CUNY - City College

Department of Electrical Engineering

EE 42500

Professor H. Pekcan

**Lab Report #4**

**Discrete-time Series Averaging Filters**

**Group members:** Maciej Jachimowicz and Jonathan M

**Due date**: 07/01/2020

**Semester:** Summer 2020

**I. Introduction:**

The objective of this laboratory experiment is to implement moving average filters with application to discrete time series using x[n] as input. To learn about this, we were given a series of tasks with instructions that we had to follow and reflect to with our experiment data.

**II. Task#1 (Laboratory)**

The first part of the experiment involved getting familiar to the new template of code (.asm file) and understanding it. To analyze and understand this template better, we were asked to compile and simulate the code and look closer into certain registers of interest. For that we used the Watch feature to monitor the register named “value” which corresponded to the values of our x[n]. Below, there is a code representing original template that we were given to work for this experiment and there is a figure showing the completion of the first task.

**This is the original template of code that we had to work with:**

;;;;;;; P5 for QwikFlash board ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; Use this template for Experiment 5

; This file was created by AC on 3/31/2020

;

;;;;;;; Assembler directives ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

list P=PIC18F4520, F=INHX32, C=160, N=0, ST=OFF, MM=OFF, R=DEC, X=ON

#include <P18F4520.inc>

\_\_CONFIG \_CONFIG1H, \_OSC\_HS\_1H ;HS oscillator

\_\_CONFIG \_CONFIG2L, \_PWRT\_ON\_2L & \_BOREN\_ON\_2L & \_BORV\_2\_2L ;Reset

\_\_CONFIG \_CONFIG2H, \_WDT\_OFF\_2H ;Watchdog timer disabled

\_\_CONFIG \_CONFIG3H, \_CCP2MX\_PORTC\_3H ;CCP2 to RC1 (rather than to RB3)

\_\_CONFIG \_CONFIG4L, \_LVP\_OFF\_4L & \_XINST\_OFF\_4L ;RB5 enabled for I/O

errorlevel -314, -315 ;Ignore lfsr messages

;;;;;;; Variables ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

cblock 0x000 ;Beginning of Access RAM

; --- BEGIN variables for TABLAT POINTER

; DO NOT MODIFY (created by AC)

value

counter

; --- END variables for TABLAT POINTER

; Create your variables starting from here

endc

;;;;;;; Macro definitions ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOVLF macro literal,dest

movlw literal

movwf dest

endm

;;;;;;; Vectors ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

org 0x0000 ;Reset vector

nop

goto Mainline

org 0x0008 ;High priority interrupt vector

goto $ ;Trap

org 0x0018 ;Low priority interrupt vector

goto $ ;Trap

;;;;;;; Mainline program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Mainline

rcall Initial ;Initialize everything

Loop

MOVLF 10,counter

MOVLF upper SimpleTable,TBLPTRU

MOVLF high SimpleTable,TBLPTRH

MOVLF low SimpleTable,TBLPTRL

label\_A

TBLRD\*+

movf TABLAT, W

movwf value ; value = x[n]

;;;;;;; NOTE FOR STUDENTS:

;

; Write the code for your moving average filter in

; the empty spaces below. Please create subroutines

; to make code your code transparent and easier to debug

;

; DO NOT MODIFY ANY OTHER PART OF THE THIS LOOP IN THE MAINLINE

;

; --------------------------------------------------------------

; BEGIN WRTING CODE HERE

; ---------------------------------

; (1) WRITE CODE FOR MEMORY BUFFER HERE

; you may write the full code

; here or call a subroutine

; ---------------------------------

; (2) WRITE CODE FOR ADDER AND DIVIDER HERE

; you may write the full code

; here or call a subroutine

; FINISH WRTING CODE HERE

; --------------------------------------------------------------

decf counter,F

bz label\_B

bra label\_A

label\_B

bra Loop

;;;;;;; Initial subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; This subroutine performs all initializations of variables and registers.

Initial

MOVLF B'10001110',ADCON1 ;Enable PORTA & PORTE digital I/O pins

MOVLF B'11100001',TRISA ;Set I/O for PORTA 0 = output, 1 = input

MOVLF B'11011100',TRISB ;Set I/O for PORTB

MOVLF B'11010000',TRISC ;Set I/0 for PORTC

MOVLF B'00001111',TRISD ;Set I/O for PORTD

MOVLF B'00000000',TRISE ;Set I/O for PORTE

MOVLF B'10001000',T0CON ;Set up Timer0 for a looptime of 10 ms; bit7=1 enables timer; bit3=1 bypass prescaler

MOVLF B'00010000',PORTA ;Turn off all four LEDs driven from PORTA ; See pin diagrams of Page 5 in DataSheet

return

;;;;;;; TIME SERIES DATA

;

; The following bytes are stored in program memory.

; Created by AC

; DO NOT MODIFY

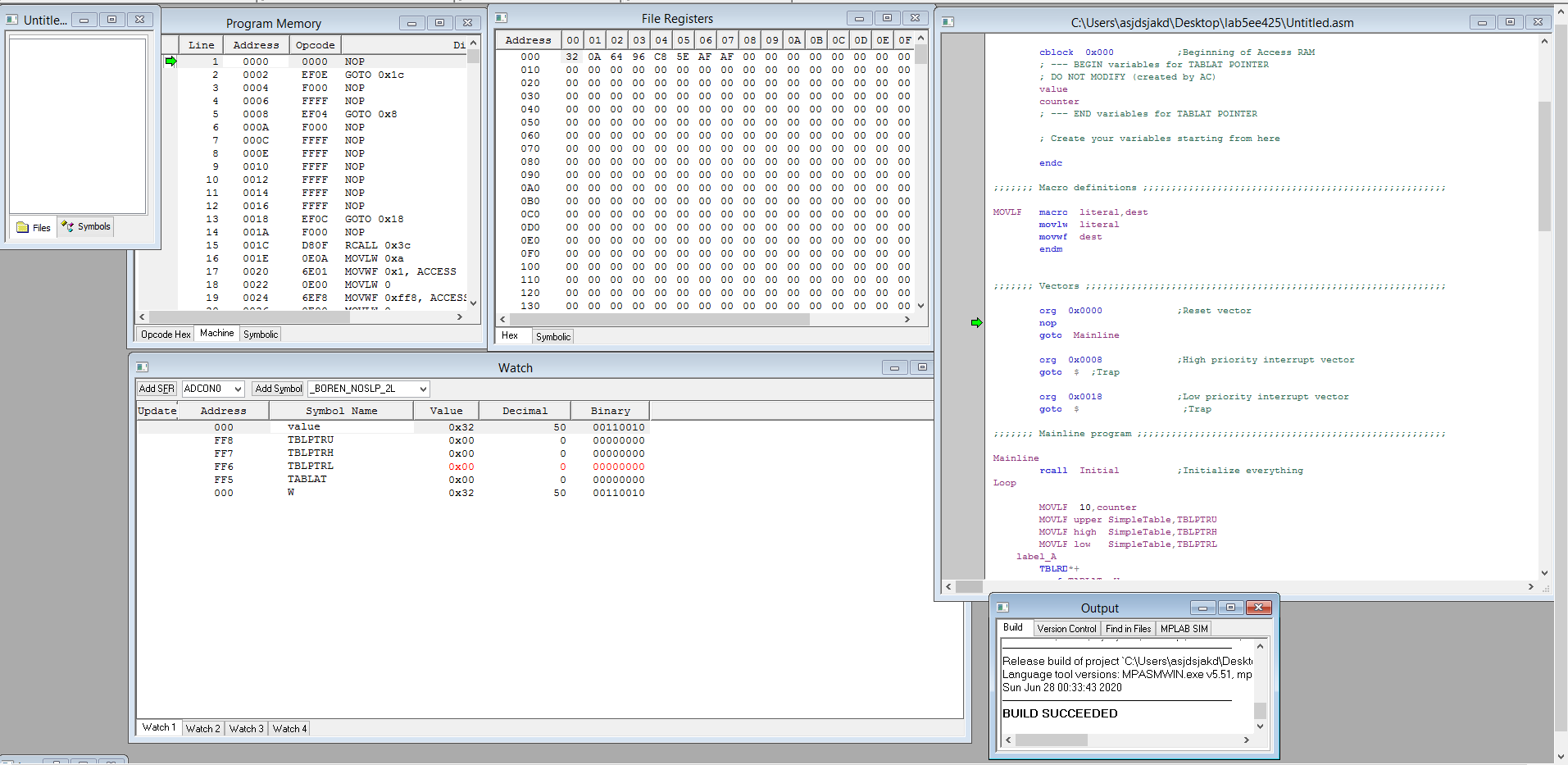
;

SimpleTable

db 0,50,100,150,200,250,200,150,100,50

; --------------------------------------------------------------

end

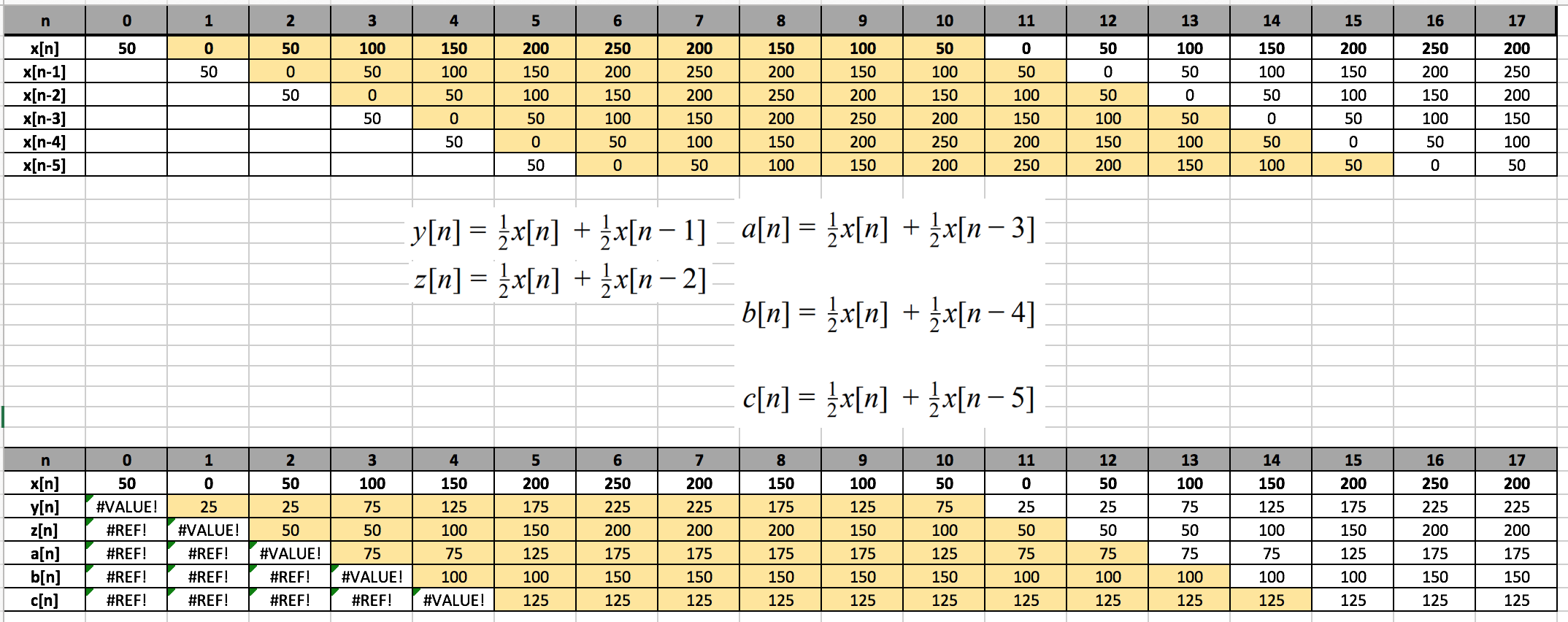


**Figure #1: This is the image of the completed Task #1.**

One thing that we noticed is that once the code is being compiled and ran for a while, the value of register “value” varies based on the values that are in simple table within time. In other words, the register “value” is assigned values from our table, 0,50,100,150,200,250,200,150,100,50 and the cycle is being repeated as the code is running.

**III. Task#2 (Laboratory)**

The second task of this experiment involved creating a table for several moving average filters by using a specific difference equation. The figure below will show all the tables had to be analyzed:



**Figure #2: This is the complete table for Task #2.**

This whole process starts with a simple x[n]as an input that has a series of numbers, exactly ten-numbers and follows by other data tables that are being calculated from other equations. What can be noticed is that for every subtraction from n in x[n], the series of numbers are shifted to right by a number that is being subtracted from. For the second table, x[n], y[n], z[n]. etc., the equation just says that the output is simply calculated for each n, as the average of two values of x[n] at consecutive time indices, namely, n and n-1. In others words, the current data value is assed to the previous data values and finally the resulting sum is divided by two. One thing that’s noticeable by looking at both tables is that if we looking for value of y[n], all that we have to do is just add value of x[n] and x[n-1] and divide by two. And this works for every different value of n for all other equations as well.

**IV. Task#3 (Laboratory)**

For the next part of this experiment we were asked to implement the moving average filters on the PIC using assembly language. At first, we worked on the y[n] which was a first-order filter in this experiment. One important concept to understand was that a certain values of x[n] had to be found first in order to compute any outputs for filters. For example, the values of x[n] and x[n-1] had to be found first to compute the value of y[n]. This was important because if this step was skipped, the code would never work. As a result of that, a memory buffer was necessary to be created to make the program work.

The function of the memory buffer is to save the data corresponding to three consecutive time indices in our computer memory. As the time series evolved in time, our memory buffer must be updated with appropriate consecutive values so that our filter would produce the correct average output at each time index.

**Assembly Language for the output of y[n] filter:**

;;;;;;; Assembler directives ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

list P=PIC18F4520, F=INHX32, C=160, N=0, ST=OFF, MM=OFF, R=DEC, X=ON

#include <P18F4520.inc>

\_\_CONFIG \_CONFIG1H, \_OSC\_HS\_1H ;HS oscillator

\_\_CONFIG \_CONFIG2L, \_PWRT\_ON\_2L & \_BOREN\_ON\_2L & \_BORV\_2\_2L ;Reset

\_\_CONFIG \_CONFIG2H, \_WDT\_OFF\_2H ;Watchdog timer disabled

\_\_CONFIG \_CONFIG3H, \_CCP2MX\_PORTC\_3H ;CCP2 to RC1 (rather than to RB3)

\_\_CONFIG \_CONFIG4L, \_LVP\_OFF\_4L & \_XINST\_OFF\_4L ;RB5 enabled for I/O

errorlevel -314, -315 ;Ignore lfsr messages

;;;;;;; Variables ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

cblock 0x000 ;Beginning of Access RAM

; --- BEGIN variables for TABLAT POINTER

; DO NOT MODIFY (created by AC)

value

counter

; --- END variables for TABLAT POINTER

; Create your variables starting from here

ProgMem

TEMP

TEMPH

valueH

SUML

SUMH

ResL

ResH

endc

;;;;;;; Macro definitions ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOVLF macro literal,dest

movlw literal

movwf dest

endm

;;;;;;; Vectors ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

org 0x0000 ;Reset vector

nop

goto Mainline

org 0x0008 ;High priority interrupt vector

goto $ ;Trap

org 0x0018 ;Low priority interrupt vector

goto $ ;Trap

;;;;;;; Mainline program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Mainline

rcall Initial ;Initialize everything

Loop

MOVLF 10,counter

MOVLF upper SimpleTable,TBLPTRU

MOVLF high SimpleTable,TBLPTRH

MOVLF low SimpleTable,TBLPTRL

incf TBLPTRL; Simpletable was expanded, adjust is made to the pointer

label\_A

tblrd\*+

movf TABLAT, W

movwf value ; value = x[n]

;;;;;;; NOTE FOR STUDENTS:

;

; Write the code for your moving average filter in

; the empty spaces below. Please create subroutines

; to make code your code transparent and easier to debug

;

; DO NOT MODIFY ANY OTHER PART OF THE THIS LOOP IN THE MAINLINE

;

; --------------------------------------------------------------

; BEGIN WRTING CODE HERE

movff TBLPTRL,ProgMem ;save pointer location

tblrd\*-

tblrd\*-

tblrd\*- ;change pointer to 3 steps back, points to x[n-1]

movf TABLAT, W ;TABLAT is the value that corresponds to x[n-1]

movwf TEMP

addwf value,W ; x[n]+x[n-1]

movwf SUML ;result into SUML

movf TEMPH,W

addwfc valueH,w ;x[n]+x[n-1] with bit carry

movwf SUMH ;result into SUMH

rrcf SUMH, W ;rotate to the right for both high and low, for division by 2

movwf SUMH

rrcf SUML, W

movwf SUML

;;;;;;;;;;;;;;;;;;;;;;Final Result;;;;;;;;;;;;;;;;;;;;;

movff SUML, ResL ;copy final result into ResL and ResH

movff SUMH, ResH

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

movff ProgMem, TBLPTRL ; restore pointer location

; FINISH WRTING CODE HERE

; --------------------------------------------------------------

decf counter,F

bz label\_B

bra label\_A

label\_B

bra Loop

;;;;;;; Initial subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; This subroutine performs all initializations of variables and registers.

Initial

MOVLF B'10001110',ADCON1 ;Enable PORTA & PORTE digital I/O pins

MOVLF B'11100001',TRISA ;Set I/O for PORTA 0 = output, 1 = input

MOVLF B'11011100',TRISB ;Set I/O for PORTB

MOVLF B'11010000',TRISC ;Set I/0 for PORTC

MOVLF B'00001111',TRISD ;Set I/O for PORTD

MOVLF B'00000000',TRISE ;Set I/O for PORTE

MOVLF B'10001000',T0CON ;Set up Timer0 for a looptime of 10 ms; bit7=1 enables timer; bit3=1 bypass prescaler

MOVLF B'00010000',PORTA ;Turn off all four LEDs driven from PORTA ; See pin diagrams of Page 5 in DataSheet

MOVLF B'00000000',ProgMem

MOVLF B'00000000',TEMP

MOVLF B'00000000',TEMPH

MOVLF B'00000000',value

MOVLF B'00000000',valueH

MOVLF B'00000000',SUML

MOVLF B'00000000',SUMH

MOVLF B'00000000',ResL

MOVLF B'00000000',ResH

return

;;;;;;; TIME SERIES DATA

;

; The following bytes are stored in program memory.

; Created by AC

; DO NOT MODIFY

;

SimpleTable

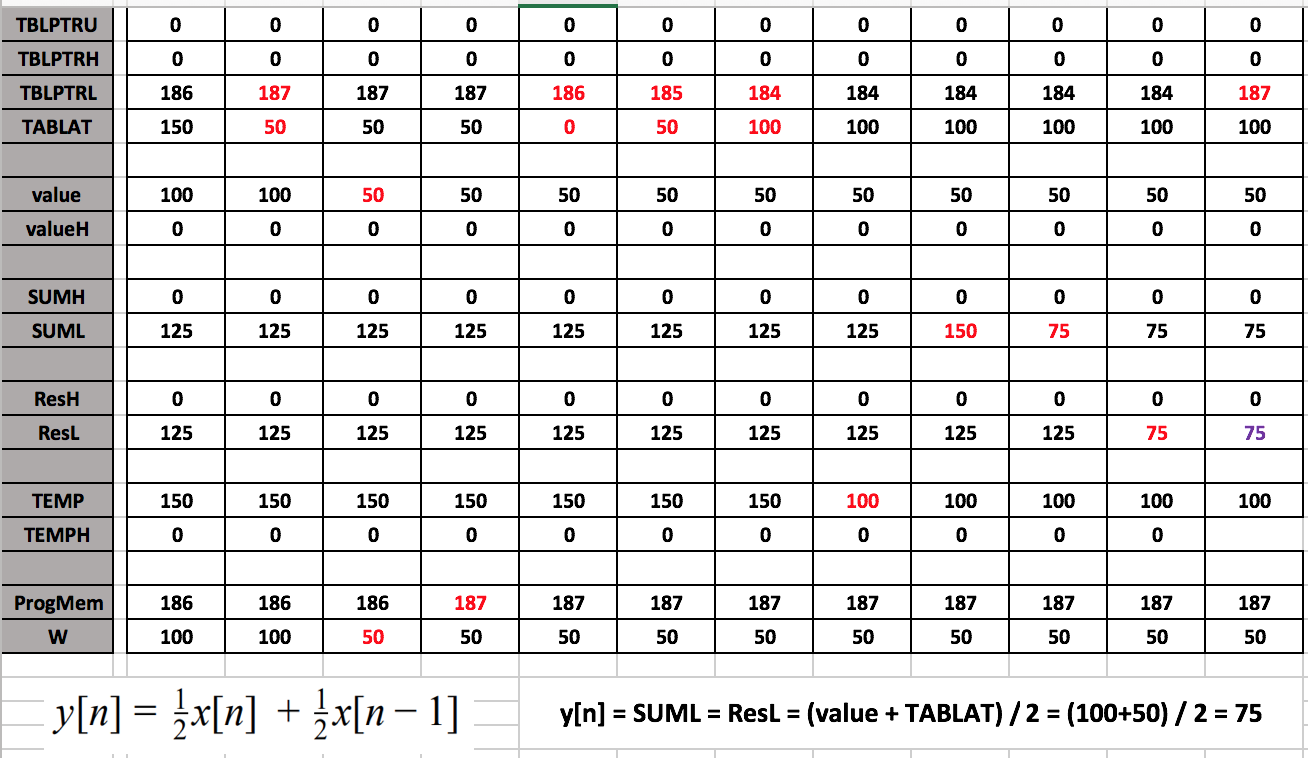
db 50,0,50,100,150,200,250,200,150,100,50,0

; --------------------------------------------------------------

end

Above is the code that we wrote to obtain the output value of y[n]. The code includes the template that was given to us with certain additions. First thing, we declared new variables which are ProgMem, TEMP, TEMPH, valueH, SUML, SUMH, ResL, ResH. All of these new variables were initialized in the initial subroutine and were assigned binary value of zero.

To simplify understanding the code, I will include a table with binary representation of how registers change as the code is being compiled.

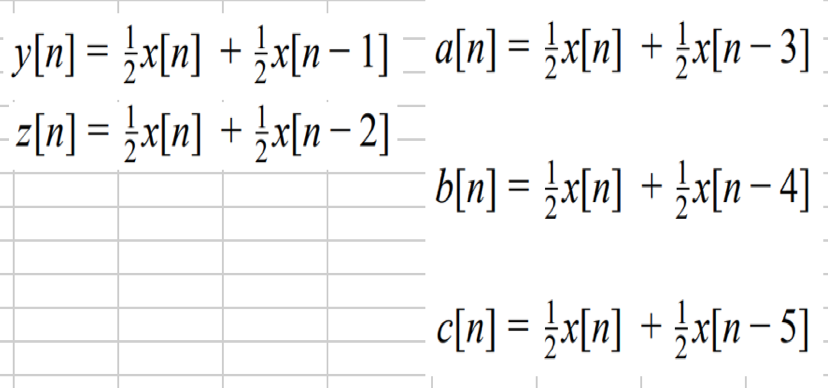


**Figure #3: This is one cycle of assembly language for Task#3**

As, I mentioned above the first thing was to declare new variables in order to start with the code. Second, it was finding a way to save and store two values of x[n] and x[n-1] within the code in order be able to calculate the value of our output y[n]. To do so, first we assigned value of x[n] to register named “value” and we saved the pointer location in the ProgMem register using movff command. Then, we used tblrd\*- command to change pointer location to 3-three steps back which pointed us to x[n-1] value. We assigned that value of x[n-1] to TABLAT register and then performed mathematic equation to find y[n]. The command of addwf was used to add both registers and the result of addition was stored in the SUML register. By using the command of rrcf, our result was divided by 2-two which was what we needed. Finally, the final result is stored in the register of SUML. The table above clearly illustrates how values in the registers shift and it’s easier to navigate and understand the entire concept. Although, this table only shows one cycle, because the SImpleTable consists of more data values of x[n], therefore this cycle is ran for all the values. At the end of the code, we restore the pointer location.

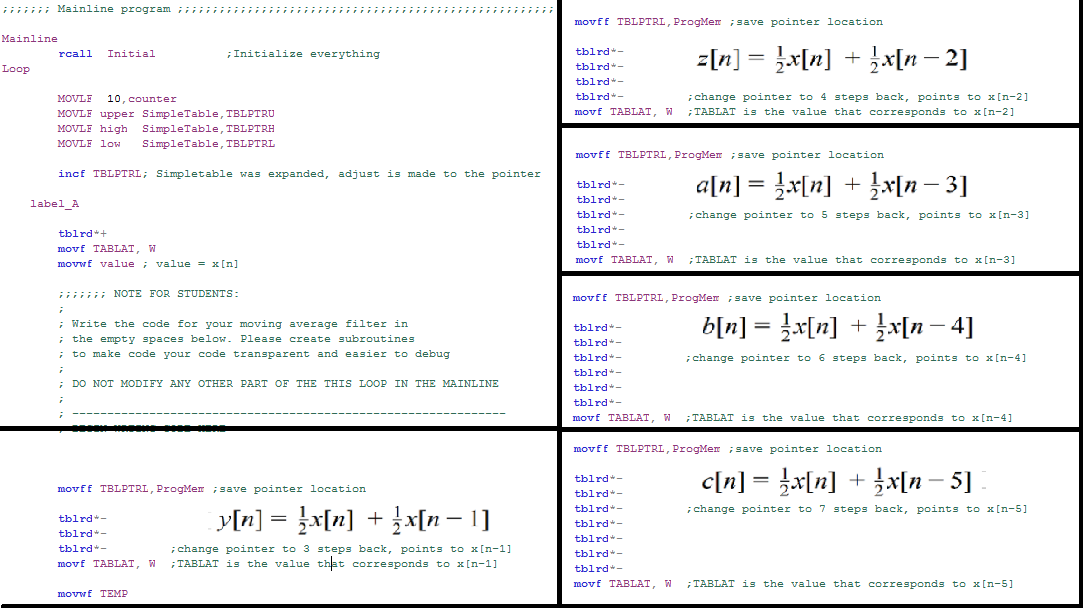
**V. Task#4 (Laboratory)**

The last task was not much different that task#3 because it involved the same procedure and only required adjustment was to change pointer location for each moving average filter. For this task, we were asked to write individual .asm codes for z[n], a[n], b[n], c[n] filters.



**Figure #4: Equations for moving average filters**

The figure above shows all the equations that we had to use to find all our moving average filters. Like, I said task #3 and task#4 are very alike because all that had to be changed is the pointer location or in simple words the location of second register that includes subtraction from n value (n-1, n-2, n-3,etc.)

The code itself for all filters is exactly the same as code for filter of y[n] with only little variation to the code, therefore I will only show what is being modified for every single filter instead of repeating the same code several times in this document.

**Figure #5: The .asm code variation for all filters**

Now, the figure #5 shows what has to be changed in the .asm code for filter y[n] in order to obtain values for other moving average filters. As It can be seen, the only modification is the amount of tblrd\*- commands used for each filter, which corresponds for moving pointer location backwards. For instance, to obtain the output of y[n] we had to assign and store the value of x[n-1] , therefore we had to change the pointer location to 3-three steps back and store the value into declared by us register. This same procedure is done for all other moving average filters with only difference of changing the pointer location by different number of steps backwards.

**VI. Conclusion:**

This experiment was a great experience to learn how to implement moving average filters on the PIC using the assembly language. Not that only, we got a better understanding of the subject, but we also learned about memory buffers, new set of instructions (such as movff, movf, addwf, rrcf, and andlw), and also familiarized ourselves better with MPLAB software. In one sentence, I can conclude that the experiment went successfully and without any complications providing us with clear and correct data results.