Mid-Term Exam

CSC 252/452 Fall 2025
University of Rochester
Oct. 8th, 2025

Instructions:

1. This exam has 13 pages including this page. Check that your copy has all the pages.
2. Do not start the exam until 3:25 pm and do not continue working after 4:40 pm. You may ask a proctor if you are unsure of the time.
3. For fairness, the instructor and proctors will not answer questions about the exam content. If an exam question is unclear to you, make assumptions and write down any assumptions with your answer.
4. **For students who are taking 452**, all problems, including those marked as extra credit, are part of the total score.
5. "I don't know" is given 15% partial credit, but you must erase everything else. This does not apply to extra credit questions.
6. Your answers to all questions must be contained in the given boxes. Use spare space to show all supporting work to earn partial credit.
7. List of problems (76 points total + 3 points extra credit):
    - Problem 0 (2 points)
    - Problem 1 (19 points)
    - Problem 2 (17 points + 3 points extra credit)
    - Problem 3 (16 points)
    - Problem 4 (12 points)
    - Problem 5 (10 points)

Your name:

Signature:

Good luck!!!

Problem 0: Warm-up (2 Points)

Select all the options that apply. Multiple selections are allowed.

- ☐ I have been to TA office hours.
- ☐ I have asked a question on Piazza.
- ☐ I have found answers to my question on Piazza.
- ☐ I like programming in Assembly.

Problem 1: Integer Arithmetic (19 points)

Part a) (4 points) Represent the binary number (unsigned) 10110010 in decimal form.

178

Part b) (4 points) Represent the decimal number 74 in hexadecimal form **and** binary form.

4A
01001010

Part c) (4 points) What are the two's complement representations of the decimal numbers -15 **and** 23? Assume a 10-bit representation.

1111110001

0000010111

Part d) (3 points) Assume you are working with 6-bit two's complement numbers. Answer the following questions:

1. What is the largest representable positive integer?
2. What is the smallest representable negative integer?
3. Perform the following addition in 6-bit signed arithmetic. Write the result in **decimal**.
   101110 + 110100

31
-32
-30

Part e) (4 points) Consider two 4-bit registers `R1` and `R2`; `R1`=1001, `R2`=0110 (both values are in binary form). Answer the following questions:

1. What are the values of the carry, overflow, zero, and sign flags after the operation "`add R1, R2`"?
2. What are the values of the carry, overflow, zero, and sign flags after the operation "`sub R1, R2`"?

CF=0 OF=0 ZF=0 SF=1
CF=0 OF=1 ZF=0 SF=0

Problem 2: Floating-Point Arithmetic (17 points + 3 points extra credits)

Part a) (6 points) Consider a decimal number F = 48.00

(3 points) Put F into the normalized scientific notation **(in binary)**.

1.1*2^5

(3 points) What are the advantages of floating-point representation, compared to fixed-point representation?

Part b) (8 points) Assume we are using a new floating-point standard whose characteristics are compliant with the floating-point representations we discussed in the class. For this representation, exponent bias is 7. The smallest positive number that can be represented is 2^{-11}.

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

5

(4 points) What is the floating-point representation of 0xB in this format?

0 1010 01100

Part C) (3 points+ 3 points extra credit) Assume you are working with a 9-bit floating-point format, which consists of 1 sign bit, 4 exponent bits, and 4 fraction bits. Assume number A is represented as
0 0000 1110 in this format. If floating point addition gives A+B=A, answer the following questions.

(3 points) What is a possible **non-zero** value of B? Write your answer in the given floating-point format.

Does not exist

(3 points extra credit) What is the **largest** possible value of B? Write your answer in the given floating-point format.

Does not exist

Problem 3: Assembly Programming (16 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`** (signed comparison).
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 three arguments, the first argument is stored in **`%edi`** (**`%rdi`**), the second is stored in **`%esi`** (**`%rsi`**), and the third is stored in **`%edx`** (**`%rdx`**) 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. We use **Row-Major** Ordering for 2D arrays.

C code:

```
long foo(long val1, long val2, long* arr)
{
     long temp = 100;
     if (val2 * 31 + 4 > 10)
          return temp * 20;
     else
          return arr[val1*3+4];
}
```

Assembly code:

```
     # some irrelevant instructions at the beginning are omitted
     pushq     %rbp
     movq %rdi, -24(%rbp)
     movq %rsi, -32(%rbp)
     movq %rdx, -40(%rbp)
     movq $100, -8(%rbp)
     movq -32(%rbp), %rdx
     movq %rdx, %rax
     salq $5, %rax
     __A__ %rdx, %rax
     addq $4, %rax
     cmpq __B__, %rax
     jle  .L2
     movq -8(%rbp), %rdx
     movq %rdx, %rax
     salq $2, %rax
     __C__ %rdx, %rax
     salq $2, %rax
     __D__ .L3
.L2:
     movq -24(%rbp), %rdx
     movq %rdx, %rax
     addq %rax, %rax
     addq %rdx, %rax
     __E__ $3, %rax
     __F__ 32(%rax), %rdx
     movq -40(%rbp), %rax
     addq %rdx, %rax
     movq (%rax), __G__
.L3:
     __H__ %rbp
     ret
```

Complete the missing pieces in the assembly code.

(2 points) A:

subq

(2 points) B:

$10

(2 points) C:

addq

(2 points) D:

jmp

(2 points) E:

sal

(2 points) F:

leaq

(2 points) G:

%rax

(2 points) H:

popq

Problem 4: Data Structure (12 points)

Conventions: same with the ones in Problem 3.

Consider the following C code. `CPU` is a structure to represent the critical features of a CPU.

```
struct CPU {
    char cache[2][3];
    int tlb[3][2][2];
    double clock_speed_GHz;
    int cores;
    char *name;
};

struct CPU CPU_List[10];
```

(3 points) Without data alignment, what will be the value printed when we run `printf("%d\n", sizeof(struct CPU));`?

74

(3 points) Without data alignment, if the start of `CPU_List[0]` is stored at `-0x24(%rbp)`, where in memory is `CPU_List[1].cache[1][2]` stored?

0x2B(%rbp)

(3 points) With proper data alignment, assume `CPU_List = 0x7fff0000`, what is the value of `CPU_List+2`?

0x7fff000a0

(3 points) If data alignment is required, how to reorder the members in the structure to be **most** space efficient?

Multiple answers

Problem 5: ISA (10 points)

The designers of Y86-64 are adding an **indirect** jump instruction to their ISA:

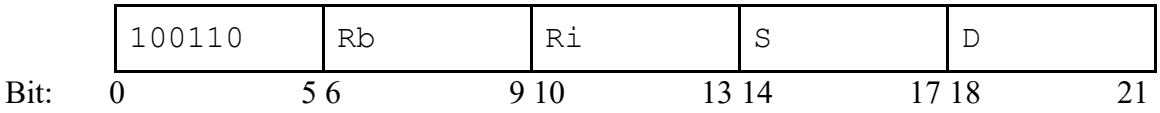
```
jmp *D(Rb, Ri, S)
```

This instruction calculates the memory address `Rb+Ri*S+D`, then fetches the jump target (jump address) from this memory location, and then makes the jump. For example, the following instruction fetches the jump target from address `%rax+%rdx*4+12`, and makes the jump based on the jump target:

```
jmp *$12(%rax, %rdx, 4)
```

As we learned in class, `D(Rb, Ri, S)` represents the complete address mode. The elements `Rb`, `Ri`, `S`, and `D` could be omitted if not needed.

The Y86-64 designers proposed the following encoding format for this instruction:



The entire `leaq` instruction is 22 bits long. Bits 0-5 are used for the opcode, bits 6-9 are used for `Rb`, bits 10-13 are used for `Ri`, bits 14-17 are used for `S`, bits 18-21 are used for `D`. `S` and `D` are stored in two's complement format.

Assume that the registers in Y86-64 are encoded using the 4-bit values shown in the table below:

Register	Encoding	Register	Encoding
%rax	0000	%r8	1000
%rbx	0001	%r9	1001
%rcx	0010	%r10	1010
%rdx	0011	%r11	1011
%rsi	0100	%r12	1100
%rdi	0101	%r13	1101
%rsp	0110	%r14	1110
%rbp	0111	No-register	1111

(3 points) Give the assembly form of the instruction encoded as `10011000110110010010 11`.

jmp *-5(%rdx, %rsp, 4)

(3 points) If the jump target for an indirect jump is stored at the memory location with an address of `8( , %rdx, 4)`, what is the complete encoding of this instruction?

10011011110011010 0xxxx

(4 points) Compared to fixed-length instruction encoding, what are the advantages and disadvantages of variable-length instruction encoding?