Computer Systems, Spring 2019

Week 4 Lab

Assembly Language Instructions

This lab exercise is an introduction to computer architecture, using Sigma16. This material is essential for the rest of the course. The aims are

- to solidify your understanding of the basic instructions: add/sub/mul/div for doing arithmetic, load/store for copying variables between memory and registers, and lea for putting a constant into a register.
- to learn how to write a complete program that terminates properly and that defines its variables;
- to see how to assemble a program and run it on the emulator.

First solve the paper problems *with paper and pencil*. It is important to understand what you're doing before starting to type code into a computer. Later, as we get into more advanced programming, we will take the same approach: start by working on paper and really thinking about what you are doing, and only then typing it into the computer.

The exercise is not assessed; there is nothing to hand in, and you are encouraged to work with your friends and neighbors.

Problems to solve on paper

1. Suppose we have some variables in the registers. Just use the register name as a variable name: for example, think of the register R4 as a variable name. Avoid using R0 and R15; thus the variables should be restricted to R1, R2, ..., R14. Write a sequence of instructions that carry out this calculation: `R3 := R1 + R2 * R3`. Write a comment on each instruction that describes what it does. Just use arithmetic instructions, and just write a code fragment, not a complete program.

2. Now suppose that we have variables x, y, z in memory. Write a sequence of instructions that carry out this calculation: `z := x - 13 * y`. Write a comment on each instruction that describes what it does. The comments should be as informative as possible: for example, if R1 and R2 contain variables x and y respectively, then a good comment for add R5,R1,R2 would be `R5 := x+y` and a comment like `R5 := R1+R2` gives little "added value".

3. Hand-execute the following program fragment. After each instruction, show what register has changed, and its new value. Add a comment to each instruction that describes what it does.

```
lea   R2,23[R0]
lea   R3,4[R0]
lea   R5,30[R0]
```

```
add   R3,R5,R3
add   R4,R2,R0
sub   R6,R4,R5
```

4. Explain the difference between the `load` and `store` instructions. Write the instructions needed to perform

```
R4 := x
total := R5
```

5. Explain the difference between the `load` and `lea` instructions. Write the instructions needed to perform

```
R1 := 27
R2 := x
```

6. Sometimes in programming you need to swap the values of two variables. For example, suppose that x = 29 and y = 67. After swapping the variables, the values are x = 67 and y = 29.

   (a) Explain why you can't just write x := y; y := x.

   (b) Write assignment statements that swap the two variables in a high level language.

   (c) Write assembly language instructions that swap the two variables. Hint: you'll need to use registers and some load and store instructions; there is no way to do this without passing the data through registers.

   (d) Discuss the two versions of swap that you wrote (the high level language version and the assembly language version).

   (e) Write a complete assembly language program that defines two variables, x (with initial value 3) and y (with initial value 19). The program should swap the variables and then halt. The first line of the program should be a comment that gives the name of the program: `; Program Swap`.

Problems using the computer

Circuits

If you like, try experimenting with some small circuits, especially the ones that are used in the RTM circuit. You can do this with any circuit simulator, and it's also valuable to do simple circuit design and simulation on paper without any software tools.

Sigma16

We will be using Sigma16 for the rest of the course. This lab gives a brief introduction.

Follow the instructions on Moodle to find the Sigma16 application and launch it.

The documentation contains a tutorial called *Tutorial: Run an example program*. (Click on that in the table of contents.) Follow the steps in the tutorial, which guides you through running an example program, which is called *Program Add*.

Study the example program, which calculates `z := x + y`. Be sure you understand what each of the instructions does. Hand execute the program: write down the effect of each instruction. To get you started, the effect of the first instruction is to set R1 to 23 (the hex value is 0017). Here is a listing of the program:

```
; Program Add
; A minimal program that adds two integer variables

; Execution starts at location 0, where the first instruction will be
; placed when the program is executed.

      load   R1,x[R0]   ; R1 := x
      load   R2,y[R0]   ; R2 := y
      add    R3,R1,R2   ; R3 := x + y
      store  R3,z[R0]   ; z := x + y
      trap   R0,R0,R0   ; terminate

; Static variables are placed in memory after the program

x     data  23
y     data  14
z     data  99
```

Run the Add program as follows:

1. Launch the Sigma16 application (see Moodle for instructions on how to launch it).
2. Click the Editor tab. Either type in the Add program, or click Open and navigate to Examples/Simple/Add.asm.txt.
3. Click the Assembler tab.
   - Click Assemble. This will show the result of translating the assembly language program to machine language.
   - Find the address of the variable `x`. There are several ways to do this. In the Assembler pane, there's a table at the bottom that shows the value of each label; this says that Symbol x has value 0008. (This does *not* mean that the variable x contains 8; the value of a symbol is the corresponding address in memory. In other words, the variable x has value 23 and its address is 0008.)
4. Click the Processor tab
   - Click Boot

- Look at the first instruction, think about what it means, and *predict the result of executing it*. Now click Step, which executes one instruction. Check the registers to see what actually happened. In the assembly listing window, the instruction that has just executed is highlighted in red, and changes to registers or memory are also highlighted in red. Any registers or memory locations that have been accessed are highlighted in blue.
- Step through the rest of the program until it terminates with the trap instruction. Each time, *first* think about what the next instruction does and predict its effect; *then* click Step and see if you were right. Avoid the temptation of just blindly clicking Step repeatedly!
- To run the program again, click Boot which will reinitialise the memory. You can click Run or Run Display to execute the instructions automatically.

Now go back to the Editor pane and type in Program Swap (which you wrote above). Assemble it, boot it, and step through it in the Processor pane.