

GENERAL GUIDELINES

Properly documenting your code is a critical element of the programming process. But documentation is not only for other readers of your code, it can also help you immensely if you do it properly. Too often, novice programmers wait until after their code is complete, then go back and add comments. If you do this, you gain absolutely nothing from the documentation process. By documenting your code as you go, you force yourself to continuously describe your program logic and methods, which helps you to more readily see logical errors and inconsistencies in your program. This way, you reap the benefits of your documentation, instead of just making it a chore at the end of the process with no value to you (other than avoiding deductions to your homework grades).

The purpose of **comments** in your program is to explain why a line of code is there, not to simply reiterate what the line of code says. Ample comments within your code should explain your program logic so that someone reading it for the first time can see your thought process. Good commenting also allows programs to be debugged (you'll be doing a LOT of this) and maintained much more easily. Using descriptive subroutine and label names goes a long way to making programs more self-documenting, and is a must for this course and good programming practice.

For example, **DON'T** do this;

```
; Name: subroutine1
; Description: changes display
; Assumes:      r0 - values

; Returns:      None
; Modifies:     r1-2
subroutine1
    mov r2, #768 ;load 768 in r2
    eor r1,r1,r1 ;xor r1 with itself
    mvn r1, r1
    strb r1, [R2] ;write data
    add r2,r2,#1 ;increment r2
    mov r1, r0, LSR #8
    strb r0, [R2] ;write data
    strb r1, [R2, #-1] ;write data
    mov r15, r14
```

Note that the label (the subroutine name) told you absolutely nothing about its purpose, and the comments told you nothing that wasn't already apparent from the code. Poor documenting techniques have rendered this very simple code fragment nearly unreadable...

DO this instead;

```
; Name: UpdateDisplay
; Description: Updates multiplexed seven segment display
;             Digit latch is at address 300h.
;             Segment latch is at address 301h.
; Assumes:    R0 - Segment driver latch value
;             R1 - Digit select latch value
; Returns:    None
; Modifies:   R2-3
UpdateDisplay
    mov  R2, #0x300      ;load display base address
    mvn  R3, #0          ;load all 1s
    strb R3, [R2]        ;turn off all digit drivers
    strb R0, [R2, #1]    ;write segment latch
    strb R1, [R2]        ;write digit latch
    bx   LR              ;return
```

Note that the subroutine header now succinctly explains UpdateDisplay's purpose and usage, and the columnar formatting makes it easy to distinguish (and read) labels, mnemonics, operands and comments. Alignment and neatness are **required** to enhance the readability of your code. All labels must start in the first column, then all mnemonics, operands, and comments should be aligned. **Whitespace** (blank lines) should be used to visually separate different functional areas of code. Judicious use of whitespace will enhance program readability.

SPECIFIC REQUIREMENTS

In addition to the general guidelines above, there are specific documentation requirements for source code files and procedures, as shown below.

FILE HEADERS

Every source code file will have a comment header as shown below;

```
; Filename:    hmwk3.s
; Author:      Will B. Engineer
; Description:  Homework Assignment #3
;             Executes variable timing loop to control LED intensity
;             using pulse width modulation based on ADC input
```

SUBROUTINE HEADERS

Each subroutine will have a comment header as shown below. Registers used are listed in alphanumeric order, and describe what information is passed/returned in each register. If a register is changed by the procedure (except the PC), it will be listed in the "Modifies:" section.

```
; Name: RotateStepper
; Description: Rotates stepper motor at the given address the
;             specified number of revolutions in the specified direction
; Assumes:    R0 - direction (CW if R0 > 0, CCW otherwise)
;             R1 - revolution count
;             R2 - stepper driver address
; Returns:    None
; Modifies:   R1
```

DATA/CODE AREA LOCATIONS

In general, the data AREA should be located at the top of a source code file. The code AREA should be located below it. Constants in the code AREA are generally placed at the end of the AREA. If they are placed at the top of the AREA, be sure to include an ALIGN directive after them to ensure that any instruction labels will be correctly word-aligned.