

2 hours (maximum), Closed Book

- You may use two sides of one sheet (8.5x11) of paper with any notes you like.
- This exam has XX pages, including this cover page and a blank page at the end. Do all your work on these exam sheets, use the backs of the pages if needed.
- Show all your work if you wish to be considered for partial credit.

During the final you will need to write LC3 assembly code, MIPS assembly code, and C assembly code. Here are some recommendations for information you might want on your cheat sheet:

- LC3, MIPS, and C syntax
- example activation records

NOTE: There will be more whitespace on the final.

NOTE: The below is just an example point breakdown.

Question	Points	Score
1	15	
2	20	
3	15	
4	10	
5	5	
6	5	
7	10	
8	20	

Name: _____

Email: _____

DO NOT TURN TO NEXT PAGE TILL YOU GET PERMISSION

1. [XX points] gdb

Consider the following sequence of code

```
165     int i;
166     struct mystruct s_array[2] = { { 0, {1, 2} }, { 3, {4, 5} } };
167     for (i=0; i<3; i++) {
168         s_array[0].a[i] = s_array[0].a[i] * s_array[0].a[i];
169     }
170
```

and the following sequence of commands and results in gdb

```
(gdb) break 166
Breakpoint 1 at 0x1d19: file recit-bomb.c, line 166.
(gdb) run
Starting program: /Users/mstrout/SVNWorkDirs/Classes/CS270Fall108/recits/R13/a.out
Reading symbols for shared libraries ++. done
...
Breakpoint 1, phase_5 () at recit-bomb.c:166
166     struct mystruct s_array[2] = { { 0, {1, 2} }, { 3, {4, 5} } };
(gdb) n
167     for (i=0; i<3; i++) {
(gdb) x /10d &s_array
0xbffff2a4: 0 1 2 3
0xbffff2b4: 4 5 6814 0
0xbffff2c4: 6814 -1073745176
(gdb) break 170
Breakpoint 2 at 0x1d6f: file recit-bomb.c, line 170.
(gdb) cont
Continuing.

Breakpoint 2, phase_5 () at recit-bomb.c:171
171     if (s_array[1].i!=3) {
(gdb) x /10d &s_array
0xbffff2a4: 0 1 4 9
0xbffff2b4: 4 5 3 0
0xbffff2c4: 6814 -1073745176
(gdb)
```

s_array is a local variable. Draw the layout of s_array on the stack that corresponds to the gdb output.

Why does the for loop modify `s_array[1]` even though only `s_array[0]` is being accessed in the loop?

What small modification to the loop initialization, loop condition, or loop update could make it so that `s_array[1]` is not modified by the loop? (more than one correct answer is possible)

Other possible gdb questions: Note that the example gdb question given above involves structures and therefore will not be part of the final unless it is an extra credit question. However, the form of the above gdb question is similar to what will be in the final. Study the `class-bomb.c` bugs that we covered in class and the `recit-bomb-questions`. Be able to answer questions where the code and the output of gdb are provided and you must draw the run-time stack and/or heap and then indicate what the bug in the code is and how to fix it.

3. [XX points] LC3 assembly

Why is JSRR necessary? Couldn't we just use JSR?

The following LC3 assembly code pushes the values in R7 and R5 onto the stack:

```
ADD R6, R6, #-1
```

```
STR R7, R6, #0
```

```
ADD R6, R6, #-1
```

```
STR R5, R6, #0
```

Draw the run-time stack after the above code executes. (Just draw the part relevant to the above code).

Add immediate values to the following list of instructions so that their execution results in the same run-time stack.

```
STR R7, R6,
```

```
ADD R6, R6,
```

```
STR R5, R6,
```

```
ADD R6, R6,
```

Exercise(s) from the book: 7.15, 9.5, 9.13

4. [XX points] Addressing Modes

The five addressing modes in LC3 are immediate, register, PC-relative, indirect, and base+offset. For each of the operands (including the destinations) in the following MIPS commands, which of those five addressing modes is being used?

```
addi $t2, $t1, 42
```

```
lw $t6, 4($t7)
```

5. [XX points] LC3 assembler

Given a small LC3 program example such as the one shown below (Hint: recall all of those test programs students submitted for PA4):

```

        .ORIG x3000
        AND R0, R0, #0 ;R0 = 0
        AND R1, R1, #0 ;R1 = 0
        AND R2, R2, #0 ;R2 = 0
        ADD R0, R0, #8 ;R0 = 8
        ADD R1, R1, #-2 ;R1 = -2
LOOP0   ADD R2, R2, #2 ;R2 = R2 + 2
        ADD R0, R0, R1 ;R0 = R0 + R1
        BRp LOOP0      ;LOOP0 x3005
        HALT
        .END
    
```

fill in the below symbol table and determine the PC offsets for the BR, JSR, LD, LDI, ST, and/or STI instructions.

Symbol Table

Label	Address

Exercise(s) from the book: 7.13

6. [XX points] I/O

HW1 I/O questions.

Exercise(s) from the book: 8.1, 8.7, 8.9, 8.11, 8.15

7. [XX points] C memory model

Using one sentence each to describe each of the following in terms of their inputs and outputs: C preprocessor, C compiler, assembler, and linker.

For the following function

```
int p = 256;

void foo()
{
    int x = p;
    int y = x + p;
    {
        int a = 2;
        x = a;
        int y = x + p;
        printf("%d\n", a);
        printf("%d\n", y);
    }
    printf("%d\n", x);
    printf("%d\n", y);
}
```

(a) draw the activation record using the LC3 convention for placing local variables,

(b) what is the output for the function?

Exercise(s) from the book: 12.5, 12.6, 14.19

8. [XX points] Run-time stack

Exercise(s) from the book: 14.5 (explain by drawing and then referring to the run-time stack), 14.7, 14.11, 17.5, 17.7 (draw the run-time stack at its largest when the input number is 3).

9. [XX points] Converting C to assembly and converting between LC3 and MIPS

Convert the following C function to MIPS assembly and then LC3 assembly (NOTE: don't forget the prologue and epilogue):

```
int foo(int x, int y, int z) {
    int i;
    for (i=0; i<x; i+=2) {
        y = y -1;
    }
    return x + y + z;
}
```

Convert the program in Figure 12.8 into LC3 assembly and then into MIPS assembly. (Hint: read the chapter)

Exercise from the book: 14.3

10. [XX points] Pointers and character string arrays

Using the below C code,

```
void foo() {
    int a = 4;
    int *p;
    int *s;

    s = &a;
    p = s;
    *s = 42;
    printf("%d\n", *p);
    printf("%d\n", *s);
}
```

to do the following:

- (a) What is the output of the above code?
- (b) Write the MIPS code that sets up the activation record for the function foo().
- (c) Write the MIPS code for the instructions “s=&a”, “p=s”, and “*s=42”.
- (d) Fill in the below chart with the values in the run-time stack after each of the instructions has been executed. Also indicate where the frame pointer and stack pointers point.

Addresses	Stack after s=&a	Stack after p=s	Stack after *s=42
0xD4			
0xD8			
0xDC			
0xE0			
0xE4			
0xE8			
0xEC			
0xF0			
0xF4			
0xF8			
0xFC			

(d) Do parts (a)-(c) for LC3 as well.

Exercises from the book: 16.3, 16.7

11. [XX points] The Heap

Given the following C code:

```
int * baz( int x ) {
    int * p;
    p = (int*)malloc( sizeof(int) );
    *p = x;
    return p;
}

void main() {
    int a=3, b=4, c=5;
    int *x, *y, *z;

    x = baz(a);
    x = baz(b);    // x = baz(b), the 'x' is NOT a typo
    z = baz(c);
}
```

(a) Write the MIPS and LC3 code for baz and main.

(b) Draw the stack and the heap at the end of main.

12. [XX points] Extra credit possibility: Arrays and data structures

Exercises from the book: 16.11, 19.1, 19.5