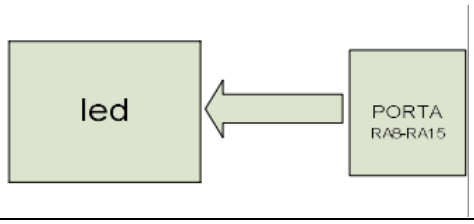
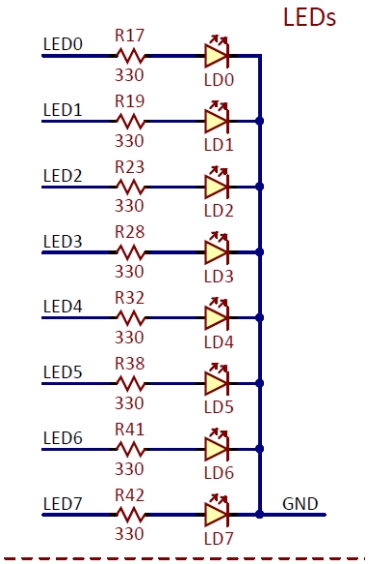
8-Leds

Description	PIC32 pin	Schematic name	Label
Led 0	TMS/CTED1/**RA0**	LED0	**LD0**
Led 1	TCK/CTED2/**RA1**	LED1	**LD1**
Led 2	SCL2/**RA2**	LED2	**LD2**
Led 3	SDA2/**RA3**	LED3	**LD3**
Led 4	TDI/CTED9/**RA4**	LED4	**LD4**
Led 5	TDO/**RA5**	LED5	**LD5**
Led 6	TRCLK/**RA6**	LED6	**LD6**
Led 7	TRD3/CTED8/**RA7**	LED7	**LD7**





Ports Macro's:

#define **LD0** PORTAbits.**RA0**
#define **LD1** PORTAbits.**RA1**
#define **LD2** PORTAbits.**RA2**
#define **LD3** PORTAbits.**RA3**
#define **LD4** PORTAbits.**RA4**
#define **LD5** PORTAbits.**RA5**
#define **LD6** PORTAbits.**RA6**
#define **LD7** PORTAbits.**RA7**

```
#include <xc.h>
#pragma config JTAGEN = OFF
#pragma config FWDTEN = OFF
#pragma config FNOSC =FRCPLL
#pragma config FSOSCEN =     OFF
#pragma config POSCMOD =     EC
#pragma config OSCIOFNC =    ON
#pragma config FPBDIV =      DIV_1
#pragma config FPLLIDIV =    DIV_2
#pragma config FPLLMUL =
MUL_20
#pragma config FPLLODIV =    DIV_1

void main(){
    //init starts *********
     TRISA &= 0xff00;
     //init ends ***********
     PORTA = 0x55;
}
```

TRIS

1 = input	**0 = output**
הגדרת הפורט ככניסה לבקר. (קוראים מהפורט – הפורט משתנה בעקבות שינוי חיצוני: לחיצת/עזיבת מקש וכו..	הגדרת הפורט כיציאה מהבקר. (כותבים ליציאה – הפורט משתנה בעקבות שינוי בקוד: כיבוי והפעלת לדים וכו..

הגדרת PORTA כיציאה (RA0-RA7). //

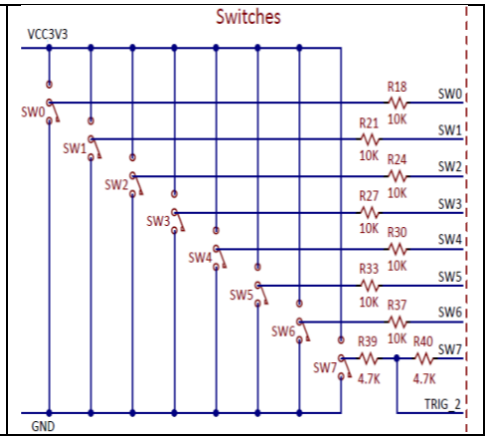
הדלקת הלדים RA**i** (**i** מספר זוגי: 0,2,4,6). //
כי 0x55=0b10101010

Switches

Description	PIC32 pin	Schem' name	Label
Switch 0	RPF3/**RF3**	SW0	SW0
Switch 1	RPF5/PMA8/**RF5**	SW1	SW1
Switch 2	RPF4/PMA9/**RF4**	SW2	SW2
Switch 3	RPD15/**RD15**	SW3	SW3
Switch 4	RPD14/**RD14**	SW4	SW4
Switch 5	**AN11**/PMA12/**RB11**	SW5	SW5
Switch 6	CVREFOUT/**AN10**/RPB10/CTED11PMA13/**RB10**	SW6	SW6
Switch 7	**AN9**/RPB9/CTED4/**RB9**	SW7	SW7

Ports Macro's:

#define **SW0** PORTFbits.**RF3**
#define **SW1** PORTFbits.**RF5**
#define **SW2** PORTFbits.**RF4**
#define **SW3** PORTDbits.**RD15**
#define **SW4** PORTDbits.**RD14**
#define **SW5** PORTBbits.**RB11**
#define **SW6** PORTBbits.**RB10**
#define **SW7** PORTBbits.**RB9**




```c
#include <xc.h>
/*standard "pragma config's" in here… */
void main() {
  //init starts *********
  TRISFbits.TRISF3 = 1;
  TRISFbits.TRISF5 = 1;
  TRISFbits.TRISF4 = 1;
  TRISDbits.TRISD15 = 1;
  TRISDbits.TRISD14 = 1;
  TRISBbits.TRISB11 = 1;
  TRISBbits.TRISB10 = 1;
  TRISBbits.TRISB9 = 1;

  ANSELBbits.ANSB11 = 0;//SW5 an disable
  ANSELBbits.ANSB10 = 0;//SW6 an disable
  ANSELBbits.ANSB9 = 0; //SW7 an disable
  //init ends ***********

  while (1) {
    PORTAbits.RA0 = PORTFbits.RF3;
    PORTAbits.RA1 = PORTFbits.RF5;
    PORTAbits.RA2 = PORTFbits.RF4;
    PORTAbits.RA3 = PORTDbits.RD15;
    PORTAbits.RA4 = PORTDbits.RD14;
    PORTAbits.RA5 = PORTBbits.RB11;
    PORTAbits.RA6 = PORTBbits.RB10;
    PORTAbits.RA7 = PORTBbits.RB9;
  }
}
```

*מגדירים לכל פורט של switch את הTRIS להיות 1 כי הוא PORT כניסה.
* ע"י: TRISFbits.TRISF3 = 1

*פורטי כניסה מסויימים ניתן להגדיר כאנלוגיים ולא כדיגיטליים (0/1 כלומר דלוק\כבוי).
נגדיר את הפורטים ל**sw5,sw6,sw7** להיות דיגיטליים ע"י ביצוע disable לAnalog שלהם.

- נטרול ע"י: ANSELBbits.ANSB9 = **0**

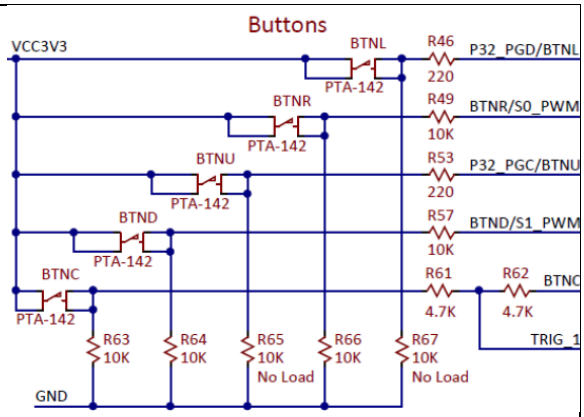
*כאן כל כפתור מדליק\מכבה את הלד המתאים לו (שנמצא פיזית מעליו).

User Buttons

Description	PIC32 pin	Pin Shared With	Schem' name	Label
Left	PGED1/**AN0**/RPB0/**RB0**	PGD	BTN**L**	BTN**L**
Right	**AN8**/RPB8/CTED10/**RB8**	S0_PWM	BTN**R**	BTN**R**
Center	RPF0/PMD11/**RF0**	TRIG_1	BTN**C**	BTN**C**
Up	PGEC1/**AN1**/RPB1/CTED12/**RB1**	PGC	BTN**U**	BTN**U**
Down	RPA15/**RA15**	S1_PWM	BTN**D**	BTN**D**

Ports Macro's:

```
#define tris_BTNL   TRISBbits.TRISB0
#define ansel_BTNL  ANSELBbits.ANSB0
#define prt_BTNL    PORTBbits.RB0

#define tris_BTNR   TRISBbits.TRISB8
#define ansel_BTNR  ANSELBbits.ANSB8
#define prt_BTNR    PORTBbits.RB8

#define tris_BTNC   TRISFbits.TRISF0
#define prt_BTNC    PORTFbits.RF0

#define tris_BTNU   TRISBbits.TRISB1
#define ansel_BTNU  ANSELBbits.ANSB1
#define prt_BTNU    PORTBbits.RB1

#define tris_BTND   TRISAbits.TRISA15
#define prt_BTND    PORTAbits.RA15
```



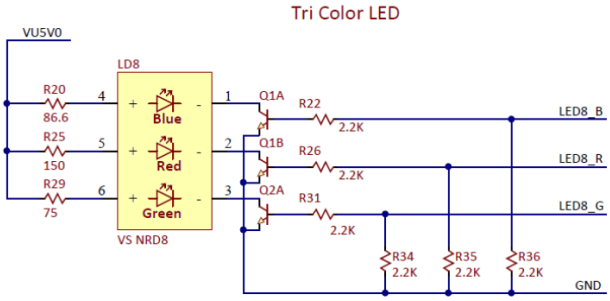
דוגמא: תוכנית שמדפיסה לLCD את התו המייצג של הכפתור שנלחץ.

```
#include <xc.h>
/*standard "pragma config's" in here… */

/*User Buttons defines and prototypes*/
#define tris_BTNL   TRISBbits.TRISB0
#define ansel_BTNL  ANSELBbits.ANSB0
#define prt_BTNL    PORTBbits.RB0

#define tris_BTNR   TRISBbits.TRISB8
#define ansel_BTNR  ANSELBbits.ANSB8
#define prt_BTNR    PORTBbits.RB8

#define tris_BTNC   TRISFbits.TRISF0
#define prt_BTNC    PORTFbits.RF0

#define tris_BTNU   TRISBbits.TRISB1
#define ansel_BTNU  ANSELBbits.ANSB1
#define prt_BTNU    PORTBbits.RB1

#define tris_BTND   TRISAbits.TRISA15
#define prt_BTND   PORTAbits.RA15

void delay();


/*LCD defines and prototypes*/
void busy(void);
void initLCD();
#define DISP_EXEC PORTDbits.RD4 = 1;PORTDbits.RD4 = 0;busy();


void main() {
  /*Init User Buttons*/
  tris_BTNU = 1;
  tris_BTNL = 1;
  tris_BTNC = 1;
  tris_BTNR = 1;
  tris_BTND = 1;

  ansel_BTNU = 0;
  ansel_BTNL = 0;
  ansel_BTNR = 0;

  /*LCD*/
  initLCD();
  PORTBbits.RB15 = 0; //command mode
  PORTE = 0x1;
  DISP_EXEC
  PORTBbits.RB15 = 1; //data mode
```

```
  char btn_name = '_';
  int counter = 0;
  while (1) {

        while (1) {
              if (prt_BTNL) {
                btn_name = 'L';
                break;
              }
              else if (prt_BTNR) {
                btn_name = 'R';
                break;
              }
              else if (prt_BTNC) {
                btn_name = 'C';
                break;
              }
              else if (prt_BTNU) {
                btn_name = 'U';
                break;
              }
              else if (prt_BTND) {
                btn_name = 'D';
                break;
              }
        }

        PORTE = btn_name;
        DISP_EXEC
        delay();

        counter++;
        if (counter == 16) {
          PORTBbits.RB15 = 0; //command mode
          PORTE = 0xc0; //write to second line
          DISP_EXEC
          PORTBbits.RB15 = 1; //data mode
        }
        if (counter == 32) {
          PORTBbits.RB15 = 0; //command mode
          PORTE = 0x1; //flush LCD and write in first line
          DISP_EXEC
          PORTBbits.RB15 = 1; //data mode
          counter = 0;
        }
    }
}

void delay() {
  int i;
  for (i = 0; i < 64000; i++); for (i = 0; i < 64000; i++);
  for (i = 0; i < 64000; i++); for (i = 0; i < 64000; i++);
  for (i = 0; i < 64000; i++);  for (i = 0; i < 64000; i++);
}
```

RGB Leds

PIC32 pin	Schem' name	Label
AN25/RPD2/**RD2**	LED8_R	**R**
RPD12/PMD12/**RD12**	LED8_G	**G**
AN26/RPD3/	LED8_B	**B**



Ports Macro's:

#define **RGB_R** PORTDbits.**RD2**
#define **RGB_G** PORTDbits.**RD12**
#define **RGB_B** PORTDbits.**RD3**

```c
#include <stdio.h>
/*standard "pragma config's" in here…*/

void main() {

  TRISFbits.TRISF3 = 1;
  TRISFbits.TRISF5 = 1;
  TRISFbits.TRISF4 = 1;

  //init starts *********
  TRISDbits.TRISD2 = 0;
  TRISDbits.TRISD12 = 0;
  TRISDbits.TRISD3 = 0;
  ANSELDbits.ANSD2 = 0;
  ANSELDbits.ANSD3 = 0;
  //init ends ***********

  while (1) {
    PORTDbits.RD2 = PORTFbits.RF3;
    PORTDbits.RD12 = PORTFbits.RF5;
    PORTDbits.RD3 = PORTFbits.RF4;
  }
}
```

sw0,sw1,sw2 פורטים **כניסה**.

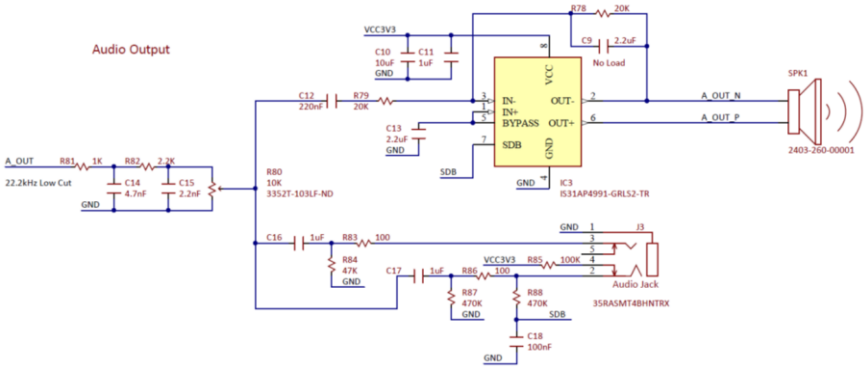
LED8_R, LED8_G, LED8_B פורטים **יציאה**.

LED8_R, LED8_B פורטים דיגיטליים.

*sw0 מקושר ל**led8_r**
*sw1 מדליק\מכבה **led8_g**
*sw2 מדליק\מכבה **led8_b**

Speaker

Name	PIC32 pin
A_OUT	**AN14**/RPB14/CTED5/PMA1/**RB14**



כדי שספיקר יפעל צריך לתת לתת מתח\להפסיק מתח כל 640 פעולות (זה מה שיוצר רטט ברכיב הספיקר מה שמוציא סאונד).

צפצוף אחיד:

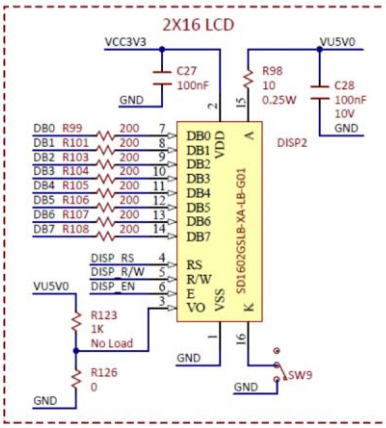
```
#include <stdio.h>
/*standard "pragma config's" in here… */

void main() {
    //init starts***********
    TRISBbits.TRISB14 = 0; //Speaker - as output
    ANSELBbits.ANSB14 = 0; //Speaker - disabled Analog
    //init ends*************

    int i=0;
    PORTBbits.RB14 = 0;

    while(1){
        if(i%640==0)
            PORTBbits.RB14 = !PORTBbits.RB14;
        i++;
    }
}
```

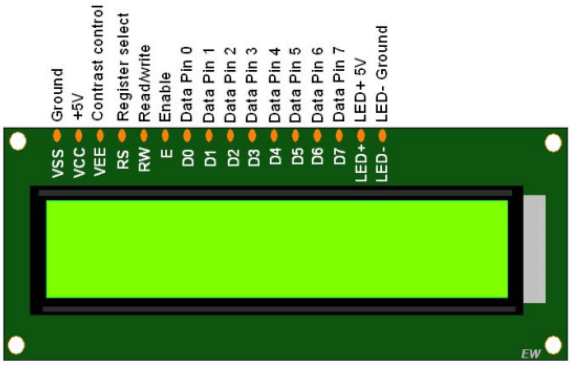
צפצוף לסירוגין:

```
#include <stdio.h>
/*standard "pragma config's" in here… */
void main() {
    //init starts***********
    TRISBbits.TRISB14 = 0; //Speaker - as output
    ANSELBbits.ANSB14 = 0; //Speaker - disabled analog
    //init ends*************

    int i;
    PORTBbits.RB14 = 0;
    while(1){
        for(i=0;i<640000;i++);
        for(i=0;i<640000;i++){
            if(i%640==0)
                PORTBbits.RB14 = !PORTBbits.RB14;
        }
    }
}
```

LCD

Name	PIN32 Pin	Description
DISP_RS	**AN15**/RPB15/OCFB/CTED6/PMA0/**RB15**	**0** for command, **1** for data (inside DB0-DB7)
DISP_RW	RPD5/PMRD/**RD5**	**1** for **R**ead **,0** for **W**rite (from/to DB0-DB7)
DISP_EN	RPD4/PMWR/**RD4**	**signal (1** and then **0)**
DB0	PMD0/**RE0**	**DB0-DB7 (PORTE)**
DB1	PMD1/**RE1**	
DB2	**AN20**/PMD2/**RE2**	
DB3	RPE3/CTPLS/PMD3/**RE3**	
DB4	**AN21**/PMD4/**RE4**	
DB5	**AN22**/RPE5/PMD5/**RE5**	
DB6	**AN23**/PMD6/**RE6**	
DB7	**AN27**/PMD7/**RE7**	





הסבר:

1. תחילה נבצע את הפקודות: (מתבצע בפונקציית **initLCD**)

```
TRISE &= 0xff00;
PORTDbits.RD5 = 0; // DISP_RW is Write
```

שמגדירה את DB0-DB7 כיציאות שאליהן אנחנו כותבים.

2. נבצע את הפקודה:

```
PORTBbits.RB15 = 0;
```

כדי להבהיר שהמידע שאנחנו **כותבים** לפורטים DB0-DB7 הוא פעולה שאנחנו רוצים שהמסך יבצע.

פעולות נפוצות:

0x1 = ניקוי מסך ותחילת כתיבה בשורה הראשונה של המסך.

0x80 = תחילת כתיבה בשורה הראשונה של המסך.

0xc0 = תחילת כתיבה בשורה השניה של המסך.

0x38 = <u>הגדרת ה-LCD</u> ל8 סיביות ופיקסל בגודל 8x5 (צריך לבצע 3 פעמים באיתחול המסך).

3. נבצע את הפקודה:

```
PORTBbits.RB15 = 1;
```

כדי להבהיר שהמידע שאנחנו כותבים לפורטים DB0-DB7 (PORTE) הוא תו שאנחנו רוצים שהמסך ידפיס.

4. נבצע את הפקודות:

```
//signal
PORTDbits.RD4=1;
PORTDbits.RD4=0;

busy();
```

כאשר אנו רוצים שהמסך יבצע את הפעולה שכתבנו לפורטי הדאטה או שאנו רוצים שהמסך ידפיס תו שכתבנו לפורטי הדאטה.

דוגמא – הדפסת:

Shahar
Hikri_

```
#include <xc.h>
#include <string.h>
/*standard "pragma config's" in here… */
void busy(void); void initLCD();

void main() {
    char* line1 = "Shahar";
    char* line2 = "Hikri";
    int i;
    initLCD();

    /*PRINT TO 1'ST LINE*/
    /*command mode*/
    PORTBbits.RB15 = 0;
    /*flush LCD and print to 1'st line command execute*/
    PORTE = 0x1;
    //execute command
    PORTDbits.RD4 = 1;
    PORTDbits.RD4 = 0;
    busy();

    /*data mode*/
    PORTBbits.RB15 = 1;
    /*print line1 = "Shahar"*/
    for(i=0;i<strlen(line1);i++){
        PORTE=line1[i];

        //print char
        PORTDbits.RD4 = 1;
        PORTDbits.RD4 = 0;
        busy();
    }
    /*PRINT TO 2'ND LINE*/
    /*command mode*/
    PORTBbits.RB15 = 0;
    /*flush LCD and print to 1'st line command execute*/
    PORTE = 0xc0;
    //execute command
    PORTDbits.RD4 = 1;
    PORTDbits.RD4 = 0;
    busy();
    /*data mode*/
    PORTBbits.RB15 = 1;
    /*print line2 = "Hikri"*/
    for(i=0;i<strlen(line2);i++){
        PORTE=line2[i];
        //print char
        PORTDbits.RD4 = 1;
        PORTDbits.RD4 = 0;
        busy();
    }
}
```

```
void busy(void) {
  char RD = PORTDbits.RD5;
  char RS = PORTBbits.RB15;
  int STATUS_TRISE = TRISE;

  PORTBbits.RB15 = 0; //Command mode

  PORTDbits.RD5 = 1; //Read mode (from DB0-DB7)
  //DB7 Tris set to input (BF register)
  TRISE |= 0x80;

  do {
    PORTDbits.RD4 = 1; //enable=1
    PORTDbits.RD4 = 0; //enable=0
  }
  while (PORTEbits.RE7); // (DB7) BF register

  PORTDbits.RD5 = RD;
  PORTBbits.RB15 = RS;
  TRISE = STATUS_TRISE;
}

void initLCD() {

  char control[] = {0x38,0x38,0x38,0xe,0x6,0x1};

  //init DISP_RS DISP_RW DISP_EN
  TRISBbits.TRISB15 = 0; // DISP_RS (RB15) as Output
  ANSELBbits.ANSB15 = 0; // DISP_RS (RB15) disable analog
  TRISDbits.TRISD5 = 0; // DISP_RW (RD5) set as an Output
  TRISDbits.TRISD4 = 0; // DISP_EN (RD4) set as an Output

  //init DB0-DB7
  TRISE &= 0xff00;
  ANSELEbits.ANSE2 = 0;
  ANSELEbits.ANSE4 = 0;
  ANSELEbits.ANSE5 = 0;
  ANSELEbits.ANSE6 = 0;
  ANSELEbits.ANSE7 = 0;

  PORTBbits.RB15 = 0; //rs=0 (Command)
  PORTDbits.RD5 = 0; //w=0 (Write into )

  int i;
  for (i = 0; i < 6; i++) {
    PORTE = control[i];

    //signal
    PORTDbits.RD4 = 1;
    PORTDbits.RD4 = 0;

    busy();
  }
}
```

<u>רגיסטר ה-BF</u> – לאחר שהמידע נקרא מdb0-db7 אז הdb7 הופך להיות ה(Busy Flag(bf כלומר כאשר הוא 1 המסך כרגע מעבד את המידע ומבצע את הפעולה\מדפיס תו. לכן רק שהBF יורד ל0 ניתן לעשות פעולה חדשה\להדפיס תו נוסף על המסך. (מופיע בפונקציה Busy).

הוספת תווים שלא קיימים

CGROM – (כמעט כל) תווי הASCII מאוכסנים בזיכרון זה.

CGRAM – זיכרון שמכיל תווים שאנחנו יוצרים. יצירת תו הינה מערך של char-ים שכל אחד מהם מייצג שורה בהתאמה, כל ביט דלוק בו הוא פיקסל דלוק וכל ביט כבוי הוא פיקסל כבוי.

- צורבים לCGRAM את התו בעזרת הפעולה **0x40**.
- ניתן לאחסן בCGRAM עד 8 תווים חדשים ו"קוד הASCII" שלהם בבקר יהיו מ0 עד 8 בהתאם לסדר צריבתם.

דוגמא לצריבת התווים 'ש','ת' והדפסת:

שST

```
#include <xc.h>
/*standard "pragma config's" in here… */

void busy(void);
void initLCD();
#define DISP_EXEC      PORTDbits.RD4 = 1;PORTDbits.RD4 = 0;busy();

void main() {
    initLCD();
   char  CG_shin[8] = {0b10101,
                       0b10101,
                       0b10101,
                       0b10101,
                       0b10101,
                       0b10101,
                       0b10101,
                       0b11111};

   char  CG_taf[8] = { 0b01111,
                       0b01001,
                       0b01001,
                       0b01001,
                       0b01001,
                       0b01001,
                       0b01001,
                       0b11001};
    char shin = 0;
    char taf = 1;

    PORTBbits.RB15 = 0;//COMMAND mode
    PORTE=0x40;         //Add char's to CDRAM
    DISP_EXEC

    PORTBbits.RB15 = 1;//DATA
    /*Write char's to CDRAM*/
    int i;
    for(i = 0;i < 8;i++){ //writing 'ש' to CGRAM
        PORTE=CG_shin[i];
        DISP_EXEC
    }
    for(i = 0;i < 8;i++){//writing 'ת' to CGRAM
        PORTE=CG_taf[i];
        DISP_EXEC
    }
```

```
    /*Print 'ПШST'*/
    PORTBbits.RB15 = 0; //COMMAND mode
    PORTE=0x80;         //Write in 1'st line command
    DISP_EXEC

    /*data mode*/
    PORTBbits.RB15 = 1;
    PORTE=taf;
    DISP_EXEC

    PORTE=shin;
    DISP_EXEC

    PORTE='T';
    DISP_EXEC

    PORTE='S';
    DISP_EXEC
}
```

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μs

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

מקלדת (Pull Up)

Pmod pin	Schematic Label	PIC32 pin	Description
PMODA_1	JA_1	RPC2/**RC2**	X0
PMODA_2	JA_2	RPC1/**RC1**	X1
PMODA_3	JA_3	RPC4/CTED7/**RC4**	X2
PMODA_4	JA_4	**AN16**/C1IND/RPG6/SCK2/PMA5/**RG6**	X3
PMODA_7	JA_7	RPC3/**RC3**	Y0
PMODA_8	JA_8	**AN17**/C1INC/RPG7/PMA4/**RG7**	Y1
PMODA_9	JA_9	**AN18**/C2IND/RPG8/PMA3/**RG8**	Y2
PMODA_10	JA_10	**AN19**/C2INC/RPG9/PMA2/**RG9**	Y3

X-ים – פורטים **יציאה** (אותם משנים בקוד - מכבים לשם הבדיקה ואז מדליקים חזרה).

Y-ים – פורטים **כניסה** (אותם בודקים בקוד – אם כבויים).

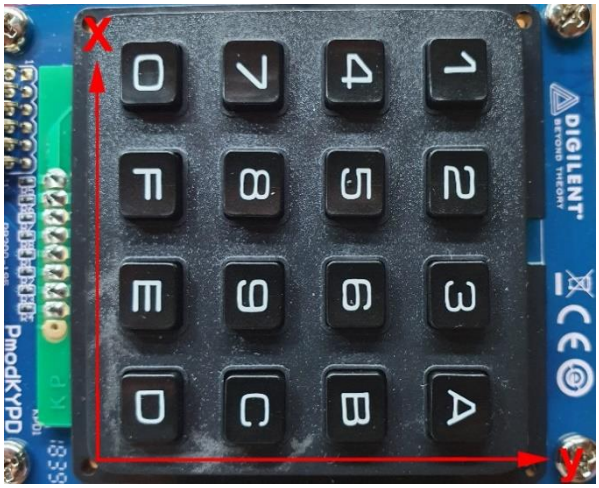
<u>בדיקה האם כפתור נלחץ:</u>

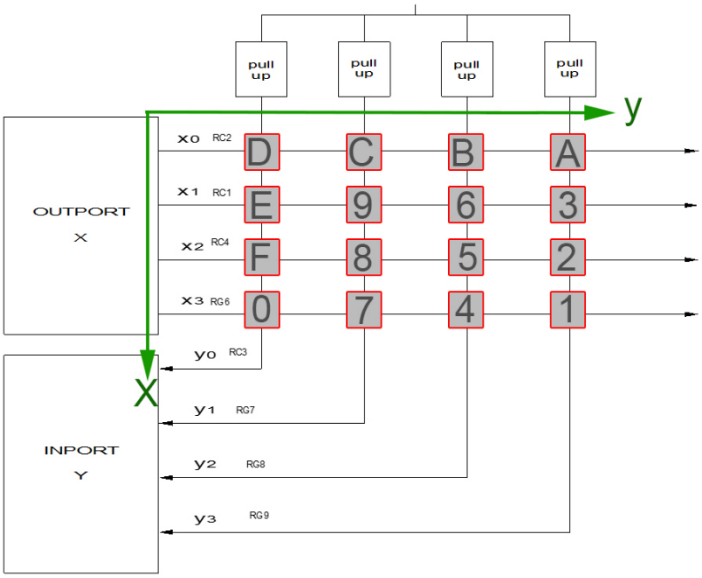
כאשר רוצים לבדוק האם הכפתור ב(X1,Y2) (הכפתור '6') נלחץ:

נכבה את X0.

נבדוק אם Y2 כבוי (אם כן אז הכפתור ב(X1,Y2) נלחץ).

נדליק חזרה את X0.





```c
#define X0 PORTCbits.RC2
#define X1 PORTCbits.RC1
#define X2 PORTCbits.RC4
#define X3 PORTGbits.RG6
#define Y0 PORTCbits.RC3
#define Y1 PORTGbits.RG7
#define Y2 PORTGbits.RG8
#define Y3 PORTGbits.RG9

/*Left digit represents the X number
  Right digit represents te Y number*/

int getPressedKbBtn(){
    X0=1; X1=1; X2=1; X3=1;

    X0=0;
    if(!Y0)
        return 00;
    if(!Y1)
        return 01;
    if(!Y2)
        return 02;
    if(!Y3)
        return 03;
    X0=1;

    X1=0;
    if(!Y0)
        return 10;
    if(!Y1)
        return 11;
    if(!Y2)
        return 12;
    if(!Y3)
        return 13;
    X1=1;

    X2=0;
    if(!Y0)
        return 20;
    if(!Y1)
        return 21;
    if(!Y2)
        return 22;
    if(!Y3)
        return 23;
    X2=1;

    X3=0;
    if(!Y0)
        return 30;
    if(!Y1)
        return 31;
    if(!Y2)
        return 32;
    if(!Y3)
        return 33;
    X3=1;

    return -1;
}
```

```
void initKeyBoard(){
    /* X's init */
    TRISCbits.TRISC2=0;//X0 (RC2) as Output
    TRISCbits.TRISC1=0;//X1 (RC1) as Output
    TRISCbits.TRISC4=0;//X2 (RC4) as Output
    TRISGbits.TRISG6=0;//X3 (RG6) as Output
    ANSELGbits.ANSG6=0;//         disable Anaglog
    X0=1; X1=1; X2=1; X3=1;


    /* Y's init */
    TRISCbits.TRISC3 =1;//Y0 (RC3) as Input
    CNPUCbits.CNPUC3;   //Pull Up Resistor enable
    TRISGbits.TRISG7=1;//Y1 (RG7) as Input
    ANSELGbits.ANSG7=0;
    CNPUGbits.CNPUG7;
    TRISGbits.TRISG8=1;//Y2 (RG8) as Input
    ANSELGbits.ANSG8=0;
    CNPUGbits.CNPUG8;
    TRISGbits.TRISG9=1;//Y3 (RG9) as Input
    ANSELGbits.ANSG9=0;
    CNPUGbits.CNPUG9;
}
```

<u>השהיות בין הלחיצות כדי לקבל לחיצה בודדת</u>:

בין בדיקה לבדיקה (בדיקה = איזה מקש נלחץ ~ הפעלת פונ' getPressedBtn())

נריץ את השורות:

```
for(i=0;i<64000;i++);

for(i=0;i<64000;i++);

for(i=0;i<64000;i++);

for(i=0;i<64000;i++);

for(i=0;i<64000;i++);

for(i=0;i<64000;i++);
```

(לדוגמא שרוצים להדפיס על המסך את התו המייצג את המקש שנלחץ – אם לא נשים השהיה בין הלחיצות יודפס למסך כמה פעמים עבור כל לחיצה).

דוגמא 2 – תוכנית של מנחם:

```
void initKeyBoard(); int in_y(int a);

/* תוכנית שמדפיסה על הלדים את המספר של הכפתור שנלחץ בייצוג בינארי*/
void main(void) {
  TRISA&=0xff00;//led
  initKeyBoard();
  char scan_key[] = {
  //  XY
    0x44, 1, //0b 0100 0100
    0x34, 2, //0b 0011 0100
    0x24, 3, //0b 0010 0100
    0x43, 4, //0b 0100 0100
    0x33, 5, //0b 0100 0100
    0x23, 6, //0b 0100 0100
    0x42, 7, //0b 0100 0100
    0x32, 8, //0b 0100 0100
    0x22, 9, //0b 0100 0100
    0x41, 0  //0b 0100 0100
  };

  int i, xy;
  while (1) {
    while (1) {
      i = 1;

      PORTCbits.RC2 = 1;
      PORTCbits.RC1 = 1;
      PORTCbits.RC4 = 1;
      PORTGbits.RG6 = 1;
      i = 1;
      PORTCbits.RC2 = 0;
      xy = in_y(1);
      if (xy != 0xff)
        break;
      PORTCbits.RC2 = 1;
      i = 2;
      PORTCbits.RC1 = 0;
      xy = in_y(2);
      if (xy != 0xff)
        break;
      PORTCbits.RC1 = 1;
      i = 3;
      PORTCbits.RC4 = 0;
      xy = in_y(3);
      if (xy != 0xff)
        break;
      PORTCbits.RC4 = 1;
      i = 4;
      PORTGbits.RG6 = 0;
      xy = in_y(4);
      if (xy != 0xff)
        break;
      PORTGbits.RG6 = 1;
    }
    for (i = 0; i < 20; i += 2)
      if (scan_key[i] == xy)
        PORTA = scan_key[i + 1];
  }
}
```

```
int in_y(int a) {
  int j = 1, flag = 0;
  if (!PORTCbits.RC3) {
    flag = 1;
    j = 1;
  } else if (!PORTGbits.RG7) {
    flag = 1;
    j = 2;
  } else if (!PORTGbits.RG8) {
    flag = 1;
    j = 3;
  } else if (!PORTGbits.RG9) {
    flag = 1;
    j = 4;
  }
  if (flag == 0)
    return (0xff);
  else
    return (j | (a << 4));

}
```

ADC (Analog to Digital)

רכיב אשר **מקבל** מתח מכניסה (אנלוגית) נבחרת מ AN0 - AN15.

- כל כניסה **ANi** מחוברת ליציאה **RBi**.

מחזיר מספר בעל 10 ביט (0-1024) המייצג את המתח שקיבל.

לדוגמא:

הכניסה הנבחרת AN2 שהמתח בה נע בין 0v ל3.3v

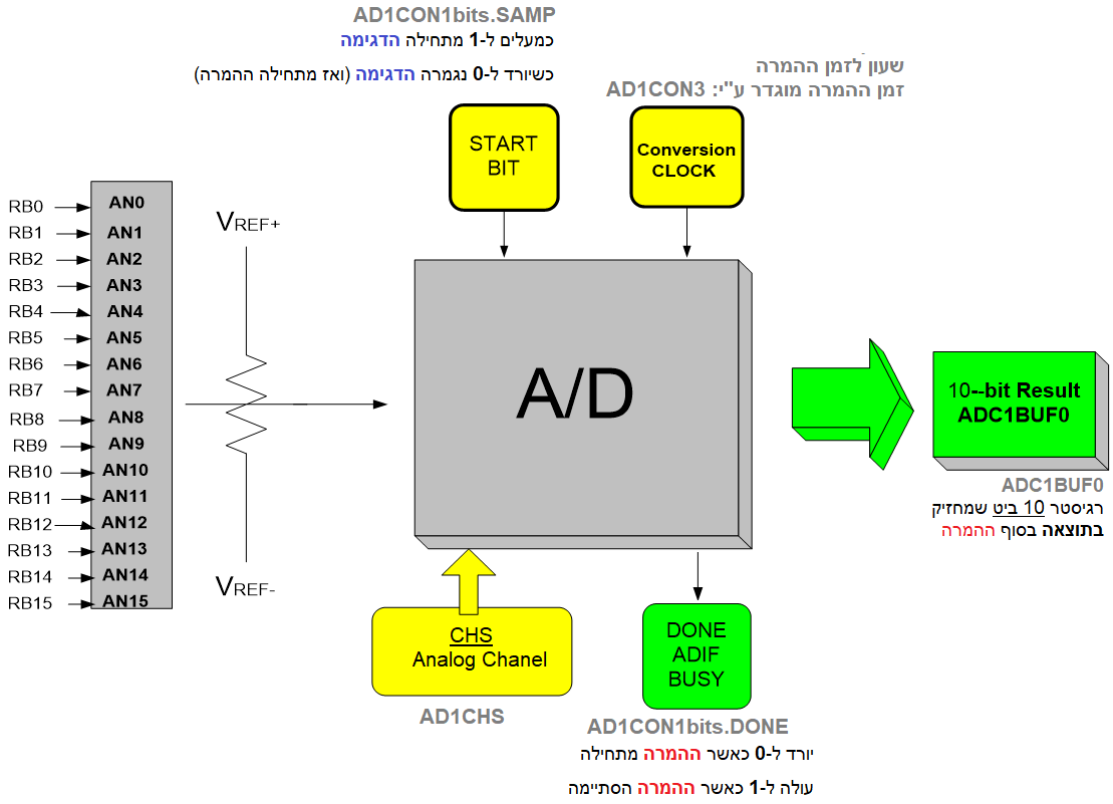
אם המתח שנדגם בכניסה AN2(שמחוברת ליציאה RB2) הוא 2v

אז יחזיר: $\frac{2v}{3.3v-0v} * 1024 = \mathbf{621}$

המרה בקוד : `float V = (((float)ADC_AnalogRead(2)) / 1023) * 3.3;`

לפני ההפעלה צריך להפעיל את האנלוג:

```
TRISBbits.TRISB2 = 1;
ANSELBbits.ANSB2 = 1;
```



התהליך שהADC מבצע:

1. **דגימה(Sampling)** – דוגם את המתח ביציאה הנבחרת.

* **AD1CON1bits.SAMP = 1** – מבצעים להתחלת את הדגימה.

* **AD1CON1bits.SAMP is 0** – כאשר הדגימה מסתיימת (ואז ההמרה מתחילה אוטומטית)

2. **המרה(Conversion)** – ממיר את המתח למספר מייצג (בגודל 10-ביט) שאותו שם על הרגיסטר ASVBUF0.

* **AD1CON1bits.DONE is 0** – כאשר מתחילה ההמרה.

* **AD1CON1bits.DONE is 1** – כאשר מסתיימת ההמרה ההמרה.

* **AD1CON3** – הערך בו קובע את משך זמן ההמרה (המקסימלי), ככל שיהיה משך זמן גדול יותר אז יהיו פחות שגיאות אך ההמרה תיקח יותר זמן.

```
void ADC_Init(){

    AD1CON1 =0;
    AD1CON1bits.SSRC = 7;   //Auto convert (after sampeling)
    AD1CON1bits.FORM = 0;   //16 bit integer result

    // Setup for manual sampling
    AD1CSSL =0;
    AD1CON3 =0x0002; //ADC Conversion Clock
    AD1CON2 =0;
    AD1CON2bits.VCFG = 0;

    // Turn on ADC
    AD1CON1bits.ON = 1; //ADC on (activate)
}


unsigned int ADC_AnalogRead(unsigned char analogPIN){

    int adc_val = 0;

    //IEC0bits.T2IE = 0;
    AD1CHS = analogPIN << 16;       // AD1CHS<16:19> controls which
                                    //analog pin goes to the ADC

    AD1CON1bits.SAMP = 1;           // Start Bit -  Begin sampling
    while( AD1CON1bits.SAMP );      // wait until acquisition is done
    while( ! AD1CON1bits.DONE );    // wait until conversion is done

    adc_val = ADC1BUF0;
    //IEC0bits.T2IE = 1;
    return adc_val;
}
```

Virtual Address	Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BF80_9000	AD1CON1	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	ON	FRZ	SIDL	—	—	FORM2	FORM1	FORM0
		7:0	SSRC2	SSRC1	SSRC0	CLRASAM	—	ASAM	SAMP	DONE
BF80_9004	AD1CON1CLR	31:0	Write clears selected bits in AD1CON1, read yields undefined value							
BF80_9008	AD1CON1SET	31:0	Write sets selected bits in AD1CON1, read yields undefined value							
BF80_900C	AD1CON1INV	31:0	Write inverts selected bits in AD1CON1, read yields undefined value							
BF80_9010	AD1CON2	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	VCFG2	VCFG1	VCFG0	OFFCAL	—	CSCNA	—	—
		7:0	BUFS	—	SMPI3	SMPI2	SMPI1	SMPI0	BUFM	ALTS
BF80_9014	AD1CON2CLR	31:0	Write clears selected bits in AD1CON2, read yields undefined value							
BF80_9018	AD1CON2SET	31:0	Write sets selected bits in AD1CON2, read yields undefined value							
BF80_901C	AD1CON2INV	31:0	Write inverts selected bits in AD1CON2, read yields undefined value							
BF80_9020	AD1CON3	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	ADRC	—	—	SAMC4	SAMC3	SAMC2	SAMC1	SAMC0
		7:0	ADCS7	ADCS6	ADCS5	ADCS4	ADCS3	ADCS2	ADCS1	ADCS0
BF80_9024	AD1CON3CLR	31:0	Write clears selected bits in AD1CON3, read yields undefined value							
BF80_9028	AD1CON3SET	31:0	Write sets selected bits in AD1CON3, read yields undefined value							
BF80_902C	AD1CON3INV	31:0	Write inverts selected bits in AD1CON3, read yields undefined value							
BF80_9040	AD1CHS	31:24	CH0NB	—	—	—	CH0SB3	CH0SB2	CH0SB1	CH0SB0
		23:16	CH0NA	—	—	—	CH0SA3	CH0SA2	CH0SA1	CH0SA0
		15:8	—	—	—	—	—	—	—	—
		7:0	—	—	—	—	—	—	—	—
BF80_9044	AD1CHSCLR	31:0	Write clears selected bits in AD1CHS, read yields undefined value							
BF80_9048	AD1CHSSET	31:0	Write sets selected bits in AD1CHS, read yields undefined value							
BF80_904C	AD1CHSINV	31:0	Write inverts selected bits in AD1CHS, read yields undefined value							
BF80_9060	AD1PCFG	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
		7:0	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
BF80_9064	AD1PCFGCLR	31:0	Write clears selected bits in AD1PCFG, read yields undefined value							
BF80_9068	AD1PCFGSET	31:0	Write sets selected bits in AD1PCFG, read yields undefined value							
BF80_906C	AD1PCFGINV	31:0	Write inverts selected bits in AD1PCFG, read yields undefined value							
BF80_9050	AD1CSSL	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8
		7:0	CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0

REGISTER 22-1: AD1CON1: ADC CONTROL REGISTER 1 (CONTINUED)

bit 4 **CLRASAM:** Stop Conversion Sequence bit (when the first A/D converter interrupt is generated)
1 = Stop conversions when the first ADC interrupt is generated. Hardware clears the ASAM bit when the ADC interrupt is generated. פסיקות
0 = Normal operation, buffer contents will be overwritten by the next conversion sequence

bit 3 **Reserved:** Write '0'; ignore read

bit 2 **ASAM:** ADC Sample Auto-Start bit צורת הפעלה
1 = Sampling begins immediately after last conversion completes; SAMP bit is automatically set.
0 = Sampling begins when SAMP bit is set

bit 1 **SAMP:** ADC Sample Enable bit אפשור דגימה
1 = The ADC SHA is sampling
0 = The ADC sample/hold amplifier is holding
When ASAM = 0, writing '1' to this bit starts sampling.
When SSRC = 000, writing '0' to this bit will end sampling and start conversion.

bit 0 **DONE:** A/D Conversion Status bit דגל BUSY
1 = A/D conversion is done
0 = A/D conversion is not done or has not started
Clearing this bit will not affect any operation in progress.
Note: The DONE bit is not persistent in automatic modes. It is cleared by hardware at the beginning of the next sample.

REGISTER 22-3: AD1CON3: ADC CONTROL REGISTER 3

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit

R/W-0	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADRC	—	—	SAMC<4:0>				
bit 15							b

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADCS<7:0>							
bit 7							b

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **Reserved:** Write '0'; ignore read

bit 15 **ADRC:** ADC Conversion Clock Source bit קצב המרה
1 = ADC internal RC clock
0 = Clock derived from Peripheral Bus Clock (PBClock)

bit 14-13 **Reserved:** Write '0'; ignore read

bit 12-8 **SAMC<4:0>:** Auto-Sample Time bits
11111 = 31 T_{AD}
. . . . .
00001 = 1 T_{AD}
00000 = 0 T_{AD} (Not allowed)

bit 7-0 **ADCS<7:0>:** ADC Conversion Clock Select bits
11111111 = $T_{PB} \cdot (ADCS<7:0> + 1) \cdot 2 = 512 \cdot T_{PB} = T_{AD}$
. . . . . .
00000001 = $T_{PB} \cdot (ADCS<7:0> + 1) \cdot 2 = 4 \cdot T_{PB} = T_{AD}$
00000000 = $T_{PB} \cdot (ADCS<7:0> + 1) \cdot 2 = 2 \cdot T_{PB} = T_{AD}$

REGISTER 22-2: AD1CON2: ADC CONTROL REGISTER 2

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit

R/W-0	R/W-0	R/W-0	R/W-0	r-x	R/W-0	r-x	r-x
VCFG<2:0>			OFFCAL	—	CSCNA	—	—
bit 15							bi

R-0	r-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BUFS	—	SMPI<3:0>				BUFM	ALTS
bit 7							bi

Legend:

R = Readable bit W = Writable bit P = Programmable bit r = Reserved bit

U = Unimplemented bit -n = Bit Value at POR: ('0', '1', x = Unknown)

bit 31-16 **Reserved:** Write '0'; ignore read

bit 15-13 **VCFG<2:0>:** Voltage Reference Configuration bits

	ADC VR+	ADC VR-
000	AVDD	AVSS
001	External VREF+ pin	AVSS
010	AVDD	External VREF- pin
011	External VREF+ pin	External VREF- pin
1xx	AVDD	AVSS

bit 12 **OFFCAL:** Input Offset Calibration Mode Select bit

מתח יחוס

1 = Enable Offset Calibration mode
VINH and VINL of the SHA are connected to VR-

0 = Disable Offset Calibration mode
The inputs to the SHA are controlled by AD1CHS or AD1CSSL

bit 11 **Reserved:** Write '0'; ignore read

bit 10 **CSCNA:** Scan Input Selections for CH0+ SHA Input for MUX A Input Multiplexer Setting bit

1 = Scan inputs
0 = Do not scan inputs

סריקה של הכניסות

bit 9-8 **Reserved:** Write '0'; ignore read

bit 7 **BUFS:** Buffer Fill Status bit

רגיסטר תוצאה

Only valid when BUFM = 1 (ADRES split into 2 x 8-word buffers).
1 = ADC is currently filling buffer 0x8-0xF, user should access data in 0x0-0x7
0 = ADC is currently filling buffer 0x0-0x7, user should access data in 0x8-0xF

bit 6 **Reserved:** Write '0'; ignore read

REGISTER 22-2: AD1CON2: ADC CONTROL REGISTER 2 (CONTINUED)

bit 5-2 **SMPI<3:0>:** Sample/Convert Sequences Per Interrupt Selection bits

פסיקות

1111 = Interrupts at the completion of conversion for each 16th sample/convert sequence
1110 = Interrupts at the completion of conversion for each 15th sample/convert sequence
. . . . .
0001 = Interrupts at the completion of conversion for each 2nd sample/convert sequence
0000 = Interrupts at the completion of conversion for each sample/convert sequence

bit 1 **BUFM:** ADC Result Buffer Mode Select bit

1 = Buffer configured as two 8-word buffers, ADC1BUF(7...0), ADC1BUF(15...8)
0 = Buffer configured as one 16-word buffer ADC1BUF(15...0.)

תוצאה

bit 0 **ALTS:** Alternate Input Sample Mode Select bit

1 = Uses MUX A input multiplexer settings for first sample, then alternates between MUX B and MUX A input multiplexer settings for all subsequent samples
0 = Always use MUX A input multiplexer settings

שימוש ב-MUX

REGISTER 22-4: AD1CHS: ADC INPUT SELECT REGISTER

R/W-0	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
CH0NB	—	—	—	CH0SB<3:0>			
bit 31							bit 24

R/W-0	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
CH0NA	—	—	—	CH0SA<3:0>			
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31 **CH0NB:** Negative Input Select for MUX B bit
1 = Channel 0 negative input is AN1
0 = Channel 0 negative input is VR-

bit 30-28 **Reserved:** Write '0'; ignore read

bit 27-24 **CH0SB<3:0>:** Positive Input Select for MUX B bits
1111 = Channel 0 positive input is AN15
1110 = Channel 0 positive input is AN14
1101 = Channel 0 positive input is AN13
.....
0001 = Channel 0 positive input is AN1
0000 = Channel 0 positive input is AN0

bit 23 **CH0NA:** Negative Input Select for MUX A Multiplexer Setting bit(2)
1 = Channel 0 negative input is AN1
0 = Channel 0 negative input is VR-

bit 22-20 **Reserved:** Write '0'; ignore read

bit 19-16 **CH0SA<3:0>:** Positive Input Select for MUX A Multiplexer Setting bits
1111 = Channel 0 positive input is AN15
1110 = Channel 0 positive input is AN14
1101 = Channel 0 positive input is AN13
.....
0001 = Channel 0 positive input is AN1
0000 = Channel 0 positive input is AN0

bit 15-0 **Reserved:** Write '0'; ignore read

REGISTER 22-5: AD1PCFG: ADC PORT CONFIGURATION REGISTER

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W	R/W-0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **Reserved:** Reserved for future use, maintain as '0'

bit 15-0 **PCFG<15:0>:** Analog Input Pin Configuration Control bits

1 = Analog input pin in Digital mode, port read input enabled, ADC input multiplexer input for this analog input connected to AVss

0 = Analog input pin in Analog mode, digital port read will return as a '1' without regard to the voltage on the pin, ADC samples pin voltage

REGISTER 22-6: AD1CSSL: ADC INPUT SCAN SELECT REGISTER

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W	R/W-0
CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-16 **Reserved:** Write '0'

bit 15-0 **CSSL<15:0>:** ADC Input Pin Scan Selection bits

1 = Select ANx for input scan

0 = Skip ANx for input scan

TABLE 22-3: PINS ASSOCIATED WITH THE ADC MODULE

Pin Name	Module Control	Controlling Bit Field	Pin Type	Buffer Type	TRIS	Description
AN0	ON	AD1PCFG<0>	A	—	Input	Analog Input
AN1	ON	AD1PCFG<1>	A	—	Input	Analog Input
AN2	ON	AD1PCFG<2>	A	—	Input	Analog Input
AN3	ON	AD1PCFG<3>	A	—	Input	Analog Input
AN4	ON	AD1PCFG<4>	A	—	Input	Analog Input
AN5	ON	AD1PCFG<5>	A	—	Input	Analog Input
AN6	ON	AD1PCFG<6>	A	—	Input	Analog Input
AN7	ON	AD1PCFG<7>	A	—	Input	Analog Input
AN8	ON	AD1PCFG<8>	A	—	Input	Analog Input
AN9	ON	AD1PCFG<9>	A	—	Input	Analog Input
AN10	ON	AD1PCFG<10>	A	—	Input	Analog Input
AN11	ON	AD1PCFG<11>	A	—	Input	Analog Input
AN12	ON	AD1PCFG<12>	A	—	Input	Analog Input
AN13	ON	AD1PCFG<13>	A	—	Input	Analog Input
AN14	ON	AD1PCFG<14>	A	—	Input	Analog Input
AN15	ON	AD1PCFG<15>	A	—	Input	Analog Input
VREF+	ON	AD1CON2<15:13>	P	—	—	Positive Voltage Reference
VREF-	ON	AD1CON2<15:13>	P	—	—	Negative Voltage Reference

Legend: ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
A = Analog
P = Power

Timers

טיימר הוא רכיב בעל גביש שנותן פולס בסוף כל פרק זמן (קבוע).

אורך פרק הזמן

אורך האינטרוול קבוע ומוגדר מראש ע"י הערכי PR,TMR,PS של הטיימר באופן הבא:

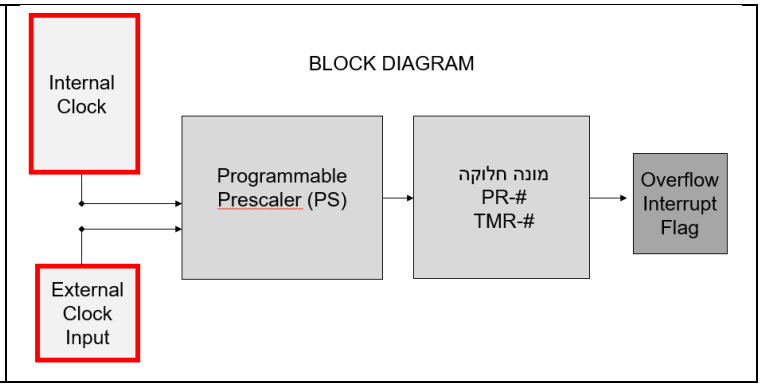
$$t = \frac{PS * (PR - TMR)}{f} \; [sec]$$

$$\textbf{in PIC32}: \boldsymbol{f} = 80MHz = \mathbf{80 * 10^6}\; Hz \;\; (Hz = {}^{1}/_{\text{sec}})$$

עושים init לטיימר באופן הבא (בדוגמא מאתחלים ומפעילים את TIMER1):

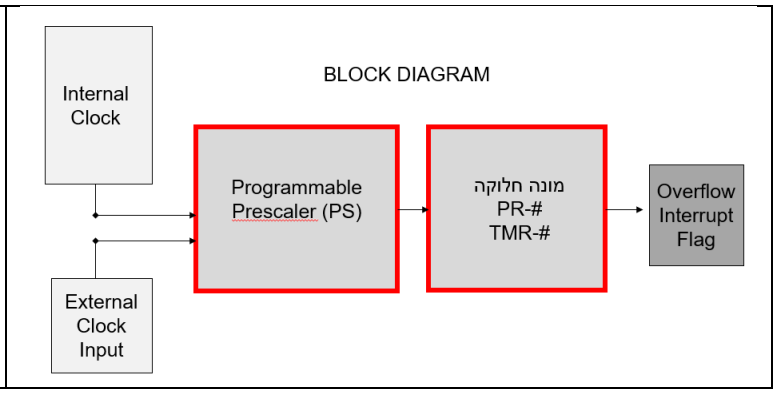
שלב 1 – מקור הפולסים:

T1CONbits.TCS=0;
ערך 0 – הגביש הפנימי של הטיימר
ערך 1 – פולסים מיציאה חיצונית.



שלב 2 - ערכים שקובעים את האינטרוול בין הפולסים של הגביש:

```
//Level 2 – (PR-TMR)
PR1 = 0x04FF;
TMR1 = 0;

//Level 1 – (PS)
T1CONbits.TCKPS0=1;
T1CONbits.TCKPS1=1;
```



PS0+PS1 = 2 ביטים שמייצגים מספר – בtimer1: 00:1, 01:8, 10:64, 11: 256

(בצד שמאל ערך הביטים בצד ימין הערך האמיתי של הPrescaler).

ככל שהPS גדול יותר, כך האינטרוול יהיה ארוך/איטי יותר.

שלב 3 – שורת איתחול נוספת (לא הורחב עליה)

```
T1CONbits.TGATE=0;
```

שלב 4 – איפוס ביט הIF (חשוב!)

רגיסטר הIF (Interrupt Flag) של הטיימר הוא ביט שעולה (מקבל ערך 1) בכל פעם שהגביש נותן פולס.

אם ברגיסטר היה ערך 1 וניתן פולס אז הוא ישאר 1..

לכן לפני שהIF עולה ל-1 צריך לאפס אותו.

```
IFS0bits.T1IF=0;
```

שלב 5 – הפעלת הטיימר

```
T1CONbits.ON=1;
```

דוגמא:

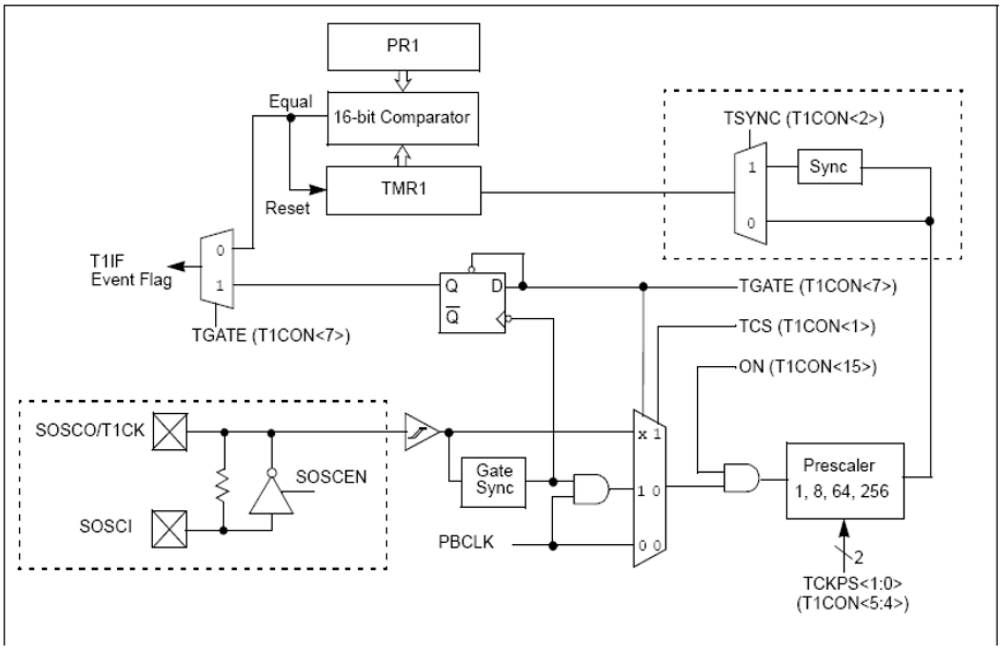
תוכנית שגורמת ללד הראשון לשנות מצב (להתכבות\להידלק) כל 0.2 שניות.

```
/*Left digit represents the X number
  Right digit represents te Y number*/

void main() {
    /*INIT TMR1*/
    //stage1
    TMR1=0;
    PR1=0xFFFF; //0xFFFF = 65535
    T1CONbits.TCKPS0=1;
    T1CONbits.TCKPS1=1;
    //stage2
    T1CONbits.TCS=0;
    //stage 3
    T1CONbits.TGATE=0;

    TRISA = 0;
    PORTA = 0;
    while(1){
        PORTAbits.RA0^=1;

        //DELAY by TMR1
        IFS0bits.T1IF=0;            //IF down (stag4)
        T1CONbits.ON=1;             //TMR1 on (stage5)
        while(!IFS0bits.T1IF);      //wait until IF up
        T1CONbits.ON=0;             //TMR1 off
    }
}
```

טיימרים: (בINIT ניתנו הזמנים המקסימליים לאינטרוול)

TMR1

FIGURE 13-1: TIMER1 BLOCK DIAGRAM(1)

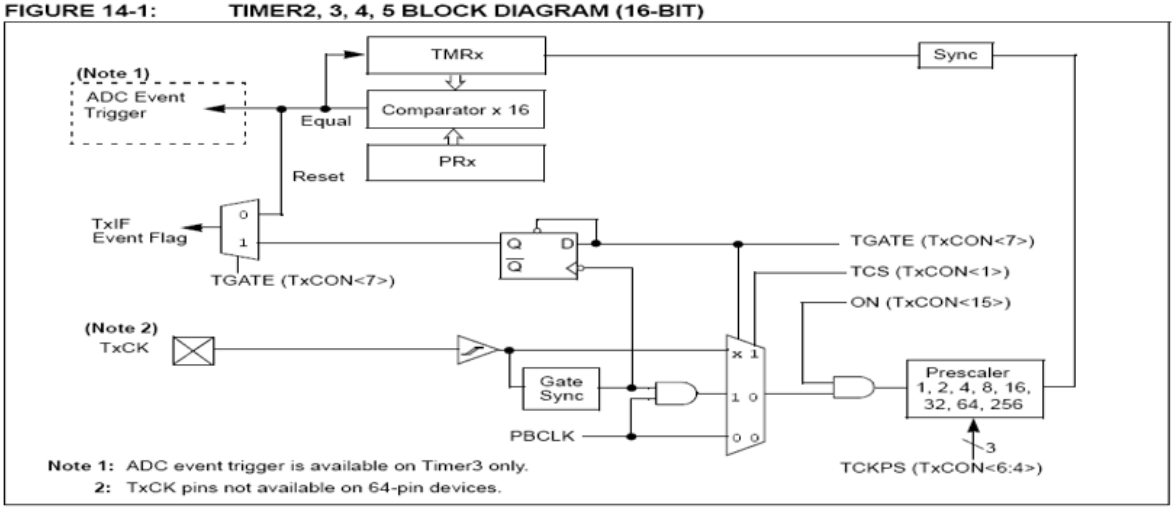



```
/*INIT TMR1*/
//stage1 (time=0.209 sec)
TMR1=0;
PR1=0xFFFF; //=65535
T1CONbits.TCKPS0=1; //PS=256
T1CONbits.TCKPS1=1;
//stage2
T1CONbits.TCS=0;
//stage 3
T1CONbits.TGATE=0;
T1CONbits.TSYNC=1;
//stag4
IFS0bits.T1IF=0;
//stag5
T1CONbits.ON=1;
```

```
void delay(void){
    T1CONbits.ON=0; //TMR off

    //stage1
    TMR1=0;
    PR1=0xFFFF; //0xFFFF = 65535
    T1CONbits.TCKPS0=1;
    T1CONbits.TCKPS1=1;
    //stage2
    T1CONbits.TCS=0;
    //stage 3
    T1CONbits.TGATE=0;
    T1CONbits.TSYNC=1;
    //stag4
    IFS0bits.T1IF=0;
    //stag5
    T1CONbits.ON=1;

    while(!IFS0bits.T1IF);//waiting until IF up
}
```

TMR2,3,4,5

FIGURE 14-1: TIMER2, 3, 4, 5 BLOCK DIAGRAM (16-BIT)




```
/*INIT TMR2*/
//stage1 (time=0.209 sec)
TMR2=0;
PR2=0xFFFF; //=65535
T2CONbits.TCKPS0=1; //PS=256
T2CONbits.TCKPS1=1;
T2CONbits.TCKPS2=1;
//stage2
T2CONbits.TCS=0;
//stage 3
T2CONbits.TGATE=0;
//stag4
IFS0bits.T2IF=0;
//stag5
T2CONbits.ON=1;
```

```
void delay(void){
    T2CONbits.ON=0; //TMR off

    //stage1
    TMR2=0;
    PR2=0xFFFF; //0xFFFF = 65535
    T2CONbits.TCKPS0=1;
    T2CONbits.TCKPS1=1;
    T2CONbits.TCKPS2=1;
    //stage2
    T2CONbits.TCS=0;
    //stage 3
    T2CONbits.TGATE=0;
    //stag4
    IFS0bits.T2IF=0;
    //stag5
    T2CONbits.ON=1;

    while(!IFS0bits.T2IF);//waiting until IF up
}
```

TMR23,45

מורכב מ2 הטיימרים TMR2, TMR3 על מנת לתת טיימר אחד שהאינטרוול שלו יותר גדול (הספירה מPR לTMR היא 32 ביט ולא 16 ביט).

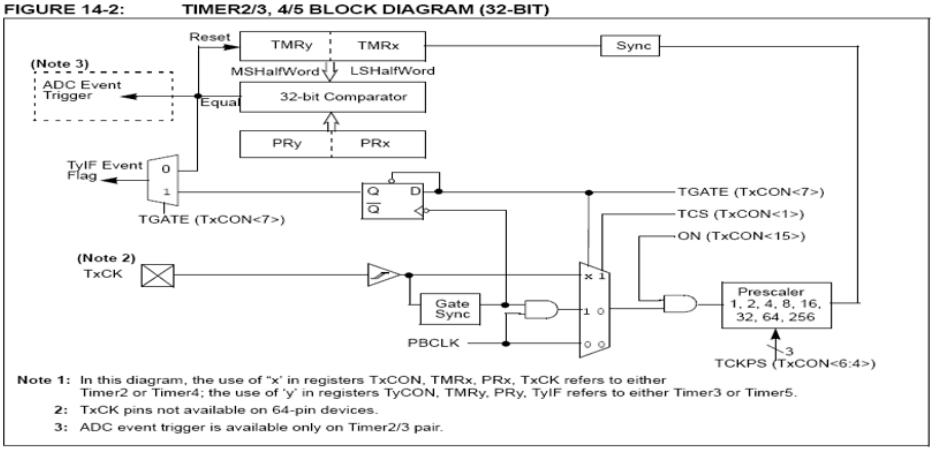
- TMR&PR:

PR23 = [PR**3** | PR**2**] **TMR23** = [TMR**3** | TMR**2**]

MSB LSB MSB LSB

- כל שאר הרגיסטרים הם של TMR2 למעט הIF ששייך לTMR3.
- בנוסף יש להפעיל את הmode 32bit כ"י הפעולה: T2CONbits.T32=1;

FIGURE 14-2: TIMER2/3, 4/5 BLOCK DIAGRAM (32-BIT)




```
/*INIT TMR23*/


//stage1(time=1sec)
TMR3=0;       //TMR msb
TMR2=0;       //TMR lsb
PR3= 0x4;     //PR 0x4c4b4=312500
PR2= 0xc4b4;
T2CONbits.TCKPS0=1;// PS=256
T2CONbits.TCKPS1=1;
T2CONbits.TCKPS2=1;
//stage2
T2CONbits.TCS=0;
//stage 3
T2CONbits.TGATE=0;
//stag4
IFS0bits.T2IF=0;
IFS0bits.T3IF=0;
//stag5
T2CONbits.T32=1;//mode 32bit
T2CONbits.ON=1;
```

```
void delay(void){
    T2CONbits.ON=0;
    T3CONbits.ON=0;
    //stage1(time=1sec)
    TMR3=0;       //TMR msb
    TMR2=0;       //TMR lsb
    PR3=0x4;      //PR msb
    PR2=0xc4b4;  //PR lsb
    T2CONbits.TCKPS0=1;// PS=256
    T2CONbits.TCKPS1=1;
    T2CONbits.TCKPS2=1;
    //stage2
    T2CONbits.TCS=0;
    //stage3
    T2CONbits.TGATE=0;
    //stage4
    IFS0bits.T2IF=0;
    IFS0bits.T3IF=0;
    //stage5
    T2CONbits.T32=1;//mode 32bit
    T2CONbits.ON=1;

    while(!IFS0bits.T3IF);
}
```

Interrupts

ניתן להגדיר לכל טיימר פונקציית interrupt שכל פעם שהIF של הטיימר עולה ל1 (הגביש נתן פולס) אז הפונקציה מתבצעת אוטומטית.

(ניתן להגדיר פונקציית פסיקה לכל מערכת שנותנת interrupt פיזי לpic32).

- כשמגדירים את הפונקציה מגדירים **priority** וגם **subpriority** כך שאם 2 פונקציות אמורות להיתבצע באותו הזמן אז תתבצע זו עם העדיפות הגבוהה יותר.
- בנוסף חייבים לייבא את: **sys/attribs.h**
- **Priority** 7 פסיקה בעדיפות הגבוהה ביותר, 1 בעדיפות הנמוכה ביותר, 0 פסיקה לא פעילה.
- **Subpriority(1-3)** 3 עדיפות גבוהה, 1 עדיפות נמוכה.

TMR2,3,4,5

```
#include <xc.h>
/*standard "pragma config's" in here… */

#include <sys/attribs.h>

void __ISR(_TIMER_5_VECTOR, ipl2auto) Timer5SR(void);
void __ISR(_TIMER_5_VECTOR, ipl2) Timer5SR(void) {
   PORTA++;
   IFS0bits.T5IF = 0;
}

void main() {
   int j;
   TRISA &= 0xff00;

    /*INIT TMR5*/
    //stage1 (time=0.209 sec)
    TMR5=0;
    PR5=0xFFFF; //=65535
    T5CONbits.TCKPS0=1; //PS=256
    T5CONbits.TCKPS1=1;
    T5CONbits.TCKPS2=1;
    //stage2
    T5CONbits.TCS=0;
    //stage 3
    T5CONbits.TGATE=0;
    //stag4
    IFS0bits.T5IF=0;
    //stag5
    T5CONbits.ON=1;

   /*Priority TMR5*/
   IPC5bits.T5IP = 2; // Priority
   IPC5bits.T5IS = 0; // Subpriority
   IEC0bits.T5IE = 1; // Enable Interrupt

   /*Enable Interrupts*/
   INTCONbits.MVEC = 1; //vector interrupt
   IPTMR = 0;           //INTERRUPT PROXIMITY TIMER REGISTER
   asm("ei");           //on interrupt
}
```

Timer1

```
#include <xc.h>
/*standard "pragma config's" in here… */

#include <sys/attribs.h>

void __ISR(_TIMER_1_VECTOR, ipl5auto) Timer1SR(void);
void __ISR(_TIMER_1_VECTOR, ipl5) Timer1SR(void) {
   PORTA++;
   IFS0bits.T1IF = 0;
}

void main() {
   int j;
   TRISA &= 0xff00;

    /*INIT TMR1*/
    //stage1 (time=0.209 sec)
    TMR1=0;
    PR1=0xFFFF; //=65535
    T1CONbits.TCKPS0=1; //PS=256
    T1CONbits.TCKPS1=1;
    //stage2
    T1CONbits.TCS=0;
    //stage 3
    T1CONbits.TGATE=0;
    //stag4
    IFS0bits.T1IF=0;
    //stag5
    T1CONbits.ON=1;

   /*Priority TMR1*/
   IPC1bits.T1IP = 5; // Priority
   IPC1bits.T1IS = 0; // Subpriority
   IEC0bits.T1IE = 1; // Enable Interrupt

   /*Enable Interrupts*/
   INTCONbits.MVEC = 1; //vector interrupt
   IPTMR = 0;           //INTERRUPT PROXIMITY TIMER REGISTER
   asm("ei");           //on interrupt
}
```

Timer23,45

```
#include <xc.h>
/*standard "pragma config's" in here… */

#include <sys/attribs.h>

void __ISR(_TIMER_3_VECTOR, ipl6auto) Timer23SR(void);
void __ISR(_TIMER_3_VECTOR, ipl6) Timer23SR(void) {
  PORTA++;
  IFS0bits.T3IF = 0;
}

void main() {
  int j;
  TRISA &= 0xff00;
  PORTA = 0;
  /*INIT TMR23*/
  //stage1(time=1sec)
  TMR3 = 0; //TMR msb
  TMR2 = 0; //TMR lsb
  PR3 = 0x4; //PR 0x4c4b4 = 312500
  PR2 = 0xc4b4;
  T2CONbits.TCKPS0 = 1; // PS=256
  T2CONbits.TCKPS1 = 1;
  T2CONbits.TCKPS2 = 1;
  //stage2
  T2CONbits.TCS = 0;
  //stage 3
  T2CONbits.TGATE = 0;
  //stag4
  IFS0bits.T2IF = 0;
  IFS0bits.T3IF = 0;
  //stag5
  T2CONbits.T32 = 1; //mode 32bit
  T2CONbits.ON = 1;

  /*Priority TMR23*/
  IPC3bits.T3IP = 6; // priority
  IPC3bits.T3IS = 0; // subpriority
  IEC0bits.T3IE = 1; // Enable Interrupt

  /*Enable Interrupts*/
  INTCONbits.MVEC = 1; //vector interrupt
  IPTMR = 0; //INTERRUPT PROXIMITY TIMER REGISTER
  asm("ei"); //on interrupt
}
```