VLSI System Design (Graduate Level)
Fall 2010

HOMEWORK II

` Due day: midnight Nov. 7 (Sunday), 2010

This homework is to let you familiar with Verilog and SMILE CPU. It includes the second part of a simplified multi-cycle **(SMILE)** CPU with 17 instructions. You will complete the CPU. It shall be noted that some problems have reference code. They are for your reference. 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

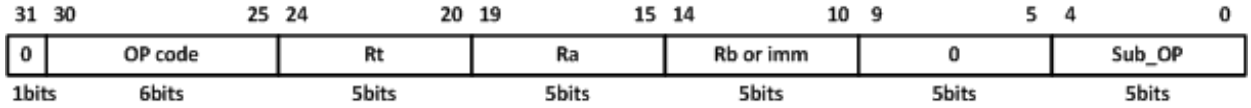
d. Learned lesson and Conclusion

3. Please write in MS word and follow the convention for the file name of your report: n09299992_蕭育書_hw2_report.doc

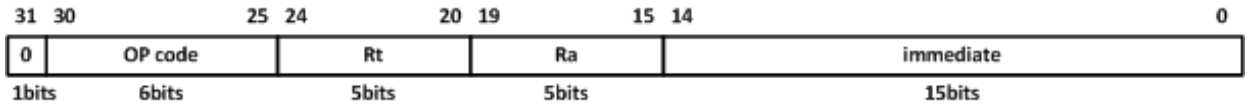
Exercise

SMILE CPU has the following instruction format

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

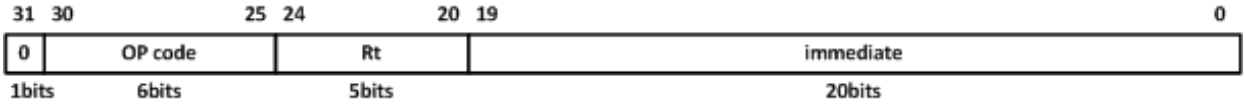


OP code	Mnemonics	DST	SRC1	SRC2	Sub OP	Description
100000	NOP	00000	00000	00000	01001	No operation
100000	ADD	$Rt	$Ra	$Rb	00000	Rt = Ra + Rb
100000	SUB	$Rt	$Ra	$Rb	00001	Rt = Ra - Rb
100000	AND	$Rt	$Ra	$Rb	00010	Rt = Ra & Rb
100000	OR	$Rt	$Ra	$Rb	00100	Rt = Ra | Rb
100000	XOR	$Rt	$Ra	$Rb	00011	Rt = Ra ^ 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



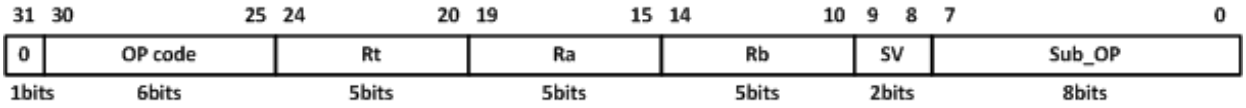
OP code	Mnemonics	DST	SRC1	SRC2	Description
101000	ADDI	$Rt	$Ra	imm	Rt = Ra + SE(imm)
101100	ORI	$Rt	$Ra	imm	Rt = Ra | ZE(imm)
101011	XORI	$Rt	$Ra	imm	Rt = Ra ^ ZE(imm)
000010	LWI	$Rt	$Ra	imm	Rt = Mem[Ra + (imm << 2)]
001010	SWI	$Rt	$Ra	imm	Mem[Ra + (imm << 2)] = Rt

HOMEWORK II



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

☞Load and store



OP code	Mnemonics	DST	SRC1	SRC2	Sub OP	Description
011100	LW	$Rt	$Ra	$Rb	00000010	Rt = Mem[Ra + (Rb << sv)]
011100	SW	$Rt	$Ra	$Rb	00001010	Mem[Ra + (Rb << sv)] = Rt

1. **(20 points) Add in a 32x32bit instruction memory into the CPU system that designed in HWI(part 2) exercise 4. Please load the following machine code into the instruction memory and modified the controller so that the CPU system could read the sequence of the machine codes from the instruction memory and implement the corresponding operations. The machine code is as below:**

```
0_100010_00000_00000000000000000100      //MOVI   R0=20'd4
0_101000_00000_00000_000000000001101     //ADDI   R0=R0 + 4'b1101
0_101100_00001_00000_000000000000010     //ORI    R1=R0 | 4'b0010
0_101011_00001_00001_000000000000111     //XORI   R1=R1 ^ 4'b0111
0_100000_00000_00000_00000_00000_01001   //NOP
0_100000_00001_00000_00001_00000_00000   //ADD    R1=R0 + R1
0_100000_00001_00001_00000_00000_00001   //SUB    R1=R1 - R0
0_100000_00001_00001_00000_00000_00010   //AND    R1=R1 & R0
0_100000_00000_00000_00001_00000_00100   //OR     R0=R0 | R1
0_100000_00010_00000_00001_00000_00011   //XOR    R2=R0 ^ R1
0_100000_00010_00000_00011_00000_01001   //SRLI   R2=R0 SRL(3)
0_100000_00010_00010_00011_00000_01000   //SLLI   R2=R2 SLL(3)
0_100000_00010_00000_11101_00000_01011   //ROTRI  R2=R0 ROTR(29)
```

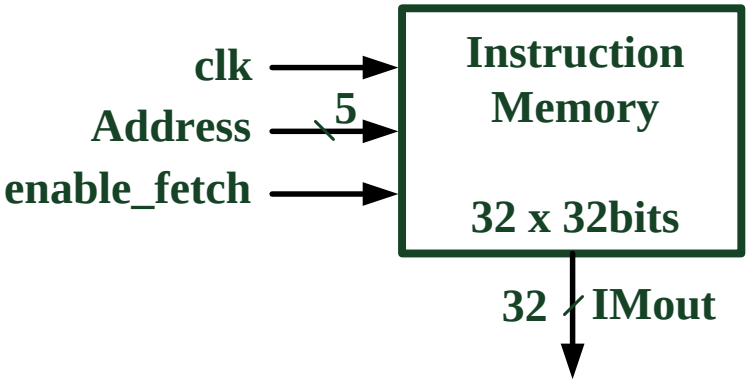


Fig. 1 Conceptual Diagram of Instruction Memory

You can load the machine code by the following procedures:

- Added the following block into the testbench to load mins.prog into Instruction memory
  - initial
  - begin : prog_load
    $readmemb("mins.prog",u_memory_Instruction.mem_data1);
  - end
- The system task $readmemb (reads in binary) and $readmemh (reads in hex) will read in a file.
- It takes 2 arguments, the file name and the memory structure name. It reads the file into the memory structure.

Reference code for instruction memory :

```
module IM(clk,IM_address,enable_fetch,IMout);
input clk,enable_fetch;
input [4:0]IM_address;
output [31:0]IMout;

wire [4:0]IM_address; // from PC
reg  [31:0]IMout;
reg  [31:0]mem_data1[31:0];


always@(posedge clk)
begin
     if(enable_fetch)begin
       IMout=mem_data1[IM_address];
     end
end

endmodule
```

HOMEWORK II

2. (20 points) Add in a 128x8bit byte addressing data memory into the CPU system. And modify your register file become three read ports as show in Fig.3. Please load the mins.prog(you have to convert it to machine code first by yourself) in the next page into the instruction memory and modified the CPU system so that it could execute the 17 instructions as show in the table at the beginning.

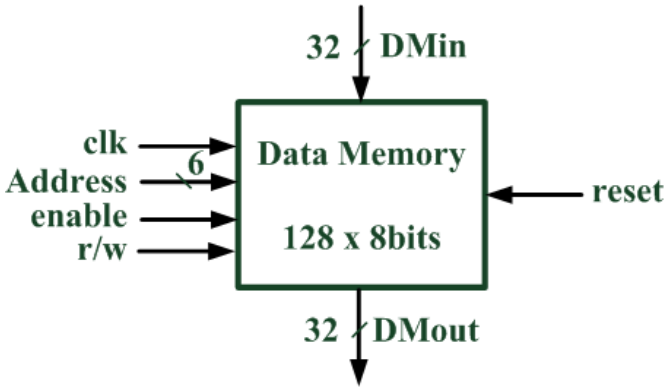


Fig. 2 Conceptual Diagram of Data Memory

Reference code for data memory :

```
module DM(clk,rst,enable,read_write,DMin,DMout,DM_address);

parameter data_size=32;
parameter mem_width=8;
parameter addr_size=7;
parameter mem_size=128;

input clk;
input rst;
input enable;
input read_write;
input  [data_size-1:0]DMin;
input  [addr_size-1:0]DM_address; // address of DM  = Ra + (Rb << sv)
output [data_size-1:0]DMout;
reg [data_size-1:0]DMout;
reg [mem_width-1:0]mem_data[mem_size-1:0];


integer i;
always@(posedge clk)
begin
   if(rst)begin
       for(i=0;i<mem_size;i=i+1)
         mem_data[i]=0;
       DMout=0;
   end

   else if(enable==1)begin
      if(read_write==1)
        DMout={mem_data[DM_address],mem_data[DM_address+1],mem_data[DM_address+2],mem_data[DM_address+3]};
      else
        {mem_data[DM_address],mem_data[DM_address+1],mem_data[DM_address+2],mem_data[DM_address+3]}=DMin;
   end
end

endmodule
```

mins.prog content :

★0. MOVI R0=20'd3
★1. SW M0=R0
★2. LW R1=M0
★3. ADDI R1=R1+ 4'b1001
★4. ORI R0=R0 | 4'b0100
★5. XORI R1=R1^ 4'b1010
★6. NOP
★7. ADD R1=R0 + R1
★8. SUB R1=R1 – R0
★9. AND R1=R1 & R0
★10. SW M20=R1
★11. LW R2=M20
★12. OR R2=R1 | R0
★13. XOR R2=R2 ^ R1
★14. SRLI R2=R0 SRL(2)
★15. SLLI R2=R2 SLL(4)
★16. ROTRI R2=R0 ROTR(30)
★17. SW M56=R2
★18. LW R3=M56
★19. SUB R3=R3 - R0
★20. AND R1=R1 & R3

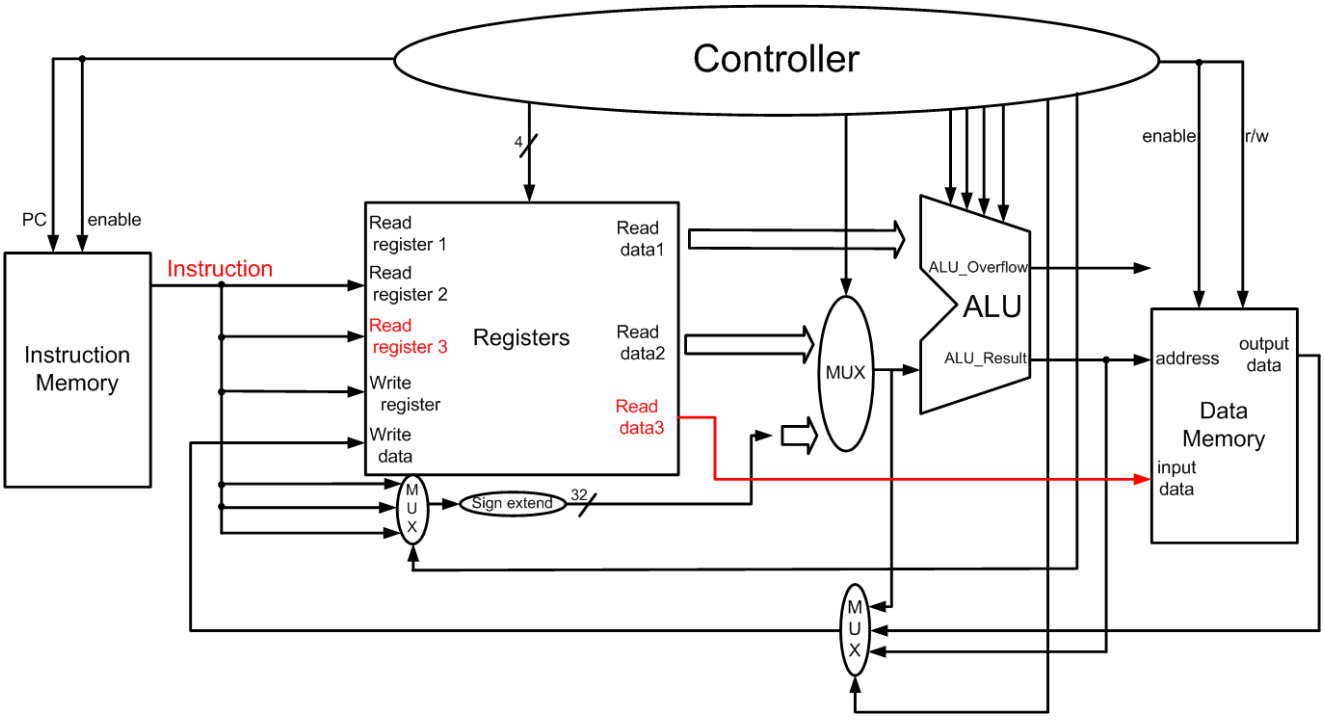


Fig. 3 Block diagram of multi-cycle (**SMILE**) CPU

HOMEWORK II

// *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

//