JOINT INSTITUTE
交大密西根学院

Ve370 Introduction to Computer Organization

Homework 2

楊毅文 519370910053

Assigned: September 30, 2021

Due: 2:00pm on October 12, 2021

Submit a PDF file on Canvas

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.

```
lui s1, 0x0F000

lb x20, 2(s1)
```

Content in x20: 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
      addi x10, x0, 1
      lw x12, 0(sp)
      lw x1, 4(sp)
```

```
        addi sp, sp, 8
        jalr x0, 0(x1)
L1:     addi x12, x12, -1
        jal x1, FACT
        # addi x10, x0, 1
        lw x12, 0(sp)
        mul x10, x12, x10
        lw x1, 4(sp)
        addi sp, sp, 8
        jalr x0, 0(x1)
```

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 themost appropriate instruction format?

I-type.

Immediate[11:0]	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

Imm[] stores the index of `loop`, opcode and funct3 decides it is a `rpt` function, `rs1` and `rd` are the same as `x29`.

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

```
bgt x29, x0, loop
loop:
    addi x29, x29, -1
    ……
```

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?

```
bne x20, x22, -12
```

If x20 not equals to x22, jump to instruction which is at PC - 12.

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

B-type.

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

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

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

```
1111 1110 0000 0001 0010 1010 1000 0011
```

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

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.

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

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?

Immediate[7:0]	rs1	funct3	rd	opcode
8 bits	7 bits	3 bits	7 bits	7 bits

The immediate number should be within 8 bits if opcode and funct3 are not changed.

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.

beq is B-type, if rs1 and rs2 are changed to 7 bits, then immediate should be shortened to 8 bits in total, which is -256~252; The range of address will be immediate * 2, PC – 512 ~ PC + 508.

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

```
LOOP:     blt x0, x5, ELSE
          jal x0, DONE
ELSE:     addi x5, x5, -1
          addiu x25, x25, 2
          jal x0, LOOP
DONE:     …
```

```
0000 0000 0101 0000 0100 0100 0110 0011
0000 0001 0000 0000 0000 0000 0110 1111
1111 1111 1111 0010 1000 0010 1001 0011
0000 0000 0010 1100 1000 1100 1001 0011
1111 1111 0001 1111 1111 0000 0110 1111
```

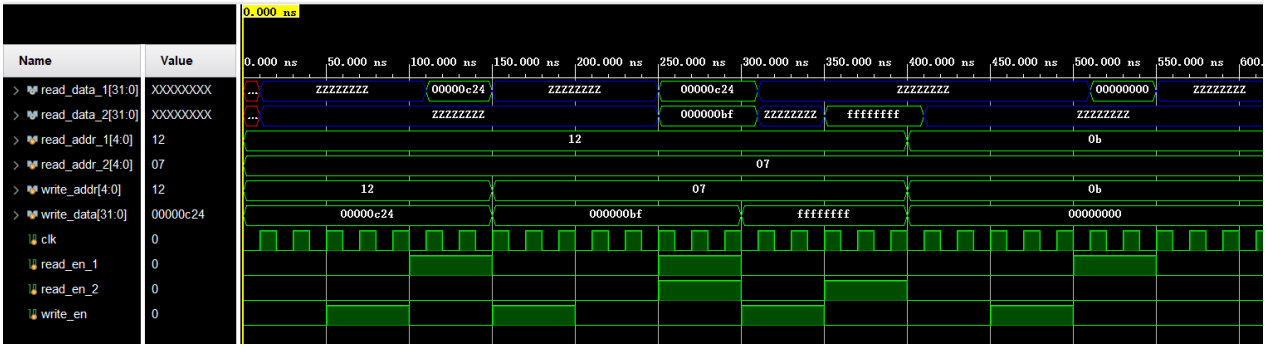
8. (15 points) Model the Register File component shown below in Verilog HDL. Show source code and screen shots of simulation results.


```
module Reg32bit (

   input clk,

   input read_en_1,

   input read_en_2,

   input write_en,

   input [4:0] read_addr_1,

   input [4:0] read_addr_2,

   input [4:0] write_addr,

   input [31:0] write_data,

   output reg [31:0] read_data_1,

   output reg [31:0] read_data_2
```

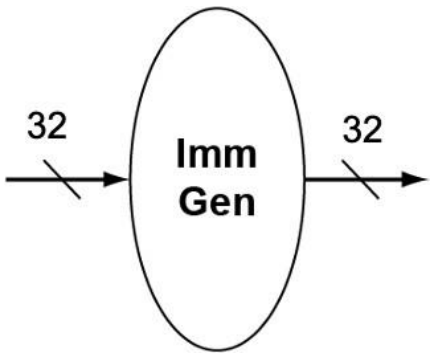
```
    );
        reg [31:0] register [31:0];
        always @(posedge clk) begin
            read_data_1 = 'bz;
            read_data_2 = 'bz;
            if (write_en)
                register[write_addr] = write_data;
            else begin
                if (read_en_1)
                    read_data_1 = register[read_addr_1];
                if (read_en_2)
                    read_data_2 = register[read_addr_2];
            end
        end
endmodule : Reg32bit


module sim(

    );
    wire [31:0] read_data_1, read_data_2;
    reg [4:0] read_addr_1, read_addr_2, write_addr;
    reg [31:0] write_data;
    reg clk, read_en_1, read_en_2, write_en;
    Reg32bit uut (clk, read_en_1, read_en_2, write_en,
read_addr_1, read_addr_2, write_addr, write_data,
read_data_1, read_data_2);
```

```
initial begin
    #0  clk = 0;
        read_addr_1 = 5'b10010;
        read_addr_2 = 5'b00111;
        write_data = 32'd3108;
        read_en_1 = 0;
        read_en_2 = 0;
        write_en = 0;
        write_addr = 5'b10010;
    #50 write_en = 1;
    #50 write_en = 0;
        read_en_1 = 1;
    #50 read_en_1 = 0;
        write_en = 1;
        write_addr = 5'b00111;
        write_data = 32'd191;
    #50 write_en = 0;
    #50 read_en_1 = 1;
        read_en_2 = 1;
    #50 read_en_1 = 0;
        read_en_2 = 0;
        write_en = 1;
        write_data = 32'hffffffff;
    #50 write_en = 0;
        read_en_2 = 1;
    #50 read_en_2 = 0;
        read_addr_1 = 5'b01011;
```

```
            write_addr = 5'b01011;
            write_data = 32'b0;
        #50 write_en = 1;
        #50 write_en = 0;
            read_en_1 = 1;
        #50 read_en_1 = 0;
    end
    initial begin
        forever #10 clk = ~ clk;
    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 ImmGen32bit (
    input [31:0] instruction,
    output reg [31:0] immediate

);
    always @(*)
        case (instruction[6:0])
            // R-type
            7'b0110011: assign immediate = 'bz;
            // I-type
            7'b0000011: assign immediate =
{{20{instruction[31]}}, instruction[31:20]};
            7'b0001111: assign immediate =
{{20{instruction[31]}}, instruction[31:20]};
            7'b0010011: assign immediate =
{{20{instruction[31]}}, instruction[31:20]};
            7'b1100111: assign immediate =
{{20{instruction[31]}}, instruction[31:20]};
            7'b1110011: assign immediate =
{{20{instruction[31]}}, instruction[31:20]};
            // S-type
            7'b0100011: assign immediate =
{{20{instruction[31]}}, instruction[31:25],
instruction[11:7]};
            // B-type
            7'b1100011: assign immediate =
{{20{instruction[31]}}, instruction[31], instruction[7],
instruction[30:25], instruction[11:8]};
            // U-type
```

```
            7'b0010111: assign immediate =
{{12{instruction[31]}}, instruction[31:12]};
            7'b0110111: assign immediate =
{{12{instruction[31]}}, instruction[31:12]};
            // J-type
            7'b1101111: assign immediate =
{{12{instruction[31]}}, instruction[31], instruction[19:12],
                instruction[20],
                instruction[30:21]};
            default: assign immediate = 'bz;
        endcase
endmodule : ImmGen32bit

module sim(

    );
    reg [31:0] instruction;
    wire [31:0] immediate;
    ImmGen32bit uut1 (instruction, immediate);
    initial begin
        #0 instruction =
32'b00000000110000000000100100110011; // add x18, x0, x12
        #50 instruction =
32'b11111110000000010010101010000011; // lw x21, -32(sp)
        #50 instruction =
32'b00000000000100010010010000100011; // sw x1, 8(sp)
        #50 instruction =
32'b00000000101010000100010001100011; // blt x0, x5, 8
        #50 instruction =
```

```
32'b00010000000000000000010100110111; // lui x10, 0x10000

        #50 instruction =
32'b11111111000111111111000001101111; // jal x0, -16

    initial

        #300 $stop;

endmodule.
```