Midterm Exam
CSC 252
3 March 2023
Computer Science Department
University of Rochester

Instructor: Yuhao Zhu
TAs: Yifan Zhu, Matthew Nappo, Yekai Pan, Toranosuke Ozawa, Zeyu Nie, Yumeng He, Stela Ciko, Nisarg Ujjainkar

Name: _____

Problem 0 (2 points):	_____
Problem 1 (9 points):	_____
Problem 2 (20 points):	_____
Problem 3 (10 points):	_____
Problem 4 (24 points):	_____
Problem 5 (25 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)
Have you been to TA office hours before?

YES/NO

Problem 1: Fixed-Point Arithmetics (9 points + 2 points extra credit)

Part a) (2 points) Represent decimal number 24 in hexadecimal form.

18

Part b) (3 points) Represent octal (base 8) number 53 in decimal form **and** binary form.

43;101011

Part c) (4 points + 2 points extra credit) Consider two unsigned binary numbers A = 1011 and B = 0101. Do the math below. Use as many bits as needed to precisely represent the results.

(2 points) What's the result of A+B?

10000

(2 points) What's the result of A-B?

0110

(2 points extra credit) What's the result of A * B?

110111

Problem 2: Floating-Point Arithmetics (20 points)

Part a) (6 points) Consider a decimal number F = -100.50:

(2 points) Give the binary representation of F:

-1100100.1

(2 points) Put F into the normalized scientific notation:

-1.1001001 x 2^6

(2 points) Write the significand in binary:

1001001

Part b) (4 points) Consider a hex value F = 0x7f800000 that represents a floating point number encoded using the IEEE 754 single-precision format.

(2 points) What is the exponent value (in decimal)?

128

(2 points) What does F represent?

Positive infinity

Part c) (10 points) Professor Bob teaches a computer science course, and he is designing the scoring system of a midterm exam. The exam consists of 64 multiple choice questions and students will receive 1 point for each correct answer, 0 point for leaving it blank, and -0.25 points for a wrong answer.

To store all the possible scores, Bob wants to incorporate knowledge from CSC 252 to improve the space efficiency by designing a new floating point representation, which should meet three criteria:

1. The characteristics are compliant with the floating-point representations we discussed in class;
2. Can accurately represent any valid score;
3. Use as few bits as possible.

(2 points) How many bits are used for the significand in this new floating point representation?

7 (We need at most 8 bits for all of them but the first bit comes for free.)

(2 points) How many bits are used for the exponent fields in this new floating point representation?

4 (2^-2 to 2^6, where 3 bits is not enough for exponents 3,4,5,6)

(2 points) What's the bias?

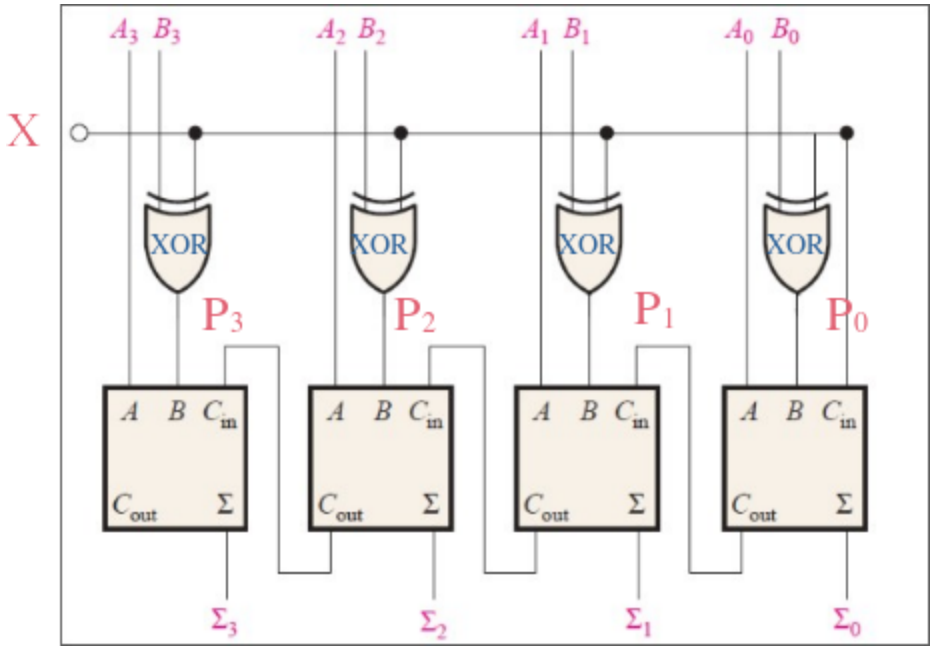
7 = (2^(4-1)) - 1

(4 points) Can you use a fixed point representation for the same purpose? If yes, at least how many bits do you need? If not, explain the reason.

yes ; 10 Note: 1 bit for the sign, 7 bits for the integer part, and 2 bits for the fraction part, where 0.25 is the implied number.

Problem 3: Logic Design (10 points + 3 points extra credit)

Consider the following circuit. Each square-shaped component represents a 1-bit full adder: A and B are its two 1-bit inputs, C_{in} is the 1-bit carry in, Σ is 1 one-bit output, and C_{out} is the 1-bit the carry out. A_3, A_2, A_1, A_0, B_3, B_2, B_1, B_0 are all 1-bit inputs. P_3, P_2, P_1, P_0 are the outputs of the four XOR gates. Σ_3, Σ_2, Σ_1, Σ_0 are 1-bit outputs of the four full adders.



The truth table of a 1-bit full adder is below:

A	**B**	**C_{in}**	**Σ**	**C_{out}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Part a) (4 points) Assuming $A_3A_2A_1A_0 = (1001)_2$ and $B_3B_2B_1B_0 = (0001)_2$:

(2 points) If X = 0, what is the value of $P_3P_2P_1P_0$?

0001

(2 points) If X = 1, what is the value of $P_3P_2P_1P_0$?

1110

Part b) (4 points) Still assuming $A_3A_2A_1A_0 = (1001)_2$ and $B_3B_2B_1B_0 = (0001)_2$:

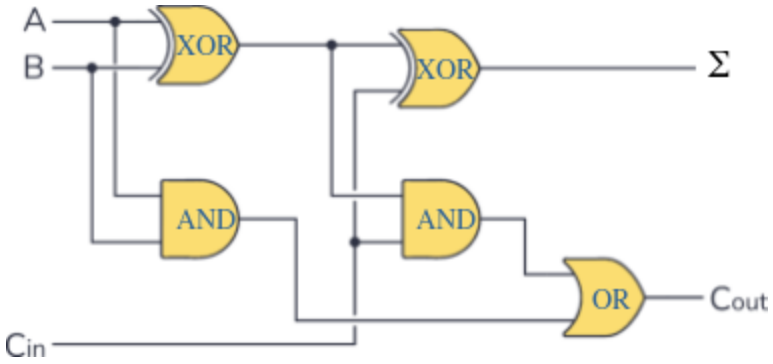
(2 points) If X = 0, what is the value of $\Sigma_3\Sigma_2\Sigma_1\Sigma_0$?

1010

(2 points) If X = 1, what is the value of $\Sigma_3\Sigma_2\Sigma_1\Sigma_0$?

1000

Part c) (2 points + 3 points extra credit) Assume that the 1-bit full adder is implemented as shown in the figure below. The delay at each gate is 1ps.



(2 points) What is the propagation delay of the 1-bit full adder?

3 ps

(3 points extra credit) What is the total propagation delay of the entire circuit shown at the beginning of this problem? Assuming that the XOR gates outside the full adder have a delay of 1ps, too.

9 ps Note: the critical path is B_0 -(1 gate)- P_0 P_0 -(3 gate)- $Cout_0$ $Cout_0$ -(2 gate)- $Cout_1$ $Cout_1$ -(2 gate)- $Cout_2$ $Cout_2$ -(1 gate)- Σ_3

Problem 4: Assembly Programming (24 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. We use the x86 calling convention. That is, for functions that take two arguments, the first argument is stored in **%edi (%rdi)** and the second is stored in **%esi (%rsi)** at the time the function is called; the return value of a function is stored in **%eax (%rax)** at the time the function returns.
4. We use the **Little Endian** byte order when storing multi-byte variables in memory.

Part a) (9 points)

Consider the following C code:

```
struct Car{
      char *model;
      int year;
      long mpg;
};

struct Car myCars[2];
```

The following assembly is part of a function **print_info()**, which, as you can see, calls another function **get_cars(),** which initializes **myCars** and returns the pointer that points to **myCars**.

```
1:    call    get_cars
2:    movq    %rax, -8(%rbp)
3:    movq    -8(%rbp), %rax
4:    addq    $24, %rax
5:    movq    16(%rax), %rax
6:    movq    %rax, -16(%rbp)
7:    movq    -8(%rbp), %rax
8:    movq    (%rax), %rax
9:    movb    3(%rax), %al
```

(3 points) What is the size (number of bytes) of **struct Car**, assuming proper alignment?

24

(3 points) At the end of line 5, what C variable does `%rax` contain? Express in C syntax.

myCars[1].mpg

(3 points) At the end of line 9, what C variable does `%rl` contain? Express in C syntax.

myCars[0].model[3]

Part b) (15 points)

Consider the following function **p2**. Its C code and assembly code are both *partially* given. Assuming that the input arguments to **p2**are **x=4**, **y=4**.

Then, use the rest of the assembly to find the return value of **p2()** given

```
unsigned long p2(unsigned long x, unsigned long y){
     unsigned long t1= x + 2*y;
     unsigned long t2= . . . ; // intentionally hidden
     return t2;
}

// some irrelevant instructions at the beginning omitted
1:    movq %rdi, -24(%rbp)
2:    movq %rsi, -32(%rbp)
3:    movq -32(%rbp), %rax
4:    A    (%rax,%rax), %rdx
5:    movq -24(%rbp), %rax
6:    addq  B , C
7:    movq %rax, -8(%rbp)
8:    movq -8(%rbp), %rax
9:    shrq $2, %rax
10:   movq %rax, -16(%rbp)
11:   movq -16(%rbp), %rax
12:   ret
```

Complete the three missing pieces in the assembly code.

(3 points) A:

leaq

(3 points) B:

%rdx

(3 points) C:

%rax

(3 points) What is the byte contained in `-8(%rbp)` after line 7?

oxC

(3 points) What is the return value of **`p2`**?

3

Problem 5: Processor architecture (25 points + 2 points extra credit)

Part a) (**4 Points**)
Consider a microarchitecture that is designed for an ISA with 256 instructions and 16 general-purpose registers. The ISA has a 16-bit address space. Assuming the opcode fields in all the instructions are encoded using the same amount of bits.

(**2 Points**) If an instruction in this ISA takes the form of `ins r1,r2,r3`, where `ins` is the opcode, and `r1`, `r2`, and `r3` are three general-purpose registers. How many bits are required to encode this instruction?

20

(**2 Points**) Assuming the jump instructions in this ISA are encoded using absolute addresses. What is the number of bits needed to encode a jump instruction?

24

Part b) (**21 Points + 2 points extra credit**)
A single-cycle processor has a cycle time of **30 ns**. Consider two different pipelined implementations of the same processor, each with 5 and 10 stages, respectively. Assume that the stages divide the computation uniformly. In both pipelined processors, it takes **1 ns** for the output of a stage to latch on to the pipeline registers.

(**3 Points**) What is the shortest plausible clock period for the 5-stage pipelined processor?

7 ns

(**3 Points**) What is the shortest clock period for the 10-stage pipelined processor?

4 ns

(**3 Points**) What is the execution time per instruction in the 10-stage pipelined processor?

39 ns (40 accepted)

(**3 Points**) Assuming no pipeline stall of any kind, how long does it take to execute 50 instructions in the 5-stage pipeline?

54*7 = 378 ns

(**3 Points**) Assuming no pipeline stall of any kind, how long does it take to execute 50 instructions in the 10-stage pipeline?

59*4 = 236 ns

(**3 Points**) Assuming no pipeline stall of any kind, express the total execution time of 50 instructions as a function of the pipeline stages T.

(x-1+50)*(30/x+1)

(**2 Points Extra Credit**) What number of pipeline stages will give you the minimum total execution time for 50 instructions?

38