Midterm Exam

CSC 252
7 March 2019
Computer Science Department
University of Rochester

Instructor: Yuhao Zhu
TAs: Jessica Ervin, Yu Feng, Max Kimmelman, Olivia Morton, Yawo Alphonse Siatitse, Yiyang Su, Amir Taherin, Samuel Triest, Minh Tran

Name: _____

Problem 0 (2 points):	_____
Problem 1 (15 points):	_____
Problem 2 (16 points):	_____
Problem 3 (14 points):	_____
Problem 4 (14 points):	_____
Problem 5 (14 points):	_____
Total (75 points):	_____
Extra Credit (20 points)	_____

Remember "**I don't know**" is given 15% partial credit, but you must erase everything else. This does not apply to extra credit questions.

Your answers to all questions must be contained in the given boxes. The lengths of the boxes should be more or less indicative of the lengths of your answers. Use spare space to show all supporting work to earn partial credit.

You have 75 minutes to work.

Please sign the following. I have not given nor received any unauthorized help on this exam.

Signature:_____

GOOD LUCK!!!

Problem 0: Warm-up (2 Points)

Who do you usually study CSC 252 with?

Hopefully you have a study group when studying 252. When you are able to clearly explain something to others, that's when you know you really understand it.

Problem 1: Fixed-Point Arithmetics (15 points + 3 points extra credit)
Part a) (4 points) Represent the decimal value 92 in hexadecimal.

0x5c

Part b) (4 points) Represent the binary value 1110110111 in hexadecimal.

0x3b7

Part c) (4 points) Octal is the base-8 number system, and uses the digits 0 to 7. Represent the octal value 273 in binary.

10111011

Part d) (3 points) Suppose the registers **`%esi`**, **`%ebx`**, and **`%edx`** are initialized with the values shown below.

`%esi`	**`%ebx`**	**`%edx`** (before)
0xA2E058	0x800	0x0

What would the value of **`%edx`** be after the instruction: **`lea (%esi, %ebx, 8), %edx`**?

0xA32058

Part e) (3 points extra credit) Let A and B be two unknown 8-bit 2's complement numbers. We know the results of A ^ B and A & B as shown below:

A ^ B	00110100
A & B	11001001

(1 point) What is the sum A + B expressed in the 8-bit two's complement notation?

11000110

On an x86 system, would the carry flag be set after A + B? What about the overflow flag?

Will carry flag be set? **(1 point)**	Yes
Will overflow flag be set? **(1 point)**	No

Problem 2: Floating-Point Arithmetics (16 points + 2 points extra credit)

Part a) (4 points) Put $4\frac{3}{16}$ in binary normalized form.

1.000011 x (2^+2)

Part b) (12 points + 2 points extra credits) In this problem, we assume that IEEE decided to add a new 12-bit representation, with its main characteristics consistent with the other IEEE standards.

In this 12-bit representation, the value 155/256 is represented exactly as 001000110110.

(4 points) How many bits are needed for exponent?

3

(4 points) How many bits are needed for fraction?

8

(4 points) In this 12-bit representation, what is the smallest positive number that can be represented?

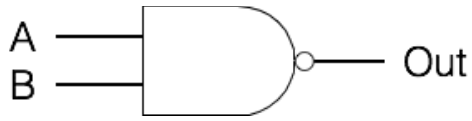
2^-10

(2 points extra credit) You want to calculate the sum of the following three numbers that are represented using this 12-bit floating point format: (A) 110111100100, (B) 010101110000, and (C) 011011000000. Give an order in which the addition will generate the expected sum.

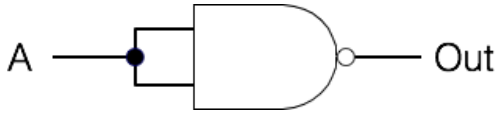
Anything that isn't BCA or CBA.

Problem 3: Logic Design (14 points)

The following is the schematic of a NAND gate, which takes in two 1-bit inputs **`A`** and **`B`**, and generates one 1-bit output **`Out`**. The NAND gate functions in such a way that **`Out = !(A & B)`**.

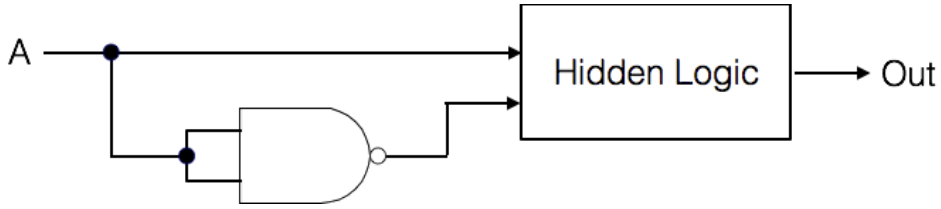


Part a) (3 points) Now we build the following piece of combinational logic using the NAND gate, which takes in one 1-bit input **`A`**, and generates one 1-bit output **`Out`**. What's the relationship between **`Out`** and **`A`**?



Out = !A

Part b) (11 points) We have the combinational circuit shown below with part of its logic hidden. It takes in one 1-bit input: **`A`**, and produces one 1-bit output: **`Out`**. The relationship between **`A`** and **`Out`** is shown in the accompanying truth table.



`A`	**`Out`**
0	0
1	0

(3 points) What is the functionality of the hidden logic in the circuit? You can denote the two inputs to the hidden logic as **`In1`** and **`In2`**.

AND, NOR

(5 points) Implement the hidden logic using only NAND gates. Draw its schematic below.

(IN1, IN2) -> NAND -> IN3 then (IN3, IN3) -> NAND -> OUT

(3 points) Given your above implementation and assuming the delay of a NAND gate is 1ps, what is the delay of the entire combinational circuit?

3ps.

Problem 4: Assembly Programming (14 points + 6 points extra credits)

Below is the assembly code for a mystery function in C. Assume this function takes in an unsigned integer from 1 ~ 8 in **`%edi`** and returns a value to **`%eax`**. The function prototype is the following: **`unsigned int mystery(unsigned int);`**

Assembly code:

```
0x0000000000400556 <+0>:   push   %rbp
0x0000000000400557 <+1>:   mov    %rsp,%rbp
0x000000000040055a <+4>:   sub    $0x10,%rsp

0x000000000040055e <+8>:   mov    %edi,-0x4(%rbp)

0x0000000000400561 <+11>:  cmpl   $0x2,-0x4(%rbp)
0x0000000000400565 <+15>:  jne    0x40056e <mystery+24>
0x0000000000400567 <+17>:  mov    $0x2,%eax
0x000000000040056c <+22>:  jmp    0x4005a5 <mystery+79>

0x000000000040056e <+24>:  cmpl   $0x1,-0x4(%rbp)
0x0000000000400572 <+28>:  jg     0x400583 <mystery+45>
0x0000000000400574 <+30>:  mov    -0x4(%rbp),%eax
0x0000000000400577 <+33>:  sub    $0x1,%eax
0x000000000040057a <+36>:  mov    %eax,%edi
0x000000000040057c <+38>:  callq  0x400556 <mystery>
0x0000000000400581 <+43>:  jmp    0x4005a5 <mystery+79>

0x0000000000400583 <+45>:  cmpl   $0x1,-0x4(%rbp)
0x0000000000400587 <+49>:  jle    0x4005a5 <mystery+79>
0x0000000000400589 <+51>:  mov    -0x4(%rbp),%eax
0x000000000040058c <+54>:  sub    $0x2,%eax
0x000000000040058f <+57>:  mov    %eax,%edi
0x0000000000400591 <+59>:  callq  0x400556 <mystery>
0x0000000000400596 <+64>:  imul   -0x4(%rbp),%eax
0x000000000040059a <+68>:  mov    -0x4(%rbp),%edx
0x000000000040059d <+71>:  sub    $0x1,%edx
0x00000000004005a0 <+74>:  imul   %edx,%eax
0x00000000004005a3 <+77>:  jmp    0x4005a5 <mystery+79>

0x00000000004005a5 <+79>:  leaveq
0x00000000004005a6 <+80>:  retq
```

Part a) (4 points) Assume 2 is stored in `%edi` at the beginning of the function execution. Which lines of assembly will have been executed after the function finishes execution (denote as function offset e.g. <+24>)?

0, 1, 4, 8, 11, 15, 17, 22, 79, 80

Part b) (5 points) This function produces integer over/underflow for some input values. What are these values? Recall that the input is an unsigned integer from 1 ~ 8 stored in `%edi`.

1, 3, 5, 7

Part c) (5 points) For the input values that do not cause integer over/underflow, what does this function return? Please express it as a closed-form function of the input (you could denote the input as x).

x!

Part d) (6 points extra credit) There are some ways to modify this assembly program to make the function work for all input values 1 ~ 8. What is one set of 3 or fewer lines of changes you can make? Note that deletion and replacement are valid changes, insertion is not.

Line# (denote as function offset):

36 OR 15

Change to:

mov 0x01, %eax OR jg 0x400583

Line# (denote as function offset):

38 OR 17

Change to:

delete/ret OR mov-0x4(%rbp), %eax

Line# (denote as function offset):

N/A (hopefully)

Change to:

Problem 5: ISA and Microarchitecture (14 points + 9 points extra credits)

Suppose you are working for a microprocessor company RoCChip. You take on the job of designing the ISA and microarchitecture for a new computer. You want the ISA to have two types of instructions detailed below.

The first type of instructions (Type A) has the following general format:
`Opcode Ra,Rb,Imm`

Instructions of this type operate as follows. We perform an operation between the values in registers **`Ra`** and **`Rb`**, and store the result to the memory address specified by the immediate value **`Imm`**. The immediate value is treated as an absolute (as opposed to relative) memory address. The exact operation to be performed between **`Ra`** and **`Rb`** depends on the specific `Opcode`.

The binary encoding for this type of instructions is the following. The most significant bit is always 0, indicating that that this is a Type A instruction.

0	Opcode	Ra	Rb	Imm

The second type of instructions (Type B) has the following general format:
`Opcode Ra,Rb,Rc`

Instructions of this type operate as follows. We perform an operation between the values in registers **`Ra`** and **`Rb`**, and store the result to the memory address specified by value in register **`Rc`**. Similarly, the exact operation to be performed between **`Ra`** and **`Rb`** depends on the specific `Opcode`.

The binary encoding for this type of instructions is the following. The most significant bit is always 1, indicating that that this is a Type B instruction.

1	Opcode	Ra	Rb	Rc

You want your machine to be byte addressable (i.e., each addressable memory location is one byte, just like x86). The total memory capacity is 2^{20} Bytes. Each register in this machine hold 4 bytes of data.

For each instruction type, you plan to support 8 different arithmetic and logic operations (same 8 for each type). You also have the limitation that the length of Type A instructions must be 4 bytes.

Part a) (3 points) What is the minimum number of bits to represent the **`Opcode`** field in both types of instructions?

3

Part b) (3 points) What is the minimum number of bits to represent the **`Imm`** field in Type A?

20

Part c) (4 points) Using the number of bits for **`Opcode`** and **`Imm`** that you came up with in (a) and (b), what is the maximum number registers that your ISA can support?

16

Part d) (4 points) Assume that we have a program with 1000 instructions. 20% of them are of Type A, and 80% of Type B. How much space in the memory is occupied by this program?

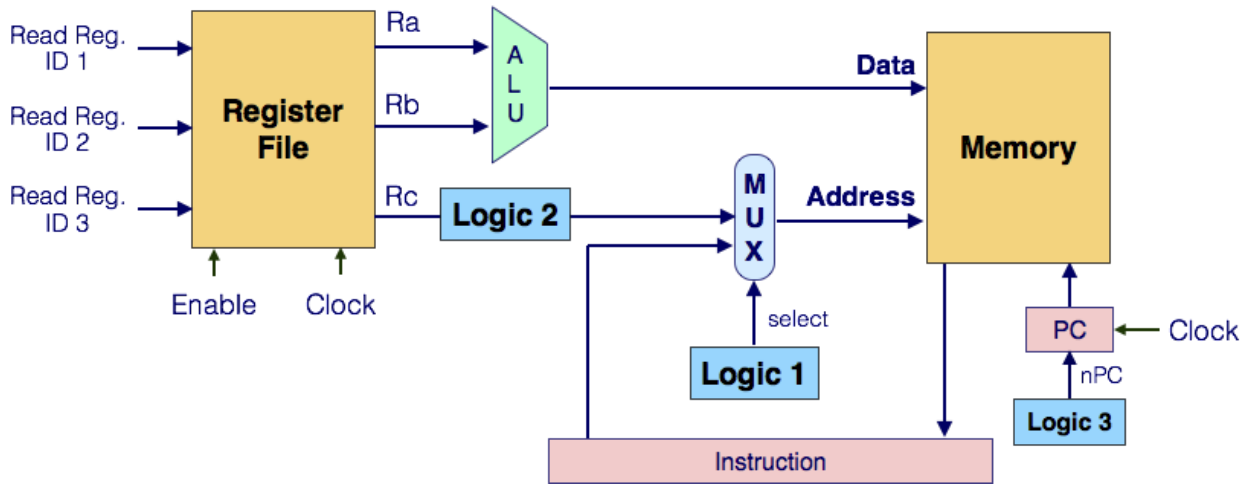
200*32 + 800*16 = 19,200 bits = 2,400 bytes

Part e) (9 points extra credit) Now that you finish designing the ISA, you start working on the microarchitecture, which only has to support the two types of instructions. Below is the partially complete schematic of the microarchitecture.

The "Address" port of the memory takes the memory address to be written to, and the "Data" port of the memory takes the data to be written to the memory.

The register file has three read ports and can read three registers, **`Ra`**, **`Rb`**, and **`Rc`**, simultaneously. How the three Read Reg. IDs are generated is irrelevant to this problem.

There are three hidden logics in this partially complete microarchitecture. You job: determine the functionalities of the three hidden logics. Note that not all the input signals to the logics are shown. You need to figure out what signals each logic needs.



(3 points) Logic 1 generates the select signal to the MUX that produces the memory address. Briefly explain how Logic 1 generates the select signal.

Select bet. rC and imm. Look @ first bit of instruction. 0->imm, 1->register

(3 points) Logic 2 takes the value of `Rc` and generates one of the two pieces of data that go into the MUX. Briefly explain how Logic 2 generates its output.

Truncate rC to get 20 lsb from the 32 bits in rC

(3 points) Logic 3 generates the next PC (nPC), which contains the address of the next instruction to be fetched and executed. Briefly explain how Logic 3 generates nPC.

Length of each instr. Depends on first bit. Add offset equal to instruction length, which is determined w/ the bit in PC.