CUNY - City College

Department of Electrical Engineering

EE 42500

Professor H. Pekcan

**Lab Report #6**

**Discrete-time Series Averaging Filters – Part II**

**Group members:**

Maciej Jachimowicz and Jonathan Martinez

**Due date**: 07/08/2020

**Semester:** Summer 2020

**I. Introduction:**

The objective of this laboratory experiment is very similar to our previous lab experiment. It is still about implementing moving average filters with application to discrete time series, however this time is a little more complex. This time, we have to work with more variables and not just two like in Lab #5. As a result of that, certain methods will have to be taken into consideration in order to compile the code and make It to work. For example, addition can’t be performed with more than two registers at a time and this is the main idea of this experiment. Our primary goal in this experiment is to learn how to work with many variables at the same time to reach the final result. Therefore, 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

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

; Change value for counter depending

; on period of time series that you wish to use

;

MOVLF 2,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

;

; Choose your Periodic Sequence

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

; time series X1

SimpleTable ; ---> period 2

db 180,240

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

; time series X2

;SimpleTable ; ---> period 4

;db 180,240,200,244

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

; time series X3

;SimpleTable ; ---> period 6

;db 180,240,200,244,216,236

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

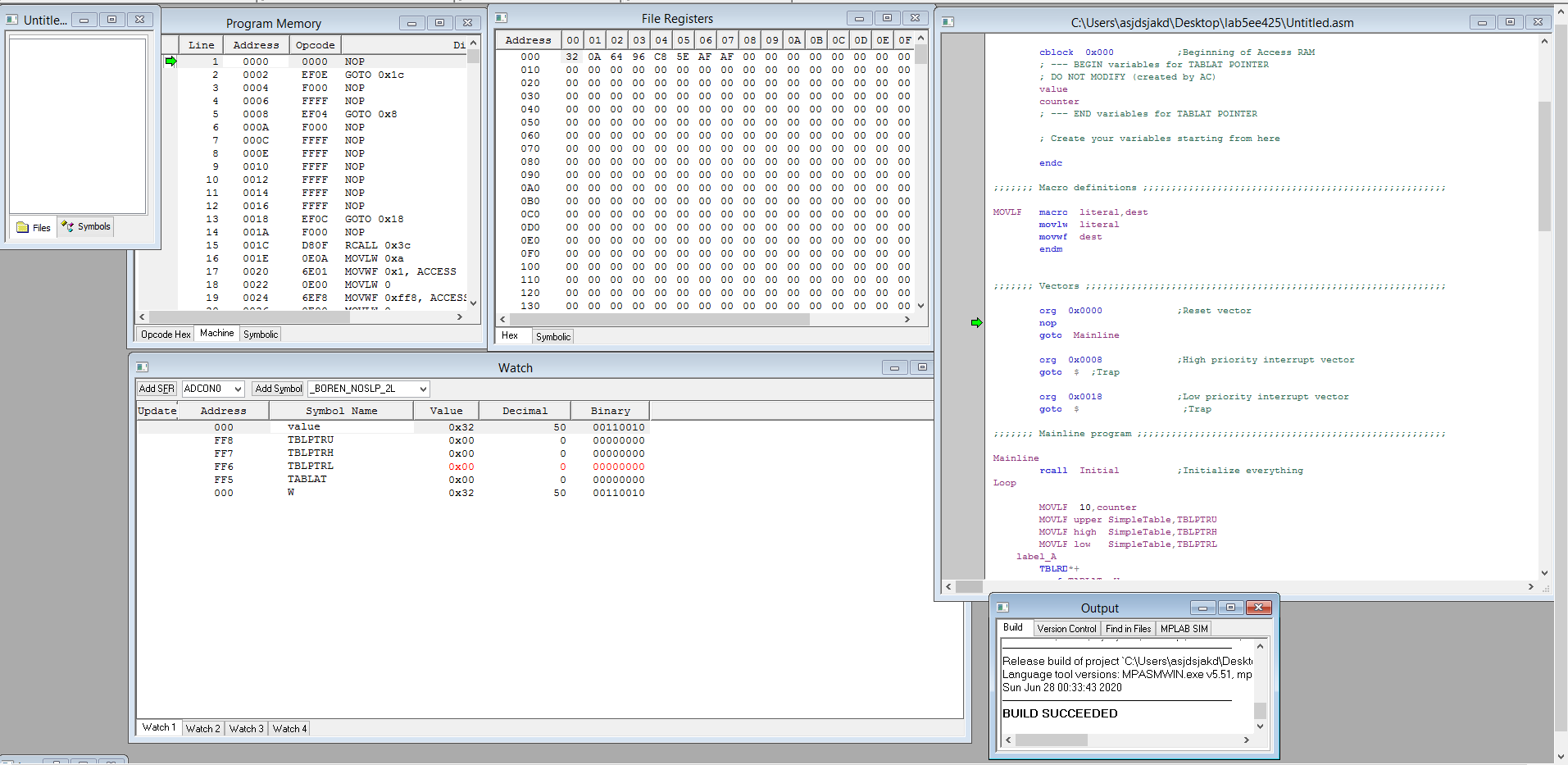
; time series X4

;SimpleTable ; ---> period 8

;db 180,240,200,244,216,236,160,176

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

end

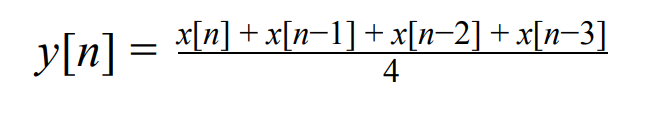


**Figure #1: This is the image of the completed Task #1.**

For the first task, we were asked to comment out and uncomment certain lines of code and also change the number of period between 2,4,6, and 8. This is because for every individual period, there was a corresponding simple table within the code that contained a different number of values. The first task was just an introduction of what had to be done in the next tasks.

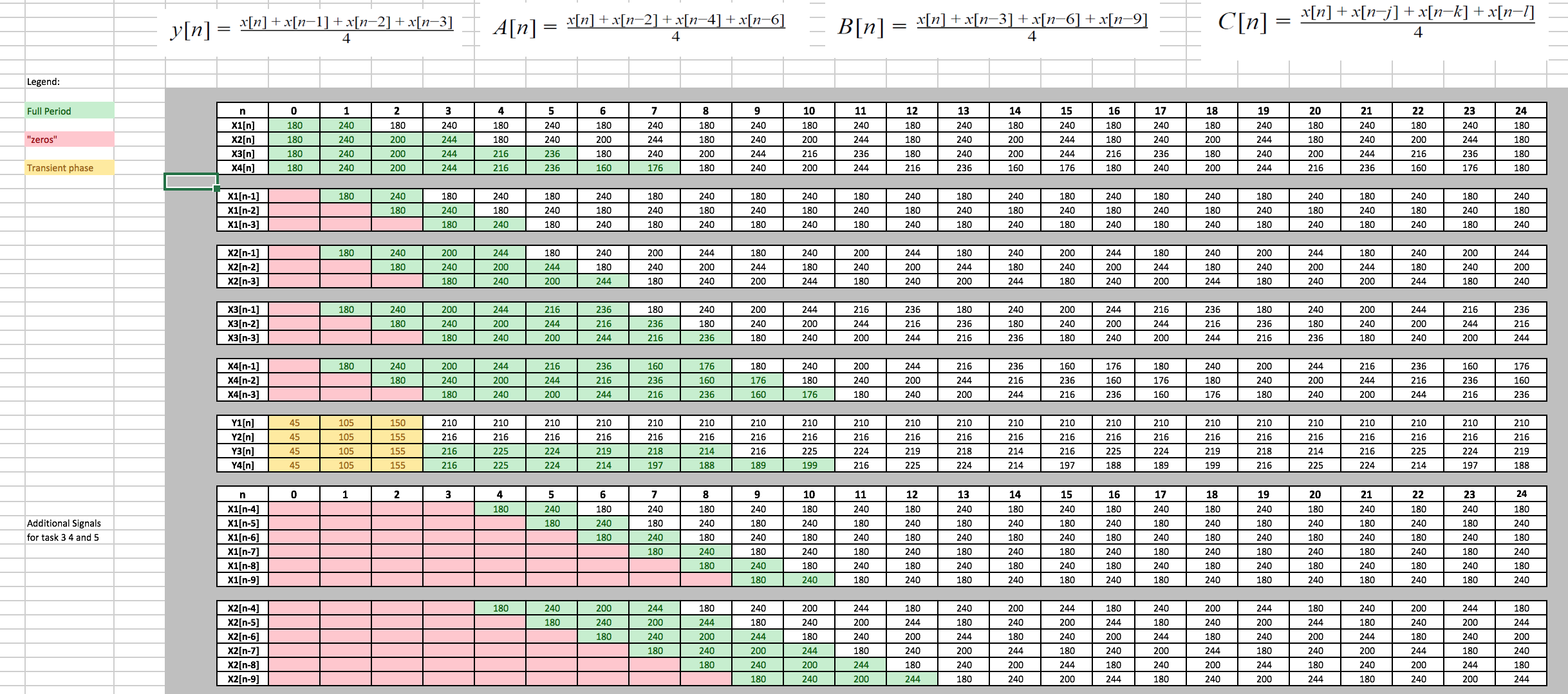
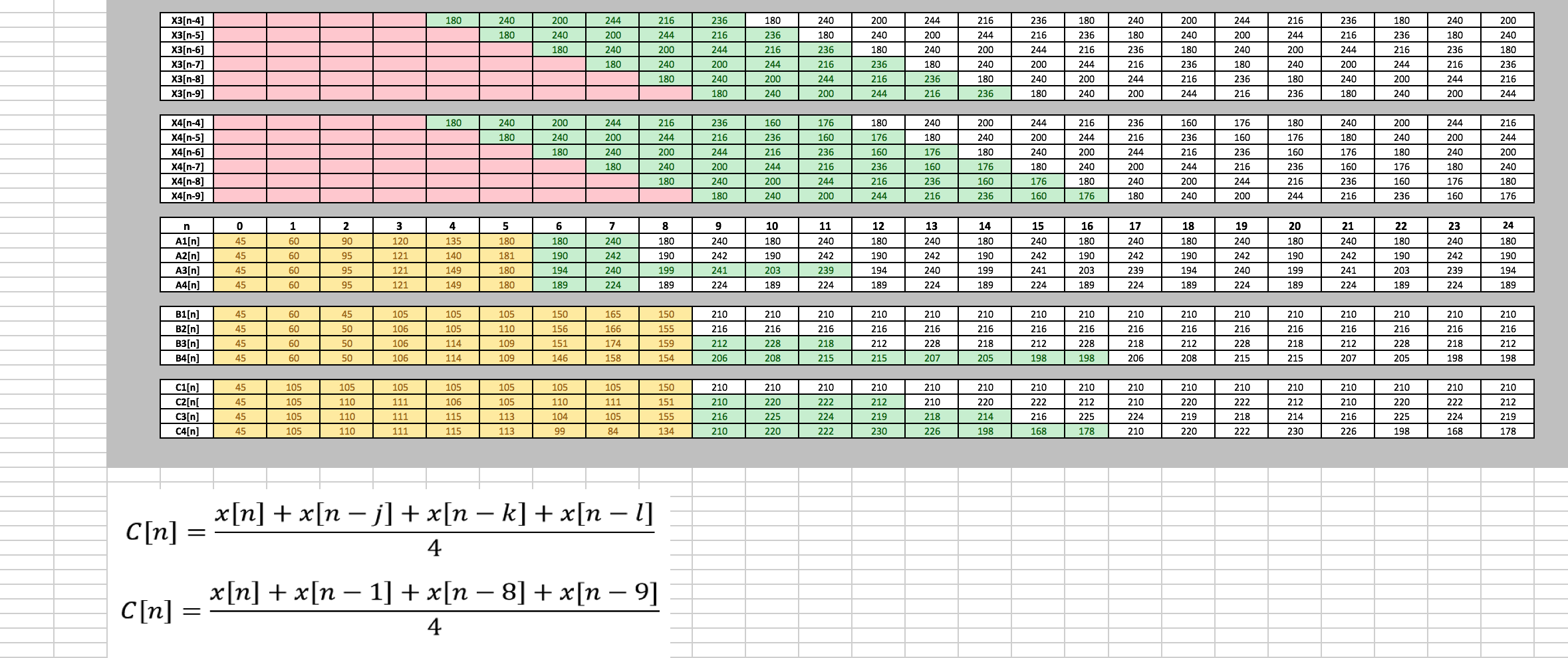
**III. Task#2 (Laboratory)**

The second task of this experiment involved writing an assembly language code to produce the output for the following equation:



Our goal was to use the template of the code that was initially provided to us and modify it, so it would be able to produce this output. Also, this output had to be done in all periods of 2,4,6, and 8 that each represented a different SimepleTable data. Now, like I said before, unlike the previous lab this experiment forces us to work with more variables and therefore more variables and initializations are required to make this program run.

Before we even started coding and figuring out how to complete these tasks, we created a table for every single output equation that we had to work with and calculated the results by hand to get a better understanding of what we are looking for. Below there is a table representing all data results for every output equation for every period that we will have to write assembly language codes.



**Figure #2: Table representing output values for every equations and every period**

The way we approached this experiment, we created two summations where we added two registers at a time and stored the results into two individual registers. After, the two summations were done, we added the first and second summations to our final summation. Lastly, our final summation we divided twice by performing rotation to the right for both high and low, and therefore the result was divided by an overall division of 4. Eventually, this result we placed into registers ResL and ResH for our final result.

**This is the .asm for Task#2**

* **Y[n]**
* **T = 2 (Period =2)**

;;;;;;; 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

ProgMem

valueH

xn1

xn2

xn3

SUMH

SUML

SUM2H

SUM2L

TEMP

TEMPH

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

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

; Change value for counter depending

; on period of time series that you wish to use

;

MOVLF 2,counter

MOVLF upper SimpleTable,TBLPTRU

MOVLF high SimpleTable,TBLPTRH

MOVLF low SimpleTable,TBLPTRL

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

; BEGIN WRTING CODE HERE

label\_A

movff xn2, xn3 ;xn3 = xn2

movff xn1, xn2 ;xn2 = xn1

movff value, xn1 ;xn1 = value

TBLRD\*+

movf TABLAT, W

movwf value ; value = TABLAT (current value of the program memory)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;First Summation ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

movf xn3, W ; changes wreg to xn3

addwf value,W ; x[n]+x[n-3]=SUML

movwf SUML ;result into SUML

movf TEMPH,W

addwfc valueH,w ;x[n]+x[n-3]=SUMH with bit carry

movwf SUMH ;result into SUMH

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Second Summation;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

movf xn1,W ;changes wreg to xn1

addwf xn2,W ; x[n-1]+x[n-2]=SUM2L

movwf SUM2L ;result into SUM2L

movf TEMPH,W

addwfc valueH,w ;x[n-1]+x[n-2]=SUM2H with bit carry

movwf SUM2H ;result into SUM2H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Final Summation;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

movf SUML,W

addwf SUM2L,W ; SUML+SUM2L=SUM2L

movwf SUM2L ;result into SUM2L

movf SUMH,W

addwfc SUM2H,w ; SUMH+SUM2H=SUM2H

movwf SUM2H ;result into SUM2H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Division by 4;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

rrcf SUM2H, W ;rotate to the right for both high and low, for division by 2

movwf SUM2H

rrcf SUM2L, W

movwf SUM2L

rrcf SUM2H, W ;2nd rotate is performed for both high and low, this produces

movwf SUM2H ;an overall division by 4

rrcf SUM2L, W

movwf SUM2L

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

movff SUM2L, ResL ;copy final result into ResL and ResH

movff SUM2H, ResH

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

; 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',value

MOVLF B'00000000',valueH

MOVLF B'00000000',xn1

MOVLF B'00000000',xn2

MOVLF B'00000000',xn3

MOVLF B'00000000',SUML

MOVLF B'00000000',SUMH

MOVLF B'00000000',SUM2L

MOVLF B'00000000',SUM2H

MOVLF B'00000000',TEMP

MOVLF B'00000000',TEMPH

MOVLF B'00000000',ResL

MOVLF B'00000000',ResH

return

;;;;;;; TIME SERIES DATA

;

; The following bytes are stored in program memory.

; Created by AC

;

; Choose your Periodic Sequence

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

; time series X1

SimpleTable ; ---> period 2

db 180,240

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

; time series X2

;SimpleTable ; ---> period 4

;db 180,240,200,244

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

; time series X3

;SimpleTable ; ---> period 6

;db 180,240,200,244,216,236

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

; time series X4

;SimpleTable ; ---> period 8

;db 180,240,200,244,216,236,160,176

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

end

Now, we will talk step by step of how this code works and how it obtains the values for the output equation of y[n]. In this case, we are working with period of 2 which has two values in the SimpleTable, 180db and 240db. As the code runs, the counter gets value of 2 and will run till the value goes to 0 and then the cycle repeats. The TABLAT is the current value of the program memory and it obtains value of 180 db as program runs at first. Since, we initialized all our registers to values of 0, the program needs to run for a bit because our values for xn1 to xn3 are at 0, and the code needs to run a few cycles to assign proper values based on the SimpleTable values. Therefore, the first initial summations are basically not important till all of our xn regsiters have right values assigned. Later, the xn1 register gets a value of the value register and therefore gets value of 180 db, while the value register moves to value of 240 db. Next, the xn2 register gets a value of xn1 register and it is at 180 db, while the x1 register gets a value of 240 db. The value register goes back to 180 db. After, the code runs for a while this is how the registers of xn3, xn2, xn1 and value will look like.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Register** | **value (xn)** | **xn1** | **xn2** | **xn3** |
| **Value** | **240** | **180** | **240** | **180** |

At this point, when we continue running the code, we will finally obtain our final result for our y[n] equation because, now our code assigned proper values to our registers and therefore we have needed values to complete the equation.

In the first summation, the registers x[n] and x[n-3] are added and the result is stored in SUML. If the number is too high to be stored in one register, then it will be stored in SUMH as well. This is slightly confusing, but we have to remember that we are working in a 16 bit microcontroller therefore when we deal with higher numbers, we have to assign them to multiple registers in order to carry bit. So, we see here that SUML = 164, where SUMH = 1 or SUML = 10100100 where SUMH = 00000001. At the end this is equal to **0000000110100100 = 420.** Second summations produce the same output of 420, yet we use different registers to store it. We use the SUM2L and the SUM2H.

**First summation (Table):**

|  |  |  |
| --- | --- | --- |
| **Register** | **SUML** | **SUMH** |
| **Value (Decimal)** | **164** | **1** |
| **Value (Binary)** | **10100100** | **00000001** |
| **Final Value** | **0000000110100100 = 420** | |

**Second summation (Table):**

|  |  |  |
| --- | --- | --- |
| **Register** | **SUM2L** | **SUM2H** |
| **Value (Decimal)** | **164** | **1** |
| **Value (Binary)** | **10100100** | **00000001** |
| **Final Value** | **0000000110100100 = 420** | |

Lastly, in the final summation we add the register of SUML, SUMH and SUM2L, SUM2H to each other and store the value to the register SUM2L and the SUM2H. This produced the output of **SUM2L = 72 = 01001000** and **SUM2H = 3 = 00000011.** At the end this is equal to **0000000110100100 = 840.**

**Final summation (Table):**

|  |  |  |
| --- | --- | --- |
| **Register** | **SUML+SUM2L=SUM2L** | **SUMH+SUM2H=SUM2H** |
| **Value (Decimal)** | **72** | **3** |
| **Value (Binary)** | **01001000** | **00000011** |
| **Final Value** | **0000001101001000 = 840** | |

After two separate additions and storing the final result into one register, we use command or “rrcf” to rotate to the right for both our high and low registers. This causes our result to be divided by two. We perform this process twice as the output equations requires to divide by four.

**First division by 2 (Table):**

|  |  |  |
| --- | --- | --- |
| **Register** | **SUM2L** | **SUM2H** |
| **Value (Decimal)** | **164** | **1** |
| **Value (Binary)** | **10100100** | **00000001** |
| **Final Value** | **0000000110100100 = 420** | |

**Second division by 2 (Table):**

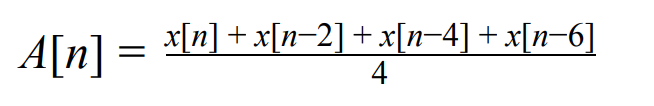
|  |  |  |
| --- | --- | --- |
| **Register** | **SUM2L** | **SUM2H** |
| **Value (Decimal)** | **210** | **0** |
| **Value (Binary)** | **11010010** | **00000000** |
| **Final Value** | **0000000011010010 = 210** | |

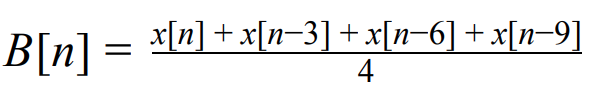
After completing two divisions by two our final result is stored in the registers of ResL and ResH and it’s equal to **210,** which is the correct value.

For other output of yi[n] for i = 1,2,3,4, the same code was used and it worked without any errors. The only modification that has to be done in the code is changing the counter value to either 2,4,6, or 8 and commenting out and uncommenting the proper SimpleTable lines in the code, therefore I believe there is no need for further explanation for y2[n], y3[n], and y4[n] since the process is the same as for y[n].

**IV. Tasks #3-4 (Laboratory)**

Both tasks #3 and task #4 are very similar to task#2 and therefore we used the same assembly language code as for task #2 to obtain the results for them. Below are the equations that we had to work for these tasks:





Now, in the previous task, we were basically adding four registers to each other and then dividing them by four. The difference is that, the registers in task #2 included values one after another, while for task #3, the odd values are skipped and for task #4, every 3rd value is added.

**This is the .asm for Tasks #3**

* **Y[n]**
* **T = 2 (Period =2)**
* ;;;;;;; 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
* ProgMem
* valueH
* xn1
* xn2
* xn3
* xn4
* xn5
* xn6
* xn7
* xn8
* xn9
* SUMH
* SUML
* SUM2H
* SUM2L
* TEMP
* TEMPH
* 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
* ; --------------------------------------------------------------
* ; Change value for counter depending
* ; on period of time series that you wish to use
* ;
* MOVLF 2,counter
* MOVLF upper SimpleTable,TBLPTRU
* MOVLF high SimpleTable,TBLPTRH
* MOVLF low SimpleTable,TBLPTRL
* ; --------------------------------------------------------------
* ; BEGIN WRTING CODE HERE
* label\_A
* movff xn8, xn9 ;xn9 = xn8 (xn9=x[n-9], xn8=x[n-8],...)
* movff xn7, xn8 ;xn8 = xn7
* movff xn6, xn7 ;xn7 = xn6
* movff xn5, xn6 ;xn6 = xn5
* movff xn4, xn5 ;xn5 = xn4
* movff xn3, xn4 ;xn4 = xn3
* movff xn2, xn3 ;xn3 = xn2
* movff xn1, xn2 ;xn2 = xn1
* movff value, xn1 ;xn1 = value
* TBLRD\*+
* movf TABLAT, W
* movwf value ; value = TABLAT (current value of the program memory)
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;First Summation ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* movf xn2, W ; changes wreg to xn2
* addwf value,W ; x[n]+x[n-2]=SUML
* movwf SUML ;result into SUML
* movf TEMPH,W
* addwfc valueH,w ;x[n]+x[n-2]=SUMH with bit carry
* movwf SUMH ;result into SUMH
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Second Summation;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* movf xn4,W ;changes wreg to xn4
* addwf xn6,W ; x[n-4]+x[n-6]=SUM2L
* movwf SUM2L ;result into SUM2L
* movf TEMPH,W
* addwfc valueH,w ;x[n-4]+x[n-6]=SUM2H with bit carry
* movwf SUM2H ;result into SUM2H
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Final Summation;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* movf SUML,W
* addwf SUM2L,W ; SUML+SUM2L=SUM2L
* movwf SUM2L ;result into SUM2L
* movf SUMH,W
* addwfc SUM2H,w ; SUMH+SUM2H=SUM2H
* movwf SUM2H ;result into SUM2H
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Division by 4;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* rrcf SUM2H, W ;rotate to the right for both high and low, for division by 2
* movwf SUM2H
* rrcf SUM2L, W
* movwf SUM2L
* rrcf SUM2H, W ;2nd rotate is performed for both high and low, this produces
* movwf SUM2H ;an overall division by 4
* rrcf SUM2L, W
* movwf SUM2L
* ;;;;;;;;;;;;;;;;;;;;;;Final Result;;;;;;;;;;;;;;;;;;;;;
* movff SUM2L, ResL ;copy final result into ResL and ResH
* movff SUM2H, ResH
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* ; 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',value
* MOVLF B'00000000',valueH
* MOVLF B'00000000',xn1
* MOVLF B'00000000',xn2
* MOVLF B'00000000',xn3
* MOVLF B'00000000',xn4
* MOVLF B'00000000',xn5
* MOVLF B'00000000',xn6
* MOVLF B'00000000',xn7
* MOVLF B'00000000',xn8
* MOVLF B'00000000',xn9
* MOVLF B'00000000',SUML
* MOVLF B'00000000',SUMH
* MOVLF B'00000000',SUM2L
* MOVLF B'00000000',SUM2H
* MOVLF B'00000000',TEMP
* MOVLF B'00000000',TEMPH
* MOVLF B'00000000',ResL
* MOVLF B'00000000',ResH
* return
* ;;;;;;; TIME SERIES DATA
* ;
* ; The following bytes are stored in program memory.
* ; Created by AC
* ;
* ; Choose your Periodic Sequence
* ;--------------------------------------------------------------
* ; time series X1
* SimpleTable ; ---> period 2
* db 180,240
* ;--------------------------------------------------------------
* ; time series X2
* ;SimpleTable ; ---> period 4
* ;db 180,240,200,244
* ;--------------------------------------------------------------
* ; time series X3
* ;SimpleTable ; ---> period 6
* ;db 180,240,200,244,216,236
* ;--------------------------------------------------------------
* ; time series X4
* ;SimpleTable ; ---> period 8
* ;db 180,240,200,244,216,236,160,176
* ; --------------------------------------------------------------
* end

Now this code is exactly the same as code for the task#2, except that more variables are being introduced and initialized. Instead of having xn1 to xn3 as for the task #2 this time we are using xn1 to xn9 because this is how many we need for both task #3 and task #4. We are not changing anything else, but also the variables/registers that are being added to each other. So again we are performing two summations and then adding them together into final summations and then diving the final summation twice by two and then storing it into the final registers of ResL and ResH. The process and method is exactly the same as for task#2, therefore I don’t think further explanation is needed.

**VI. Conclusion:**

This experiment was a great continuation on learning about moving average filters on the PIC using the assembly language. This time we were given a change to work with harder tasks and learn how to critically think on solving them. In one sentence, I can conclude that the experiment went successfully and without any complications providing us with clear and correct data results.