University of Rhode Island
ELE 208: Intro to Computing Systems

Style Guidelines for Assembly Language Programming

Beyond making sure that your programs function correctly, it is important that they are documented thoroughly and correctly. Good programming "style" makes your programs easy to read and to understand, both by you and by anyone else who needs to understand or to change your code in some way. Furthermore, programs written following appropriate style guidelines are easier to write and debug than programs written in a haphazard, disorganized style. While the assembler does not care about the programming style you use, the goal of good style is to make the program easy for people to read.

While you may think that a short little program is not worth the effort required to document it appropriately, the Y2K bug has shown us that programs and programming decisions can long outlive their authors and their initial circumstances. Thus, it is important that you always carefully document all of your code. Always assume that someone, yourself included, will have to return some day in the distant (or not so distant) future to understand the code you are writing today. The programming style you learn now will determine whether this becomes an easy or an impossible task.

The following guidelines will help you write well documented, easyto understand assembly language programs.

1. Your assembly language programs should have the following comment block at the top of the file.

```
;
; Name:
; Course number:
; Term:
; Lab/assignment number:
; Due date:
;
```

2. Insert the following comment block at the start of the program and at the start of each subroutine.

```
; SUMMARY
; Briefly describe what the program or subroutine does.
;
; INPUT
; Describe what inputs are expected and explain any error checking the
; program performs on the inputs.
;
; OUTPUT
; Describe the output produced by the program, or any values the
; program or subroutine is expected to return.
;
; ASSUMPTIONS
; Describe any particular assumptions your program makes. For instance,
; whether a certain register is always valid, is destroyed on return,
```

```
; and so on.
;
; REGISTERS
; List all of the registers used by the program, what they are used
; for, whether they are caller or calleesaved, and any other
; relevant information.
;
```

3. All symbolic labels must be meaningful. It is helpful to use both lower and upper case to help make the names easy to read. For example, use SaveR1 as a label instead of something more cryptic, such as SR1. Other good examples include names such as, EndProgram, ReadKeyboard, PrintResult, ShiftLeft, and so on.

4. Add a comment after each instruction to describe what it is doing with regard to the overall program.

5. Use white space (i.e., spaces, tabs, carriage returns) generously to emphasize the structure of your program and to make it easy to read. Make sure you indent and align statements and comments appropriately.