VE370 HW2 5937910084 Wang Lan

1. (5 points) Following memory location has address 0x0F000000 and content 0x15C78933.

	0	1	2	3
0x0F000000	33	89	C7	15

Write RISC-V assembly instructions to load the byte C7 as a signed number into register x20, then show the content of x20 after the operations.

```
1 .data:
2 a: .word 0x15c78933
3
4 .text:
5 main:
6     lw x5 a
7     lui x6 0x0f000
8     sw x5 0(x6)
9     lb x20 2(x6)
```

x20	s4	0xffffffc7

2. (10 points) The RISC-V assembly program below computes the factorial of a given input n (n!). The integer input is passed through register x12, and the result is returned in register x10. In the assembly code below, there are a few errors. Correct the errors.

```
FACT: addi sp, sp, 8
      sw x1, 4(sp)
      sw x12, 0(sp)
      add x18, x0, x12
      addi x5, x0, 2
      bge x12, x5, L1
      mul x10, x18, x10
      addi sp, sp, -8
      jalr x0, 0(x1)
L1:   addi x12, x12, -1
      jal x1, FACT
      addi x10, x0, 1
      lw x12, 4(sp)
      lw x1, 0(sp)
      addi sp, sp, -8
      jalr x0, 0(x1)
```

```
10 FACT:
11     addi sp, sp, 8
12     sw x1, 4(sp)
13     sw x12, 0(sp)
14     addi x5, x0, 2
15     bge x12, x5, L1
16     addi x10, x0, 1
17     addi sp, sp, -8
18     jalr x0, x1(0)
19
20 L1:
21     add x18, x0, x12
22     addi x12, x12, -1
23     jal x1, FACT
24     mul x10, x18, x10
25     lw x12, 0(sp)
26     addi x18 x12 1
27     lw x1, 4(sp)
28     addi sp, sp, -8
29     jalr x0, x1(0)
```

3. (10 points) Consider a proposed new instruction named `rpt`. This instruction combines a loop's condition check and counter decrement into a single instruction. For example,

```
rpt x29, loop
```

would do the following:

```
if (x29 > 0) {
    x29=x29-1;
    goto loop;
}
```

1) (5 points) If this instruction were to be added to the RISC-V instruction set, what is the most appropriate instruction format?

2) (5 points) What is the shortest sequence of RISC-V instructions that performs the same operation?

1) B-type

2)
```
LOOP: ble  x29 x0  EXIT
      addi x29 x29 -1
      bgt  x29 x0  LOOP
EXIT: ...
```

4. (7 points) Given a 32-bit RISC-V machine instruction:

```
1111 1111 0110 1010 0001 1010 1110 0011
```

1) (6 points) What does the assembly instruction do?

2) (1 point) What type of instruction is it?

1) opcode: 1100011 funct3: 001 ⇒ bne

immediate = $(111111111010)_2 = -6$ ⇒ target PC = current PC − 12

rs1 = $(10100)_2 = 20$ ⇒ x20

rs2 = $(10110)_2 = 22$ ⇒ x22

⇒ if x20 != x22, go to (current PC − 12)

2) B-type

5. (6 points) Given RISC-V assembly instruction:

```
lw x21, -32(sp)
```

1) (5 points) What is the corresponding binary representation?

2) (1 point) What type of instruction is it?

1) immediate $= (-32)_{10} = 111111100000$

rs1 = sp = x2 $\Rightarrow$ 00010

rd = x21 $\Rightarrow$ 10101

funct3 : 010

opcode : 0000011

$\Rightarrow$ 1111 1110 0000 0001 0010 1010 1000 0011

2) I-type

6. (12 points) If the RISC-V processor is modified to have 128 registers rather than 32 registers:

1) (4 points) show the bit fields of an R-type format instruction assuming opcode and func fields are not changed.

2) (4 points) What would happen to the I-type instruction if we want to keep the total number of bits for an instruction unchanged?

3) (4 points) What is the impact on the range of addresses for a `beq` instruction? Assume all instructions remain 32 bits long and the size of opcode and func fields don't change.

1)

funct7	rs2	rs1	funct3	rd	opcode
7bits	7bits	7bits	3bits	7bits	7bits

2) Only 8 bits are remained for immediate value

3) Before : [current PC - 4096 , current PC + 4094]

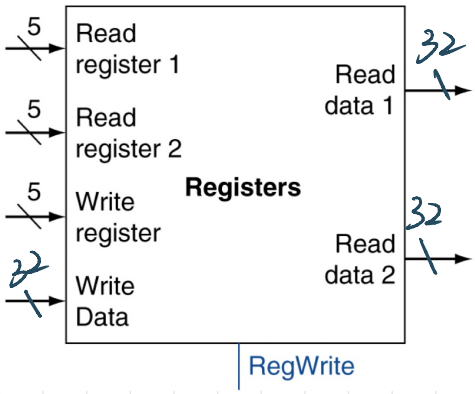
Now : [current PC - 256 , current PC + 254]

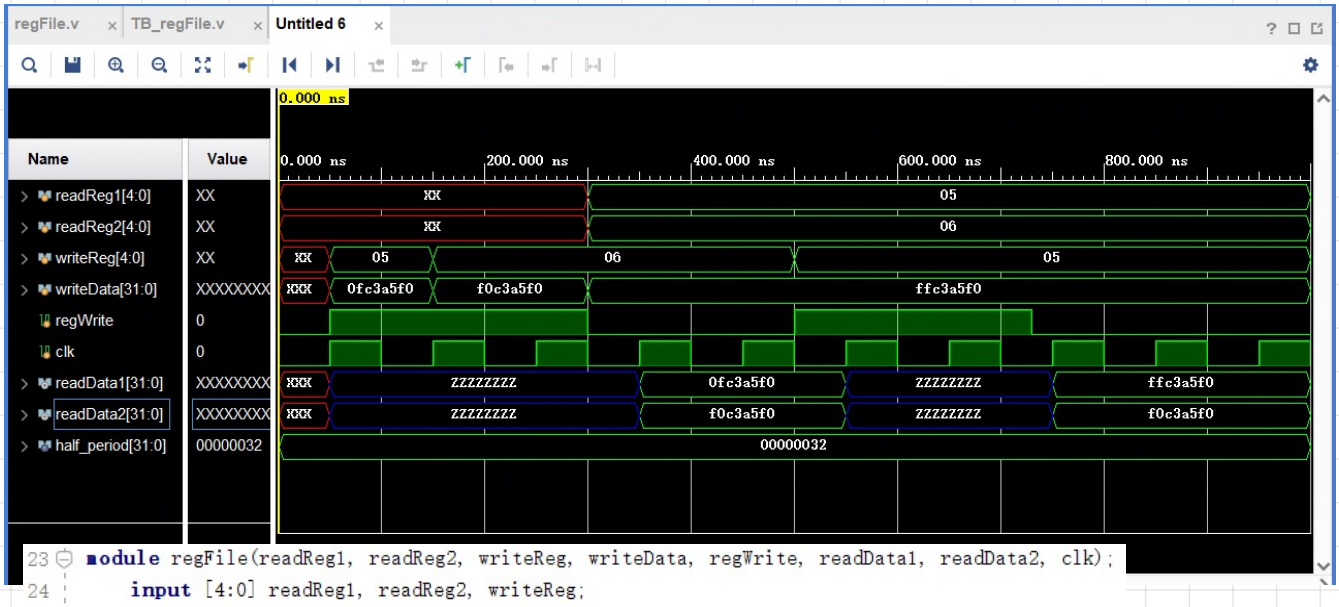
7. (15 points) Convert the following assembly code fragment into machine code, assuming the memory location of the first instruction (LOOP) is 0x1000F400

```
0x1000f400 LOOP:    blt x0, x5, ELSE
                    jal x0, DONE
0x1000f408 ELSE:    addi x5, x5, -1
                    addi x25, x25, 2
                    jal x0, LOOP
0x1000f414 DONE:    ...
```

0000 000 0 0101 0000 0100 1000 0110 0011
0000 0000 1000 0000 0000 0000 0110 1111
1111 1111 1111 0010 1000 0010 1001 0011
0000 0000 0010 1100 1000 1100 1001 0011
1111 1111 1001 1111 1111 0000 0110 1111

8. (15 points) Model the Register File component shown below in Verilog HDL. Show source code and screen shots of simulation results.



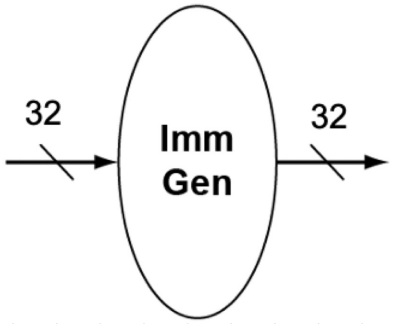


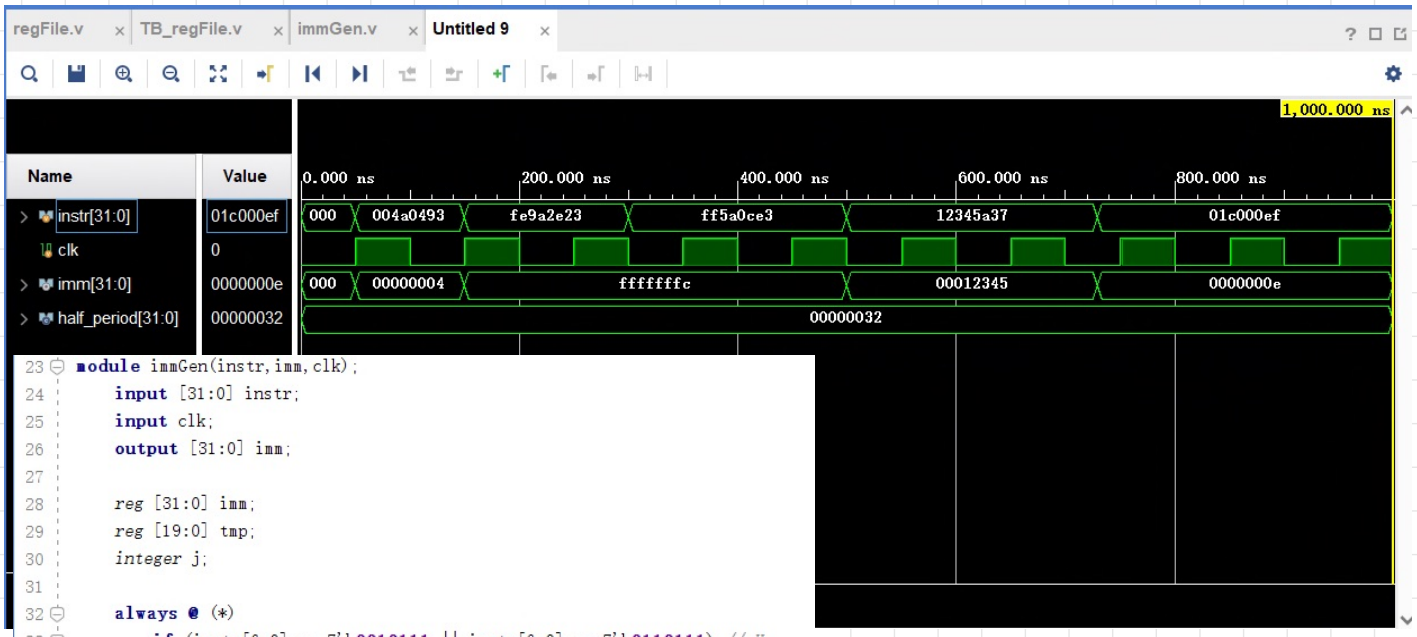

```
module regFile(readReg1, readReg2, writeReg, writeData, regWrite, readData1, readData2, clk);
    input [4:0] readReg1, readReg2, writeReg;
    input [31:0] writeData;
    input regWrite, clk;
    output [31:0] readData1, readData2;

    reg [31:0] readData1, readData2;
    reg [31:0] regs [31:0];

    always @ (posedge clk) begin
        readData1 = 32'bz;
        readData2 = 32'bz;
        if (regWrite) regs[writeReg] = writeData;
        else begin
            readData1 = regs[readReg1];
            readData2 = regs[readReg2];
            end
    end
endmodule
```

9. (20 points) Model the following Immediate Generator component in Verilog HDL. Show source code, and simulation results of one instruction for each type involving immediate numbers.






```
module immGen(instr,imm,clk);
    input [31:0] instr;
    input clk;
    output [31:0] imm;

    reg [31:0] imm;
    reg [19:0] tmp;
    integer j;

    always @ (*)
        if (instr[6:0] == 7'b0010111 || instr[6:0] == 7'b0110111) // U
        begin
            for(j=0;j<20;j=j+1)
                imm[j]=instr[j+12];
            for(j=20;j<32;j=j+1)
                imm[j]=0;
        end
        else if (instr[6:0] == 7'b1101111) // J
        begin
            imm[19]=instr[31];
            imm[9:0]=instr[30:21];
            imm[10]=instr[20];
            imm[18:11]=instr[19:12];
            for(j=20;j<32;j=j+1)
                imm[j]=imm[19];
        end
        else if (instr[6:4] == 3'b000 || instr[6:4] == 3'b001 || instr[6:4] == 3'b111 || instr[6:0] == 7'b1100111) // I
        begin
            imm[11:0]=instr[31:20];
            for(j=12;j<32;j=j+1)
                imm[j]=imm[11];
        end
        else if (instr[6:4] == 3'b010) // S
        begin
            imm[11:5]=instr[31:25];
            imm[4:0]=instr[11:7];
            for(j=12;j<32;j=j+1)
                imm[j]=imm[11];
        end
        else if (instr[6:4] == 3'b110) // B
        begin
            imm[11]=instr[31];
            imm[9:4]=instr[30:25];
            imm[3:0]=instr[11:8];
            imm[10]=instr[7];
            for(j=12;j<32;j=j+1)
                imm[j]=imm[11];
        end
endmodule
```