EE371 Assembly Language Coding Standard

The reason for this standard is to insure all embedded firmware meets minimum levels of readability and maintainability. This standard is tailored for EE371 and a more fully defined development standard example can be found in *A Firmware Development Standard*, by The Ganssle Group[1] Your assembly language programs will be graded as part of your lab. To receive full credit your code must conform to the following standards:

1. Assembly language programs will use the following program style:

Header Comments

All programs must have a header with the date, short description of the program and author's names. Header comments include:

```
; Program file name
; Author(s) name(s) (include lab partner)
; Date
; Program purpose
; Revisions
; Other information to help the reader know what the program is supposed to do
; including, perhaps, register and resource use (especially important for subroutines).
```

Global Symbols

Global symbols are those symbols defined in this module (XDEF) or in another module (XREF)
Example:

```
XDEF        Entry          ; Entry point for the program
XREF        sub1, sub2     ; Subroutines used
```

Equates (EQU)

The EQU section defines constants to be used in the program. You should organize this section into three sub-sections

Constant Equates:

Constants used by the program. All constants must have a comment indicating what the constant is for.

Debug12 Monitor Equates (if used):

Definition of the vectors for Debug12 monitor routines.

Memory Map Equates:

Definitions of addresses of various parts of the program and for I/O registers.

Constants are important, particularly for constants used in your program.

[1] http://www.ganssle.com/fsm.htm

Example:

```
; Constant Equates
CR          EQU   $0d           ; Carriage return code
LF          EQU   $0a           ; Line feed code
LED_ENAB    EQU   %00100000     ; Enable bit for LED
; Debug12 Monitor Equates
putchar EQU    $fe04       ; Vector for putchar
; Memory Map Equates
REGS:       EQU   0             ; Registers base address
PORTP:      EQU   REGS+$0056    ; Port P address
DDRP:       EQU   REGS+$0057    ; Data Direction Register
```

Code Section

The program code is located in the code section:

Example:

```
MyCode:     SECTION
```

Program Body

Your program goes here. The first instruction in your program must initialize the stack pointer. All programs must have Adesign@ comments and may have comments on individual instructions.
For programs in the lab, this section should end with the SWI instruction to go back to the monitor.

Constant Data Section

Constant data is to be located immediately following the program code.

Example:

```
; Constant data definitions
MyConstant: SECTION
MSG         DC.B       ›This is a message:
```

Variable Data Section

All variable data storage is located in RAM memory:

Example:

```
; Variable data storage
MyData:     SECTION
```

Variable Data Storage Allocation

Any variable data element must have storage allocated with the Define Storage (DS.B) assembler directive.

Example:

```
    counter         DS.B            1       ; Allocate one byte for a counter
```

2. Comments in programs shall follow the ***Rules and Regulations for Comments in Programs***. See <http://www.coe.montana.edu/ee/courses/ee/ee371/pdffiles/comments.pdf>.

3. All programs are to consist of only SEQUENCEs of logical blocks, IF-THEN-ELSE decisional elements and REPETITION loops like DO-WHILE or WHILE-DO.

4. A **SEQUENCE** block must start with a BEGIN comment and end with an END comment.

```
; BEGIN comments on what the block is to do
        Code for the block
; END
```

(Exception: If the sequential element that the design calls for is implemented with only a few lines of code, the BEGIN and END comments can be eliminated.)

No branches are allowed from outside the sequence block into the block. Branches within the block are allowed. Branches to subroutines are allowed.

5. An **IF-THEN-ELSE** decision block is to be coded

```
; IF (condition to be tested is true)
        Code to test for true
        Branch if condition true THEN_PART_n
; ELSE part comments
        Code to be done if condition is not true
        BRA ENDIF_n
; THEN part comments
THEN_PART_n
        Code to be done if condition is true
ENDIF_n
```

or

```
; IF (condition to be tested is true)
        Code to test for true
        Branch if condition not true ELSE_PART_n
; THEN part comments
        Code to be done if condition is true
        BRA ENDIF_n
; ELSE part comments
        Code to be done if condition is not true
ENDIF_n
```

6. A repetition is to be coded

(**DO-WHILE**)

```
; DO comments on what the block is to do
DO_WHILE_n
        Code for the block
; WHILE (condition to be tested for true)
        Code to test for true
        Branch condition true DO_WHILE_n
; END_DO_WHILE_n
```

or (**WHILE-DO**)

```
; WHILE (condition to be tested is true)
WHILE_DO_n
        Code to test for true
        Branch condition not true END_WHILE_DO_n
; DO comments on what the block is to do
        Code for the block
        BRA WHILE_DO_n
END_WHILE_DO_n
; END_WHILE_DO
```

7. Programming modules are to be used. Modules must have headers which describe the function of the subroutine and all entry and exit requirements for all registers and variable data use. Subroutine style shall follow the ***Rules and Regulations for Modules and Subroutines***. See <http://www.coe.montana.edu/ee/courses/ee/ee371/pdffiles/subrules.pdf>.

8. Blocks of code (the span between a BEGIN and END) should be no greater than 50 lines of code including comment lines.

9. Other miscellaneous rules

    a. No magic numbers
    b. One input/one output of all blocks
    c. Stack pointer must be initialized
    d. Program code, constants, and variable data must be located assuming pseudo-rom

($4000-$5FFF) and RAM ($6000-$7FFF) in the EVB.

e No line more than 80 characters.

f. Place labels on lines by themselves.

G:\1wpdocs\univ\dept\courses\Ee371\asm_coding_standard.doc