查子未　　520370910037

1. lui x5, 0x0F000
   lb x18, 1(x5)

content of x18: 0xfffffffe

JOINT INSTITUTE
交大密西根学院

ECE3700J Introduction to Computer Organization

Homework 2

Assigned: September 29, 2022

Due: 2:00pm on October 13, 2022

Submit a PDF file on Canvas

1. (5 points) Following memory location has address 0x0F000000 and content 0x18D5FE00.

	0	1	2	3
0x0F000000	00	FE	D5	18

Write RISC-V assembly instructions to load the byte FE as a signed number into register x18, then show the content of x18 after the operations.

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      -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 x10, x0, 1
      addi sp, sp, -8     8
      jalr x0, 0(x1)
L1:   addi x12, x12, -1
      jal x1, FACT
      addi x10, x0, 1     addi x6, x10, 0
      lw x12, 4(sp)       0
      lw x1, 0(sp)        4
      addi sp, sp, -8     8
      jalr x0, 0(x1)      mul x10, x10, x6
```

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?

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

```
1111 1111 1110 1010 0010 1010 1110 1111
```

1) (6 points) What is the corresponding assembly instruction? What's its operation(s)?
2) (1 point) What type of instruction is it?

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

```
sw x21, -16(sp)
```

1) (5 points) What is the corresponding binary representation?
2) (1 point) What type of instruction is it?

6. (12 points) If the RISC-V processor is modified to have 64 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.

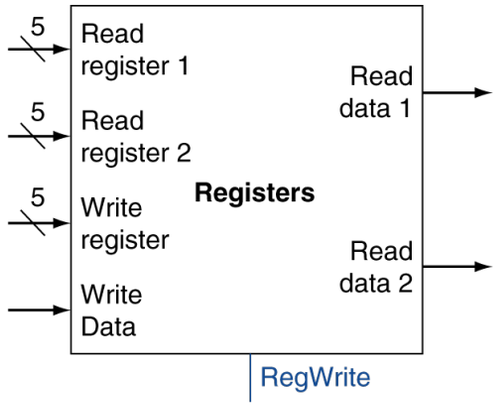
2) (4 points) What would happen to the I-type instruction if we want to keep the total number of bits for an instruction unchanged?

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 funct fields don't change.

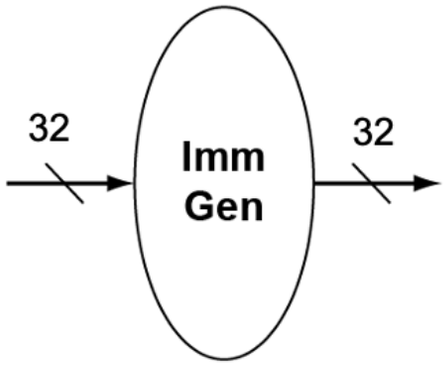
7. (15 points) Convert the following assembly code fragment into machine code, assuming the memory location of the first instruction (LOOP) is 0x1000F400

```
LOOP:    bge x0, x5, ELSE
         jal x0, DONE
ELSE:    addi x5, x5, -1
         add x25, x25, x5
         jal x0, LOOP
DONE:    …
```

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



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.



3. 1) B-type

2)
```
addi x5, x0, 1
blt x29, x5, 8
addi x29, x29, -1
jal x1, loop
```

4. 1) jal, x21, Target, immediate = 11101000011111111110

Target address = PC + 11101000011111111110

jump and link to target address at PC-(382976)$_{10}$ / PC-0x2EC01

2) J-type

5. 1) 1111111 10101 00010 010 10000 0100011

2) S-type

6. 1)

funct 7	rs2	rs1	funct3	rd	opcode
7 bits	6 bits	6 bits	3 bits	6bits	7 bits

2) The length of immediate would shrink to 10 bits

3) We only have 10 bits of signed immediate

The branch range will shrink to $-512 \sim 511$

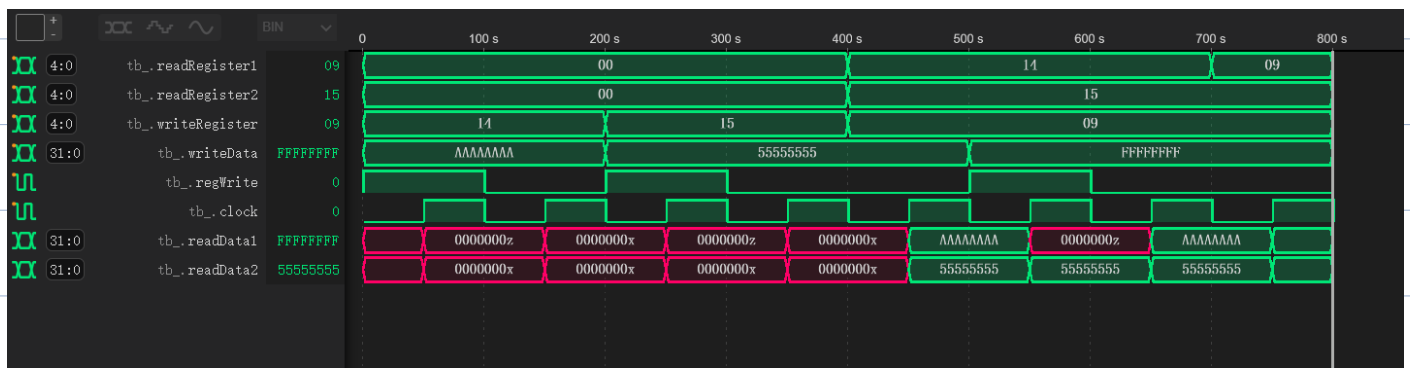
$-1024 - 1022$ away from PC

7.

0x1000F400	0	000000	00101	00000	101	0100 0	1100011
0x1000F404	0	0000000	1000	0	00000000	00000	1101111
0x1000F408	111111111111			00101	000	00101	0010011
0x1000F40C	0000000			00101	11001	000	11001	0110011
0x1000F410	1	1111 11	1000	1	11111111	00000	1101111
0x1000F414							

8.

```
module registerFile (readRegister1, readRegister2, writeRegister, writeData, regWrite, readData1, readData2, clock);
    parameter width = 32;
    parameter addr_width = 5;
    parameter number = 2**addr_width;

    output [width-1:0]      readData1, readData2;
    input  [width-1:0]      writeData;
    input  [addr_width-1:0] readRegister1, readRegister2, writeRegister;
    input                   regWrite, clock;

    reg [width-1:0] readData1, readData2;
    reg [width-1:0] memory [number-1:0];

    always @(posedge clock) begin
        readData1 = 'bz;
        readData2 = 'bz;
        if (regWrite) memory[writeRegister] = writeData;
        else
            readData1 = memory[readRegister1];
            readData2 = memory[readRegister2];
    end
endmodule

module tb_;
parameter width = 32;
parameter addr_width = 5;

wire [width-1:0]      readData1, readData2;
reg  [width-1:0]      writeData;
reg  [addr_width-1:0] readRegister1, readRegister2, writeRegister;
reg                   regWrite, clock;

registerFile RF (
    .readData1 (readData1),
    .readData2 (readData2),
    .writeData (writeData),
    .readRegister1 (readRegister1),
    .readRegister2 (readRegister2),
    .writeRegister (writeRegister),
    .regWrite (regWrite),
    .clock (clock)
);

localparam CLK_PERIOD = 100;
always #(CLK_PERIOD/2) clock=~clock;

initial begin
    $dumpfile("tb_.vcd");
    $dumpvars(0, tb_);
end

initial begin
    #0 clock = 0; regWrite = 1; readRegister1 = 0; readRegister2 = 0; writeRegister = 20; writeData = 32'b10101010101010101010101010101010;
    #100 regWrite = 0;
    #100 regWrite = 1; writeRegister = 21; writeData = 32'b01010101010101010101010101010101;
    #100 regWrite = 0;
    #100 readRegister1 = 20; readRegister2 = 21; writeRegister = 9;
    #100 writeData = readData1 + readData2; regWrite = 1;
    #100 regWrite = 0;
    #100 readRegister1 = 9;
    #100 $finish;
end
endmodule
```



9

```
module immGen (instruct, extend);
    parameter width = 32;
    output reg [width-1:0] extend;
    input  [width-1:0] instruct;
    always @(instruct) begin
       case (instruct[6:0])
        7'b0000011: // I with sign
            if (instruct[14] | !instruct[31]) extend = instruct[31:20];
            else extend = 32'b11111111111111111111100000000000 + instruct[31:20];
        7'b0001111: // I
            if (!instruct[31]) extend = instruct[31:20];
            else extend = 32'b11111111111111111111100000000000 + instruct[31:20];
        7'b0010011: // I with sign
            if (instruct[14:12] == 3'b011 | !instruct[31]) extend = instruct[31:20];
            else extend = 32'b11111111111111111111100000000000 + instruct[31:20];
        7'b0010111: // U
            if (!instruct[31]) extend = instruct[31:12];
            else extend = 32'b11111111111100000000000000000000 + instruct[31:12];
        7'b0100011: // S
            if (!instruct[31]) extend = instruct[11:7] + instruct[31:25]*2**5;
            else extend = 32'b11111111111111111111100000000000 + instruct[11:7] + instruct[31:25]*2**5;
        7'b0110111: // U
            if (instruct[14:12] == 3'b011 | !instruct[31]) extend = instruct[31:12];
            else extend = 32'b11111111111100000000000000000000 + instruct[31:12];
        7'b1100011: // B with sign
            if (!instruct[31] | instruct[14:13] == 2'b11) extend = instruct[31]*2**11 + instruct[7]*2**10 + instruct[30:25]*2**4 + instruct[11:8];
            else extend = 32'b11111111111111111111100000000000 + instruct[31]*2**11 + instruct[7]*2**10 + instruct[30:25]*2**4 + instruct[11:8];
        7'b1100111: // I
            if (!instruct[31]) extend = instruct[31:20];
            else extend = 32'b11111111111111111111100000000000 + instruct[31:20];
        7'b1101111: // J
            if (!instruct[31]) extend = instruct[31]*2**19 + instruct[19:12]*2**11 + instruct[20]*2**10 + instruct[30:21];
            else extend = 32'b11111111111100000000000000000000 +  instruct[31]*2**19 + instruct[19:12]*2**11 + instruct[20]*2**10 + instruct[30:21];
        7'b1110011: // I
            if (!instruct[31]) extend = instruct[31:20];
            else extend = 32'b11111111111111111111100000000000 + instruct[31:20];
        default: extend = 'bz;
       endcase
    end
endmodule

module tb_;
parameter width = 32;

wire [width-1:0]      extend;
reg  [width-1:0]      instruct;

immGen IG (
    .extend (extend),
    .instruct (instruct)
);

initial begin
    $dumpfile("tb_.vcd");
    $dumpvars(0, tb_);
end

initial begin
    #0   instruct = 32'b00000000001001010000010010010011;
    #100 instruct = 32'b11111111110010100010010000011;
    #100 instruct = 32'b11111110100110100010111000100011;
    #100 instruct = 32'b11111111010110100000110011100011;
    #100 instruct = 32'b00010010001101000101101000110111;
    #100 instruct = 32'b00000000111000000000000011101111;
    #100 $finish;
end
endmodule
```