

Midterm Exam
CSC 252
25 October 2021
Computer Science Department
University of Rochester

Instructor: Alan Beadle

TAs: Abhishek Tyagi, Matthew DeWeese, Elana Elman, Yifan "Iven" Jiang, Yanghui "Woody" Wu, Zeyu "George" Wu

Name: _____

Problem 0 (2 points):	_____
Problem 1 (15 points):	_____
Problem 2 (12 points):	_____
Problem 3 (15 points):	_____
Problem 4 (17 points):	_____
Problem 5 (14 points)	_____
Total (75 points):	_____
Extra Credit (4 points)	_____

Remember "**I don't know**" is given 15% partial credit, but you must erase or cross out 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 until the end of the class to work (~75 minutes).

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 aspect of how computers work do you want to learn more about most of all?
(all answers accepted)

Common answers:

Compiler/assembler design, code efficiency, GPU, security exploits, multi-core, hardware design, operating systems, "everything", and various topics on this exam

Problem 1: Fixed-point Arithmetic (15 points)

Part a) (4 points) Convert the decimal number 108 to hexadecimal.

0x6C

Part b) (4 points) What is the decimal representation of the base 3 number 120?

15

Part c) (4 points) What is the two's complement representation of the decimal number -43? Assume an 8-bit representation. Express the answer in hexadecimal.

0xD5

Partial credit for:
11010101

Part d) (3 points) If 4-bit registers R1 and R2 contain 1100 and 0100 respectively, what are the values of the carry, overflow, and sign flags after the operation "add R1 R2"?

Carry = 1
Overflow = 0
Sign = 0

Problem 2: Floating-Point Arithmetics (12 points)

Part a) (4 points) Put $9\frac{3}{8}$ into the binary normalized form.

1.001011x2³

1 point partial credit for correct bit pattern

1 point partial credit for only having a 1 to the left of the decimal point

Part b) The IEEE has decided to introduce a 11-bit floating-point standard, whose main characteristics are consistent with existing floating-point number representations that we discussed in the class.

The following bit sequence contains the exact encoding of $\frac{33}{256}$ in this new standard:

00100000010

(2 points) How many bits are used for the fraction in this new standard?

6

(3 points) How many bits are used for the exponent in this new standard, and what is the bias?

4 exponent bits

bias is 7

(1.5 points each)

(3 points) What is the smallest positive value that can be represented in this new standard? (Write your answer in decimal. You may use an exponent.)

1 point partial credit for observing that it should be denormalized

1 more for 0 0000 000001

Full credit for 2^{-12} or equivalent. ($2^{-12} = 0.000244140625$)

(2 points extra credit) How would positive infinity be represented in this new float type? (write your answer in binary)

0 1111 000000

Problem 3: Data types and Assembly (15 points + 2 points extra credit)

Part a) Consider the following C struct:

```
struct rec {  
    char c;  
    int i;  
    char d;  
    unsigned long id;  
};
```

(3 points) Memory alignment requirements of variable types mean that there may be extra padding in a struct. How many bytes would the above struct use on the CS department machines?

24

Partial credit: up to 1 point for either correct variable sizes or an awareness of struct padding.

It looks like many people got 32, thinking that all variables needed to be padded to 8 because of the double. Award 1 point partial credit for this type of answer.

(3 points) Reorder the variables in the above struct to use the least possible space. Write your answer as a C struct definition (like the example).

1 point for correct syntax
2 points for correct order (long, int, char, char)

(2 points) How much space does your improved struct use?

16 (bytes)

Part b) Consider the following objdump output, produced from a C function which has been compiled with gcc:

0000000000001141 <foo>:

```
1141: 89 7c 24 ec      mov  %edi,-0x14(%rsp)
1145: c7 44 24 fc 00 00 00  movl $0x0,-0x4(%rsp)
114c: 00
114d: c7 44 24 f8 00 00 00  movl $0x0,-0x8(%rsp)
1154: 00
1155: eb 13           jmp  116a <foo+0x29>
1157: 8b 54 24 fc      mov  -0x4(%rsp),%edx
115b: 8b 44 24 ec      mov  -0x14(%rsp),%eax
115f: 01 d0           add  %edx,%eax
1161: 89 44 24 fc      mov  %eax,-0x4(%rsp)
1165: 83 44 24 f8 01  addl $0x1,-0x8(%rsp)
116a: 8b 44 24 f8      mov  -0x8(%rsp),%eax
116e: 39 44 24 ec      cmp  %eax,-0x14(%rsp)
1172: 77 e3           ja   1157 <foo+0x16>
1174: 8b 44 24 fc      mov  -0x4(%rsp),%eax
1178: c3             retq
```

(1 point) Where is the return value stored when retq is executed?

%eax, or %rax, or %ax, or eax, or rax, or ax.

(2 points) If the C program calls foo(3), what value is returned?

9

(4 points) Write a C function “foo” that does the same thing as the assembly code above, in the same way. Be sure to show any loops and local variables needed. **Makes sure that your variable/argument types make sense given the assembly code above.**

```
unsigned foo(unsigned int x){
    unsigned sum = 0;
    for(int i=0; i<x; i++){
        sum += x
    }
    return sum;
}
```

This one is hard to grade. Be flexible and give code that works properly a full score.

1 point for the subtle “unsigned” int which can be inferred from the “ja” instruction instead of “jg”. Otherwise the code would not correctly square negative values. Making it unsigned completely avoids this issue.

(2 points extra credit) The assembly code shown is not very efficient. What x86_64 instruction would achieve the same result without looping? (you do not need to specify registers, just the instruction name that would appear in objdump)

imul or imulq or lea or leaq

And call/callq, because it seems like some people thought of recursion. Nice haha. Although this would not be more efficient, it does avoid loops.

Problem 4: Logic Design (17 points)

Part a) bit-wise operations

(3 points) What is the result of a bit-wise XOR between 0101 and 0110?

0011

(3 points) What is the result of a bit-wise NOR between 0101 and 0110?

1000

Part b) NAND gates!

It is possible to build any other gate out of combinations of only NAND gates. (We therefore say that NAND is “complete”)

For example, if you connect the inputs of a NAND gate together to act as one input, it will behave as a NOT gate.

(3 points) Draw a NOR gate made entirely from 4 NAND gates.

A and B each go into a separate NOT (as described above)
Then each one goes into one input of a NAND
The output of the NAND goes into a final NOT

Also acceptable to make the NOT gates by sending “1” to one of the inputs of a NAND.

Also acceptable to write it in the form of an expression of only NAND:
 $((A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B))$
NAND
 $(A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B)$

(2 points) Assume that our NAND gates each have a maximum delay of 1ps. What is the maximum delay of the NOR gate you designed above?

3 ps

(1 point for just "3")

(3 points) In class we looked at the S-R latch, a memory circuit that can store one bit. In our slides, it was made from 2 OR gates and 2 NOT gates. Draw one made from 8 NAND gates (hint: use your NOR gate from above)

Two crosslinked instances of answer above. Labeling of inputs/outputs optional.

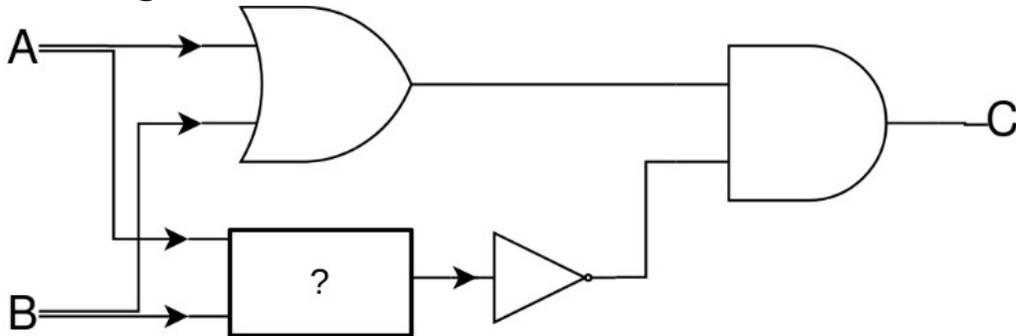
Partial credit: 1 point for labeling either R and S inputs or Q+/Q- outputs

Part c) The logic circuit below is missing one gate, marked with the rectangle "?". The output of the entire circuit should match the truth table shown below.

Truth table:

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

Circuit Diagram:



(3 points) What type of gate is missing?

AND

(All other answers are wrong)

Problem 5: Microarchitecture (14 points)

Part a) Pipeline question!

Consider the CPU pipeline shown below.



(2 points) What is the minimum cycle time of this CPU?

170 ns

(2 points) What is the instruction latency of this CPU?

$170 * 3 = 510$ ns

(3 points) Suppose that the hardware designer decided to improve the design above by splitting the slowest stage into two separate stages. It would be a 4-stage CPU. What is the best possible throughput of this new design?

1/120 ns (because stage B will be the new slowest one)

Half credit for 120 ns. (throughput is measuring instructions per unit of time)

Part b) Now imagine that the 3-stage pipeline from the diagram above has the following stages: First stage is fetch. Second stage is decode, and the third stage does everything else (execute, memory, writeback). Consider the following assembly code fragment:

```
.L1:  
sub $0x01, %rax  
cmpq %rax, $0x01  
jge .L1  
add $0x99, %rsi  
mov %rsi, %rax  
ret
```

(3 points) If the jump is not taken, but the CPU predicts taken, how many cycles will it take for this code to finish? Assume that all data dependencies are handled by data forwarding, and assume that the branch misprediction is detected in the execute stage.

Hint: Remember that the pipeline starts empty, and the program is not done until the last instruction has finished all stages.

10 cycles

Partial credit:

2 points partial credit for 9 cycles (Correctly understood code but thought that bubble was only 1 instruction, or forgot to add 1 cycle for the last instruction to exit the pipeline).

Or

1 point if both mistakes together results in 8 cycles.

Or

1 point partial credit for showing correct use of a 3-stage diagram but misunderstood code

Part c) Suppose that you have written an assembly program with one conditional jump in it which will be taken twice in a row and then not taken. Your CPU has a two-bit branch predictor which will initially predict “not taken”.

(2 points) How many mispredictions will there be?

3

(2 points) How many mispredictions would there be if it was a 1-bit branch predictor instead?

2