VLSI System Design (Graduate Level)
Fall 2010

HOMEWORK II

` Due day: midnight Nov. 18 (Thursday), 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 19 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

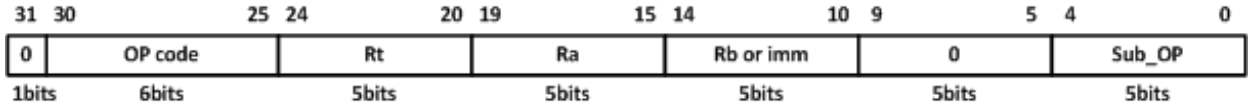
d. Learned lesson and Conclusion

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

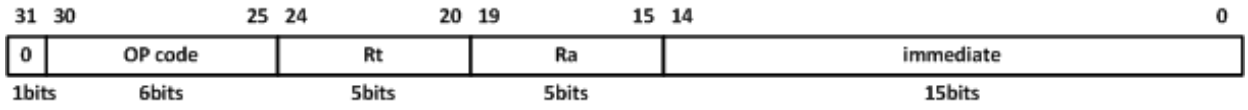
Exercise

SMILE CPU has the following instruction format

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

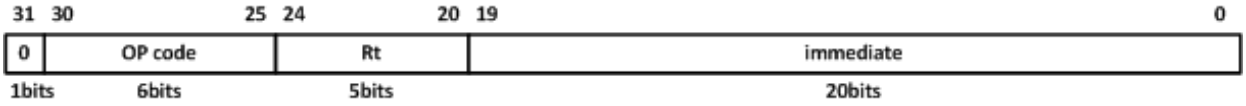


OP code	Mnemonics	DST	SRC1	SRC2	Sub OP	Description
100000	NOP	00000	00000	00000	01001	No operation
100000	ADD	$Rt	$Ra	$Rb	00000	Rt = Ra + Rb
100000	SUB	$Rt	$Ra	$Rb	00001	Rt = Ra - Rb
100000	AND	$Rt	$Ra	$Rb	00010	Rt = Ra & Rb
100000	OR	$Rt	$Ra	$Rb	00100	Rt = Ra | Rb
100000	XOR	$Rt	$Ra	$Rb	00011	Rt = Ra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



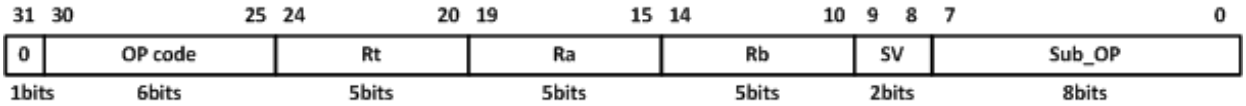
OP code	Mnemonics	DST	SRC1	SRC2	Description
101000	ADDI	$Rt	$Ra	imm	Rt = Ra + SE(imm)
101100	ORI	$Rt	$Ra	imm	Rt = Ra | ZE(imm)
101011	XORI	$Rt	$Ra	imm	Rt = Ra ^ ZE(imm)
000010	LWI	$Rt	$Ra	imm	Rt = Mem[Ra + (imm << 2)]
001010	SWI	$Rt	$Ra	imm	Mem[Ra + (imm << 2)] = Rt

HOMEWORK II



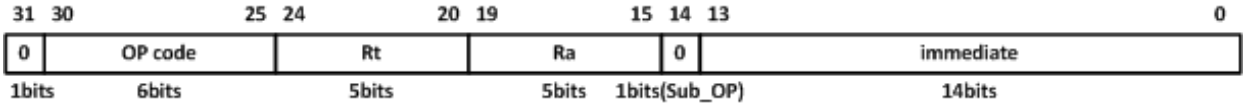
OP code	Mnemonics	DST	SRC1	Description
100010	MOVI	$Rt	imm	Rt = SE(immediate)

☞Load and store



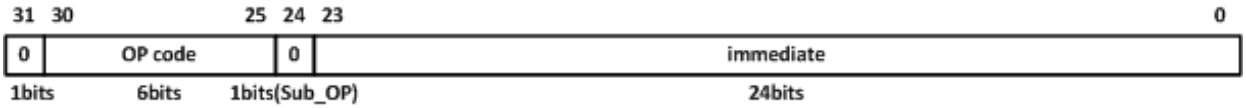
OP code	Mnemonics	DST	SRC1	SRC2	Sub OP	Description
011100	LW	$Rt	$Ra	$Rb	00000010	Rt = Mem[Ra + (Rb << sv)]
011100	SW	$Rt	$Ra	$Rb	00001010	Mem[Ra + (Rb << sv)] = Rt

☞Branch (SE:sign-extened)



OP code	Mnemonics	DST	SRC1	Sub OP	Description
100110	BEQ	$Rt	$Ra	0	if(Rt == Ra) PC = PC + SE(imm << 1) else PC = PC + 4

☞Jump (SE:sign-extened)



OP code	Mnemonics	Sub OP	Description
100100	J	0	PC = PC + SE(imm << 1)

VLSI System Design (Graduate Level)
Fall 2010

HOMEWORK II

1. **(60 points) Write and verify SMILE CPU based on the specification.**
   - i. **Implement the 19 instructions as listed**
   - ii. **Redesgin the FSM for the controller**
   - iii. **Register File size: 32x32bits**
   - iv. **Assume Instruction Memory and Data Memory with no delay**
   - v. **Instruction memory size:100Mx32bit**
   - vi. **Data memory size: 256Kx8bits**
   - vii. **timescale 1ns/10ps**
   - viii. **Clock period: 20ns**

At least perform the following CPU verification.

a) Execute the "mins.prog"

★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		

b) Write a machine code to carry out the following operations and show that your answers are correct.

- i. **130 + 760**
- ii. **1-65536**
- iii. **255*(-7)**
- iv. **40*2 - 160/4**
- v. Read a data array, whose size is 5, from data memory and restore them back with the sorted order in the same data array. Please give 3 examples to show that the machine code is working fine.

2. (100 points) Used AndeSight to run the following C/C++ code and extract the machine code from the tool. Then execute those machine codes in your CPU system and verify the result of your CPU system with AndeSight. If yours is wrong, please modify your CPU system until your result meets the result of AndeSight.

a)

```
#include <iostream>
using namespace std;

int main()
{
    int A, B, C, D, E;

    A = 170;
    B = 240;
    C = 15;
    D = 85;

    E = A & B;
    E = E | C;
    E = E ^ D;

    cout << "E = " << E << endl;
    return 0;
}
```

b)

```
#include <iostream>
using namespace std;

int main()
{
	unsigned int L, M, N, X, Y, Z;

	L = 200;
	M = 184;
	N = 311;

	X = L >> 5;
	Y = M << 3;
	Z = 0;

	if(X != 5) {
		Y = Z - Y;
	}
	goto logic;

back:
	Z--;

	cout << "X = " << X << endl;
	cout << "Y = " << Y << endl;
	cout << "Z = " << Z << endl;
	return 0;

logic:
	Z = X ^ Y;
	Z = Z | N;
	goto back;
}
```

3. (Bonus 100 points) Complete a working synthesized CPU design. You need to demonstrate your design is working through the same testbenches for pre-synthesis design. Note that instruction memory and data memory do not need to be synthesized.

// *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

//