مقدمه :

تاریخچه میکروکنترولر PIC : میکروکنترولر PIC در اصل در حدود سال 1980 توسط شرکت جنرال اینسترومنت[1] به عنوان یک میکروکنترولر کوچک ، سریع و ارزان که توانایی های I /O قوی دارد طراحی شد.

واژه PIC مخفف "Peripheral Intrerface Controller" به معنای « کنترولر رابط محیطی » می باشد . شرکت جنرال اینسترومنت که پتانسیل ویژه ای را برای این میکروکنترولر تشخیص داده بود در نهایت بخشی از سهام و شعبات خود را برای تبدیل به یک شرکت جدید به نام میکرو چیپ[2] ، بین سهام داران تقسیم کرد تا اینکه در این شرکت به ساخت و فروش محصولات PIC بپردازند . (که البته امروزه محصولات متنوعی در این شرکت تولید می شود)

میکرو کنترولر PIC در بسیاری از کاربردها دارای مزایایی نسبت به تراشه ههای قدیمی تر 8052 / 8051 / 8041 و مشتقات آن از شرکت اینتل یا تراشه های MC6805/6hHc11 از شرکت موتورولا و بسیاری دیگر از میکروهای شرکت های دیگر می باشد . معماری غیرمعمول آن برای کاربردهای کنترلی سازگاری یافته است . تقریباً تمام دستورات آن در تعداد یکسانی از سیکل های ساعت اجرا می شود که این امر به نوبه خود کنترل زمانی (Timing Control) را بسیار آسان تر می کند .میکروکنترولر PIC یک طراحی RISC (Reduced Instruction Set Computer) است که تنها در حدود 35 دستورالعمل دارد که این خود یادگیری برنامه نویسی آن را آسان تر می کند .

در واقع قیمت پایین ، سرعت های کلاک بالای قابل دست یابی ، اندازه کوچک و راحتی استفاده از میکروکنترولر PIC نکات مهمی در افزایش روزافزون به کارگیری این میکروکنترولر می باشد . (برای طراحی هایی که حساسیت کمتری نسبت به زمان دارند) سرعت کلاک می تواند از سرعت های پایین تا حدود رنج 20 MHz باشد .

خانواده های مختلف میکروکنترولر های PIC دارای ترکیب های مختلفی از ROM ، EPROM ، OTP(One - Time - Programable) EPROM ، EEPROM ، FLASH Program و حافظه داده

---

[1] - Genral Instrument

[2] - MicroChip

مـی باشـد بنابرایـن در بسـیاری از موارد ، طراحی با میکروکنترولرهای PIC بسیار سودمندتر و مقرون به صرفه تر از طراحی با میکروکنترولرهای قدیمی تر و بزرگ تر می باشد .

کاربـردهای میکروکنـترولر PIC : میکروکنترولرهای PIC در رنج باورنکردنی از محصولات یافت مـی شـوند . کنترل از راه دور ها ، پانل های نمایش ، اتوموبیل ها ، وسایل خانگی ، اییستگاه های هـوا شناسـی ، تجهـیزات فرسـتنده مـوج کوتـاه رادیویی ، ساعت ها ، کنترل کننده های موتور ، سنسـور هـا ، ترموسـتات هـای قـبال بـرنامه ریزی ، ربات ها ، اسباب بازی ها ، شارژ کننده های باتـری و تقریـباً هـر چـیزی کـه در آن نوعـی بـرنامه پذیـری منطقـی بـه کـار رفته باشد ، یافت می شود .

خلاصه سخت افزار :

PIC16F84 میکروکنترولری هشت بیتی و 18 پایه است که دارای حافظه FLASH/EEPROM می باشد .

مشخصات پردازشگر این میکروکنترولر :

- CPU آن RISC است .
- تنها 35 دستورالعمل دارد .
- تمام دستورالعمل ها یک سیکلی (Single Cycle) هستند . به جز آن دستورالعمل هایی که مربوط به انشعابات برنامه می باشند که دو سیکلی هستند .
- سرعت اجرا :

کلاک ورودی DC – 10 MHz

سیکل دستورالعمل DC – 400 ns

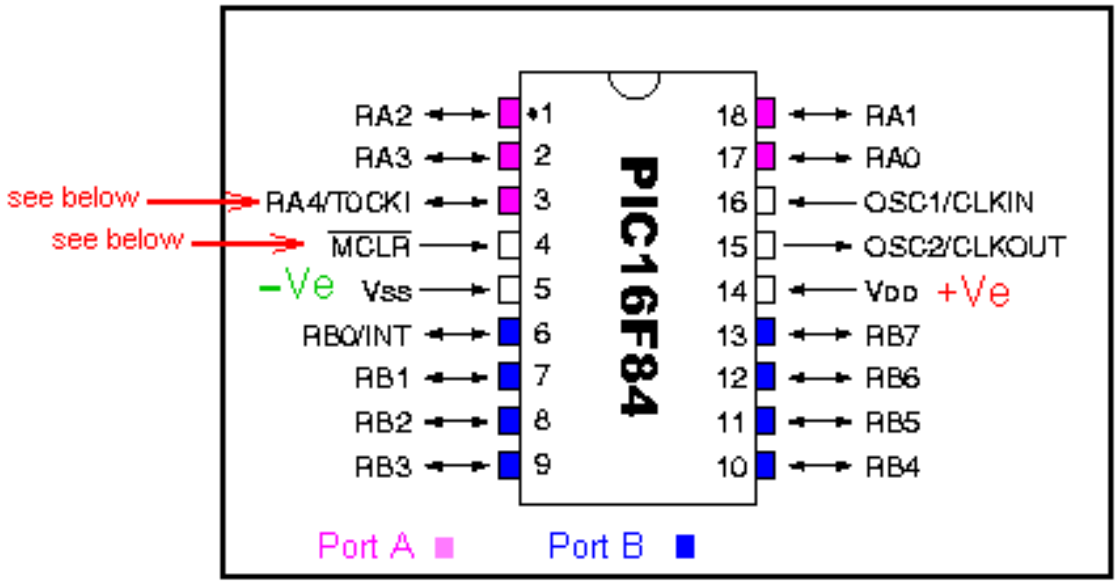
Figure - 1

Device	Program Memory (words)	Data RAM (bytes)	Data EEPROM (bytes)	Max. Freq (MHz)
PIC16F84	1 K Flash	68	64	10

1024 lines of code!
(1024 instructions!!)

- طول هر بایت داده (Data Patch) هشت بیت می باشد .
- دارای 15 ثبات سخت افزاری کاربرد خاص (SFR) می باشد .
- دارای پشته سخت افزاری هشت سطحی (Level – Eight) می باشد .
- دارای قابلیت آدرس دهی به روش های مستقیم ، غیر مستقیم و آدرس دهی نسبی می باشد .
- دارای چهار منبع وقفه به قرار زیر می باشد :
  - وقفه خارجی از پایه RB0/INT
  - وقفه سرریز تایمر TMR0
  - وقفه ناشی از تغییر در پایه های چهار تا هفت پورت B
  - وقفه ناشی از تکمیل عملیات نوشتن در EEPROM

- توانایی حدود 1000 بار نوشتن و پاک کردن حافظه برنامه FLASH .
- توانایی حدود 10000000 بار نوشتن و پاک کردن حافظه داده EEPROM .



مشخ

- 3

- جریان بالای Sink / Source برای درایو مستقیم LED .
- حداکثر جریانی که هر پایه می تواند بکشد 25 ma است .
- حداکثر جریانی که به هر پایه می توان داد 20 ma است .
- TMR0 : یک شمارنده / تایمر هشت بیتی همراه با Prescaler برنامه پذیر هشت بیت است .

مشخصات خاص میکروکنترولر :

- ICSP (In – Cicuit Serial Programing) قابلیت برنامه ریزی سریال در داخل مدار توسط دو پایه
- قابلیت Reset – On – Power (POR)
- قابلیت Timer – Up – Power (PWRT)
- قابلیت انتخاب نوع اسیلاتور
- دارابودن WatchDog Timer ، که مجهز به اسیلاتور RC اختصاصی بوده که به این ترتیب آن را قابل اطمینان تر می کند .
- قابلیت Sleep برای جلوگیری از مصرف توان زیاد .
- قابلیت حفاظت از کد نوشته شده در حافظه برنامه .

		PIC16F84
Clock	Maximum Frequency of Operation (MHz)	10
Memory	Flash Program Memory	1K
	EEPROM Program Memory	—
	ROM Program Memory	—
	Data Memory (bytes)	68
	Data EEPROM (bytes)	64
Peripherals	Timer Module(s)	TMR0
	Interrupt Sources	4
Features	I/O Pins	13
	Voltage Range (Volts)	2.0-6.0
	Packages	18-pin DIP, SOIC

Table – 1 PIC16F8X Family Devices

Pin Name	DIP No.	SOIC No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	I	ST/CMOS (3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$	4	4	I/P	ST	Master clear (reset) input/programming voltage input. This pin is an active low reset to the device.
					PORTA is a bi-directional I/O port.
RA0	17	17	I/O	TTL	
RA1	18	18	I/O	TTL	
RA2	1	1	I/O	TTL	
RA3	2	2	I/O	TTL	
RA4/T0CKI	3	3	I/O	ST	Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	6	6	I/O	TTL/ST (1)	RB0/INT can also be selected as an external interrupt pin.
RB1	7	7	I/O	TTL	
RB2	8	8	I/O	TTL	
RB3	9	9	I/O	TTL	
RB4	10	10	I/O	TTL	Interrupt on change pin.
RB5	11	11	I/O	TTL	Interrupt on change pin.
RB6	12	12	I/O	TTL/ST (2)	Interrupt on change pin. Serial programming clock.
RB7	13	13	I/O	TTL/ST (2)	Interrupt on change pin. Serial programming data.
Vss	5	5	P	—	Ground reference for logic and I/O pins.
VDD	14	14	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = Input/Output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

Table – 2 PIC16F8X PinOut Description

مروری بر معماری میکروکنترولر PIC :

کارایی بالای میکروکنترولرهای خانواده PIC را می توان به برخی از ویژگیهای خاص معماری ، که به طور عادی در پردازشگرهای RISC یافت می شود نسبت داد.در ابتدا باید بیان کرد که میکروکنترولرهای خانواده PIC دارای معماری هاروارد می باشند .به این معنی که در این نوع معماری حافظه داده و برنامه جدا از یکدیگر قرار دارند وقابل دسترسی هستند .بنابراین میکروکنترولر دارای یک گذرگاه داده و یک گذرگاه برنامه به طور جداگانه می باشد.

در این نوع معماری پهنای باند نسبت به معماری وان نیومن[1] که در آن داده و برنامه از یک حافظه مشترک برداشت[2] می شوند افزایش می یابد . علاوه بر این جدا بودن حافظه داده از برنامه این اجازه را می دهد که دستورالعمل ها در اندازه ایی متفاوت از اندازه هشت بیتی کلمات داده باشد.کدهای عملیاتی PIC16CXX , 14 بیتی هستند بنابراین یک گذرگاه حافظه برنامه 14 بیتی یک دستورالعمل 14 بیتی را در یک سیکل برداشت می کند .

PIC16CXX دارای یک ALU هشت بیتی و ثبات های کاری می باشد . ALU در واقع یک واحد محاسباتی همه منظوره است که اعمال ریاضی و منطقی را بر روی داده در ثباتهای کاری و هر یک از رجیستر فایلها انجام می دهد .این ALU هشت بیتی قادر به انجام عملیات جمع ، تفریق ، شیفت و عملیات منطقی می باشد .باید توجه داشت که این عملیات ریاضی به شیوه مکمل دو انجام می شوند.در دستورالعملهای دو عملوندی معمولا یک عملوند ، یک ثبات کاری بوده و عملوند دیگر یک رجیستر فایل یا یک مقدار ثابت می باشد.در دستورالعملهای یک عملوندی ، عملوند می تواند یک ثبات کاری یا یک رجیستر فایل باشد.یک بلوک دیاگرام ساده شده از PIC16F8X در شکل – 3 نشان داده شده است .

---

۱ - Von Neuman

۲ - Fetch

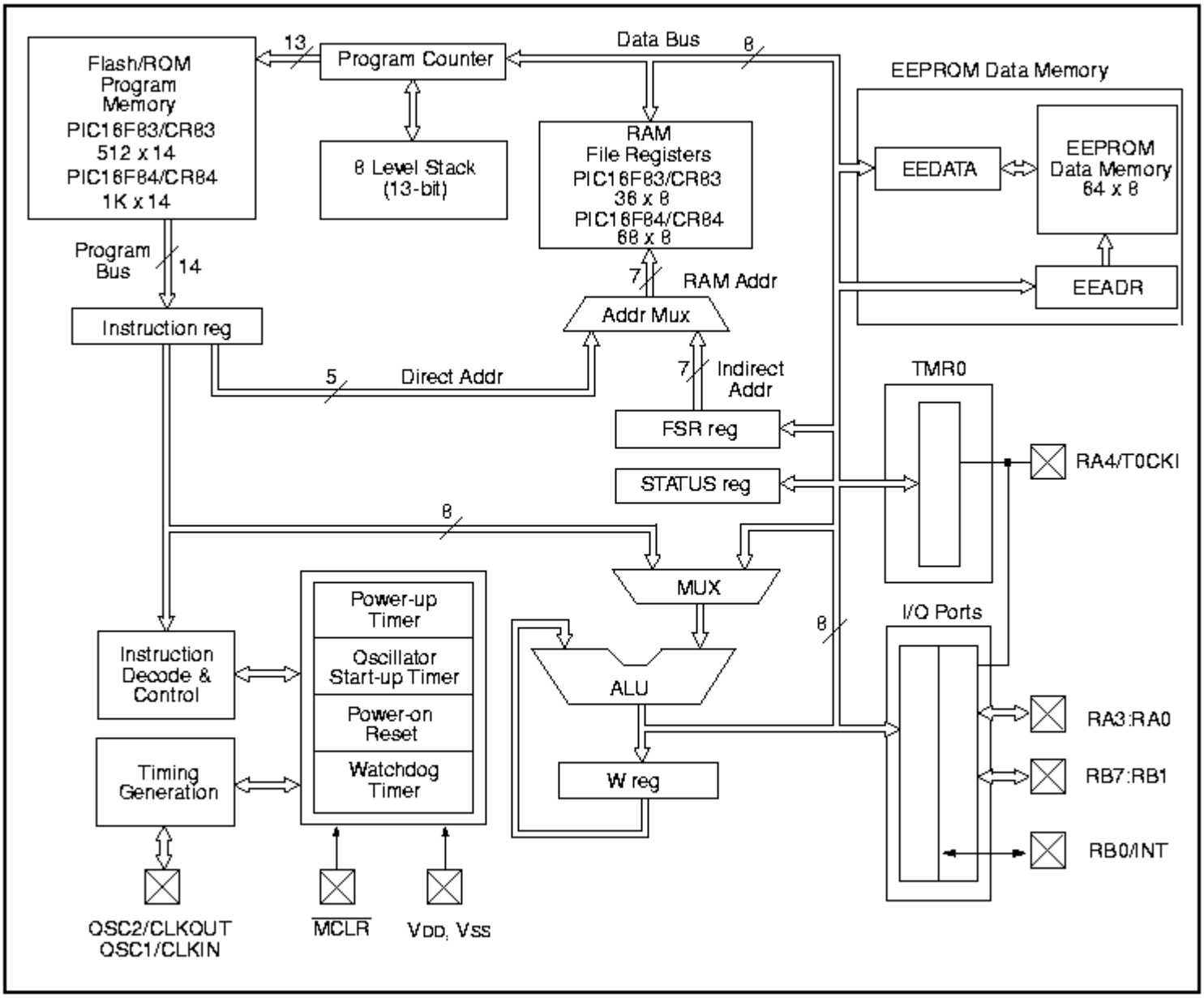


Figure – 3 PIC 16F8X Block Diagram

ساختار حافظه :

دو واحـد حافظـه در PIC16F84 وجـود دارد که عبارتند از حافظه برنامه و حافظه داده . هر واحد حافظه دارای گذرگاه مخصوص به خود می باشد و بنابراین دست رسی به هر واحد در طی همان سیکل اسیلاتور ممکن می باشد .

حافظـه داده نـیز خـود بـه دو قسمت RAM همه منظوره و رجیستر های با کاربرد خاص (SSR) تقسـیم مـی شـود . محـدوده حافظـه داده شـامل یـک قسـمت Data EEROM نیز می باشد . این قسـمت بـه طـور مسـتقیم به حافظه داده نگاشت نشده است بلکه به صورت غیر مستقیم نگاشت شده است به همین خاطر یک اشاره گر آدرس غیر مستقیم ، آدرسی از حافظه EEPROM را که قـرار اسـت نوشـته یا خوانده شود مشخص می کند . 64 بایت حافظه Data EEPROM آدرسی در محدوده 00 H تا 3F H دارد .

ساختار حافظه برنامه :

PIC16FXX یـک شـمارنده برنامه 13 بیتی دارد که توانایی آدرس دهی به فضای حافظه ای برابر بـا 14 bit * 8 k را دارا مـی باشد . در PIC16F83 و PIC16CR83 اولین ۵۱۲ × 14 bit یعنی محدوده آدرس (01FF H – 0000 H) بـه طـور فیزیکـی پـیاده سـازی شـده اسـت امـا در PIC16F84 و PIC16CR84 ، 1 k × 14 bit یعنی محدوده (03FF H – 0000 H) به طور فیزیکی پیاده سازی شده است .

دسترسـی بـه مکانی از حافظه که بالاتر از این محدوده که به طور فیزیکی پیاده سازی شده است موجـب یـک چـرخش بـه مکـان هـای پاییـن تـر حافظـه می شود ، به عنوان مثال در PIC16F84 مکـان هـای حافظـه 20 H , 420 H , 820 H , C20 H , 1020 H, 1420 H , 1820 H , 1C20 H همگـی بـیانگر یـک آدرس از حافظه می باشند که همان آدرس 20 H است . که در محدوده پیاده سازی شده فیزیکی می باشد .

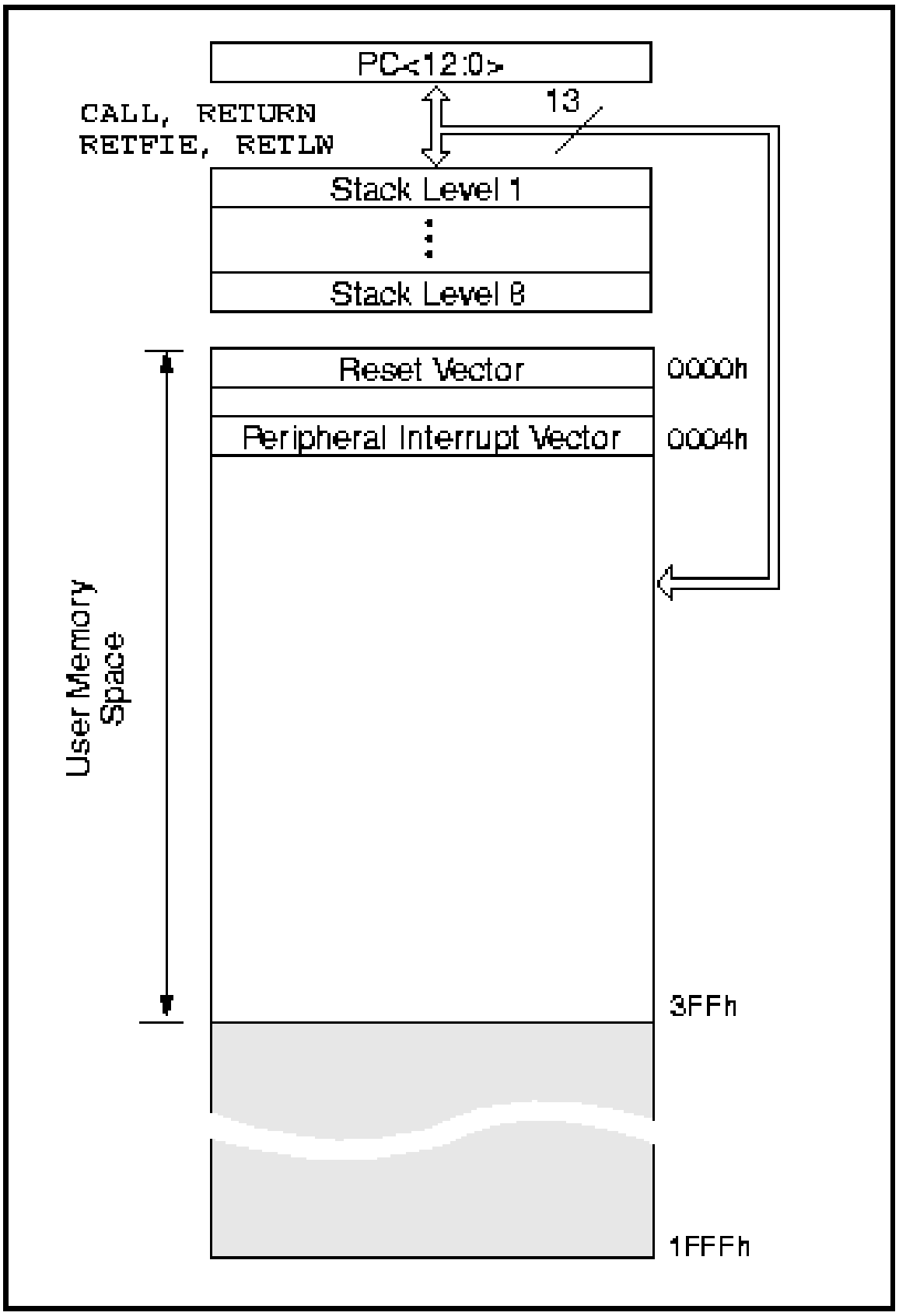


Figure – 4 Program Memory Map & Stack
PIC16F84 / CR84

ساختار حافظه داده :

حافظـه داده بـه دو قسـمت تقسـیم شـده اسـت . قسـمت اول ناحـیه ثـبات هـای با کاربرد خاص (SFR) اسـت و ناحـیه دوم به نام ثبات های با کاربرد عمومی (GPR)شناخته می شود . SFRها عملکرد میکروکنترولر را کنترل می کنند .

حافظـه داده بـه بخـش هایـی به نام بانک تقسیم شده است که شامل هر دو قسمت SFR و GPR مـی باشـد . برای انتخاب بانک ها باید از بیت های کنترلی استفاده کرد این بیت های کنترلی در ثبات STATUS قرار دارند .

دسترسـی بـه حافظـه داده بـه طـور مسـتقیم از طـریق دادن آدرس دقـیق هـر ثـبات یـا به طور غـیر مسـتقیم از طـریق ثـبات FSR امکـان پذیـر اسـت . در آدرس دهی غیر مستقیم با توجه به مقدار RP0 دسترسی به بانک فعال امکان پذیر است .

در این جا حافظه داده به دو بانک تقسیم شده است انتخاب بانک صفر از طریق صفر کردن بیت RP0 از ثـبات STATUS صـورت مـی گـیرد و همچنین با یک کردن این بیت ، بانک یک انتخاب خواهـد شـد . بـه طـور کلـی 12 بایـت ابتدایـی هـر بـانک به ثبات های با کاربرد خاص و بقیه به ثبات های عمومی تعلق دارد .

ساختمان ثبات ها :

۱ – ثـبات هـای همـه مـنظوره (GPR) : همـه میکروکنـترولرها دارای مقـداری از ثـبات های با کاربـرد عمومـی مـی باشـند هـر ثبات با کاربرد عمومی 8 بیتی است و به صورت مستقیم یا غیر مسـتقیم و از طـریق FSR قـابل دسترسی است . آدرس های GPR در بانک یک به همان آدرس هـا در بـانک صـفر نگاشت شده اند . به عنوان مثال آدرس مکان های 0C H یا 8C H هر دو یک GPR را نشانه می روند .

۲– ثـبات هـای بـا کاربـرد خـاص (SFR) : SFR هـا توسـط CPU و ادوات جانبـی بـرای کنترل عملکـرد میکروکنـترولر اسـتفاده می شوند . این ثبات ها از نوع SRAM هستند . این ثبات ها را مـی تـوان بـه دو مجموعـه « مرکزی » و « محیطی » تقسیم نمود که دسته اول برای کنترل حالت عملکرد CPU و دسته دوم برای کنترل وسایل جانبی میکروکنترولر مثل تایمر به کار می رود .

ثبات STATUS :

ایـن ثـبات شـامل بیـت های بیان کننده حالت عملکرد ریاضی ALU ، حالت RESET و بیت های انـتخاب کنـنده بانک فعال حافظه داده می باشند . بیت های T0 و PD در این ثبات فقط خواندنی می باشند و نمی توان چیزی در آن ها نوشت .

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
0 = Bank 0, 1 (00h - FFh)
1 = Bank 2, 3 (100h - 1FFh)
The IRP bit is not used by the PIC16F8X. IRP should be maintained clear.

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
10 = Bank 2 (100h - 17Fh)
11 = Bank 3 (180h - 1FFh)
Each bank is 128 bytes. Only bit RP0 is used by the PIC16F8X. RP1 should be maintained clear.

bit 4: **$\overline{TO}$**: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3: **$\overline{PD}$**: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{borrow}$ bit (for ADDWF and ADDLW instructions) (For $\overline{borrow}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{borrow}$ bit (for ADDWF and ADDLW instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{borrow}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Figure – 5 Status Register (Address 03 H , 83 H)

ثبات OPTION – REG :

این ثبات یک ثبات خواندنی و نوشتنی است وشامل بیت های کنترلی مختلف برای تعیین حالات TMR0 / WDT Prescaler ، وقفه خارجی،تایمرصفروفعال کردن Pull UP درگاه B می باشد .

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

bit 7: **$\overline{RBPU}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled (by individual port latch values)

bit 6: **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5: **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4: **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3: **PSA**: Prescaler Assignment bit
1 = Prescaler assigned to the WDT
0 = Prescaler assigned to TMR0

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

Figure – 6 Option_Reg Register
(Address 81 H)

ثبات INTCON :

این ثبات شامل بیت های فعال ساز مختلف برای تمامی منابع وقفه می باشد .

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

bit 7: **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

Note: For the operation of the interrupt structure, please refer to Section 8.5.

bit 6: **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 5: **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4: **INTE**: RB0/INT Interrupt Enable bit
1 = Enables the RB0/INT interrupt
0 = Disables the RB0/INT interrupt

R	= Readable bit
W	= Writable bit
U	= Unimplemented bit, read as '0'
- n	= Value at POR reset

bit 3: **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2: **T0IF**: TMR0 overflow interrupt flag bit
1 = TMR0 has overflowed (must be cleared in software)
0 = TMR0 did not overflow

bit 1: **INTF**: RB0/INT Interrupt Flag bit
1 = The RB0/INT interrupt occurred
0 = The RB0/INT interrupt did not occur

bit 0: **RBIF**: RB Port Change Interrupt Flag bit
1 = When at least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Figure – 7 INTCON Register (Address 0B H , 8B H)

شمارنده برنامه : PCL و PCLATH

شمارنده برنامه (PC) در این میکروکنترولر 13 بیتی است . بایت کم ارزش آن در ثبات PCL ، که یک ثبات قابل خواندن و نوشتن است قرار دارد . 5 بیت بالایی شمارنده برنامه (PC < 12:8 >) به طور مستقیم نه قابل خواندن و نه قابل نوشتن است و در ثبات PCLATH قرار دارد . ثبات PCLATH (PC Latch High) یک ثبات نگه دارنده برای بیت های 8 تا 12 شمارنده برنامه می باشد .

هنگامی که شمارنده برنامه توسط یک مقدار جدید بارگذاری می شود ، بایت بالای آن (5 بیت بالایی آن) توشط محتویات ثبات PCLATH جایگزین می شود که این عمل در طول اجرای دستوراتی مانند CALL ، GOTO یا نوشتن در ثبات PCL صورت می پذیرد . بارگذاری PC در حالت های مختلف در شکل زیر نشان داده شده است :

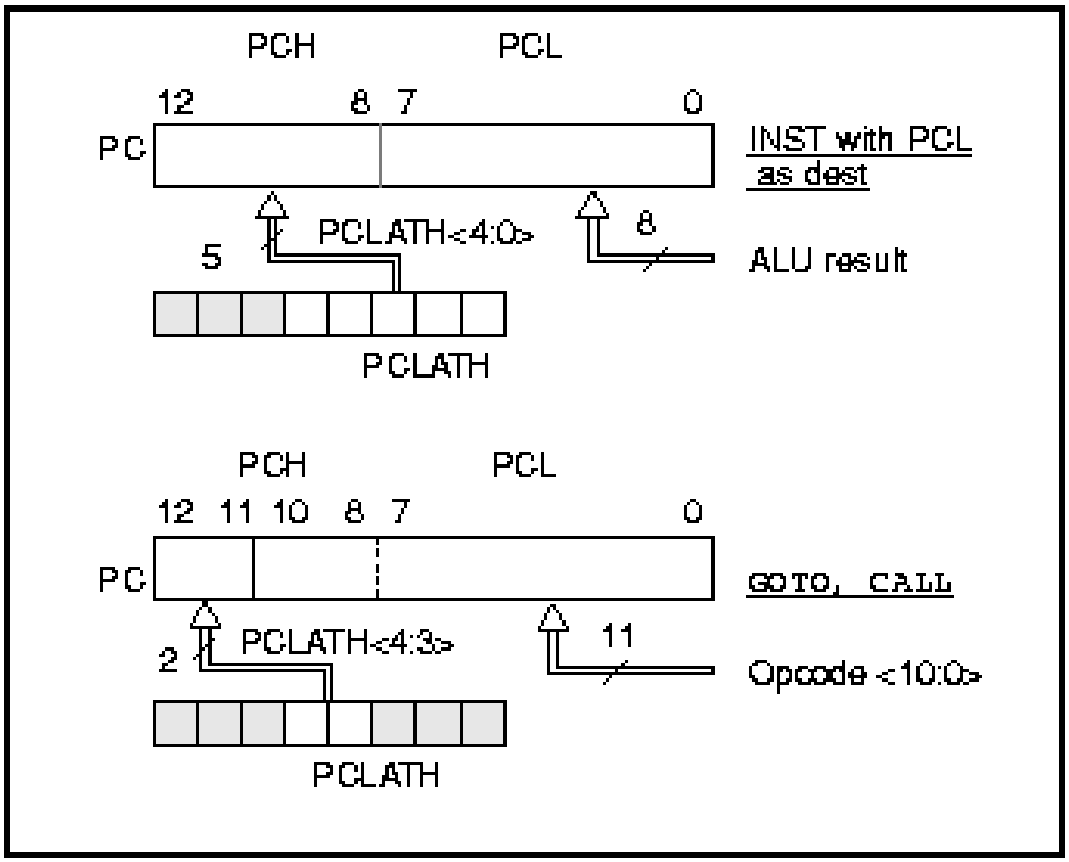


Figure – 8 Loading Of PC In Difference Situation

صفحه بندی حافظه برنامه :

میکروکنترولرهای PIC16F83 ، PIC16CR83 ، 512 کلمه حافظه برنامه دارند این در حالی است که PIC16F84 و PIC16CR84 ، یک کیلو از حافظه برنامه را دارا می باشند . دستورالعمل های CALL و GOTO یک آدرس 11 بیتی را در خود به همراه دارند که این 11 بیت آدرس ، اجازه پیمایش در یک صفحه 2 کیلویی از حافظه برنامه را می دهد . اما برای توسعه حافظه برنامه در یک PIC16F8X در آینده ، نیاز به لااقل دوبیت دیگر برای مشخص کردن صفحه مورد نظر در حافظه برنامه می باشد .

این بیت های صفحه بندی از بیت های سه و چهار ثبات PCLATH می آید (شکل قبل) . هنگامی که دستورالعمل هایی مانند CALL و GOTO قرار است که انجام شوند ، کاربر باید مطمئن باشد که این بیت های صفحه بندی ، برای پیمایش در صفحه درست حافظه برنامه ، برنامه ریزی شده اند .

هنگامی که یک دستور CALL و یا یک وقفه اجرا می شود تمام 13 بیت PC داخل پشته PUSH شده و بنابراین هنگام بازگشت به برنامه اصلی دیگر نیازی به دستکاری ثبات PCLATH نیست چون دوباره همان 13 بیت PC از حافظه پشته POP می شود .

توجه :

PIC16F8X از بیت های سه و چهار PCLATH که برای مشخص کردن صفحات 1 , 2 , 3 (800 H – FFF H) به کار می رود چشم پوشی می کند . همچنین استفاده از بیت های سه و چهار PCLATH به عنوان بیت های R / W همه منظوره توصیه نمی شود زیرا ممکن است سازگاری مناسب با محصولات بعدی را تحت تأثیر قرار دهد .

حافظه پشته :

میکروکنترولر PIC16FXX یک حافظه سخت افزاری پشته 13* 8 بیتی دارد . فضای حافظه پشته نه بخشی از فضای حافظه برنامه است و نه بخشی از فضای حافظه برنامه داده . همچنین اشاره گر پشته نه قابل خواندن است و نه قابل نوشتن .

هنگامی که دستورالعملی مانند CALL اجرا می شود و یا وقفه ای اعلام می گردد تمام 13 بیت PC در حافظه پشت PUSH شده و هنگامی که دستورالعمل هایی از قبیلRETLW , RETURN و یا RETFIE اجرا می شود این سیزده بیت از پشته POP می شوند . باید توجه داشت که PCLATH توسط عملیات PUSH یا POP تغییر نمی کند .

> *توجه :*
> *به خاطر داشته باشید که هیچ دستورالعملی تحت نام PUSH و یا POP وجود ندارد . درواقع این اعمال اعمالی هستند که در ضمن اجرای دستوراتی مانند , RETLW CALL , RETURN و RETFIE اتفاق می افتند*

از این که پشته بصورت سرریز PUSH شد ، نهمین عمل PUSH کردن باعث می شود که مقداری که در اولین PUSH ، ذخیره شده بود از بین برود و دهمین PUSH مقداری را که در دومین PUSH ذخیره شده بود از بین می برد و به همین ترتیب ...

همچنین اگر پشته به طور مرتب ، نه بار POP شود در نهمین POP مقدار PC همان مقداری می شود که در اولین عمل POP شده بود .

> *توجه :*
> *هیچ بیت وضعیتی که Overflow و Underflow[1] را برای پشته مشخص کند وجود ندارد.*

ساختار حافظه DATA EEPROM :

این فضای حافظه یک نوع حافظه خواندنی و نوشتنی است . این قسمت از حافظه به طور مستقیم در فضای رجیستر فایل قرار ندارد و دسترسی به آن از طریق غیر مستقیم و SFR ها امکان پذیر

[1] - Cicular Buffer

است چهار SFR برای دسترسی به این حافظه و خواندن و نوشتن در آن مورد استفاده قرار می گیرند که عبارتند از EECON1 , EECON2 , EEDATA و EEADR .
EEDATA هشت بیت داده که باید نوشته و یا خوانده شود را در خود نگاه می دارد و EEADR آدرس مکان حافظه مورد نظر را در خود نگاه می دارد . PIC16F84 دارای 64 بایت حافظه DATA EEPROM با محدوده آدرس 00 H تا 3F H است .

ساختار حافظه FLASH EPROM :

در اوایل دهه1990 ، FLASH EPROM به عنوان یک تراشه حافظه قابل برنامه ریزی توسط کاربر متداول گردید . از آنجا که عمل پاک کردن همه محتویات آن در کمتر از یک ثانیه انجام می گیرد ، بنابراین ، این حافظه سریع (FLASH) نام گرفت . همچنین به دلیل این که روش پاک کردن آن الکتریکی است بعضی اوقات به آن FLASH EEPROM نیز می گویند اما برای جلوگیری از اشتباه به طور خلاصه به آن FLASH گفته می شود .
تفاوت اصلی بین EEPROM و حافظه FLASH این است که در عمل پاک کردن حافظه FLASH ، کل حافظه پاک می شود در حالی که در EEPROM ، می توان یک بخش یا یک بایت دلخواه را پاک کرد .
اخیراً حافظه هایی از نوع FLASH ساخته شده اند که در آنها محتویات به بلوک هایی تقسیم شده و عمل پاک شدن می تواند بلوک به بلوک انجام شود . ولی هنوز حق انتخاب برای پاک کردن بایت ها وجود ندارد . با توجه به این مطلب که حافظه FLASH می تواند در حالی که در سوکت روی بورد سیستم قرار دارد ، برنامه ریزی شود ، استفاده از این حافظه به عنوان روشی برای به روز درآوردن BIOS ROM در رایانه ها بسیار متداول شده است . زمان دست یابی حافظه های FLASH در حدود نانو ثانیه است .

آدرس دهی غیر مستقیم و ثبات های INDF و SFR :

ثبات INDF یک ثبات فیزیکی نیست . آدرس دهی به ثبات INDF در حقیقت ثباتی را که آدرسش در ثبات FSR می باشد ، آدرس دهی می کند . (FSR در واقع یک اشاره گر می باشد) بنابراین این نوع آدرس دهی را آدرس دهی غیر مستقیم می گویند .

مثال 1) آدرس دهی غیر مستقیم

- رجیستر فایل 05 شامل مقدار 10 H است .
- رجیستر فایل 06 شامل مقدار 0A H است .
- مقدار 05 را در ثبات FSR بارگذاری می کنیم .
- در این زمان خواندن ثبات INDF مقدار 10 H را برخواهد گرداند .
- یک واحد مقدار FSR را افزایش می دهیم . (FSR = 06)
- هم اکنون خواندن ثبات INDF مقدار 0A H را برخواهد گرداند .

مثال 2) یک برنامه ساده برای پاک کردن مکان های 20 H – CF H از RAM با استفاده از روش آدرس دهی غیر مستقیم در این مثال نشان داده شده است :

```
            movlw   0x20  ;initialize pointer
            movwf   FSR   ;   to RAM
NEXT        clrf    INDF  ;clear INDF register
            incf    FSR   ;inc pointer
            btfss   FSR,4 ;all done?
            goto    NEXT  ;NO, clear next
CONTINUE
            :             ;YES, continue
```

یک آدرس بیت هفت STATUS) به دست خواهد آمد . اگر چه IRP در PIC16F8X به کار برده نمی شود .

Figure – 9 Direct / InDirect Addressing

متـناظر با آن بیت در پورت A به عنوان ورودی مشخص شود و پاک نمودن بیت موجب می شود تا بیت متناظر در پورت A به عنوان خروجی در نظر گرفته شود .

در هـنگام خوانـدن از پورت A مقادیر پایه ها خوانده می شود و هنگام نوشتن در پورت ، مقادیر در LATCH پـورت نوشـته مـی شوند . پایه چهارم این پورت به صورت مالتی پلکس شده وظیفه ورودی CLOCK تایمر صفر را نیز بر عهده دارد و دارای خروجی Open Draim می باشد .

Name	Bit0	Buffer Type	Function
RA0	bit0	TTL	Input/output
RA1	bit1	TTL	Input/output
RA2	bit2	TTL	Input/output
RA3	bit3	TTL	Input/output
RA4/T0CKI	bit4	ST	Input/output or external clock input for TMR0. Output is open drain type.

Table – 3 PORTA Function

Legend: TTL = TTL input, ST = Schmitt Trigger input

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Table – 4 Summery Of Registers Associated With PORTA

Legend: x = unknown, u = unchanged, – = unimplemented read as '0'. Shaded cells are unimplemented, read as '0'

: TRISB

پورت B یک پورت 8 بیتی و دو طرفه است که تعیین پورت به عنوان ورودی و یا خروجی توسط وضعیت بیت های ثبات TRISB مشخص می شود که عملکردی مشابه ثبات TRISA برای پورت A دارد .

هـر یک از پین های پورت B دارای Pull – Up داخلی قابل برنامه ریزی است که توسط یک بیت کنترلی می توان تمامی این Pull – Up ها را فعال کرد که این عمل توسط صفر کردن بیت RBP0 از ثبات OPTION – REG صورت می گیرد .

ایـن Pull – Up هـا بـه صورت اتوماتیک و در هنگامی که پایه ها به عنوان خروجی در نظر گرفته مـی شـوند ، قطـع هسـتند . چهـار پایـه پـورت B (پایه های 4 تا 7) دارای ویژگی ارائه وقفه در صـورت تغییـر مقـدار هستند و تنها هنگامی که این پایه ها به عنوان ورودی در نظر گرفته شوند می توان از این ویژگی آن ها استفاده کرد .

Name	Bit	Buffer Type	I/O Consistency Function
RB0/INT	bit0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3	bit3	TTL	Input/output pin. Internal software programmable weak pull-up.

تایمر صفر :

تایمـر صـفر یـک تایمـر / شـمارنده 8 بیتـی قـابل خواندن و نوشتن است . امکان انتخاب کلاک داخلـی و خارجـی را دارد . امکـان انـتخاب لـبه بـالارونده و یا پایین رونده برای کلاک خارجی را دارد . همچنین فعال کردن وقفه در صورت سرریز تایمر وجود دارد .

حالت عملکرد تایمر صفر توسط بیت TOCS از ثبات OPTION – REG مشخص می شود . حالت تایمـر بـا صـفر کـردن ایـن بیت انتخاب می شود در هنگام استفاده از حالت شمارنده باید پالس خارجی اعمال شده به پایه RA4 با کلاک داخلی سیستم سنکرون می باشد .

واحد PRESCALER :

یـک شمارنده 8 بیتی است که در میکروکنترولر وجود دارد که می تواند برای واحد تایمر صفر و یا برای WatchDog Timer به کار رود . این واحد قابل خواندن و نوشتن نیست و پایه های PSA و PS0 : PS2 از ثـبات OPTION مشـخص مـی کـند کـه ایـن واحـد بـه کدام قسمت و با چه نسبتی اختصاص داده شده است .

صـفر نمـودن بیت PSA موجب اختصاص این واحد به تایمر صفر می شود و در این حالت مقادیر 1 : 2 ، 1 : 4 و 1 : 256 را می توان انتخاب کرد و با یک نمودن بیت PSA این واحد به WatchDog Timer اختصاص می یابد و مقادیر 1 : 1 ، 1 : 2 ، ... و 1 : 128 را می توان برای آن انتخاب کرد .

مثال 3) تغییر PRESCALER (Timer 0 → WDT)

```
BCF     STATUS, RP0  ;Bank 0
CLRF    TMR0         ;Clear TMR0
                     ; and Prescaler
BSF     STATUS, RP0  ;Bank 1
CLRWDT               ;Clears WDT
MOVLW   b'xxxx1xxx'  ;Select new
MOVWF   OPTION_REG       ; prescale value
BCF     STATUS, RP0  ;Bank 0
```

مثال 4) تغییر PRESCALER (WDT → Timer 0)

```
CLRWDT                  ;Clear WDT and
                        ; prescaler
BSF     STATUS, RP0     ;Bank 1
MOVLW   b'xxxx0xxx'     ;Select TMR0, new
                        ; prescale value
                        ' and clock source
MOVWF   OPTION_REG          ;
BCF     STATUS, RP0     ;Bank 0
```

مدارات ویژه

PIC16F84 دارای چند ویژگی برای بالا بردن توانایی سیستم ، کم کردن هزینه ، کم کردن نیاز به قطعات خارجی ، ذخیره توان و مواردی دیگر است .

- انتخاب نوع اسیلاتور
- Reset
- وقفه
- WatchDog Timer (WDT)
- حالت Sleep برای ذخیره توان
- حفاظت از کد
- ID Location

PIC16F84 دارای یک WatchDog Timer است که می تواند تنها با مقدار دهی به یک بیت سیستم را از نوع راه اندازی کند . این تایمر برای راه اندازی دارای یک اسیلاتور RC داخلی است که کاملاً مستقل از اسیلاتور سایر قسمت های سیستم است .

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
2007h	Config. bits	(2)	(2)	(2)	(2)	PWRTE(1)	WDTE	FOSC1	FOSC0	(2)	
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown. Shaded cells are not used by the WDT.
Note 1: See Figure 8-1 and Figure 8-2 for operation of the PWRTE bit.
2: See Figure 8-1, Figure 8-2 and Section 8.13 for operation of the Code and Data protection bits.

Table – 7 Summery Of Registers Associated With The WatchDog Timer

ایـن میکروکنـترولر همچنیـن دارای 2 تایمـر بـرای ایجـاد تأخـیرهای لازم در هنگام راه اندازی است . یک تایمر راه انداز اسیلاتور که سیستم را تا زمانی که اسیلاتور به حالت پایدار خود برسد در حالت غیر فعال نگاه می دارد . تایمر دیگر Power Up است که یک تأخیر ثابت 72 ms پس از هر بار روشن شدن سیستم فراهم می کند و این کار را برای این انجام می دهد که پس از روشن شـدن منـبع تغزیه بتواند به مقدار پایدار و ثابت خود برسد . با توجه به این دو تایمر در بسیاری از کاربرد ها نیاز به مدار خارجی برای انجام Reset از بین رفته است .

در حالـت Sleep جـریان بسیار کمی توسط میکروکنترولر کشیده شده و درنتیجه مصرف توان ، بسـیار پاییـن است . می توان میکروکنترولر را توسط Reset خارجی ، سرریز WDT و یا از طریق یک وقفه از حالتSleep خارج کرد .

ایـن میکروکنـترولر دارای چهار حالت اسیلاتور است که می توان با توجه به مورد مصرف ، یکی از آن هـا را نـتخاب کرد . انتخاب اسیلاتور RCهزینه سیستم را کاهش می دهد ، در صورتی که انـتخاب اسیلاتور LP مصرف توان را کاهش می دهد . در فرکانس های بالاتر اسیلاتور XT و HS را باید انتخاب کرد .

وقفه ها :

PIC16F84 دارای چهار منبع وقفه است :

۱ – وقفه خارجی

۲ – وقفه سرریز تایمر صفر

۳ – وقفه تغییر پورت B

۴ – وقفه اتمام عملیات نوشتن در DATAEEROM

ثـبات کنـترل وقفـه (INTCON) درخواسـت وقفـه هـای مـنفرد[۱] را در بیـت هـای پـرچم[۲] ظبط مـی کـند . همچنیـن ایـن ثـبات شامل بیت های فعال ساز وقفه های منفرد و وقفه های سراسری می باشد .

---

[۱] - Individual Interrupt

[۲] - Flag

بیت فعال ساز وقفه سراسری ، GIE ، (INTCON < 7 >) اگر Set شود تمام وقفه های آشکار شده[1] را فعال می سازد و اگر پاک شود تمام وقفه ها را غیر فعال می سازد . دستورالعمل بازگشت از وقفه ، RETFIE ، باعث خارج شدن از روال وقفه[2] می شود . اما به محض اینکه بیت GIE یک شود ، وقفه ها دوباره فعال شده و برنامه مجدداً وارد روال وقفه می شود .

پرچم وقفه های مختلف اعم از وقفه پایه RB0 / INT ، وقفه تغییر پورت B و وقفه سرریز تایمر صفر در ثبات کنترل تایمر (INTCON) قرار دارند . هنگامی که یک وقفه پاسخ داده می شود در این هنگام بیت GIE پاک می شود تا سایر وقفه ها در هنگام پاسخ دادن به آن وقفه ، غیر فعال شود . بنابراین آدرس بازگشت در اشاره گر پشته ذخیره شده و شمارشگر برنامه (PC) با مقدار H 0004 بارگذاری می شود . در هنگام روی دادن وقفه های خارجی همچون وقفه پایه RB0 / INT و یا وقفه تغییر پورت B ، Interrupt Latency سه الی چهار سیکل دستورالعمل به طول خواهد انجامید . Interrupt Latency برای دستورالعمل های یک سیکلی و دو سیکلی به طور یکسان انجام می شود . علاوه بر این در روال سرویس وقفه[3] منابع وقفه ها می توانند با استناد به بیت پرچم وقفه ها مشخص شوند . باید توجه داشت که بیت پرچم وقفه ها باید به طور نرم افزاری قبل از دوباره فعال شدن وقفه ها پاک شود تا از بروز درخواست های ناخواسته و تعریف نشده وقفه اجتناب شود .

توجه :

بیت پرچم وقفه های منفرد بدون توجه به وضعیت بیت پنهان متناظر با آن و یا بدون توجه به بیت GIE ، Set می شود .

وقفه خارجی در پایه RB0 / INT حساس به لبه می باشد . به این صورت که اگر بیت 6 از ثبات OPTION-REG به نام بیت INTEDG ، یک باشد ، حساس به لبه بالا رونده است و اگر بیت INTEDG پاک شود ، حساس به لبه پایین رونده می باشد . هنگامی که یک لبه تعریف شده در پایه RB0 / INT ظاهر شود ، بیت INTF از ثبات INTCON ، Set می شود .

---

[1] - Un-Masked

[2] - Interrupt Routine

[3] - Interrupt Service Routine

این وقفه با پاک کردن بیت کنترلی INTE از ثبات INTCON غیر فعال می شود . باید توجه داشت کـه بیـت پـرچم INTF بایدبـه وسـیله روال سرویس وقفه و پیش از آن که این وقفه مجدداً فعال شود ، به صورت نرم افزاری پاک گردد .

توجه :

وقفـه INT مـی توانـد پردازشـگر را از حالت Sleep خارج سازد ، البته تنها در صورتی که بیت INTE پیش از آن که پردازشگر وارد حالت Sleep شود ، Set شده باشد .

یـک سـرریز در تایمـر صـفر (FF H → 00 H) ، بیـت T0IF از ثبات INTCON را Set می کند . این وقف به وسیله یک یا صفر شدن بیت فعال ساز T0IE می تواند فعال یا غیر فعال شود .

وقفه پورت B :

یـک تغییـر ورودی در بیـت هـای 4 تـا 7 پـورت B ، بیـت پـرچم RBIF از ثـبات INTCON را Set مـی کـند . ایـن وقفـه به وسیله یک یا صفر شدن بیت فعال ساز RBIE می تواند فعال یا غیر فعال شود .

نگهداری جایگاه (Context) در طول اجرای یک وقفه :

در طـول اجـرای یک وقفه ، تنها مقدار بازگشتی PC در پشته ذخیره می شود . غالباً کاربران میل دارند که مقادیر ثبات های کلیدی را در طول یک وقفه ذخیره کنند (مثلاً مقادیر ثبات های W و STATUS) . باید توجه داشت که این عمل به صورت نرم افزاری قابل پیاده سازی است .

مثال زیر مقادیر ثبات های W و STATUS را ذخیره و بازیابی می کند . ثبات های تعریف شده به وسیله کاربر W – TEMP و STATUS – TEMP در واقع مکان های ذخیره سازی موقت برای ذخیره مقادیر ثبات های W و STATUS می باشند .

مثال 5)

```
PUSH    MOVWF   W_TEMP          ; Copy W to TEMP register,
        SWAPF   STATUS, W       ; Swap status to be saved into W
        MOVWF   STATUS_TEMP     ; Save status to STATUS_TEMP register
ISR     :                       :
        :                       ; Interrupt Service Routine
        :                       ;  should configure Bank as required
        :                       ;
POP     SWAPF   STATUS_TEMP, W  ; Swap nibbles in STATUS_TEMP register
                                ; and place result into W
        MOVWF   STATUS          ; Move W into STATUS register
                                ;   (sets bank to original state)
        SWAPF   W_TEMP, F       ; Swap nibbles in W_TEMP and place result in W_TEMP
```

مجموعه دستورالعمل ها :

هـر دسـتورالعمل در PIC16CXX ، چهـارده بیتـی بـوده که این چهارده بیت به چند بخش تقسیم مـی شـوند . بـه ایـن صـورت کـه یـک قسـمت از آن Opcode مـی باشـد کـه مشخص می کند که دستورالعمل از چه نوعی است و بخش های دیگر از یک یا چند عملوند که عملکرد دستورالعمل را مشخص می کنند تشکیل شده است .

جدول هشت توصیف فیلد Opcode را نشان می دهد :

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
label	Label name
TOS	Top of Stack
PC	Program Counter
PCLATH	Program Counter High Latch
GIE	Global Interrupt Enable bit
WDT	Watchdog Timer/Counter
$\overline{TO}$	Time-out bit
$\overline{PD}$	Power-down bit
dest	Destination either the W register or the specified register file location
[ ]	Options
( )	Contents
→	Assigned to
< >	Register bit field
∈	In the set of
italics	User defined term (font is courier)

Table – 8 Opcode Field Discription

دستورالعمل های این میکروکنترولر به سه دسته تقسیم می شوند که عبارتند از :

- دستورالعمل های بایت گرا (Byte – Oriented)
- دستور العمل های بیت گرا (Bit – Oriented)
- دستورالعمل های کنترل و لیترال (Literal & Control)

تقسیم می شوند .

در دستور العمل های بایت گرا " f " نمایانگر یک اشاره گر[1] رجسیتر فایل و " d " نمایانگر اشاره گر مقصد است . اشاره گر رجیستر فایل مشخص می کند که کدام رجیستر فایل باید به وسیله دستورالعمل مورد استفاده قرار گیرد . و اشاره گر مقصد بیان می دارد که نتیجه عملکرد ، کجا باید قرار گیرد . اگر " d " صفر باشد نتیجه عملکرد ، در ثبات کاری W گذاشته می شود و در صورتی که " d " یک باشد ، نتیجه در رجیستر فایلی که توسط دستورالعمل مشخص شده است جای می گیرد .

در دستورالعمل های بیت گرا ، " b " نمایانگر یک اشاره گر فیلد بیت است که تعداد بیت هایی که توسط عملیات تحت تأثیر قرار گرفته اند را انتخاب می کند . و " f " نمایانگر تعداد فایل هایی است که به وسیله بیت ها تعیین محل می شود . در دستورالعملهای لیترال و کنترل " k " بیانگر یک ثابت یا مقدار عددی هشت یا یازده بیتی می باشد .

تمام دستورالعمل ها در یک سیکل اجرا می شوند مگر این که با یک گزاره شرطی که ارزشش درست (True) است مواجه شوند و یا در نتیجه یک دستورالعمل خاص (از گروه Branch) مقدار PC تغییر یافته باشد که در این صورت دستورالعمل در دو سیکل انجام می شود .

توجه :

برای حفظ سازگاری بیشتر با محصولات آینده PIC16CXX از دستورالعملهای OPTION و TRIS استفاده نکنید .

[1] - Designator

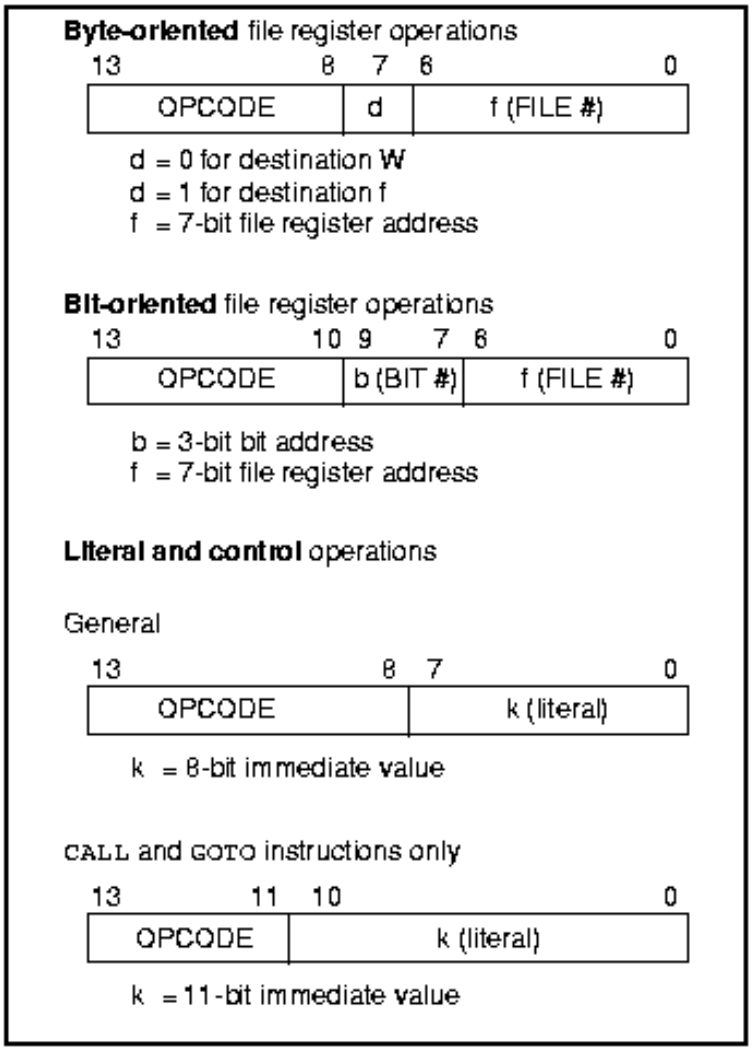


Figure – 10 General Format For Instruction

ضمیمه الف

خلاصه مجموعه دستورالعمل های PIC16F84

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

ضمیمه ب

تعریف دستورالعمل های PIC16F84

ADDLW — **Add Literal and W**

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: (W) + k → (W)

Status Affected: C, DC, Z

Encoding:

11	111x	kkkk	kkkk

Description: The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example:

```
ADDLW   0x15
```

Before Instruction
W = 0x10
After Instruction
W = 0x25

ANDLW — **AND Literal with W**

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (W) .AND. (k) → (W)

Status Affected: Z

Encoding:

11	1001	kkkk	kkkk

Description: The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example

```
ANDLW   0x5F
```

Before Instruction
W = 0xA3
After Instruction
W = 0x03

ADDWF — **Add W and f**

Syntax: [*label*] ADDWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W) + (f) → (destination)

Status Affected: C, DC, Z

Encoding:

00	0111	dfff	ffff

Description: Add the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
ADDWF   FSR, 0
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0xD9
FSR = 0xC2

ANDWF — **AND W with f**

Syntax: [*label*] ANDWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W) .AND. (f) → (destination)

Status Affected: Z

Encoding:

00	0101	dfff	ffff

Description: AND the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
ANDWF   FSR, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0x17
FSR = 0x02

BCF **Bit Clear f**

Syntax: [*label*] BCF f,b

Operands: $0 \le f \le 127$
$0 \le b \le 7$

Operation: $0 \rightarrow (f<b>)$

Status Affected: None

Encoding:

01	00bb	bfff	ffff

Description: Bit 'b' in register 'f' is cleared.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example

```
BCF     FLAG_REG, 7
```

Before Instruction
FLAG_REG = 0xC7
After Instruction
FLAG_REG = 0x47

BSF **Bit Set f**

Syntax: [*label*] BSF f,b

Operands: $0 \le f \le 127$
$0 \le b \le 7$

Operation: $1 \rightarrow (f<b>)$

Status Affected: None

Encoding:

01	01bb	bfff	ffff

Description: Bit 'b' in register 'f' is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example

```
BSF     FLAG_REG,   7
```

Before Instruction
FLAG_REG = 0x0A
After Instruction
FLAG_REG = 0x8A

BTFSC **Bit Test, Skip If Clear**

Syntax: [*label*] BTFSC f,b

Operands: $0 \le f \le 127$
$0 \le b \le 7$

Operation: skip if (f<b>) = 0

Status Affected: None

Encoding:

01	10bb	bfff	ffff

Description: If bit 'b' in register 'f' is '1' then the next instruction is executed.
If bit 'b', in register 'f', is '0' then the next instruction is discarded, and a NOP is executed instead, making this a 2Tcy instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No-Operation

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE    BTFSC   FLAG,1
FALSE   GOTO    PROCESS_CODE
TRUE    •
        •
        •
```

Before Instruction
PC = address HERE
After Instruction
if FLAG<1> = 0,
PC = address TRUE
if FLAG<1>=1,
PC = address FALSE

BTFSS **Bit Test f, Skip if Set**

Syntax: [*label*] BTFSS f,b

Operands: 0 ≤ f ≤ 127
0 ≤ b < 7

Operation: skip if (f<b>) = 1

Status Affected: None

Encoding:

01	11bb	bfff	ffff

Description: If bit 'b' in register 'f' is '0' then the next instruction is executed.
If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2TCY instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No-Operation

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE    BTFSC   FLAG,1
FALSE   GOTO    PROCESS_CODE
TRUE    •
        •
        •
```

Before Instruction
PC = address HERE
After Instruction
if FLAG<1> = 0,
PC = address FALSE
if FLAG<1> = 1,
PC = address TRUE

CALL **Call Subroutine**

Syntax: [*label*] CALL k

Operands: 0 ≤ k ≤ 2047

Operation: (PC)+ 1→ TOS,
k → PC<10:0>,
(PCLATH<4:3>) → PC<12:11>

Status Affected: None

Encoding:

10	0kkk	kkkk	kkkk

Description: Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k', Push PC to Stack	Process data	Write to PC
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE    CALL    THERE
```

Before Instruction
PC = Address HERE
After Instruction
PC = Address THERE
TOS = Address HERE+1

CLRF **Clear f**

Syntax: [*label*] CLRF f

Operands: 0 ≤ f ≤ 127

Operation: 00h → (f)
1 → Z

Status Affected: Z

Encoding:

00	0001	lfff	ffff

Description: The contents of register 'f' are cleared and the Z bit is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example

```
CLRF    FLAG_REG
```

Before Instruction
FLAG_REG = 0x5A
After Instruction
FLAG_REG = 0x00
Z = 1

CLRW **Clear W**

Syntax: [*label*] CLRW

Operands: None

Operation: 00h → (W)
1 → Z

Status Affected: Z

Encoding:

00	0001	0xxx	xxxx

Description: W register is cleared. Zero bit (Z) is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	Process data	Write to W

Example

```
CLRW
```

Before Instruction
W = 0x5A
After Instruction
W = 0x00
Z = 1

CLRWDT **Clear Watchdog Timer**

Syntax: [*label*] CLRWDT

Operands: None

Operation: 00h → WDT
0 → WDT prescaler,
1 → $\overline{TO}$
1 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0100

Description: CLRWDT instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	Process data	Clear WDT Counter

Example

```
CLRWDT
```

Before Instruction
WDT counter = ?
After Instruction
WDT counter = 0x00
WDT prescaler = 0
$\overline{TO}$ = 1
$\overline{PD}$ = 1

COMF **Complement f**

Syntax: [*label*] COMF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: $(\bar{f}) \rightarrow$ (destination)

Status Affected: Z

Encoding:

00	1001	dfff	ffff

Description: The contents of register 'f' are complemented. If 'd' is 0 the result is stored in W. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
COMF    REG1,0
```

Before Instruction
REG1 = 0x13
After Instruction
REG1 = 0x13
W = 0xEC

DECF **Decrement f**

Syntax: [*label*] DECF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - 1 → (destination)

Status Affected: Z

Encoding:

00	0011	dfff	ffff

Description: Decrement register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
DECF    CNT, 1
```

Before Instruction
CNT = 0x01
Z = 0
After Instruction
CNT = 0x00
Z = 1

DECFSZ **Decrement f, Skip if 0**

Syntax: [*label*] DECFSZ f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - 1 → (destination);
skip if result = 0

Status Affected: None

Encoding:

00	1011	dfff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
If the result is 1, the next instruction, is executed. If the result is 0, then a NOP is executed instead making it a 2TCY instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE    DECFSZ  CNT, 1
        GOTO    LOOP
CONTINUE •
         •
         •
```

Before Instruction
PC = address HERE
After Instruction
CNT = CNT - 1
if CNT = 0,
PC = address CONTINUE
if CNT ≠ 0,
PC = address HERE+1

GOTO **Unconditional Branch**

Syntax: [*label*] GOTO k

Operands: 0 ≤ k ≤ 2047

Operation: k → PC<10:0>
PCLATH<4:3> → PC<12:11>

Status Affected: None

Encoding:

10	1kkk	kkkk	kkkk

Description: GOTO is an unconditional branch. The eleven bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k'	Process data	Write to PC
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
GOTO THERE
```

After Instruction
PC = Address THERE

INCF **Increment f**

Syntax: [*label*] INCF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) + 1 → (destination)

Status Affected: Z

Encoding:

00	1010	dfff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example

```
INCF CNT, 1
```

Before Instruction
CNT = 0xFF
Z = 0
After Instruction
CNT = 0x00
Z = 1

INCFSZ **Increment f, Skip If 0**

Syntax: [*label*] INCFSZ f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) + 1 → (destination), skip if result = 0

Status Affected: None

Encoding:

00	1111	dfff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
If the result is 1, the next instruction is executed. If the result is 0, a NOP is executed instead making it a 2TCY instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

If Skip: (2nd Cycle)

Q1	Q2	Q3	Q4
No-Operation	No-Operation	No-Operation	No-Operation

Example

```
HERE    INCFSZ  CNT, 1
        GOTO    LOOP
CONTINUE •
         •
         •
```

Before Instruction
PC = address HERE
After Instruction
CNT = CNT + 1
if CNT= 0,
PC = address CONTINUE
if CNT≠ 0,
PC = address HERE +1

IORLW **Inclusive OR Literal with W**

Syntax: [*label*] IORLW k

Operands: 0 ≤ k ≤ 255

Operation: (W) .OR. k → (W)

Status Affected: Z

Encoding:

11	1000	kkkk	kkkk

Description: The contents of the W register is OR'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example

```
IORLW 0x35
```

Before Instruction
W = 0x9A
After Instruction
W = 0xBF
Z = 1

IORWF **Inclusive OR W with f**

Syntax: [*label*] IORWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W) .OR. (f) → (destination)

Status Affected: Z

Encoding:

00	0100	dfff	ffff

Description: Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example
```
IORWF   RESULT, 0
```
Before Instruction
RESULT = 0x13
W = 0x91
After Instruction
RESULT = 0x13
W = 0x93
Z = 1

MOVF **Move f**

Syntax: [*label*] MOVF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) → (destination)

Status Affected: Z

Encoding:

00	1000	dfff	ffff

Description: The contents of register f is moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register since status flag Z is affected.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example
```
MOVF    FSR, 0
```
After Instruction
W = value in FSR register
Z = 1

MOVLW **Move Literal to W**

Syntax: [*label*] MOVLW k

Operands: $0 \le k \le 255$

Operation: k → (W)

Status Affected: None

Encoding:

11	00xx	kkkk	kkkk

Description: The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example
```
MOVLW   0x5A
```
After Instruction
W = 0x5A

MOVWF **Move W to f**

Syntax: [*label*] MOVWF f

Operands: $0 \le f \le 127$

Operation: (W) → (f)

Status Affected: None

Encoding:

00	0000	1fff	ffff

Description: Move data from W register to register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example
```
MOVWF   OPTION_REG
```
Before Instruction
OPTION = 0xFF
W = 0x4F
After Instruction
OPTION = 0x4F
W = 0x4F

NOP — **No Operation**

Syntax: [*label*] NOP

Operands: None

Operation: No operation

Status Affected: None

Encoding:

00	0000	0xx0	0000

Description: No operation.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No-Operation	No-Operation	No-Operation

Example

```
NOP
```

OPTION — **Load Option Register**

Syntax: [*label*] OPTION

Operands: None

Operation: (W) → OPTION

Status Affected: None

Encoding:

00	0000	0110	0010

Description: The contents of the W register are loaded in the OPTION register. This instruction is supported for code compatibility with PIC16C5X products. Since OPTION is a readable/writable register, the user can directly address it.

Words: 1

Cycles: 1

Example

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

RETFIE — **Return from Interrupt**

Syntax: [*label*] RETFIE

Operands: None

Operation: TOS → PC,
1 → GIE

Status Affected: None

Encoding:

00	0000	0000	1001

Description: Return from Interrupt. Stack is POPed and Top of Stack (TOS) is loaded in the PC. Interrupts are enabled by setting Global Interrupt Enable bit, GIE (INTCON<7>). This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	No-Operation	Set the GIE bit	Pop from the Stack
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
RETFIE
```

After Interrupt
PC = TOS
GIE = 1

RETLW — **Return with Literal in W**

Syntax: [*label*] RETLW k

Operands: 0 ≤ k ≤ 255

Operation: k → (W);
TOS → PC

Status Affected: None

Encoding:

11	01xx	kkkk	kkkk

Description: The W register is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	Read literal 'k'	No-Operation	Write to W, Pop from the Stack
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
CALL TABLE  ;W contains table
            ;offset value
            ;W now has table value
            .
            .
            .
TABLE       ADDWF PC    ;W = offset
            RETLW k1    ;Begin table
            RETLW k2    ;
            .
            .
            .
            RETLW kn    ; End of table
```

Before Instruction
W = 0x07
After Instruction
W = value of k8

RETURN — **Return from Subroutine**

Syntax: [*label*] RETURN

Operands: None

Operation: TOS → PC

Status Affected: None

Encoding:

00	0000	0000	1000

Description: Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

	Q1	Q2	Q3	Q4
1st Cycle	Decode	No-Operation	No-Operation	Pop from the Stack
2nd Cycle	No-Operation	No-Operation	No-Operation	No-Operation

Example

```
RETURN
```

After Interrupt
PC = TOS

SUBWF **Subtract W from f**

Syntax: [*label*] SUBWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f) - (W) → (destination)

Status Affected: C, DC, Z

Encoding:

00	0010	dfff	ffff

Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1:
```
SUBWF    REG1,1
```
Before Instruction
```
REG1 = 3
W    = 2
C    = ?
Z    = ?
```
After Instruction
```
REG1 = 1
W    = 2
C    = 1; result is positive
Z    = 0
```
Example 2: Before Instruction
```
REG1 = 2
W    = 2
C    = ?
Z    = ?
```
After Instruction
```
REG1 = 0
W    = 2
C    = 1; result is zero
Z    = 1
```

SWAPF **Swap Nibbles in f**

Syntax: [*label*] SWAPF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (f<3:0>) → (destination<7:4>),
(f<7:4>) → (destination<3:0>)

Status Affected: None

Encoding:

00	1110	dfff	ffff

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0 the result is placed in W register. If 'd' is 1 the result is placed in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example
```
SWAPF REG,  0
```
Before Instruction
```
REG1 = 0xA5
```
After Instruction
```
REG1 = 0xA5
W    = 0x5A
```

XORLW **Exclusive OR Literal with W**

Syntax: [*label*] XORLW k

Operands: 0 ≤ k ≤ 255

Operation: (W) .XOR. k → (W)

Status Affected: Z

Encoding:

11	1010	kkkk	kkkk

Description: The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W

Example:
```
XORLW    0xAF
```
Before Instruction
```
W = 0xB5
```
After Instruction
```
W = 0x1A
```

XORWF **Exclusive OR W with f**

Syntax: [*label*] XORWF f,d

Operands: 0 ≤ f ≤ 127
d ∈ [0,1]

Operation: (W) .XOR. (f) → (destination)

Status Affected: Z

Encoding:

00	0110	dfff	ffff

Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example
```
XORWF    REG  1
```
Before Instruction
```
REG = 0xAF
W   = 0xB5
```
After Instruction
```
REG = 0x1A
W   = 0xB5
```

TRIS **Load TRIS Register**

Syntax: [*label*] TRIS f

Operands: 5 ≤ f ≤ 7

Operation: (W) → TRIS register f;

Status Affected: None

Encoding:

00	0000	0110	0fff

Description: The instruction is supported for code compatibility with the PIC16C5X products. Since TRIS registers are readable and writable, the user can directly address them.

Words: 1

Cycles: 1

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.