Carleton University

Department of Systems and Computer Engineering

SYSC 3006 (Computer Organization) summer 2020

Lab / Assignment 5

Prerequisites

Lab / Assignment 4, series 6 part 1, 2 and all related materials posted on cuLearn.

Goal

- Program a given computer system.
- Implement and test using the Assembler and Debugger (Logisim) circuit.

Introduction

Until this lab, we have entered instructions manually into the main memory. In this lab we are going to take one step further toward the software. We are going to write assembly language into an editor. Then you are going to use an Assembler to assemble source code for the instruction syntax discussed in class into a memory image (called Object code). A Logisim Debugger circuit will be provided for you. The Debugger provides an interface for controlling software execution. You are going to take your object code generated from the assembler in step 1, then load it into the Debugger's Main Memory component to execute and debug your code.

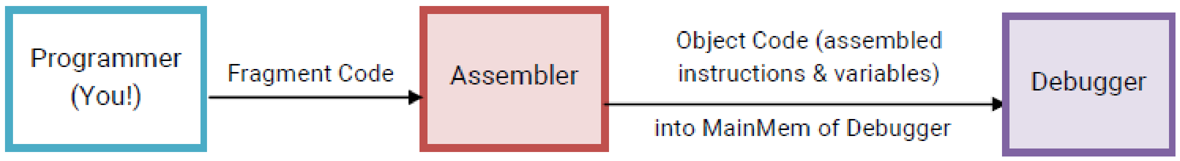

Figure 1. Overview of the software development and debugging (testing) workflow in this lab.

Logisim files setup and components information

The Software Tools needed for this lab are posted on the course website under the tab "Resources". The tools include the Assembler and the Debugger circuit. Do not change any wiring inside the Debugger circuit.

Tutorial Assembling and debugging Fragment 1

To understand how to assemble and debug a given fragment, follow this video tutorial (5 minutes; no audio) which uses Fragment 1 as an example. Ensure that by the end of the tutorial, you have similar main memory values (Fragment1_MMem.txt). The video is sufficient for the purposes of this lab. For more details, you can always refer to the pdf guides accompanying each tool on cuLearn: Debugger User Guide and Assembler User Guide.

As discussed in the Debugger User Manual: Be sure that the clock has been enabled to run freely in the simulation: Simulate → Ticks Enabled. **Note: Ticks are disabled by default when Logisim is loaded, so you must enable ticks before anything will happen.**

DCD Directive

The DCD (DeClare Data) directive is used to declare a memory location to be used as a variable. The general syntax for the directive is:

[Label] DCD [numeric-literal] [; comment]

Anything enclosed in [and] is optional. If present, the Label is the name of the variable. If present the numeric literal defines the initial value that should be loaded into the variable's memory location when the program is loaded. As always, comments are optional.

Here are some examples:

```
      DCD        ; reserves an un-named, un-initialized word
X     DCD        ; reserves a memory word for an un-initialized
                   variable named X
Y     DCD   #12  ; reserves a memory word for a variable named Y,
                    initialized to 12 (decimal)
X     DCD   #0x10 ; reserves a memory word for a variable named Z,
                   initialized to 10 (hex)
```

Note that decimal values for DCD numeric-literals may be specified as negative (using 2's complement encoding). For example

```
Minus1 DCD #-1   ; reserves a memory word for a variable named Minus 1,
                   initialized to -1
                 ; the variable is initialized to the value 0xFFFFFFFF
                   i.e. -1 under 2's complement
```

Note: Also Review the Debugger User Guide and the Assembler User guide. These documents are distributed with the tools and provide essential information on how to use the tools. The documents do not describe the computer system instructions or the associated assembly language syntax … these are discussed in the lecture slides. The "breakpoint instruction" is discussed in the Debugger User Guide.

Your Assignment

Use the included editable .docx file (MS-Word) for your answers, then save it as PDF file before submitting it. Here we refer to tables to be filled in, they are not shown here but they do in the assignment .docx file.

Part 1 – Fragment 1 [3-mark/5]

Questions about Fragment1 SRC.txt (included in lab 5.zip)

1. [0.25-mark] What is the high-level objective (purpose) of the code fragment? Explain the objective in terms of the net effect of the fragment on the variables it modifies.

2. [0.25-mark] Write C-like pseudocode that accomplishes the same objective (see below for an example of C-like pseudocode). Pseudocode is code that is concise but does not necessarily have a compiler. In this case (as in class), the pseudocode refers to registers instead of variables. This pseudo code also omits the C-language ";" separators and uses assembly language-style comments (;) instead of C-style comments (//).

Consider the following C-like pseudocode that processes an array to replace every element with its absolute value:

```
    for ( R11 = 0; R11< Arr_Size; R11++ )   ; R11 is index into array, index = 0
      {  if ( Arr[ R11 ] < 0 )
           { Arr[ R11 ] = abs( Arr[ R11 ] ) ; abs is absolute value function
           } //end_if
      } //end_for
```

3. [1-mark] Starting after the SkipOverVariables declaration, add comments to the instructions that document what is being done ... the comments should be at the level of the pseudocode objective, not at the RTL level. For example, consider the instruction:

   MOV R4, #1

   An RTL-level comment for the instruction might be: "; move #1 into R4", which is accurate but says nothing about the net programming objective (i.e. why is loading #1 useful in the context of the program's objective?). A more appropriate comment might be: "; R4 = address of Arr_Size".

```
B SkipOverVariables

       ; Arr is an array of 3 words
Arr_Size DCD #3
Arr
   DCD #20 ; first (0 - th)element of Arr = 20
   DCD #-4; second (1 - th)element of Arr = -4
   DCD #0 ; third (2 - th)element of Arr = 0

SkipOverVariables
MOV R2, Arr              ; x
LDR R3, [ Arr_Size ]     ; x
CMP R3, #0               ; x
BEQ Done                 ; x
SUB R3, R3, #1           ; x
```

```
                              ; x
                              ; x
                              ; x
Loop                          ; x
LDR R5, [R2, R3 ]             ; x
                              ; x
                              ; x
                              ; x
ADD R5, R5, #10               ; x
STR R5, [R2, R3]              ; x
SUB R3, R3, #1                ; x
BPL Loop                      ; x
                              ; x

Done
   DCD #0xFFFFFFFF ; breakpoint instruction
```

4. [0. 5-mark] When the fragment is executed, how many instructions will be executed (including the breakpoint instruction)?

5. [0.5-mark] When assembled, how many words of memory will the fragment occupy?

6. [0.5-mark] Assemble and run Fragment 1 SRC. To validate running the fragment in your lab report, submit the contents of Main Memory RAM before and after executing the fragment. (Hint: right-click on RAM → Save Image ...).

Before execution:

```
v2.0 raw
x
```

After execution:

```
v2.0 raw
x
```

Part 2 – Fragment 2 [2-mark/5]

A template for the assembly language code to implement the search is provided in Fragment2 SRC.txt. In the SRC file, the original pseudocode is present as comments (along with some additional comments) to help guide code development. The template has "***" instead of some necessary details.

1. [1.5-mark] complete the code by replacing all occurrences of "***" with the necessary details and execute the processing for the data values in the template. Do not add additional instructions. Submit your completed (working) SRC fragment.

Edit your final completed Fragment2 SRC code here

2. [0.5-mark] Assemble and run Fragment 2. To validate running the fragment in your lab report, submit the contents of Main Memory RAM before and after executing the fragment. (Hint: right-click on RAM → Save Image ...).

   Before execution:

```
v2.0 raw
x
```

   After execution:

```
v2.0 raw
x
```

Submission deadline

Must be submitted on cuLearn, locate (Assignment 4 submission) and follow instructions. Submission exact deadline (date and time) is displayed clearly within the Assignment 4 submission on cuLearn.

Note: If you have any question please contact your respective group TA (see TA / group information posted on cuLearn) or use Discord class server.

Good Luck