

micros assignement

by Karl Plummer

FILE	4115_KARL_PLUMMER_MICROS_ASSIGNEMENT_153432_1987131192.D OCX (186.84K)		
TIME SUBMITTED	31-JAN-2019 03:56PM (UTC+0000)	WORD COUNT	2286
SUBMISSION ID	100073561	CHARACTER COUNT	10857

Micros peripherals & interfaces

Module code: ACDA4004

By Karl plummer

Student number: 1707253

MPLAB (one page)

Introduction

MPLAB is a program used to write code for PIC microcontrollers, with the program it is able to compile and imprint the program onto the PIC using a data transfer cable which the PIC is connected from. Once imprinted the PIC is then able to be unplugged and used with the program installed on it. MPLAB has other features such as being able to test your code before hand and make sure there are no build errors. MPLAB is able to program any PIC microcontrollers that support MPLAB and this includes ones of varying sizes and pins. With the code used within MPLAB you need to read through the PIC microcontrollers datasheet to be able to allocate the address to the correct registers and pins for the PIC as even though a lot of PICs are similar, they will have differences especially when you start increasing in size and therefore pins. The program will also need compiling which MPLAB comes with its own compiler when you check for build errors, as if the program was left uncompiled it would be too large to fit onto the PIC microcontroller. A feature of the MPLAB is the versatility of programming using this method as you can copy and past the code into other programs like proteus.

Summary

In summary after some extensive but basic use of the program MPLAB I have come to the following conclusions about its use and functionality. To start with the build errors and compiler is great for when you need to check if one of the commands that you have entered is incorrect, but the problem faced with this if you have written you code correctly it could still be in the wrong order and not work as a result. Being able to edit code very easily on the PIC microcontroller by as simply as plugging back into the adaptor for the computer removing the old program and compiling and checking build for errors and saving back to the PIC and its finished and ready to be used again. MPLAB uses a basic low level programming language so is relatively easy to use with the main issue being learning the commands and how the registers work, as if you have problems with this the programming takes a long time to understand.

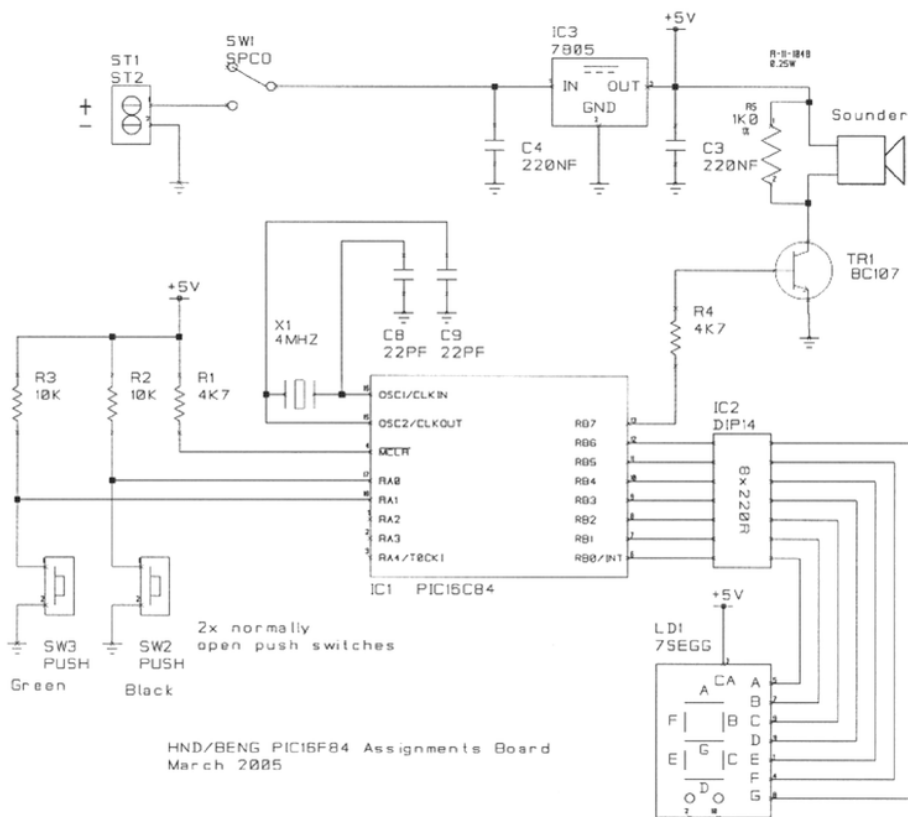
"count" errors explain/how fixed

The first task we were given as a class was Count.asm file to work through with MPLAB and to check that it was fully working in the build. There were some issues with the code we had to address before the code was able to be used, the following are the problems faced with the coding and how I fixed them: -

- ➔ The first of many problems that I encountered was there was a few typo's or spelling errors for commands and this was the first thing I set out to correct including:
 - En d changed to just end
 - 0xffff changed to 0xff
 - Cont change to count
- ➔ The last problem I faced was going through and checking that the code had been written in the correct order and the right commands had been used the only way of

doing this was through trial and error unfortunately as this was the first time using MPLAB

Explanation of circuit diagram/components



Above we have the circuit of which we attached our programmed PIC microcontroller to. Below is a list containing the components and the uses from the circuit diagram:

- 1) Starting with the straight forward components being the sounder, which the main use for this being to create sound on set wave forms from the PIC microcontroller.
- 2) The push switchers or buttons as they are typically called are for the user to input commands to the PIC microcontroller without having to remove the PIC and reprogram it using MPLAB. The buttons can be configured to allow a multitude of different inputs from them
- 3) The LED display is so that you can display single digits or characters using this to spell out different words or count up through numbers (Hexadecimal number are the

easiest for this application). This will need to be pre programmed to display different characters from the PIC microcontroller and other inputs can affect the speed.

- 4) The crystal oscillator is used almost like an internal clock and when a voltage is applied across the crystal (the most common type being Quartz) it will mimic a RLC circuit.
- 5) Basic components in this circuit include resistors and capacitors which take place in almost every circuit as the resistor will limit the flow of current and the capacitor stores charge.
- 6) The last two main points from the circuit diagram being the input voltage (or power in) and the main switch to turn the circuit on and off, these are used for obvious reasons.

Work on modifying code need flow chart for each

sound

```
;; FILE:          count.asm
; AUTHOR:         JMY
; DATE:           24th March 2005
; DESCRIPTION:    Count to 255 at 1 second intervals and display on a seven
;                  segment display in binary.
;                  The pause subroutine takes apx. 0.2 sec.
;                  Pause is called 5 times

list    p=16F84          ; select processor type
radix   hex              ; define radix as hexadecimal
;    #include "p16f84.inc"
;-----
;    cpu equates (memory map)

portA   equ 0x05          ; port A address
trisa   equ 0x05          ; trisa address in register bank 1

portB   equ 0x06          ;port B address
```

```
trisb      equ 0x06      ;trisB address in register bank 1
```

```
Status equ 0x03      ;status register
```

```
RP0      equ 5      ;bit 5 of status register
```

```
RP1      equ 6      ;bit 6 of status register
```

```
black equ 0      ; black botton connects to bit o port A
```

```
green equ 1      ; green botton connects to bit 1 port A
```

```
nCountequ 0x0d ; pause routine inner loop counter
```

```
mCount      equ 0x0e ; pause routine outer loop counter
```

```
;-----
```

```
org 0x000
```

```
start
```

```
bcf      Status,RP0 ; clear bank selection bits in the status reg
```

```
bcf      Status,RP1
```

```
bsf      Status,RP0 ; select bank 1
```

```
movlw 0x00 ; load w with 0x00
```

```
movwf trisb ; load it into trisb to make portb all outputs
```

```
movlw 0x03 ;
```

```
movwf trisa ; Setting RA0 and RA1 as input
```

```
bcf      Status,RP0 ; reselect bank 0
```

```
    clrf        portB        ; clear all lines low
```

```
Loop
```

```
    btfss    portA,black    ; in black botton pressed?
```

```
    call     sounder
```

```
    goto     Loop        ; repeat forever
```

```
sounder
```

```
    bsf      portB,7        ; turn sounder on
```

```
    call     pause
```

```
    bcf      portB,7        ; turn sounder off
```

```
    call     pause
```

```
    return
```

```
;-----
```

```
;subroutine pause: lets waste some time....
```

```
;
```

```
;inputs: nothing
```

```
;outputs: nothing
```

```
;
```

```
;does: nothing but waste some time
```

```
;corrupts: register W
```

```
;
```

```
;description: two loops using "mCount" and "nCount" register files
```

```
;
```

```
                "mCount" is the outer loop counter (255 down to zero)
```

```
;
```

```
                "nCount" is the inner loop counter (255 down to zero)
```

```
;
```

;-----

pause

movlw 0x13 ; set w = 255 decimal

movwf mCount ; mCount = w: outer loop counter

loadN

movlw 0x13 ; set w = 255 decimal

movwf nCount ; nCount = w: inner loop counter

decN

decfsz nCount, f ; nCount--

goto decN ; if nCount != 0 then repeat nCount--

decfsz mCount, f ; else decrement Count

goto loadN ; if mCount != 0 then

; reload nCount to 255 and decrement

return ; else exit subroutine

delay1s

call pause

call pause

call pause

call pause

call pause

return

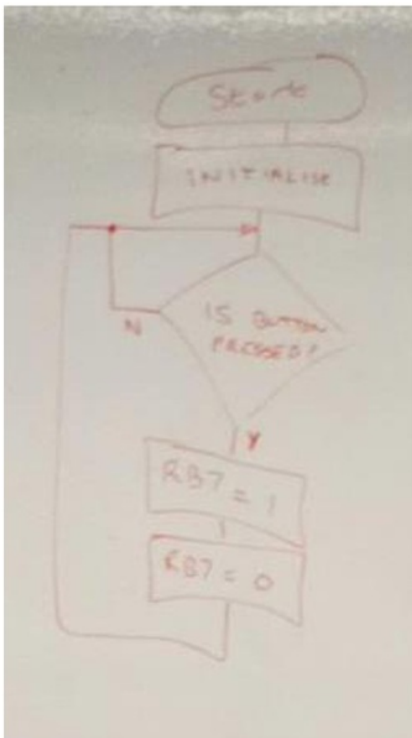
end

;-----

```

; Configuration! Important!
;
; at blast time, select:
;     memory uprotected
;     watchdog timer disabled
;     standard crystal (4 MHz)
;     power-up timer on

```



4

Count up with green down with black

```

;; FILE:      count.asm
; AUTHOR:     JMY
; DATE:       24th March 2005
; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
;              segment display in binary.

```

```
;                                The pause subroutine takes apx. 0.2 sec.  
;  
;                                Pause is called 5 times
```

```
list    p=16F84                ; select processor type  
radix    hex                    ; define radix as hexadecimal  
;    #include "p16f84.inc"  
;-----  
;    cpu equates (memory map)  
indr    equ 0x00  
fsr      equ 0x04                ;file select register  
  
portA    equ 0x05                ; port A address  
trisa     equ 0x05                ; trisA address in register bank 1  
  
portB    equ 0x06                ;port B address  
trisb     equ 0x06                ;trisB address in register bank 1  
  
Status    equ 0x03                ;status register  
RP0       equ 5                  ;bit 5 of status register  
RP1       equ 6                  ;bit 6 of status register  
  
black     equ 0                  ; black botton connects to bit o port A  
green     equ 1                  ; green botton connects to bit 1 port A  
  
loc        equ 0x0c                ; pointer to lookup table  
  
nCountequ 0x0d                ; pause routine inner loop counter  
mCount     equ 0x0e                ; pause routine outer loop counter
```

```
Count equ 0x0f ; to hold currently displayed numbers
```

```
LUtable equ 0x10 ;Start of lookup table for characters for LED display
```

```
;-----
```

```
org 0x000
```

```
start
```

```
bcf Status,RP0 ; clear bank selection bits in the status reg
```

```
bcf Status,RP1
```

```
bsf Status,RP0 ; select bank 1
```

```
movlw 0x00 ; load w with 0x00
```

```
movwf trisb ; load it into trisb to make portb all outputs
```

```
movlw 0x03 ;
```

```
movwf trisa ; Setting RA0 and RA1 as input
```

```
bcf Status,RP0 ; reselect bank 0
```

```
clrf portB ; clear all lines low
```

```
clrf loc ; reset pointer to zero
```

```
movlw 0x40 ; a'0' on the LED display
```

```
movwf LUtable ; put it in lookup table
```

```
movlw LUtable ; put base address of look up table in W
```

```

        addwf loc,f          ;add base address to pointer and store in loc
        incf      loc,f      ; increment pointer address to 2nd location in
LU table
        movf  loc,w

        movwf fsr           ; increment pointer address to 2nd loction in LU table
        movlw 0x79          ; a '1' on the LED display
        movwf indr

        incf      fsr,f      ; increment pointer address to 3rd loction in LU
table
        movlw 0x24          ; a '2' on the LED display
        movwf indr

        incf      fsr,f      ; increment pointer address to 4th loction in LU
table
        movlw 0x30          ; a '3' on the LED display
        movwf indr

        incf      fsr,f      ; increment pointer address to 5th loction in LU
table
        movlw 0x19          ; a '4' on the LED display
        movwf indr

        incf      fsr,f      ; increment pointer address to 6th loction in LU
table
        movlw 0x12          ; a '5' on the LED display
        movwf indr

        incf      fsr,f      ; increment pointer address to 7th loction in LU
table

```

	movlw 0x02	; a '6' on the LED display
	movwf indr	
table	incf fsr,f	; increment pointer address to 8th loction in LU
	movlw 0x78	; a '7' on the LED display
	movwf indr	
table	incf fsr,f	; increment pointer address to 9th loction in LU
	movlw 0x00	; a '8' on the LED display
	movwf indr	
LU table	incf fsr,f	; increment pointer address to 10th loction in
	movlw 0x18	; a '9' on the LED display
	movwf indr	
LU table	incf fsr,f	; increment pointer address to 11th loction in
	movlw 0x08	; a 'A' on the LED display
	movwf indr	
LU table	incf fsr,f	; increment pointer address to 12th loction in
	movlw 0x03	; a 'b' on the LED display
	movwf indr	
LU table	incf fsr,f	; increment pointer address to 13th loction in
	movlw 0x46	; a 'c' on the LED display

```
movwf indr
```

```
incf      fsr,f      ; increment pointer address to 14th loction in  
LU table
```

```
movlw 0x21      ; a 'd' on the LED display
```

```
movwf indr
```

```
incf      fsr,f      ; increment pointer address to 15th loction in  
LU table
```

```
movlw 0x06      ; a 'E' on the LED display
```

```
movwf indr
```

```
incf      fsr,f      ; increment pointer address to 16th loction in  
LU table
```

```
movlw 0x0e      ; a 'F' on the LED display
```

```
movwf indr
```

```
clrf      Count      ; set count = 0
```

```
clrf      loc      ; reset point to zero
```

```
movlw LUtabel      ; put base address of look up table in W
```

```
addwf loc,f      ; add base address to pointer and store in loc
```

```
write_hex
```

```
btfss portA,black  ; in black botton pressed?
```

```
incf      Count,f      ; count =count + 1
```

```
btfss portA,green  ; in green botton pressed?
```

```

    decf    Count,f          ; Count=Count-1

    btfsc   Count,4          ; if count = 16
    clrf    Count           ; count = 0

    movf    Count,w          ; put count to W reg
    addwf   loc,w            ; w = loc + w
    movwf   fsr              ; set file select register to w reg (count + LUtable base
addr)

    movf    indr,w           ; Read value from location in lookup table
    movwf   portB           ; put value from look up table on port B


    call     delay1s
    goto     write_hex       ; repeat forever


;-----
;subroutine pause: lets waste some time....
;
;inputs: nothing
;outputs: nothing
;
;does: nothing but waste some time
;corrupts: register W
;
;description: two loops using "mCount" and "nCount" register files
;              "mCount" is the outer loop counter (255 down to zero)

```

```

;               "nCount" is the inner loop counter (255 down to zero)
;
;-----
pause
    movlw 0xff      ; set w = 255 decimal
    movwf mCount    ; mCount = w: outer loop counter
loadN
    movlw 0xff      ; set w = 255 decimal
    movwf nCount    ; nCount = w: inner loop counter
decN
    decfsz nCount, f ; nCount--
    goto   decN      ; if nCount != 0 then repeat nCount--

    decfsz mCount, f ; else decrement Count
    goto   loadN     ; if mCount != 0 then

                        ; reload nCount to 255 and decrement
    return          ; else exit subroutine

delay1s

    call   pause
    call   pause
    call   pause
    call   pause
    call   pause

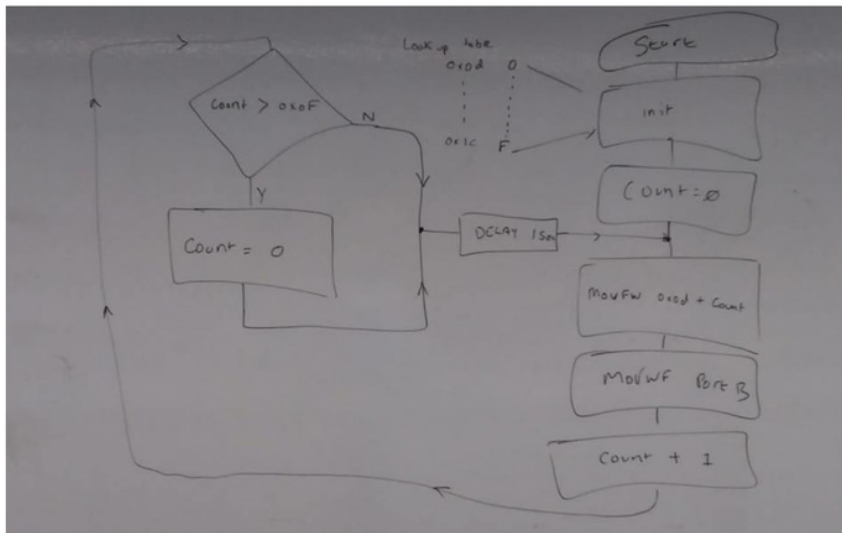
return

```

```

end
;-----
; Configuration! Important!
;
; at blast time, select:
;     memory uprotected
;     watchdog timer disabled
;     standard crystal (4 MHz)
;     power-up timer on

```



References

CRYSTAL OSCILLATOR

[1]"Crystal oscillator", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Crystal_oscillator. [Accessed: 31- Jan- 2019].



MPLAB IDE overview

[2]"Microchip: MPLAB Integrated Development Environment (IDE) Overveiw", 2015.[online].available <https://www.element14.com/community/docs/DOC-39318/1/microchip-mplab-integrated-development-environment-ide-overview> .[accessed 28-jan-2019].

micros assignement

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

42/100

PAGE 1

PAGE 2



Comment 1

connected to

PAGE 3

PAGE 4



Comment 2

In what way does it mimic an RLC circuit?



Comment 3

I think this may have been accidentally left in

PAGE 5

PAGE 6

PAGE 7

PAGE 8



Comment 4

I was expecting you to reproduce the flowcharts and to discuss each section of the floawchart and how they relate to your code, not provide me with a photo of a flowchart we discussed in class.

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16



Comment 5

You have provided me with two reference sources but you have not indicated where in your text this information is used or discussed. You need to cite these locations in your text with a corresponding number [1].

GRADING FORM: SMU GENERIC_SMOSEY

KARL PLUMMER

41.80

ENGAGEMENT WITH LITERATURE

including reading, referencing, academic conventions and academic honesty



You have provided me with two reference sources but you have not indicated where in your text this information is used or discussed. You need to cite these locations in your text with a corresponding number [1].

8

KNOWLEDGE & UNDERSTANDING

Knowledge of the basic underlying concepts and principles of a subject.



Brief description of MPLAB with broad statements about functionality. While you have provided solutions to two of the problems, but you have not provided descriptions of how they operate so I cannot judge your understanding of the solutions.

8

COGNITIVE & INTELLECTUAL

Evaluate underlying concepts and principles of a subject and interpret qualitative and quantitative data in order to develop lines of argument and make sound judgements.



Noit in mark scheme

PRACTICAL APPLICATION SKILLS

Different approaches to solving problems in particular contexts.



While you have provided solutions to two of the problems, but you have not provided descriptions of how they operate so I cannot judge your understanding of your solutions.

16.8

TRANSFERABLE SKILLS

Communicate the results of their study/work accurately and reliably, and with structured and coherent arguments; the qualities needed for employment requiring the exercise of some personal responsibility

within a structured and managed environment.



Can communicate in a suitable format but with some room for improvement. Work lacks coherence in places and could be better structured.

9