VLSI System Design (Graduate Level)
Fall 2012

HOMEWORK I (Part II) ver. 6.1

Due day: 9:00am Oct. 25 (Thursday), 2012

This homework is to let you familiar with tools and Verilog language. It includes A) basic exercises and B) the first part of a simplified multi-cycle **(SMILE)** CPU with only 17 instructions. Part B is to let the first-time CPU designer be familiar with the instruction set architecture, dataflow modeling and controller design. Some problems have reference code. They are for your reference only. Most likely, you need to modify them to fit the problem specification.

General rules for deliverables

- This homework needs to be completed by INDIVIDUAL student.
- Compress all files described in the problem statements into one zip or rar.
- Submit the compress file to the course website before the due day. **Warning!** AVOID submit in the last minute. Late submission is not accepted.

Grading Notes

- **Important!** DO remember to include your Verilog code. NO code, NO grades. Also, if your code can not be recompiled by TA successfully using tools in SoC Lab, you receive NO credit.
- Write your report seriously and professionally. Incomplete description and information will reduce your chances to get more credits.
- If extra works (like synthesis, post-simulation or additional instructions) are done, please describe them in your final report clearly for bonus points.
- Please follow course policy.

Deliverables

1. All CPU Verilog codes including components, testbenches and machine code for each lab exercise. NOTE: Please DO NOT include source code in the report!
2. A homework report that includes
    a. A summary in the beginning to state what has been done (such as SMILE CPU, synthesis, post-synthesis simulation, additional branch instruction with verification)
    b. A block diagram for your completed SMILE CPU indicating all necessary components and I/O pins. Note please use MS Visio that is

available in computer center in the university.

c. Simulated waveforms with proper explanation

d. Learned lessons (skills learned, where do you stuck most, exciting things etc.)

3. Please write in MS word and follow the convention for the file name of your report: n26984795_蕭育書_hw1_report.doc

4. All homework requirements should be upload in this file hierarchy. Your file name/folder name is labeled in color red, specifications in color black.

N2601xxxx (Your student ID number)(Folder)
- N2601xxxx.doc (Your homework report)
- P3(Folder for Problem 3)
  - top.v (top module, use "`include" to include all files related)
  - top_tb.v (testbench, DO NOT include any file)
  - Any other Verilog files for your design
- P4(Folder for Problem 4)
  - top.v (top module, use "`include" to include all files related)
  - top_tb1.v (testbench with given instructions, DO NOT include any file)
  - top_tb2.v (testbench with your instructions, DO NOT include any file)
  - Any other Verilog files for your design
- P5(Folder for Problem 5)(bonus)
  - top.v (top module, use "`include" to include all files related)
  - top_tb.v (testbench, DO NOT include any file)
  - Any other Verilog files for your design
  - Any other files to support your testbench

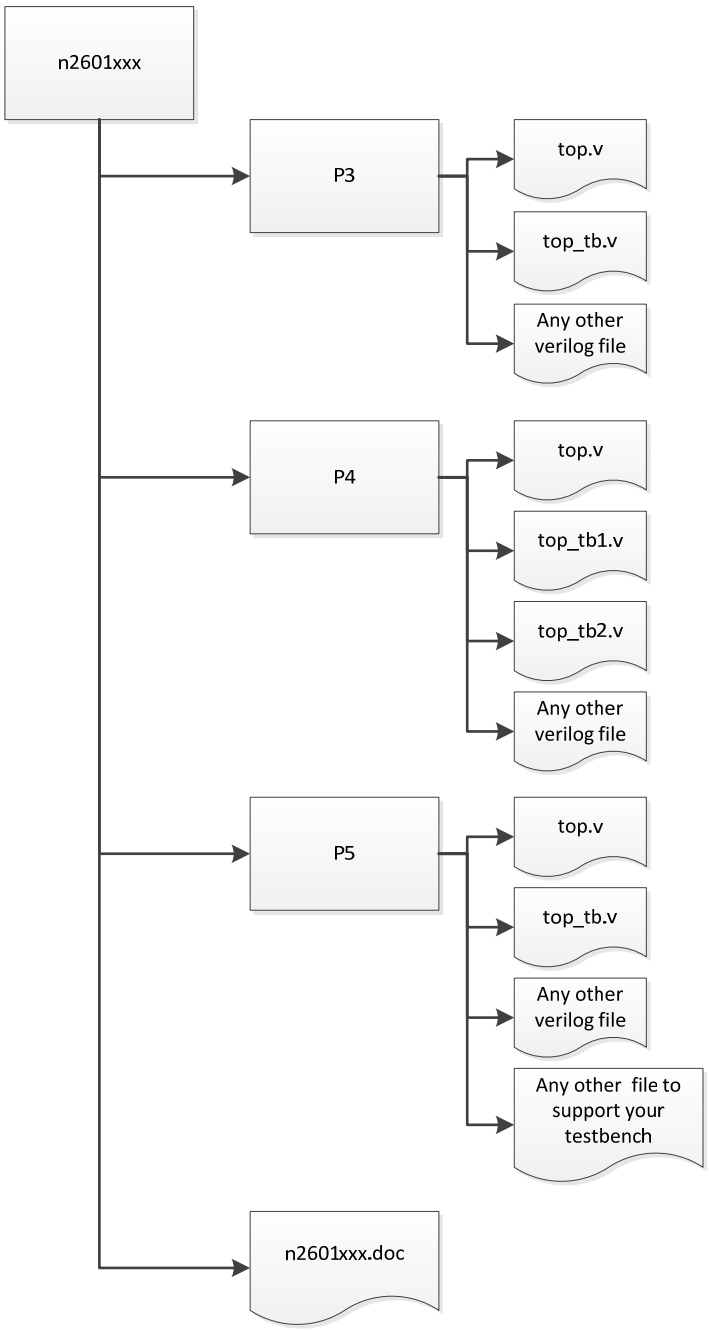


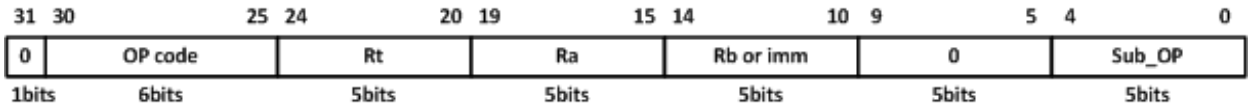
Fig. 3 File hierarchy for Homework submission

Exercise

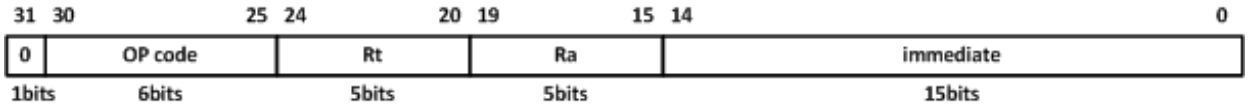
We just use some of the instruction format in this homework.

☞Data-processing (SE:sign-extened, ZE:zero-extened)

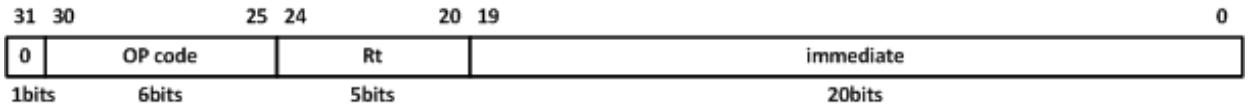
*** Note that NOP and SRLI has the same OP + sub OP. This is durable. If you exam their DST, SRC1 and SRC2, you will be able to find their differences and find a proper way to implement.



OP code	Mnemonics	DST	SRC1	SRC2	Sub OP	Description
100000	NOP	00000	00000	00000	01001	No operation
100000	ADD	$Rt	$Ra	$Rb	00000	Rt = Ra + Rb
100000	SUB	$Rt	$Ra	$Rb	00001	Rt = Ra - Rb
100000	AND	$Rt	$Ra	$Rb	00010	Rt = Ra & Rb
100000	OR	$Rt	$Ra	$Rb	00100	Rt = Ra | Rb
100000	XOR	$Rt	$Ra	$Rb	00011	Rt = Ra ^ Rb
100000	SRLI	$Rt	$Ra	imm	01001	Shift right : Rt = Ra >> imm
100000	SLLI	$Rt	$Ra	imm	01000	Shift left : Rt = Ra << imm
100000	ROTRI	$Rt	$Ra	imm	01011	Rotate right : Rt = Ra >> imm



OP code	Mnemonics	DST	SRC1	SRC2	Description
101000	ADDI	$Rt	$Ra	imm	Rt = Ra + SE(imm)
101100	ORI	$Rt	$Ra	imm	Rt = Ra | ZE(imm)
101011	XORI	$Rt	$Ra	imm	Rt = Ra ^ ZE(imm)



OP code	Mnemonics	DST	SRC1	Description
100010	MOVI	$Rt	imm	Rt = SE(immediate)

3. **(30/100) Write and verify an ALU which is from exercise 2 in part I combined with a register file as shown in Fig. 4. The 32-bit register file contains 32 registers. This register is shown in Fig. 5 consists of two Read ports and two data output ports (for source register and destination register) and one Write Address Port and one Write Data Port. It shall have the following functions:**
   - **Working at positive edge of clock**
   - **When reset, data values in all registers are reset to zero**
   - **When enable is high and write is asserted, the write_data is stored into the register as pointed by write_address**
   - **When enable is high and read is asserted, the data that are stored in registers as pointed by read_address1 & read_address2 are written to src1 & src2**
   - **Your design MUST follow the module specifications in Table. 1 .**
   - **Your top module MUST follow the top module specification as indicated in the testbench skeleton code, specifically the module name and I/O ports.**
   - **Some skeleton codes are attached for your reference. Most likely, you need to modify the code to make them work as specified.**

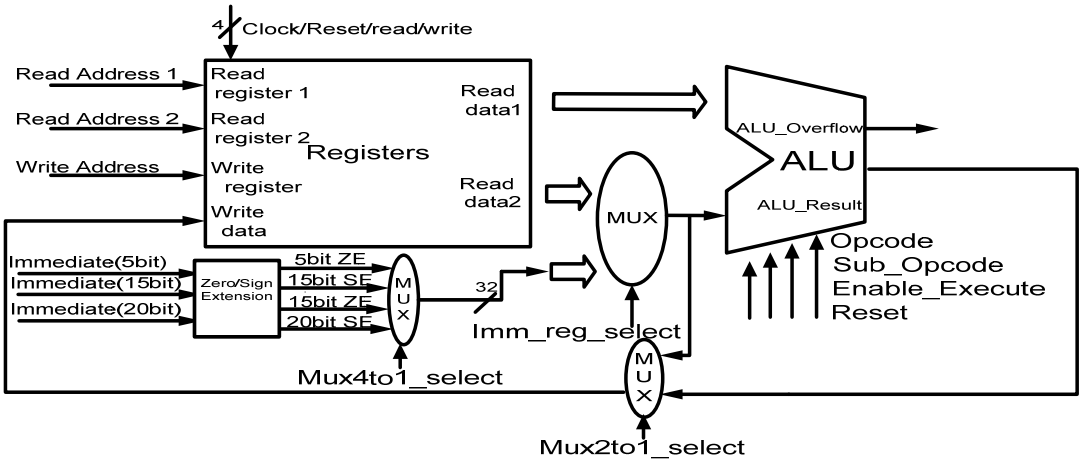


Fig. 4 ALU + Register File

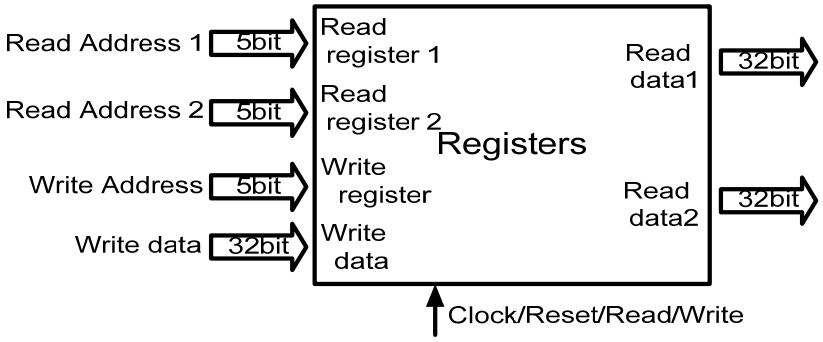


Fig. 5 Register File

Module Type	Specifications
Mux4to1 (for different immediate data selection)	Mux4to1_select： 0: 5bit (ZE) 1:15 bit(SE) 2: 15bit (ZE) 3: 20bit (SE) (SE:sign-extened, ZE:zero-extened)
imm_reg_select_mux (for ALU source selection)	When imm_reg_select is 1, select imm_out When imm_reg_select is 0, select read_data
writeback_sel_mux (for write data selection)	When writeback_select is 1, select imm_data When writeback_select is 0, select alu_result
Register file	The variable for storing data of your register file module MUST be declared as “reg [31:0] rw_reg [31:0]”

Top module	module top(clk, rst,
	//Register
	read_address1,
	read_address2,
	write_address,
	enable_fetch,
	enable_writeback,
	//imm_sel
	imm_5bit,
	imm_15bit,
	imm_20bit,
	mux3to1_select,
	mux2to1_select,
	imm_reg_select,
	//ALU

	enable_execute, opcode, sub_opcode, //OUTPUT alu_overflow); regfile regfile1(…) /* other sub-modules */

Table. 1 Module Specifications

Report Requirements

a. Proper explanation of your design is required for full credits.
b. A figure (block diagram with logic gates) shall be draw to depict your design in the end.
c. Verify your code with a testbench. These testbenches need to show all major situations. The testbench skeleton code is also attached below for your reference.
d. Show your snapshot of waveform for different cases in your reports and illustrate the correctness of your results.
e. Use nLint to analyze your code, report the final results and 3~5 most frequent warning/errors in your code. Describe how you modify your code to comply with the nLint requirements.

Reference code for the register file

```
module regfile(read_data1, read_data2, read_address1, read_address2,
                write_address, write_data, clk, reset, read, write);

  parameter DataSize = 32;
  parameter AddrSize = 5;

  output reg [DataSize-1:0]read_data1;
  output reg [DataSize-1:0]read_data2;

  input [AddrSize-1:0]read_address1;
  input [AddrSize-1:0]read_address2;
  input [AddrSize-1:0]write_address;
  input [DataSize-1:0]write_data;
  input clk, reset, read, write;

  reg [DataSize-1:0]rw_reg[31:0];
  integer i;

  always@(posedge clk, posedge reset)begin
    if(reset)begin
      for(i=0;i<32;i=i+1)
        rw_reg[i]<=32'b0;
    end
    else begin
      if(read)begin
        read_data1<=rw_reg[read_address1];
        read_data2<=rw_reg[read_address2];
      end
      else if(write)begin
        rw_reg[write_address]<=write_data;
      end
      else begin
        read_data1<=32'b0;
        read_data2<=32'b0;
      end
    end
  end
endmodule
```

Skeleton code for the testbench

```
1  `timescale 1ns/10ps
2
3  module top_tb;
4
5    parameter DataSize = 32;
6    parameter AddrSize = 5;
7
8    reg clk,rst;
9    //Register
10   reg [AddrSize-1:0]read_address1;
11   reg [AddrSize-1:0]read_address2;
12   reg [AddrSize-1:0]write_address;
13   reg enable_fetch;
14   reg enable_writeback;
15   //imm_sel
16   reg [4:0]imm_5bit;
17   reg [14:0]imm_15bit;
18   reg [19:0]imm_20bit;
19   reg [1:0]mux3to1_select;
20   reg mux2to1_select;
21   reg imm_reg_select;
22   //ALU
23   reg enable_execute;
24   reg [5:0] opcode;
25   reg [4:0] sub_opcode;
26   //OUT
27   wire alu_overflow;
28   integer i,err_num;
29   //test &debug
30   reg [DataSize-1:0]golden_reg[31:0];
31
32 top TOP(
33   clk,
34   rst,
35   //Register
36   read_address1,
37   read_address2,
38   write_address,
39   enable_fetch,
40   enable_writeback,
41   //imm_sel
42   imm_5bit,
43   imm_15bit,
44   imm_20bit,
45   mux3to1_select,
46   mux2to1_select,
47   imm_reg_select,
48   //ALU
49   enable_execute,
50   opcode,
```

```
51    sub_opcode,
52    //OUTPUT
53    alu_overflow
54  );
55
56    //clock gen.
57    always #5 clk=~clk;
58
59    initial begin
60    clk=0;
61    rst=1'b0;
62  //IDLE
63    //Register
64    read_address1='d0;
65    read_address2='d0;
66    write_address='d0;
67    enable_fetch=1'b0;
68    enable_writeback=1'b0;
69    //imm_sel
70    imm_5bit='d0;
71    imm_15bit='d0;
72    imm_20bit='d0;
73    mux3to1_select=2'b0;
74    mux2to1_select=1'b0;
75    imm_reg_select=1'b0;
76    //ALU
77    enable_execute=1'b0;
78    opcode='b0;
79    sub_opcode='d0;
80
81    #5 rst=1'b1;
82    #5 rst=1'b0;
83
84  //TEST MOVEI Reg[0]=200(Dec)
85    //Register
86    read_address1='d0;
87    read_address2='d0;
88    write_address='d0;
89    enable_fetch=1'b0;
90    enable_writeback=1'b1;
91    //imm_sel
92    imm_5bit='d0;
93    imm_15bit='d0;
94    imm_20bit='d200;
95    mux3to1_select=2'b11;
96    mux2to1_select=1'b1;
97    imm_reg_select=1'b1;
98    //ALU
99    enable_execute=1'b0;
100   opcode='d0;
```

```
101     sub_opcode='d0;
102
103
104   //TEST ADDI Reg[1]=Reg[0]+100(Dec)  read_cycle
105     //Register
106   #10 read_address1='d0;
107     read_address2='d0;
108     write_address='d0;
109     enable_fetch=1'b1;
110     enable_writeback=1'b0;
111     //imm_sel
112     imm_5bit='d0;
113     imm_15bit='d100;
114     imm_20bit='d0;
115     mux3to1_select=2'b1;
116     mux2to1_select=1'b0;
117     imm_reg_select=1'b1;
118     //ALU
119     enable_execute=1'b1;
120     opcode='b101000;
121     sub_opcode='d0;
122
123   //TEST ADDI Reg[1]=Reg[0]+100(Dec)  writeback_cycle
124     //Register
125   #10 read_address1='d0;
126     read_address2='d0;
127     write_address='d1;
128     enable_fetch=1'b0;
129     enable_writeback=1'b1;
130     //imm_sel
131     imm_5bit='d0;
132     imm_15bit='d100;
133     imm_20bit='d0;
134     mux3to1_select=2'b1;
135     mux2to1_select=1'b0;
136     imm_reg_select=1'b1;
137     //ALU
138     enable_execute=1'b1;
139     opcode='b101000;
140     sub_opcode='d0;
141
142   //IDLE
143   #10 read_address1='d0;
144     read_address2='d0;
145     write_address='d0;
146     enable_fetch=1'b0;
147     enable_writeback=1'b0;
148     //imm_sel
149     imm_5bit='d0;
150     imm_15bit='d0;
```

```
151     imm_20bit='d0;
152     mux3to1_select=2'b0;
153     mux2to1_select=1'b0;
154     imm_reg_select=1'b0;
155     //ALU
156     enable_execute=1'b0;
157     opcode='b0;
158     sub_opcode='d0;
159
160     initial begin
161     $fsdbDumpfile("top.fsdb");
162     $fsdbDumpvars;
163     end
164 endmodule
```

4. (40/100) Design a controller unit that is able to decode an input instruction as shown in Fig. 6 and control the datapath unit designed in Problem 3 to carry out the designated operation. A reference state diagram is attached in Fig. 7 for your reference. Develop a testbench that includes the instructions with the format as shown in Fig. 8 and verify the overall system (controller + ALU + register file). You MUST follow the module specifications in Table. 2.

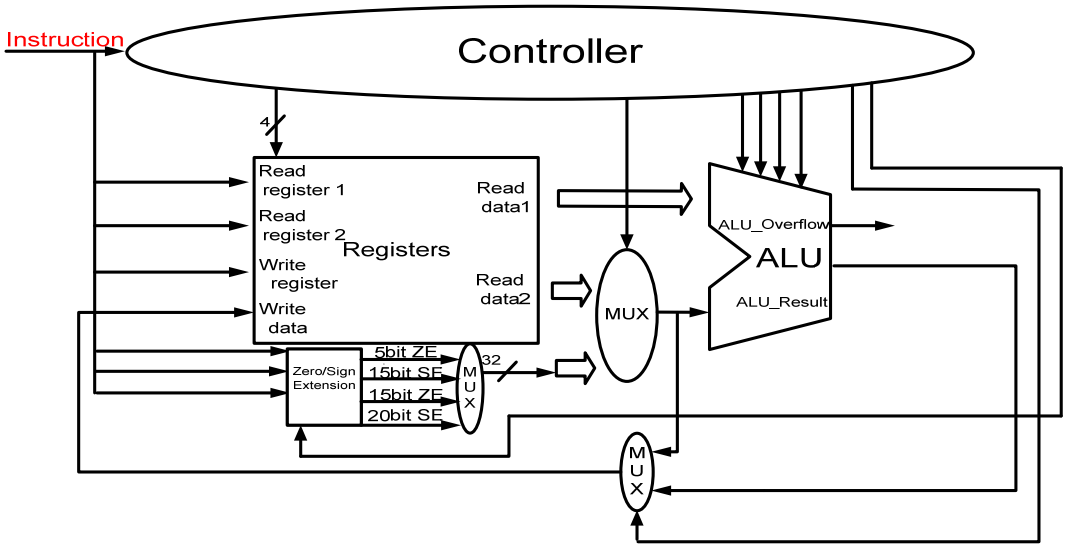


Fig. 6 Block diagram of a controller

The controller shall have the following features:

- All operations initiated at the positive edge trigger
- Control signals:
  - enable_fetch: To fetch and store the designated data from instruction memory into the register file and prepare them to be ready for execution
  - enable_execute: To order ALU to execute the operation
  - enable_writeback: To store the ALU result back to the register file or data memory
- 4 different operating states for the controller, i.e. stop operation, fetch, execute and write-back

Its state diagram of the controller is as shown in Fig. 6.

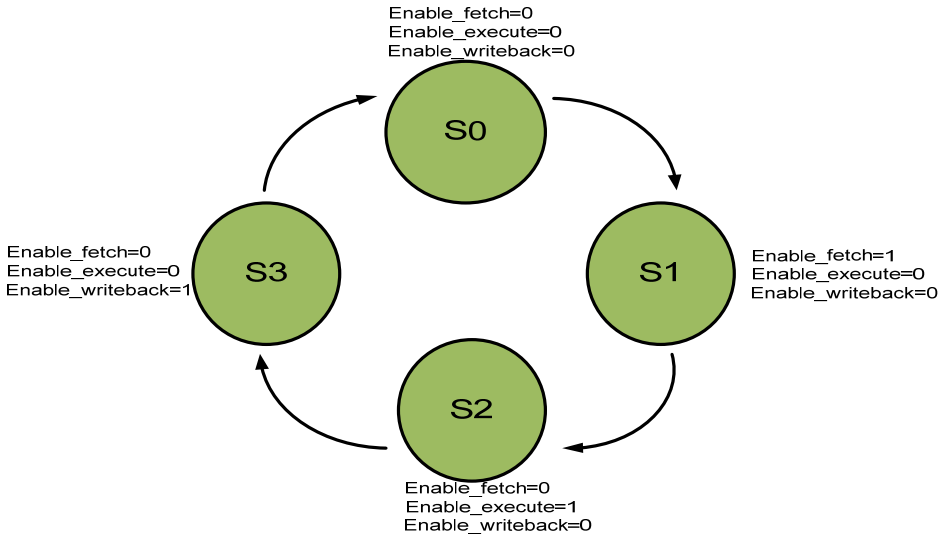


Fig. 7 State Diagram of the controller

```
0_101000_00000_00000_0000_0000_0001_101; //ADDI   R0 = R0 +5'b01101
0_101000_00001_00001_0000_0000_0001_100; //ADDI   R1 = R1 +5'b01100
0_100010_00010_0000_0000_0000_0001_0000; //MOVI   R2 = 5'b01000
0_100000_00011_00000_00001_00000_00000;  //ADD    R3 = R0 + R1
0_100000_00100_00000_00001_00000_00001;  //SUB    R4 = R0 - R1
0_100000_00101_00011_00100_00000_00010;  //AND    R5 = R3 & R4
0_100000_00110_00011_00100_00000_00100;  //OR     R6 = R3 | R4
0_100000_00111_00011_00100_00000_00011;  //XOR    R7 = R3 ^ R4
0_100000_01000_00000_00100_00000_01000;  //SLLI   R8 = R0<<5'b00100
0_100000_01001_00001_01000_00000_01011;  //ROTORI R9 = R1>>5'b01000
0_101100_00000_00000_0000_0000_0011_111; //ORI    R0 = R0 | 5'b11111
0_101011_00001_00001_0000_0000_0010_101; //XORI   R1 = R1 + 5'b10101
```

Fig. 8 Input instruction in binary format for verifying Problem 4

Report Requirements

a. Proper explanation of your design is required for full credits.
b. A figure (block diagram with logic gates) shall be draw to depict your design in the end.
c. Write a testbench that includes the instructions with the format as shown in Fig. 7 and verify your code. Show your snapshot of waveform for different cases in your reports and illustrate the correctness of your results.
d. Write another testbench and verify your code. These testbenches need to show all major situations. You will need to write your own instructions to test your code. Show your snapshot of waveform for different cases in your reports and illustrate the correctness of your results.
e. Use nLint to analyze your code, report the final results and 3~5 most frequent warning/errors in your code. Describe how you modify your code to comply with the nLint requirements.

Module Type	**Specifications**
Top Module	top(instruction, clk, reset); /* Ports and specifations */ Regfile regfile1(…) /* other sub-modules */
Register file	The internal variable for storing data of your register file module MUST be declaired as “reg [31:0] rw_reg [31:0]”

Table. 2 Module Specifications

Skeleton code of the controller

```
// concroller
`define OPCODE  ir[30:25]
`define SUBOPCODE ir[4:0]
`define SRLI 5'b01001
`define SLLI 5'b01000
`define ROTRI 5'b01011

module controller(enable_execute, enable_fetch, enable_writeback, opcode, sub_opcode,
                  mux3to1_select, mux2to1_select, imm_reg_select, clock, reset, PC, ir);
  input clock;
  input reset;
  input [31:0] PC;
  input [31:0] ir;

  output reg enable_execute;
  output reg enable_fetch;
  output reg enable_writeback;
  output [5:0] opcode;
  output [4:0] sub_opcode;
  output reg mux3to1_select;
  output reg mux2to1_select;
  output reg imm_reg_select;

  wire [5:0] opcode = ir[30:25];
  wire [4:0] sub_opcode = ir[4:0];
  reg [1:0] current_state;
  reg [1:0] next_state;
  reg [31:0] present_instruction;

  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

  always @(posedge clock)
  begin
    if(reset)
      current_state = S0;
    else
      current_state = next_state;
  end

  always @(current_state)
  begin
    case(current_state)
    S0 : begin
      next_state = S1;
      enable_fetch = 0;
      enable_execute = 0;
      enable_writeback = 0;
      mux3to1_select = 0;
      mux2to1_select = 0;
      imm_reg_select = 0;
         end
        .
        .
        .
    endcase
  end

  always @(posedge enable_fetch)
  begin
    if(PC == 0)
      present_instruction = 0;
    else
      present_instruction = ir;
  end

endmodule
```

5. **(Bonus 20/100) Write a testbench that is able to distinguish faults for the design in problem 4. The testbench needs to show the results as PASS or FAIL in any format at the end of execution. You will gain more credits if your testbench is able to print more details about errors, such as location in code, time in the simulation waveform, discrepancy in values, etc.**

   Report Requirements

   a. Proper explanation of your testbench is required for full credits. You must prove your testbench is able to distinguish between PASS and FAIL by showing results.
   b. Show us how good your testbench is by verifying the testbench with a good design and an intentionally faulted design. Demonstrate your snapshot of waveforms for two cases in your report and illustrate the correctness of your results. Also explain the waveform for the faulty design and how your testbench distinguish it as a FAIL design.
   c. If your testbench is capable of printing more detail information about the errors, show a snapshot of result for detail information. Also explain how your testbench compare between correct outputs and incorrect outputs and how to print error information.
   d. List any extra condition for your design to work properly and explain the reason.

// *Reference* //

☞SMILE CPU has the following instruction format (Andes ISA)

The Andes 32bit instruction formats and the meaning of each filed are described below:

- Type-0 Instruction Format

0	Opc_6	{sub_1, imm_24}

- Type-1 Instruction Format

0	Opc_6	rt_5	imm_20	
0	Opc_6	rt_5	sub_4	imm_16

- Type-2 Instruction Format

0	Opc_6	rt_5	ra_5	imm_15
0	Opc_6	rt_5	ra_5	{sub_1, imm_14}

- Type-3 Instruction Format

0	Opc_6	rt_5	ra_5	rb_5	sub_10
0	Opc_6	rt_5	ra_5	imm_5	sub_10

- Type-4 Instruction Format

0	Opc_6	rt_5	ra_5	rb_5	rd_5	sub_5
0	Opc_6	rt_5	ra_5	imm1_5	imm2_5	sub_5

- opc_6: 6-bit opcode
- rt_5: target register in 5-bit index register set
- ra_5: source register in 5-bit index register set
- rb_5: source register in 5-bit index register set
- rd_5: destination register in 5-bit index register set
- sub_10: 10-bit sub-opcode
- sub_5: 5-bit sub-opcode
- sub_4: 4-bit sub-opcode
- sub_1: 1-bit sub-opcode
- imm_24: 24-bit immediate value, for unconditional jump instructions (J, JAL). The
- immediate value is used as the lower 24-bit offset of same 32MB memory block (new
- PC[31:0] = {current PC[31:25], imm_24, 1'b0}
- imm_20: 20-bit immediate value. Sign-extended to 32-bit for MOVI operations.
- imm_16: signed PC relative address displacement for branch instructions.
- imm_15: 15-bit immediate value. Zero extended to 32-bit for unsigned operations,
- while sign extended to 32-bit for signed operations.
- imm_14: signed PC relative address displacement for branch instructions.
- imm_5, imm1_5, imm2_5: 5-bit unsigned count value or index value

//