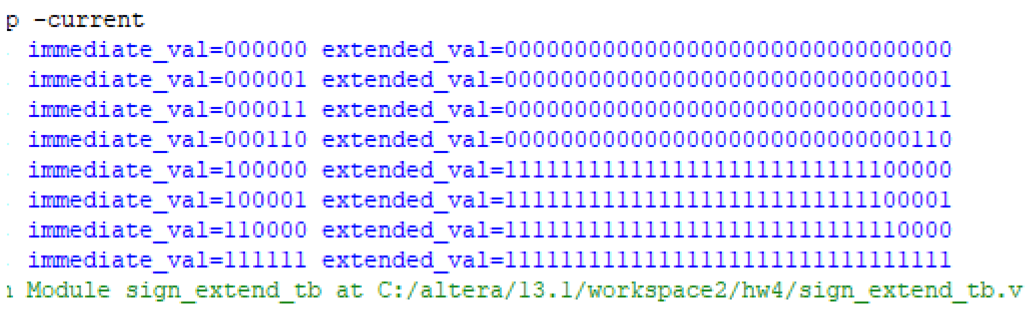


```
# time: 0, ALUOP=xxx, func=xxx, ALUCTRL=xxx
# time: 5, ALUOP=000, func=000, ALUCTRL=000
# time:10, ALUOP=000, func=001, ALUCTRL=000
# time:15, ALUOP=000, func=010, ALUCTRL=000
# time:20, ALUOP=000, func=011, ALUCTRL=000
# time:25, ALUOP=000, func=100, ALUCTRL=000
# time:30, ALUOP=000, func=101, ALUCTRL=000
# time:35, ALUOP=001, func=000, ALUCTRL=001
# time:40, ALUOP=000, func=001, ALUCTRL=000
# time:45, ALUOP=101, func=010, ALUCTRL=101
# time:50, ALUOP=100, func=011, ALUCTRL=100
# time:55, ALUOP=010, func=100, ALUCTRL=010
# time:60, ALUOP=110, func=101, ALUCTRL=110
# time:65, ALUOP=111, func=000, ALUCTRL=111
# time:70, ALUOP=001, func=001, ALUCTRL=001
```

14. Sign Extend Testbench

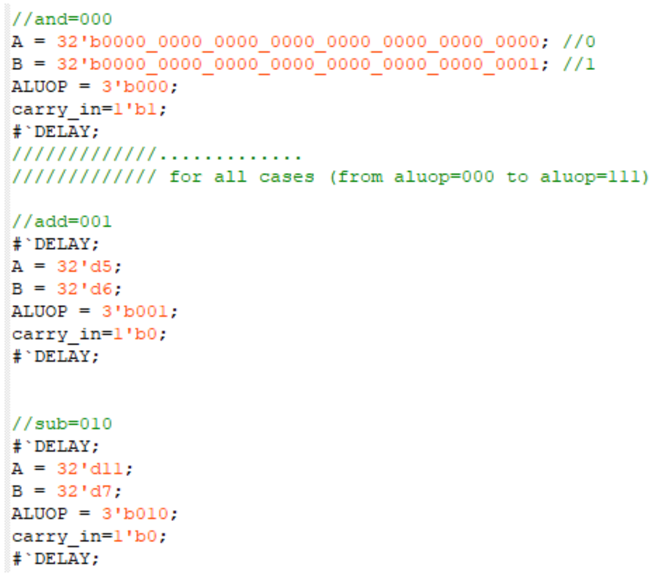


```
initial
    begin
        immediate_val=6'b0;
    end
always
    begin
        #`DELAY2;
    end

initial begin

    #5 ;
        immediate_val=6'b1;
    #5 ;
        immediate_val=6'b01;
    #5 ;
        immediate_val=6'b11;
    #5 ;
        immediate_val=6'b110;
    #5 ;
        immediate_val=6'b100000;
    #5 ;
        immediate_val=6'b100001;
    #5 ;
        immediate_val=6'b110000;
    #5 ;
        immediate_val=6'b111111;
    #5 ;
    $stop;

end
```

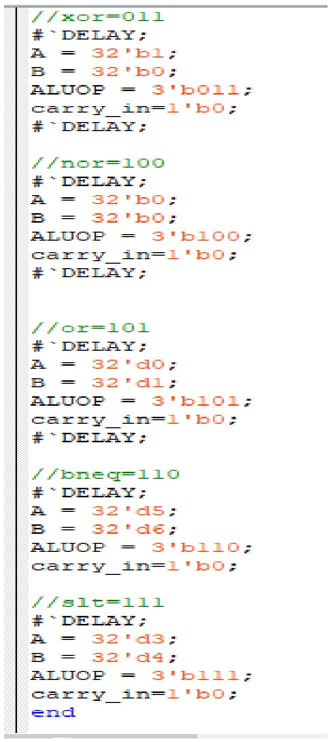


```
p -current
immediate_val=000000 extended_val=00000000000000000000000000000000
immediate_val=000001 extended_val=00000000000000000000000000000001
immediate_val=000011 extended_val=00000000000000000000000000000011
immediate_val=000110 extended_val=00000000000000000000000000000110
immediate_val=100000 extended_val=11111111111111111111111111100000
immediate_val=100001 extended_val=11111111111111111111111111100001
immediate_val=110000 extended_val=11111111111111111111111111110000
immediate_val=111111 extended_val=11111111111111111111111111111111
Module sign_extend_tb at C:/altera/13.1/workspace2/hw4/sign_extend_tb.v
```

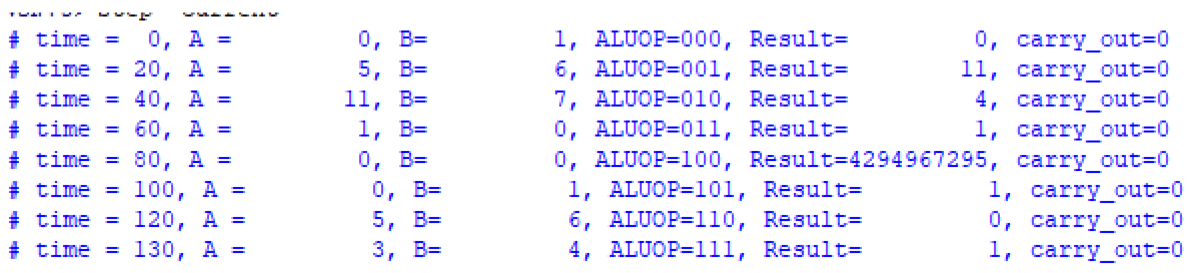
15. ALU Testbench



```
//and=000
A = 32'b0000_0000_0000_0000_0000_0000_0000_0000; //0
B = 32'b0000_0000_0000_0000_0000_0000_0000_0001; //1
ALUOP = 3'b000;
carry_in=1'b1;
#`DELAY;
/////////////.............
///////////// for all cases (from aluop=000 to aluop=111)

//add=001
#`DELAY;
A = 32'd5;
B = 32'd6;
ALUOP = 3'b001;
carry_in=1'b0;
#`DELAY;


//sub=010
#`DELAY;
A = 32'd11;
B = 32'd7;
ALUOP = 3'b010;
carry_in=1'b0;
#`DELAY;
```

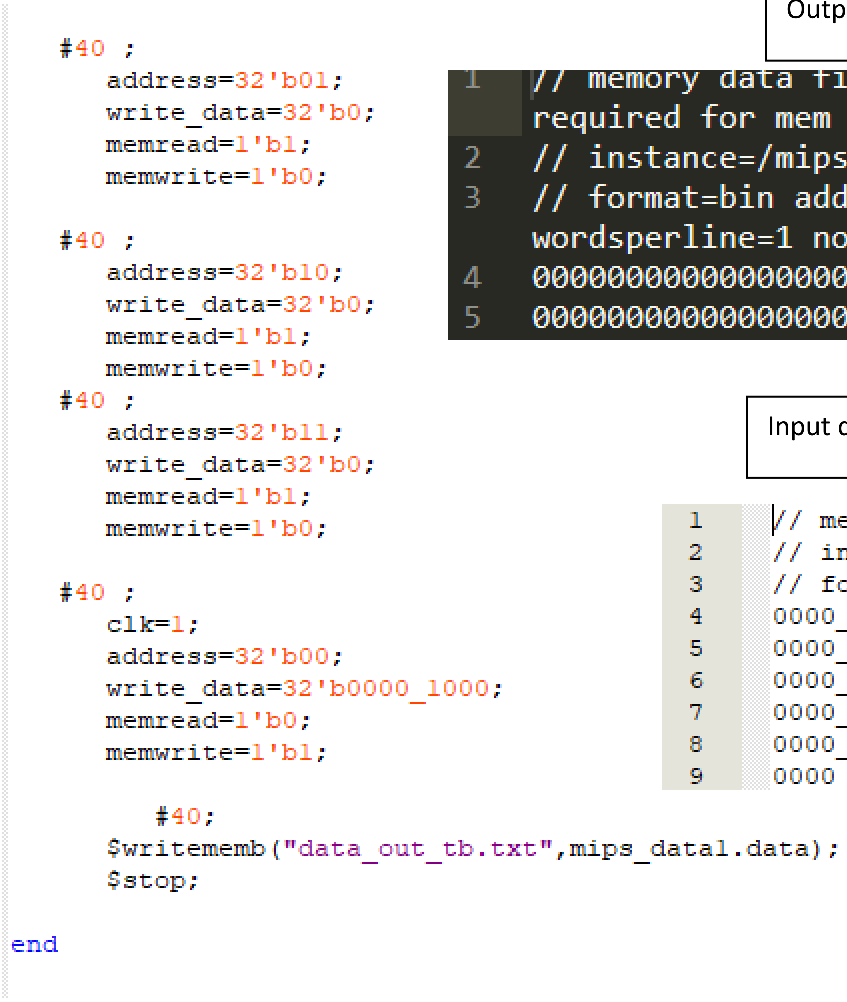


```
//xor=011
#`DELAY;
A = 32'b1;
B = 32'b0;
ALUOP = 3'b011;
carry_in=1'b0;
#`DELAY;

//nor=100
#`DELAY;
A = 32'b0;
B = 32'b0;
ALUOP = 3'b100;
carry_in=1'b0;
#`DELAY;


//or=101
#`DELAY;
A = 32'd0;
B = 32'd1;
ALUOP = 3'b101;
carry_in=1'b0;
#`DELAY;

//bneq=110
#`DELAY;
A = 32'd5;
B = 32'd6;
ALUOP = 3'b110;
carry_in=1'b0;

//slt=111
#`DELAY;
A = 32'd3;
B = 32'd4;
ALUOP = 3'b111;
carry_in=1'b0;
end
```



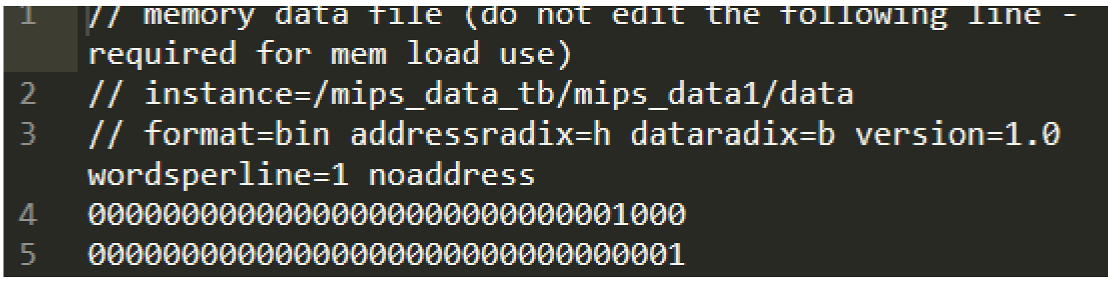
```
[illegible]
# time =  0, A =         0, B=         1, ALUOP=000, Result=         0, carry_out=0
# time = 20, A =         5, B=         6, ALUOP=001, Result=        11, carry_out=0
# time = 40, A =        11, B=         7, ALUOP=010, Result=         4, carry_out=0
# time = 60, A =         1, B=         0, ALUOP=011, Result=         1, carry_out=0
# time = 80, A =         0, B=         0, ALUOP=100, Result=4294967295, carry_out=0
# time = 100, A =         0, B=         1, ALUOP=101, Result=         1, carry_out=0
# time = 120, A =         5, B=         6, ALUOP=110, Result=         0, carry_out=0
# time = 130, A =         3, B=         4, ALUOP=111, Result=         1, carry_out=0
```

16. Data unit testbench



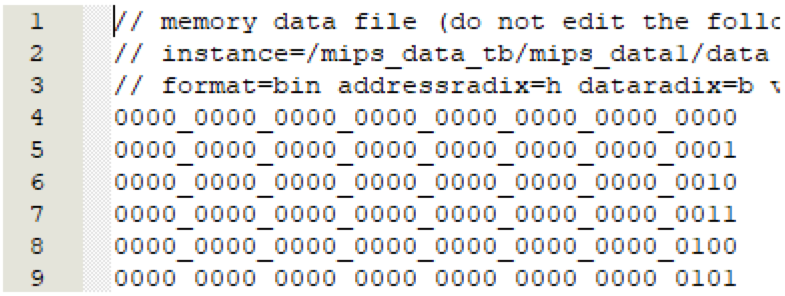
```
   #40 ;
      address=32'b01;
      write_data=32'b0;
      memread=1'b1;
      memwrite=1'b0;

   #40 ;
      address=32'b10;
      write_data=32'b0;
      memread=1'b1;
      memwrite=1'b0;
   #40 ;
      address=32'b11;
      write_data=32'b0;
      memread=1'b1;
      memwrite=1'b0;

   #40 ;
      clk=1;
      address=32'b00;
      write_data=32'b0000_1000;
      memread=1'b0;
      memwrite=1'b1;

         #40;
      $writememb("data_out_tb.txt",mips_data1.data);
      $stop;

end
```

Output data of testbench

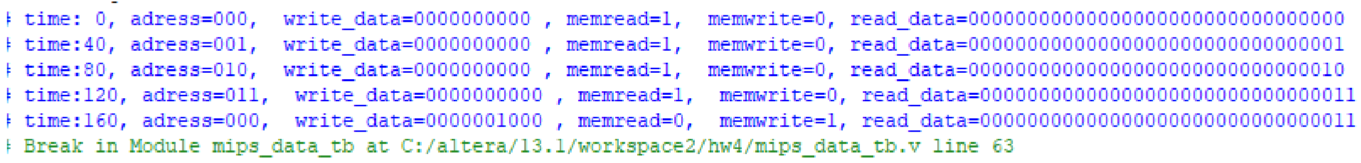


```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_data_tb/mips_data1/data
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000001000
00000000000000000000000000000001
```

Input data of testbench

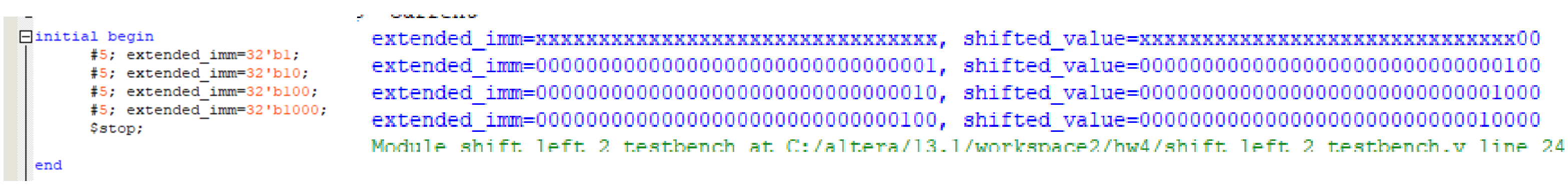


```
// memory data file (do not edit the follc
// instance=/mips_data_tb/mips_data1/data
// format=bin addressradix=h dataradix=b v
0000_0000_0000_0000_0000_0000_0000_0000
0000_0000_0000_0000_0000_0000_0000_0001
0000_0000_0000_0000_0000_0000_0000_0010
0000_0000_0000_0000_0000_0000_0000_0011
0000_0000_0000_0000_0000_0000_0000_0100
0000_0000_0000_0000_0000_0000_0000_0101
```



```
# time: 0, adress=000,  write_data=0000000000 , memread=1,  memwrite=0, read_data=00000000000000000000000000000000
# time:40, adress=001,  write_data=0000000000 , memread=1,  memwrite=0, read_data=00000000000000000000000000000001
# time:80, adress=010,  write_data=0000000000 , memread=1,  memwrite=0, read_data=00000000000000000000000000000010
# time:120, adress=011,  write_data=0000000000 , memread=1,  memwrite=0, read_data=00000000000000000000000000000011
# time:160, adress=000,  write_data=0000001000 , memread=0,  memwrite=1, read_data=00000000000000000000000000000011
# Break in Module mips_data_tb at C:/altera/13.1/workspace2/hw4/mips_data_tb.v line 63
```

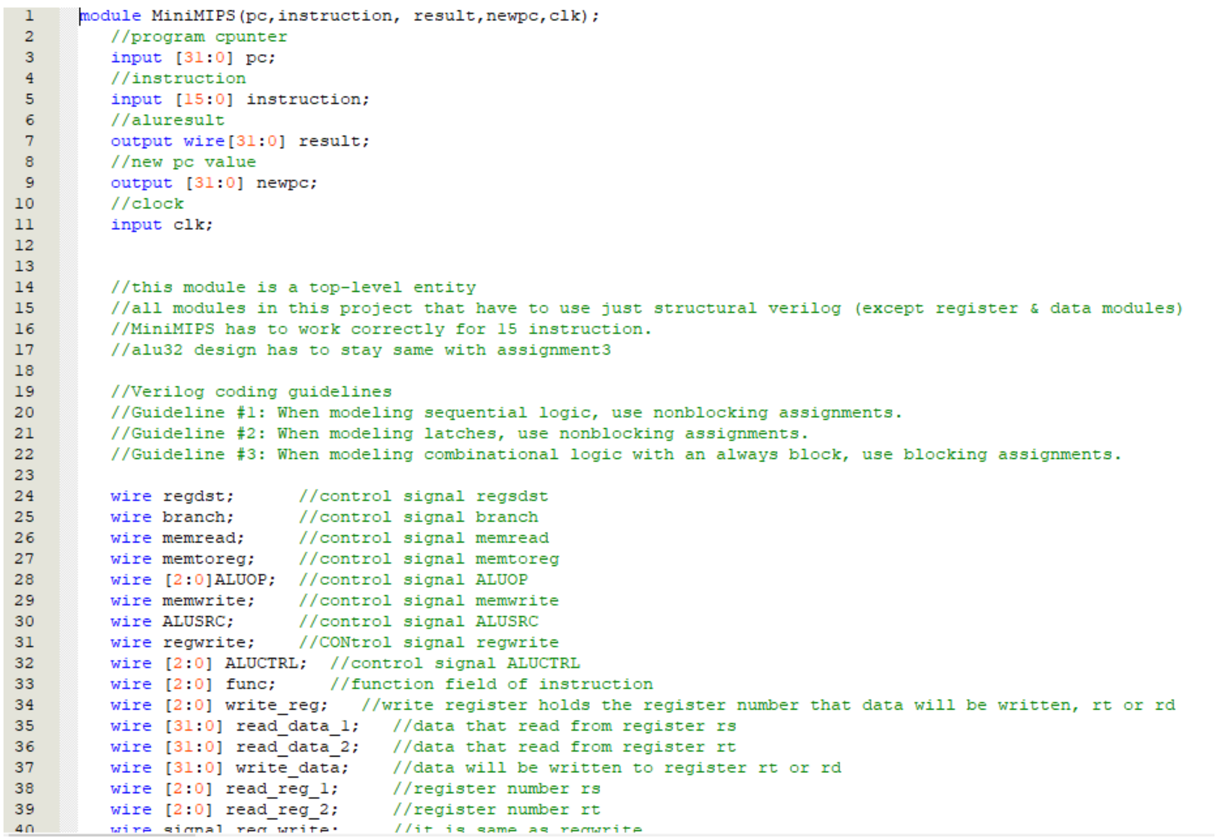
17. Shift left 2 unit testbench



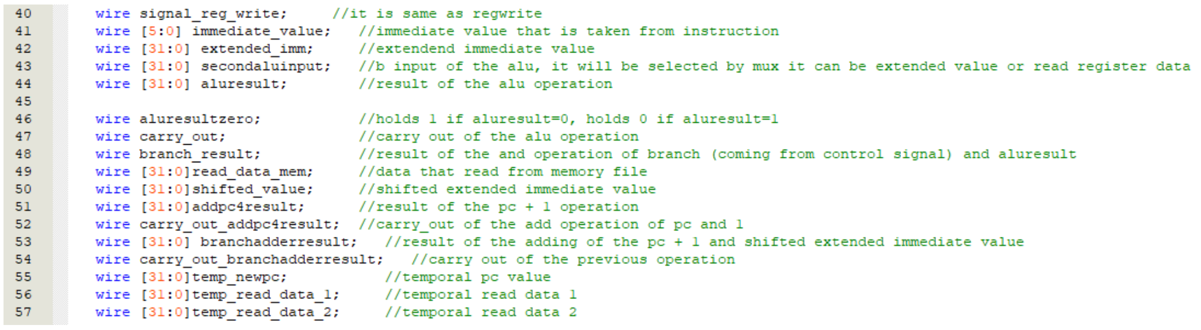
```
initial begin
	#5; extended_imm=32'b1;
	#5; extended_imm=32'b10;
	#5; extended_imm=32'b100;
	#5; extended_imm=32'b1000;
	$stop;

end
```

```
extended_imm=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, shifted_value=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx00
extended_imm=00000000000000000000000000000001, shifted_value=00000000000000000000000000000100
extended_imm=00000000000000000000000000000010, shifted_value=00000000000000000000000000001000
extended_imm=00000000000000000000000000000100, shifted_value=00000000000000000000000000010000
Module shift_left_2_testbench at C:/altera/13.1/workspace2/hw4/shift_left_2_testbench.v line 24
```

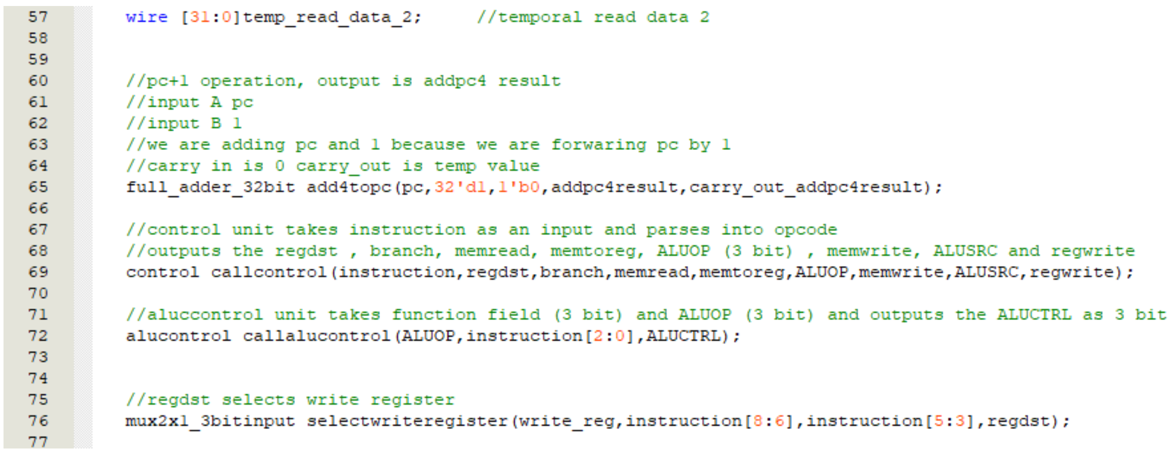
18. MiniMIPS



```
module MiniMIPS(pc,instruction, result,newpc,clk);
	//program cpunter
	input [31:0] pc;
	//instruction
	input [15:0] instruction;
	//aluresult
	output wire[31:0] result;
	//new pc value
	output [31:0] newpc;
	//clock
	input clk;


	//this module is a top-level entity
	//all modules in this project that have to use just structural verilog (except register & data modules)
	//MiniMIPS has to work correctly for 15 instruction.
	//alu32 design has to stay same with assignment3

	//Verilog coding guidelines
	//Guideline #1: When modeling sequential logic, use nonblocking assignments.
	//Guideline #2: When modeling latches, use nonblocking assignments.
	//Guideline #3: When modeling combinational logic with an always block, use blocking assignments.

	wire regdst;       //control signal regsdst
	wire branch;       //control signal branch
	wire memread;      //control signal memread
	wire memtoreg;     //control signal memtoreg
	wire [2:0]ALUOP;   //control signal ALUOP
	wire memwrite;     //control signal memwrite
	wire ALUSRC;       //control signal ALUSRC
	wire regwrite;     //CONtrol signal regwrite
	wire [2:0] ALUCTRL;   //control signal ALUCTRL
	wire [2:0] func;      //function field of instruction
	wire [2:0] write_reg;    //write register holds the register number that data will be written, rt or rd
	wire [31:0] read_data_1;   //data that read from register rs
	wire [31:0] read_data_2;   //data that read from register rt
	wire [31:0] write_data;    //data will be written to register rt or rd
	wire [2:0] read_reg_1;     //register number rs
	wire [2:0] read_reg_2;     //register number rt
```

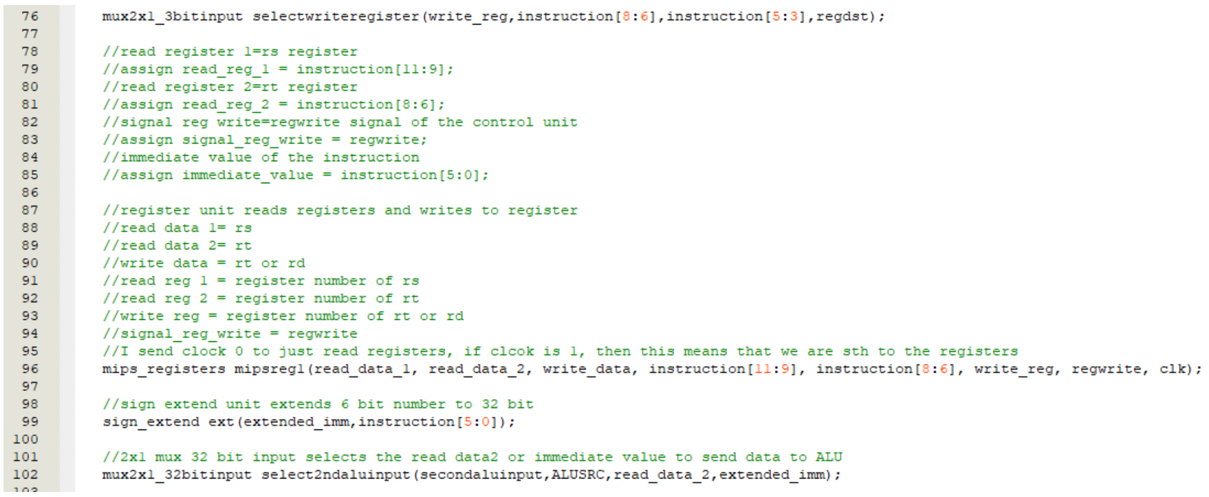


```
	wire signal_reg_write;     //it is same as regwrite
	wire [5:0] immediate_value;   //immediate value that is taken from instruction
	wire [31:0] extended_imm;     //extendend immediate value
	wire [31:0] secondaluinput;   //b input of the alu, it will be selected by mux it can be extended value or read register data
	wire [31:0] aluresult;        //result of the alu operation

	wire aluresultzero;           //holds 1 if aluresult=0, holds 0 if aluresult=1
	wire carry_out;               //carry out of the alu operation
	wire branch_result;           //result of the and operation of branch (coming from control signal) and aluresult
	wire [31:0]read_data_mem;     //data that read from memory file
	wire [31:0]shifted_value;     //shifted extended immediate value
	wire [31:0]addpc4result;      //result of the pc + 1 operation
	wire carry_out_addpc4result;  //carry_out of the add operation of pc and 1
	wire [31:0] branchadderresult;   //result of the adding of the pc + 1 and shifted extended immediate value
	wire carry_out_branchadderresult;   //carry out of the previous operation
	wire [31:0]temp_newpc;          //temporal pc value
	wire [31:0]temp_read_data_1;    //temporal read data 1
	wire [31:0]temp_read_data_2;    //temporal read data 2
```

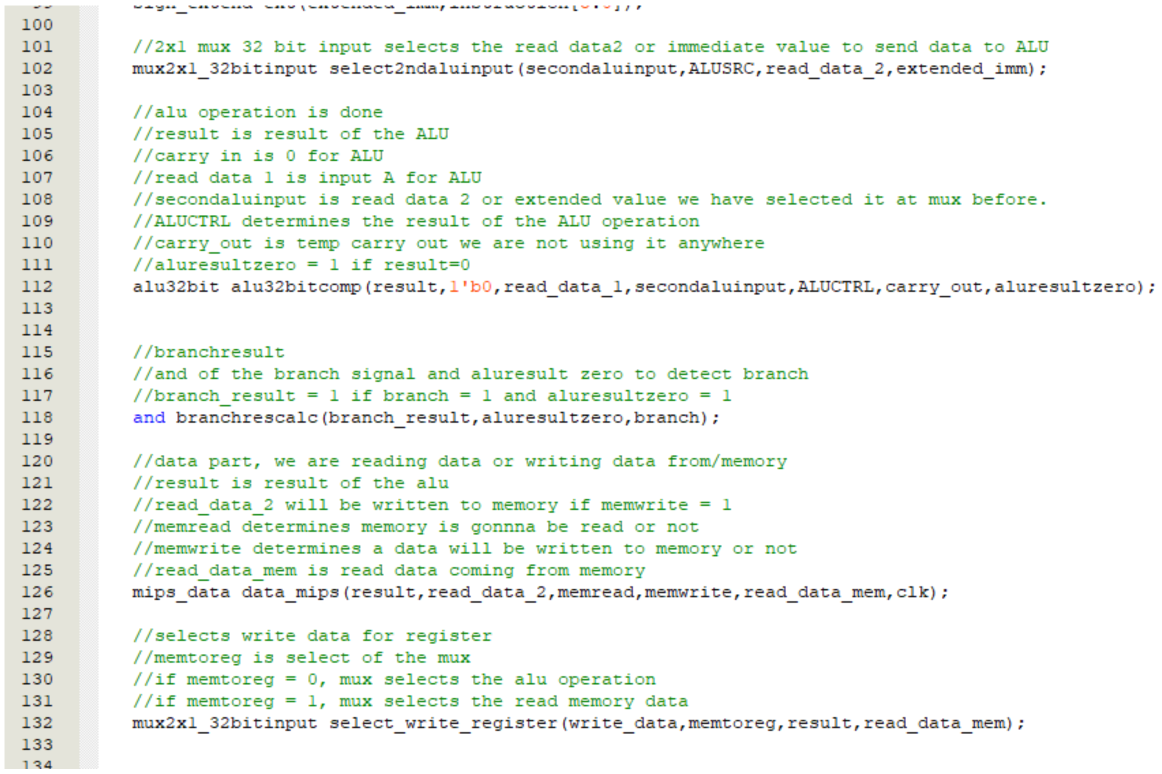


```
	wire [31:0]temp_read_data_2;    //temporal read data 2
```

```


	//pc+1 operation, output is addpc4 result
	//input A pc
	//input B 1
	//we are adding pc and 1 because we are forwaring pc by 1
	//carry in is 0 carry_out is temp value
	full_adder_32bit add4topc(pc,32'd1,1'b0,addpc4result,carry_out_addpc4result);

	//control unit takes instruction as an input and parses into opcode
	//outputs the regdst , branch, memread, memtoreg, ALUOP (3 bit) , memwrite, ALUSRC and regwrite
	control callcontrol(instruction,regdst,branch,memread,memtoreg,ALUOP,memwrite,ALUSRC,regwrite);

	//aluccontrol unit takes function field (3 bit) and ALUOP (3 bit) and outputs the ALUCTRL as 3 bit
	alucontrol callalucontrol(ALUOP,instruction[2:0],ALUCTRL);


	//regdst selects write register
	mux2x1_3bitinput selectwriteregister(write_reg,instruction[8:6],instruction[5:3],regdst);
```

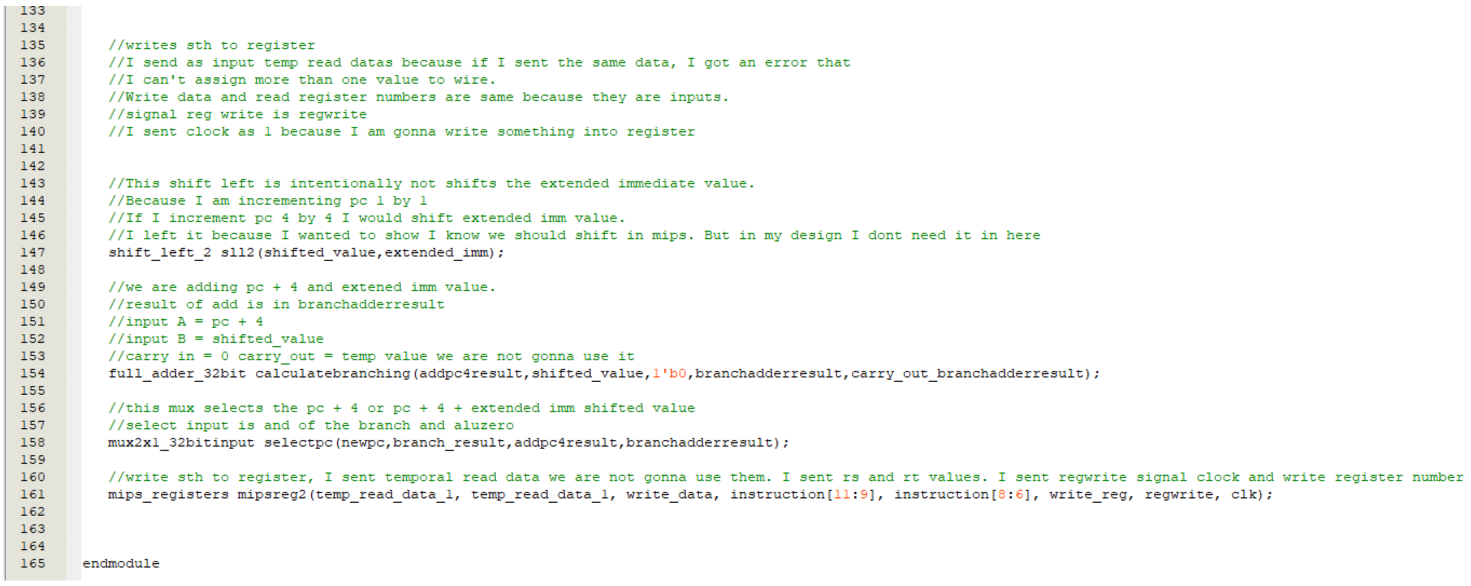


```verilog
mux2x1_3bitinput selectwriteregister(write_reg,instruction[8:6],instruction[5:3],regdst);

//read register 1=rs register
//assign read_reg_1 = instruction[11:9];
//read register 2=rt register
//assign read_reg_2 = instruction[8:6];
//signal reg write=regwrite signal of the control unit
//assign signal_reg_write = regwrite;
//immediate value of the instruction
//assign immediate_value = instruction[5:0];

//register unit reads registers and writes to register
//read data 1= rs
//read data 2= rt
//write data = rt or rd
//read reg 1 = register number of rs
//read reg 2 = register number of rt
//write reg = register number of rt or rd
//signal_reg_write = regwrite
//I send clock 0 to just read registers, if clcok is 1, then this means that we are sth to the registers
mips_registers mipsreg1(read_data_1, read_data_2, write_data, instruction[11:9], instruction[8:6], write_reg, regwrite, clk);

//sign extend unit extends 6 bit number to 32 bit
sign_extend ext(extended_imm,instruction[5:0]);

//2x1 mux 32 bit input selects the read data2 or immediate value to send data to ALU
mux2x1_32bitinput select2ndaluinput(secondaluinput,ALUSRC,read_data_2,extended_imm);
```



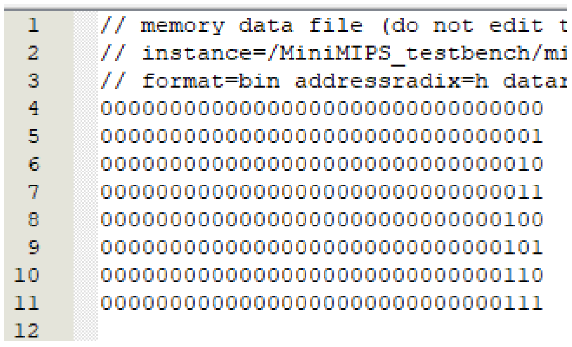
```verilog
//2x1 mux 32 bit input selects the read data2 or immediate value to send data to ALU
mux2x1_32bitinput select2ndaluinput(secondaluinput,ALUSRC,read_data_2,extended_imm);
```

```verilog
//alu operation is done
//result is result of the ALU
//carry in is 0 for ALU
//read data 1 is input A for ALU
//secondaluinput is read data 2 or extended value we have selected it at mux before.
//ALUCTRL determines the result of the ALU operation
//carry_out is temp carry out we are not using it anywhere
//aluresultzero = 1 if result=0
alu32bit alu32bitcomp(result,1'b0,read_data_1,secondaluinput,ALUCTRL,carry_out,aluresultzero);


//branchresult
//and of the branch signal and aluresult zero to detect branch
//branch_result = 1 if branch = 1 and aluresultzero = 1
and branchrescalc(branch_result,aluresultzero,branch);

//data part, we are reading data or writing data from/memory
//result is result of the alu
//read_data_2 will be written to memory if memwrite = 1
//memread determines memory is gonnna be read or not
//memwrite determines a data will be written to memory or not
//read_data_mem is read data coming from memory
mips_data data_mips(result,read_data_2,memread,memwrite,read_data_mem,clk);

//selects write data for register
//memtoreg is select of the mux
//if memtoreg = 0, mux selects the alu operation
//if memtoreg = 1, mux selects the read memory data
mux2x1_32bitinput select_write_register(write_data,memtoreg,result,read_data_mem);
```



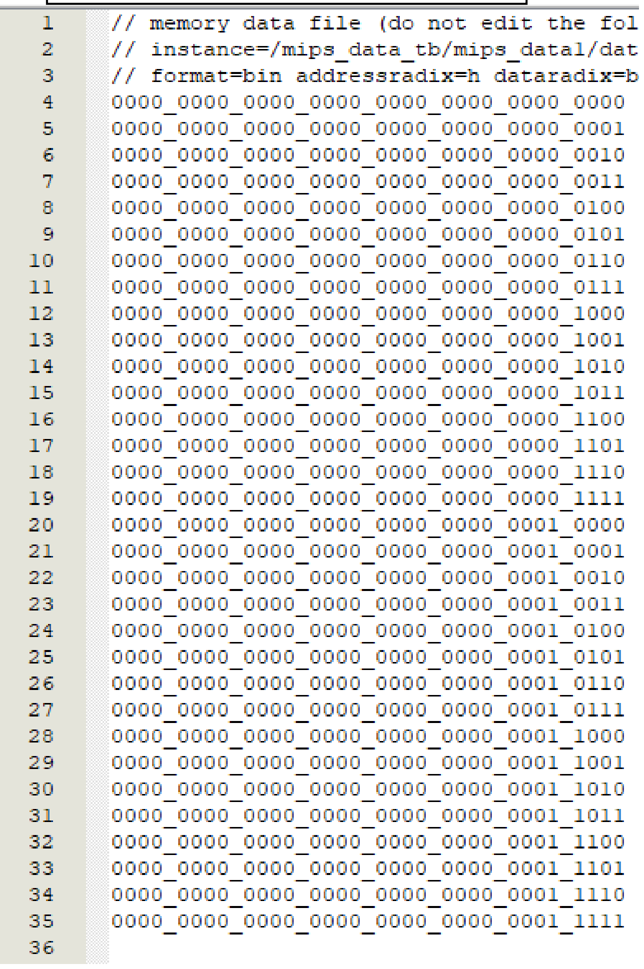
```verilog
//writes sth to register
//I send as input temp read datas because if I sent the same data, I got an error that
//I can't assign more than one value to wire.
//Write data and read register numbers are same because they are inputs.
//signal reg write is regwrite
//I sent clock as 1 because I am gonna write something into register


//This shift left is intentionally not shifts the extended immediate value.
//Because I am incrementing pc 1 by 1
//If I increment pc 4 by 4 I would shift extended imm value.
//I left it because I wanted to show I know we should shift in mips. But in my design I dont need it in here
shift_left_2 sll2(shifted_value,extended_imm);

//we are adding pc + 4 and extened imm value.
//result of add is in branchadderresult
//input A = pc + 4
//input B = shifted_value
//carry in = 0 carry_out = temp value we are not gonna use it
full_adder_32bit calculatebranching(addpc4result,shifted_value,1'b0,branchadderresult,carry_out_branchadderresult);

//this mux selects the pc + 4 or pc + 4 + extended imm shifted value
//select input is and of the branch and aluzero
mux2x1_32bitinput selectpc(newpc,branch_result,addpc4result,branchadderresult);

//write sth to register, I sent temporal read data we are not gonna use them. I sent rs and rt values. I sent regwrite signal clock and write register number
mips_registers mipsreg2(temp_read_data_1, temp_read_data_1, write_data, instruction[11:9], instruction[8:6], write_reg, regwrite, clk);



endmodule
```

My initial register file



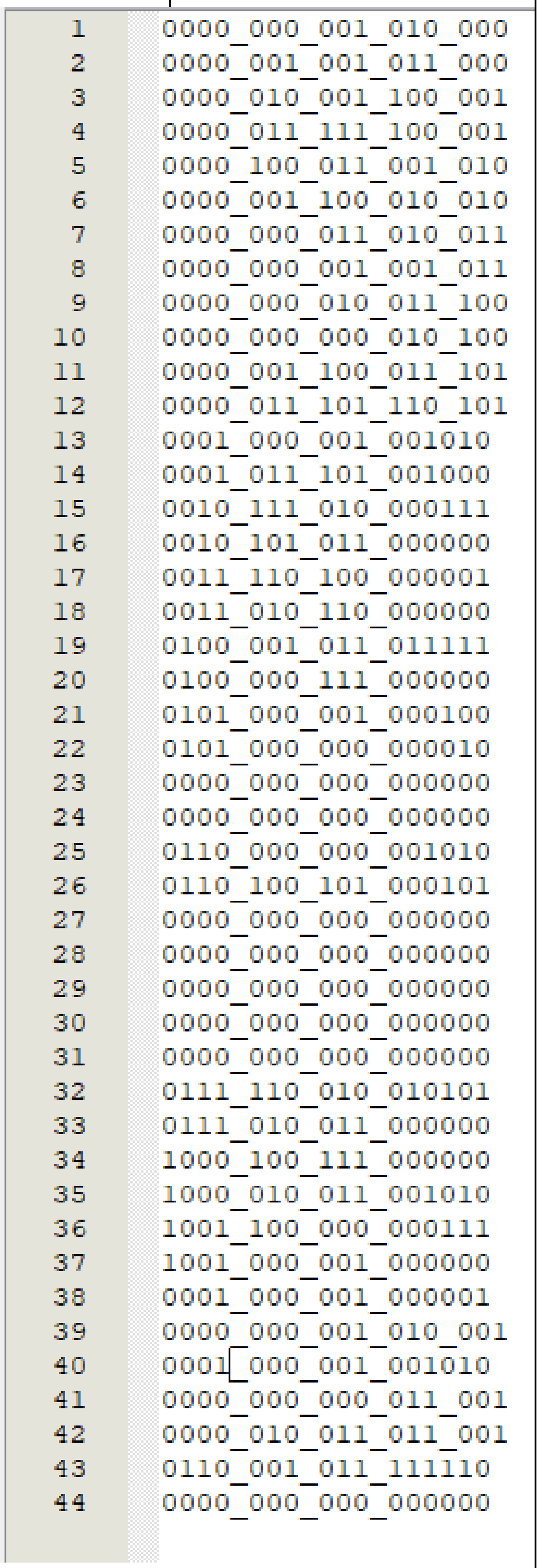
```
// memory data file (do not edit t
// instance=/MiniMIPS_testbench/mi
// format=bin addressradix=h datar
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111

```

My initial memory file



```
// memory data file (do not edit the fol
// instance=/mips_data_tb/mips_datal/dat
// format=bin addressradix=h dataradix=b
0000_0000_0000_0000_0000_0000_0000_0000
0000_0000_0000_0000_0000_0000_0000_0001
0000_0000_0000_0000_0000_0000_0000_0010
0000_0000_0000_0000_0000_0000_0000_0011
0000_0000_0000_0000_0000_0000_0000_0100
0000_0000_0000_0000_0000_0000_0000_0101
0000_0000_0000_0000_0000_0000_0000_0110
0000_0000_0000_0000_0000_0000_0000_0111
0000_0000_0000_0000_0000_0000_0000_1000
0000_0000_0000_0000_0000_0000_0000_1001
0000_0000_0000_0000_0000_0000_0000_1010
0000_0000_0000_0000_0000_0000_0000_1011
0000_0000_0000_0000_0000_0000_0000_1100
0000_0000_0000_0000_0000_0000_0000_1101
0000_0000_0000_0000_0000_0000_0000_1110
0000_0000_0000_0000_0000_0000_0000_1111
0000_0000_0000_0000_0000_0000_0001_0000
0000_0000_0000_0000_0000_0000_0001_0001
0000_0000_0000_0000_0000_0000_0001_0010
0000_0000_0000_0000_0000_0000_0001_0011
0000_0000_0000_0000_0000_0000_0001_0100
0000_0000_0000_0000_0000_0000_0001_0101
0000_0000_0000_0000_0000_0000_0001_0110
0000_0000_0000_0000_0000_0000_0001_0111
0000_0000_0000_0000_0000_0000_0001_1000
0000_0000_0000_0000_0000_0000_0001_1001
0000_0000_0000_0000_0000_0000_0001_1010
0000_0000_0000_0000_0000_0000_0001_1011
0000_0000_0000_0000_0000_0000_0001_1100
0000_0000_0000_0000_0000_0000_0001_1101
0000_0000_0000_0000_0000_0000_0001_1110
0000_0000_0000_0000_0000_0000_0001_1111

```

My instructions



```
0000_000_001_010_000
0000_001_001_011_000
0000_010_001_100_001
0000_011_111_100_001
0000_100_011_001_010
0000_001_100_010_010
0000_000_011_010_011
0000_000_001_001_011
0000_000_010_011_100
0000_000_000_010_100
0000_001_100_011_101
0000_011_101_110_101
0001_000_001_001010
0001_011_101_001000
0010_111_010_000111
0010_101_011_000000
0011_110_100_000001
0011_010_110_000000
0100_001_011_011111
0100_000_111_000000
0101_000_001_000100
0101_000_000_000010
0000_000_000_000000
0000_000_000_000000
0110_000_000_001010
0110_100_101_000101
0000_000_000_000000
0000_000_000_000000
0000_000_000_000000
0000_000_000_000000
0000_000_000_000000
0111_110_010_010101
0111_010_011_000000
1000_100_111_000000
1000_010_011_001010
1001_100_000_000111
1001_000_001_000000
0001_000_001_000001
0000_000_001_010_001
0001_000_001_001010
0000_000_000_011_001
0000_010_011_011_001
0110_001_011_111110
0000_000_000_000000
```

INSTRUCTIONS

And $0,$1,$2

And $1,$1,$3

Add $2,$1,$4

Add $3,$7,$4

Sub $4,$3,$1

Sub $1,$4,$2

Xor $0,$3,$2

Xor $0,$2,$3

Nor $0,$2,$3

Nor $0,$0,$2

Or $1,$4,$3

Or $3, $5,$6

Addi $0,$1,10

Addi $3,$5,8

Andi $7,$2,7

Andi $5,$3,0

Ori $6,$4,1

Ori $2,$0,6

Nori $1,$3,63

Nori $0,$7,0

Beq $0,$1,4

Beq $0,$0,2

Bneq $0,$0,10

Bneq $4,$5,5

Slt $6,$2,21

Slt $2,$3,0

Lw $7,0 ($4)

Lw $3,10($2)

Sw $0,7($4)

Sw $1,0($0)

Addi $0,$1,1

Add $0,$1,$2

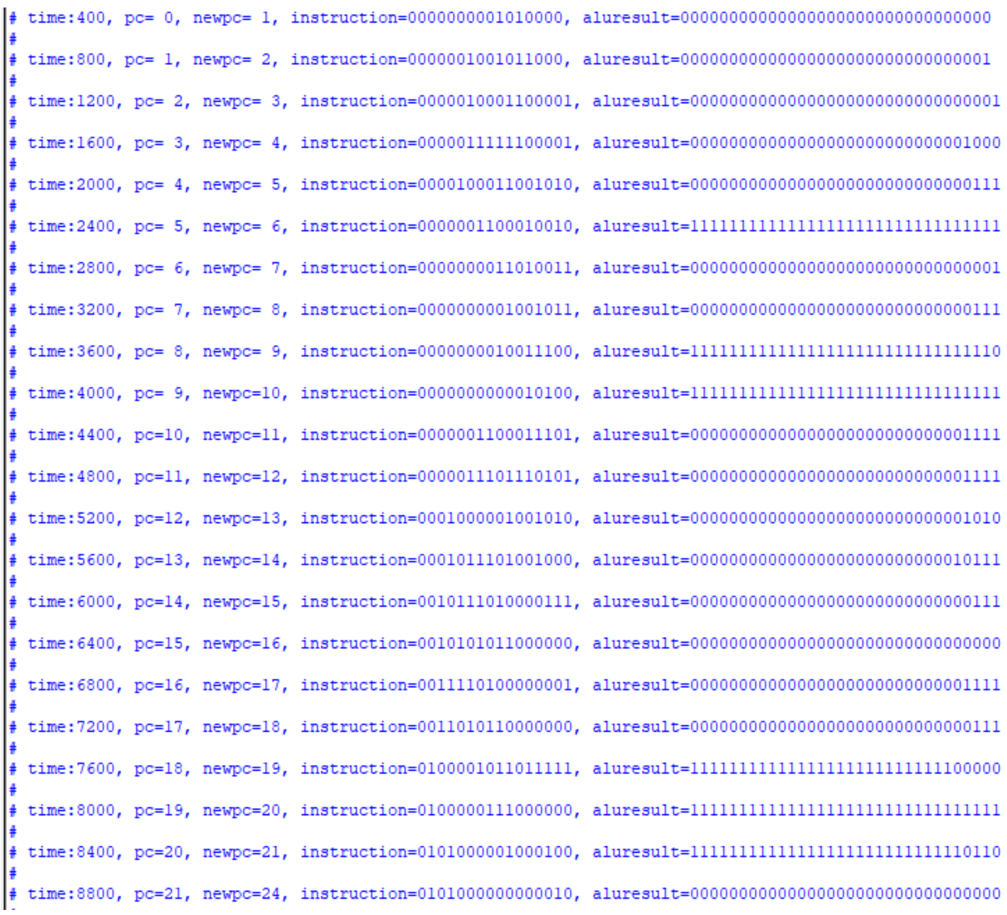
Addi $0,$1,10

Add $0,$0,$3

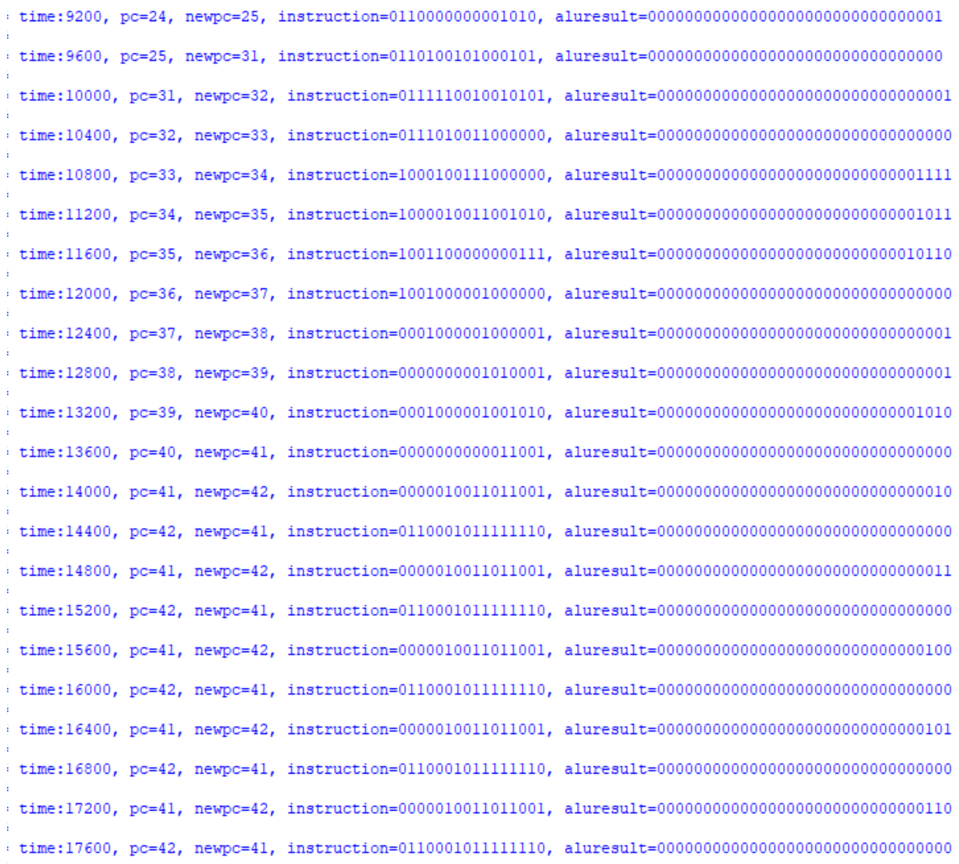
Add $2,$3,$3

Bneq $1,$3 (my for loop is in here)

Empty instruction



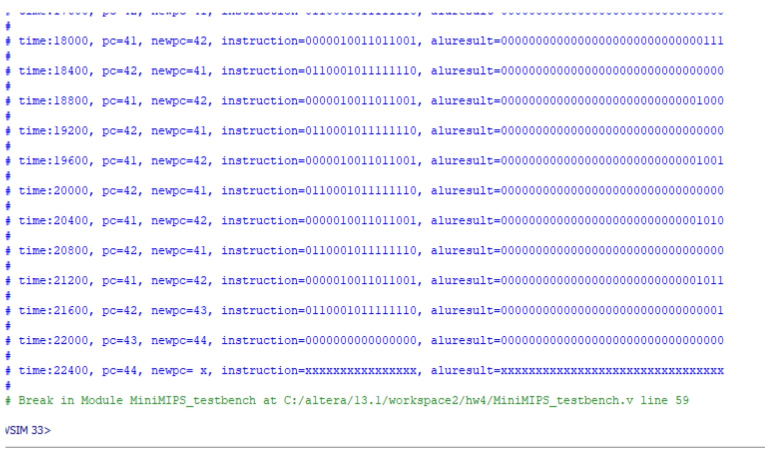
```
# time:400, pc= 0, newpc= 1, instruction=0000000001010000, aluresult=00000000000000000000000000000000
#
# time:800, pc= 1, newpc= 2, instruction=0000001001011000, aluresult=00000000000000000000000000000001
#
# time:1200, pc= 2, newpc= 3, instruction=0000010001100001, aluresult=00000000000000000000000000000001
#
# time:1600, pc= 3, newpc= 4, instruction=0000011111100001, aluresult=00000000000000000000000000001000
#
# time:2000, pc= 4, newpc= 5, instruction=0000100011001010, aluresult=00000000000000000000000000000111
#
# time:2400, pc= 5, newpc= 6, instruction=0000001100010010, aluresult=11111111111111111111111111111111
#
# time:2800, pc= 6, newpc= 7, instruction=0000000011010011, aluresult=00000000000000000000000000000001
#
# time:3200, pc= 7, newpc= 8, instruction=0000000001001011, aluresult=00000000000000000000000000000111
#
# time:3600, pc= 8, newpc= 9, instruction=0000000010011100, aluresult=11111111111111111111111111111110
#
# time:4000, pc= 9, newpc=10, instruction=0000000000010100, aluresult=11111111111111111111111111111111
#
# time:4400, pc=10, newpc=11, instruction=0000001100011101, aluresult=00000000000000000000000000001111
#
# time:4800, pc=11, newpc=12, instruction=0000011101110101, aluresult=00000000000000000000000000001111
#
# time:5200, pc=12, newpc=13, instruction=0001000001001010, aluresult=00000000000000000000000000001010
#
# time:5600, pc=13, newpc=14, instruction=0001011101001000, aluresult=00000000000000000000000000010111
#
# time:6000, pc=14, newpc=15, instruction=0010111010000111, aluresult=00000000000000000000000000000111
#
# time:6400, pc=15, newpc=16, instruction=0010101011000000, aluresult=00000000000000000000000000000000
#
# time:6800, pc=16, newpc=17, instruction=0011110100000001, aluresult=00000000000000000000000000001111
#
# time:7200, pc=17, newpc=18, instruction=0011010110000000, aluresult=00000000000000000000000000000111
#
# time:7600, pc=18, newpc=19, instruction=0100001011011111, aluresult=11111111111111111111111111100000
#
# time:8000, pc=19, newpc=20, instruction=0100000111000000, aluresult=11111111111111111111111111111111
#
# time:8400, pc=20, newpc=21, instruction=0101000001000100, aluresult=11111111111111111111111111110110
#
# time:8800, pc=21, newpc=24, instruction=0101000000000010, aluresult=00000000000000000000000000000000
```



```
time:9200, pc=24, newpc=25, instruction=0110000000001010, aluresult=00000000000000000000000000000001

time:9600, pc=25, newpc=31, instruction=0110100101000101, aluresult=00000000000000000000000000000000

time:10000, pc=31, newpc=32, instruction=0111110010010101, aluresult=00000000000000000000000000000001

time:10400, pc=32, newpc=33, instruction=0111010011000000, aluresult=00000000000000000000000000000000

time:10800, pc=33, newpc=34, instruction=1000100111000000, aluresult=00000000000000000000000000001111

time:11200, pc=34, newpc=35, instruction=1000010011001010, aluresult=00000000000000000000000000001011

time:11600, pc=35, newpc=36, instruction=1001100000000111, aluresult=00000000000000000000000000010110

time:12000, pc=36, newpc=37, instruction=1001000001000000, aluresult=00000000000000000000000000000000

time:12400, pc=37, newpc=38, instruction=0001000001000001, aluresult=00000000000000000000000000000001

time:12800, pc=38, newpc=39, instruction=0000000001010001, aluresult=00000000000000000000000000000001

time:13200, pc=39, newpc=40, instruction=0001000001001010, aluresult=00000000000000000000000000001010
```

For loop start

```
time:13600, pc=40, newpc=41, instruction=0000000000011001, aluresult=00000000000000000000000000000000

time:14000, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000000010

time:14400, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000

time:14800, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000000011

time:15200, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000

time:15600, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000000100

time:16000, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000

time:16400, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000000101

time:16800, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000

time:17200, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000000110

time:17600, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000
```

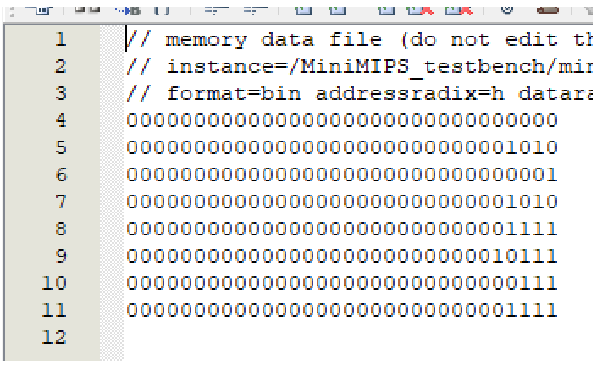


```
#
# time:18000, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000000111
#
# time:18400, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000
#
# time:18800, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000001000
#
# time:19200, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000
#
# time:19600, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000001001
#
# time:20000, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000
#
```

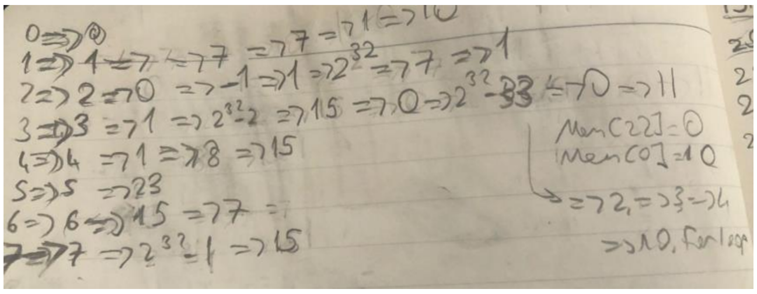
For loop end

```
# time:20400, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000001010
#
# time:20800, pc=42, newpc=41, instruction=0110001011111110, aluresult=00000000000000000000000000000000
#
# time:21200, pc=41, newpc=42, instruction=0000010011011001, aluresult=00000000000000000000000000001011
#
# time:21600, pc=42, newpc=43, instruction=0110001011111110, aluresult=00000000000000000000000000000001
#
# time:22000, pc=43, newpc=44, instruction=0000000000000000, aluresult=00000000000000000000000000000000
#
# time:22400, pc=44, newpc= x, instruction=xxxxxxxxxxxxxxxx, aluresult=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
# Break in Module MiniMIPS_testbench at C:/altera/13.1/workspace2/hw4/MiniMIPS_testbench.v line 59

VSIM 33>
```

Register output file



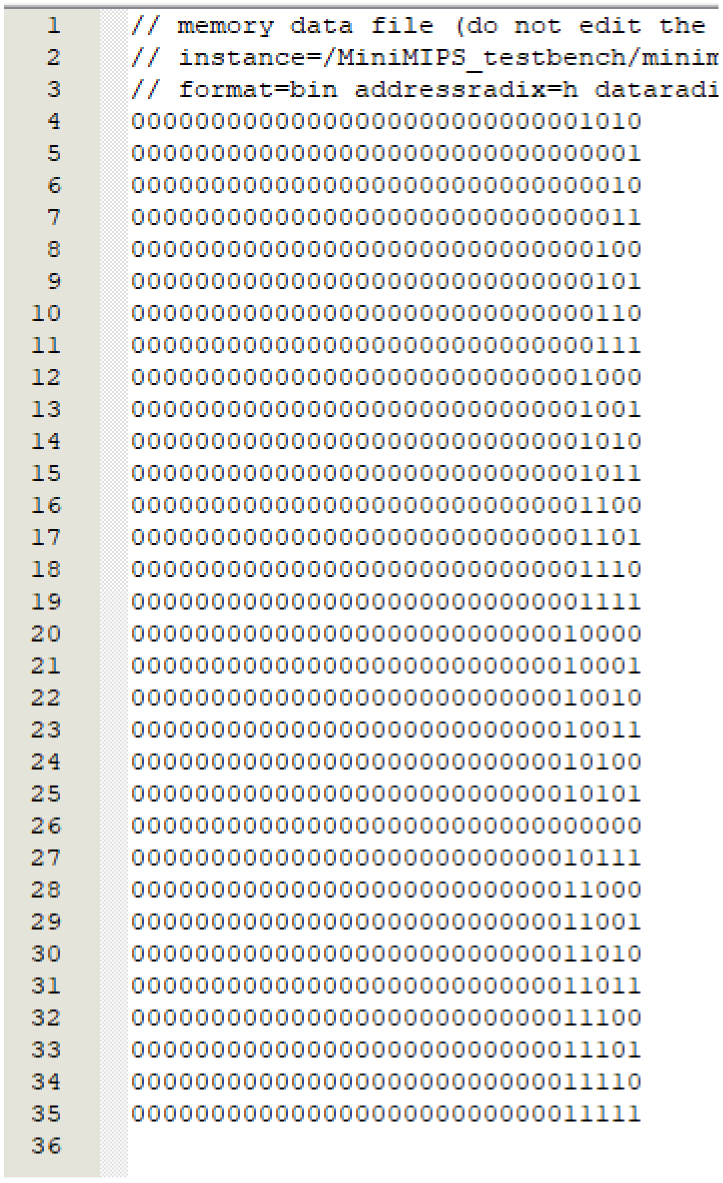
```
// memory data file (do not edit th
// instance=/MiniMIPS_testbench/mir
// format=bin addressradix=h datara
00000000000000000000000000000000
00000000000000000000000000001010
00000000000000000000000000000001
00000000000000000000000000001010
00000000000000000000000000001111
00000000000000000000000000010111
00000000000000000000000000000111
00000000000000000000000000001111
```



0 => 0
1 => ~~1 => 7~~ => 7 => 7 => 7 => 1 => 10
=> 2^{32} => 7 => 1
2 => 2 => 0 => -1 => 1 => 2^{32} => 7 => 1
=> 0 => 2^{32} - 33 => 0 => 11
3 => 3 => 1 => 2^{32}-2 => 15 => 0
4 => 4 => 1 => 8 => 15
5 => 5 => 23
6 => 6 => 15 => 7 =
7 => 7 => 2^{32}-1 => 15

Mem[22] = 0
Mem[0] = 10
=> 2, => 3 => 4
=> 10, for loop

Data output file



```
// memory data file (do not edit the
// instance=/MiniMIPS_testbench/minim
// format=bin addressradix=h dataradi
00000000000000000000000000001010
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000000000
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111
```