Midterm Exam
CSC 252
3 March 2022
Computer Science Department
University of Rochester

Instructor: Yuhao Zhu
TAs: Nisarg Ujjainkar, Abhishek Tyag, Kalen Frieberg, Gunnar Hammonds, Mandar Juvekar, Zihao Lin, Vladimir Maksimovski, Yiyao (Jack) Yu

Name: _____

Problem 0 (2 points):	_____
Problem 1 (16 points):	_____
Problem 2 (10 points):	_____
Problem 3 (16 points):	_____
Problem 4 (30 points):	_____
Problem 5 (16 points)	_____
Total (90 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. 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)
What's the most unexpected thing you've learned in 252 so far?

Problem 1: Fixed-Point Arithmetics (16 points)

Part a) (3 points) Represent decimal number 42 in hexadecimal form.

Part b) (3 points) Represent octal (base 8) number 35 in decimal form **and** binary form.

Part c) (10 points)

(2 points) Represent signed integer values -25, -18 in 2's complement form, assuming a 6-bit representation.

(6 points) If -25 and -18 are stored in two 6-bit registers R1 and R2, respectively, what are the values of the zero flag, sign flag, and overflow flag after the operation "`add R1, R2`"?

(2 points) What is the value in R2 after "`add R1, R2`"? Expressed the result in binary.

Problem 2: Floating-Point Arithmetics (10 points)

Part a) (2 points) Write -35.75 in the normalized scientific notation.

Part b) (4 points)

Suppose we are using a new 13-bit floating-point standard whose characteristics are compliant with the floating-point representations we discussed in the class. For this representation, exponent bias is 7.

(2 points) How many bits are used for exponent and fraction?

(2 points) What is the floating point representation of 0x0A90 in this format?

Part c) (4 points) Consider two numbers `f1` and `f2`, where `f1` < `f2`. Now we express `f1` and `f2` in the IEEE single-precision format and interpret the resulting bitstreams as unsigned integers. Let's call the unsigned integers `i1` and `i2`. Can we say `i1` < `i2`? If not, provide a counter-example. If yes, explain.

Problem 3: Logic Design (16 points)

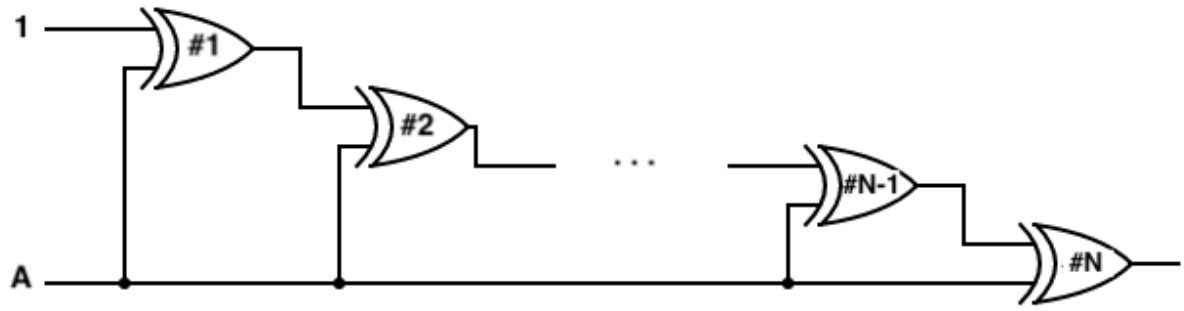
Part a) (4 points)

(2 points) What is the result of a bitwise NAND operation between 101101 and 010110?

(2 points) What is the result of a bitwise XOR operation between 101010 and 111011?

Part b) (6 points)

The given circuit consists of N XOR gates cascaded as shown.



(4 points) What is the output of the circuit for the following values of N? You can write the output as a function of A.

N = 50

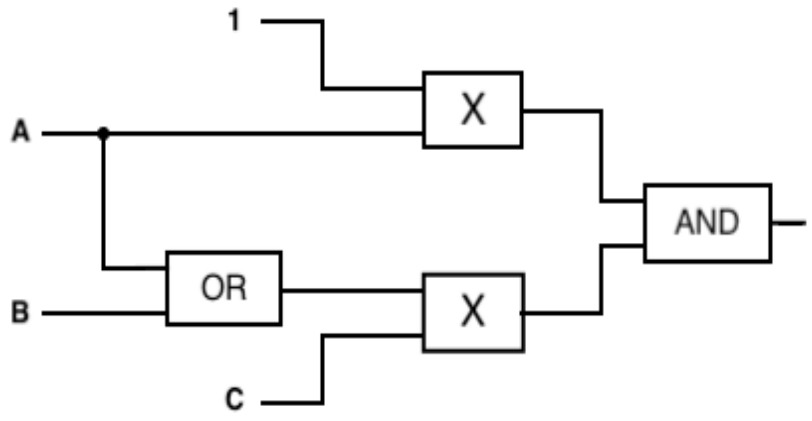
N = 63

(2 points) Can you find a relationship between values of N and the output in terms of A?

Part c) (6 points)

The combinatorial circuit shown below takes three 1-bit inputs: **A**, **B**, and **C**, and produces one 1-bit output. The relationship between the inputs and the output is shown below as a truth table.

The circuit contains two identical, unknown gates X.



A	B	C	Out
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(3 points) What is gate X?

(3 points) Assuming the delay of each gate is 1ps, what is the delay of the entire circuit?

Problem 4: Assembly Programming (30 points)

Conventions:

1. For this section, the assembly shown uses the AT&T/GAS syntax **`opcode src, dst`** for instructions with two arguments where **`src`** is the source argument and **`dst`** is the destination argument. For example, this means that **`mov a, b`** moves the value **`a`** into **`b`** and **`cmp a, b`** then **`jge c`** would compare **`b`** to **`a`** then jump to **`c`** if **`b ≥ a.`**
2. All C code is compiled on a **64-bit machine**, where arrays grow toward higher addresses.
3. For functions that take two arguments, the first argument is stored in **`%edi`** and the second is stored in **`%esi`** at the time the function is called. The return value of this function is stored in **`%eax`** at the time the function returns.

Part a) (20 points)

The following is the C definition of **`struct Data`**:

```
struct Data {
    int matrix[3][3];
    int *value;
};
```

(4 points) What is the size of **`struct Data`**?

(4 points) If the start of **`struct Data`** `d1` is stored at `-0x30(%rbp)`, where in memory is **`d1.matrix[2][1]`** stored?

(6 points) Still assuming the **`struct Data`** `d1` is stored at `-0x30(%rbp)`. Complete the instructions below used to access **`*(d1.value)`**.

```
mov  A  (%rbp),%rdx
 B  (%rdx),%eax
```

A:

B:

Consider the C function **`is_equal()`**:

```
int is_equal(struct Data *p1, struct Data *p2) {
    char *ptr1 = (char *) p1;
    char *ptr2 = (char *) p2;
    for (int i = 0; i < sizeof(struct Data); i++) {
      if (ptr1[i] != ptr2[i]) return 0;
    }
    return 1;
}
```

(3 points) What does **`is_equal()`** do?

Now assuming `d1` and `d2` are initialized as follows:

```
struct Data d1 = {
  .matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
  },
  .value = NULL
};

struct Data d2 = {
  .matrix = {
    {1, 2, 3},
    {4, 5, 6},
```

```
    {7, 8, 9}
  },
  .value = NULL
};
```

(3 points) Will **`is_equal(&d1, &d2)`** always return 1? Explain.

Part b) (10 points) The following is the assembly code for a mystery function **`foo()`**:

```
0000000000000000 <foo>:
   0:  xor %eax,%eax
   2:  xor %ecx,%ecx
   5:  mov %esi,%edx
   7:  add (%rdi,%eax,4),%ecx
   b:  sub %eax,%edx
   d:  dec %edx
   f:  jle 16 <foo+0x16>
  11:  inc %eax
  14:  jmp 5 <foo+0x5>
  16:  mov %ecx,%eax
  19:  ret
```

At the beginning of execution, the value stored in `%esi` is 4, and `%rdi` contains the start address of a 4-element integer array `[9, 8, 7, 6]`.

(3 points) What is the value of `%ecx` after the second execution of `add (%rdi,%eax,4),%ecx`?

(3 points) How many times is `jmp 5` executed?

(4 points) Does this function terminate? If not, why? If so, what value does the function return?

Problem 5: Processor architecture (16 points)

Part a) (6 points)

We have two processors:

- Processor 1 has a 5 stage pipeline with stages: Fetch, Decode, Execute, Memory, Write Back and a 2.5 GHz clock frequency (i.e., 400 ps per cycle). The functionalities of the five stages are the same as we discussed in the class.
- Processor 2 has a 3 stage pipeline with stages: Fetch, Decode, EMW, where the EMW stage combines the Execute, Memory, and Write Back stages in Processor 1. This processor has a clock frequency of 1.6 GHz (i.e., 625 ps per cycle)

Note that there are 10^{12} ps per second.

(4 points) What's the throughput (in instructions per second) for Processor 1 and Processor 2, respectively, assuming no stalls?

(2 points) For what reasons would you expect processor 2 to have a higher clock frequency than processor 1? You should assume that everything between the processors are the same other than clock frequency and pipeline structure.

Part b) (10 points)

Assume a processor architecture implementing the x86 ISA that is pipelined in 5 stages (fetch, decode, execute, memory, and write back) as discussed in lectures. The program is executing the piece of assembly shown below on the left. Assuming the pipeline is empty at the start of the program, and `%rip` points to the first instruction.

Assembly Code	Relevant part of the memory & registers at the start of the program

`.L1: incq %rax`
`.L2: cmpq %rax, %rbx`
`.L3: jge .L8`
`.L4: addq $0x3, $rax`
`.L5: xorq %rax, %rbx`
`.L6: cmpq %rbx, %rax`
`.L7: jmp .L9`
`.L8: nop`
`.L9: leaq -0x8(rcx), %rdx`
`.L10: addq (%rdx), %rax`
`.L11: movq 0x8(%rcx), %rbx`
`.L12: cmpq %rax, %rbx`
`.L13: jle .L15`
`.L14: negq %rax`
`.L15: addq %rax, %rbx`
	Registers:
	`%rax: 0x196`
	`%rbx: 0x200`
	`%rcx: 0x40000810`
	`%rdx: 0`
	
	Memory:
	`0x40000800: 0x205`
	`0x40000808: 0x207`
	`0x40000010: 0x352`
	`0x40000018: 0x595`
	`0x40000020: 0x400`

(3 points) Which jump instructions get executed and are taken (i.e., do not fall-through)? List them by their labels, e.g. L3, L7, L13.

(5 points) Write down the list of stalls due to **control dependencies**. Write down the labels of *all* the pairs of instructions creating such stalls, e.g., L1 -> L2 if L1 and L2 create a control dependency.

(2 points) Where in the pipeline is a control dependency resolved?