Computer Systems, Spring 2020

Week 5 Lab

Control Structures

The aims of this lab exercise are

- To understand how to translate high level statements to low level statements. The low level statements are: assignments, goto, if-then-goto.
- To understand how to translate low level statements to assembly language.
- To understand how to translate assembly language to machine language.
- To solidify your understanding of the instructions by hand executing some small programs.
- To understand how the machine uses the control registers while executing a program.
- To learn more about how to use the assembler and emulator in the Sigma16 app.

Notice that we are not simply writing assembly language programs from scratch. We are developing them systematically by

1. writing the algorithm in a higher level language
2. translating this to low level language (with goto and if-then-goto but without any complex control structures)
3. translating the low level language to assembly language
4. using hand-execution and comparing this with what happens when the computer runs the program

This systematic approach is important for several reasons. It provides a good way to understand what high level language statements mean, and it also provides a good way to develop assembly language programs in a clean and professional style. It's a lot easier to get an assembly language program to work if you follow this systematic approach. Experience has shown that programmers who skip the intermediate steps (the high level, the low level, the hand executions) sometimes end up spending *more* time than programmers who go through the whole process, and they end up with a lower quality result.

All three versions of each program (high level, low level, assembly) should be given in one file. The high and low level statements are comments; only the assembly language instructions are not comments. And each instruction should contain its own comment. The program will end up containing more comments than code.

As you write the code, refer back to the lecture slides. Follow the style used in the examples. Pay attention to the comments and the indentation and general layout. You may use either the "statement by statement style" or the

"register variable style". Most of the example programs use the register variable style, but you can use the other style if you prefer.

Each problem has a part to do on paper and a part to do using the computer. Remember: first solve the paper problems *on paper* and then try them out on the computer.

The exercise is not assessed; there is nothing to hand in, and you are encouraged to work with your friends and neighbors. What's important is that you understand everything, and it's fine if you get some help on the road to this understanding. This material is extremely important: it will be needed for the following topics in the course, for the future lab exercises, and for the examination.

1 An assignment statement

Here is a small program: just one line of code! The variables have these initial values: `x` = 0, `a` = 20, `b` = 13.

```
x := a - b;
```

1. **To do on paper.** Since this program is just an assignment statement, it is already in the low level form. Translate it into a complete Sigma16 assembly language program. The program should (1) load `a` and `b` into registers, (2) do the arithmetic, (3) store the result into `x`, (4) terminate execution, and then (5) there should be data statements to define the variables and given them initial values. See the example program Add.asm.txt — your program should be similar.

2. **To do on paper.** Hand-execute your assembly language program. To do this, write a box on paper for each variable in memory and for each register that you're using. Label each box, and write in the initial values. You can write the contents of a box in decimal or hexadecimal, as you prefer, as long as you know which it is! For example, if your program uses `R1`, then there will be a box labelled `R1` and it will contain 0; the box for `a` should contain 13 (the initial decimal value) or 000d (that is 13 in hexadecimal). Execute the program one instruction at a time, and update the boxes with the new contents.

3. **To do on the computer with Sigma16.** Enter your program in the Editor tab, assemble it in the Assembler tab, and execute it in the Processor tab. When you execute the program, just step through one instruction at a time. After each instruction executes, look at the display and see whether the machine did the same thing as your hand execution. Don't just run the program at full speed; check the result of each instruction.

2 An if-then-else statement

Here is another short program. The variables have these initial values: `a` = 0, `x` = 3, `y` = 20.

```
if x>y
   then { a := x; }
   else { a := y; }
a := 2 * a;
```

1. **To do on paper.** Translate this to a program containing only low level statements.

2. **To do on paper.** Translate the low level program to a complete Sigma16 assembly langauge program.

3. **To do on paper.** Hand-execute the program. Keep track of which instruction is being executed, as well as the values of the variables and registers you're using.

4. **To do on the computer.** Enter the program into Sigma16, assemble it, and execute it one instruction at a time. For each instruction, predict what it should do and compare your prediction with what the computer actually does.

5. **To do on paper.** Change the program so that the initial value of `x` is 25. This requires changing just one line of code, the data statement that defines `x` and gives its initial value. Hand execute the program.

6. **To do on the computer.** Change the program in the computer and execute it one instruction at a time. Again, compare your hand execution with what the computer actually does.

3 A while loop

Here is a program that contains a while loop. The initial value of `n` is 5; the other variables have initial value 0.

```
sum := 0;
i := 0;
while i < n do
   { sum := sum + i;
     i := i + 1;
   }
```

1. **To do on paper.** What is the difference between saying "all of the variables have initial value 0" followed by the statement "`n := 4`", and saying "the initial value of `n` is 4"?

2. **To do on paper.** Translate this program into a lower level, which contains just assignment statements, goto statements, and if-then-goto statements.

3. **To do on paper.** Translate the lower level program into assembly language.

4. **To do on paper.** Hand execute the assembly language program. Note the final value of `sum`. But don't just look at the program and realise what the final value of `sum` should be—please actually hand execute the program. (Don't worry: we won't always be hand executing complete loops! But this is worth doing now, because it is essential that you fully understand how this loop works.)

5. **To do on the computer.** Run the program on the Sigma16 app, stepping through one instruction at a time. For each instruction, check to see if your hand execution does the same thing as the computer. Check to see if the final value of `sum` is right.

4 Machine language and control registers

1. **To do on paper.** The lecture slides contain both the example Program Add and an explanation of its machine language code. Try converting the assembly code to machine language for yourself, and compare with the lecture.

2. **To do on the computer.** Enter the Add program into the editor. You can do this just by going to the Editor tab and clicking *Example*.

3. **To do on the computer.** Assemble the program, go to the processor tab, and boot the program. Look at the memory display and check that the machine language code is where it should be in the memory.

4. **To do on the computer.** Step through the program using the emulator, one instruction at a time. Check the `pc`, `ir`, and `adr` registers before and after each instruction. Be sure you understand what these registers contain, and why.