

PIC BASIC PRO

PIC Basic

سادگی و آسانی مهمترین ویژگی یک زبان برنامه نویسی سطح بالا می باشد که موجب شده است بطور گسترده ای در میکروکنترلر ها برای کارهای بزرگ استفاده شود. زبان برنامه نویسی سطح بالا با بکارگیری توابع آماده زمان برنامه نویسی را کوتاه می کند و همچنین پیچیدگی برنامه تنها به موضوع اصلی مورد نظر کاربر محدود می شود. برنامه نویس می تواند تمرکز بیشتری بر روی کار مورد نظر خود داشته باشد و از اتلاف وقت بر روی نوشتن کدهای جانبی که دائما در پروژه های مختلف تکرار می شود جلوگیری کند. مثلا در بسیاری از پروژه ها به برنامه برای ارسال اطلاعات به قطعه دیگر یا کامپیوتر نیاز است و یا نمایش پیغامی بر روی نمایشگر LCD و یا تولید پالس PWM و غیره نیاز است که همه اینها بصورت آماده در یک زبان سطح بالا مانند PIC Basic موجود است. برای نوشتن و کامپایل کردن برنامه PIC Basic از نرم افزاری Micro CODE STUDIO نسخه 1.4 شرکت Mecanique استفاده می کنیم . برنامه خود را در این ویرایشگر می نویسیم و با پسوند BAS ذخیره می کنیم . با کامپایل این برنامه دو مرحله پشت سر هم روی می دهد ، مرحله یک کامپایلر فایل BAS را به کد اسمبلی تبدیل می کند و با همان اسم و پسوند ASM. ذخیره می کند و در مرحله بعد فایل ASM توسط اسمبلر به کد HEX تبدیل می شود و برای ریختن در حافظه برنامه نویسی میکروکنترلر آماده می شود. نرم افزار Micro CODE STUDIO از طریق نرم افزار EPIC Win کد HEX را در میکروکنترلر می ریزد. در زیر، برنامه BLINK بعنوان مثال نشان داده شده است.

```
Program: BLINK.BAS

' Example of a program where the LED diode connected on
' PORT B pin 7 switches on and off every 0.5 seconds

Loop:
    High PORTB.7      ' Switch on LED on pin 7 of port B
    Pause 500         ' 0.5 sec pause

    Low PORTB.7      ' Switch off LED on pin 7 of port B
    Pause 500         ' 0.5 sec pause

    Goto loop        ' Go back to Loop

End                  ' End of program
```

شکل 1-1. برنامه BLINK به زبان بیسیک

```
Program: BLINK.HEX
Page: 1 / 1

:100000002828A301A200FF30A207031CA307031C9A
:1000100023280330A100DF300F200328A101E83E90
:10002000A000A109FC30031C1828A00703181528FC
:10003000A0076400A10F152820181E28A01C222844
:1000400000002228080083130313831264000800B1
:1000500006148316061083120130A300F430022028
:1000600006108316061083120130A300F43002201C
:0600700028286300392876
:02400E00753DFE
:00000001FF
```

شکل 2-1) تبدیل به فایل HEX ، برنامه BLINK

۲-۱. عناصر پایه زبان PIC BASIC

۱-۱-۱. شناسه ها

شناسه ها برای نامیدن یکی از عناصر PIC BASIC استفاده می شود. شناسه ها علاوه بر استفاده از حروف ، از اعداد نیز می تواند استفاده شود با این محدودیت که در ابتدای کلمه آورده نشود. شناسه به کوچکی یا بزرگی حرف حساس نیست و طول آن حداکثر ۳۲ کارکتر می باشد.

Symbol Taster = PORTA.0

Symbol LED_0 = PORTB.0

۱-۱-۲. برچسبها

برای مشخص کردن خطی که برنامه به آنجا جهش می کند بکار می رود

```
symbol Taster = PORTA.0
```

```
symbol LED_0 = PORTB.0
```

```
B0 var byte
```

```
Main:          ^ Label Main
```

```
    B0 = 0
```

```
    button Set,0,255,0,B0,1,LED_toggle
```

```
    goto Main
```

```
LED_toggle:    ^ Label LED_toggle
```

```
    toggle LED_0
```

```
    goto Main
```

```
    end
```

Name_constants con value_constants

با این دستور می توان به مقادیر ثابتی که در برنامه کاربرد خاصی دارند اسم داد تا برنامه بازخوانی راحتتری داشته باشد

```
minute con 60 ' No. of seconds in a minute
if seconds < minute then minute = minute + 1 ' If the number of seconds is different
' from 60, raise the variable minutes
```

ثابتها می توانند دسیمال ، هگزادسیمال و یا باینری باشد . ثابتهای دسیمال بدون پیشوند نوشته می شود و ثابتهای هگزا دسیمال با پیشوند \$ و باینری با % نشان داده می شوند.

```
56 ' 56 decimal
$0F ' 15 hexadecimal
%10001100 ' 140 binary
"A" ' ASCII value for decimal 65
"d" ' ASCII value for decimal 100
```

Name_variable var Type_variable

متغیرها در موقعیتهای RAM میکروکنترلر ذخیره می شوند و به این معنی است که تعداد متغیرها به اندازه RAM میکروکنترلر مورد نظر بستگی دارد. تعریف متغیر با کلمه VAR صورت می گیرد. PIC BASIC متغیرهای از نوع bit (۰ و ۱) ، byte (۰ تا ۲۵۵) و word (۰ تا ۶۵۵۳۵) را پشتیبانی می کند.

```
Fleg var bit ' Fleg is a variable of the type bit
B0 var byte ' B0 is a variable of the type byte
W0 var word ' W0 is a variable of the type word
B0 var W0.byte0 ' B0 is a first byte of the word W0
B1 var W0.byte1 ' B1 is a second byte of the word W0
```

Name_sequence var type_element [number of the elements]

متغیر آرایه ای از تعدادی متغیر متوالی تشکیل شده است که با اسم Type_element نشان داده می شود. این متغیر می تواند بیتی یا بایتی و یا کلمه باشد. تعداد عناصر آرایه در داخل "[]" نشان داده می شوند. هر عنصر با یک شاخص مشخص می شود که اولین عنصر با شاخص صفر قابل دسترسی است و تعداد این شاخصها بستگی به حافظه RAM میکروکنترلر دارد. مقدار ماکزیمم آرایه در جدول زیر نشان داده شده است.

| اندازه آرایه | |
|----------------|--------------------|
| نوع عنصر آرایه | تعداد ماکزیمم عنصر |
| BIT | 256 |
| BYTE | 96* |
| WORD | 48* |

* وابسته به نوع میکروکنترلر

جدول ۱-۱

۱۰ المان پشت سرهم از نوع byte (آرایه ۱۰ المانه) `Sequence1 var byte[10]` اولین عنصر آرایه را در بر دارد و `Sequence1[9]` آخرین عنصر را در بر دارد.

۱-۱-۶ اصلاح کننده ها

`new_name var old_name`

بوسیله اصلاح کننده ها این امکان ایجاد می شود که اسم جدیدی برای متغیر از قبل تعریف شده انتخاب کنیم. این دستور بندرت استفاده می شود.

`ADCResult var word`

`HigherByte var ADCresult.byte0`

‘ The new name for the higher byte of the

‘ word ADCresult

۱-۱-۷ سمبلها

`symbol old_name = new_name`

سمبلها شبیه اصلاح کننده ها اسم جدیدی از یک متغیر یا یک ثابت را به یک اسم قدیمی نسبت می دهند.

`symbol Taster = PORTA.0`

‘ Taster is a new name for RA0

`symbol LED_0 = PORTB.0`

‘ LED_0 is a new name for RB0

۱-۱-۸ دستور INCLUDE

دستور INCLUDE برای وارد کردن یک بخش از فایل BASIC بکار می رود. در این فایل ممکن است چندین تعریف از متغیرها و یا زیربرنامه ها ذخیره شده باشد که در قسمت های مختلف برنامه بکار رفته باشد. استفاده از زیر برنامه ها و متغیرهای فایل الحاقی شبیه این است که این زیربرنامه ها و متغیرها را خودتان در برنامه نوشته اید.

Include "modedefs.bas"

' The transfer modes that use the
' commands SERIN and SEROUT

symbol SO = PORTA.3

symbol SI = PORTB.0

B0 var byte

Loop:

serin SI,T2400,B0

serout SO,T2400,[B0]

goto Loop

end

۹-۱-۱ توضیحات

توضیحات برای بهتر فهمیدن برنامه بکار برده می شود. و برای استفاده از آن از علامت " " استفاده می کنیم

۱۰-۱-۱ خط برنامه با دستورات بیشتر

برای مترجم کردن و دید بیشتر یک برنامه می توان دستورات را دسته بندی کرد و هر دسته را در یک خط جای داد. وبا علامت " : " دستورات در یک خط از یکدیگر جدا می شوند.

B2 = B0

B0 = B1

B1 = B2

سه دستور بالا می توانند در یک خط قرار بگیرند.

B2 = B0 : B0 = B1 : B1 = B2

۱۱-۱-۱ انتقال یک دستور به خط دیگر

گاهی اتفاق می افتد که یک دستور بیشتر از اندازه یک خط نیاز دارد و یا برای رویت بهتر لازم است دنباله دستور در خط بعدی باشد برای این کار از علامت " _ " استفاده می کنیم. برای دستورات lookup ، brunch و sound بیشتر ممکن است این حالت پیش بیاید.

```
lookup KeyPress,["1","4","7","*","2","5","8","0","3",_
"6","9","#","N"],var1
```

۱۲-۱-۱ تعریف (DEFINE)

DEFINE the value parameter

دستورات زبان PIC BASIC دارای پارامترهای هستند که از پیش تعریف شده است و می توان با این دستور پارامترهای از پیش تعیین شده آنها را تغییر داد. برای مثال فرکانس اسیلاتور از پیش

تعریف شده ۴ مگا هرتز می باشد در نتیجه بسیاری از دستورات که وابسته به کلاک اسیلاتور می باشند خود را با فرکانس ۴ مگا هرتز تنظیم می کنند و اگر اسیلاتور ۱۲ مگا هرتز قرار داده شود کار این دستورات از حالت تنظیم شده در می آید. بنابراین با این دستور براحتی فرکانس اسیلاتور را تعیین می کنیم. به جدول ۲-۲ مراجعه کنید.

DISABLE ۱-۱-۱۳

قبل از وارد کردن زیر برنامه وقفه ، لازم است که وقفه ها غیر فعال شوند تا از وقفه جدید جلوگیری شود. به عبارت دیگر دستور DISABLE بیت GIE را از رجیستر INTCON بازنشانه می کند.

غیر فعال کردن اینتراپتها ' Desable

Myint:

led = 1

....

Resume

Enable

ENABLE ۱-۱-۱۴

بعد از اتمام اجرای زیر برنامه وقفه ، وقفه بایستی دوباره فعال شود که با دستور ENABLE صورت می گیرد. در واقع بیت GIE نشانه می شود.

ON INTERRUPT ۱-۱-۱۵

On interrupt LABEL

با این دستور بر چسب مکانی را که هنگام روی دادن وقفه به آنجا پرش صورت می گیرد را مشخص می کند. برچسب در ابتدای زیر برنامه وقفه قرار دارد.

On interuupt ISR ^۱ The interruption routine starts from the label ISR

Main: ^۱ Main program

goto Main

Disable

ISR: ^۱ Start of the interruption routine

.

.

^۱ The end of the interruption routine

Resume

Enable

| | | | |
|-------------------|-------|---|--|
| I2C_HOLD | 1 | pause I2C transfer while the tact is on a low level | I2COUT, I2COUT |
| I2C_INTERNAL | 1 | internal EEPROM in series 16Cexxx and 12Cxxx of the PIC microcontroller | I2COUT, I2COUT |
| I2C_SCLOUT | 1 | serial tact is a bipolar at the place of an open collector | I2CWRITE, I2CREAD |
| I2C_SLOW | 1 | for the tact > BMHz OSC with the devices of a standard velocity | I2CWRITE, I2CREAD |
| LCD_DREG | PORTD | LCD data port | LCDOUT, LCDIN |
| LCD_DBIT | 0 | Initial bit of a data 0 or 4 | LCDOUT, LCDIN |
| LCD_RSREG | PORTD | RS (Register select) port | LCDOUT, LCDIN |
| LCD_RSBIT | 4 | RS (Register select) pin | LCDOUT, LCDIN |
| LCD_EREG | PORTD | enable port | LCDOUT, LCDIN |
| LCD_EBIT | 3 | enable bit | LCDOUT, LCDIN |
| LCD_RWREG | PORTD | read/write port | LCDOUT, LCDIN |
| LCD_RWBIT | 2 | read/write bit | LCDOUT, LCDIN |
| LCD_LINES | 2 | No of LCD lines | LCDOUT, LCDIN |
| LCD_INSTRUCTIONUS | 2000 | the time of delay of instruction in microseconds (us) | LCDOUT, LCDIN |
| LCD_DATAUS | 50 | the time of delay of data in microseconds | LCDOUT, LCDIN |
| OSC | 4 | tact of the oscillator in MHz: 3(3.58) 4 8 10 12 16 20 25 32 33 40 | all instructions of the serial transfer and next pause |
| OSCCAL_1K | 1 | setting of OSCCAL for PIC12C671/CE673 microcontrollers | |
| OSCCAL_2K | 1 | the number of data bits | |
| SER2_BITS | 8 | the slowing of the tact of transfer | SHIFTOUT, SHIFIN |
| SHIFT_PAUSEUS | 50 | instruction LFSR in 18Cxxx microcontrollers | LFSR |
| BUTTON_PAUSE | 10 | | BUTTON |
| CHAR_PACING | 1000 | | SEROUT, SERIN |
| HSER_BAUD | 2400 | | HSEROUT, HSERIN |
| HSER_SBRG | 25 | | HSEROUT, HSERIN |

جدول 1-2

بازگشت از زیر برنامه وقفه را به برنامه اصلی بر عهده دارد

۱-۲ عملگرها

۱-۲-۱ عملگرهای ریاضی

عملگرهای ریاضی با دقت ۱۶ بیتی با مقادیر بدون علامت (۰ تا ۶۵۵۳۵) کار می کنند.

$$A = (B + C) * (D - E)$$

جدول زیر عملگرهای ریاضی که PIC BASIC پشتیبانی می کند را نشان می دهد.

| شرح عملگرها | | | |
|-------------|-----------------------------------|-------|-------------------------------|
| عملگر | شرح | عملگر | شرح |
| + | جمع | ABS | قدر مطلق یک عدد |
| - | تفریق | COS | کسینوس یک زاویه |
| * | ضرب | DCD | دیکد کردن بیتی |
| ** | نتایج در ۱۶ بیت بالا قرار می گیرد | DIG | مقدار یک رقم از یک عدد دسیمال |
| */ | نتایج در ۱۶ بیت بالا قرار می گیرد | MAX | ماکزیمم دو عدد |
| / | تقسیم | MIN | مینیمم دو عدد |
| // | باقیمانده | NCD | کدینگ بر اساس حق تقدم |
| << | شیفت به چپ | REV | برگرداندن بیت |
| >> | شیفت به راست | SIN | سینوس یک زاویه |
| = | انتقال مقدار | SQR | ریشه دوم یک عدد |

جدول ۱-۳

نکات و مثالها:

× ضرب : PIC BASIC از ضرب اعداد بطور کامل پشتیبانی نمی کند بلکه مقدار حاصل را در دو ۱۶ بیتی قرار می دهد

```
L0 var word
W1 var word
W2 var word
```

Main:

```
L0 = W1 * 100      ' Multiplies value W1 with 100 and
                  ' stores the result in lower 16 bits of L0
L0 = W1 ** 100     ' Multiplies value W1 with 100 and
                  ' stores result in 16 higher bits of L0
L0 = W1 */ W2      ' Reverts the 16 middle bits of the result
Loop: goto Loop
END
```

× تقسیم : در PIC BASIC تقسیم بصورت ۱۶ بیتی انجام می گیرد و همچنین می توان باقیمانده تقسیم را جداگانه بدست آورد. در مثال زیر این دو حالت نشان داده شده است.

```
W0 var word
W1 var word
W2 var word
```

Main:

```
W0 = W1 / 100      ' Divide the value W0 with 100 and
                  ' store the integer result in W1
W2 = W1 // 100     ' Remainder store in W2
```

```
Loop: goto Loop
END
```

× شیفت دادن : عملگر شیفت می تواند تعداد ۰ تا ۱۵ شیفت به چپ و یا راست را پشتیبانی کند.

Main:

```
W0 = W0 << 3      ' Shift W0 three places to the left
                  ' (same as multiplication with 8)
W0 = W0 >> 1      ' Shift W0 one place to the right
                  ' (same as division with 2)
```

```
Loop: goto Loop
END
```

× قدر مطلق: اگر یک BYTE بیشتر از 127 (بیت MSB آن یک) باشد مقدار ABS – 256 value را بر می گرداند. اگر یک WORD بزرگتر از 32767 (بیت MSB آن یک است) باشد، ABS مقدار value – 65535 را بر می گرداند.

B1 = ABS B0

× COS و SIN: زاویه ۰ تا ۳۵۹ مورد نیاز این عملگر بصورت یک عدد ۰ تا ۲۵۵ تامین می شود و برای این تبدیل می توان از یک lookup table استفاده کنیم.

B0 var byte

B1 var byte

B2 var byte

Main:

B0 = COS B1 ' 8-bit value of cosine B1 store in B0

' (index of Lookup table)

Lookup B0, [constant to determine_cosine], B2

' After this instruction the true value of

' cosine is stored in B2

Loop: goto Loop

END

× DCD: این عملگر مقدار ۰ تا ۱۵ را تبدیل به شماره بیت می کنند. اگر مقدار صفر را قرار دهیم بیت صفرم یک می شود و بقیه صفر می شوند.

B0 var byte

Main:

B0 = DCD 2 ' Contents B0 is %00000100

Loop: goto Loop

END

× DIG: تعداد رقمهای که نشان می دهند بین ۰ تا ۳ می باشد که عدد ۰ آخرین رقم از سمت راست می باشد.

B0 var byte
B1 var byte

Main:

```
B1 = 5843
B0 = B1 DIG 0    ' Contents B0 is 3
B0 = B1 DIG 1    ' Contents B0 is 4
B0 = B1 DIG 2    ' Contents B0 is 8
B0 = B1 DIG 3    ' Contents B0 is 5
```

Loop: goto Loop
END

× تعیین مقدار ماکزیمم و مینیمم : برای محدود کردن مقادیر به یک عدد استفاده می شود

B1 = B0 MAX 100

محتوای B1 مقدار بزرگتر بین B0 و ۱۰۰ می باشد (B1 مقداری بین ۱۰۰ و ۲۵۵ می باشد)

B1 = B0 MIN 100

محتوای B1 مقدار کوچکتر بین B0 و ۱۰۰ می باشد (B1 مقداری بین ۱۰۰ و ۰ می باشد)

× NCD : شماره اولین بیتی که از سمت چپ یک است را می دهد

B0 var byte

Main:

```
B0 = NCD %01001000    ' Contents B0 is 7
B0 = NCD %00001111    ' Contents B0 is 4
```

Loop: goto Loop
END

× REV : تعداد بیت‌های که می توانند یک شوند ۱ تا ۱۶ می باشد و با توجه به عدد مورد نظر
بیت‌های پایینتر معکوس می شوند.

B0 var byte

Main:

```
B0 = %10101100 REV 4    ' Contents B0 is %10100011
```

Loop: goto Loop
END

× SQR : ریشه دوم عدد مورد نظر در یک بایت قرار می گیرد.

B0 var byte

W1 var word

Main:

```
B0 = SQR W1    ' Square root of W1 store into B0
```

Loop: goto Loop
END

| عملگرهای بیتی | |
|---------------|----------------------|
| عملگرها | توضیح |
| & | AND منطقی روی بیتها |
| | OR منطقی روی بیتها |
| ^ | XOR منطقی روی بیتها |
| ~ | NOT منطقی روی بیتها |
| &/ | NAND منطقی روی بیتها |
| / | NOR منطقی روی بیتها |
| ^/ | NXOR منطقی روی بیتها |

جدول ۱-۴

| عملگرهای مقایسه ای | |
|--------------------|-----------|
| عملگر | توضیح |
| = or == | تساوی |
| <> or != | نامساوی |
| < | کوچکتر از |
| > | بزرگتر از |
| <= | کوچکتر از |
| >= | بزرگتر از |

جدول ۱-۵

| عملگرهای منطقی | |
|----------------|------------|
| عملگر | شرح |
| AND or && | AND منطقی |
| OR or | OR منطقی |
| XOR or ^ ^ | XOR منطقی |
| NOT | NOT منطقی |
| NOT AND | NAND منطقی |
| NOT OR | NOR منطقی |
| NOT XOR | NXOR منطقی |

جدول ۱-۶

۳-۱. دستورات PICBasic

دستورات به چهار بخش تقسیم می شوند

- ۱) دستورهای شاخه ای (مانند IF)
- ۲) دستورهای تکرارکننده (FOR ... NEXT و WHILE ... WEND)
- ۳) دستورهای پرش به خط (GOTO) یا یک زیربرنامه (CALL ، BRANCHL ، BRANCH ، RETURN ، GOSUB)
- ۴) دستورهای دستیابی به وسایل جانبی (مثلا ارتباط با LCD) با این دستورات برنامه نویس فکر خود را بر روی ماهیت اصلی برنامه خود متمرکز می کند و از اتلاف وقت خود بر روی مواردی همچون ارتباط با LCD جلوگیری می کند .

دستورات به ترتیب حروف الفبا لاتین :

۳-۱-۱. @ : وارد کردن یک خط برنامه با کد اسمبلی

ترکیب:

@ assembler's instruction

شرح:

اگر در ابتدای خط از علامت @ استفاده شود کد اسمبلی را در ترکیب برنامه PicBasic می توان نوشت. دستور @ برای الحاق کتابخانه های نوشته شده به زبان اسمبلی نیز استفاده می شود. توجه کنید در صورت استفاده از متغیرهای درون برنامه در این قسمت قبل از نام متغیر از خط زیر (دست) استفاده می شود.

در مثال زیر متغیر B0 در دستور @ بصورت _B0_ بکار برده می شود.

مثال :

```
B0 var byte
```

```
Main:
```

```
@ bsf _B0,7 ' نشانه گذاری بیت هفتم از متغیر B0
```

```
Loop: goto Loop
```

```
End
```

۳-۱-۲. ASM ... ENDASM : وارد کردن یک بلوک از دستورات اسمبلی

ترکیب :

```
ASM
```

```
/
```

دستورات اسمبلی

```
/
```

```
ENDASM
```

شرح :

دستورات بین ASM و ENDASM از نوع اسمبلی می باشد. حداکثر کد اسمبلی نوشته شده به مقدار حافظه قابل برنامه نویسی آن میکروکنترلر وابسته است. مثلا در میکروکنترلر PIC16F877 کد اسمبلر حداکثر 8K می باشد.

مثال :

Main:

```
asm
    bsf    PORTA,0    ' نشاندن RA0
    bcf    PORTB,3    ' باز نشاندن RB0
endasm
Loop: goto Loop
end
```

۳-۳-۱. ADCIN دریافت مقادیر از ورودی مبدل A/D

ترکیب :

ADCIN *channel, variable*

شرح :

ADCIN تبدیل آنالوگ به دیجیتال (A/D) را از یک سیگنال آنالوگ ورودی در میکروکنترلرهای که مبدل A/D دارند، انجام دهد (مانند PIC16F877) مقدار خوانده شده در داخل متغیر بکار رفته ذخیره می شود. قبل از استفاده از دستور ADCIN رجیستر TRIS مربوطه بایستی مقداردهی اولیه شود تا پایه مورد استفاده بطور مشخص بصورت ورودی در آید. بعلاوه اینکه از رجیستر ADCON1 برای مشخص کردن اینکه پینهای ورودی به حالت آنالوگ باشد یا دیجیتال استفاده می شود. برای تنظیم A/D علاوه بر استفاده از DEFINE می توان بصورت مستقیم رجیسترهای مربوط به A/D را مقدار دهی کرد.

مثال :

```
DEFINE ADC_BITS 8          ' نتایج تبدیل شده ۱۰،۸ یا ۱۲ بیتی می باشد
DEFINE ADC_CLOCK 3         ' کلاک مبدل آنالوگ به دیجیتال
DEFINE ADC_SAMPLEUS 10    ' زمانبندی نمونه برداری مورد نظر
B0 var byte
Main :
    TRISA = $FF            ' همه پینهای پورت A ورودی می باشند
    ADCON1 = 0            ' پورت A در حالت ورودی آنالوگ است
    adcin 0, B0           ' خواندن کانال ۰ و ذخیره نتایج داخل متغیر B0
Loop : goto Loop
end
```


۴-۳-۱. BRANCH پریدن به برجسب مشخص شده توسط شاخص

ترکیب :

BRANCH *index*, [*label1* {*label...*}]

شرح :

با توجه به مقدار شاخص پرش به برجسب هم نظیر آن انجام می گیرد. مثلا اگر شاخص صفر باشد پرش به اولین برجسب قرار گرفته در براکت صورت می گیرد و اگر ۱ باشد پرش به دومین برجسب صورت می گیرد و اگر شاخص برابر تعداد برجسبها یا بیشتر از آن باشد هیچ پرشی صورت نمی گیرد و دستور بعدی اجرا می شود.

در ضمن شاخص یک متغیر یک بیتی است و حداکثر برای ۲۵۵ برجسب این دستور قابل اجرا است. (برای ۱۸CXXX ۲۵۶ برجسب)

با استفاده از IF ... THEN نیز شبیه دستور BRANCH عمل کرد.

```
If B0 = 0 then lab1
If B0 = 1 then lab2
If B0 = 2 then lab3
```

مثال:

```
B0 var byte
Branch B0,[ lab1 , lab2 , lab3 ]
Loop: goto Loop
Lab1:
Lab2:
Lab3:
end
```

۵-۳-۱. BRANCHL پرشی بلند به برجسب نظیر شاخص

ترکیب :

BRANCHL *index*, [*label1* {*label...*}]

شرح :

این دستور کاملا شبیه دستور BRANCH می باشد تنها تفاوت آن این است که BRANCHL پرش به موقعیتهای قرار گرفته در کد سگمنتهای دیگر را نیز محقق بسازد .

کد بدست آمده از BRANCHL تقریبا دو برابر بزرگتر از کد BRANCH می باشد. اگر پرشها تنها قرار است در یک کد سگمنت روی بدهد و یا کد برنامه در یک کد سگمنت می باشد و یا حافظه میکروکنترلر مورد نظر کمتر از یک کد سگمنت (کمتر از ۲ کیلو بایت) باشد بهتر است از BRANCH استفاده شود زیرا حافظه استفاده شده برای کدهای برنامه را کاهش می دهد.

۱۲۷ برجسب (۲۵۶ تا برای ۱۸CXXX) می توان با دستور BRANCHL آدرس دهی کرد.

مثال:

```
B0 var byte
Main:
branchl B0 , [ Lab1 , Lab2 , Lab3 ]
```

```

Loop:
goto Loop
Lab1: /
/
Lab2:/
/
Lab3:/
/
end

```

۶-۳-۱. BUTTON : خواندن حالت دکمه روی پین ورودی

ترکیب :

BUTTON *Pin, State, Delay, Speed, Variable, Action, Label*

شرح :

دستور **BUTTON** تماس لرزان ناشی از فشردن دکمه را حذف می کند (Debouncing) این حالت را توسط تاخیر در اجرای برنامه در هر بار تکرار انجام می دهد تاخیر Debouncing پیش فرض ۱۰ میلی ثانیه می باشد که برای تغییر آن از **DEFINE BUTTON_PAUSE** **TIME(ms)** استفاده می کنیم . مثلا دکمه با تاخیر Debouncing ۵۰ میلی ثانیه بصورت زیر تعیین می شود .

```
DEFINE BUTTON_PAUSE 50
```

تکرار خودکار که توسط **Speed** تعیین می شود در حقیقت مدت فشرده نگه داشتن کلید را تعیین می کند. دستور **BUTTON** معمولا در یک حلقه به کار می رود تا زده شدن کلید را چک کند .

pin : شماره پایه را نشان می دهد که ممکن است یک عدد بین ۰ تا ۱۵ یا یک متغیر که مقدارش بین ۰ تا ۱۵ می باشد یا اسم پین (مانند PORTA.0) باشد .
State : وضعیت پایه به هنگام فشار دادن کلید (۰ یا ۱)
Delay : میزان تاخیر قبل از شروع تکرار خودکار (۰ تا ۲۵۵)
Speed : نرخ تکرار خودکار (۰ تا ۲۵۵)
Variable : متغیر یک بیتی مورد نیاز برای تاخیر تکرار . قبل از استفاده باید با عدد صفر مقدار دهی اولیه شده باشد.

Action : وضعیت پایه برای اجرای فرمان **GOTO** .

"۰" : اگر فشار داده نشده باشد.

"۱" : اگر فشار داده شده باشد.

Lable : نقطه ای اجرای برنامه در صورت صحیح بودن **Action** ، باید از آنجا ادامه پیدا کند.

مثال :

```
DEFINE    BUTTON_PAUSE  50
TRISA = 0
TRISB = 255
B0  var  byte
Main:
    B0 = 0
    Button  PORTB.0 , 0 , 100 , 10 , B0 , 1 , Led
    Goto    Main
Led:
    Toggle  PORTA.0
    Goto    Main
End
```

۳-۷-۱ . CALL : فراخوانی زیر برنامه های اسمبلی

ترکیب :

CALL *label*

شرح :

زیر برنامه با نام *label* را در زبان اسمبلی فراخوانی می کند.

مثال:

در این مثال فایل "init.asm" به برنامه الحاق شده است.

```
@ include  "init.asm"
Main:
    Call  init_sys
Loop:
    Goto  Loop
End
```

۳-۸-۱ . CLEAR : مقدار همه متغیرها را به صفر تغییر دادن

ترکیب :

CLEAR

شرح :

با دستور CLEAR تمام رجیسترهای RAM در همه دیتا بانکها به صفر تغییر می یابد. این دستور،

برای اینکه تمام متغیرها را بطور همزمان صفر کنیم مناسب است.

مثال :

```
Clear
Main:
    Goto  Main
End
```

۹-۳-۱. CLEARWDT : باز نشاندن تایمر Watchdog (سگ نگهبان)

ترکیب :

```
CLEARWDT
```

توضیح :

باز نشاندن تایمر Watchdog را برعهده دارد . این دستور می تواند بعد از راه اندازی تایمر Watchdog ، از هنگ شدن میکروکنترلر جلوگیری کند.

مثال :

```
Clearwdt
```

```
Main:
```

```
    Goto  Main
```

```
End
```

۱۰-۳-۱. COUNT : شمردن پالسهای روی پین ورودی

ترکیب :

```
COUNT Pin , Period , No_Impulses
```

شرح :

این دستور تعداد ضربه های روی پین مورد نظر را در مدت زمان تعریف شده (Period) می شمرد و در متغیر No_Impulse ذخیره می کند . پین بطور اتوماتیک بصورت ورودی طراحی می شود. Period بر حسب میلی ثانیه تعریف می شود . اگر اسیلاتور ۴ مگاهرتز باشد، چک کردن وضعیت پین هر ۲۰ میکرو ثانیه انجام می گیرد. اگر خواسته باشیم با اسیلاتور دیگری کار کنیم باید از DEFINE OSC استفاده کنیم .

Pin ممکن است یک عدد بین ۰ تا ۱۵ یا یک متغیر که مقدارش بین ۰ تا ۱۵ است یا اسم پین (مانند PORTA.0) باشد.

در این روش ما به آسانی فرکانس یک سیگنال را بصورت ساده ای از روی تعداد ایمپالسها محاسبه می کنیم. بالاترین فرکانس قابل محاسبه با اسیلاتور ۴ مگاهرتز هست ۲۵ کیلو هرتز است و اگر اسیلاتور ۲۰ مگاهرتز استفاده شود تا ۱۲۵ کیلو هرتز را می توان محاسبه کرد .

مثال :

```
W0 var byte
```

```
TRISA = $FF
```

```
Main:
```

```
    Count PORTA.0,1000,W0 ' شمردن ضربه ها در یک ثانیه'
```

```
    PORTB = W0
```

```
    Goto  Main
```

```
End
```

ترکیب :

{Lable} DATA {@PoCadr},constant, constant,...

شرح :

DATA مقادیر ثابت را در داخل EEPROM داخلی در ابتدای نوشتن کد میکروکنترلر ذخیره می کند. اگر آدرس در دستور مورد نظر حذف شده باشد ثابتها از آدرس صفر ذخیره می شود . ثابت ممکن است از نوع INTEGER یا STRING باشد یک ثابت STRING بصورت بایتهای متوالی از مقادیر کد ASCII ذخیره می شود . اگر لازم باشد که یک عدد دو بیتی در EEPROM ذخیره شود بایستی از کلمه "WORD" قبل از ثابت استفاده کرد در غیر اینصورت ، تنها بایت با ارزش کمتر ذخیره می شود. دستور DATA تنها در میکروکنترلر های PIC که دارای EEPROM داخلی می باشد مانند PIC16F84 یا سری 16F87X قابل اجرا است. DATA تنها یک مرتبه در فضای حافظه EEPROM می تواند ذخیره شود و آن هم در زمان برنامه ریزی میکروکنترلر می باشد و نه هر زمانی که برنامه اجرا می شود . دستور WRITE برای قرار دادن مقادیر بر روی EEPROM داخلی در هنگام اجرای برنامه (RUNTIME) می تواند استفاده شود.

مثال :

data @5,1,2,3

نوشتن مقدارهای ۱ ، ۲ و ۳ روی موقعیتهای ۵ ، ۶ و ۷ حافظه EEPROM

data word \$1234

نوشتن مقدارهای \$12 و \$34 در موقعیتهای ۰ و ۱ حافظه EEPROM

Data (4),0(10)

پریدن به موقعیت چهارم و ذخیره 10 0s

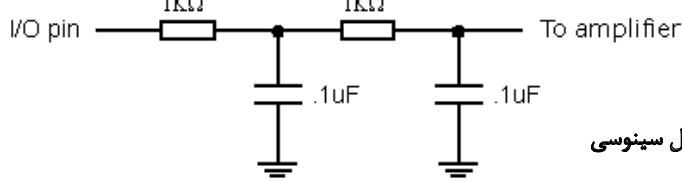
۱۲-۳-۱. DTMFOUT : تولید کردن سیگنال صدا (تون) شمارگیری روی پین خروجی

ترکیب :

DTMFOUT PIN , {ONMS,OFFMS},{TONE{TONE...}}

شرح :

دستور DTMFOUT تون شمارگیری را (مثلا برای تلفنها) تولید می کند . پارامتر ONMS مدت زمان هر شمارگیری را نشان می دهد. و OFFMS مدت قطع بین دو صدای پی درپی را مشخص می کند . اگر پارامترهای فوق مقدارگذاری نشوند ONMS ۲۰۰ میلی ثانیه و OFFMS ۵۰ میلی ثانیه مقدار گذاری می شوند . این دستور برای ۱۶ شماره صدا تولید می کند که برای شمارههای ۰ تا ۹ صدایی مطابق با صدای شمارگیر تلفن تولید می کند و صدای ۱۰ همان صدای کلید * بر روی کیبورد (key board) تلفن می باشد و ۱۱ همان صدای کلید # را تولید می کند و ۱۲ تا ۱۵ صدای مشابه کلیدهای A تا D تولید می کنند .



شکل 3-1) فیلتر برای بدست آوردن شکل سینوسی

برای بدست آوردن شکل سینوسی در خروجی یک فیلتر مناسب (تطبیقی) احتیاج می باشد تا تعدادی از هارمونیکها را از بین ببرد و سینوسی را صافتر کند .
 DTMFOUT با اسیلاتور ۲۰ مگاهرتز بهتر کار می کند البته با اسیلاتور ۴ مگاهرتز نیز کار می کند ولی فیلتر کردن آن و تبدیل آن به سینوسی سختتر است. دستور DTMFOUT از تابع FREQOUT برای تولید صدای شمارگیری استفاده می کند. FREQOUT صدای مورد نظر را بصورت مدولاسیون پهنای پالس تولید می کند.
 مثال :

```

TRISB = $FF
Main:
    Dtmfout PORTB.1,[2,1,2]
Loop:
    goto Loop
End
    
```

۱۳-۳-۱. EEPROM: مجموعه ثابتهای اولیه برای برنامه نویسی EEPROM

ترکیب:

```
EEPROM { @location, } constant {, constant}
```

شرح :

مجموعه ای از ثابتها را در EEPROM داخلی میکروکنترلر قرار می دهد . این دستور تنها در ابتدا و یا انتهای برنامه قرار می گیرد و برای خواندن و نوشتن در وسط برنامه و در حال اجرا از دستورات READ و WRITE استفاده می شود.

این دستور در میکروکنترلرهای که با I2C با EEPROM خارجی ارتباط برقرار می کنند قابل اجرا نیست .

مثال :

```
EEPROM @5,1,2,3
```

Writes in the values 1, 2, 3 on the locations 5, 6 and 7 in EEPROM memory.

```
EEPROM word $1234
```

Writes in the values \$12 AND \$34 on the locations 0 and 1 in EEPROM memory.

EEPROM @5,1,2,3

مقادیر ۱، ۲ و ۳ را در موقعیتهای ۵، ۶ و ۷ EEPROM قرار می دهد.

EEPROM @5,1,2,3

مقادیر \$12 و \$34 را در موقعیتهای ۰ و ۱ EEPROM قرار می دهد.

۱۴-۳-۱. END : نشانه گذاری پایان برنامه

ترکیب :

END

شرح :

از اجرای بیشتر برنامه جلوگیری می کند و برنامه را به حالت کم مصرف در می آورد شبیه دستور SLEEP که در یک حلقه قرار گرفته است. این دستور در انتهای برنامه نوشته می شود.

۱۵-۳-۱. FOR ... NEXT : تکرار کردن یک قسمت از برنامه

ترکیب :

```
FOR Index = Start TO End {Step {-} Inc }  
{ instructions,  
instructions }  
NEXT {Index}
```

شرح :

"Index" متغیری است که برای کنترل چگونگی تعداد بارهای که حلقه اجرا می شود استفاده می شود. اگر پارامتر step استفاده نشود متغیر یکی یکی افزایش می یابد (index = index + 1)

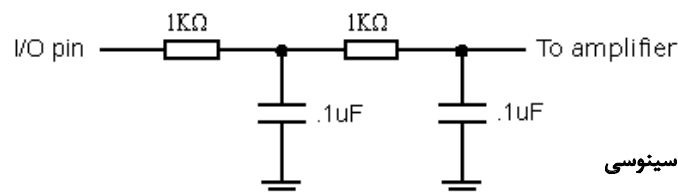
۱۶-۳-۱. FREQOUT : تولید سیگنال با فرکانس مشخص روی پین خروجی

ترکیب :

FREQOUT Pin, Onms, Freq1, Freq2

شرح :

FREQOUT سیگنال PWM با رنج فرکانسی از ۰ تا ۳۲۷۶۷ هرتز روی پین تعریف شده در پارامتر " pin " و با مدت زمان " onms " تولید می کند . FREQOUT با اسیلاتور 20MHz بهتر کار می کند. "onms" مدت زمان سیگنال بر حسب میلی ثانیه را نشان می دهد.



شکل 4-1) فیلتر برای بدست آوردن شکل سینوسی

برای بدست آوردن سیگنال سینوسی مطلوب در خروجی نصب یک فیلتر لازم است.
مثال :

freqout PORTB.1,2000,1000

سیگنال با فرکانس 1000Hz در مدت ۲ ثانیه در پین 1 پورت B تولید می کند.

۱۷-۳-۱ . **GOSUB** : فراخوانی زیر برنامه های BASIC

ترکیب :

GOSUB *label*

شرح :

با این دستور برنامه به زیر برنامه ای که بین " label " و دستور RETURN قرار دارد می رود. زیر برنامه ها می توانند تودرتو باشند. این حالت تا چهار سطح می تواند عمق داشته باشد و دلیل آن محدود بودن پشته می باشد.

مثال :

Main :

gosub Blink

' Blink فراخوانی زیر برنامه

Loop: goto Loop

Blink:

' Blink زیر برنامه

PORTB = \$FF

Pause 1000

PORTB = \$00

Pause 1000

Return

End

۱۸-۳-۱ . **GOTO** : ادامه برنامه از برچسب مشخص شده

ترکیب :

GOTO *label*

شرح :

این برنامه پرش به هر جایی از برنامه را ممکن می سازد . تکرار زیاد این دستور در برنامه توصیه نمی شود زیرا قابل فهم بودن برنامه را کمتر می کند.

۱۹-۳-۱ . **HIGH** : نشاندن یک منطقی روی پین خروجی

ترکیب :

HIGH *Pin*

شرح :

پین مورد نظر را به سطح بالا می آورد. این دستور بطور خودکار پین را در گرایش خروجی قرار می دهد. پین ممکن است یک عدد ثابت بین ۰ تا ۱۵ یا یک عدد متغیر که محتوای آن بین ۰ تا ۱۵ و یا اسم پورت باشد. اگر پین مورد نظر در گرایش خروجی باشد استفاده از عبارت زیر مناسبتر است.

PORTB.0 = 1.

Main:

```
high PORTA.0 ' Pin RAO set on the high level
```

```
Loop: goto Loop
```

```
End
```

مثال :

۲۰-۳-۱. HSERIN :

سخت افزار ورودی آسنکرون

ترکیب :

HSERIN {Error,}{Timeout, Label,}[Modifier(,...)]

شرح :

HSERIN اطلاعات را بصورت سریال دریافت می کند. این دستور با میکروکنترلرهای که دارای واحد سخت افزاری ارتباط سریال (سخت افزار USART) می باشند، بکار می رود (مانند میکروکنترلر 16F877). پارامترهای انتقال سریال با دستور DEFINE تعیین می شود.

```
DEFINE HSER_RCSTA 90h ' Setting of a receiving register
```

```
DEFINE HSER_TXSTA20h ' PSetting of a emitting register
```

```
DEFINE HSER_BAUD 2400 ' Flow in bauds
```

```
DEFINE HSER_SPBRG 25 ' Direct setting of the SPBRG register
```

عملکرد HSERIN با اسیلاتور 4MHz از پیش تعریف شده است و برای کار کردن با اسیلاتورهای دیگر بایستی از دستور اسیلاتور مورد نظر را در ابتدای برنامه تعریف کرد.

| اصلاح کننده | چگونگی کارکرد |
|----------------------------|--|
| BIN {1..16} | گرفتن رقمهای باینری |
| DEC {1..5} | گرفتن رقمهای دسیمال |
| HEX {1..4} | گرفتن رقمهای هگزا دسیمال |
| SKIP n | نگرفتن کرکتر بعد از کرکتر nم |
| STR ArrayVar\n{c} | گرفتن متوالی n کرکتر تا زمانی که کرکتر c (اختیاری) بیاید |
| WAIT () | انتظار برای کرکتر های متوالی |
| WAITSTR ArrayVar{n} | انتظار برای رشته |

DEFINE OSC tact ' تعیین فرکانس مورد نظر

پارامترهای " Time out " و " Label " موجب می شوند که اگر هیچ کرکتری دریافت نشد برنامه بعد از زمان " Time out " (بر حسب میلی ثانیه) از برجسب Label ادامه می یابد. قالب بندی اطلاعات سریال می تواند 8N1 (۸ بیت اطلاعات بدون بیت توازن وبا تنها یک بیت توقف) و 7E1 (۷بیت از اطلاعات ، بیت توازن و یک بیت توقف) استفاده شود که با دستور DEFINE تعیین می شود.

```
DEFINE HSER_EVEN 1 ' چک کردن parity
```

```
DEFINE HSER_ODD 1 ' Non_parity چک کردن
```

برنامه ممکن است دارای یک برجسب "Error" باشد که با توجه به خطا و درست نبودن پریتی به آن برجسب می پرد. اصلاح کننده ها در جدول زیر آمده است.

مثال:

```
B0 var byte
```

```
W1 var word
```

```
Main:
```

```
hserin [B0, dec W1] 'Take dec. digit from serial line
```

```
goto Main
```

```
end
```

۲۱-۳-۱. HPWM : تولید کردن سیگنال PWM روی پین میکروکنترلر

ترکیب :

HPWM Channel,Relation_on_off, Frequency

شرح :

این فرمان از سخت افزار PWM روی میکروکنترلر های که دارای این امکان هستند استفاده کرده است و سیگنال PWM را تولید می کند.

پارامتر "channel" کانال PWM را تعریف می کند تعداد این کانالها بستگی به نوع میکروکنترلر دارد و بین ۱ تا ۳ کانال است . برای دو کاناله ها هنگام راه اندازی هر دو کانال فرکانس آنها بایستی یکسان باشد. Relation_on_off نسبت بین سیگنال on و off را مشخص می کند و مقدار صفر سیگنال خاموش را روی پین می فرستد و مقدار ۲۵۵ سیگنال همیشه روشن را روی پین می فرستد. پارامتر "Frequency" فرکانس سیگنال PWM را مشخص می کند. (کمترین فرکانس در اسیلاتور 4MHz ، 245Hz و بالاترین فرکانس ممکن در اسیلاتور سریع ۳۲۷۶۷ هرتز است). HPWM بطور پیوسته در زمان اجرای برنامه کار می کند. تعدادی از قطعات ممکن است پینهای چند کار که HPWM را دارند داشته باشند . DEFINE اجازه استفاده از این پینها را در این راستا می دهد.

```
DEFINE CCP1_REG PORTC 'Hpwm 1 pin port
```

```
DEFINE CCP1_BIT 2 'Hpwm 1 pin bit
```

DEFINE CCP2_REG PORTC 'Hpwm 2 pin port
 DEFINE CCP2_BIT 1 'Hpwm 2 pin bit

تعریفهای زیر نشان می دهند که تایمر ۱ یا ۲ برای تولید PWM روی کانال ۲ یا ۳ صورت بگیرد.

DEFINE HPWM2_TIMER 1 'انتخاب تایمر برای Hpwm 2

DEFINE HPWM3_TIMER 1 'انتخاب تایمر برای Hpwm 3

مثال :

DEFINE HPWM2_TIMER 1 'کانال دوم با استفاده از تایمر ۱
 hpwm 2, 64, 1000 ' 25% PWM on 1kHz

۲۲-۳-۱. HSEROUT : سخت افزار خروجی آسنکرون

ترکیب :

HSEROUT [Item{,Item...}]

شرح :

HSEROUT اطلاعات را از طریق سخت افزار USART به صورت سریال می فرستد. پارامترهای انتقال بوسیله DEFINE تعریف می شود.

DEFINE HSER_RCSTA 90h 'قرار دادن مقدار مناسب در رجیستر دریافت
 DEFINE HSER_TXSTA 20h 'قرار دادن مقدار مناسب در رجیستر انتشار
 DEFINE HSER_BAUD 2400 'سرعت انتقال (Baud rate)
 DEFINE HSER_SPBRG 25 'قرار دادن مقدار مناسب بطور مستقیم در SPBRG

این سرعتهای انتقال که تعریف شد برای میکروکنترلری با اسیلاتور 4MHz می باشد و برای دیگر اسیلاتورها بایستی تعریف در ابتدای برنامه صورت بگیرد. قالب بندی های مختلف نیز با دستور DEFINE تعریف می شود.

DEFINE HSER_EVEN 1 'چک کردن parity
 DEFINE HSER_ODD 1 'چک کردن Non_parity

| اصلاح کننده ها | فرستاده شده |
|-------------------|------------------------------|
| {I}{S} BIN{1..16} | مقدار binary |
| {I}{S} DEC{1..5} | مقدار decimal |
| {I}{S} HEX{1..4} | مقدار hexadecimal |
| REP c/n | کرکنتر C, n بار تکرار می شود |
| STR ArrayVar {n} | n کرکنتر رشته |

جدول ۷-۱

مثال :

B0 var byte
 B0 = 4

Main :

hserout [dec B0, 10] مقدار ۱۰ و مقدار B0 از متغیر

Loop: goto Loop

End

۲۳-۳-۱. I2C READ : خواندن اطلاعات از وسایل جانبی I2C

ترکیب :

I2C READ Data, Frequency, Control_byte, {Address,} {Variable, Variable...} {Label}

شرح :

اطلاعات آدرس و کنترل ارسال شده از خط I2C در متغیر "Variable" ذخیره می شود .
I2C READ و I2C WRITE برای خواندن و نوشتن اطلاعات بر روی واحدهای جانبی استفاده می
شود. این دستورها می توانند ارتباط با وسایل که دارای خط ارتباطی I2C هستند ارتباط برقرار کند
از قبیل سنسورهای دما ، A/D ها ، EEPROM های سریال و غیره.

۷ بیت بالاتر کد کنترلی برای انتخاب چیپ مورد نظر بکار می رود بیت پایینتر (LSB) روش جاری
(خواندن یا نوشتن) را تعیین می کند. برای مثال برای ارتباط با EEPROM سریال 24LC01B ،
آدرس درخواستی ۸بیت است که کد کنترلی 1010% است و انتخاب چیپ در اینجا بکار نمی آید
پس بیت کنترلی ممکن است 10100000% باشد.

قالب بندی چندین EEPROM سریال در زیر نشان داده شده است.

| EEPROM | ظرفیت | کلمه کنترلی | اندازه آدرس |
|---------|-----------|-------------|-------------|
| 24LC01B | 128 bytes | %1010xxx0 | 1 byte |
| 24LC02B | 256 bytes | %1010xxx0 | 1 byte |
| 24LC04B | 512 bytes | %1010xxb0 | 1 byte |
| 24LC08B | 1K bytes | %1010xbb0 | 1 byte |
| 24LC16B | 2K bytes | %1010bbb0 | 1 byte |
| 24LC32B | 4K bytes | %1010ddd0 | 2 bytes |
| 24LC65 | 8K bytes | %1010ddd0 | 2 bytes |

bbb = انتخاب بلوک

ddd = بیت‌های انتخاب وسیله

xxx = بدون اثر

جدول ۱-۸

اگر اطلاعات دو بیتی (WORD) دریافت شود ابتدا بایت بالاتر دریافت می شود و بعد بایت پایینتر.
برای دریافت رشته ، STR بعد اسم رشته می آید و تعداد کرکترها بعد از ۱ می آید.

a var byte[8]

I2C READ PORTC.4, PORTC.3, \$a0, 0, [STR a\8]

اگر از بر چسب استفاده شود ، در صورت دریافت نکردن چیزی بر روی مدار واسط I2C به آن
برچسب می پرد. سرعت انتقال استاندارد 100KHZ می باشد که با اسیلاتور 8MHZ بدست می آید و

برای سرعت‌های بالاتر (400KHZ) از اسیلاتور 20MHZ استفاده می شود. اگر از اسیلاتور کندتر استفاده شود بایستی دستورالعمل زیر برای تعریف آن بکار رود.

```
DEFINE I2C_SLOW 1
```

مثال :

```
B0 var byte
addr var byte
cont con %10100000 ' آدرس کنترلی EEPROM
addr = 17 ' اطلاعات از آدرس ۱۷ خوانده می شود
```

Main:

```
I2C_READ PORTA.0, PORTA.1, cont, addr, [B0] ' قرار دادن اطلاعات در B0
```

```
Loop: goto Loop
```

```
End
```

۲۴-۳-۱ . I2CWRITE : نوشتن اطلاعات بر روی وسایل جانبی I2C

ترکیب :

```
I2CWRITE Data, Frequency, Control_byte, {Address,} [Vari {, Vari...}] {Label}
```

شرح :

I2CWRITE اطلاعات کنترل و آدرس را توسط مدار واسط I2C می فرستد. اگر وسیله جانبی EEPROM سریال باشد ، انتظار ۱۰ میلی ثانیه برای پایان گرفتن نوشتن لازم است. اگر قبل از این انتظار اطلاعاتی را بفرستیم از نوشتن آن جلوگیری می شود. اندازه آدرس ۱ یا ۲ بایت می باشد (وابسته به وسیله ای که متصل شده است) . اگر ما از وسایل بدون تاخیر استفاده کنیم و دوبایت (WORD) را به سوی آن بفرستیم ابتدا بایت بالاتر و سپس بایت پایینتر فرستاده می شود. برای انتقال رشته ، STR قبل از اسم رشته می آید و تعداد کرکترها بعد از " \ " می آید.

```
a var byte[8]
```

```
I2CWRITE PORTC.4, PORTC.3, $a0, 0, [STR a\8]
```

اگر سیگنال پاسخی روی مدار واسط I2C وجود نداشته باشد برنامه به برجسب اختیاری خواهد پرید . سرعت انتقال استاندارد 100KHZ می باشد که با اسیلاتور 8MHZ بدست می آید و برای سرعت‌های بالاتر (400KHZ) از اسیلاتور 20MHZ استفاده می شود. اگر از اسیلاتور کندتر استفاده شود بایستی دستورالعمل زیر برای تعریف آن بکار رود.

```
DEFINE I2C_SLOW 1
```

برای داشتن کلاک مدار واسط I2C دو قطبی (Bipolar) و نه یک open collector دستور العمل DEFINE به صورت زیر استفاده شود.

```
DEFINE I2C_SCLOUT
```

برای عملکرد واحدهای جانبی دارای ارتباط I2C بایستی راهنما و مشخصات سازنده خوانده شود تا بیهیهای کنترلی و تاخیرها مناسب بدست آید.

مثال :

```
B0 var byte
```

```
addr var byte
```

```
cont con %10100000 ' آدرس کنترلی EEPROM
```

```
Main:
```

```
addr = 17 ' اطلاعات در آدرس ۱۷ ریخته می شود
```

```
i2cwrite PORTA.0, PORTA.1, cont, addr, [6] ' مقدار ۶ در آدرس ۱۷ قرار می گیرد
```

```
pause 10 ' انتظار ۱۰ میلی ثانیه تا نوشتن تمام شود
```

```
addr = 1 ' اطلاعات در آدرس ۱ ریخته می شود
```

```
B0 = 23
```

```
i2cwrite PORTA.0, PORTA.1, cont, addr, [B0] ' مقدار B0 در آدرس ۱ قرار می گیرد
```

```
pause 10 ' انتظار ۱۰ میلی ثانیه تا نوشتن تمام شود
```

```
Loop: goto Loop
```

```
End
```

۲۵-۳-۱. INPUT : برگزیدن پین I/O در گرایش ورودی

ترکیب :

```
INPUT Pin
```

شرح :

این دستور گرایش ورودی یک پین را تعیین می کند ما می توانیم همه پینها و یا یک پین را از طریق مقدار گذاری در TRIS بدست آوریم

مثال :

```
TRISB = %11111111
```

```
TRISA.1 = 1
```

۲۶-۳-۱. IF ... THEN ... ELSE : گزینش یک قسمت از برنامه

ترکیب :

```
IF Expression1 { AND / OR Expression2} THEN Label
```

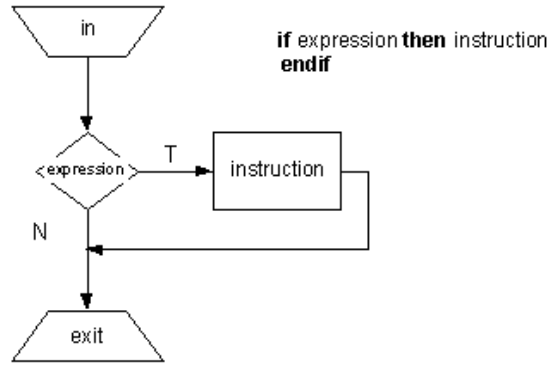
```
{instructions}
```

```
ELSE
```

```
{instructions}
```

```
ENDIF
```

شرح : این دستور یکی از دو امکان مسیر برنامه را انتخاب می کند .



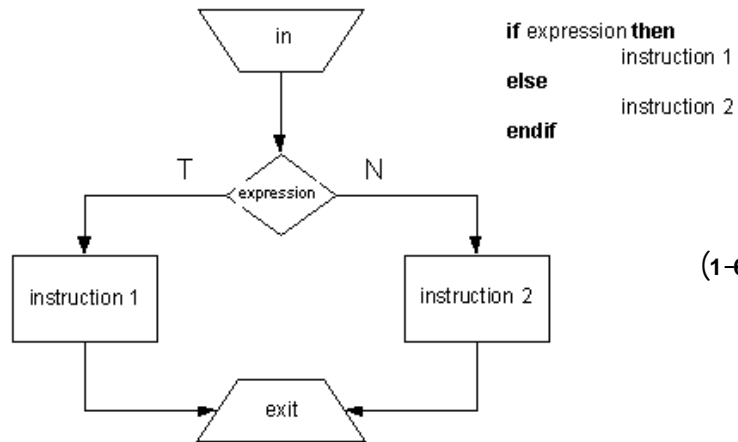
شکل 5-1 ()

دستور IF دستور اصلی شاخه گزینی در PIC BASIC و آن می تواند به چندین طریق برای انعطاف پذیری در فهم ساخت یک تصمیم منطقی استفاده می شود .
مثال برای شکل ساده این دستور :

```

w var byte
Main :
  IF PORTB.0=0 THEN Add
  goto Main
Add : W=W+1
End
  
```

شکل پیچیده تر این دستور استفاده از ELSE می باشد.



شکل 6-1 ()

```

if expression then
  instruction 1
else
  instruction 2
endif
  
```

w var byte

Main :

```

IF (PORTB = $F0) && (PORTA.0 = 1) THEN Add
ELSE Subtract
ENDIF
goto Main
Add : W=W+1
Subtract : W=W-1
End

```

برنامه بالا را می توان بصورت زیر نیز نوشت :

w var byte

Main :

```

IF PORTB.0=0 THEN W=W+1
ELSE W=W-1
ENDIF
goto Main
End

```

۲۷-۳-۱ . LCDOUT : نوشتن اطلاعات بر روی نمایشگر LCD

ترکیب :

LCDOUT Data {, Data...}

شرح :

LCDOUT اطلاعات را به LCD (Liquid Crystal Display) می فرستد. PIC BASIC مدلهای گوناگون LCD که کنترلر Hitachi 44780 را دارند و یا شبیه آن عمل می کنند را پشتیبانی می کند. LCD معمولا ۱۴ یا ۱۶ پایه دارد که به میکروکنترلر متصل می شوند. اگر کرکتر " # " قبل از اطلاعات فرستاده شود مقدار ASCII هر یک از اطلاعات را می فرستد. LCDOUT اصلاح کننده های در جدول زیر آمده است .

| اصلاح کننده ها | فرستاده شده |
|-------------------|-----------------------------|
| {I}{S} BIN{1..16} | مقدار binary |
| {I}{S} DEC{1..5} | مقدار decimal |
| {I}{S} HEX{1..4} | مقدار hexadecimal |
| REP c/n | کرکتر c، n بار تکرار می شود |
| STR ArrayVar {n} | n کرکتر رشته |

جدول ۹-۱

نمایشگر LCD می تواند با باس ۴بیتی یا ۸بیتی به یک میکروکنترلر متصل شود. اگر از باس ۸بیتی استفاده شود همه ۸بیت بایستی به یک پورت وصل شوند. وقتی در حالت باس ۴بیتی هر ۴بیت می تواند یا به ۴بیت پایین و یا بالای پورت متصل شود. اگر از LCD تنها برای نمایش استفاده می شود بایستی خط R/W به زمین متصل شود در حالت پیش فرض پایه های LCD در حالت ۴بیتی (DB4 – DB7) به RA0-RA3 متصل می شوند. بیت RS به RA4 متصل می شود و بین E به RB3 وصل می شود. در این حالت LCD، ۱۶ * ۲ در نظر گرفته می شود برای تغییر در هر یک از وضعیتها بالا بایستی از دستور DEFINE استفاده شود.

| | | |
|---------------------|-------|---------------------------------------|
| DEFINE LCD_DREG | PORTB | انتخاب پورت |
| DEFINE LCD_DBIT | 4 | انتخاب ۰ یا ۴ در حالت باس ۴ بیتی |
| DEFINE LCD_RSREG | PORTB | انتخاب پورت برای اتصال پین RS |
| DEFINE LCD_RSBIT | 1 | انتخاب بیت برای RS |
| DEFINE LCD_EREG | PORTB | انتخاب پورت برای اتصال پین E |
| DEFINE LCD_EBIT | 0 | انتخاب بیت برای E |
| DEFINE LCD_BITS | 4 | انتخاب باس ۴ بیتی یا ۸ بیتی |
| DEFINE LCD_LINES | 2 | تعیین تعداد خطهای LCD |
| DEFINE LCD_COMMANDS | 2000 | تاخیر ارسال دستور برحسب میکرو ثانیه |
| DEFINE LCD_DATAUS | 50 | تاخیر ارسال اطلاعات برحسب میکرو ثانیه |

تعریفهای بالا، معرفی می کنند LCD ۲خطی را با باس ۴بیتی که به ۴بیت بالای پورت B متصل است و پایه انتخاب رجیستر (RS) به PORTB.1 و پایه فعال سازی به PORTB.0 متصل است. هر کنترلر LCD دارای دستورات معینی است. این دستورات بصورت LCDOUT \$FE, \$Kod فرستاده می شوند. این دستورات در جدول زیر نشان داده شده است.

| دستورات | عملکرد |
|------------|---|
| \$FE, 1 | پاک کردن صفحه نمایش |
| \$FE, 2 | شروع از ابتدای خط اول |
| \$FE, \$0C | خاموش کردن مکان نما |
| \$FE, \$0E | روشن کردن مکان نمای از نوع خط زیر (_) |
| \$FE, \$0F | روشن کردن مکان نمای از نوع خالی |
| \$FE, \$10 | شیفت مکان نما به چپ |
| \$FE, \$14 | شیفت مکان نما به چپ |
| \$FE, \$C0 | قرار دادن مکان نما در ابتدای خط دوم |
| \$FE, \$94 | قرار دادن مکان نما در ابتدای خط سوم |
| \$FE, \$D4 | قرار دادن مکان نما در ابتدای خط چهارم |

مثال :

B0 var byte

B1 var byte

Main:

lcdout \$FE, 1, "Hello" ' پاک کردن صفحه نمایش و نوشتن "Hello"

lcdout \$FE, \$C0 ' رفتن به خط دوم

lcdout B0 ' نمایش مقدار B0

lcdout #B1 ' نمایش کد اسکی متغیر B1

Loop: goto Loop

End

۲۸-۳-۱. LOOKDOWN : جستجو کردن جدول ثابتها

ترکیب :

LOOKDOWN Value, [Const {, Const...}], Var

شرح :

دستور LOOKDOWN وجود مقدار معینی را در فهرستی از ثابتها جستجو می کند اگر value در فهرست وجود داشته باشد شاخص اختصاص داده شده به ثابت (شماره ثابت) در Var ذخیره می شود. اگر ثابت اولی با مقدار value یکی باشد مقدار ۰ در Var ذخیره می شود. و اگر ثابت دومی از فهرست جور شود مقدار ۱ ذخیره می شود و غیره ... اگر مقدار value در فهرست نباشد مقدار Var تغییر نمی کند. فهرست ثابتها می تواند هم عددی و هم کارکتری باشد. هر کارکتر رشته را بصورت یک مقدار ASCII رفتار می کند. تعداد مجاز ثابتها ۲۵۵ تا (۲۵۶ تا برای 18CXXX) ثابت در فهرست می باشد.

مثال :

B0 var byte

B1 var byte

B0=\$f

Main:

lookdown B0, ("01234567890ABCDEF"), B1

PORTB=B1

loop: goto loop

End

۲۹-۳-۱. LOOKUP : بدست آوردن مقداری از جدول ثابتها

ترکیب :

LOOKUP Index, (Constant {, Constant}), Var

شرح :

LOOKUP برای خواندن مقداری از جدول ثابتها متناظر با مقدار Index استفاده می شود. اگر Index برابر صفر باشد ، در Var مقدار ثابت اولی قرار می گیرد و اگر Index برابر یک باشد Var

مقدار ثابت دومی را در خود جای می دهد و غیره . واگر مقدار Index از تعداد ثابتها بیشتر باشد ، Var بدون تغییر می ماند. تعداد ۲۵۵ (۲۵۶ تا برای 18CXXX) ثابت در فهرست می تواند قرار بگیرد.

مثال :

برنامه زیر استفاده از lookup برای رقمهای روی نمایشگر سون سگمنت (seven segment) را نشان می دهد.

```
Digit var byte      متغیر digit برای رقمها
Mask var byte      متغیر Mask برای تبدیل شده ها
Main:
  for i=0 to 9
    Digit=i
    Lookup Digit, [$3F, $06, $5B, $4F, $66, $6D, $7D, $07,
      $7F, $6F], Mask
    PORTB=Mask      فرستادن مقدار تبدیل شده رقم به پورت B
    pause 500      تاخیر برای دیدن مقدار خروجی
  next i          افزایش i
goto Main        تکرار دوباره برنامه
end
```

۳-۳۰-۱. LOW : قرار دادن صفر منطقی در پین خروجی

ترکیب :

LOW Pin

شرح :

تغییر پین مورد نظر به صفر . پینها بصورت خودکار در گرایش خروجی قرار می گیرند. می توان مقدار را به صورت مستقیم در رجیستر قرار داد. (البته بایستی پورت مورد نظر در گرایش خروجی قرار گیرد.)

LOW 7

و با

```
TRISB.7 = 0
PORTB.7 = 0
```

۳-۳۱-۱. NAP : خاموش کردن برای یک دوره زمانی کوتاه

ترکیب :

NAP period

شرح :

این دستور میکروکنترلر را برای یک دوره زمانی در حالت توان کم قرار می دهد. در این مدت انرژی مصرفی میکروکنترلر به کمترین حالت خود می رسد. این دوره زمانی با استفاده از تایمر WatchDog تامین می شود و وابسته به تراشه و دما می باشد.

| Period | Delay [ms] |
|--------|------------|
| 0 | 18 |
| 1 | 36 |
| 2 | 72 |
| 3 | 144 |
| 4 | 288 |
| 5 | 576 |
| 6 | 1152 |
| 7 | 2304 |

جدول ۱-۱۰

مثال:

```
Main:
    nap 7      ' take a nap for 2.304 seconds
Loop: goto Loop
End
```

۳۲-۳-۱ . OUTPUT : گماشتن پین I/O در گرایش خروجی

ترکیب :

OUTPUT pin

شرح :

پین مورد نظر را بصورت خروجی پیکربندی می کند

OUTPUT PORTB.7

شبیه به :

TRISB.7 = 0

۳۳-۳-۱ . OWIN : دریافت اطلاعات از طریق ارتباط یک سیم

ترکیب :

OWIN Pin, Mode, [Var1, Var2...]

شرح :

پارامتر " Pin " نشان دهنده پینی است که ارتباط یک سیمه را با وسیله دیگر می خواهد برقرار کند. پارامتر " Mode " مقدار تعریف شده پارامترهای ارتباط را در خود قرار می دهد.

| بیت "Mode" | چگونگی عملکرد |
|------------|--|
| 0 | فرستادن سیگنال ریست قبل از ارسال اطلاعات = 1 |
| 1 | فرستادن سیگنال ریست بعد از ارسال اطلاعات = 1 |
| 2 | 0 = 8-bit data 1 = 1-bit data |

جدول ۱-۱۱

پارامترهای "Var1" و "Var2" متغیرهای شامل اطلاعات خوانده شده می باشند.
مثال:

Temperature var byte

Main:

```
OWIN PORTC.0, 0, [Temperature]    خواندن دما
PORTB=Temperature                 نشان دادن دما روی پورت B
goto Main
```

End

۳-۳۴-۱. OWOUT : ارسال اطلاعات از طریق یک سیم (One_Wire Communication)

ترکیب :

OWOUT Pin, Mode, [Var1, Var2...]

شرح :

پارامتر " Pin " نشان دهنده پینی است که ارتباط یک سیمه را با وسیله دیگر می خواهد برقرار کند. پارامتر " Mode " مقدار تعریف شده پارامترهای ارتباط را در خود قرار می دهد.

| بیت "Mode" | چگونگی عملکرد |
|------------|--|
| 0 | فرستادن سیگنال ریست قبل از ارسال اطلاعات = 1 |
| 1 | فرستادن سیگنال ریست بعد از ارسال اطلاعات = 1 |
| 2 | 0 = 8-bit data 1 = 1-bit data |

جدول ۱-۱۲

مثال :

Main :

```
OWOUT PORTC.0, 1, [$CC, $BE]      فرستادن سیگنال ریست بعد از دو مقدار
goto Main
End
```

۳-۳۵-۱. PAUSE : تاخیر برحسب میلی ثانیه

ترکیب :

PAUSE Period (in milliseconds)

شرح :

این دستور به اندازه Period میلی ثانیه تاخیر ایجاد می کند. Period یک مقدار ۱۶ بیتی است و حداکثر تاخیر ۶۵۵۳۵ میلی ثانیه می باشد. برخلاف دستورهای NAP و SLEEP ، دستور PAUSE میکروکنترلر را در حالت توان کم قرار نمی دهد. بنابراین این دستور توان بیشتری را مصرف می کند ولی زمان دقیقتری را به ما می دهد. (دقت آن به کلاک اسیلاتور بستگی دارد) این دستور با اسیلاتور از پیش تعریف شده 4MHZ کار می کند و اسیلاتورهای دیگر با دستور DEFINE بایستی تعریف بشوند.

مثال :

```
TRISB = 0
Main:
    PORTB = 255
    pause 1000 ' تاخیر ۱ ثانیه ای
    PORTB = 0
    pause 2000 ' تاخیر ۲ ثانیه ای
goto Main
End
```

۳-۳۶-۱. PAUSEUS : تاخیر برحسب میکروثانیه

ترکیب :

PAUSEUS *Period (in miliseconds)*

شرح :

PAUSEUS برنامه را برای Period میکروثانیه متوقف می کند. Period ۱۶ بیتی (WORD) می باشد و بیشترین تاخیر 65535 μ s است. این دستور با اسیلاتور از پیش تعریف شده 4MHZ کار می کند و اسیلاتورهای دیگر با دستور DEFINE بایستی تعریف بشوند. این دستور با صرف انرژی بیشتر زمان دقیقتری را می دهد. کمترین تاخیر با این دستور به فرکانس اسیلاتور بستگی دارد.

| OSC | Minimal delay |
|----------|---------------|
| 3 (3.58) | 20 us |
| 4 | 24 us |
| 8 | 12 us |
| 10 | 8 us |
| 12 | 7 us |
| 16 | 5 us |
| 20 | 3 us |

جدول ۱۳-۱

مثال :

TRISB = 0

Main:

PORTB = 255

pauseus 100

PORTB = 0

pauseus 3450 goto Main

End

۳۷-۳-۱ . POT : برگرداندن مقدار مقاومت متصل به پین

ترکیب :

POT Pin, Scale, Var

شرح :

دستور POT مقدار پتانسیومتری را که روی پین موردنظر قرار گرفته محاسبه می کند. مقدار مقاومت می تواند با RC های مختلف پایدار محاسبه شود. مقاومت با زمان تخلیه خازن در مقاومت محاسبه می شود.

Scale برای تنظیم تغییرات ضریب ثابت شبکه RC به کار می رود برای ضرایب ثابت بزرگ RC ، مقدار Scale را 1 انتخاب می کنیم و برای ضرایب ثابت کوچک Scale را روی حداکثر مقدار خود ۲۵۵ قرار می دهیم. اگر مقدار Scale درست انتخاب شده باشد مقدار متغیر Var در حالتی که مقدار مقاومت حداقل است صفر می شود و هنگامی که حداکثر مقاومت را داریم ۲۵۵ می شود. (متاسفانه Scale بایستی بصورت تجربی بدست آید)

انواع مبدل‌های مقاومتی وجود دارند که می توانند توسط دستور POT آنها را خواند. این مبدل‌ها ولتاژ را اندازه نمی گیرند بلکه مقاومت را اندازه می گیرند.

مثال :

برای تنظیم Scale می توان از برنامه زیر استفاده کرد:

B0 var byte

skala var byte

Main :

FOR skala=1 TO 255

pot PORTA.0, scale, B0 خواندن مقدار پتانسیومتر روی پین RA0

IF B0>253 Then Over

NEXT skala

Over : PORTB=scale

نشان دادن مقدار مقیاس بروی پورت B

goto Main

End

۳۸-۳-۱ . PULSIN : محاسبه عرض پالس روی پایه ورودی

ترکیب :

PULSIN Pin, Level, Var

شرح :

این دستور عرض پالس را با دقت $10\mu s$ (وقتی با اسیلاتور $4MHz$ کار می کند) روی پین مورد نظر محاسبه می کند و اگر متغیر Level ، صفر باشد قسمت پایین عرض پالس اندازه گیری می شود و اگر یک باشد قسمت بالای پالس اندازه گیری می شود مقدار اندازه گیری شده در متغیر Var ذخیره می شود. زمان اندازه گیری می تواند از ۱۰ تا ۶۵۵۳۵ میکرو ثانیه طول بکشد. (متغیر ۱۶بیتی است) اگر متغیر ۸بیتی باشد ۸ بیت پایین از متغیر ۱۶بیتی WORD استفاده می شود قدرت تفکیک به فرکانس اسیلاتور بستگی دارد و برای اسیلاتور $4MHz$ حد تفکیک $10\mu s$ می باشد. وقتی از اسیلاتور $20 MHz$ استفاده می شود ، حد تفکیک $2\mu s$ می شود.

مثال :

W0 var word

Main :

pulsin PORTB.0, 1, W0 'RB0 محاسبه ایمپالسهای یک روی پین

goto Main

End

۳۹-۳-۱. **PULSOUT** : تولید کردن پالس در پین خروجی

ترکیب :

PULSOUT Pin, Period

شرح :

این دستور پالس با حد تفکیک ۱۰ میکروثانیه را تولید می کند. پالس با تغییر وضعیت یک پین به حالت دیگر تولید می شود. پین بصورت خودکار در گرایش ورودی قرار می گیرد.

مثال :

Main :

pulsout PORTB.7, 100 'RB7 تولید ایمپالسها روی پین

goto Main

End

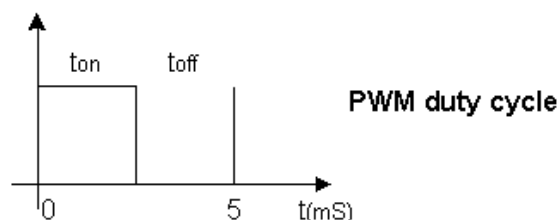
۴۰-۳-۱. **PWM** : تولید سیگنال PWM روی پین

ترکیب :

PWM Pin, Ratio, Cycle

شرح :

این دستور پالسهای PWM (Pulse_width Modulation) برای پین مشخص شده می فرستد. این سیکل برای تعداد بارهای که تعریف شده با پارامتر Cycle خودش را تکرار می کند. مدت دوره



به اسیلاتور بستگی دارد. در حالتی که اسیلاتور 4MHz است مدت دوره تناوب 5ms است. وقتی اسیلاتور 8MHz باشد این دوره تناوب به 1ms می رسد. با دستور PWM به سادگی با یک مدار RC می توان ولتاژ DC تولید کرد و در واقع یک مبدل D/A ساده ساخته می شود.
مثال :

Main :

```
pwm PORTB.7, 127, 100 ' RB7 روی پین ۵۰% PWM فرستادن سیگنال
goto Main
End
```

۴۱-۳-۱. RANDOM : تولید عدد تصادفی

ترکیب :

RANDOM Variable

شرح :

این دستور عدد تصادفی را در یک متغیر ۱۶ بیتی ذخیره می کند. (تنها صفر را تولید نمی کند)

مثال :

W0 var word

Main :

```
random W0 ' قرار دادن مقدار راندم در متغیر W0
lcdout #W0 ' نمایش مقدار راندم در LCD
goto Main
End
```

۴۲-۳-۱. RCTIME : محاسبه پالس روی پین (مشابه PULSIN)

ترکیب :

RCTIME Pin, State, Variable

شرح :

این دستور مدت زمان را که پین در یک حالت معینی قرار می گیرد ، اندازه می گیرد. اگر پین بدون تغییر بماند متغیر Variable صفر می شود. RCTIME برای خواندن پتانسیمر با عناصر مقاومتی دیگر استفاده می شود. حد تفکیک به فرکانس اسیلاتور بستگی دارد. و برای اسیلاتور 4MHz حد تفکیک 10µs می باشد. وقتی از اسیلاتور 20 MHz استفاده می شود ، حد تفکیک 2µs می شود

مثال :

W0 var word

Main :

```
low PORTA.0 ' تخلیه خازن
pause 10 ' تخلیه به اندازه ۱۰ میلی ثانیه
rctime PORTA.0, 0, W0 ' محاسبه مدت شارژ شدن
lcdout #W0 ' نمایش مقدار W0 روی LCD
goto Main
End
```

۳-۳-۱. READ : خواندن یک بایت از اطلاعات EEPROM

ترکیب :

READ Address, Variable

شرح :

این دستور اطلاعات را از حافظه EEPROM داخلی از آدرس مشخص شده می خواند و نتایج را در Variable ذخیره می کند. این دستور تنها با میکروکنترلرهای که دارای EEPROM داخلی است اجرا می شوند. از دستور I2C_READ برای EEPROM های خارجی استفاده می شود.

B0 var byte

W var word

Main :

READ 5, B0 ' خواندن مقدار آدرس ۵ و ریختن در متغیر B0

READ 6, W.BYTE0 ' خواندن یک مقدار ۱۶ بیتی از آدرس ۶ و ۷

READ 7, W.BYTE1

Loop: goto Loop

End

۳-۳-۱. READCODE : خواندن ۲ بایت (WORD) از کد برنامه

ترکیب :

READ Address, Variable

شرح : این دستور کد برنامه را از آدرس داده شده می خواند و نتیجه را در متغیر ۱۶ بیتی می گذارد. میکروکنترلرهای PIC16F87X خواندن و نوشتن کد برنامه را در حال اجرا اجازه می دهند.

مثال :

Wo var word

Main :

readcode 100, W0 ' خواندن اطلاعات از حافظه فلش در آدرس ۱۰۰ و ریختن در متغیر W0

Loop : goto Loop

End

۳-۳-۱. REVERSE : تغییر گرایش پین

ترکیب :

REVERSE Pin

شرح :

این دستور گرایش پین مورد نظر را وارونه می کند. اگر پینی در گرایش ورودی باشد با این دستور در گرایش خروجی قرار می گیرد.

ترکیب :

SERIN Pin, Mode, {Timeout, Label}, {[Qual...]}, {Item...}

شرح :

SERIN مقادیری را با قالب بندی آسنکرون (۸ بیت بدون بیت توازن و یک بیت توقف) از پین مشخص شده دریافت و در متغیر ذخیره می کند. بجای مقدار عددی تنظیمی ۰ تا ۱۵ (Mode) می توان یک اسم از کتابخانه "modedefs.inc" را قرار داد. (جدول ۱۴-۲)

دستور SERIN می تواند دارای برچسب (پارامتر Label) نیز باشد که در صورت دریافت نکردن اطلاعاتی در زمان معین برنامه به آن برچسب می پرد. (حالت از پیش تعریف شده پارامتر Timeout ، 1ms است)

اگر مقداری یا کرکتری داخل براکت " [] " قرار بگیرد میکروکنترلر قبل از دریافت اطلاعات اصلی بایستی مقدار قیدگذاری شده را دریافت کند. قیدگذار می تواند یک مقدار ثابت یا یک متغیر و یا یک رشته باشد. اگر علامت "#" قبل از متغیر قرار گیرد مقدار دسیمال دریافتی به ASCII تبدیل می شود و نتیجه را در متغیر ذخیره می کند.

| Mode | Mode number | Baud rate | State |
|-------|-------------|-----------|----------|
| T2400 | 0 | 2400 | True |
| T1200 | 1 | 1200 | |
| T9600 | 2 | 9600 | |
| T300 | 3 | 300 | |
| N2400 | 4 | 2400 | Inverted |
| N1200 | 5 | 1200 | |
| N9600 | 6 | 9600 | |
| N300 | 7 | 300 | |

جدول ۱۴-۱

این دستور با اسیلاتور 4MHZ درست کار می کند و برای اینکه با اسیلاتورهای دیگر درست کار کند بایستی فرکانس اسیلاتور با دستور DEFINE تعریف شود.

مثال :

```
Include " modedefs.inc "
```

```
B0 var byte
```

```
Main :
```

انتظار برای رسیدن کرکتر "A" روی پین RB0 و سپس دریافت کرکتهای بعدی

```
serin PORTB.0, N2400, ["A"], B0
```

```
variable B0
  lcdout B0 ' LCD روی B0 محتوای
Loop : goto Loop
End
```

۳-۴۷- ۱. SEROUT : خروجی سریال آسنکرون

ترکیب :

SEROUT *Pin, Mode, [Item{, Item...}]*

شرح :

اطلاعات را با قالب بندی آسنکرون استاندارد 8N1 از طریق پین مشخص شده می فرستد . مدهای انتقال ("Mode") به این شرح می باشند.
 بجای مقدار عددی تنظیمی ۰ تا ۱۵ (Mode) می توان یک اسم از کتابخانه "modedefs.inc" را قرار داد.

| Mode | Mode number | Baud Rate | State |
|--------|-------------|-----------|-----------------|
| T2400 | 0 | 2400 | Driven True |
| T1200 | 1 | 1200 | |
| T9600 | 2 | 9600 | |
| T300 | 3 | 300 | |
| N2400 | 4 | 2400 | Driven Inverted |
| N1200 | 5 | 1200 | |
| N9600 | 6 | 9600 | |
| N300 | 7 | 300 | |
| OT2400 | 8 | 2400 | Open True |
| OT1200 | 9 | 1200 | |
| OT9600 | 10 | 9600 | |
| OT300 | 11 | 300 | |
| ON2400 | 12 | 2400 | Open Inverted |
| ON1200 | 13 | 1200 | |
| ON9600 | 14 | 9600 | |
| ON300 | 15 | 300 | |

جدول ۱-۱۵

اگر علامت "#" قبل از متغیر SEROUT قرار گیرد مقدار دسیمال به کد ASCII تبدیل می شود مثلا اگر B برابر ۳۴ باشد آنگاه #B در این دستور "۳" و "۴" را می فرستد. این دستور با اسیلاتور 4MHz درست کار می کند و برای اینکه با اسیلاتورهای دیگر درست کار کند بایستی فرکانس اسیلاتور با دستور DEFINE تعریف شود. تاخیر بعد از فرستادن هر مقدار با DEFINE تعریف می شود که زمان ۱ تا ۶۵۵۳۵ میکرو ثانیه را در بر می گیرد.

DEFINE CHAR_PACING 1000 ' ۱ میلی ثانیه تاخیر بین دو کرکتر

مثال :

DO Var\Byte

Main:

B0 = 25

serout PORTA.3, N2400, [#B0, 13] 'RA0 و عدد ثابت ۱۳ با B0 فرستادن مقدار کد اسکی

Loop : goto Loop

End

۴۸-۳-۱ . SHIF TIN : ورودی سریال سنکرون

ترکیب :

SHIF TIN DataPin, ClockPin, Mode, [Var\Bits]...

شرح :

این دستور بیت‌های دریافتی از پین مشخصی را بصورت سنکرون با سیگنال فرکانسی "ClockPin" دریافت می کند و آنها را در متغیری ذخیره می کند. "Var\Bits" مقدار بیت‌ها شیف ت داده شده را بصورت دلخواه تعیین می کند مقدار پیش فرض آن ۸ می باشد. با توجه به شیف ت دادن مستقیم (از MSB به LSB) مدهای انتقال گوناگون می تواند تعریف شود. برای استفاده از این مدها بایستی کتابخانه "modedefs.BAS" به برنامه الحاق کنید.

| "Mode" | Mode number | عملکرد |
|----------|-------------|--|
| MSBP RE | 0 | ابتدا بیت بالاتر شیف ت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی صفر منطقی غیر فعال می شود. |
| LSBP RE | 1 | ابتدا بیت پایینتر شیف ت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی صفر منطقی غیر فعال می شود. |
| MSBP OST | 2 | ابتدا بیت بالاتر شیف ت داده می شود اطلاعات بعد از کلاک فرستاده شده خوانده می شود. کلاک روی صفر منطقی غیر فعال می شود. |
| LSBP OST | 3 | ابتدا بیت پایینتر شیف ت داده می شود اطلاعات بعد از کلاک فرستاده شده خوانده می شود. کلاک روی صفر منطقی غیر فعال می شود |
| | 4 | ابتدا بیت بالاتر شیف ت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی یک منطقی غیر فعال می شود. |
| | 5 | ابتدا بیت پایینتر شیف ت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی یک منطقی غیر فعال می شود. |
| | 6 | ابتدا بیت بالاتر شیف ت داده می شود اطلاعات بعد از کلاک فرستاده شده خوانده می شود. کلاک روی یک منطقی غیر فعال می شود. |
| | 7 | ابتدا بیت پایینتر شیف ت داده می شود اطلاعات بعد از کلاک فرستاده شده خوانده می شود. کلاک روی یک منطقی غیر فعال می شود. |

جدول 1-16

فرکانس شیف ت حدود 50 KHZ است که وابسته به اسیلاتور می باشد. با استفاده از DEFINE تاخیر اضافی را برای آهسته کردن کلاک تعریف می کنیم.

DEFINE SHIF T_PAUSEUS 100 ' Slowing down the clock for additional 100ms

مثال :

shiftin Data, Clock, MSBPRE, [RxData]

SHIFTOUT ۱-۳-۴۹ : خروجی سریال سنکرون

ترکیب :

SHIFTOUT DataPin, ClockPin, Mode, [Var{Bits}...]

شرح :

این دستور بیتها را از طریق پین مشخصی بصورت سنکرون با سیگنال فرکانسی "ClockPin" ارسال می کند. "Var\Bits" مقدار بیتها شیفت داده شده را تعیین می کند. با توجه به شیفت دادن مستقیم (از MSB به LSB) مدهای انتقال گوناگون می تواند تعریف شود. برای استفاده از این مدها بایستی کتابخانه "modedefs.BAS" را به برنامه الحاق کنید

| "Mode" | Mode number | عملکرد |
|----------|-------------|---|
| LSBFIRST | 0 | ابتدا بیت بالاتر شیفت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی صفر منطقی غیر فعال می شود. |
| MSBFIRST | 1 | ابتدا بیت پایینتر شیفت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی صفر منطقی غیر فعال می شود. |
| | 4 | ابتدا بیت بالاتر شیفت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی یک منطقی غیر فعال می شود. |
| | 5 | ابتدا بیت پایینتر شیفت داده می شود اطلاعات پیش از کلاک فرستاده شده خوانده می شود. کلاک روی یک منطقی غیر فعال می شود. |

جدول ۱-۱۷

مثال :

B0 var byte

B1 var byte

W0 var byte

Main :

shiftout PORTA.0, PORTA.1, MSBFIRST, [B0, B1]

فرستادن محتوای B0 و B1 بصورت شیفت دادن به خروجی بطوری که اولین بیت شیفت داده شده بیت MSB می باشد

shiftout PORTA.0, PORTA.1, MSBFIRST, [W0\4]

فرستادن چهار بیت از متغیر W0 بطوریکه اولین بیت انتقال یافته بیت MSB می باشد.

Loop : goto Loop

End

SLEEP ۱-۳-۵۰ : خاموش کردن پردازنده برای یک پرپود زمانی معین

ترکیب :

SLEEP Period

شرح :

این دستور میکروکنترلر را به حالت کم مصرف برای Period ثانیه قرار می دهند. Period یک مقدار ۱۶بیتی می باشد که حداکثر ۶۵۵۳۵ ثانیه (حدود ۱۸ ساعت) تاخیر را ایجاد می کند. دستور SLEEP از تایمر WDT استفاده می کند که بعلت استفاده کردن از RC داخلی دقت کمتری نسبت به کلاک اسیلاتور دارد.

۵۱-۳-۱. SOUND : تولید کردن صدا یا نویز سفید روی یک پین مشخص

ترکیب :

SOUND Pin, (Note, Duration{, Note, Duration})

شرح :

این دستور صدا و نویز روی پین مشخصی تولید می کند. برای $Note = 0$ صدای وجود ندارد. اگر Note مقداری بین ۱ تا ۱۲۷ باشد صدا تولید می کند و اگر بین ۱۲۸ تا ۲۵۵ باشد نویز تولید می کند. $Note = 1$ دارای فرکانس 78.74Hz و $Note = 127$ دارای فرکانس 10KHz می باشد. Duration مقداری بین ۰ تا ۲۵۵ میباشد که مدت زمان تولید صدا را تعیین می کند. هر یک واحد افزایش این متغیر تقریباً معادل 12ms می باشد. سیگنال خروجی، یک موج مربعی با سطح ولتاژ TTL می باشد. یک بلندگو کوچک به همراه خازن مستقیم می تواند توسط پایه تولید سیگنال راه اندازی شود. بلندگوهای کریستالی (Piezo) به طور مستقیم راه اندازی می شوند.

مثال :

فرستادن ۲ صدای متوالی به خروجی ' (100, 10, 50, 10) sound PORTB.7,

۵۲-۳-۱. STOP : توقف برنامه در حال اجرا

شرح :

این دستور برنامه اجرای را بوسیله قرار دادن در حلقه بی نهایت متوقف می کند. این دستور میکروکنترلر را به حالت توان کم نمی برد.

۵۳-۳-۱. SWAP : مبادله مقادیر دو متغیر

ترکیب :

SWAP Variable1, Variable1

شرح :

این دستور با انواع متغیرها (bit ، byte ، word) استفاده می شود. SWAP با رشته ها نیز استفاده می شود ولی بایستی شاخص آنها یکسان باشد.

مثال :

```
B0 var byte
B1 var byte
temp var byte
Main :
    temp = B0
    B0 = B1
    B1 = temp    ' راه قدیمی انجام آن '
    swap B0, B1 ' ..و راه آسانتر آن '
Loop : goto Loop
End
```

۳-۵۴-۱. TOGGLE : وارون کردن وضعیت پین

ترکیب :

TOGGLE *Pin*

شرح :

این دستور وضعیت پین را وارون می کند . در این دستور پین بصورت خودکار در گرایش خروجی قرار می گیرد.

مثال :

```
Main :
    low PORTB.0
    toggle PORTB.0
Loop : goto Loop
End
```

۳-۵۵-۱. WRITE : نوشتن اطلاعات در EEPROM داخلی

ترکیب :

WRITE *Address, Value*

شرح :

این دستور Value را در آدرس تعیین شده در EEPROM داخلی می ریزد . اگر متغیر دو بایتی را خواسته باشیم ذخیره کنیم بایستی هر بایت را جدا گانه ذخیره کرد.

```
WRITE Address, Variable.BYTE0
WRITE Address, Variable.BYTE1
```


اگر در برنامه از وقفه استفاده می شود قبل از این دستور کلیه وقفه ها (با دستور DISABLE) غیر فعال شوند.

مثال :

```
B0 var byte
Main :
    B0 = $EA
    write 5, B0 'EEPROM $EA در موقعیت ۵ حافظه
Loop : goto Loop
End
```

۵۶-۳-۱. WRITECODE : نوشتن دوبایت از اطلاعات بر روی حافظه برنامه

ترکیب :

WRITECODE *Address, Value*

شرح :

این دستور " Value " را در آدرس معینی به حافظه برنامه اضافه می کند. این دستور تنها با میکروکنترلرهای PIC که دارای حافظه FLASH می باشند (مانند PIC16F87X)، بکار می رود. وقفه ها موقع بکارگیری این دستور بایستی غیر فعال باشند.

مثال :

```
W0 var byte
Main :
    W0 = $12FE
    writecode 100, W0 'حافظه FLASH برنامه ۱۰۰
Loop : goto Loop
End
```

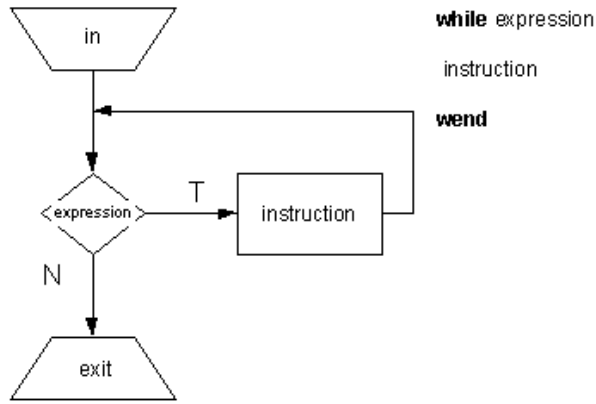
۵۷-۳-۱. WHILE – WEND : اجرای قسمتی از برنامه تا وقتی که شرط برقرار است

ترکیب :

WHILE *Condition*
Instructions...
WEND

شرح :

هدف از این دستور قرار دادن قسمتی از برنامه بین WHILE و WEND تا موقعی که شرط برقرار است و بعد از آن از حلقه خارج می شود.



مثال :

i Var byte

Main :

i = 1

WHILE i < 10 در دهمین تکرار برنامه متوقف می شود و پورت B مقدار ۹ را نشان می دهد

i = i + 1

PORTB = i

Pause 1000

WEND

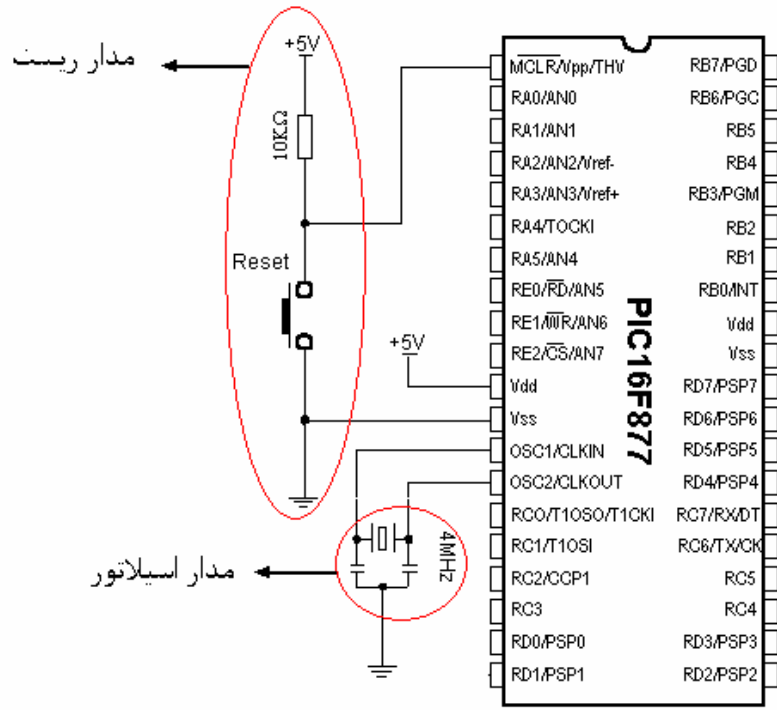
goto Main

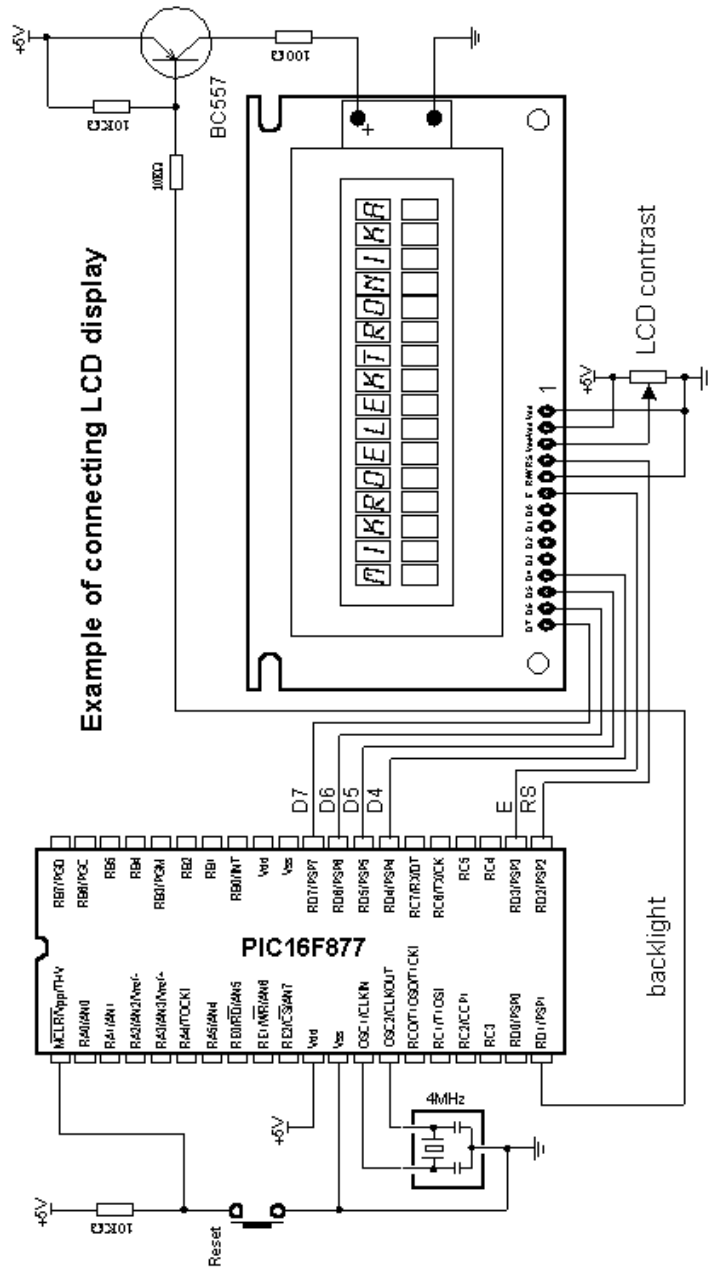
End

میکروکنترلر PIC

مثالها به همراه مدارهای مورد نیاز

مدار راه انداز میکروکنترلرهای PIC (برای نمونه PIC 16F877)





```

DEFINE LCD_DREG    PORTD  \ پورت اتصال به LCD
DEFINE LCD_DBIT    4
DEFINE LCD_RSREG   PORTD
DEFINE LCD_RSBIT    2      \ پین انتخاب رجیستر RS
DEFINE LCD_EREG    PORTD
DEFINE LCD_EBIT    3      \ پین فعال سازی E
DEFINE LCD_BITS    4      \ پس اطلاعات 4 بیتی
DEFINE LCD_LINES   2      \ LCD دارای دو خط می باشد

high    PORTD.1          \ روشن کردن LED روشن کننده زمینه LCD

Main:
\ شروع برنامه
low     PORTD.1          \ روشن کردن LED روشن کننده زمینه LCD
lcdout  $fe,1           \ پاک کردن صفحه نمایش LCD
\
lcdout  "pic programmer"
pause   2000            \ تاخیر 2 ثانیه ای

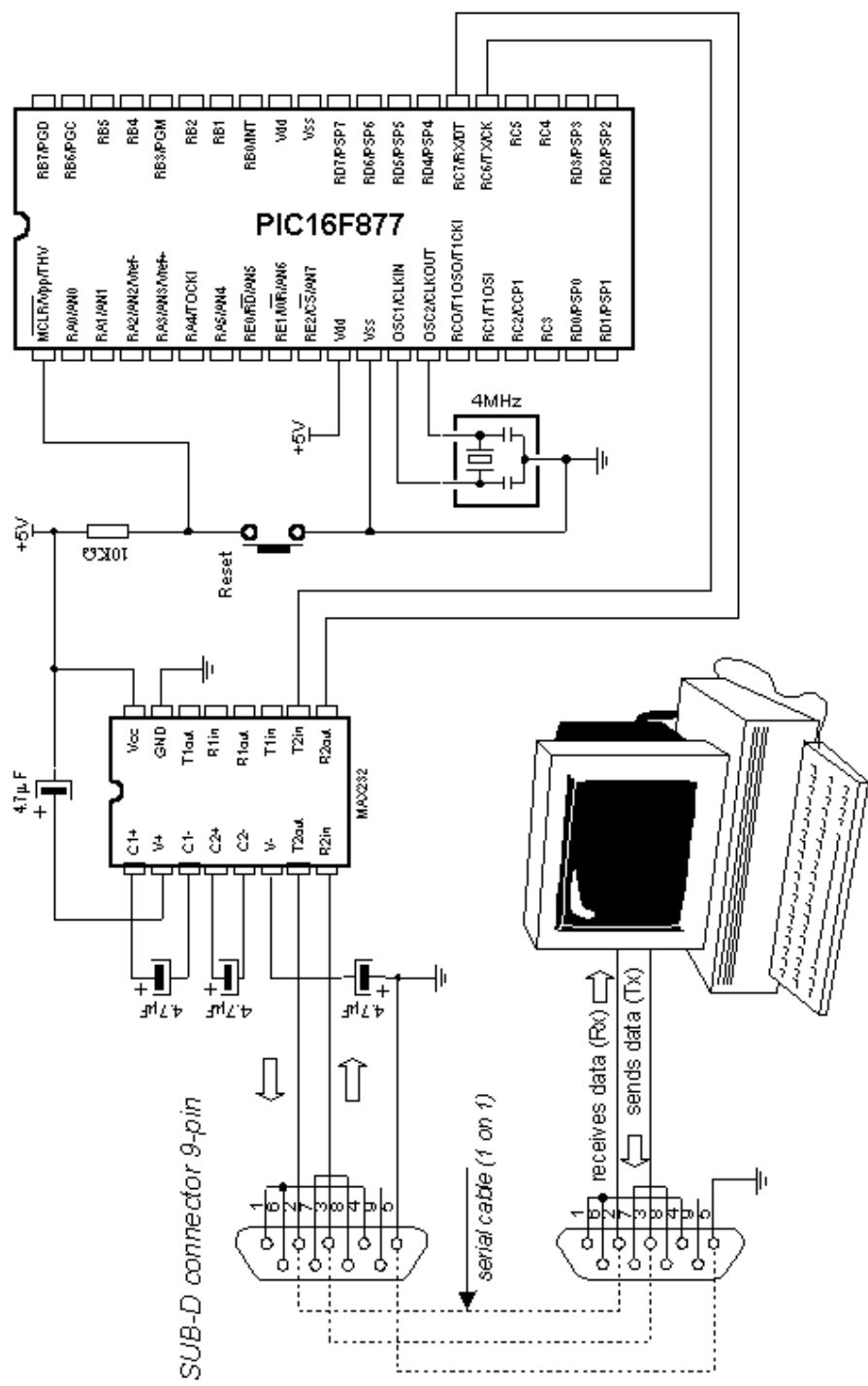
lcdout  $fe,1           \ پاک کردن صفحه نمایش LCD
\ نوشتن "LCD example" در خط اول
lcdout  "LCD example"
lcdout  $fe,$c0         \ رفتن به خط دوم
\ نوشتن " Second line " در خط دوم
lcdout  " Second line "

pause   2000            \ تاخیر 2 ثانیه ای
high    PORTD.1          \ خاموش کردن LED روشن کننده زمینه LCD
pause   2000

goto   Main             \ تکرار حلقه
end
\ پایان برنامه

```

مثال ۲: ارتباط سریال با کامپیوتر



```

BO var byte          ' Var. for storing received data
TRISC = %101111111  ' PC6 output TX pin, rest is input
SPBRG = 25           ' Set Baud rate to 2400
RCSTA = %10010000   ' Enable serial port and reception
TXSTA = %00100000   ' Enable asynchronous data sending

Main:
Gosub charin        ' Receiving data via serial line
If BO = 0 Then Main ' Data is not received
Gosub charout       ' If received send it back
Goto Main           ' Repeat the loop

charin:
BO = 0              ' B=0 if data is not received
If PIR1.5 = 1 Then ' If PIR1.5 = 1 data
                    ' is in RCREG
BO = RCREG          ' Load the data from the receiving
                    ' register RCREG to BO

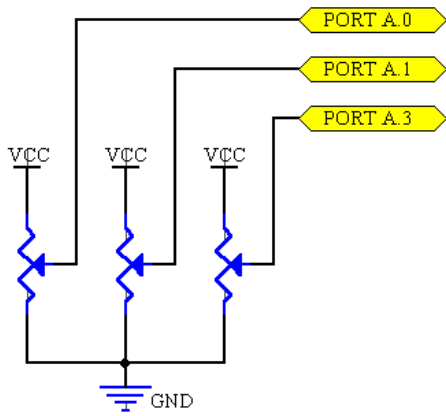
Endif
Return

charout:
If PIR1.4 = 0 Then charout      ' Wait for sending
                                ' register to be ready
TXREG = BO                      ' Send the data to
                                ' sending register

Return

End                             ' End of the program

```

مثال ۳: استفاده از پورت A بعنوان ورودی آنالوگ و نشان دادن مقادیر تبدیل شده از آنالوگ به دیجیتال

```

===== Define LCD pins =====
Define LCD_DREG    PORTD
Define LCD_DBIT    4
Define LCD_RSREG   PORTE
Define LCD_RSBIT   0
Define LCD_EREGL   PORTE
Define LCD_EBIT    1

===== Allocate variables =====
x  var  byte
y  var  byte
z  var  byte

ADCON1 = 4      ' Set PortA 0, 1, 3 to A/D inputs

Low PORTE.2    ' LCD R/W line low (W)
Pause 100      ' Wait for LCD to start

Goto  mainloop ' Skip subroutines

=====Subroutine to read a/d convertor =====
getad:
Pauseus 50     ' Wait for channel to setup

ADCON0.2 = 1   ' Start conversion

```

```
Pause 50      ' Wait for conversion
Return
```

```
'===== Subroutine to get pot x value =====
```

```
getx:
```

```
ADCON0 = $41      ' Set A/D to Fosc/8, Channel 0, On
Gosub getad
x = ADRESH
Return
```

```
'===== Subroutine to get pot y value =====
```

```
gety:
```

```
ADCON0 = $49      ' Set A/D to Fosc/8, Channel 1, On
Gosub getad
y = ADRESH
Return
```

```
'===== Subroutine to get pot z value =====
```

```
getz:
```

```
ADCON0 = $59      ' Set A/D to Fosc/8, Channel 3, On
Gosub getad
z = ADRESH
Return
```

```
'=====Main program =====
```

```
mainloop:
```

```
Gosub getx      ' Get x value
Gosub gety      ' Get y value
Gosub getz      ' Get z value
```

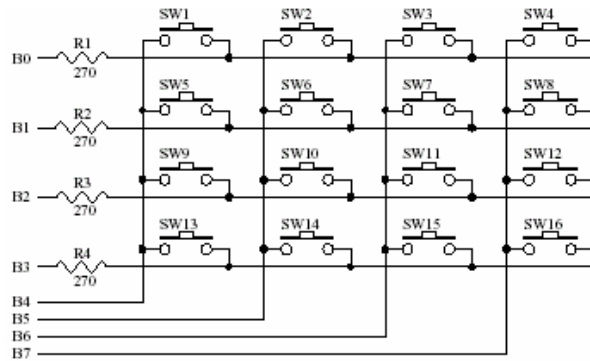
```
Lcdout $fe, 1, "x=", #x, " y=", #y, " z=", #z ' Send to LCD
```

```
Pause 100      ' Do it about 10 times a second
```

```
Goto mainloop  ' Do it forever
```

```
End
```

مثال ۴: ساخت کیبورد سریال



```
INCLUDE "modedefs.bas"
```

```
col          VAR   BYTE    ' Keypad column
row          VAR   BYTE    ' Keypad row
key          VAR   BYTE    ' Key value
baud        VAR   PortA.0  ' Baud select pin
serpin      VAR   PortA.1  ' Serial output pin
CMCON       =     7       ' PortA = digital I/O
VRCON       =     0       ' Voltage reference disabled
TRISA       =   %00000001 ' PortA.0 = baud select pin
OPTION_REG.7 = 0         ' Enable PORTB pull-ups

loop:
    GOSUB getkey          'Get key from keypad

send:
    IF baud = 1 THEN fast 'If baud = 1 then N9600,else N2400
    SEROUT serpin,N2400,[key] 'Send key value out PortA.1
    GOTO loop
```

```

fast:
    SEROUT serpin,N9600,[key]
    GOTO loop                'Do it forever

getkey:
    PAUSE 50                 'Debounce key-input

getkeyu: ' Wait for all keys up
    PORTB = 0                ' All output-pins low
    TRISB = $f0              ' Bottom 4-pins out, top 4-pins in
    IF ((PORTB >> 4) != $f) THEN getkeyu 'If keys down, loop
    PAUSE 50                 ' Debounce key-input

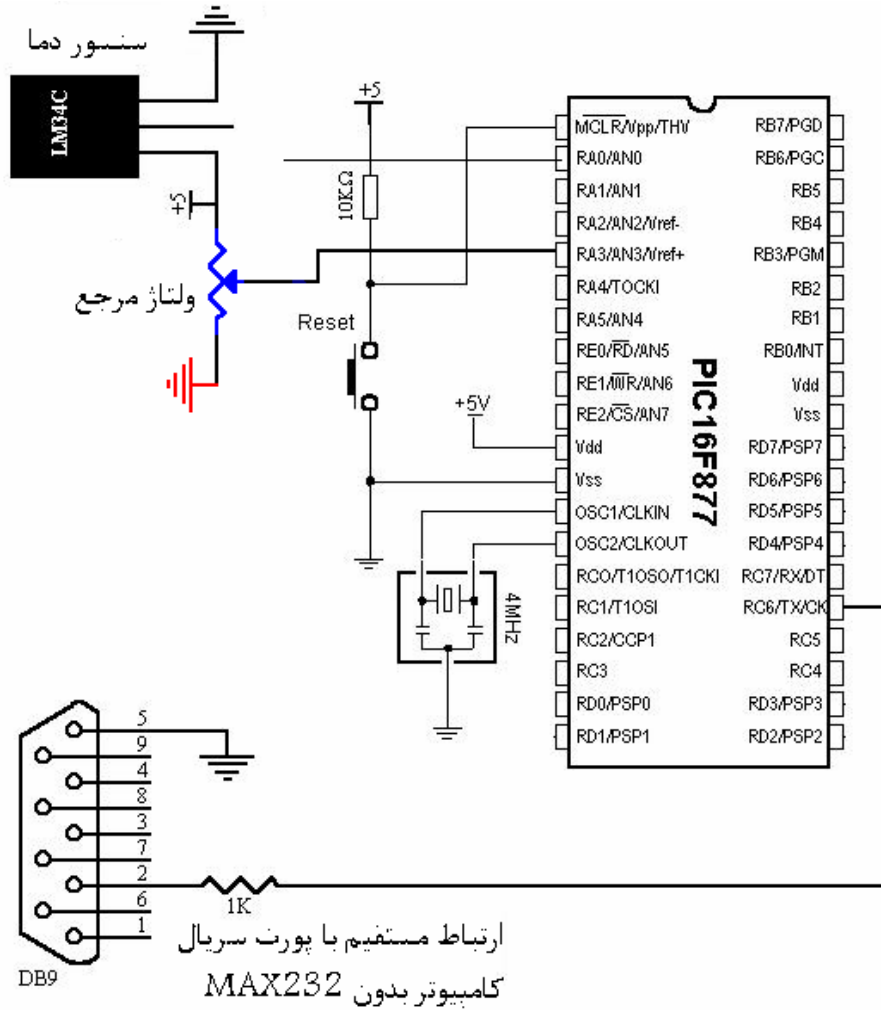
getkeyp: ' Wait for keypress
    FOR row = 0 TO 3        ' 4 rows in keypad
        PORTB = 0          ' All output-pins low
        TRISB = (DCD row) ^ $ff ' Set one row pin to output
        col = PORTB >> 4    ' Read columns
        IF col != $f THEN gotkey ' If any keydown, exit
    NEXT row
    GOTO getkeyp            ' No keys down, go look again

gotkey: ' Change row and column to key number 1 - 16
    key = (row * 4) + (NCD (col ^ $f))
    'NOTE: for 12-key keypad, change to key = (row * 3)
    RETURN                  ' Subroutine over

END

```

مثال ۵: اندازه گیری دما و نمایش آن در پنجره ارتباط سریال MicriCode Studio



```

DEFINE loader_used 1      ' Boot loader is being used
DEFINE debug_mode 0      ' Debug sending True serial data
DEFINE debug_reg portc   ' Debug Port = PortC
DEFINE debug_bit 6       ' Debug.bit = PortC.6
DEFINE debug_baud 9600   ' Default baud rate = 9600
DEFINE osc 4             ' We're using a 4 MHz oscillator
    
```

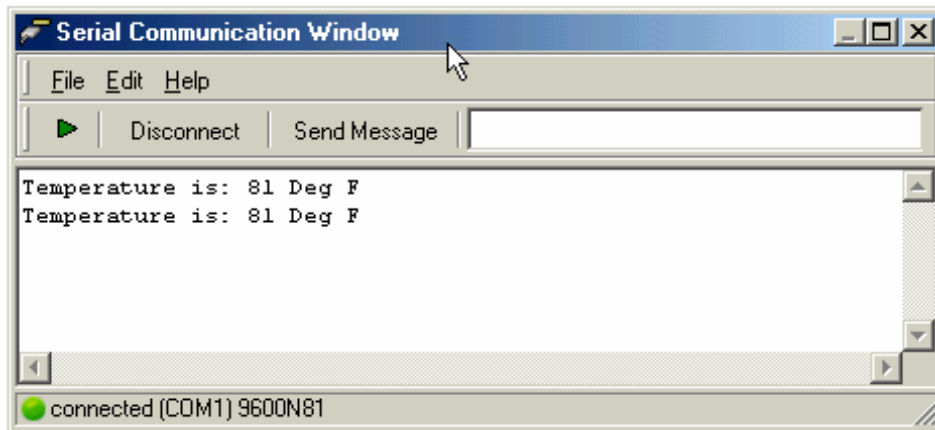
```

DEFINE ADC_BITS 8      ' Set A/D for 8-bit operation
DEFINE ADC_CLOCK 1     ' Set A/D clock Fosc/8
DEFINE ADC_SAMPLEUS 50 ' Set A/D sampling time @ 50 uS
samples VAR WORD      ' Multiple A/D sample accumulator
sample VAR BYTE      ' Holds number of samples to take
temp VAR BYTE       ' Temperature storage
samples = 0             ' Clear samples accumulator on power-up
    TRISA = %11111111   ' Set PORTA to all input
    ADCON1 = %00000011 ' Set PORTA.0,1,2,5 = A/D, PortA.3 = +Vref
    PAUSE 500         ' Wait .5 second

loop:
    FOR sample = 1 TO 20 ' Take 20 samples
        ADCIN 0, temp ' Read channel 0 into temp variable
        samples = samples + temp ' Accumulate 20 samples
        PAUSE 250 ' Wait approximately 1/4 seconds per loop
    NEXT sample
    temp = samples/20
    DEBUG "Temperature is: ",DEC temp," Deg F",10,13
    samples = 0 ' Clear old sample accumulator
    GOTO loop ' Do it forever

END

```



نمایش نتایج حاصل ارسال شده از میکروکنترلر به کامپیوتر از طریق پنجره ارتباط سریال نرم افزار

MicroCode Studio

مشخصات و قابلیت‌های میکروکنترلر PIC16F877

این میکروکنترلر دارای ۴۰ پایه می باشد که در شکل زیر نشان داده شده است.

مشخصات داخلی میکروکنترلر :

- عملکرد بالای CPU ، RISC
- تنها ۳۵ دستور برای یادگیری دارد
- همه دستورات در یک سیکل اجرا می شوند بجز دستورات گزینشی
- سرعت عملکرد آن از DC تا 20MHz می باشد
- حافظه برنامه نویسی آن از نوع FLASH است (۱۴ * ۸ کلمه)
- حافظه RAM ۸ * ۳۶۸ بایت
- حافظه EEPROM ۸ * ۲۵۶ بایت
- قابلیت وقفه (۱۴ منبع)
- هشت سطح برای پشته
- مدهای آدرس دهی مستقیم ، غیر مستقیم و نسبی
- ریست شدن در هنگام وصل شدن منبع تغذیه (POR)
- تایمر روشن نگهدار (PWRT) و تایمر شروع به کار اسیلاتور (OSC)
- تایمر Watchdog (WDT) با اسیلاتور RC مجزا بر روی تراشه
- مد SLEEP برای کم مصرف کردن انرژی
- قابل انتخاب بودن اسیلاتور
- قابلیت برنامه ریزی درون مدار با استفاده از دو پایه (ICSP)

○ قابلیت برنامه ریزی درون مدار با ولتاژ ۵ ولت

○ خواندن و نوشتن حافظه برنامه ریزی

○ عملکرد در ولتاژهای بین ۲ تا ۵,۵ ولت

○ جریان Sink و Source ۲۵ میلی آمپر

○ مصرف توان کم

- <2mA typical @ 5V, 4 MHz

- 20mA typical @ 3V, 32 kHz

- <1mA typical standby current

مشخصات جانبی

○ تایمر ۰: تایمر و یا شمارنده ۸بیتی با ۸ بیت prescaler

○ تایمر ۱: تایمر و یا شمارنده ۱۶ بیتی با prescaler می تواند در حالت sleep با کلاک یا

کریستال خارجی کار کند

○ تایمر ۲: تایمر و یا شمارنده ۸بیتی با رجیستر ۸ بیتی prescaler و postscaler

○ دو Capture و مقایسه کننده و مولد PWM (CCP)

Capture ۱۶ بیتی است با حد تفکیک ۱۲,۵ نانوثانیه

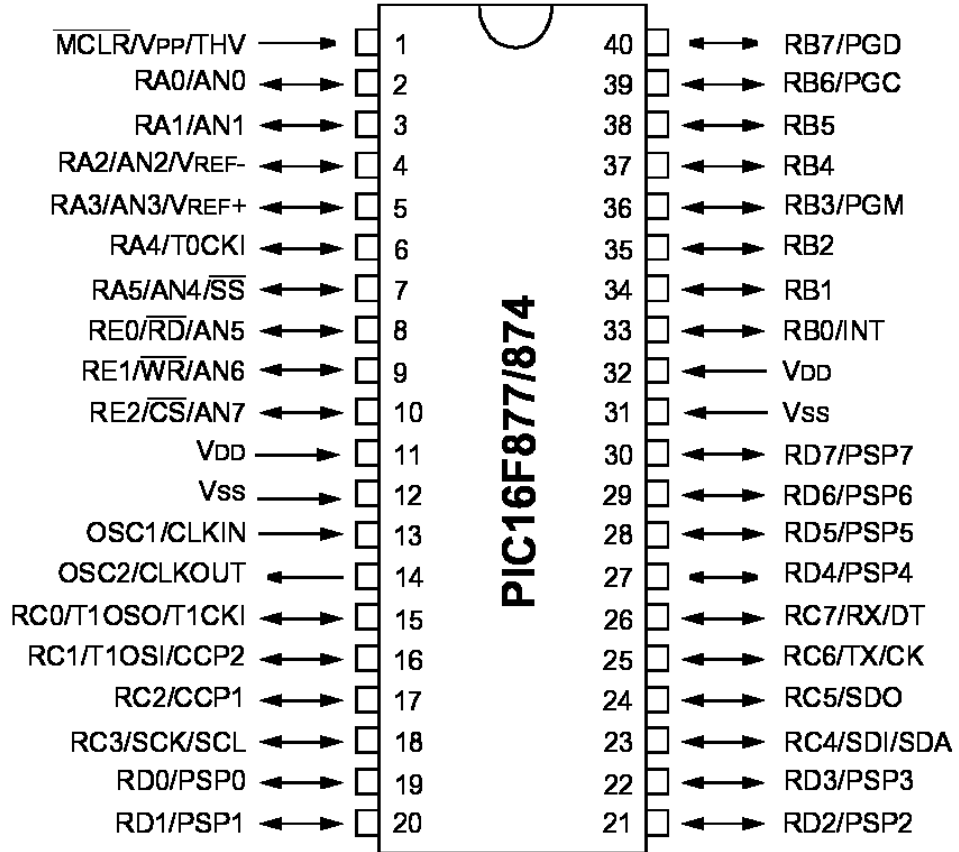
مقایسه کننده ۱۶ بیتی است با حد تفکیک ۲۰۰ نانوثانیه

PWM با حدتفکیک ۱۰ بیت

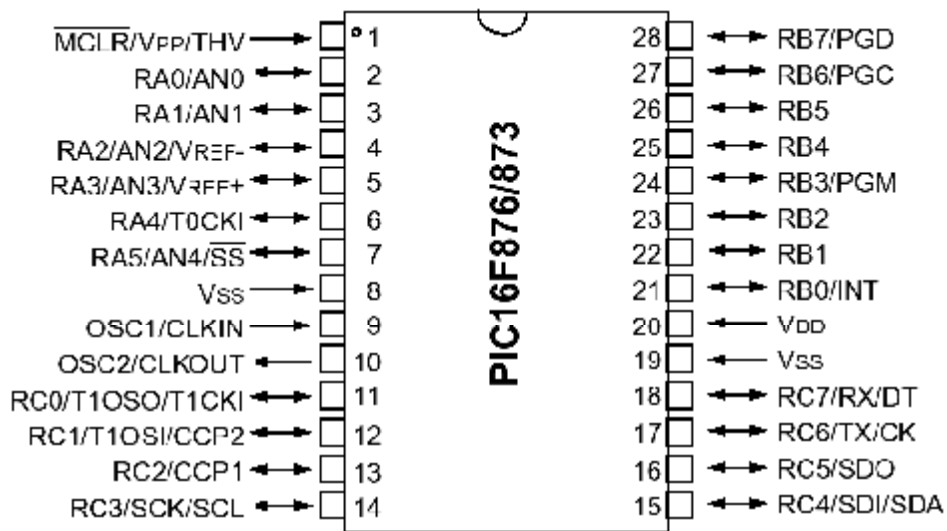
○ مبدل A/D چند کاناله با دقت ۱۰ بیت

○ پورت Slave موازی (PSP) با ۸بیت پهنا با پایه های کنترلی \overline{RD} ، \overline{WR} و \overline{CS} خارجی

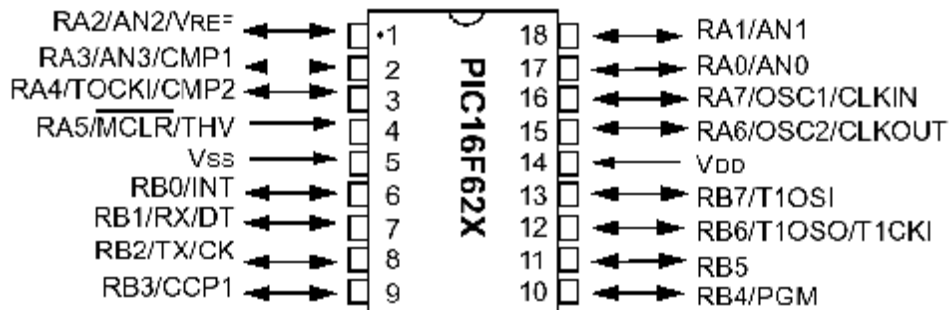
○ آشکار سازی کاهش ولتاژ منبع برای ریست کردن (BOR)



شکل 1-1) میکروکنترلر PIC16F877



شکل 2-2 میکروکنترلر PIC16F876 / 873



شکل 3-3 میکروکنترلر PIC16F62X

جدول دستورات بیسیک :

| ردیف | دستور | تشریح |
|------|-------------------------|--|
| ۱ | @ | وارد کردن یک خط برنامه با کد اسمبلی |
| ۲ | ASM ... ENDASM | وارد کردن یک بلوک از دستورات اسمبلی |
| ۳ | ADCIN | دریافت مقادیر از ورودی مبدل A/D |
| ۴ | BRANCH | پریدن به برجسب مشخص شده توسط شاخص |
| ۵ | BRANCHL | پرشی بلند به برجسب نظیر شاخص |
| ۶ | BUTTON | خواندن حالت دکمه روی پین ورودی |
| ۷ | CALL | فراخوانی زیر برنامه های اسمبلی |
| ۸ | CLEAR | مقدار همه متغیرها را به صفر تغییر دادن |
| ۹ | CLEARWDT | باز نشاندن تایمر Watchdog |
| ۱۰ | COUNT | شمردن پالسهای روی پین ورودی |
| ۱۱ | DATA | نوشتن در EEPROM داخلی در ابتدای برنامه |
| ۱۲ | DTMFOUT | تولید کردن سیگنال صدا (تون) شمارگیری روی پین خروجی |
| ۱۳ | EEPROM | مجموعه ثابتهای اولیه برای برنامه نویسی EEPROM |
| ۱۴ | END | نشانه گذاری پایان برنامه |
| ۱۵ | FOR ... NEXT | تکرار کردن یک قسمت از برنامه |
| ۱۶ | FREQOUT | تولید سیگنال با فرکانس مشخص روی پین خروجی |
| ۱۷ | GOSUB | فراخوانی زیر برنامه های BASIC |
| ۱۸ | GOTO | ادامه برنامه از برجسب مشخص شده |
| ۱۹ | HIGH | نشاندن یک منطقی روی پین خروجی |
| ۲۰ | HESERIN | سخت افزار ورودی آسنکرون |
| ۲۱ | HPWM | تولید کردن سیگنال PWM روی پین میکروکنترلر |
| ۲۲ | HSEROUT | سخت افزار خروجی آسنکرون |
| ۲۳ | I2CREAD | خواندن اطلاعات از وسایل جانبی I2C |
| ۲۴ | I2CWRITE | نوشتن اطلاعات بر روی وسایل جانبی I2C |
| ۲۵ | INPUT | برگزیدن پین I/O در گرایش ورودی |
| ۲۶ | IF ... THEN ... ELSE | گزینه یک قسمت از برنامه |
| ۲۷ | LCDOUT | نوشتن اطلاعات بر روی نمایشگر LCD |
| ۲۸ | LOOKDOWN | جستجو کردن جدول ثابتها |

| تشریح | دستور | ردیف |
|---|-----------------|------|
| آوردن مقداری از جدول ثابتها | LOOKUP | ۲۹ |
| قرار دادن صفر منطقی در پین خروجی | LOW | ۳۰ |
| خاموش کردن برای یک دوره زمانی کوتاه | NAP | ۳۱ |
| گماشتن پین I/O در گرایش خروجی | OUTPUT | ۳۲ |
| دریافت اطلاعات از طریق ارتباط یک سیم | OWIN | ۳۳ |
| ارسال اطلاعات از طریق یک سیم | OWOUT | ۳۴ |
| تاخیر برحسب میلی ثانیه | PAUSE | ۳۵ |
| تاخیر برحسب میکروثانیه | PAUSEUS | ۳۶ |
| برگرداندن مقدار مقاومت متصل به پین | POT | ۳۷ |
| محاسبه عرض پالس روی پایه ورودی | PULSIN | ۳۸ |
| تولید کردن پالس در پین خروجی | PULSOUT | ۳۹ |
| تولید سیگنال PWM روی پین | PWM | ۴۰ |
| تولید عدد تصادفی | RANDOM | ۴۱ |
| محاسبه پالس روی پین | RCTIME | ۴۲ |
| خواندن یک بایت از اطلاعات EEPROM | READ | ۴۳ |
| خواندن ۲ بایت (WORD) از کد برنامه | READCODE | ۴۴ |
| تغییر گرایش پین | REVERSE | ۴۵ |
| ورودی سریال آسنکرون | SERIN | ۴۶ |
| خروجی سریال آسنکرون | SEROUT | ۴۷ |
| ورودی سریال سنکرون | SHIFTIN | ۴۸ |
| خروجی سریال سنکرون | SHIFTOUT | ۴۹ |
| خاموش کردن پردازنده برای یک پریود زمانی معین | SLEEP | ۵۰ |
| تولید کردن صدا یا نویز سفید روی یک پین مشخص | SOUND | ۵۱ |
| توقف برنامه در حال اجرا | STOP | ۵۲ |
| مبادله مقادیر دو متغیر | SWAP | ۵۳ |
| وارون کردن وضعیت پین | TOGGLE | ۵۴ |
| نوشتن اطلاعات در EEPROM داخلی | WRITE | ۵۵ |
| نوشتن دوبایت از اطلاعات بر روی حافظه برنامه | WRITECODE | ۵۶ |
| اجرای قسمتی از برنامه تا وقتی که شرط برقرار است | WHILE – WEND | ۵۷ |