Lab 8 – Assembly Programming

CS1050 Computer Organization and Digital Design

Dept. of Computer Science and Engineering, University of Moratuwa

Learning Outcomes

In this lab, we will learn Assembly programming and interfacing simple input and output devices. After completing the lab, you will be able to:

- design and develop simple Assembly programs to achieve given objectives
- interface simple input and output devices to the microprocessor
- verify their functionality via simulation

Introduction

Assembly is the most basic programming language available for any microprocessor. In some cases, it is the only language for programming simple microprocessors and microcontrollers. While Assembly lacks high-level conveniences such as variables and functions, as well as it is not portable among various families of microprocessors, it can produce efficient code that runs faster. It also gives programmers the insight required to write effective code in high-level languages. Therefore, learning assembly language is well worth the time and effort of every serious programmer.

In this lab we will use the Microprocessor Simulator (a.k.a. smz32), developed by Neil Bauers, to develop and simulate Assembly programs. BASYS 3 board, we have been using so far, can also be programmed in Assembly. However, it requires installing and configuring the Xilinx PicoBlaze microcontroller core designed for FPGA devices. This requires a deeper understanding of embedded systems, VHDL or Verilog programming, and its own instruction set. We will not take this route, as those concepts are beyond the scope of this class. Interested students may look at the “Lab 3: Xilinx PicoBlaze Flow Lab” by Xilinx (available on Moodle).

The smz32 simulator emulates an 8-bit CPU that is similar to the lower 8-bits of the 80x86 family of microprocessors. It has 4, 8-bit general purpose registers (namely AL, BL, CL, and DL) and 256 bytes of RAM. smz32 has 16 input/output ports, and several simulated peripherals (e.g., keyboard, 7-segments, signal lights, and stepper motor) are attached to ports 0 to 5. It has its own assembler too. You can single step through programs or continuously run programs while varying CPU clock speed. Refer “Microprocessor Simulator V5.0 Help” for further details (available on Moodle).

In this lab we will first try a couple of examples given with smz32. We then modify some of those examples to implement detailed behavior (e.g., one using signal lights). Finally, you will develop a new Assembly program to calculate the products of integers from 1 to 5 and show the results on 7-segment displays.

Getting Familiar with smz32

Step 1: More on smz32.

To get a better idea about the smz32 simulator carefully read pages 2 - 6 on "Microprocessor Simulator V5.0 Help".

Step 2: Running smz32.

Create a new folder and name it as **Lab 8**. Download **smz32v50.zip** file from Moodle and extract it to **Lab** 8. Files can be also downloaded from http://www.softwareforeducation.com/sms32v50/

Double click on **sms32v50.exe** to execute the program.

Identify different components of the interface. Refer pages 14 - 16 on "Microprocessor Simulator V5.0 Help" for details.

Running Examples

Step 3: Load **01FIRST.ASM** file using **File → Open** menu.

Try to understand the source code. Some of the sample programs are explained in "Microprocessor Simulator V5.0 Help". Summary of Assembly instructions is given on page 58 (topics can be accessed from the links given on page 1).

Step 4: Stepping through the program.

Run the program one instruction at a time using **Step** button. As you step through the program, carefully observe how the register values change.

The entire program can be executed using the **Run F9** button (or **F9** function key). In this mode, the speed of execution can be controlled using **Slower** and **Faster** buttons. This can also be accomplished through the **Configuration tab**. Use the **Cpu Reset** button to reset the CPU and in turn the program.

Step 5: Modifying Assembly code.

Modify the Assembly code to subtract, divide, and multiply the 2 numbers using SUB, DIV, and MUL instructions. Save each as a separate Assembly (.asm) file, e.g., you may use file names like 01FIRSTSub.asm and 01FIRSTMul.asm.

Once you modify a program, it needs to be reassembled using the **Assemble** button. Rerun the program and make sure it functions correctly.

Step 6: Run several more samples.

Get familiar with the simulator and Assembly programming by at least running **03MOVE.ASM**, **04INCJMP.ASM**, and **99Heater.asm** examples.

Modifying Existing Examples

Step 7: Open **02TLIGHT.ASM** program. Carefully observe how the registers change while you simulate the program to understand how it works.

Step 8: Making signal lights do what they are supposed to do.

In this example, signals lights flash with every run. This is usually the faulty behavior of signal lights. Let us make it real by controlling the time that each light stays on. Suppose vehicles facing the signal lights on the right has a high priority (i.e., the Green light stays on for a long time).

Change **02TLIGHT.ASM** program to simulate the following behavior:

Time (CPU Cycles)	Lights on Left	Lights on Right	CPU Cycles
t	Red	Green	10
t + 10	Yellow	Yellow	1
t + 11	Green	Red	5
t + 16	Red	Green	10
t + 26	Yellow	Yellow	1
t + 27	Green	Red	5
t + 32	Red	Green	10
…	…	…	…

To simplify the implementation, you may ignore the time (CPU cycles) used by other instructions.

For more efficient ways of introducing a delay to a code see **06PROC.ASM** file. You get full marks for using a loop to introduce longer delays.

Make sure to comment the code too.

Save your assembly code as **<indexNo>_08.ASM**.

Step 9: Open **99SEVSEG.ASM** program, simulate and understand how it works.

Step 10: Modify the program to show the last 2 digits of your index number.

For example, if your index number is 200234Z, then you should display 34. Save the file as **<indexNo>_10.ASM**.

Creating a New Program

Step 11: Write a new assembly program to multiply all integers from 1 to 5.

While an equation can be derived to do this calculation, let us use a loop to perform the multiplication 1 x 2 x 3 x 4 x 5. Show the final answer as a hexadecimal value using 2, 7-segment displays. Make sure to comment the code too.

Name the source file as **<indexNo>_11.ASM**. Test the program by assembling and simulating it.

Demonstration and Lab Report

Step 12: Code the solutions to Steps 8, 10, and 11 and upload to the moodle as separate ASM files. Name your files as <indexNo>_08.ASM <indexNo>_10.ASM <indexNo>_11.ASM or otherwise, it will not be graded.

Step 13: Lab Report

You need to submit a report for this lab. Your report should include the following:

- Student name and index number. Do not attach a separate front page
- State the assigned lab tasks in a few sentences
- All Assembly files that you are required to either modify or create.

- Selected screenshots of signal lights and 7-segment displays to demonstrate your solution works.

Submit the lab report (and three separate ASM files) to the moodle.

Prepared By

- Dilum Bandara, PhD – Jan 20, 2015.
- Updated on Oct 25, 2018.