The University of Waikato
Department of Computer Science

COMPX203 Computer Systems
Exercise 1 – Introduction to the WRAMP System

Due Date: 30 March 2022

Objectives

This exercise is intended to provide an introduction to writing WRAMP assembly code, and running it on the Basys boards (or in the simulator). It assumes that you have already set up your lab environment by following the instructions in the "Getting Started" guide. Be sure to pick the correct guide for your situation. There is:

- Getting Started (Linux Hamilton and VM applicable to all)
- Getting Started Tauranga (using Cygwin under Windows)

After working through the Getting Started guide, you will all be in the position of being able to open up a terminal window providing a *bash* command-line prompt. Within the *bash* terminal, your own files are stored under **/home/<your-user-name>/** (on Linux) and **/cygdrive/h/cygwin-home/** (under Cygwin). We refer to this as your home directory. There is also a shared directory area for the course that we will be making use of. This is the same whether using Linux of Cygwin, as is: **/home/compx203/**.

To help with using the WRAMP hardware in this lab, you'll be using a few library functions which are implemented in the file:

/home/compx203/ex1/lib_ex1.o

The subroutines provided by this library are listed **at the end of this document**. You can link directly against this file by providing the full path to *wlink*; there's no reason to copy it to your own directory.

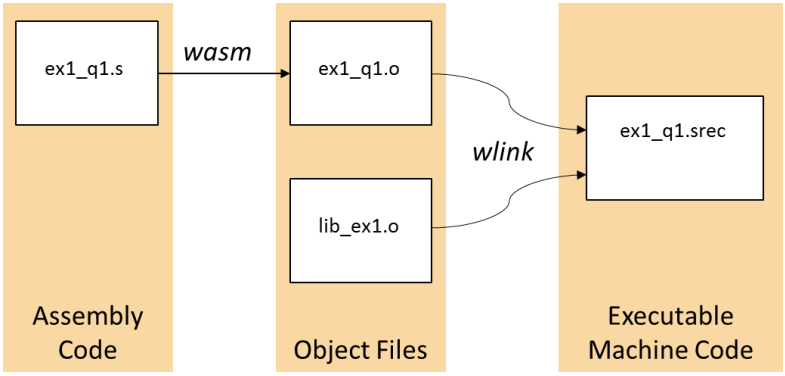

Figure 1: Assembly and Linking Example (Question 1)

*It is highly recommended that you read this entire specification **before** beginning to write any code.*

Assessment

This exercise contributes **10%** towards your **internal grade**, to be assessed in two parts:

- The correctness of your source code, to be submitted via Moodle **(7%)**
- Your completion of an online quiz about the exercise, also on Moodle **(3%)**

Both of these parts must be completed on or before the due date.

This is an individual exercise; while you may discuss the assignment with others in general terms, you must write your own code, and complete the quiz by yourself.

The name of the S-Record file must match the following format:

ex*[exercise number]*_q*[question number]*.srec

For example, the second question in this exercise would be called ***ex1_q2.srec***

Questions

1. Write a WRAMP assembly program that continuously (i.e., is in an infinite loop) reads a binary value from the 8 least significant switches, and outputs this value to the two rightmost seven-segment displays as a hexadecimal number. A switch that is down represents a '0' bit, while a switch that is up represents a '1' bit. For example, if the switches are set to the pattern shown in **Figure 2**, the seven-segment displays should show **6D**. Your source file should be called *ex1_q1.s*

2. Modify a copy of your program from **Question 1,** so that it continuously counts the number of switches currently in the 'up' or '1' position, and outputs the result to the seven-segment displays. For example, if the switches are set to the pattern shown in **Figure 2**, the seven-segment displays should show **05**, since three switches are down. This source file should be called *ex1_q2.s*

3. Modify a copy of your program from **Question 2** to encrypt the count of set switches before displaying it on the seven-segment display. The encryption algorithm to be applied is a simple mapping function defined in **Table 1**. For example, if three switches are up, your program should output **0D**. This source file should be called *ex1_q3.s*

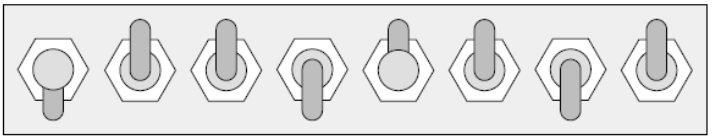

Figure 2: Example Switch Pattern

COUNT	OUTPUT (HEX)
0	A3
1	22
2	6B
3	0D
4	49
5	C0
6	7F
7	B8
8	31

Table 1: Mapping Function for Encryption Process

Submission

You are required to submit all source files that **you** have written; you do not need to submit any files that we have provided, nor any files that have been generated by a tool, e.g., *wasm* or *wlink*. Each file must follow the naming conventions indicated below.

For this exercise, the required files are:

- **ex1_q1.s** (Question 1)
- **ex1_q2.s** (Question 2)
- **ex1_q3.s** (Question 3)

These files should be compressed into a single "tar gz**"** archive called **firstName_lastName.tgz** (replace *firstName_lastName* with your own name) before being submitted to Moodle. You can create this archive from the terminal, using the command:

tar -cvzf firstName_lastName.tgz ex1_q1.s ex1_q2.s ex1_q3.s

Tar is a utility for combining multiple files into a single archive, called a *tarball*. The option 'c' means create the archive file, the 'z' says to use compression, the 'v' puts the program into verbose mode so information about what the program is doing is printed to the screen, and 'f' means the output filename will follow as the next argument. The arguments that follow this are the files to include in the 'tarball'.

Please be aware that a failure to follow these instructions correctly may result in a grade penalty. If you are unsure that what you've done is correct, please ask and we'll help you out.

I/O Routines (lib_ex1.o)

putch

Parameters: $2 The ASCII-encoded character to transmit
Returns: --- *none*

*Transmits a single ASCII character through the first serial port. This character will appear in **remote** if using the physical REX boards, or the **Serial Port 1** form if using the simulator. The character to send must be stored in register $2.*

putstr

Parameters: $2 The *address* of the first ASCII character to send
Returns: --- *none*

*Transmits a null-terminated ASCII string through the first serial port, which is passed by reference using register $2. This string will appear in **remote** if using the physical REX boards, or the **Serial Port 1** form if using the simulator.*

readswitches

Parameters: --- *none*
Returns: $1 The value represented by the switches

Reads the current value represented by the switches into register $1. The lowest 8 bits (i.e. bits 0 to 7) will have the value of the corresponding switch (on or off), while the remaining bits (8 to 31) will be set to zero. The down position is represented as a binary '1'.

writessd

Parameters: $2 The number to write to the seven segment displays
Returns: --- *none*

Writes the lowest 8 bits of register $2 to the seven segment displays, shown as hexadecimal digits. The higher bits in the register (8 to 31) are ignored.