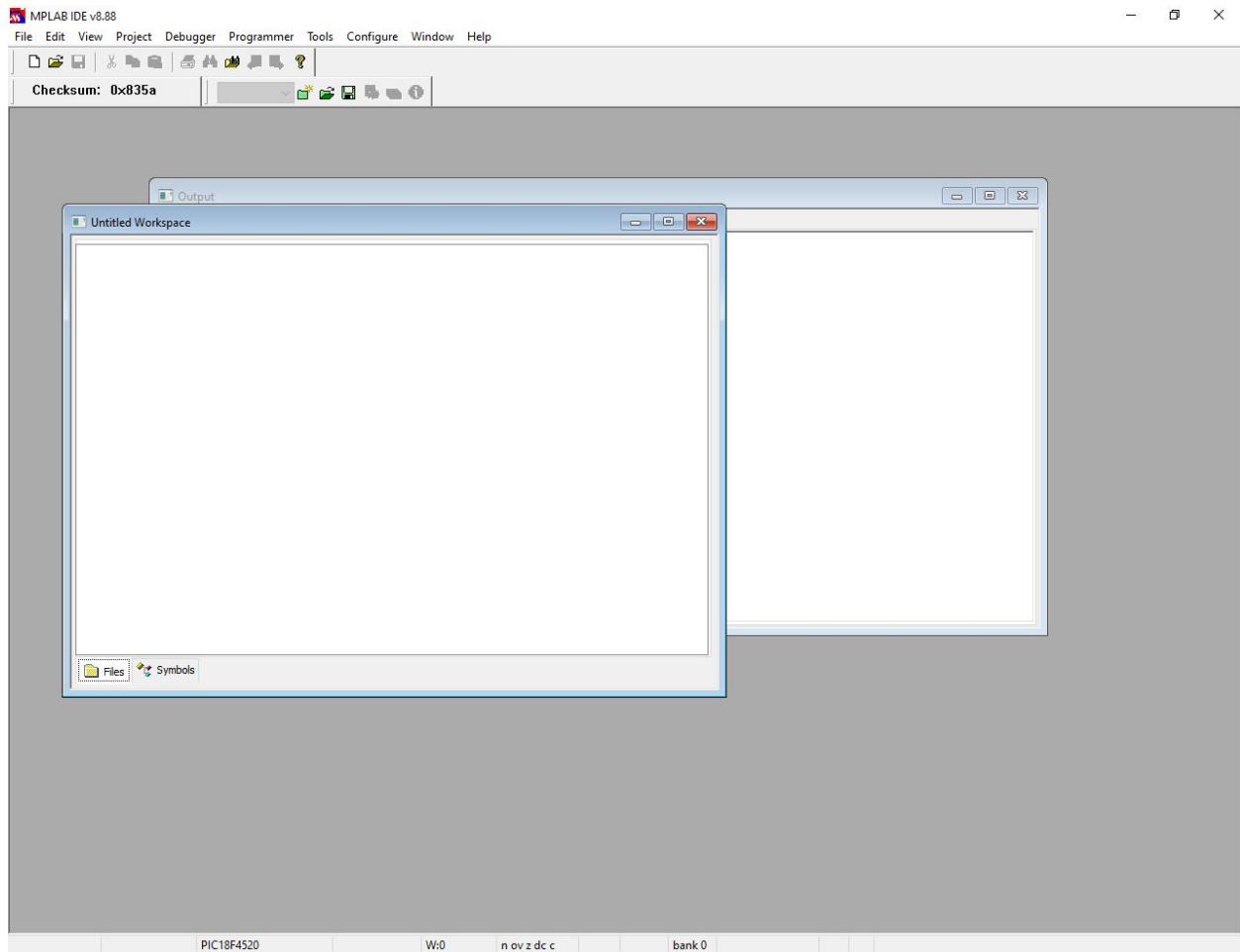


THE CITY COLLEGE OF NEW YORK
Department of Electrical Engineering
EE425 Computer Engineering Laboratory

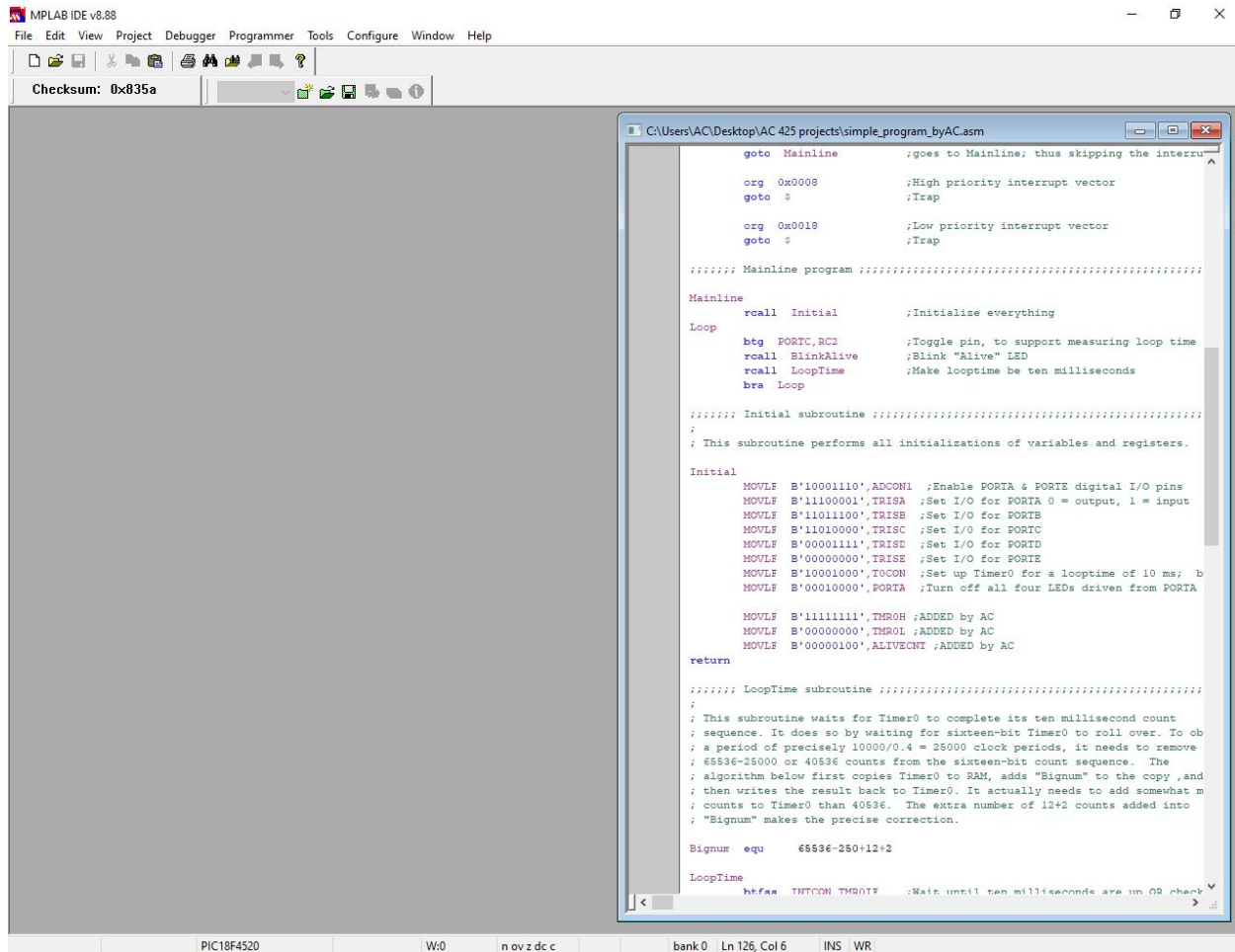
MPLAB IDE Simulator Tutorial for Beginners
by AC

1. Install the **MPLAB IDE** v8.88 (or newer) software, which can be downloaded here:
<https://pic-microcontroller.com/mplab-ide-v8-92-free-download/>
Upon launching the IDE, its interface should look something like this:



IMPORTANT NOTE: Before proceeding, click on **Configure >> Select Device... >> PIC18F4520**, and enter all the parameters of the PIC18F4520 IC we are using. For reference, please see the document you were given on the first day of class whose title is “Programming the PIC Microcontrollers.”

2. Open the .asm file titled “simple_program_byAC.asm” (which was emailed to you) by clicking on **File >> Open...** and then browsing to the directory containing the file. Your interface should look something like this:



The screenshot shows the MPLAB IDE v8.88 interface. The main window displays the assembly file 'simple_program_byAC.asm' located at 'CAUsers\AC\Desktop\AC 425 projects\simple_program_byAC.asm'. The code is as follows:

```
goto Mainline ;goes to Mainline; thus skipping the interrupt vector
org 0x0008 ;High priority interrupt vector
goto $ ;Trap

org 0x0018 ;Low priority interrupt vector
goto $ ;Trap

; Mainline program ;
Mainline
    rcall Initial ;Initialize everything
Loop
    btf PORTC,RC2 ;Toggle pin, to support measuring loop time
    rcall BlinkAlive ;Blink "Alive" LED
    rcall LoopTime ;Make loop time be ten milliseconds
    bra Loop

; Initial subroutine ;
; This subroutine performs all initializations of variables and registers.
Initial
    MOVWF B'10001110',ADCON1 ;Enable PORTA & PORTE digital I/O pins
    MOVWF B'11100001',TRISA ;Set I/O for PORTA 0 = output, 1 = input
    MOVWF B'11011100',TRISE ;Set I/O for PORTE
    MOVWF B'11010000',TRISC ;Set I/O for PORTC
    MOVWF B'00001111',TRISD ;Set I/O for PORTD
    MOVWF B'00000000',TRISE ;Set I/O for PORTE
    MOVWF B'10001000',TOCON ;Set up Timer0 for a loop time of 10 ms; b
    MOVWF B'00010000',PORTA ;Turn off all four LEDs driven from PORTA

    MOVWF B'11111111',TMR0H ;ADDED by AC
    MOVWF B'00000000',TMR0L ;ADDED by AC
    MOVWF B'000000100',ALIVECNT ;ADDED by AC
    return

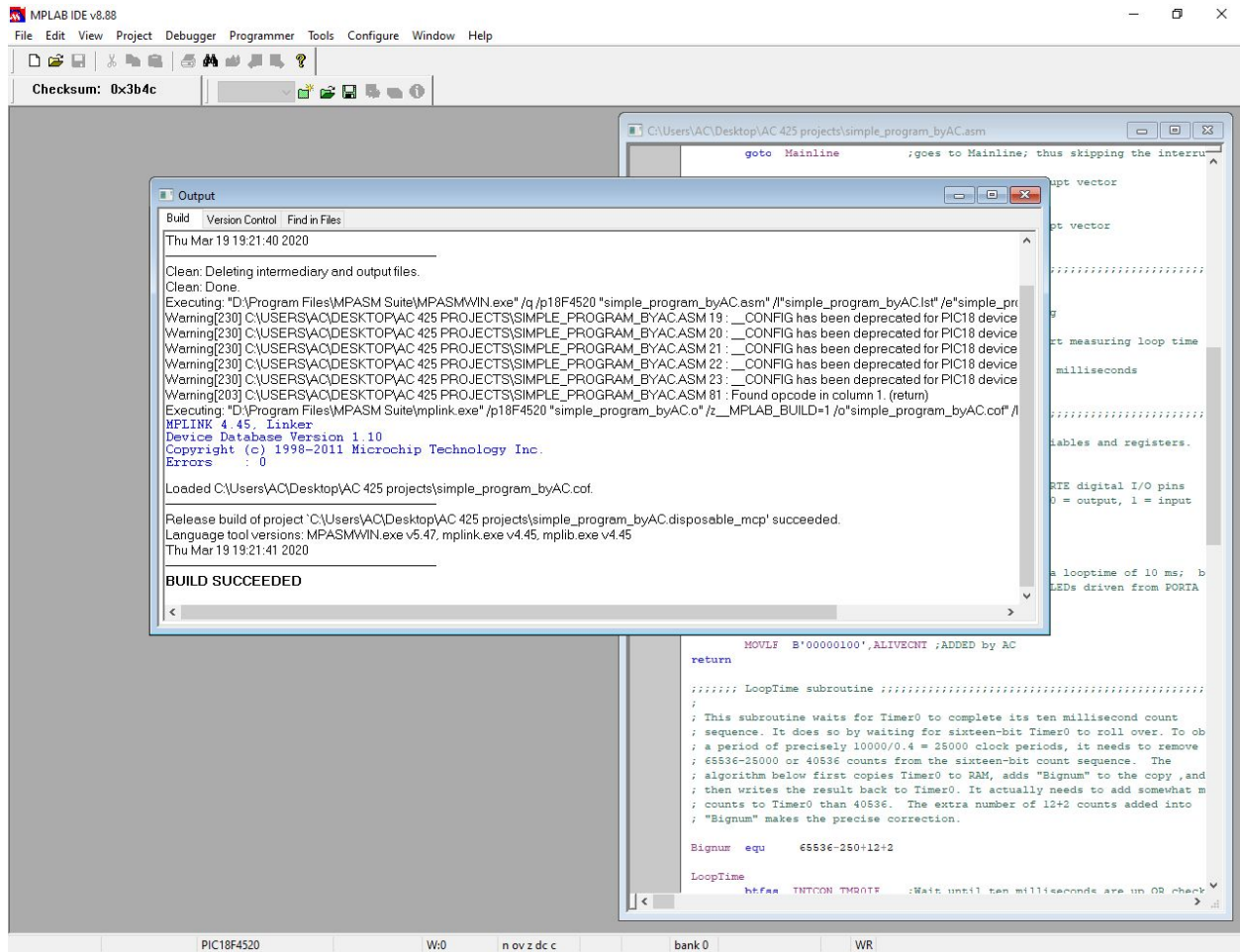
; LoopTime subroutine ;
; This subroutine waits for Timer0 to complete its ten millisecond count
; sequence. It does so by waiting for sixteen-bit Timer0 to roll over. To ob
; a period of precisely 10000/0.4 = 25000 clock periods, it needs to remove
; 65536-25000 or 40536 counts from the sixteen-bit count sequence. The
; algorithm below first copies Timer0 to RAM, adds "Bignum" to the copy, and
; then writes the result back to Timer0. It actually needs to add somewhat m
; counts to Timer0 than 40536. The extra number of 12+2 counts added into
; "Bignum" makes the precise correction.
Bignum equ 65536-250+12+2

LoopTime
    btfsc TMR0H,TMR0H ;Wait until ten milliseconds are up OR check
```

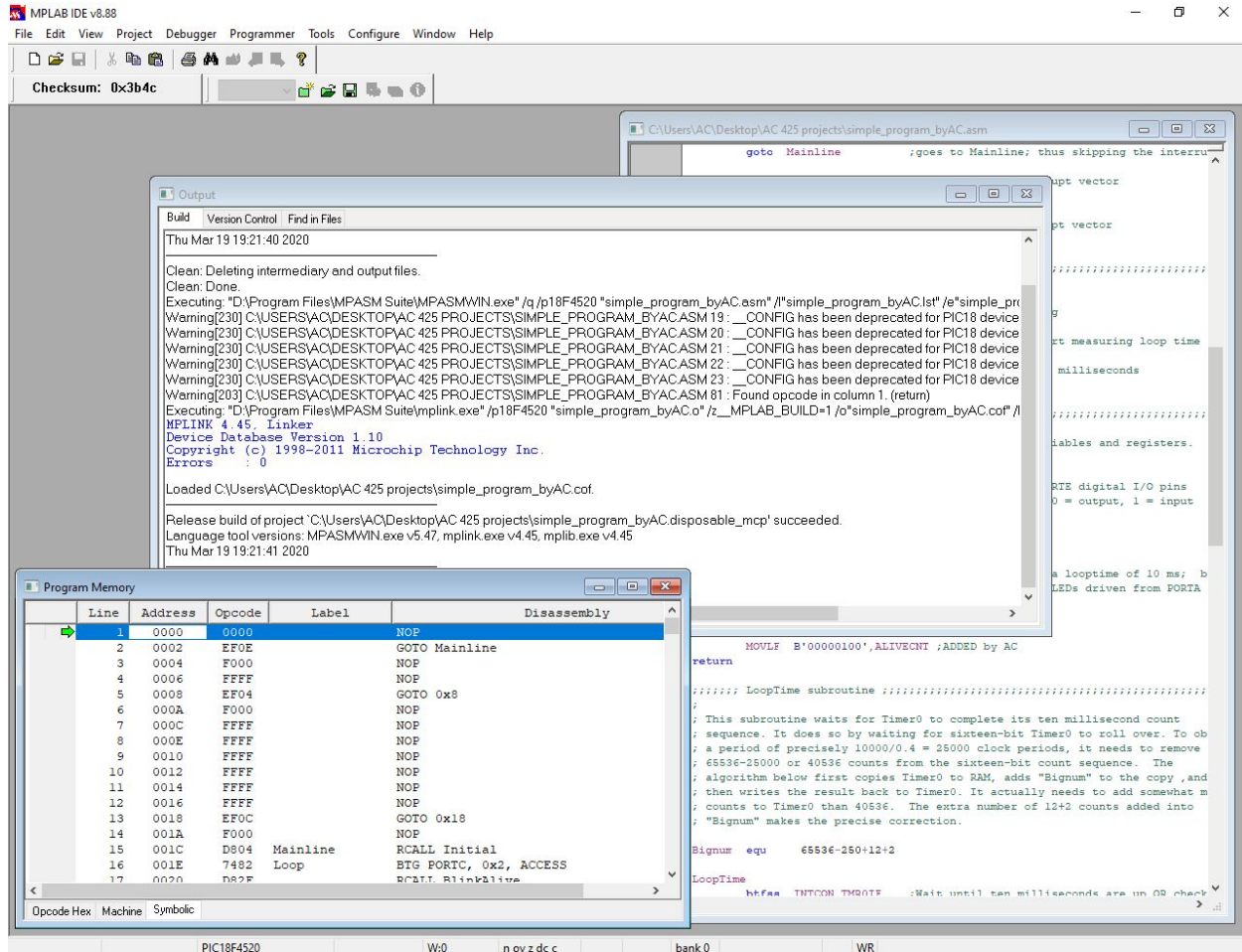
3. Take some time to read over the .asm file and note that it is very similar to the “P1_template.asm” file which was given to you at the beginning of the course. While reading the .asm file, you should try to convince yourself that this code performs the following functions:
 - a. Output a 50% duty cycle pulse train, with **0.1ms half-period**, at pin **RC2**.
 - b. Output a 75% duty cycle pulse train, with **0.4ms full-period**, at pin **RA4**.

IMPORTANT NOTE: In this .asm file, Timer0’s rollover time is determined by: “*Bignum equ 65536-250+12+2*,” i.e., LoopTime is essentially a delay of **0.1ms**. (This should be clear to you; otherwise see my Lecture 1 slides.) Please DO NOT modify this Bignum equation.

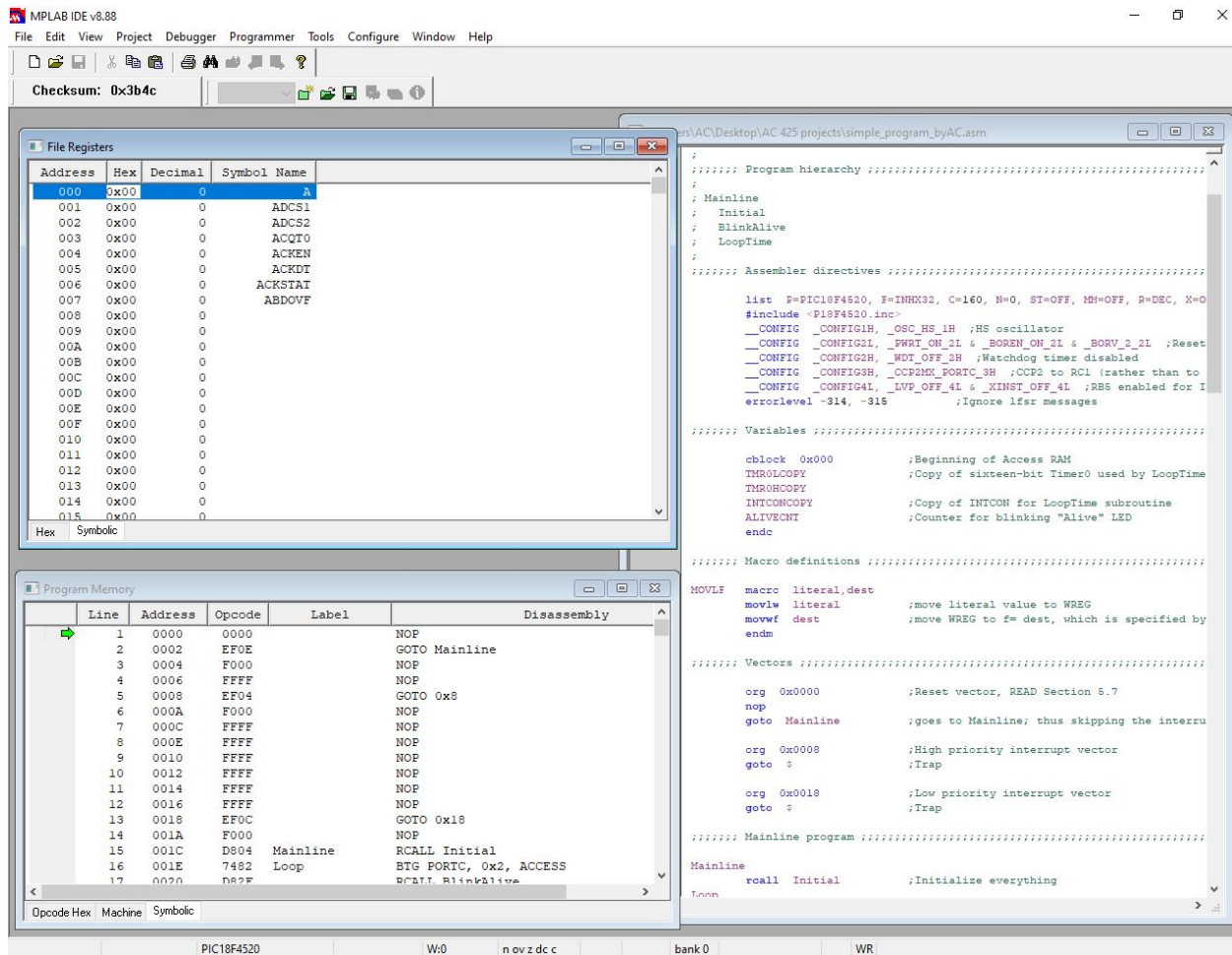
4. Press **Alt - F10** (as you usually do in the lab room) to compile the code, and make sure that you see the **BUILD SUCCEEDED** message. Your interface should look something like this:



- Click on **View >> Program Memory**, and a little window with header *Program Memory* will pop up. This window shows how the instructions in the program you just compiled are arranged in the PIC's program memory. It is imperative that you scroll up and down this *Program Memory* window and that you try to identify the instructions listed here, in a sequential order, with the instructions written in the *.asm* itself. Your interface should look something like this (make sure that you see a little green arrow, or cursor, on line 1 of the *Program Memory* window):



6. Click on **View >> File Registers** and another little window with header *File Register* will pop up. This window shows the names and contents of all general purpose registers (GPRs), special function registers (SFRs), and the programmer-defined variables stored in the PIC's operand memory, i.e., data memory. Again, it is imperative that you scroll up and down this new *File Register* window and try to see whether you can recognize some of the names of the registers (some of them should be familiar to you). Note that when you eventually simulate this *.asm* file (as you will do below), the contents of the each appropriate register in this window will change as the instructions in the code are executed by the CPU during simulation. Your interface should look something like this:



-
- The screenshot shows the MPLAB IDE v8.88 interface. The main window displays the assembly code for 'simple_program_byAC.asm'. The code includes a program hierarchy, mainline, and assembler directives. A 'Special Function Registers' window is open, showing the TMRO register at address 0x0000. The assembly code includes directives like #include, _CONFIG, and _list, as well as macro definitions and vector definitions. The status bar at the bottom shows the target device as PIC18F4520.
- Special Function Registers**
- | Address | SFR Name | Hex |
|---------|---------------|--------|
| 0x0000 | TMRO Internal | 0x0000 |
| 0x0001 | TMRO_Prescale | 0x00 |
| 0x0002 | TMR1 Internal | 0x0000 |
| 0x0003 | TMR1_Prescale | 0x00 |
| 0x0004 | TMR2_Prescale | 0x00 |
| 0x0005 | TMR3 Internal | 0x0000 |
| 0x0006 | TMR3_Prescale | 0x00 |
| 0x0007 | PORTA | 0x00 |
| 0x0008 | PORTB | 0x00 |
| 0x0009 | PORTC | 0x00 |
| 0x000A | PORTD | 0x00 |
| 0x000B | PORTF | 0x00 |
| 0x000C | LATA | 0x00 |
| 0x000D | LATB | 0x00 |
| 0x000E | LATC | 0x00 |
| 0x000F | LATD | 0x00 |
| 0x0010 | LATE | 0x00 |
| 0x0011 | TRISA | 0x00 |
| 0x0012 | TRISB | 0x00 |
| 0x0013 | TRISC | 0x00 |
| 0x0014 | TRISD | 0x00 |
| 0x0015 | TRISE | 0x00 |
- Assembly Code**
- ```

; Program hierarchy
; Mainline
; Initial
; BlinkAlive
; LoopTime

; Assembler directives

list P=PIC18F4520, F=INHX32, C=160, N=0, ST=OFF, MM=OFF, B=DEC, X=0
#include <P18F4520.inc>
_CONFIG _CONFIG1H, _OSC_HS_1H ;HS oscillator
_CONFIG _CONFIG2L, _FWDT_ON_2L & _BOREN_ON_2L & _BORV_2_2L ;Reset
_CONFIG _CONFIG2H, _WDT_OFF_2H ;Watchdog timer disabled
_CONFIG _CONFIG3H, _CCP2MX_PORTC_3H ;CCP2 to RCL (rather than to
_CONFIG _CONFIG4L, _LVP_OFF_4L & _XINST_OFF_4L ;RBS enabled for I
errorLevel -314, -315 ;Ignore 15r messages

; Variables

cblock 0x000 ;Beginning of Access RAM
TMROCOPY ;Copy of sixteen-bit Timer0 used by LoopTime
TMR0COPY
INTCONCOPY ;Copy of INTCON for LoopTime subroutine
ALIVECNT ;Counter for blinking "Alive" LED
endc

; Macro definitions

MOVLF macro literal,dest
movlw literal ;move literal value to WREG
movwf dest ;move WREG to f= dest, which is specified by
endm

; Vectors

org 0x0000 ;Reset vector, READ Section 5.7
nop
goto Mainline ;goes to Mainline; thus skipping the interrupt

org 0x0008 ;High priority interrupt vector
goto $;Trap

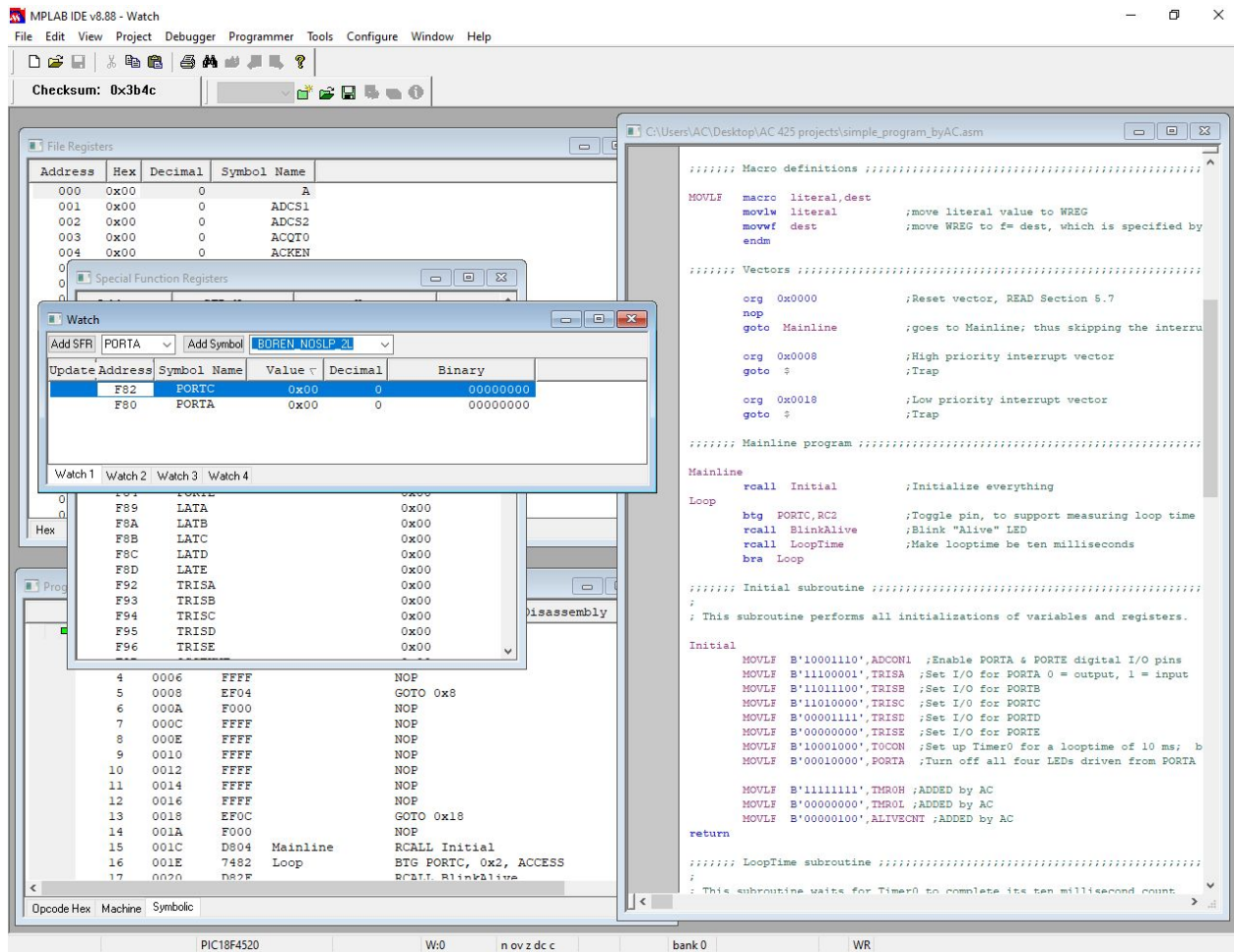
org 0x0018 ;Low priority interrupt vector
goto $;Trap

; Mainline program

Mainline
rcall Initial ;Initialize everything
Loop

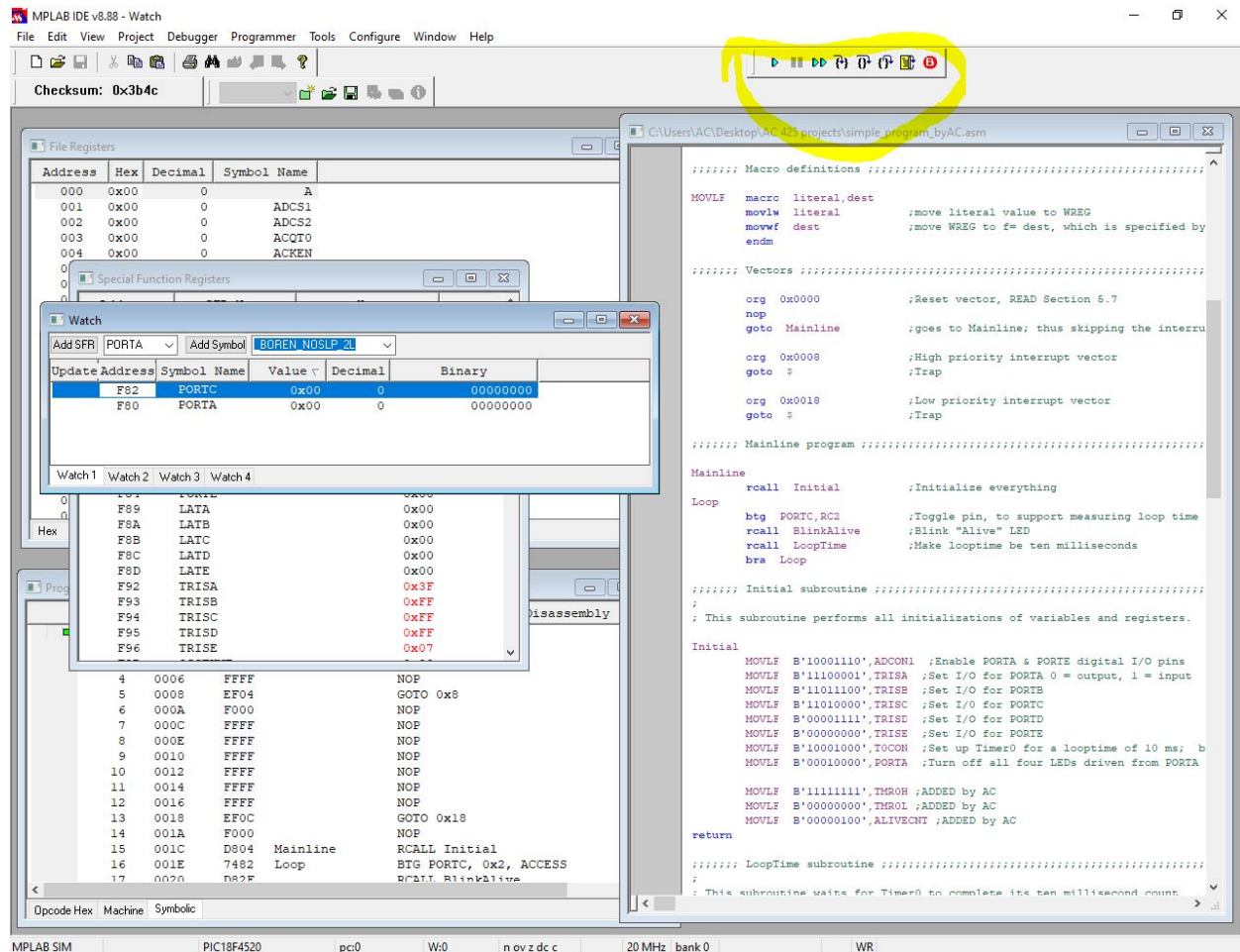
```

8. Click on **View >> Watch** and a fourth little window with header *Watch* will pop up. In this window you can select just a few registers of interest to be monitored while the simulator is running, as opposed to having *all* the registers in a single window as in the previous steps. Since we are specifically interested in monitoring the contents of the **PORTA** and **PORTC** SFRs during simulation, please consider the following steps (NOTE: at this point, the *.asm* must be already compiled before continuing):
  - a. Note that there are two drop-down menus in the *Watch* window. Click on the left drop-down menu, scroll down until you see **PORTA**, click on this register in order to select it, and finally click on the *Add SFR* button located on the left of this drop-down menu. Now you should see that the **PORTA** SFR has been added to the *Watch* window.
  - b. Repeat the above step in order to add **PORTC** SFR to the *Watch* window.
  - c. Note that by clicking on the header of the *Value* column in the *Watch* window, you may select the numerical format in which you would like to see the contents of the registers listed there; do this and select the decimal and binary representations. Your *Watch* window should look as follows:



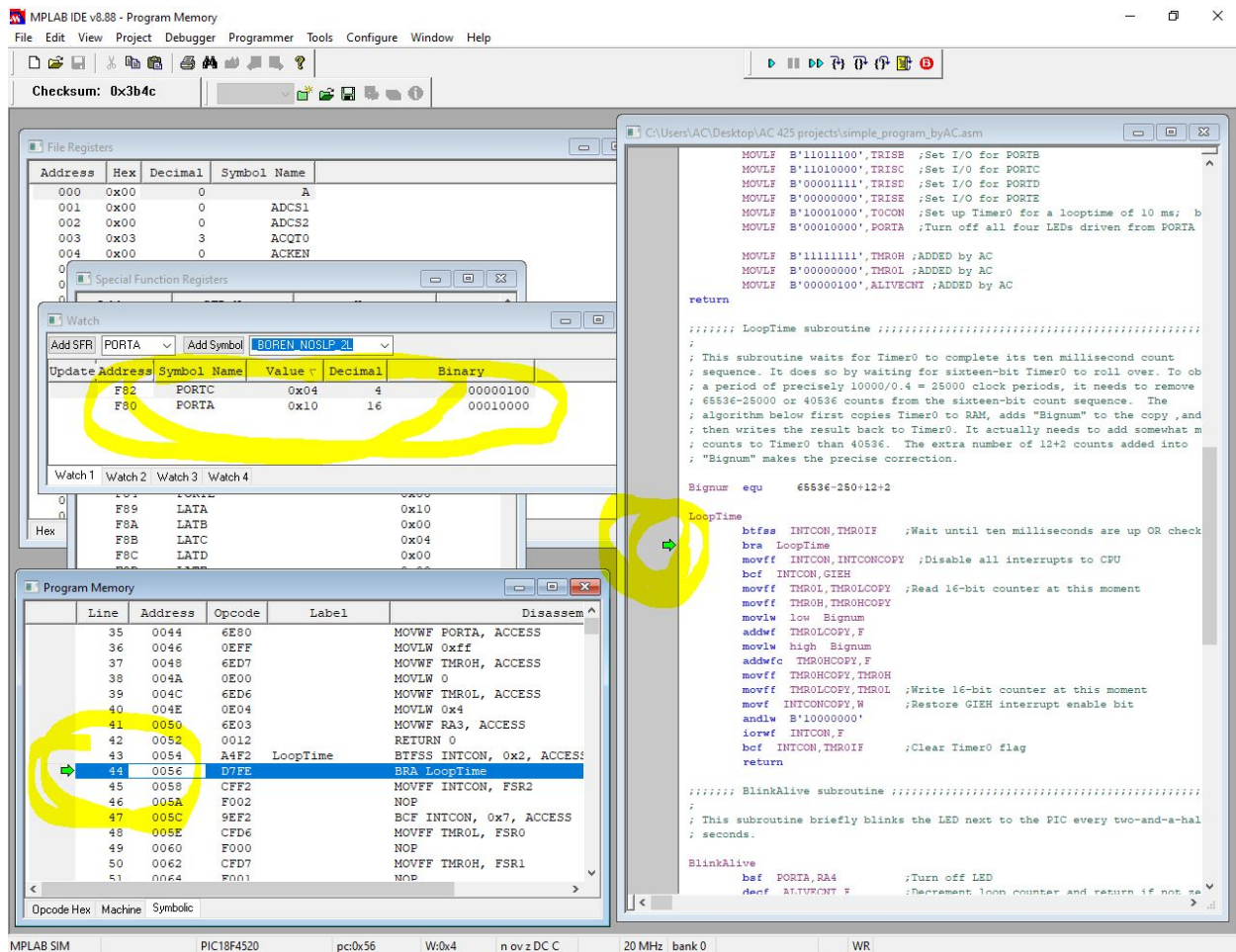
**IMPORTANT NOTE:** You may remove registers from the Watch window by right-clicking on them and then selecting *Delete* from the pop-up menu.

- Click on **Debugger >> Select Tool >> MPLAB SIM**. Note that a little tab (highlighted in yellow below) showing the *Run* and *Animate* buttons (among others) has appeared close to the top-right-hand side of your window. Please use your computer's cursor to hover over each button to discover its corresponding name and function. Your interface should look something like this:

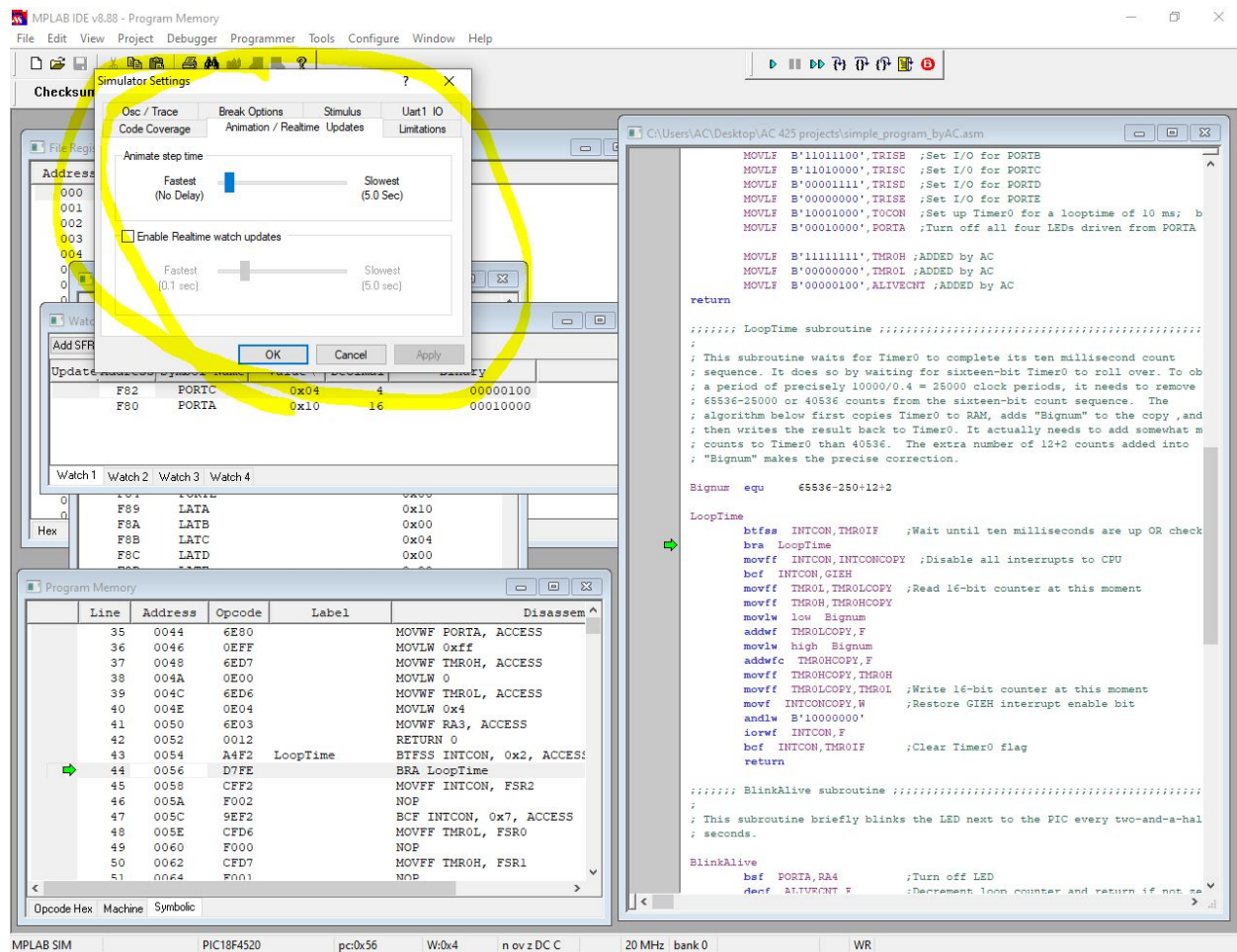




10. Click on the *Animate* button and notice how the little green arrow (a.k.a. cursor), which represents the program control of the microcontroller, moves sequentially through all the instructions in your code: See the *Program Memory* window and the *.asm* window itself. Take some time (minutes) to inspect the simulation's behaviour as it rips through your program, and, at the same time, watch the contents of the registers listed in the *Watch* window. Make sure that you see the bits (**RA4** and **RC2**) of registers **PORTA** and **PORTC** behave as discussed in step 3 above. That is, we are generating pulse trains at these pins, so make sure that you see the bits change accordingly throughout the simulation. Below is a screenshot of a particular instance during the animation:



11. Click on the *Halt* button to effectively terminate the simulation. At this point, you may be wondering whether or not the simulation time step can be increased. The answer is yes, and this is discussed in the next step.
12. Click on **Debugger >> Settings...** Then, on the new window that pops up, click on the **Animation / Realtime Updates** tab. There you will see a horizontal slider with header **Animate step time**. Moving this slider to the left will speed up the animation/simulation, while sliding it to the right will make the simulation step slower. You may set this setting to your preferred speed, and then click **OK**. I recommend sliding it all the way to the left (fastest setting). If you decide to change the time step setting here, please click on the *Animate* button again and go back to step 10 in order to inspect the contents of the registers in the *Watch* window under this different, faster setting. **NOTE:** Once you choose the fastest setting, there is no way to make the simulator go any faster. Below is a screenshot of the *Simulator Settings* window.



**IMPORTANT NOTE:** At some point before launching a new animation/simulation, you may decide that you want to clear all data in the *File Registers* window or in the *Special Function Registers* window. To do this click on **Debugger >> Clear Memory >> GPRs**. This will essentially wipe out all data in the operand memory only. That is, upon execution of a GPR memory clear, program memory will remain unaffected and thus it will not be cleared. If you, on the other hand, do wish to clear the program memory itself, then click **Debugger >> Clear Memory >> Program Memory**. Note that once you do this, the *Program Memory* window will contain a column of **NOPs**, which is the useful “no operation” instruction, i.e., program does nothing. In order to populate the Program Memory once more, it suffices to compile the code again using **ALT - F10** (I suggest that you try this.) Furthermore, there is no need to clear or wipe off the data in the registers listed in the *Watch* window because these will be updated automatically when you press the *Animate* button of the simulator.

13. Play around with the simulator as follows

- a. Add more registers to the *Watch* window and watch their contents evolve throughout a simulation.
- b. Create some new variables in the *.asm* file and try to then add them to the Watch window in order to track their behaviour. NOTE: in order to add registers which are not SFRs to the *Watch* window, you must use the second drop-down menu (which is located to the right of the drop-down menu used in step 8 above) from which you will look up the register's name and then add it using the appropriate button (*Add Symbol*).
- c. Try to be creative and enjoy visualizing the behavior of the registers as the CPU's control rips through the *.asm* file during simulation.

14. Get ready for the first assignment using the simulator. **Please take step 13 above seriously.**