PicoBlaze Assembly Language Style File

Definitions: the word "should" indicates the given guideline is *required*; it is not optional.

General:

- Every PicoBlaze assembly language program should contain three parts: 1) a comment banner, 2) assembler directives, 3) the assembly code.

    1) Comment banner: Every program file should contain a comment banner that includes information pertinent to the program. This includes names, dates, purpose, revision history etc.

    2) Assembler Directives: All *NAMESREG* and *CONSTANT* directives should appear in one area which should follow the comment banner. These directives should include comments that explain the intended purpose of each directive. These directives should contain meaningful names as a form of self-commenting.

    3) All assembly language portion of the program should be written in modular form. Each of the subroutines and areas of code that performs a single function should be delineated from other parts of the code and well commented.

- All hexadecimal constants should be in uppercase.
- All assembly mnemonics should be in uppercase.
- All hexadecimal register names should be in uppercase.
- NEVER use tabs in your source file; use multiple single spaces instead.
- Use white space to enhance the readability of your code.
- All assembly code instruction mnemonics should start in the same column.
- Examine hardcopy printouts of your source code to make sure your document looks as good in hardcopy as it did on your computer screen.
- All source code should be in a Courier font to ensure readability.

Labels:

- All labels should be in lower case (while delineates them from the upper cased used for instructions).
- Labels should exhibit self-commenting by having a meaningful name relative to the underlying code. You must be creative because the length of the label is limited to nine characters with the current PicoBlaze assembler.

Comments:

- Comments are used to describe *what* is happening and *how* something is being done. The primary purpose of comments is to further the readability and understandability of the assembly code.
- Each line of assembly code must contain a comment unless the purpose of the statement is absolutely clear. Comments should not reiterate what is clear from the code.
- Lewd or degrading remarks should not appear in comments though they are allowed to be voiced freely in most other places.
- Comments should not contain right-side delimiters.
- Multi-line comments are acceptable. The length of a comment is proportional to the complexity of the code that is being described.

Subroutines:

- Subroutine labels should be self-commenting and give a rough indication as to the function performed by the subroutine.

- Each subroutine should contain a comment banner explaining the intended purpose of the subroutine. This banner should also note the values are being passed to it (the information the subroutine expects to find in each register) and the registers that the subroutine modifies.

Here is a rough example of what your code should look like:

```
;-----------------------------------------------------------------
;- Solution: CPE 269 LabX Winter 2004
;-
;- Pat Wankaholic
;- 2-19-04
;-
;- This program does something really cool. Here's the description.
;-----------------------------------------------------------------

;-----------------------------------------------------------------
;- Port Constants
;-----------------------------------------------------------------
CONSTANT switch_port, 30          ; port for switches ----- INPUT
CONSTANT led_port, 0c             ; port for LED output --- OUTOUT
CONSTANT btn_port, 10             ; port for button input - INPUT
;-----------------------------------------------------------------

;-----------------------------------------------------------------
;- Misc Constants
;-----------------------------------------------------------------
CONSTANT BTN2_MASK, 08            ; mask all but BTN5
CONSTANT B0_MASK, 01              ; mask all but bit0
CONSTANT B1_MASK, 02              ; mask all but bit1
CONSTANT B2_MASK, 04              ; mask all but bit2
;-----------------------------------------------------------------


             ADDRESS 00
      start: ENABLE INTERRUPT             ; no particular reason

 main_loop:  INPUT   s0, btn_port         ; input status of buttons
             INPUT   s1, switch_port      ; input status of switches
             AND     s0, BTN2_MASK        ; clear all but BTN2
             JUMP    NZ, bit_wank         ; jumps when BTN2 is pressed

             ;-----------------------------------------------------------------
             ;- nibble wank portion of code
             ;-----------------------------------------------------------------
wank:        RL      s1                   ; rotate 2 times - msb-->lsb
             RL      s1
             LOAD    s0,s1                ; transfer data register to be read out
             JUMP    fin_out              ; jump unconditionally to led output
             ;-----------------------------------------------------------------

             ;-----------------------------------------------------------------
             ; bit-wank algo: do something Blah, blah, blah ...
             ;-----------------------------------------------------------------
bit_wank:    LOAD    s0,00                ; clear s0 for use as working register

             SL1     s1                   ; shift msb into carry bit
             JUMP    NC, bit1             ; jump if carry not set
             OR      s0, B0_MASK          ; set bit0
bit1:        SL1     s1                   ; shift msb into carry bit
             JUMP    NC, bit2             ; jump if carry not set
             OR      s0, B1_MASK          ; set bit1
bit2:        SL1     s1                   ; shift msb into carry bit
             JUMP    NC, bit3             ; jump if carry not set
             OR      s0, B2_MASK          ; set bit2
             ;-----------------------------------------------------------------

             CALL    my_sub              ; subroutine call
fin_out:     OUTPUT  s0, led_port         ; output data to LEDs
             JUMP    main_loop           ; endless loop
             ;-----------------------------------------------------------------

             ;-----------------------------------------------------------------
             ; my_sub: This routines does something useful. It expects to find
             ; some special data in registers s0, s1, and s2. It changes the
             ; contents of registers blah, blah, blah...
             ;-----------------------------------------------------------------
my_sub:      SL1     s1                   ; shift msb into carry bit
             JUMP    NC, bit3             ; jump if carry not set
             OR      s0, B2_MASK          ; set bit2
             RETURN
             ;-----------------------------------------------------------------
```