CSO1 **Name:** ______________________

Spring 2022

Final Exam

2022-05-9

Time Limit: 180 minutes Computing ID ______________________

Instructions:

1. This exam contains 13 pages (including this cover page) and 12 questions.
2. You have **180 minutes** to complete the examination.
3. Write your answers in this booklet. We scan this into GradeScope, so **please try to avoid writing on the backs of pages**.
4. If a question presents several options in a list, mark the bubble next to the one correct answer. All such questions on this test are single-select unless otherwise specified in a specific question .
5. You may not use a calculator or notes.
6. Because this assessment is being given in several places, we cannot fairly answer questions during it. If you find a question ambiguous or unclear, please explain that on the page by the question itself and we will consider your explanation during grading.
7. We will use the following data type sizes:

Types	size in bits
`char`	8
`short`	16
`int` and `float`	32
`long` and `double`	64

8. Function arguments are in (in order) `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`; return values are in `%rax`.
9. The next several pages contain reference material which you are welcome to refer to during the test if you would like.
10. Please sign the below Honor Code statement.

I have neither given nor received aid on this exam.

Signature: ______________________

Our Example ISA

This is the same ISA used in HW03 and HW04, but presented to fit onto one printed page.

Each instruction is one or two bytes, with the meaning of those bytes being:

7	6	5	4	3	2	1	0
`0`	`icode`			`a`		`b`	

byte at `pc`

7	6	5	4	3	2	1	0
immediate							

byte at `pc + 1`

Not all instructions have the second byte; those that do describe it below as the byte "at `pc + 1`".

In the table below `rA` means "the value stored in register number `a`" and `rB` means "the value stored in register number `b`."

`icode`	`b`	Behavior	add to `pc`
0		`rA = rB`	1
1		`rA += rB`	1
2		`rA &= rB`	1
3		`rA` = read from memory at address `rB`	1
4		write `rA` to memory at address `rB`	1
5	0	`rA = ~rA`	1
5	1	`rA = -rA`	1
5	2	`rA = !rA`	1
5	3	`rA = pc`	1
6	0	`rA` = read from memory at `pc + 1`	2
6	1	`rA` += read from memory at `pc + 1`	2
6	2	`rA` &= read from memory at `pc + 1`	2
6	3	`rA` = read from memory at the address stored at `pc + 1`	2
7		if `rA <= 0`, set `pc = rB`	N/A
7		if `rA > 0`, do nothing	1

If the first bit of the byte at `pc` is `1` instead of `0`, the above text does not define what the instruction means, but some other source (such as a question on this exam) might. If it has no defined meaning either here or elsewhere, leave the `pc` and all other registers and memory values unchanged.

NAME — memcmp

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n);
```

DESCRIPTION

The **memcmp**() function compares the first *n* bytes (each interpreted as *unsigned char*) of the memory areas *s1* and *s2*.

RETURN VALUE

The **memcmp**() function returns an integer less than, equal to, or greater than zero if the first *n* bytes of *s1* is found, respectively, to be less than, to match, or be greater than the first *n* bytes of *s2*.

For a nonzero return value, the sign is determined by the sign of the difference between the first pair of bytes (interpreted as *unsigned char*) that differ in *s1* and *s2*.

If *n* is zero, the return value is zero.

NAME — strcat, strncat

```
#include <string.h>

char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t n);
```

DESCRIPTION

The **strcat**() function appends the *src* string to the *dest* string, overwriting the terminating null byte ('\0') at the end of *dest*, and then adds a terminating null byte. The strings may not overlap, and the *dest* string must have enough space for the result. If *dest* is not large enough, program behavior is unpredictable; *buffer overruns are a favorite avenue for attacking secure programs.*

The **strncat**() function is similar, except that

- it will use at most *n* bytes from *src*; and
- *src* does not need to be null-terminated if it contains *n* or more bytes.

As with **strcat**(), the resulting string in *dest* is always null-terminated.

If *src* contains *n* or more bytes, **strncat**() writes *n+1* bytes to *dest* (*n* from *src* plus the terminating null byte). Therefore, the size of *dest* must be at least *strlen(dest)+n+1*.

1 RETURN VALUE

The **strcat**() and **strncat**() functions return a pointer to the resulting string *dest*.

NAME — atoi, atol, atoll

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *nptr);
long atol(const char *nptr);
long long atoll(const char *nptr);
```

DESCRIPTION

The **atoi**() function converts the initial portion of the string pointed to by *nptr* to *int*. The behavior is the same as

```
strtol(nptr, NULL, 10);
```

except that **atoi**() does not detect errors.

The **atol**() and **atoll**() functions behave the same as **atoi**(), except that they convert the initial portion of the string to their return type of *long* or *long long*.

RETURN VALUE

The converted value.

1. Throughout the semester we took a bottom up approach (from electricity → C) to Computer Systems and their Organization (CS01). This parts of this question takes a top down approach (from C → electricity).

(a) The program below calculates the intercept `y` given `x`, `m` and `b`: $y = mx + b$:

```
int intercept(int *x, int *m, int b) {
    return _________;
}
```

Which of the following would return the correct result if used to fill in the blank above? Select all that apply.

- ◯ `*(x * m) + b`
- ◯ `*x **m  + b`
- ◯ `*x * *m + b`
- ◯ `x * m + b`

(b) Implement the intercept function in assembly. Note that `imull` is an integer multiply instruction.

```
0       addl   (%rdx), %eax
1       addl   %edx, %eax
2       imull  (%rsi), %eax
3       imull  %esi, %eax
4       movl   (%rdi), %eax
5       movl   %edi, %eax
6       ret
```

Write the numbers of the instructions in the boxes below in the correct order. You may leave some boxes empty if you use fewer instructions than we've provided boxes. We've included the return instruction for you.

```
intercept:
    [   ]
    [   ]
    [   ]
    [   ]
    [ 6 ] ret
```

(c) Here is an implementation of the intercept program implemented using Example ISA (see page 2). Encode the following instructions and write the hex encoding the box. Do **NOT** include `0x` prefix. So instead of writing `0x03` write `03`. This makes it easier for gradescope's machine learning algorithm to scan your answer. Assume that first instruction is loaded at address 0.

```
r0 = 3   // input x
r1 = 10  // input m
r2 = 4   // input b
r1 = -r1
r3 = _____ // jump address -- several valid options here
r0 += r0
r1 += 1
if r1 <= 0 set pc = r3
r0 += r2 // output y
```

Fill in the encoding below, leave the boxes that you are not using empty.

(d) Throughout the semester we looked at various components associated with a single cycle processor. Here we designed a simple machine that we call the Interceptor 1000. This machine has only has two instructions, listed below by their opcodes:

0. `nop` which does nothing
1. `intcep $x, $m, $b` which computes $mx + b$

An example instruction for this machine might be `intcep 2 3 1` meaning x is 2, m is 3 and b is 1. This instruction would be encoded as follows:

```
 0 1 2 3 4 5 6 7 8 9 a b c d e f
|1|0 0 0 1 0|0 0 0 1 1|0 0 0 0 1|
```

In the diagram below, `O(1)` represents the opcode that is 1 bit long, while `X(5)` represents the `x` value that is 5 bits long.

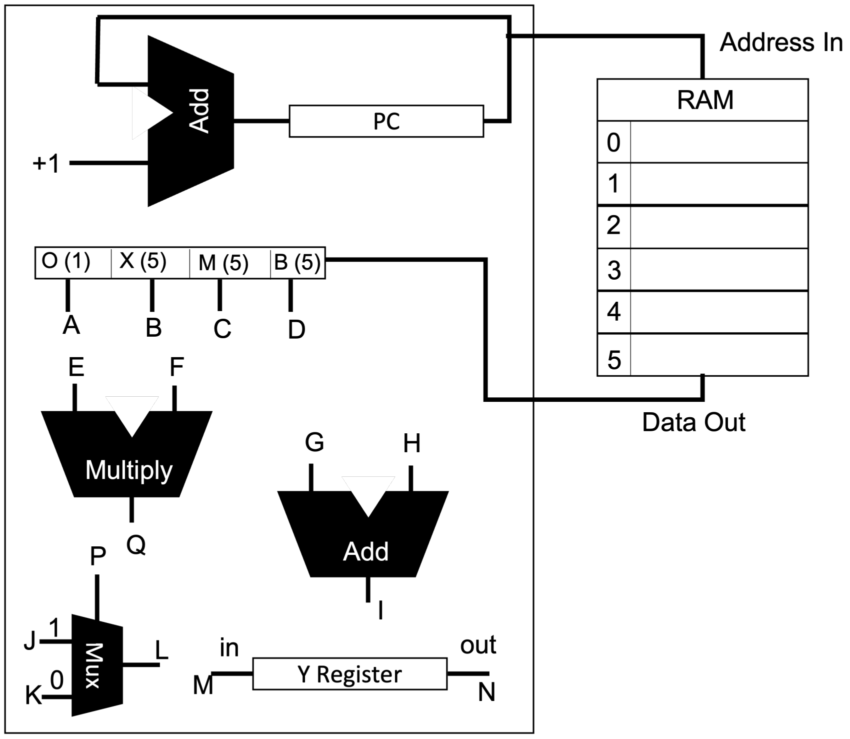

Figure 1: Block Diagram for Interceptor 10000

Show how to complete the diagram by writing the name of the wire each given wire should be connected to. For example, if A is connected to E, write E in the box next to A.

A	$\rightarrow$	
B	$\rightarrow$	
C	$\rightarrow$	
D	$\rightarrow$	

I	$\rightarrow$	
L	$\rightarrow$	
N	$\rightarrow$	
Q	$\rightarrow$	

2. Given two integers x and y, compute `x ^ y` without using `^`

   ◯ `(x | y) & (~x | ~y)`
   ◯ `(x & y) | (~x & ~y)`
   ◯ `(~x | y) & (x | ~y)`
   ◯ `(~x & y) | (~x & ~y)`
   ◯ none of the above

3. Given the following struct

```
typedef struct A{
    int x;
    char *info;
} a;
```

re-write the expression `a->info[1]` without using `->` or `[]` while maintaining the same functionality:

4. Select the print statement and loop condition that will create a program that will correctly print the command line arguments.

   for example the correct output for: `./a.out 134 apples horses` is: “134 apples horses”

```
int main(int argc, const char *argv[]) {
    for (_____________________) {
        --------------------
    }
}
```

Second blank:

   ◯ `printf(argv[i]);`
   ◯ `printf("%d ", argv[i]);`
   ◯ `printf("%s ", argv[i]);`
   ◯ `printf("%d ", argv[i+1]);`
   ◯ `printf("%s ", argv[i+1]);`

First blank:

   ◯ `int i = 0; i < argc; i++`
   ◯ `int i = 1; i < argc; i++`
   ◯ `int i = 0; i < len(argv); i++`
   ◯ `int i = 1; i < len(argv); i++`

5. Select the values for the two blanks to make `array` into an array of `n` pointers to strings.

```
array = ________ malloc(___________ * n);
```

First blank:

- ○ `(char)`
- ○ `(char*)`
- ○ `(char**)`
- ○ `(char***)`
- ○ `(*char)`
- ○ `(**char)`
- ○ `(***char)`

Second blank:

- ○ `sizeof(char)`
- ○ `sizeof(char*)`
- ○ `sizeof(char**)`
- ○ `sizeof(*char)`
- ○ `sizeof(**char)`

6. Consider the following incomplete code:

```
1 char x[13] = "Hello, ";
2 char y[13] = "Tychonievich";
3 ________
```

Which of the following, if placed in the blank on line 3 above, will result in the char array “`x`” storing parts of both the strings stored in “`x`” and in “`y`” without a buffer overflow?

Select all that apply:

- ○ `x = x + y;`
- ○ `strncat(x, y, 5);`
- ○ `strncat(x, y, 6);`
- ○ `strncat(x, y, 7);`
- ○ `strncat(x, y, 13);`
- ○ `strcat(x, y);`

7. The following code contains one or more memory errors.

```
 0 char *bad_shuffle(const char *array) {
 1    size_t length = strlen(array);
 2    char *new_array = malloc(sizeof(char) * length+1);
 3    for(int i = 0; i <= length; i++){
 4        int k = (length * rand()) / RAND_MAX;
 5        new_array[i] = array[k];
 6    }
 7    free(array);
 8    return new_array;
 9 }
10 int main(int argc, char *argv[]) {
11    char *s = bad_shuffle(argv[1]);
12    puts(argv[1]);
13    puts(s);
14 }
```

For each of the following errors, give the line where the error occurs. If the error does not occur, put an X in the box. If several lines are involved, write the first one where the error has definitely manifested itself.

☐ Stack buffer overflow

☐ Heap buffer overflow

☐ Use after free

☐ Memory leak

☐ Access uninitialized memory

8. In memory, there is following sequence of 6 bits: `110110`

   If you interpret this binary string as an 6-bit 2's complement number where the high-order bit is on the left, what is that number in decimal?

   If you interpret this binary string as an 6-bit unsigned number where the high-order bit is on the left, what is that number in decimal?

9. If the 32-bit integer `0x02010300` is to be stored in a **little** endian machine, it will be stored the same way as which array of 8-bit values?

   - ◯ `[0x02, 0x01, 0x03, 0x00]`
   - ◯ `[0x20, 0x10, 0x30, 0x00]`
   - ◯ `[0x00, 0x03, 0x01, 0x02]`
   - ◯ `[0x00, 0x30, 0x10, 0x20]`
   - ◯ none of the above

10. Consider two int variables, a and b:

Variable	Value	Address
a	0x000033	0x555540
b	0x330033	0x555548

   What is the result of `memcmp(&a,&b,1)`?

   - ◯ negative number
   - ◯ zero
   - ◯ positive number
   - ◯ none of the above.

11. The following code converts a character to a number, returning -1 on an error:

    ```
    int ctoi(char c) {
        if (c >= '0' && c <= '9') return c - '0';
        return -1;
    }
    ```

    Implement an `atoi` function. For reference we've included the manual page for `atoi` at the begin of the exam. You may (but do not need to) use `ctoi` in your solution.

```
int atoi(const char *str) {

}
```

12. How difficult was this final?

    ◯ too easy

    ◯ easy but fair

    ◯ fair

    ◯ difficult but fair

    ◯ too difficult

. . . .

You may use the space below as scratch paper