DSP Lab Submission, Madhav Lekkala (2020EEP2489)

26 November 2021 01:00

1. Develop a dspic executable program that will generate a series 2, 5, 8, 11,14,17,... and store in the memory.

Code :-

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;       Submission by   : Madhav Lekkala
;       Roll Number     : 2020EEP2489
;       Problem #       : Problem 1
;       Program des     : Generate series 2,5,8,11,14
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ __30F6010, 1

        .include "p30f6010.inc"
        .global __reset

        .text
    __reset:                            ; MAIN FUNCTION START
    MOV #0x1000, W4                     ; LOAD START ADDRESS TO W4
    MOV #0x0001, W0                     ; PROGRAM COUNTER (SELF)
    MOV #2, W1                          ; STARTING VALUE OF 2 STORED IN W1

    LOOP1:   DO #14, END1               ; RUN LOOP FOR 15 TIMES
    MOV W1, [W4]                        ; MOV UPDATED VALYE TO W4 ADDRESS
    INC2 W4, W4                         ; INCREMENT ADDRESS BY 0x02
    ADD #3, W1                          ; ADD 3 TO PREV VALUE
    END1:    INC W0,W0                  ; INCREMENT SELF PROGRAM COUNTER
```

Storage starts from 0x1000

Before Running code →

Address	00	02	04	06	08	0A	0C	0E	ASCII
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0000	0000	0000	0000	0000	0000	0000	0000	....
1010	0000	0000	0000	0000	0000	0000	0000	0000	....
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....
1090	0000	0000	0000	0000	0000	0000	0000	0000	....
10A0	0000	0000	0000	0000	0000	0000	0000	0000	....
10B0	0000	0000	0000	0000	0000	0000	0000	0000	....
10C0	0000	0000	0000	0000	0000	0000	0000	0000	....
10D0	0000	0000	0000	0000	0000	0000	0000	0000	....
10E0	0000	0000	0000	0000	0000	0000	0000	0000	....
10F0	0000	0000	0000	0000	0000	0000	0000	0000	....
1100	0000	0000	0000	0000	0000	0000	0000	0000	....
1110	0000	0000	0000	0000	0000	0000	0000	0000	....

After Running code

Address	00	02	04	06	08	0A	0C	0E	ASCII
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0002	0005	0008	000B	000E	0011	0014	0017	....
1010	001A	001D	0020	0023	0026	0029	002C	0000	.... .#. &.).,...
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....
1090	0000	0000	0000	0000	0000	0000	0000	0000	....
10A0	0000	0000	0000	0000	0000	0000	0000	0000	....
10B0	0000	0000	0000	0000	0000	0000	0000	0000	....
10C0	0000	0000	0000	0000	0000	0000	0000	0000	....
10D0	0000	0000	0000	0000	0000	0000	0000	0000	....
10E0	0000	0000	0000	0000	0000	0000	0000	0000	....
10F0	0000	0000	0000	0000	0000	0000	0000	0000	....
1100	0000	0000	0000	0000	0000	0000	0000	0000	....
1110	0000	0000	0000	0000	0000	0000	0000	0000	....
1120	0000	0000	0000	0000	0000	0000	0000	0000	....
1130	0000	0000	0000	0000	0000	0000	0000	0000	....
1140	0000	0000	0000	0000	0000	0000	0000	0000	

The series is stored accordingly in Hex

Working Registers

Before

Address	Name	Hex	Decimal	Binary	Char
0000	WREG0	0x0000	0	00000000 00000000	'..'
0002	WREG1	0x0000	0	00000000 00000000	'..'
0004	WREG2	0x0000	0	00000000 00000000	'..'
0006	WREG3	0x0000	0	00000000 00000000	'..'
0008	WREG4	0x0000	0	00000000 00000000	'..'
000A	WREG5	0x0000	0	00000000 00000000	'..'
000C	WREG6	0x0000	0	00000000 00000000	'..'
000E	WREG7	0x0000	0	00000000 00000000	'..'
0010	WREG8	0x0000	0	00000000 00000000	'..'
0012	WREG9	0x0000	0	00000000 00000000	'..'
0014	WREG10	0x0000	0	00000000 00000000	'..'
0016	WREG11	0x0000	0	00000000 00000000	'..'
0018	WREG12	0x0000	0	00000000 00000000	'..'
001A	WREG13	0x0000	0	00000000 00000000	'..'
001C	WREG14	0x0000	0	00000000 00000000	'..'
001E	WREG15	0x0000	0	00000000 00000000	'..'
0020	SPLIM	0x0000	0	00000000 00000000	'..'
0022	ACCA	0x0000000000	0	00000000 00000000 00000000 00000000 00000000	'.....'
0022	ACCAL	0x0000	0	00000000 00000000	'..'
0024	ACCAH	0x0000	0	00000000 00000000	'..'
0026	ACCAU	0x0000	0	00000000 00000000	'..'
0028	ACCB	0x0000000000	0	00000000 00000000 00000000 00000000 00000000	'.....'
0028	ACCBL	0x0000	0	00000000 00000000	'..'

After

Address	Name	Hex	Decimal	Binary	Char
0000	WREG0	0x0010	16	00000000 00010000	'..'
0002	WREG1	0x002F	47	00000000 00101111	'./'
0004	WREG2	0x0000	0	00000000 00000000	'..'
0006	WREG3	0x0000	0	00000000 00000000	'..'
0008	WREG4	0x101E	4126	00010000 00011110	'..'
000A	WREG5	0x0000	0	00000000 00000000	'..'
000C	WREG6	0x0000	0	00000000 00000000	'..'
000E	WREG7	0x0000	0	00000000 00000000	'..'
0010	WREG8	0x0000	0	00000000 00000000	'..'
0012	WREG9	0x0000	0	00000000 00000000	'..'
0014	WREG10	0x0000	0	00000000 00000000	'..'
0016	WREG11	0x0000	0	00000000 00000000	'..'
0018	WREG12	0x0000	0	00000000 00000000	'..'
001A	WREG13	0x0000	0	00000000 00000000	'..'
001C	WREG14	0x0000	0	00000000 00000000	'..'

2. Develop a dspic executable program that will sort the given series in ascending/descending order. (The series is:1, 34, 5,21,1, 2, 13, 3,55,89,8)

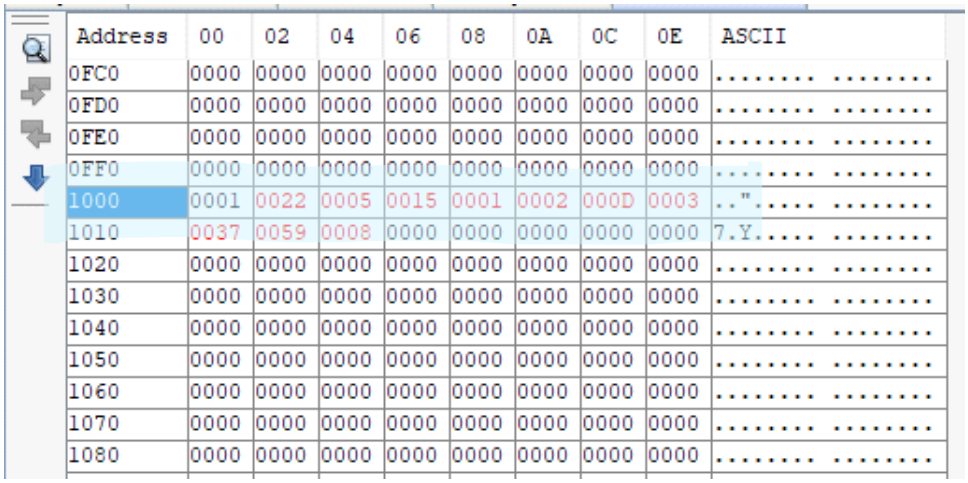
Code

```
.equ __30F6010, 1

    .include "p30f6010.inc"
    .global __reset

    .text
__reset:
MOV #1, W0
MOV W0, 0x1000
MOV #34, W0
MOV W0, 0x1002
MOV #5, W0
MOV W0, 0x1004
MOV #21, W0
MOV W0, 0x1006
MOV #1, W0
MOV W0, 0x1008
MOV #2, W0,
MOV W0, 0x100A
MOV #13, W0
MOV W0, 0x100C
MOV #3, W0
MOV W0, 0x100E
MOV #55, W0
MOV W0, 0x1010
MOV #89, W0
MOV W0, 0x1012
MOV #8, W0
MOV W0, 0x1014
```

storing the series from 0x1000

```
    MOV #0x0000, W0          ; PROGRAM COUNTER FOR INNER LOOP
    MOV #0x0000, W5          ; PROGRAM COUNTER FOR OUTER LOOP

    LOOP4: DO #9, END4       ; OUTER LOOP FOR BUBBLE SORT
    MOV #0x1000, W1          ; STARTING ADDRESS TO W1
    MOV #0x1002, W2          ; STARTING ADDRESS TO W2
    LOOP1:  DO #9, END1      ; INEER LOOP FOR IMMEDIATE SORT
        MOV [W1], W3         ; TEMP STORAGE OF VALUE 1 FOR CP
        MOV [W2], W4         ; TEMP STORAGE OF VALUE 2 FOR CP
        CP.B W3, W4          ; CP SETS CARRY IF W3>W4
        BRA C, LOOP2         ; BRANCH TO LOOP2 IF CARRY IS SET
        BRA LOOP3            ; BRANCH TO LOOP3 IS IT ISNT

        LOOP2:  MOV W3, [W2]     ; SWAPPING NUMBERS IF W3>W4
        MOV W4, [W1]             ; SWAPPING NUMBERS IF W3>W4
        BRA LOOP3                ; BRANCH TO LOOP3 (UNCONDITIONAL)

        LOOP3: INC2 W1, W1       ; INCREMENT ADDRESS 1 FOR NEXT SWEEP
        INC2 W2, W2              ; INCREMENT ADDRESS 2 FOR NEXT SWEEP
    END1: INC W0,W0              ; INNER LOOP PC INCREMENT
    END4: INC W5, W5             ; OUTER LOOP PC INCREMENT
```

Bubble sort Code



Address	00	02	04	06	08	0A	0C	0E	ASCII
0FC0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FD0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0001	0022	0005	0015	0001	0002	000D	0003	.."..... .....
1010	0037	0059	0008	0000	0000	0000	0000	0000	7.Y..... .....
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....

Before

Address	00	02	04	06	08	0A	0C	0E	ASCII
0FC0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FD0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0001	0001	0002	0003	0005	0008	000D	0015	....
1010	0022	0037	0059	0000	0000	0000	0000	0000	".7.Y... .....
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....
1090	0000	0000	0000	0000	0000	0000	0000	0000	....

After

Sorted in Ascending order

If you want to store in Descending order →

Change line 43 to → CP.B W4, W3

3. Develop a dspic executable program that will identify the minimum/maximum values in the given series and stores in memory location. (The series is:1, 34, 5,21,1, 2, 13, 3,55,89,8)

We can use problem 2 code here

Code:

```
31
32      MOV #0x0000, W0          ; PROGRAM COUNTER FOR INNER LOOP
33      MOV #0x0000, W5          ; PROGRAM COUNTER FOR OUTER LOOP
34
35      MOV #0x1000, W1          ; STARTING ADDRESS TO W1
36      MOV #0x1002, W2          ; STARTING ADDRESS TO W2
37      LOOP1:  DO #9, END1      ; INEER LOOP FOR IMMEDIATE SORT
38          MOV [W1], W3         ; TEMP STORAGE OF VALUE 1 FOR CP
39          MOV [W2], W4         ; TEMP STORAGE OF VALUE 2 FOR CP
40          CP.B W3, W4          ; CP SETS CARRY IF W3>W4
41          BRA C, LOOP2         ; BRANCH TO LOOP2 IF CARRY IS SET
42          BRA LOOP3            ; BRANCH TO LOOP3 IS IT ISNT
43
44          LOOP2:  MOV W3, [W2]     ; SWAPPING NUMBERS IF W3>W4
45          MOV W4, [W1]             ; SWAPPING NUMBERS IF W3>W4
46          BRA LOOP3                ; BRANCH TO LOOP3 (UNCONDITIONAL)
47
48          LOOP3: INC2 W1, W1       ; INCREMENT ADDRESS 1 FOR NEXT SWEEP
49          INC2 W2, W2              ; INCREMENT ADDRESS 2 FOR NEXT SWEEP
50      END1: INC W0,W0              ; INNER LOOP PC INCREMENT
51
52      ; MAXIMUM VALUE IS STORED IN 0x1014 address (last value of array)
53
54
```

Before

Address	00	02	04	06	08	0A	0C	0E	ASCII
0FC0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FD0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0001	0022	0005	0015	0001	0002	000D	0003	.."..... .....
1010	0037	0059	0008	0000	0000	0000	0000	0000	7.Y..... .....
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....

After 1 bubble sort

Address	00	02	04	06	08	0A	0C	0E	ASCII
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0001	0005	0015	0001	0002	000D	0003	0022	...."
1010	0037	0008	0059	0000	0000	0000	0000	0000	7...Y... .....
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....

Maximum value stred @ 0x1014

4. Develop a dspic executable program that will add two given (3x3) matrices and the result stores in a specified memory location.

Code

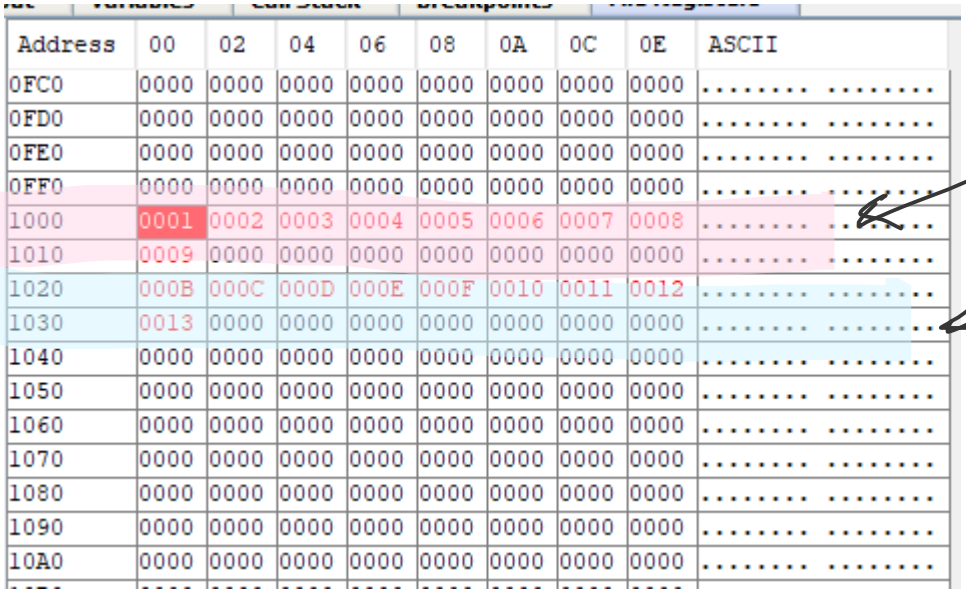
```
    .equ __30F6010, 1

        .include "p30f6010.inc"
        .global __reset

        .text
    __reset:

    ; MATRIX A
    MOV #1, W0
    MOV W0, 0x1000
    MOV #2, W0
    MOV W0, 0x1002
    MOV #3, W0
    MOV W0, 0x1004
    MOV #4, W0
    MOV W0, 0x1006
    MOV #5, W0
    MOV W0, 0x1008
    MOV #6, W0,
    MOV W0, 0x100A
    MOV #7, W0
    MOV W0, 0x100C
    MOV #8, W0
    MOV W0, 0x100E
    MOV #9, W0
    MOV W0, 0x1010

    ; MATRIX B
    MOV #11, W0
    MOV W0, 0x1020
    MOV #12, W0
    MOV W0, 0x1022
    MOV #13, W0
    MOV W0, 0x1024
    MOV #14, W0
    MOV W0, 0x1026
    MOV #15, W0
    MOV W0, 0x1028
    MOV #16, W0,
    MOV W0, 0x102A
    MOV #17, W0
    MOV W0, 0x102C
    MOV #18, W0
    MOV W0, 0x102E
    MOV #19, W0
    MOV W0, 0x1030

    MOV #0x0000, W0         ; PROGRAM COUNTER
    MOV #0x1000, W1         ; MATRIX A START ADDRESS
    MOV #0x1020, W2         ; MATRIX B START ADDRESS
    MOV #0x1040, W3         ; MATRIX C START ADDRESS

    LOOP1: DO #8, END1      ; LOOP FOR 9 TIMES (SIZE OF MATRIX)
    MOV [W1], W4            ; TEMP STORE VALUE IN W4
    MOV [W2], W5            ; TEMP STORE VALUE IN W5
    ADD W4, W5, [W3]        ; ADD W4, W5 STORE IN [W3]
    INC2 W1, W1             ; INCREMENT ADDRESS OF W1
    INC2 W2, W2             ; INCREMENT ADDRESS OF W2
    INC2 W3, W3             ; INCREMENT ADDRESS OF W3
    END1: INC W0, W0        ; PROGRAM COUNTER INCREMENT
```

Matrix A → stored from 0x1000

Matrix B → stored from 0x1020

Matrix C = A+B → stored from 0x1040

Before

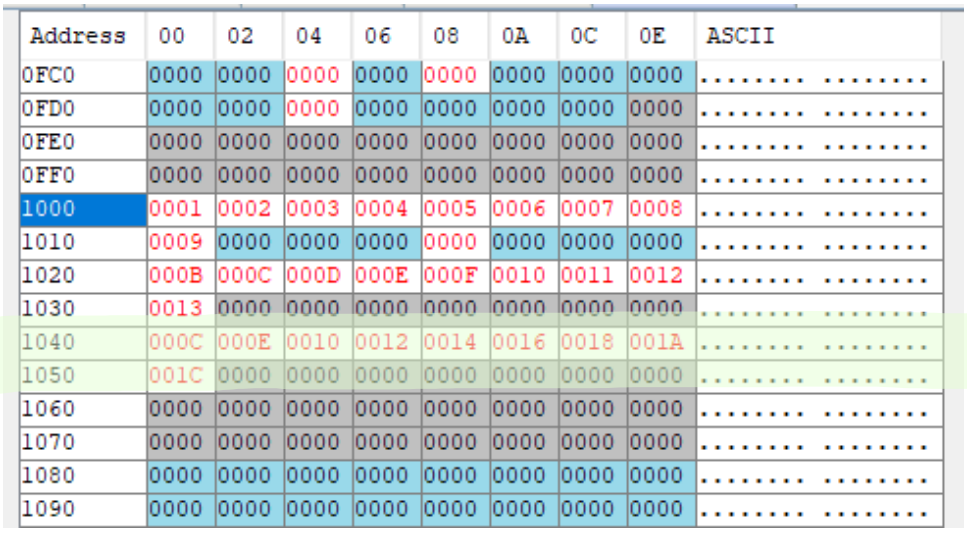


Address	00	02	04	06	08	0A	0C	0E	ASCII
0FC0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FD0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0001	0002	0003	0004	0005	0006	0007	0008	....
1010	0009	0000	0000	0000	0000	0000	0000	0000	....
1020	000B	000C	000D	000E	000F	0010	0011	0012	....
1030	0013	0000	0000	0000	0000	0000	0000	0000	....
1040	0000	0000	0000	0000	0000	0000	0000	0000	....
1050	0000	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....
1090	0000	0000	0000	0000	0000	0000	0000	0000	....
10A0	0000	0000	0000	0000	0000	0000	0000	0000	....

Matrix A

Matrix B

After



Address	00	02	04	06	08	0A	0C	0E	ASCII
0FC0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FD0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FE0	0000	0000	0000	0000	0000	0000	0000	0000	....
0FF0	0000	0000	0000	0000	0000	0000	0000	0000	....
1000	0001	0002	0003	0004	0005	0006	0007	0008	....
1010	0009	0000	0000	0000	0000	0000	0000	0000	....
1020	000B	000C	000D	000E	000F	0010	0011	0012	....
1030	0013	0000	0000	0000	0000	0000	0000	0000	....
1040	000C	000E	0010	0012	0014	0016	0018	001A	....
1050	001C	0000	0000	0000	0000	0000	0000	0000	....
1060	0000	0000	0000	0000	0000	0000	0000	0000	....
1070	0000	0000	0000	0000	0000	0000	0000	0000	....
1080	0000	0000	0000	0000	0000	0000	0000	0000	....
1090	0000	0000	0000	0000	0000	0000	0000	0000	....

Matrix C

5. Develop a dspic executable program (i) that will generate a FIBONACCI series, (ii) average of the above series and stores in the memory.

code

```
1   .equ __30F6010, 1
2
3       .include "p30f6010.inc"
4       .global __reset
5
6       .text
7   __reset:
8   ; KEPT W0, W1 RESERVED BECAUSE DIVISION IS NEEDED
9   MOV #0x0000, W3         ; STARTING VALUE 0
10  MOV #0x0001, W4         ; STARTING VALUE 1
11  MOV #0x1004, W2         ; MEMORY ADDRESS
12
13  MOV W3, 0x1000          ; STORING 1ST VALUE
14  MOV W4, 0x1002          ; STORING 2ND VALUE
15
16
17  LOOP1: DO #12, END1     ; FIRST 15 FIBONACCI NUMBERS
18  ADD W3,W4,W5            ; W3 = W1+W2
19  MOV W5, [W2]            ; W1=0, W2=1, W3=1 AND SO ON
20  INC2 W2, W2             ; INCREMENT ADDRESS
21  MOV W4, W3              ; INCREMENT ADDRESS
22  END1:   MOV W5, W4      ; INCREMENT PROGRAM COUNTER
23  ; used till W5
24
25  ; FOR AVERAGE CALCULATION
26  MOV #0x1000, W6         ; IMMEDIATE SUM
27  MOV #0x0000, W7         ; MIDDLE SUM
28  MOV #0x0000, W8         ; FINAL SUM
29  LOOP2: DO #14, END2     ; LOOP 15 TIMES
30  ADD W7, [W6], W8        ; W8 = W7+[W6]
31  MOV W8, W7              ; UPDATE VALUE
32  END2:   INC2 W6, W6     ; INCREMENT
33  ; used till W8
34
35  MOV #15, W9             ; DENOMINATOR
36  REPEAT #17              ; REQUIRED INSTRUCTION
37  DIV.U W8, W9            ; DIVISON
38
```

1000	0000	0001	0001	0002	0003	0005	0008	000D	....
1010	0015	0022	0037	0059	0090	00E9	0179	0000	..".7.Y. ....y...
1020	0000	0000	0000	0000	0000	0000	0000	0000	....
1030	0000	0000	0000	0000	0000	0000	0000	0000	....

Fibonacci series from 0x1000 in Hex

Address	Symbol	Hex	Decimal	Binary	Char
0000	WREG0	0x0041	65	00000000 01000001	'.A'
0002	WREG1	0x000B	11	00000000 00001011	'..'
0004	WREG2	0x101E	4126	00010000 00011110	'..'
0006	WREG3	0x00E9	233	00000000 11101001	'.é'
0008	WREG4	0x0179	377	00000001 01111001	'.y'
000A	WREG5	0x0179	377	00000001 01111001	'.y'
000C	WREG6	0x101E	4126	00010000 00011110	'..'
000E	WREG7	0x03DA	986	00000011 11011010	'.Ú'
0010	WREG8	0x03DA	986	00000011 11011010	'.Ú'
0012	WREG9	0x000F	15	00000000 00001111	'..'
0014	WREG10	0x0000	0	00000000 00000000	'..'
0016	WREG11	0x0000	0	00000000 00000000	'..'
0018	WREG12	0x0000	0	00000000 00000000	'..'
001A	WREG13	0x0000	0	00000000 00000000	'..'

num = 65

den = 11

= 65 · 75 (dec)

Sum of 1st 15 Fibonacci = 986 ⇒ Average = $\frac{986}{15}$

= 65.73

6. Develop a dspic executable program to generate 100 kHz/50 kHz PWM signal using "I/O pins".

Code

```
// FOSC
#pragma config FPR = XTL                // Primary Oscillator Mode (XTL)
#pragma config FOS = FRC                // Oscillator Source (Internal Fast RC)
#pragma config FCKSMEN = CSW_FSCM_OFF   // Clock Switching and Monitor (Sw Disabled, Mon Disabled)

// FWDT
#pragma config FWPSB = WDTPSB_16        // WDT Prescaler B (1:16)
#pragma config FWPSA = WDTPSA_512       // WDT Prescaler A (1:512)
#pragma config WDT = WDT_OFF            // Watchdog Timer (Disabled)

// FBORPOR
#pragma config FPWRT = PWRT_64          // POR Timer Value (64ms)
#pragma config BODENV = BORV20          // Brown Out Voltage (Reserved)
#pragma config BOREN = PBOR_ON          // PBOR Enable (Enabled)
#pragma config LPOL = PWMxL_ACT_HIGH    // Low-side PWM Output Polarity (Active High)
#pragma config HPOL = PWMxH_ACT_HIGH    // High-side PWM Output Polarity (Active High)
#pragma config PWMPIN = RST_IOPIN       // PWM Output Pin Reset (Control with PORT/TRIS regs)
#pragma config MCLRE = MCLR_DIS         // Master Clear Enable (Disabled)

// FGS
#pragma config GWRP = GWRP_OFF          // General Code Segment Write Protect (Disabled)
#pragma config GCP = CODE_PROT_OFF      // General Segment Code Protection (Disabled)

// FICD
#pragma config ICS = ICS_PGD            // Comm Channel Select (Use PGC/EMUC and PGD/EMUD)
```

Configuration Bits

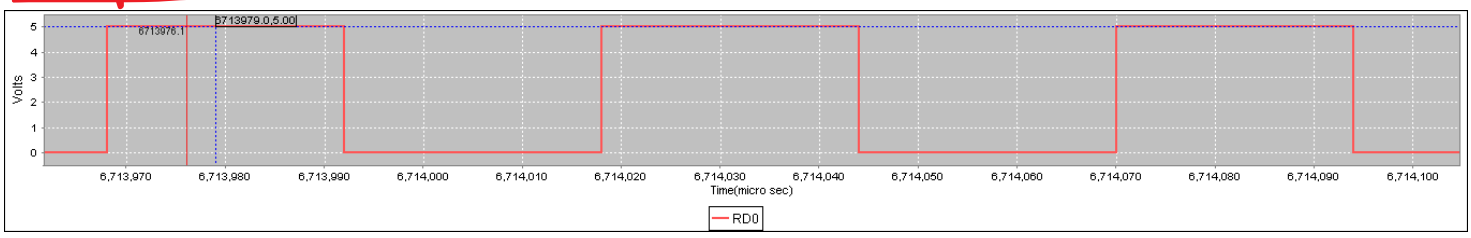
```
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
#include <libpic30.h>

int main()
{
    // Configure all four port D pins (RD0, RD1, RD2, RD3)
    // as digital outputs
    TRISD = 0b1111111111110000;

    // Set OC channel 1 pulse start and stop times
    OC1R = 0;
    OC1RS = 25;

    // Set OC channel 2 pulse start and stop times
    OC2R = 10;
    OC2RS = 35;

    // Set output compare mode for continuous pulses
    OC1CONbits.OCM = 0b101;
    OC2CONbits.OCM = 0b101;

    // Configure timer 2 (default timer for output compare)
    PR2 = 50; // 0.1ms period
    T2CONbits.TON = 1; // Enable timer 2

    while(1)
    {
        // endless loop
    }

    return 0;
}
```

] Ch 1 Duty

] Ch 2 (not used here)

] 50kHz Freq

Output ←5



@ PIN RD0 (I/O Pin)

7. Develop a dspic executable program to generate 100 kHz/50 kHz PWM using PWM-channels of the processor.

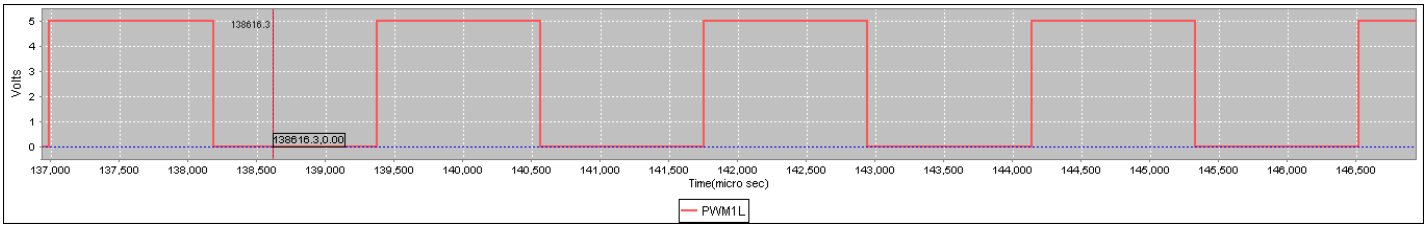
Code

```
.equ __30F6010, 1

        .include "p30f6010.inc"
        .global __reset
;.....................................................................
;Configuration bits:
;.....................................................................


        config __FWDT, WDT_OFF              ;Turn off Watchdog Timer

        config __FBORPOR, RST_IOPIN

        .text
    __reset:

```

```
    mov #0x0400, w0         ; PWM module is disabled, continue operation in
    mov w0, PTCON           ; Idle mode, special event interrupt disabled,
                            ; immediate period updates enabled, no external
                            ; synchronization
    ; Set the PWM Period
    mov #0x094D, w0         ; Select period to be approximately 2.5?s
    mov w0, PTPER           ; PLL Frequency is ~480MHz. This equates to a
                            ; clocke period of 2.1ns. The PWM period and
                            ; duty cycle registers are triggered on both +ve
                            ; and -ve edges of the PLL clock. Therefore,
                            ; one count of the PTPER and PDCx registers
                            ; equals 1.05ns.
                            ; So, to achieve a PWM period of 2.5?s, we
                            ; choose PTPER = 0x094D
    ; Select individual Duty Cycle Control
    mov #0x0001, w0         ; Fault interrupt disabled, Current Limit
    mov w0, PWMCON1         ; interrupt disabled, trigger interrupt,
                            ; disabled, Primary time base provides timing,
                            ; DC1 provides duty cycle information, positive
                            ; dead time applied, no external PWM reset,
                            ; Enable immediate duty cycle updates
    ; Duty Cycle Setting
    mov #0x094D, w0         ; To achieve a duty cycle of 50%, we choose
    mov w0, PDC1            ; the PDC1 value = 0.5*(PWM Period)
                            ; The ON time for the PWM = 1.25?s
                            ; The Duty Cycle Register will provide
                            ; positive duty cycle to the PWMxH outputs
                            ; when output polarities are active high
                            ; (see IOCON1 register)

    bset PTCON, #15         ; turn ON PWM module

```

Waveform

100kHz @ PWM1L



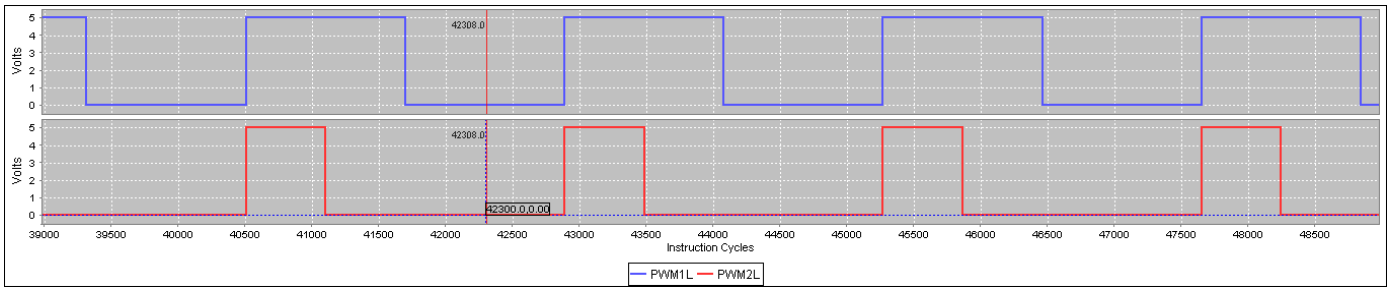
8. Develop a dspic executable program to generate 100 kHz PWM on the two PWM-channels of the processor with phase difference of (Ts/2, where f_s: 100 kHz).

Code

```
.equ __30F6010, 1

        .include "p30f6010.inc"
        .global __reset
;.....................................................................
;Configuration bits:
;.....................................................................


        config __FWDT, WDT_OFF              ;Turn off Watchdog Timer
        .text
    __reset:


    mov #0x0400, w0          ; PWM module is disabled, continue operation in
    mov w0, PTCON            ; Idle mode, special event interrupt disabled,
                             ; immediate period updates enabled, no external
                             ; synchronization
    ; Set the PWM Period
    mov #0x094D, w0          ; Select period to be approximately 2.5?s
    mov w0, PTPER            ; PLL Frequency is ~480MHz. This equates to a
                             ; clocke period of 2.1ns. The PWM period and
                             ; duty cycle registers are triggered on both +ve
                             ; and -ve edges of the PLL clock. Therefore,
                             ; one count of the PTPER and PDCx registers
                             ; equals 1.05ns.
                             ; So, to achieve a PWM period of 2.5?s, we
                             ; choose PTPER = 0x094D
    ; Select individual Duty Cycle Control
    mov #0xFFFF, w0          ; Fault interrupt disabled, Current Limit
    mov w0, PWMCON1          ; interrupt disabled, trigger interrupt,
                             ; disabled, Primary time base provides timing,
                             ; DC1 provides duty cycle information, positive
                             ; dead time applied, no external PWM reset,
                             ; Enable immediate duty cycle updates


    ; Duty Cycle Setting
    mov #0x094D, w0          ; To achieve a duty cycle of 50%, we choose
    mov w0, PDC1             ; the PDC1 value = 0.5*(PWM Period)
                             ; The ON time for the PWM = 1.25?s
                             ; The Duty Cycle Register will provide
                             ; positive duty cycle to the PWMxH outputs
                             ; when output polarities are active high
                             ; (see IOCON1 register)
    mov #0x04A6, w0          ; To achieve a duty cycle of 50%, we choose
    mov w0, PDC2             ; the PDC1 value = 0.5*(PWM Period)
                             ; The ON time for the PWM = 1.25?s
                             ; The Duty Cycle Register will provide
                             ; positive duty cycle to the PWMxH outputs
                             ; when output polarities are active high
                             ; (see IOCON1 register)

    bset PTCON, #15          ; turn ON PWM module
```

Waveform

2 PWM @ PWM 1L & 2L



9. Develop a dspic executable program to generate two PWM signals on the "I/O pins" with finite time delay between them.

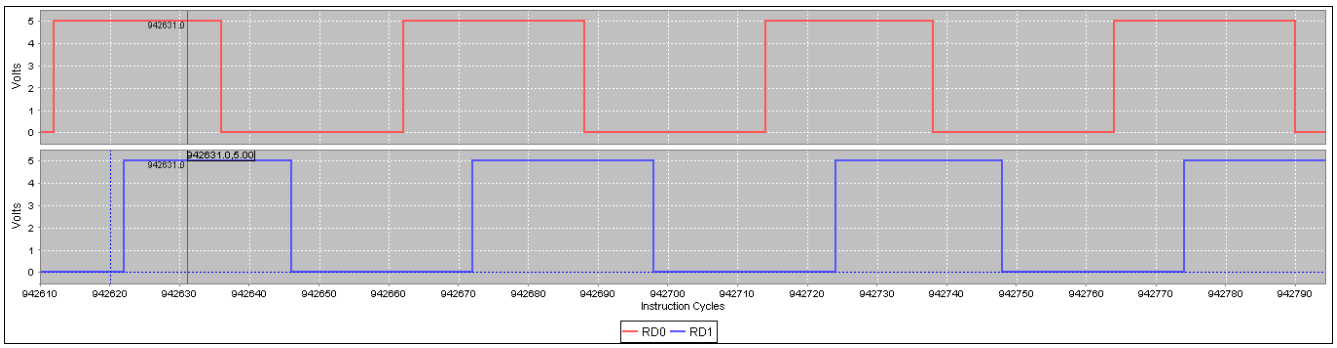
Code

```
// FOSC
#pragma config FPR = XTL                // Primary Oscillator Mode (XTL)
#pragma config FOS = FRC                // Oscillator Source (Internal Fast RC)
#pragma config FCKSMEN = CSW_FSCM_OFF   // Clock Switching and Monitor (Sw Disabled, Mon Disabled)

// FWDT
#pragma config FWPSB = WDTPSB_16        // WDT Prescaler B (1:16)
#pragma config FWPSA = WDTPSA_512       // WDT Prescaler A (1:512)
#pragma config WDT = WDT_OFF            // Watchdog Timer (Disabled)

// FBORPOR
#pragma config FPWRT = PWRT_64          // POR Timer Value (64ms)
#pragma config BODENV = BORV20          // Brown Out Voltage (Reserved)
#pragma config BOREN = PBOR_ON          // PBOR Enable (Enabled)
#pragma config LPOL = PWMxL_ACT_HIGH    // Low-side PWM Output Polarity (Active High)
#pragma config HPOL = PWMxH_ACT_HIGH    // High-side PWM Output Polarity (Active High)
#pragma config PWMPIN = RST_IOPIN       // PWM Output Pin Reset (Control with PORT/TRIS regs)
#pragma config MCLRE = MCLR_DIS         // Master Clear Enable (Disabled)

// FGS
#pragma config GWRP = GWRP_OFF          // General Code Segment Write Protect (Disabled)
#pragma config GCP = CODE_PROT_OFF      // General Segment Code Protection (Disabled)

// FICD
#pragma config ICS = ICS_PGD            // Comm Channel Select (Use PGC/EMUC and PGD/EMUD)
```

```
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
#include <libpic30.h>

int main()
{
    // Configure all four port D pins (RD0, RD1, RD2, RD3)
    // as digital outputs
    TRISD = 0b1111111111110000;

    // Set OC channel 1 pulse start and stop times
    OC1R = 0;
    OC1RS = 5;

    // Set OC channel 2 pulse start and stop times
    OC2R = 2;
    OC2RS = 7;

    // Set output compare mode for continuous pulses
    OC1CONbits.OCM = 0b101;
    OC2CONbits.OCM = 0b101;

    // Configure timer 2 (default timer for output compare)
    PR2 = 10; // 20us period
    T2CONbits.TON = 1; // Enable timer 2

    while(1)
    {
        // endless loop
    }

    return 0;
}
```

Waveform

Ts/2 phase shift @ RD0 RD1



10. Develop a dspic executable program to sense the external voltage signal using on-chip ADC and stores in the memory.

Code

```
// FOSC
#pragma config FPR = XTL                // Primary Oscillator Mode (XTL)
#pragma config FOS = FRC                // Oscillator Source (Internal Fast RC)
#pragma config FCKSMEN = CSW_FSCM_OFF   // Clock Switching and Monitor (Sw Disabled, Mon Disabled)

// FWDT
#pragma config FWPSB = WDTPSB_16        // WDT Prescaler B (1:16)
#pragma config FWPSA = WDTPSA_512       // WDT Prescaler A (1:512)
#pragma config WDT = WDT_OFF            // Watchdog Timer (Disabled)

// FBORPOR
#pragma config FPWRT = PWRT_64          // POR Timer Value (64ms)
#pragma config BODENV = BORV20          // Brown Out Voltage (Reserved)
#pragma config BOREN = PBOR_ON          // PBOR Enable (Enabled)
#pragma config LPOL = PWMxL_ACT_HIGH    // Low-side PWM Output Polarity (Active High)
#pragma config HPOL = PWMxH_ACT_HIGH    // High-side PWM Output Polarity (Active High)
#pragma config PWMPIN = RST_IOPIN       // PWM Output Pin Reset (Control with PORT/TRIS regs)
#pragma config MCLRE = MCLR_DIS         // Master Clear Enable (Disabled)

// FGS
#pragma config GWRP = GWRP_OFF          // General Code Segment Write Protect (Disabled)
#pragma config GCP = CODE_PROT_OFF      // General Segment Code Protection (Disabled)

// FICD
#pragma config ICS = ICS_PGD            // Comm Channel Select (Use PGC/EMUC and PGD/EMUD)
```

```
#include <xc.h>
#include <libpic30.h>

unsigned int read_analog_channel(int n);

int main()
{
    // Declare a variable for the step time
    // so that it can be changed easily
    int v;
    long step_time = 300000L;

    // Make RD0-3 digital outputs
    TRISD = 0b0000;

    // Configure analog inputs
    TRISB = 0x01FF;       // Port B all inputs
    ADPCFG = 0xFF00;      // Lowest 8 PORTB pins are analog inputs
    ADCON1 = 0;           // Manually clear SAMP to end sampling, start conversion
    ADCON2 = 0;           // Voltage reference from AVDD and AVSS
    ADCON3 = 0x0005;      // Manual Sample, ADCS=5 -> Tad = 3*Tcy = 0.1us
    ADCON1bits.ADON = 1; // Turn ADC ON

    // Cycle through the four windings to make
    // the stepper turn forwards
    while(1)
    {
        // Read the analog channel. The result is an
        // integer between 0 and 1023 inclusive.
        v = read_analog_channel(0);
```

Reads input @ Port B

```
        // Now, update step time.
        // Because the value get too big for 16-bit ints,
        // the constant values are explicitly marked as
        // long values so that the calculation is carried
        // out using 32-bit ints.
        step_time = 150000L + 200L * v;

        // Cycle through the four stepper windings
        LATD = 0b1000; __delay32(step_time);
        LATD = 0b0100; __delay32(step_time);
        LATD = 0b0010; __delay32(step_time);
        LATD = 0b0001; __delay32(step_time);
    }

    return 0;
}

// This function reads a single sample from the specified
// analog input. It should take less than 2.5us if the chip
// is running at about 30 MIPS.
unsigned int read_analog_channel(int channel)
{
    ADCHS = channel;          // Select the requested channel
    ADCON1bits.SAMP = 1;      // start sampling
    __delay32(30);            // 1us delay @ 30 MIPS
    ADCON1bits.SAMP = 0;      // start Converting
    while (!ADCON1bits.DONE); // Should take 12 * Tad = 1.2us
    return ADCBUF0;
}
```

ADC Read