```
 title  "Simple 16F88 LED Blinker"
; by Brooke Clarke, N6GCE
; Working 5 June 2006
; http://www.PRC68.com

;=================== How to Read the .asm Source File ===============
; This file Brooke-16F88-LED.asm was prepared using the MPLAB Editor.
; Since it's the assembler source file it has the .asm file type.
; any text after the semicolon ";" is a comment and will show up green.
; Assembler directives and PIC commands are blue.
; Labels are Violet.
; There are also colors for the different number formats.

; My comments to the listing are inside the equal signs "="
; the other comments are part of the assembly file
;
; Line labels must be left justified.
; commands need to be spaced or tabbed in from the left margin.
; If a command is on the left margin it will be colored violet
; and interperted as a label.  Once a space or tab is inserted the
; color will change to blue.
; The assembler is smart and can catch some label or command confusion
; and generate a warning.
;
; There is no spelling checker so the comments and commands might be wrong.
; The assembler will complain about misspelled commands, but does not
; look at the comments.
;
; If you're reading this in the MPLAB editor on the left of the page
; there will be a gray gutter with assembly line numbers.  These line
; numbers are printed in the assembly listing just to the left of the
; memory address.  The line numbers at the very left of a listing
; are the listing line numbers and are NOT the source code line numbers.
;====================================================================

; ====================== Using the pdf Version=======================
; An advantage of the pdf version is that it has the colors, that's
; not the case if Word is used.  Neither Word nor Acrobat captures the
; assembly line numbers that you see on the left in the MPLAB Editor.
;
; The pdf file can be used to cut and paste text.  To do this use the
; Adobe "text tool" to select the text and <CTRL><C> to copy it to
; the clipboard.  Then position the cursor in the MPLAB editor window
; where you want the clipboard contents and press <CTRL><p>.
;====================================================================

; ----- Hardware Notes -----
; /MCLR  pin 4 thru 10 k resistor to +5 Volts
; Vss pin 5 to ground
; Vdd pin 14 to +5
; LEDs can be connected to one or more of the 15 the unused pins.
; Each LED should be connected to ground using a
; resistor (say 220 to 470 Ohms).  LED flat towards ground.
;
; A large electrolytic cap 10 to 50 uF at >= 16 V across +5 to gnd.
; A 0.1 or 0.01 uF ceramic across +5 to gnd.
; A 1N400x diode across +5 to gnd with the cathode (band) to +5.
; This diode does nothing normally, but if you hook up the power
; supply backwards the diode will protect the PIC.
;
; Optional, but handy a resistor (say 220 to 470 Ohms) between
; +5 and LED.  LED flat to gnd.  This is the Power ON indicator.
;
; Approximate outputs
; A0 pin 17 = 5363 Hz = 190 us
; A1 pin 18 = 2632 Hz = 380 us
; A2 pin  1 = 1316 Hz = 760 us
; A3 pin  2 =  658 Hz = 1.52 ms
; A4 pin  3 =  333 Hz = 3 ms
; A5 pin  4 = /MCLR no output
; A6 pin 15 =  83 Hz = 12 ms
; A7 pin 16    41 Hz = 24 ms
```

```
;
; B0 pin  6 =   20 Hz = 48 ms
; B1 pin  7 =   10 Hz = 100 ms
; B2 pin  8 =    5 Hz = 200 ms
; B3 pin  9 =    2.5 Hz = 400 ms
; B4 pin 10 =    1.2 Hz = 800 ms
; B5 pin 11 =    0.6 Hz = 1.6 sec
; B6 pin 12 =    0.3 Hz = 3.2 sec
; B7 pin 13 =    0.1 Hz = 6.4 sec


;======================== Building a Project in MPLAB ===============
; (0) Help \ Topics \ MPASM Assembler has much more detail than this
; summary.
;
; (1) download and install the latest version of MPLAB.  Note that sometimes
; Microchip releases a version that's in beta test.  If two versions are
; offered and one is the beta do NOT use the beta version unless  you
; are having proplems you that might be fixed in the beta.
;
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW0
07002
;
; (2) Using Windows file manager "Explorer" (not Internet Explorer)
; create a directory called IDE directly under C: like C:/IDE.
; in the IDE directory place a sub directory with a short name for each project.
; This is done because there's a MPLAB limitation on the total path length
; of some file names.
;
;(3) Configure \ Select Device and using the drop down box select
; the device family.  For the 16F88 it's the mid range family.
; Then using the device drop down list of PIC model numbers, select
; the correct PIC.  If you don't see the PIC model number you're
; looking for be sure the family is set properly since family changes
; list of PICs.  Once this is done the Select Device window shows you
; which programmers, languages and debuggers can be used with that PIC.
; Note that the PIC Kit 1 is supported in MPLAB.
;
; Also note that this should be done first when starting something new.
; many of the other MPLAM modules use the PIC model number to know how
; to handle the meaning of the code.
;
; A number of years ago someone offered a "Cheap PIC" programmer on the
; internet. These have reproduced faster than rabbits and now we have
; the great-great-great ...- great grandson of Cheap PIC programmer.
; I suppose that if you want to save every penny possible you could use
; one of these but it's not anywhere as fast and easy as using a
; programmer that's integrated into the Integraged Development Enviornemnt (IDE).
;
;(4) I like using the source file for setting the configuration
; byte, but you can also use Configure \ Configuratin bits to do this.
; It does not hurt to use both, the the assembler directive rules.
;
;(5) Place the .asm file if you have one, into the project folder.
; If you don't have one it may be easier to use the .asm file from a
; prior project than it is to start from scratch.
;
;(6) Click on Project \ New.  A dialog box will open enter text
; describing the project in the top line and use the browse button
; to point to the project folder under IDE, like C:/IDE/Proj1.
;
;(6a) If you have a .asm file then in the Project window right click
; on "Source File" and select "add file" then be careful that the
; displayed folder is the correct project (MPLAB has a sticky problem
; and tends to point to the prior project folder here making for mistakes)
; double click the correct .asm file.  It should appear under "source
; files".  Then double click on the proj1.asm file to open the editor window.
;
;(6b)If you don't have a source file then click on File \New and the
; editor will open a window.  This file needs to be saved into the project
; folder with a .asm file type.
;
```

```
;(7) When you are ready to try click on "Build All".  This will start
; the assembly process and will generate a bunch of files in the
; project folder.  It also may open the Output folder with the "Build"
; tab active.  It's here that you fine out if there are Warning or
; Error messages.  This is a very valuable file for debugging and for
; thay use it neeeds to be read very carfully.
;
;(8) If the build is sucessful then you might want to "Save Project",
; but if there are errors you may not want to save the project.
;
;(9) If you are using one of the Microchip programming tools then you
; can get it going by selecting: Programmer \ Select Programmer \ .
; Then connecting or enabling the programmer. Once this is done it
; remains with the project.  So it's very easy once you have a good
; build to click on "earse" the "program" to get the code into the
; PIC. Pay attention to the lower bar of MPLAB during the programming
; operation and make sure the green progress dots are moving and the
; address is incrementing.  When done the last line in the Output
; window should be something about a sucessfull build.  Be sure to
; check for this becasue if there was a problem it's easy to fix here.
; Some problems are the Picstart Plus has the power plug pull out or
; you forgot to close the ZIF socker, or you put the PIC in the wrong
; ZIF socket holes, or you put the PIC in backwards, or you put a TTL
; chip in the ZIF socket instead of the PIC.  MPLAB is not
; very smart here so check for the above if you are having problems.
;
;(10) If using ICD 2 thre are two ways of using it.
;(A) for programming only.  In this case you open IDC2 as the programmer
; using Programmer \ Select Programmer then connect.  After a good
; build use Programmer erase part then programmer program.
; (B) If using ICD 2 for debug and programming leave Programmer with
; no selection and instead activate Debugger \ Select Tool \ ICD2.
; A window may pop up saying "Invalid Target Device" meaning that it
; does not see the model number PIC that's been selected in
; Configure \ Select Device.  Note that when working with PICs that
; directly support ICD2 (like the 16F88) you can Debugger\Clear Memory\
; All Memory and also Debugger\Program the part.  Remember that
; programming a part from the Debugger window adds some code for the
; debugging process and when you are done debugging to shut down the
; Debugger and open the Programmer to load a clean version of the
; code into the PIC.
;=====================================================================
;
;================================== LIST ============================
; The "LIST" below is what's called an assembler directive.  It tells the assembler what to do
; but does not generate any code.  For help click on Help\Topics\MPASM Assembler then click OK.
;  In the Help window on the left click on Search and type in LIST <enter>.
; You can move the divider left or right to allow seeing all of the topics under "List".
; Click on "Directives by Alphabetical Listing" for a list of the directives and then on the right
; click on "list - Listing Options".  The p=<processor> directive is used below as well as the
; n=<# of lines on printed page> and c=<number of columns on printed page.
;=====================================================================

  LIST P=16F88, n=.61, c=.78
;Wordpad margins T.5, L.75, R.5, B.5 <- a reminder for me

;============================== ERRORLEVEL ===========================

; The errorlevel directive allows controlling what warnings, errors, etc. are shown by the assembler.
; by far the most common warning is 302:
; " Register in operand not in bank 0.  Ensure that bank bits are correct."
; I have commented  out this directive below so that when you run "Build All" you will see warnings
for 302 and 305.
;  Then you can remove the ";" and rerun "Build All" and see a cleaner listing.
; But don't just invoke the errorlevel directive just to get rid of the warnings unless you know for
sure
; that you have properly set the bank select STATUS bits.
;=====================================================================

  ;errorlevel 0,-302, -305

;============================== #INCLUDE =============================
```

```
; The #INCLUDE directive is highly recomended.  It defines all the commonly used bytes and bits for
the
; PIC being used which both saves a lot of work, but also uses standardized names making the assembly
; listing much easier to read.  Not all of the include file names match the PIC model number.
; Some have an "x" to allow the file to work for a family of PICs.  To find the include file for your
; PIC look in: C \ Program Files \ Microchip \ MPASM Suite.
; Be very careful that you write down the exact file name.
;=====================================================================

  #INCLUDE "p16F88.inc"

 ;============================ __CONGIG================================
 ; Note this is Underline Underline then CONFIG, you must use the 2 underlines
 ; This directive controls the configuration bits that get set at code burn time
 ; and these can not be changed later by the PIC.

 ; It's possible to set them using MPLAB Configure \ Configuration Bits, but this opens
 ; the door for a mistake where the code is written for one configuration and the
 ; PIC gets burned with a different configuration.
 ; I try to always set them in the code as below.
 ;
 ; The 16F88 uses a special format of the __CONFIG directive that's used where there are two or more
 ; configuration words.  The 16F88 manual forgot to mention this but you can find it
 ; in the Help \ MPASM | search "__CONFIG" and look at the 18F examples.
 ; Note that unlike the very common mid range config that looks like __CONFIG _param1 & _param2 &
....
 ; This starts off __CONFIG _CONFIG1, _param1 & _param2 & ....
 ;                                  ^
 ;                                  |
 ; Note calling out Underscore then CONFIG1 and the comma!
 ; The separator between each parameter is the "&" (AND) operator.
 ; Each of these parameters translates into a word (16 bits) where all the not used bits are 1.
 ; This allows ANDing them together to get the proper configuration word.
 ;
 ; Once you have used "Build All" then a listing file will be in the project folder.
 ; To add it to the project click on either File \ Open or the "open File" Icon.
 ; Next at the bottom of the Open window select Files of type "List Files (.lst).
 ; Typically there will only be one file, just double click it.
 ; Now scroll down to almost the bottom of the listing file and you will see all the _parameter
 ; names for whatever PIC you are using.  Then you can cut and paste them into the __CONFIG
directive.

 ; The data sheet for each PIC goes into detail on what each of these does and what options you have.
 ; For this you must Read The F-----g Manual (RTFM).
 ; My comments below were prompted because I was using the Picstart Plus which allows _MCLR_OFF.
 ; But the ICD 2 requires _MCLR_ON.
 ;=====================================================================


 ; -------------Note _CONFIG1 is at 2007 and can only be set at burn time--------------
 ; for use with ICD2 MCLR needs a 10k pull up resistor and MCLR_ON in the config word.
 ; Once everything has been tested (except any RA5 functions) the internal MCLR_OFF configuration
 ; can be used.  Note that RA5 pin can only be used for an input so the 10k resistor can be left
 ; there and a switch added. maybe using the RC filter debounce ckt.
  __CONFIG  _CONFIG1, _CPD_OFF & _CP_OFF & _DEBUG_OFF & _LVP_OFF & _MCLR_ON & _PWRTE_ON & _WDT_OFF &
_WRT_PROTECT_OFF & _INTRC_IO & _BODEN_OFF
;

;========================== CBLOCK ===============================================
; Common Blocks can be used a number of ways.  The simplest is as shown below where the "0x20" is the
; address of the first available user RAM for the PIC you are using.
;You need to RTFM to find out what that address is!  In the case of the 16F88 it's 0x20 (i.e. hex
20).
; Each file name gets assigned to the next available RAM address.  So in this case Bshadow is
assigned to 0x22.
; you can see these assignments by looking at the bottom of the listing in alphabetical order.
;=====================================================================

;  Register Usage
 CBLOCK 0x20  ;  Start Registers at End of the Values
d1          ;  Delay counters
Ashadow
```

```
Bshadow
 ENDC

; ============================== ORG ================================

; All the above have been assembler directives that do not generate any code. Now we're getting to
where
; code will be generated.

; The origin command tells the assembler what line number to use.  There must be at least one of
these
; at the beginning of every program.  There can be more than one, like below.
;
; At power up most (maybe all) PICs start executing whatever code is at address 0.
; Some micro processors use other memory addresses to find the cold start address.
; Most PICs that have interrupt capability use address 4 as the start of the Interrupt Service
Routine (ISR).
; For this simple program in either case the program will start at the "Start" label.
;=====================================================================
;
;=======================PAGE==========================================
; This program is maybe 1 page long if all these extra comments were
; removed, but the "page" directive is so important that I want to
; mention it here.  Maybe 40 years ago I used the equivalent directive
; so that when a program was printed out each subroutine would be on
; a seperate piece of paper.  Many many years later Microsoft has this
; concept in either "Code Complete" or "How to write Solid Code". I can't
; over emphasize how important this is.  This also means that the length
; of any subroutine can not be longer than what can be printed on one
; page.  If it's longer then you need to figure out a logical way
; to break it up.  This also goes for the main program, it too needs
; to fit on one piece of paper.  Again if it does not fit, then it needs
; to be broken up into logical parts, not use a page1 page2 thing.
; The reasoning behind this gets into the span of control and other
; stuff that's not part of this getting started paper.
;
; In a program that takes many pages to print I spend some time on getting
; the page directives as outlined above and then working to refine the
; routines so they fit pages properly.
;
; When taking a photo most people look to see what's in the photo.  A
; more expreiences photographer then looks to see what should not be
; in the photo.  By controlling the page breaks you are controlling
; what you see and what you do not see and this is a very big aid
; when you are debugging code.
;
; Although there are those at Microchip that understand the importance
; of the page directive the others who are short sighted have allowed
; a bug to remain in MPLAB since about version 4.0.  The bug is that
; when  you print from MPLAB the new page commands put into the files are
; ignored.  In order to get the proper new page fomratting you need to
; use Word Pad or Word to open and print the listing file.
;=====================================================================

  PAGE
;================= Start of Code =================
  org    0
  goto Start
  org 4
Start

;======================= Housekeeping ================================
; The first thing that needs to be done is housekeeping.
; This is where the function of each PIN is defined and where the initial values are set.
; I happen to know that STATUS, PORTA and PORTB are all in memory bank 0 for the 16F88,
; but you need to RTFM section on "Memory Organization" to be sure in which bank which register is
located.
; This is the subject relating to the 302 warrnings. Note that there's a neat assembler directive
"banksel".
; It can be used after the PIC model number is defined above and sets the STATUS register bits RP0,
RP1, etc.
; for you.  I think if you're using banksel in front of every register and port operation then all
```

```
the 302 type
; problems should go away.
;=====================================================================
;
;  Enable all of A & B for Output
  clrf  STATUS ; this also sets bank 0
  clrf  PORTA
  clrf  PORTB
;======================== Input & Output memory Aid ==================
; 1 (one) for Input - note that 1 and i look very much alike.
; 0 (zero) for Output - note that 0 and O look very much alike.
; But be sure to use numbers for setting the port pin directions.
;=====================================================================

;========= TRIS command vs. TRISA, TRISB, etc register names ======
; First generation PICs used a command TRIS (tristate) that assigned
; the direction of the port pins.  More modern PICs have registers
; called TRISA, TRISB, etc. and the preferred method is to write to
; those registers to control port direction.
;=====================================================================
  movlw  0x000
  banksel TRISA
  movwf  TRISA
  banksel TRISB
  movwf  TRISB
;====================== Ports with possible A/D inputs ===============
; at this point a newbie would think that they had setup port A for
; all outputs, but when the program is run there will be NO outputs
; on port A.  When a port contains the possibility of Analog to Digital
; converters those pins are defaulted to analog inputs.  This is the safe
; thing to do to prevent damage.  In the 16F88 port A has the analog inputs
; and so the Analog Select register needs to be programmed to make whatever
; port A bits we want to be Digital I/O pins.  In this case we want all the
; pins on port A to be digital so using the memory aid above (Oh or zero for
; Output) we just clear the ANSEL register.
;=====================================================================
  banksel ANSEL
  clrf ANSEL
;
;========================= OSCCON ====================================
; One of the features of the 16F88 that's not on the 16F84 is the
; internal oscillator.  No pins need be wasted for Fosc.

  banksel OSCCON
; --------The Internal RC Frequency can be set at any time after ORG ----------------------
; Now set 4 MHz clock
  bsf OSCCON, IRCF2 ; 110 is 4 MHz
  bsf OSCCON, IRCF1
  bcf OSCCON, IRCF0
;
; ============ the 16F88 has a clock fail safe system that's not used here =
  bcf OSCCON,SCS0 ; 00 is Oscillator mode defined by FOSC <2:0>
  bcf OSCCON,SCS1 ;

  nop ; allow some time for oscillator to settle
  nop
; ============================ OSCCAL ================================
; many of the PIC chips come with an oscillator calibration value that's
; specific for each chip and by reading that value and storing it into
; the OSCCAL register the internal clock is more accurate than not using
; it.  But the 16F88 does not use that method and instead had OSCTUNE to
; tweak the internal oscillator frequency.  For most apps you can just
; forget about this.  It's here leftover from another project.
;=====================================================================
  banksel OSCTUNE
  bsf OSCTUNE,TUN5
  bcf OSCTUNE,TUN4
  bsf OSCTUNE,TUN3
  bsf OSCTUNE,TUN2
  banksel PORTA
;====================== Main Loop ====================================
; there are a number of ways a program can be constructed.
```

```
; This program has the Housekeeping above then the Loop below.
; below the loop are the subroutines.
; There are some other possible constructs.  For example:
; * instead of having the loop enclose code the program might end in
; a loop pointing to itself.
; * there may not be any housekeeping just the loop.
; * there may be interrupts added to any of the above.
; etc. etc.
;-------------------------------------

;================ Commands ========================================
; One of the selling points of the PICs is that they only have
; 30 something commands, not the huge number of commands that a
; Complex Instruction Set Computer (CISC) would have.  This is called
; a Reduced Instruction Set Computer (RISC).  RISC computers tend to
; run faster than CISC computers.  The RISC instruction set is much
; easier to keep in mind (one side of one sheet of paper) than the
; book needed for a CISC instruction set computer.
; RTFM section "Instruction set Summary" to learn what the commands do.
;==================================================================
Loop
  call  Delay80us
  incfsz  Ashadow
  goto  UpdateA
  goto UpdateAB

UpdateA
  movf Ashadow,W
  movwf PORTA
  goto Loop

UpdateAB
  movf Ashadow,W
  movwf PORTA
  INCF Bshadow
  movf Bshadow,W
  movwf PORTB
  goto Loop

;================ End of the Main Loop ===========
; below here are subroutines. They have a label
; and end with "return" or "retlw".
;================================================
;
;----------------------------------
; On line PIC delay code generator
; http://www.piclist.com/techref/piclist/codegen/delay.htm
; variable d1 moved up into top cblock

Delay80us
      ;76 cycles
  movlw 0x19
  movwf d1
Delay80us_0
  decfsz  d1, f
  goto  Delay80us_0

      ;4 cycles (including call)
  return
;----------------------------------------
;=================== END==========================================
; the last assembler directive which must be at the end of the program
; is the END statement.
;==================================================================

  end
```