ECE M116C - CS M151B Computer Architecture Systems - UCLA

Homework 1

Deadline: Friday, Oct. 7, 11:55 PM

Fall 2022

(Upload it to Gradescope.)

Q1. What is the machine code for the following instructions:

```
SLL x4, x0, x17

| 0000000 | 10001 | 00000 | 001 | 00100 | 0110011 |
|   fc7   |  rs2  |   rs1 | fc3 |  rd   |  opcode |


XORI x7, x1, -20

| 111111101100 | 00001 |  100 |  00111 | 0010011 |
|  imm[11:0]   |  rs1  |  fc3 |  rd    |  opcode |

-20 in 2's compliment binary is 101100. We sign extend it to get
the 12-bit immediate.


SB x4, x5, 11

| 0000000 |  00101 | 00100 | 000 | 01011  | 0100011 |
|imm[11:5]|   rs2  |  rs1  | fc3 |imm[4:0]|  opcode |

fc3 value dictates if it's an SB or SH or SW instruction.
000 - SB
001 - SH
010 - SW
```

Q2. What is the machine code for the following branch instructions (you don't need to find the machine code for other instructions)? Assume that the first instruction starts at address 0x100 (shown in hex) for each code snippet.

```
   a. beq x4, x0, Loop
      addi x0, x0, 0
      addi x0, x0, 0
      addi x0, x0, 0
      Loop:
         addi x1, x1, 3

|      0|000000| 000000 | 00100 | 000 |   1000 | 0  | 1100011 |
| imm[12|10:5] |  rs2   |  rs1  | fc3 | imm[4:1|11] |  opcode |
```

The beq's jump is pc+16 (jump forward 4 instructions to the 'Loop'), that makes the immediate 16, 010000 in binary which is then 0 extended to 13 bits and ignoring the LSB we get the [12:1] immediate for the instruction.

b. Back:

```
    andi x1, x2, 3
    addi x0, x0, 0
    addi x0, x0, 0
    bge t4, zero, Back
```

```
|      1|111111| 000000 | 11101 | 101 |   1010 | 1  | 1100011 |
| imm[12|10:5] |  rs2   |  rs1  | fc3 | imm[4:1|11] |  opcode |
```

The bge's jump is pc-12 (jump back 3 instructions to 'Back'), that makes the immediate -12, 10100 in binary which is then 1 extended to 13 bits and ignoring the LSB we get the [12:1] immediate for the instruction.

Q3. Translate the following machine codes into RISC-V assembly language (numbers are in hex).

400656b3

```
  0100000   00000   01100   101   01101   0110011
|   fc7   |  rs2  |   rs1 | fc3 |  rd   |  opcode |
```

Regardless of what instruction it is the lowest 7 bits are always opcode. The opcode tells us, this is definitely an R-type, now if you put it to the R-type pattern, Func3 and Func7 tells us it is an SRA instruction. Now just figure out what is rs1, rs2 and rd.

SRA x13,x12,x0

ff402183

111111110100 00000 010 00011 0000011

The opcode and func3 says it's an LW instruction, now figure out rd and rs1 and immediate, the immediate's MSB is 1 so it's a negative number, which turns out to be -12

LW x3, x0, -12

Q4. Write a RISC-V assembly language program for counting the number of zeros ("0") bits in a 32-bit input.

Use the following information in writing your assembly code.

1. The function starts at memory location `0x204`. Each instruction is 32 bits, thus the second instruction should start at `0x208` and the third at `0x20c`, and so on. Use this information to correctly compute the offset for jump and branch instructions (you are not allowed to use labels).
2. The input is passed (stored) in register `a0`.
3. The return value, c, should be stored in `a0`.
4. The return address is stored in `ra`.
5. You are free to use saved and temporary registers (don't forget to save values if you are using saved registers).
6. You are allowed to use Pseudoinstructions (e.g., `ret`, `call`, etc.)

```
0x204:      ADD t0, a0, zero //put the input to some temp register
            ADDI a0, zero, 32 // initialize a0 with 32, will have the result at the end
            ADDI t4, zero, 32 // initialize the loop counter
            ADDI t1, zero, 1 // initialize a register LSB=1 and all other bits=0
0x214:      AND t3, t1, t0 // AND the t1 with input to see if a bit is 1
            BNE t3, t1, 4 // if a bit is 1 then t1 == t3, otherwise skip next instruction
            ADDI a0, zero, -1 // if a bit is confirmed to be 1 so decrement no of 0s
in i/p
            SLLI t1, t1, 1 // move to the next bit of the input
            ADDI t4, t4, -1 // decrement counter
            BNE t4, zero, -20 // see if the loop ran 32 times if not jump to 0x214
            RET //at this point a0 should have the number of 0s in the input so return
```

This solution is a reverse approach, assumes the result as all bits being 0s (a0=32) then decrements the results depending on if a bit is found to be 1. It is done by making a bit of a second temp register as '1' and rest as '0' then ANDing it with the input, the result will be equal to the second register's value only if that particular bit is '1', and doing it for all the bits.

Alternatively, you can also make the second register all 1s but just 1 bit '0', and then running the same loop but incrementing an initial value of 0 to get the number of '0's in the list. Remember to use LUI and ADDI instructions to initialize a register to FFFFFFFE, as ADDI's immediate has a limit of 12 bits. (LUT t1, FFFFF000; ADDI t1, FFE).

You will receive credit if you've used other algorithms too for counting the bits.