The University of Calgary
Department of Computer Science
April 2020
Take-Home Final Examination
CPSC 355

L01

Time Limit: 24 hours. The exam must be submitted to the CPSC 355 D2L Dropbox by 10:30 a.m. on Saturday, April 25.

This is an open book examination. You can use the following resources and tools to complete the exam:

- The notes and example code on the D2L CPSC 355 website
- The textbooks for the course
- Notes you have taken down in lectures and tutorials
- The gcc compiler and assembler
- A code editor (to create plain text files)
- A text editor such as Word
- A drawing tool (for diagrams)
- A scanner, if submitting hand-drawn diagrams

The exam requires you to create several files in order to answer the questions. These files will be submitted to a specially-marked Dropbox on D2L. Any text files and diagrams you submit should be readable as PDF or Word files. If submitting scanned hand-drawn diagrams, make sure they are readable as PDF or Word files. Source code files for the coding questions should be submitted as plain-text files, with the appropriate suffix (for example, question1.asm). Please name your files as specified in the exam, and do **not** compress or Zip your files in any way; they must be submitted individually to the Dropbox.

You must do this exam on your own and without any collaboration with any other person. We will use software tools to check for this, and any violation will be treated severely.

Mark distribution:

Question Type	**Number**	**Weight**	**Score**
Coding question:	1	35	_______
Coding question:	2	45	_______
Code Tracing question	3	10	_______
Code Tracing question:	4	18	_______
Total		**108**	_______

Question 1: Coding Question (35 points)

Translate the following program into the equivalent ARMv8 assembly code:

```
#define LOWER_LIMIT     -95
#define UPPER_LIMIT     +95
#define INCREMENT       5
#define FILENAME        "output.bin"
#define SIZE_OF_DOUBLE  8
#define AT_FDCWD        -100


int main()
{
  // Local variables
  register int fd, value;
  register unsigned long int n;
  double temp;

  // Open a file for writing in the current working directory
  fd = openat(AT_FDCWD, FILENAME, 0101, 0700);

  // Calculate and write double values to file
  for (value = LOWER_LIMIT; value <= UPPER_LIMIT; value += INCREMENT) {
    // Convert value to a double and then do calculation
    temp = (double)value / 100.0;

    // Write out the double to file
    n = write(fd, &temp, SIZE_OF_DOUBLE);
  }

  // Close the file
  close(fd);

  return 0;
}
```

Put your code into a plain-text file called *question1.asm*. Use System I/O to open and write to a binary file. Be sure to use floating point registers, variables, and operations where required. You may use m4 macros if you wish, and comments are not required. Use register *w19* for *fd*, register *w20* for *value*, and register *x21* for *n*. The local variable *temp* must be allocated in the stack frame for main().

Question 2: Coding Question (45 points)

Translate the following program into the equivalent ARMv8 assembly code:

```
// Custom datatype
struct set {
    int a;
    long int b;
    float c;
    double d;
};

// Global variable
double value;


int calc(int x, int y)
{
    int temp;

    temp = (2 * x * x) + (3 * y);
    return temp;
}


float func(float w, float z)
{
    return w / z;
}


void main()
{
    struct set myset;

    myset.a = calc(4, 5);
    myset.b = 10;
    myset.c = func(1.5, 0.75);
    myset.d = 0.5;

    value = myset.d + (double)(myset.c);
}
```

Put your code into a plain-text file called *question2.asm*. Implement all functions as closed subroutines, following the normal conventions for returning values, passing in arguments, and allocating local variables. You may use m4 macros if you wish, and comments are not required.

Question 3: Code Tracing Question (10 points)

Trace the execution of the following program:

```
        .global main
main:   stp     x29, x30, [sp, -16]!
        mov     x29, sp

        mov     w19, 0xa5a5a5a5

        lsl     w19, w19, 2             // step a
        mov     w20, 0xff00ff00
        and     w19, w20, w19           // step b

        asr     w19, w19, 4             // step c

        mov     w20, 0xffff0000
        eor     w19, w20, w19           // step d

        lsr     w19, w19, 4
        orr     w19, w19, 0xf           // step e

        ldp     x29, x30, [sp], 16
        ret
```

What are the contents of register `w19` after steps a to e above finish executing? Indicate your answers in *binary* and *hexadecimal* in a table similar to the one below. Put the table into a PDF or Word file called *question3.pdf* or *question3.doc* or *question3.docx*.

	Binary	Hexadecimal
step a:		
step b:		
step c:		
step d:		
step e:		

Question 4: Code Tracing Question (18 points)

Given the following ARMv8 assembly code:

```
        mystruct_i = 0
        mystruct_j = 8

        b_size = 16
        alloc = -(16 + b_size) & -16
        dealloc = -alloc
        b_s = 16

        .global main
main:   stp    x29, x30, [sp, alloc]!
        mov    x29, sp

        add    x8, x29, b_s
        bl     init

        // Point B

        ldp    x29, x30, [sp], dealloc
        ret


        lvar_size = 16
        alloc = -(16 + lvar_size) & -16
        dealloc = -alloc
        lvar_s = 16

init:   stp    x29, x30, [sp, alloc]!
        mov    x29, sp

        add   x9, x29, lvar_s

        mov   x10, 13
        str   x10, [x9, mystruct_i]
        mov   x10, 42
        str   xzr, [x9, mystruct_j]

        // Point A

        ldr   x10, [x9, mystruct_i]
        str   x10, [x8, mystruct_i
        ldr   x10, [x9, mystruct_j]
        str   x10, [x8, mystruct_j]

        ldp   x29, x30, [sp], dealloc
        ret
```

Draw 2 pictures of the stack, one as it appears at Point A, and the second as it appears at Point B in the code above. Put your 2 diagrams into a file called *question4.pdf* or *question4.doc* or *question4.docx*. Be sure to label all regions of memory and show the position of SP and FP. Also show the contents of memory on the stack where it has been initialized by the code; use ??? to indicate uninitialized memory on your diagram.