Switches

Switches Control:

Name	PIC32 Pin
SW0	RPF3/RF3
SW1	RPF5/PMA8/RF5
SW2	RPF4/PMA9/RF4
SW3	RPD15/RD15
SW4	RPD14/RD14
SW5	AN11/PMA12/RB11
SW6	CVREFOUT/AN10/RPB10/CTED11PMA13/RB10
SW7	AN9/RPB9/CTED4/RB9

Example A: Configure Switches to input

1. Configure switches TRIS and disable analog:

```
TRISFbits.TRISF3 = 1; // RF3 (SW0) configured as input

TRISFbits.TRISF5 = 1; // RF5 (SW1) configured as input

TRISFbits.TRISF4 = 1; // RF4 (SW2) configured as input

TRISDbits.TRISD15 = 1; // RD15 (SW3) configured as input

TRISDbits.TRISD14 = 1; // RD14 (SW4) configured as input

TRISBbits.TRISB11 = 1; // RB11 (SW5) configured as input

ANSELBbits.ANSB11 = 0; // RB11 (SW5) disabled analog

TRISBbits.TRISB10 = 1; // RB10 (SW6) configured as input

ANSELBbits.ANSB10 = 0; // RB10 (SW6) disabled analog

TRISBbits.TRISB9 = 1; // RB9 (SW7) configured as input

ANSELBbits.ANSB9 = 0; // RB9 (SW7) disabled analog
```

2. Configure LED to be output and connect switches to LEDs (if switch value is 1 the relevant LED will turn on):

```
TRISA &= 0xff00; // set LED to output

while(1) {

PORTAbits.RA0=PORTFbits.RF3 ; // RF3 (SW0) configured as input

PORTAbits.RA1=PORTFbits.RF5 ; // RF5 (SW1) configured as input

PORTAbits.RA2=PORTFbits.RF4; // RF4 (SW2) configured as input

PORTAbits.RA3=PORTDbits.RD15 ; // RD15 (SW3) configured as input

PORTAbits.RA4=PORTDbits.RD14; // RD14 (SW4) configured as input

PORTAbits.RA5=PORTBbits.RB11 ; // RB11 (SW5) configured as input

PORTAbits.RA6=PORTBbits.RB10 ; // RB10 (SW6) configured as input

PORTAbits.RA7=PORTBbits.RB9 ; // RB9 (SW7) configured as input

}
```

LCD

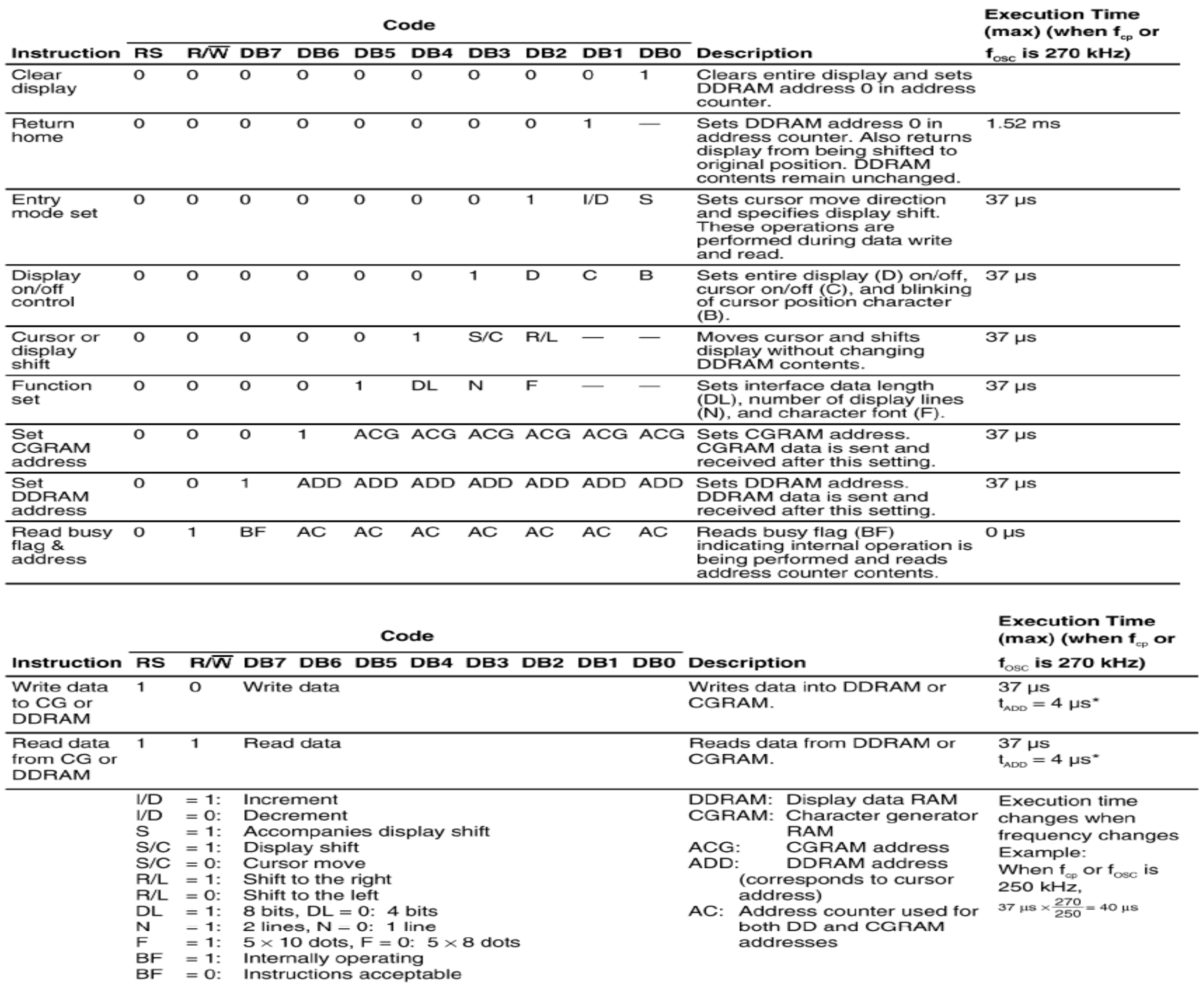
LCD Control:

In order to transfer instructions to the LCD, you need to configure the DISP_RS, DISP_RW and DISP_EN bits. Once they are configured, you need to transfer instructions and data to the LCD.

The table below describes the LCD input / output bits:

Name	PIC32 Pin	Description
DISP_RS	AN15/RPB15/OCFB/ CTED6/PMA0/RB15	Register Select: High for Data Transfer, Low for Instruction Transfer.
DISP_RW	RPD5/PMRD/RD5	Read/Write signal: High for Read mode, Low for Write mode.
DISP_EN	RPD4/PMWR/RD4	Read/Write Enable: High for Read, falling edge writes data.
DB0	PMD0/RE0	Data bits 0-7
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	

The table below describes the LCD instructions (DB0-7):



Instruction	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	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	DDRAM: Display data RAM	Execution time changes when frequency changes
I/D = 0: Decrement	CGRAM: Character generator RAM	Example:
S = 1: Accompanies display shift	ACG: CGRAM address	When f_{cp} or f_{OSC} is 250 kHz,
S/C = 1: Display shift	ADD: DDRAM address (corresponds to cursor address)	$37\ \mu s \times \frac{270}{250} = 40\ \mu s$
S/C = 0: Cursor move	AC: Address counter used for both DD and CGRAM addresses	
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		

The table below describes common examples of LCD instructions:

Code	Meaning
0x38	Configure LCD instructions to be 8 bit long, over both of LCD lines and each letter to expand over 5x8 LCD pixels (0x38 = 0011 1000).
0xE	Set the display configurations to be displayed over the LCD screen with a non-blinking cursor (0xE = 0000 1110).
0x1	Clear display (0x1 = 0000 0001).
0x40	Set the Character Generator initial address to 0 (0x40 = 0100 0000).

<u>Example A</u>: Write a string to the LCD screen

Note: the 'busy()' function is used to wait for a certain amount of time:

```
void busy(void) {
      char RD, RS;
      int STATUS_TRISE, portMap;
      RD = PORTDbits.RD5; // save settings of data / instruction transfer
      RS = PORTBbits.RB15; // save setting of read / write mode
      STATUS_TRISE = TRISE; // save TRIS settings
      PORTDbits.RD5 = 1; // set as read mode from the LCD
      PORTBbits.RB15 = 0; // set as instruction transfer to LCD
      portMap = TRISE; // save the LCD instructions to a new variable
      portMap |= 0x80; /* set DB7 (BF) to 1 in order to let LCD finish its
                          current action (if set to 1 then it is internally
                          operating */
      TRISE = portMap;
      do { // wait for LCD to finish its current action
            PORTDbits.RD4 = 1; // enable = 1
            PORTDbits.RD4 = 0; // enable = 0
      } while (PORTEbits.RE7); // register BF (status of LCD busy or not)
      PORTDbits.RD5 = RD; // restore pervious data / instruction transfer
      PORTBbits.RB15 = RS; // restore previous r / w mode
      TRISE = STATUS_TRISE; // restore previous TRIS settings
}
```

1. Configure the TRIS to output mode and disable the analog:

```
TRISBbits.TRISB15 = 0; // RB15 set as output

TRISDbits.TRISD5 = 0; // RD5 set as output

TRISDbits.TRISD4 = 0; // RD4 set as output

TRISE &= 0xff00;

ANSELBbits.ANSB15 = 0; // disable analog on RB15

ANSELEbits.ANSE2 = 0; // disable analog

ANSELEbits.ANSE4 = 0; // disable analog

ANSELEbits.ANSE5 = 0; // disable analog

ANSELEbits.ANSE6 = 0; // disable analog

ANSELEbits.ANSE7 = 0; // disable analog
```

2. Configure the PORT to instructions transfer and send the instructions:

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

PORTBbits.RB15 = 0; // set RS to receive instructions

PORTDbits.RD5 = 0; // set to write mode

for (i=0; i<6; i++) { // transfer instructions

      PORTE = instructions[i];

      // falling edge -> write data:

      PORTDbits.RD4 = 1;

      PORTDbits.RD4 = 0;

      busy();

}
```

3. Configure the PORT to data transfer and send the data:

```
char data[] = "Menachem Epstein";

PORTBbits.RB15 = 1; // set RS to data transfer

PORTDbits.RD5 = 0; // set to write mode

for (i=0; i<16; i++) { // transfer data (that is 16 characters long)

      PORTE = data[i];

      // falling edge -> write data:

      PORTDbits.RD4 = 1;

      PORTDbits.RD4 = 0;

      busy();

}
```

<u>Example B</u>: Write a new font to the LCD screen

1. Configure the TRIS to output mode and disable the analog (Example A-1).
2. Configure the PORT to instructions transfer and send the instructions:

```
char instructions[] = {0x38,0x38,0x38,0xe,0x6,0x1,0x40};

// 0x40 is for new font configuration in CGRAM.

PORTBbits.RB15 = 0; // set RS to receive instructions

PORTDbits.RD5 = 0; // set to write mode

for (i=0; i<6; i++) { // transfer instructions

      PORTE = instructions[i];

      // falling edge -> write data:

      PORTDbits.RD4 = 1;

      PORTDbits.RD4 = 0;

      busy();

}
```

3. Configure the new font via the Character Generator and store it in the RAM memory:

```
char  CG_newFont[16]={ 0x09,0x0a,0x0c,0x0c,0x0c,0x0a,0x09,0x09,   // א

                      0x26,0x29,0x31,0x21,0x29,0x2b,0x36,0x34 }; // ב

PORTBbits.RB15 = 1; // set RS to input

for(i=0; i<16; i++) {

      PORTE = CG_newFont[i];

      PORTDbits.RD4 = 1; // enable=1

      PORTDbits.RD4 = 0; // enable=0

      busy();

}
```

4. Configure the new font via the Character Generator and store it in the RAM memory:

```
PORTBbits.RB15 = 0; // set RS to instruction transfer

PORTE = 0x80; //set DDRAM address to first line (send instruction to DB 0-7)

// falling edge -> set to display to write data:

PORTDbits.RD4 = 1; // enable = 1

PORTDbits.RD4 = 0; // enable = 0

busy();

PORTBbits.RB15 = 1; // set RS to data transfer

int newFontAddress[] = {1,0,1};

for(i = 0; i<3; i++) { // write new font to the the LCD screen

      PORTE = newFontAddress[i];

      PORTDbits.RD4 = 1; // enable=1

      PORTDbits.RD4 = 0; // enable=0

      busy();

}
```

Timer

The MX3 chip has many timers that are scaled to different values.

Each timer can work internally or externally. In the lab we've only used internal clocks.

The timer uses a prescaler (hardware component) to translate its own clock-rate to another clock-rate. Thus, allows us to use less bits for a longer timer duration.

All timers work with a 16-bit or with a 32-bit register comparing. The Timer-# uses 2 registers: TMR# and PR#. Usually, TMR# is set to 0 and PR# and is set to a certain final value. The Timer-# increments the TMR# value until its value is as in PR#. When it occurs, a Timer# event flag turns on.

Calculation of the time (in seconds) using prescaler and final value (PR-#):

```
Rate = crystalValue / div / prescalerValue / finalValue [Hz]

Time = 1 / Rate [sec]
```

Calculation of final value (PR-#) for a given amount of time (in seconds):

```
finalValue = Time * (crystalValue / div / prescalerValue)
```

What is the PR that gets us a 0.3 **seconds** delay?

(80,000,000 / 2 / 256) * 0.3 = 46875 = 0xB71B

crystalValue (=ערך גביש) – Given by the chip manufacturer. At MX3 it is 80,000,000.

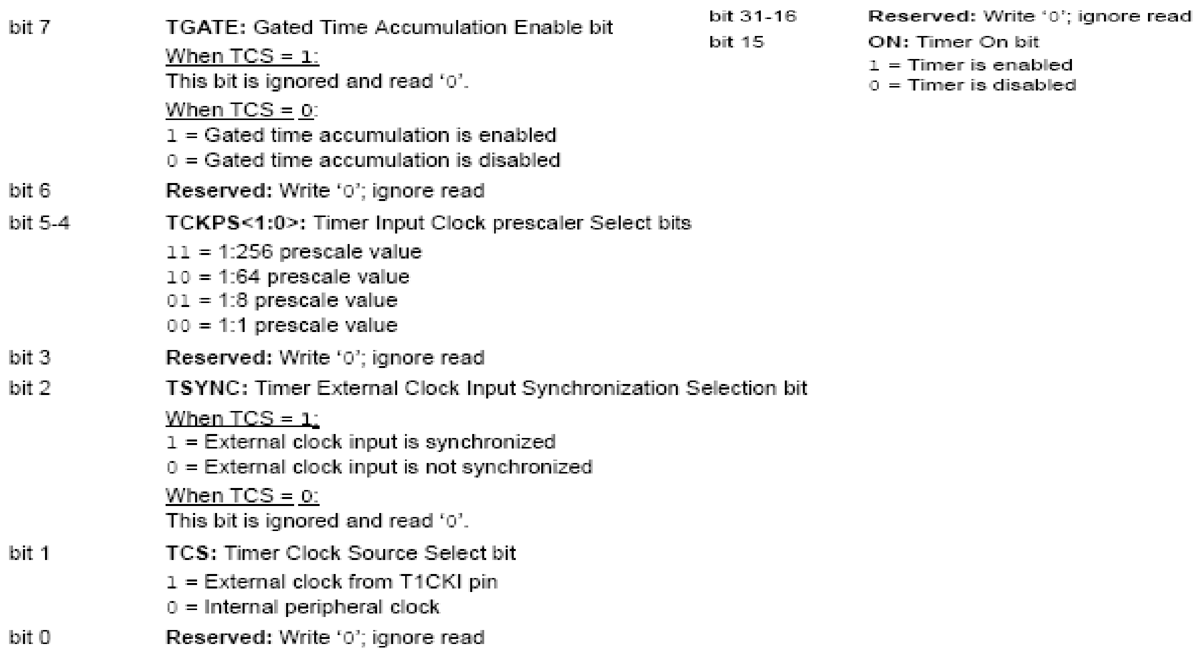
div – Normally is 2.

prescalerValue – Programmable, usually is a power of 2 (ex.: timer 1 prescaler can be 1/8/64/256).

finalValue – A programmed value (TMR-#, PR-#).

Example A: Activate Timer 1 (has a 16-bit comparator) as a stopwatch

Timer 1 Configuration:



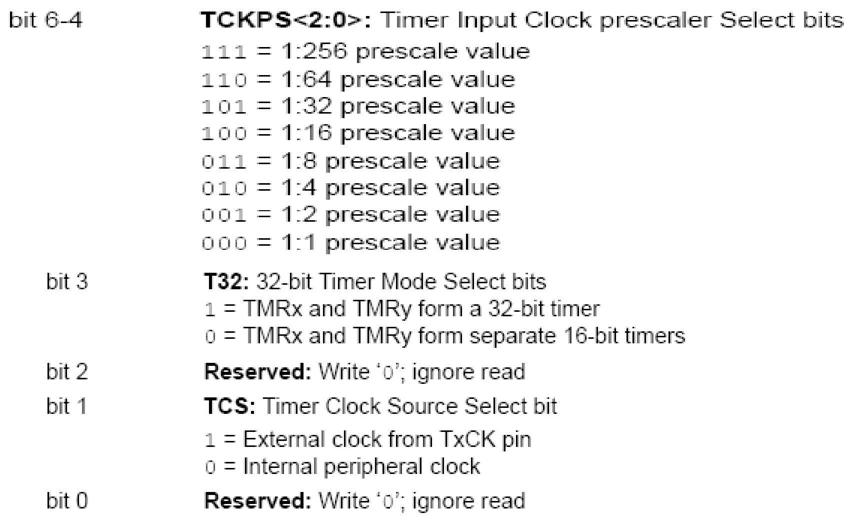
bit 31-16	**Reserved:** Write '0'; ignore read
bit 15	**ON:** Timer On bit 1 = Timer is enabled 0 = Timer is disabled
bit 7	**TGATE:** Gated Time Accumulation Enable bit When TCS = 1: This bit is ignored and read '0'. When TCS = 0: 1 = Gated time accumulation is enabled 0 = Gated time accumulation is disabled
bit 6	**Reserved:** Write '0'; ignore read
bit 5-4	**TCKPS<1:0>:** Timer Input Clock prescaler Select bits 11 = 1:256 prescale value 10 = 1:64 prescale value 01 = 1:8 prescale value 00 = 1:1 prescale value
bit 3	**Reserved:** Write '0'; ignore read
bit 2	**TSYNC:** Timer External Clock Input Synchronization Selection bit When TCS = 1: 1 = External clock input is synchronized 0 = External clock input is not synchronized When TCS = 0: This bit is ignored and read '0'.
bit 1	**TCS:** Timer Clock Source Select bit 1 = External clock from T1CKI pin 0 = Internal peripheral clock
bit 0	**Reserved:** Write '0'; ignore read

Timer 1 activation:

```
void delay(void) {

T1CONbits.ON = 0; // disable timer

T1CONbits.TGATE = 0; // disable time accumulation (clock counter)

T1CONbits.TCS = 1; // use internal clock

T1CONbits.TCKPS0 = 1; // set Prescaler to be 1:256 (according to the table)

T1CONbits.TCKPS1 = 1; // set Prescaler to be 1:256 (according to the table)

T1CONbits.TSYNC = 1; // synchronize the clock ticks

TMR1 = 0; // set initial value

PR1 = 0X04FF; // set final value

T1CONbits.ON = 1; // enable timer

IFS0bits.T1IF = 0; // set interrupt flag to 0

while (!IFS0bits.T1IF); // wait for timer to end and set T1IF bit to 1

}
```

<u>Example B</u>: Activate Timers 2 and 3 (each has a 16-bit comparator) as a 32-bit stopwatch

Timer 2 Prescaler configuration:



bit 6-4 **TCKPS<2:0>:** Timer Input Clock prescaler Select bits
111 = 1:256 prescale value
110 = 1:64 prescale value
101 = 1:32 prescale value
100 = 1:16 prescale value
011 = 1:8 prescale value
010 = 1:4 prescale value
001 = 1:2 prescale value
000 = 1:1 prescale value

bit 3 **T32:** 32-bit Timer Mode Select bits
1 = TMRx and TMRy form a 32-bit timer
0 = TMRx and TMRy form separate 16-bit timers

bit 2 **Reserved:** Write '0'; ignore read

bit 1 **TCS:** Timer Clock Source Select bit
1 = External clock from TxCK pin
0 = Internal peripheral clock

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

Timer 2 and 3 activation:

```
void delay(void) {

T2CONbits.ON = 0; // disable Timer 2

T2CONbits.TGATE = 0; // disable time accumulation (Timer 2 counter)

T2CONbits.TCS = 0; // use Timer 2 internal clock

T2CONbits.T32 = 1; // set Timer 2 register to 32-bit

T2CONbits.TCKPS0 = 1; // set Timer 2 Prescaler to 256

T2CONbits.TCKPS1 = 1; // set Timer 2 Prescaler to 256

T2CONbits.TCKPS2 = 1; // set Timer 2 Prescaler to 256

T3CONbits.ON = 0; // disable Timer 3

TMR3 = 0; // set initial value of TMRy to 0

TMR2 = 0; // set initial value of TMRx to 0

PR2 = 0X5000; // set final value of PR2x to 0x5000

PR3 = 0; // set final value of PR2y to 0

T2CONbits.ON = 1; // start Timer 2
IFS0bits.T2IF = 0; // set Timer 2 flag to 0

IFS0bits.T3IF = 0; // set Timer 3 flag to 0

while (!IFS0bits.T3IF); // wait for Timer 3 to end and set T3IF bit to 1

}
```

Example C: Activate Timer 1 as a 'x' seconds stopwatch

1. Calculation of PR-# value according to requested delay (in seconds):

```
void calculatePR(int requestedDelayInSeconds, int crystalValue,

                 int div, int prescalerValue) {

      return (crystalValue / div / prescalerValue) * requestedDelayInSeconds;

}
```

2. Delay method, with a given number of seconds, for Timer 1:

```
void delayForXseconds(int seconds) {

T1CONbits.ON = 0; // disable timer

T1CONbits.TGATE = 0; // disable time accumulation (clock counter)

T1CONbits.TCS = 1; // use internal clock

T1CONbits.TCKPS0 = 1; // set Prescaler to be 1:256 (according to the table)

T1CONbits.TCKPS1 = 1; // set Prescaler to be 1:256 (according to the table)

T1CONbits.TSYNC = 1; // synchronize the clock ticks

TMR1 = 0; // set initial value

PR1 = calculatePR(seconds, 80000000, 2, 256);

T1CONbits.ON = 1; // enable timer

IFS0bits.T1IF = 0; // set interrupt flag to 0

while (!IFS0bits.T1IF); // wait exactly 5 seconds and then set T1IF bit to 1

}
```

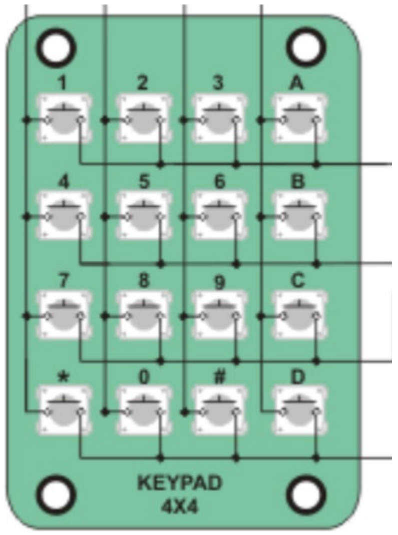
Keyboard

The keyboard is based on a matrix of 4x4. In order to figure out the pressed button, we need to iterate the matrix's rows and columns by the 'running-zero' method.

We will set the X values to be 0 and the Y values to be 1. Then we will set y0 to be 0 and check for a matching x. If a matching x is found, we can manage the coordinates of the pressed button. If no button is pressed, we will return a unique value (0xFF).

The keyboard indexes start from bottom right. Ex.:
0x23 means that the button '6' is pressed because:

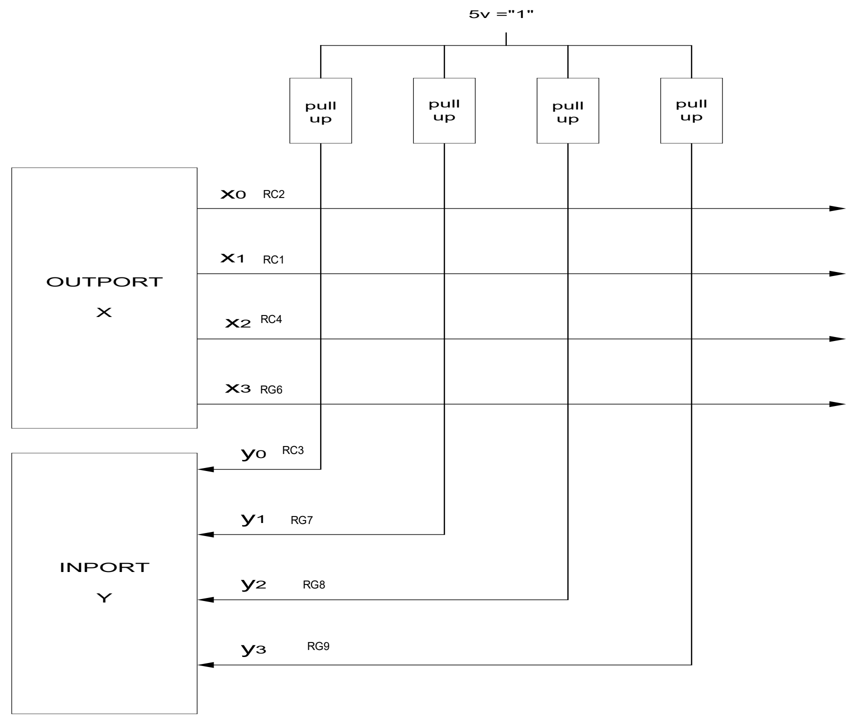
- $0x23 = (0010\ 0011)_2$.
- $2 = (0010)_2$ is the X index (column).
- $3 = (0011)_2$ is the Y index (row).



Keyboard Control:

Pmod Pin	Schematic Label	PIC32 Pin	Meaning
PMODA_1	JA1	RPC2/RC2	x0
PMODA_2	JA2	RPC1/RC1	x1
PMODA_3	JA3	RPC4/CTED7/RC4	x2
PMODA_4	JA4	AN16/C1IND/RPG6/SCK2/PMA5/RG6	x3
PMODA_7	JA7	RPC3/RC3	y0
PMODA_8	JA8	AN17/C1INC/RPG7/PMA4/RG7	y1
PMODA_9	JA9	AN18/C2IND/RPG8/PMA3/RG8	y2
PMODA_10	JA10	AN19/C2INC/RPG9/PMA2/RG9	y3

Keyboard Schema:



Example A: Keyboard management with LED

The program will show the x and y values of the pressed button.

1. Configure the TRIS and PULL UP:

```
TRISA& = 0xff00; // set LED to output

TRISCbits.TRISC2 = 0; // set RC2 (x0) to output

TRISCbits.TRISC1 = 0; // set RC1 (x1) to output

TRISCbits.TRISC4 = 0; // set RC4 (x2) to output

TRISGbits.TRISG6 = 0; // set RG6 (x3) to output

ANSELGbits.ANSG6 = 0; // disable RG6 analog

TRISCbits.TRISC3 = 1; // set RC3 (y0) to input

CNPUCbits.CNPUC3; // PULL UP נגד

TRISGbits.TRISG7 = 1; // set RG7 (y1) to input

ANSELGbits.ANSG7 = 0; // disable RG7 analog

CNPUGbits.CNPUG7; // PULL UP נגד

TRISGbits.TRISG8 = 1; // set RG8 (y2) to input
```

```
ANSELGbits.ANSG8 = 0; // disable RG8 analog

CNPUGbits.CNPUG8; // PULL UP נגד

TRISGbits.TRISG9 = 1; // set RG9 (y3) to input

ANSELGbits.ANSG9 = 0; // disable RG9 analog

CNPUGbits.CNPUG9; // PULL UP נגד
```

2. Setup the ‘y’ axis value (if pressed):

```
int in_y(int x) {

      int y = 1; // the number of the key column (y)

      int flag = 0; // 0 -> not pressed, 1 -> pressed

      if (!PORTCbits.RC3) { // check y0 status (if y0 == 0 it is pressed)

      flag = 1; y = 1; }

      else if (!PORTGbits.RG7) { // check y1 status (y1 == 0 means pressed)

      flag = 1; y = 2; }

      else if (!PORTGbits.RG8) { // check y2 status (y2 == 0 means pressed)

      flag = 1; y = 3; }

      else if (!PORTGbits.RG9) { // check y3 status (y3 == 0 means pressed)

      flag = 1; y = 4; }

      if (flag == 0) // no button is pressed, return unique value

         return 0xff;

      else

         return (y | (x << 4) ); // ex.: if x=1 and y=3 will return 0001 0011

}
```

3. Setup the 'running zero' to turn on the LED:

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

      while(1) {

      // set all X to be 1:

      PORTCbits.RC2 = 1; PORTCbits.RC1 = 1;

      PORTCbits.RC4 = 1; PORTGbits.RG6 = 1;

      i = 1;
      PORTCbits.RC2 = 0; // set x0 to 0

      xy = in_y(1); // check if any of y values is 1

      if (xy != 0xff) break; // if y is pressed then stop the loop (x=1)

      PORTCbits.RC2 = 1; // reset x0 to 1

      i=2;
      PORTCbits.RC1 = 0; // set x1 to 0

      xy = in_y(2); // check if any of y values is 1

      if (xy != 0xff) break; // if y is pressed then stop the loop (x=2)

      PORTCbits.RC1 = 1; // reset x1 to 1

      i = 3;
      PORTCbits.RC4 = 0; // set x2 to 0

      xy = in_y(3); // check if any of y values is 1

      if (xy != 0xff) break; // if y is pressed then stop the loop (x=3)

      PORTCbits.RC4 = 1; // reset x2 to 1

      i = 4;
      PORTGbits.RG6 = 0; // set x3 to 0

      xy = in_y(4); // check if any of y values is 1

      if (xy != 0xff) break; // if y is pressed then stop the loop (x=4)

      PORTGbits.RG6 = 1; // reset x3 to 1

      }

      PORTA = xy; // relevant LEDs will turn on (according to xy value)

}
```

<u>Example B</u>: Keyboard management with LCD

The program will show the ASCII value of the pressed button.

1. Configure the TRIS and PULL UP as in A-1
2. Setup the 'y' axis value (if pressed) as in A-2.
3. Setup the 'running zero' to turn as in A-3. The only difference is at the initialization of 'scan_key' sequence and the replacement of the last line with the following:

```
char scan_key[] = {0x44,'1',0x34,'2',0x24,'3',0x43,'4',0x33,'5',

                   0x23,'6',0x42,'7',0x32,'8',0x22,'9',0x41,'0'};

 ***** same as A-3 *****

for(i = 0; i < 20; i += 2) {

      if(scan_key[i] == xy) PORTE = scan_key[i+1];

      PORTDbits.RD4 = 1; // falling edge

      PORTDbits.RD4 = 0; // falling edge

      busy();

}
```

<u>Example C</u>: Configure the need to release the button before any continue

If any button is still pressed, the program will be stuck on one of the 'while' loops. When all buttons are released, the program will continue.

```
// set all X to 0 -> all X are pressed

PORTCbits.RC2 = 0; PORTCbits.RC1 = 0; PORTCbits.RC4 = 0;PORTGbits.RG6 = 0;

// wait for all Y to be 1 -> all Y are not pressed

while(!PORTCbits.RC3); while(!PORTGbits.RG7);

while(!PORTGbits.RG8); while(!PORTGbits.RG9);
```

Interrupts

An interrupt is a response by the processor to an event that needs attention from the software. Interrupt received due to an asynchronous signal that can be received by software or hardware. The interrupt allows us to execute a certain method that is not on the main execution flow and then go back to execute the main program flow.

Our chip allows us to perform hardware-initiated interrupts. It contains an interrupt vector, that is composed of N interrupt flags. Each flag has a priority, so that when several interrupts are requested, the processor executes them according to their priority. The higher the priority is, the higher probability for its interrupt method to be executed first.

When an interrupt is requested, the interrupt flag changes its value to 1 and an interrupt is made. When the interrupt is taken care of, the program flow goes back to where it was interrupt originally.

PIC32 Interrupt Control System – a short brief:

- INTCON: Interrupt Control Register.
- INTSTAT: Interrupt Status Register.
- IPTMR: Interrupt Proximity Timer Register.
- IFS0, IFS1: Interrupt Flag Status Registers.
- IEC0, IEC1: Interrupt Enable Control Registers.
- IPC0 - IPC11: Interrupt Priority Control Registers.

Example A: Interrupt by using Timer 1

1. Define the interrupt methods:

```
void __ISR(_CHANGE_NOTICE_VECTOR, IPL4) PinChangeHandler(void) {
      // keep the interrupt vector the same but set 4th index to be 0:
      PORTAbits.RA4 ^= 1;
      busy(); // wait for a specific period
      IFS1bits.CNGIF = 0; // Check if CN on BTND - RA15
      IFS1bits.CNCIF = 0; // Check if CN on BTND - RA15
}

void __ISR(_TIMER_1_VECTOR, ipl7) timer1_interrupt(void) {
      PORTAbits.RA7 ^= 1; // set 7th index on interrupt vector to 0
      // set Timer 1 interrupt flag to 0 (because the interrupt made it 1):
      IFS0bits.T1IF = 0;
}
```

2. Configure the keyboard:

```
TRISA &= 0xff00; // set LED to output
// set all X to output:
TRISCbits.TRISC2 = 0; // set RC2 to output
TRISCbits.TRISC1 = 0; // set RC1 to output
TRISCbits.TRISC4 = 0; // set RC4 to output
TRISGbits.TRISG6 = 0; // set RG6 to output
ANSELGbits.ANSG6 = 0; // disable RG6 analog
// set all Y to input:
TRISCbits.TRISC3 = 1; // set RC3 to input
CNPUCbits.CNPUC3; // PULL OUT נגד
```

```
TRISGbits.TRISG7 = 1; // set RG7 to input

ANSELGbits.ANSG7 = 0; // disable RG7 analog

CNPUGbits.CNPUG7; // PULL OUT נגד

TRISGbits.TRISG8 = 1; // set RG8 to output

ANSELGbits.ANSG8 = 0; // disable RG8 analog

CNPUGbits.CNPUG8; // PULL OUT נגד

TRISGbits.TRISG9 = 1; // set RG9 to output

ANSELGbits.ANSG9 = 0; // disable RG9 analog

CNPUGbits.CNPUG9; // PULL OUT נגד

LATGSET = 9; //???

// set all X to be 0:

PORTCbits.RC2 = 0; PORTCbits.RC1 = 0;

PORTCbits.RC4 = 0; PORTGbits.RG6 = 0;
```

3. Configure Timer 1:

```
T1CONbits.ON = 0; // stop Timer 1

T1CONbits.TGATE = 0; // disable time accumulation (clock counter)

T1CONbits.TCS = 0; // use external clock

T1CONbits.TCKPS0 = 1; // set Prescaler to be 1:256 (by the table)

T1CONbits.TCKPS1 = 1; // set Prescaler to be 1:256 (by the table)

T1CONbits.TSYNC = 1; // set Prescaler to be 1:256 (by the table)

TMR1 = 0; // set Timer 1 initial value

PR1 = 0Xffff; // set Timer 1 final value

IFS0bits.T1IF = 0; // set interrupt flag to value of 0

IEC0bits.T1IE = 1; // set flag to enable mode

IPC1bits.T1IP = 7; // set interrupt priority to 7

IPC1bits.T1IS = 0; // set interrupt sub-priority to 0
```

4. Configure the interrupt control of the keyboard to hardware control:

```
// Configure y0:

TRISCbits.TRISC3 = 1; // Set RA15 for digital input

CNPUCbits.CNPUC3 = 0; // Disable RC3 pull up resistor

CNENCbits.CNIEC3 = 1; // Enable RC3 change notification

IEC1bits.CNCIE = 1; // Enable RC3 CN interrupts

IFS1bits.CNCIF = 0; // Check if CN on BTND - RC3

// Configure y1:

TRISGbits.TRISG7 = 1; // Set RG7 for digital input

CNENGbits.CNIEG7 = 1; // Enable RG7 change notification

CNPUGbits.CNPUG7 = 0; // Disable RG8 pull up resistor

// Configure y2:

TRISGbits.TRISG8 = 1; // Set RG8 for digital input

CNPUGbits.CNPUG8 = 0; // Disable RG8 pull up resistor

CNENGbits.CNIEG8 = 1; // Enable RG8 change notification

// Configure y3:

TRISGbits.TRISG9 = 1; // Set RG9 for digital input

CNPUGbits.CNPUG9 = 0; // Disable RG9 pull up resistor

CNENGbits.CNIEG9 = 1; // Enable RG9 change notification
```

5. Configure the interrupt details (priority and enable interrupts):

```
CNCONGbits.ON = 1; // Set RA15 change notification on

IEC1bits.CNGIE = 1; // Enable RG CN interrupts

CNCONCbits.ON = 1; // Set RA15 change notification on

IEC1bits.CNCIE = 1; // Enable RC CN interrupts

// Set CN interrupt level:

IPC8bits.CNIP = 4; // Set CN interrupt group level 1

IPC8bits.CNIS = 0; // Set CN interrupt sub group level 0

IFS1bits.CNGIF = 0; // Check if CN on BTND - RG

IFS1bits.CNCIF = 0; // Check if CN on BTND - RC

// Start the interruption:

INTCONbits.MVEC = 1; // this is the interrupt vector -> set to 1 to enable it

IPTMR = 0; // INTERRUPT PROXIMITY TIMER REGISTER -> the counter -> set to 0

T1CONbits.ON = 1; // start the timer

asm("ei"); // assembly command to enable interrupt

while(1); // wait for interrupts! A real flow can be written within the while
```

Appendix A: Bit Cheat

X	Y	AND	OR	XOR
0	0	0	0	1
1	0	0	1	0
0	1	0	1	0
1	1	1	1	1

```
value AND 1 = value

value OR 0 = value

value AND 0 = 0

value OR 1 = 1
```