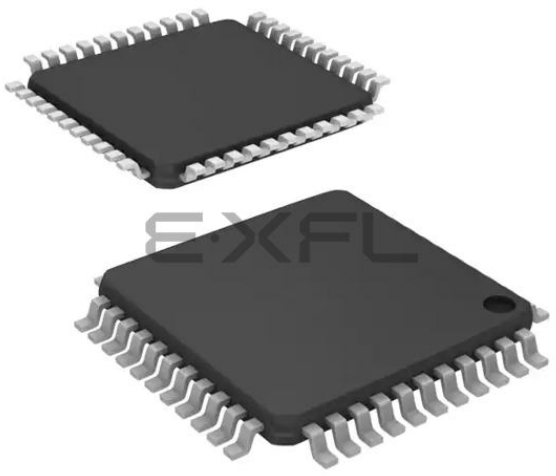


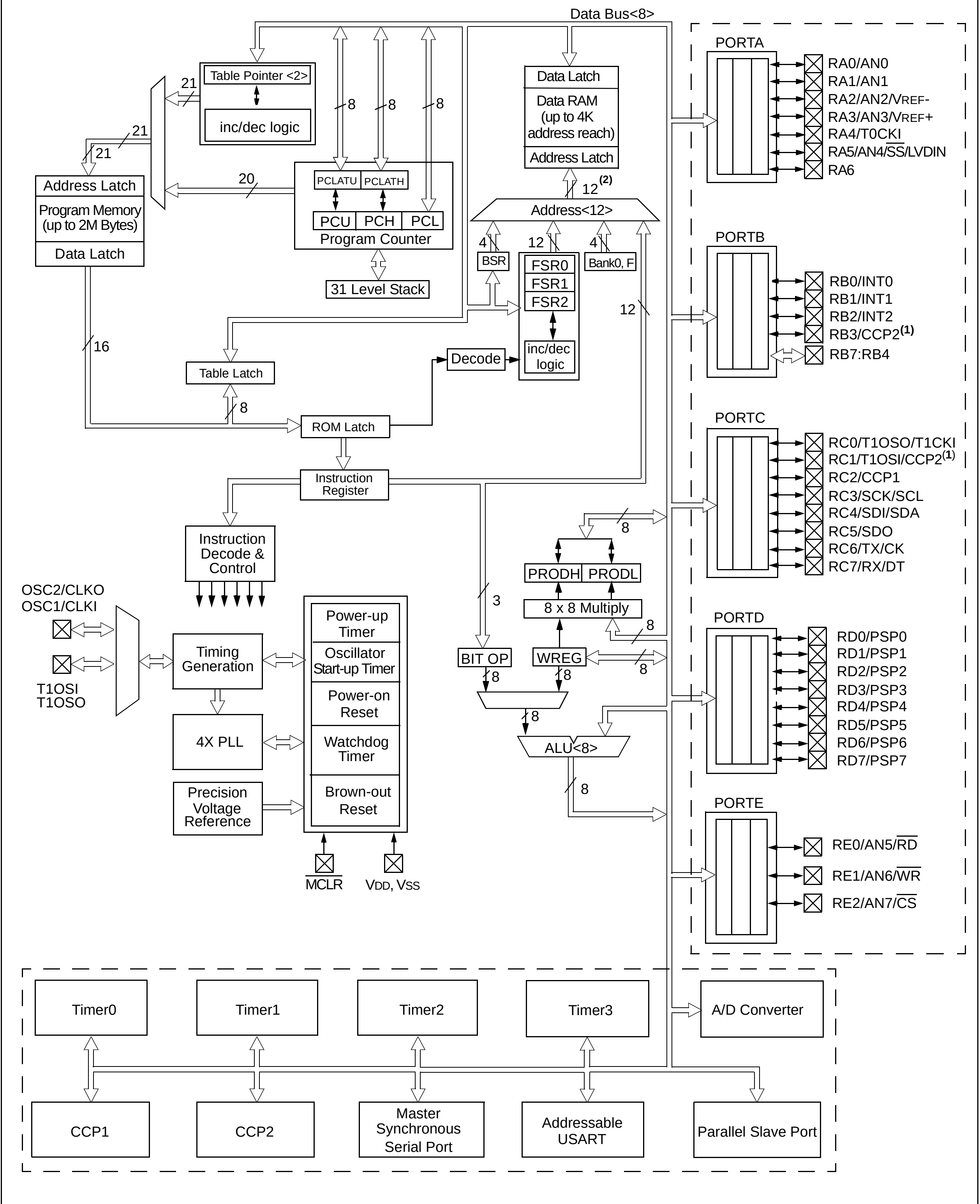
What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	40MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	33
Program Memory Size	16KB (8K x 16)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	512 x 8
Voltage - Supply (Vcc/Vdd)	4.2V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	44-TQFP
Supplier Device Package	44-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18c442t-e-pt

FIGURE 1-2: PIC18C4X2 BLOCK DIAGRAM



Note 1: Optional multiplexing of CCP2 input/output with RB3 is enabled by selection of configuration bit.

2: The high order bits of the Direct Address for the RAM are from the BSR register (except for the `MOVFF` instruction).

3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

TABLE 4-2: REGISTER FILE SUMMARY

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
TOSU	—	—	—	Top-of-Stack Upper Byte (TOS<20:16>)					`---0 0000`	37
TOSH	Top-of-Stack High Byte (TOS<15:8>)								`0000 0000`	37
TOSL	Top-of-Stack Low Byte (TOS<7:0>)								`0000 0000`	37
STKPTR	STKFUL	STKUNF	—	Return Stack Pointer					`00-0 0000`	38
PCLATU	—	—	—	Holding Register for PC<20:16>					`---0 0000`	39
PCLATH	Holding Register for PC<15:8>								`0000 0000`	39
PCL	PC Low Byte (PC<7:0>)								`0000 0000`	39
TBLPTRU	—	—	bit21[(2)]	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					`---0 0000`	57
TBLPTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								`0000 0000`	57
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								`0000 0000`	57
TABLAT	Program Memory Table Latch								`0000 0000`	57
PRODH	Product Register High Byte								`xxxx xxxx`	61
PRODL	Product Register Low Byte								`xxxx xxxx`	61
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	`0000 000x`	65
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	`1111 -1-1`	66
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	`11-0 0-00`	67
INDF0	Uses contents of FSR0 to address data memory - value of FSR0 not changed (not a physical register)								N/A	50
POSTINC0	Uses contents of FSR0 to address data memory - value of FSR0 post-incremented (not a physical register)								N/A	50
POSTDEC0	Uses contents of FSR0 to address data memory - value of FSR0 post-decremented (not a physical register)								N/A	50
PREINC0	Uses contents of FSR0 to address data memory - value of FSR0 pre-incremented (not a physical register)								N/A	50
PLUSW0	Uses contents of FSR0 to address data memory - value of FSR0 pre-incremented (not a physical register) - value of FSR0 offset by value in WREG								N/A	50
FSR0H	—	—	—	—	Indirect Data Memory Address Pointer 0 High Byte				`---- 0000`	50
FSR0L	Indirect Data Memory Address Pointer 0 Low Byte								`xxxx xxxx`	50
WREG	Working Register								`xxxx xxxx`	
INDF1	Uses contents of FSR1 to address data memory - value of FSR1 not changed (not a physical register)								N/A	50
POSTINC1	Uses contents of FSR1 to address data memory - value of FSR1 post-incremented (not a physical register)								N/A	50
POSTDEC1	Uses contents of FSR1 to address data memory - value of FSR1 post-decremented (not a physical register)								N/A	50
PREINC1	Uses contents of FSR1 to address data memory - value of FSR1 pre-incremented (not a physical register)								N/A	50
PLUSW1	Uses contents of FSR1 to address data memory - value of FSR1 pre-incremented (not a physical register) - value of FSR1 offset by value in WREG								N/A	50
FSR1H	—	—	—	—	Indirect Data Memory Address Pointer 1 High Byte				`---- 0000`	50
FSR1L	Indirect Data Memory Address Pointer 1 Low Byte								`xxxx xxxx`	50
BSR	—	—	—	—	Bank Select Register				`---- 0000`	49
INDF2	Uses contents of FSR2 to address data memory - value of FSR2 not changed (not a physical register)								N/A	50
POSTINC2	Uses contents of FSR2 to address data memory - value of FSR2 post-incremented (not a physical register)								N/A	50
POSTDEC2	Uses contents of FSR2 to address data memory - value of FSR2 post-decremented (not a physical register)								N/A	50
PREINC2	Uses contents of FSR2 to address data memory - value of FSR2 pre-incremented (not a physical register)								N/A	50
PLUSW2	Uses contents of FSR2 to address data memory - value of FSR2 pre-incremented (not a physical register) - value of FSR2 offset by value in WREG								N/A	50
FSR2H	—	—	—	—	Indirect Data Memory Address Pointer 2 High Byte				`---- 0000`	50
FSR2L	Indirect Data Memory Address Pointer 2 Low Byte								`xxxx xxxx`	50
STATUS	—	—	—	N	OV	Z	DC	C	`---x xxxx`	52
TMR0H	Timer0 Register High Byte								`0000 0000`	95
TMR0L	Timer0 Register Low Byte								`xxxx xxxx`	95
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	`1111 1111`	93
OSCCON	—	—	—	—	—	—	—	SCS	`---- ---0`	20
LVDCON	—	—	IRVST	LVDEN	LVDL3	LVDL2	LVDL1	LVDL0	`--00 0101`	175

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, `q` = value depends on condition
Note 1: RA6 and associated bits are configured as port pins in RCIO and ECIO oscillator mode only, and read '0' in all other oscillator modes.
2: Bit 21 of the TBLPTRU allows access to the device configuration bits.

4.10 Access Bank

The Access Bank is an architectural enhancement, which is very useful for C compiler code optimization. The techniques used by the C compiler may also be useful for programs written in assembly.

This data memory region can be used for:

- Intermediate computational values
- Local variables of subroutines
- Faster context saving/switching of variables
- Common variables
- Faster evaluation/control of SFRs (no banking)

The Access Bank is comprised of the upper 128 bytes in Bank 15 (SFRs) and the lower 128 bytes in Bank 0. These two sections will be referred to as Access RAM High and Access RAM Low, respectively. Figure 4-6 and Figure 4-7 indicate the Access RAM areas.

A bit in the instruction word specifies if the operation is to occur in the bank specified by the BSR register or in the Access Bank. This bit is denoted by the ’a’ bit (for access bit).

When forced in the Access Bank (a = ’0’), the last address in Access RAM Low is followed by the first address in Access RAM High. Access RAM High maps the Special Function registers, so that these registers can be accessed without any software overhead. This is useful for testing status flags and modifying control bits.

4.11 Bank Select Register (BSR)

The need for a large general purpose memory space dictates a RAM banking scheme. The data memory is partitioned into sixteen banks. When using direct addressing, the BSR should be configured for the desired bank.

BSR<3:0> holds the upper 4 bits of the 12-bit RAM address. The BSR<7:4> bits will always read ’0’s, and writes will have no effect.

A `MOVLB` instruction has been provided in the instruction set to assist in selecting banks.

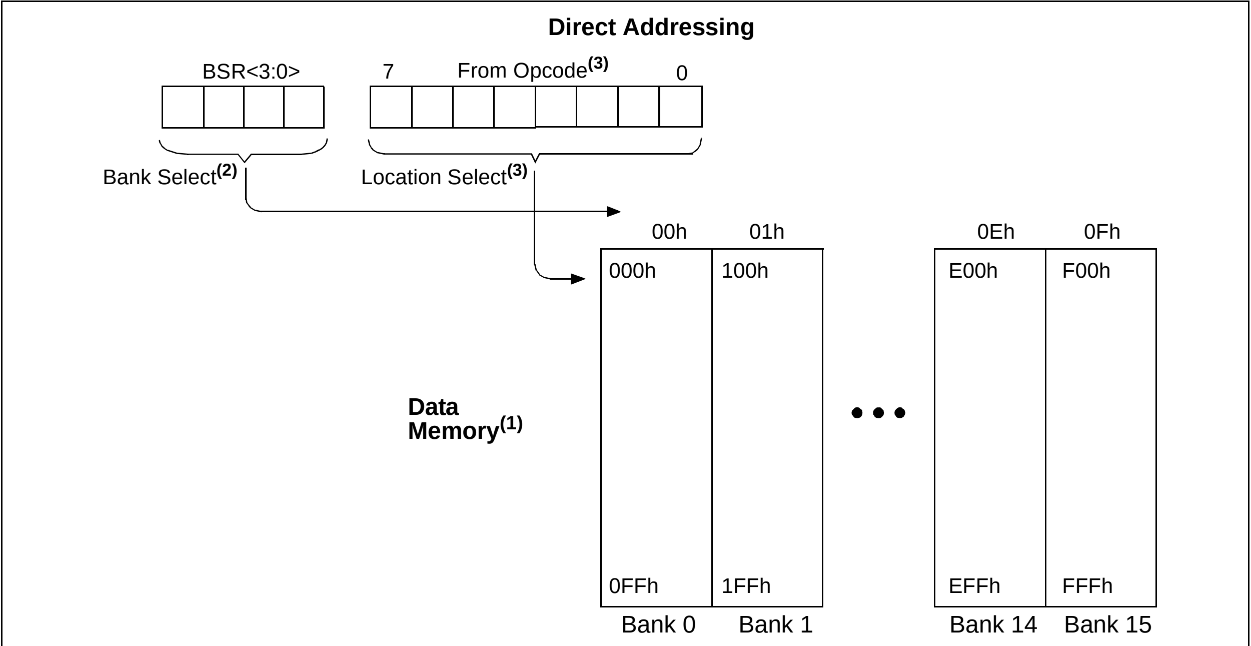
If the currently selected bank is not implemented, any read will return all '0's and all writes are ignored. The STATUS register bits will be set/cleared as appropriate for the instruction performed.

Each Bank extends up to FFh (256 bytes). All data memory is implemented as static RAM.

A `MOVFF` instruction ignores the BSR, since the 12-bit addresses are embedded into the instruction word.

Section 4.12 provides a description of indirect addressing, which allows linear addressing of the entire RAM space.

FIGURE 4-8: DIRECT ADDRESSING



Note 1: For register file map detail, see Table 4-1.

2: The access bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

3: The `MOVFF` instruction embeds the entire 12-bit address in the instruction.

4.12 Indirect Addressing, INDF and FSR Registers

Indirect addressing is a mode of addressing data memory, where the data memory address in the instruction is not fixed. An FSR register is used as a pointer to the data memory location that is to be read or written. Since this pointer is in RAM, the contents can be modified by the program. This can be useful for data tables in the data memory and for software stacks. Figure 4-9 shows the operation of indirect addressing. This shows the moving of the value to the data memory address, specified by the value of the FSR register.

Indirect addressing is possible by using one of the INDF registers. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = '0'), will read 00h. Writing to the INDF register indirectly, results in a no operation. The FSR register contains a 12-bit address, which is shown in Figure 4-10.

The INDFn register is not a physical register. Addressing INDFn actually addresses the register whose address is contained in the FSRn register (FSRn is a pointer). This is indirect addressing.

Example 4-4 shows a simple use of indirect addressing to clear the RAM in Bank1 (locations 100h-1FFh) in a minimum number of instructions.

EXAMPLE 4-4: HOW TO CLEAR RAM (BANK1) USING INDIRECT ADDRESSING

```
      LFSR  FSR0, 0x100  ;
NEXT  CLRF  POSTINC0     ; Clear INDF register
                         ; & inc pointer
      BTFSS FSR0H, 1     ; All done w/ Bank1?
      GOTO  NEXT         ; NO, clear next
CONTINUE                 ; YES, continue
```

There are three indirect addressing registers. To address the entire data memory space (4096 bytes), these registers are 12-bit wide. To store the 12-bits of addressing information, two 8-bit registers are required. These indirect addressing registers are:

1. FSR0: composed of FSR0H:FSR0L
2. FSR1: composed of FSR1H:FSR1L
3. FSR2: composed of FSR2H:FSR2L

In addition, there are registers INDF0, INDF1 and INDF2, which are not physically implemented. Reading or writing to these registers activates indirect addressing, with the value in the corresponding FSR register being the address of the data.

If an instruction writes a value to INDF0, the value will be written to the address pointed to by FSR0H:FSR0L. A read from INDF1 reads the data from the address pointed to by FSR1H:FSR1L. INDFn can be used in code anywhere an operand can be used.

If INDF0, INDF1 or INDF2 are read indirectly via an FSR, all '0's are read (zero bit is set). Similarly, if INDF0, INDF1 or INDF2 are written to indirectly, the operation will be equivalent to a `NOP` instruction and the STATUS bits are not affected.

4.12.1 INDIRECT ADDRESSING OPERATION

Each FSR register has an INDF register associated with it, plus four additional register addresses. Performing an operation on one of these five registers determines how the FSR will be modified during indirect addressing.

When data access is done to one of the five INDFn locations, the address selected will configure the FSRn register to:

- Do nothing to FSRn after an indirect access (no change) - INDFn
- Auto-decrement FSRn after an indirect access (post-decrement) - POSTDECn
- Auto-increment FSRn after an indirect access (post-increment) - POSTINCn
- Auto-increment FSRn before an indirect access (pre-increment) - PREINCn
- Use the value in the WREG register as an offset to FSRn. Do not modify the value of the WREG or the FSRn register after an indirect access (no change) - PLUSWn

When using the auto-increment or auto-decrement features, the effect on the FSR is not reflected in the STATUS register. For example, if the indirect address causes the FSR to equal '0', the Z bit will not be set.

Incrementing or decrementing an FSR affects all 12 bits. That is, when FSRnL overflows from an increment, FSRnH will be incremented automatically.

Adding these features allows the FSRn to be used as a stack pointer, in addition to its uses for table operations in data memory.

Each FSR has an address associated with it that performs an indexed indirect access. When a data access to this INDFn location (PLUSWn) occurs, the FSRn is configured to add the signed value in the WREG register and the value in FSR to form the address before an indirect access. The FSR value is not changed.

If an FSR register contains a value that points to one of the INDFn, an indirect read will read 00h (zero bit is set), while an indirect write will be equivalent to a `NOP` (STATUS bits are not affected).

4.13 STATUS Register

The STATUS register, shown in Register 4-2, contains the arithmetic status of the ALU. The STATUS register can be the destination for any instruction, as with any other register. If the STATUS register is the destination for an instruction that affects the Z, DC, C, OV or N bits, then the write to these five bits is disabled. These bits are set or cleared according to the device logic. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper three bits and set the Z bit. This leaves the STATUS register as `000u u1uu` (where `u` = unchanged).

It is recommended, therefore, that only `BCF`, `BSF`, `SWAPF`, `MOVFF` and `MOVWF` instructions are used to alter the STATUS register, because these instructions do not affect the Z, C, DC, OV or N bits from the STATUS register. For other instructions not affecting any status bits, see Table 19-2.

Note:	The C and DC bits operate as a $\overline{\text{borrow}}$ and digit $\overline{\text{borrow}}$ bit respectively, in subtraction.

REGISTER 4-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC	C
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **N:** Negative bit
This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative, (ALU MSB = 1).
`1` = Result was negative
`0` = Result was positive

bit 3 **OV:** Overflow bit
This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude, which causes the sign bit (bit7) to change state.
`1` = Overflow occurred for signed arithmetic (in this arithmetic operation)
`0` = No overflow occurred

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/$\overline{\text{borrow}}$ bit
For `ADDWF`, `ADDLW`, `SUBLW`, and `SUBWF` instructions
`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

Note: For $\overline{\text{borrow}}$, 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 bit 4 or bit 3 of the source register.

bit 0 **C:** Carry/$\overline{\text{borrow}}$ bit
For `ADDWF`, `ADDLW`, `SUBLW`, and `SUBWF` 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 $\overline{\text{borrow}}$, 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.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

7.5 RCON Register

The RCON register contains the bit which is used to enable prioritized interrupts (IPEN).

REGISTER 7-10: RCON REGISTER

R/W-0	R/W-0	U-0	R/W-1	R-1	R-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
`1` = Enable priority levels on interrupts
`0` = Disable priority levels on interrupts (16CXXX compatibility mode)

bit 6 **LWRT:** Long Write Enable bit
For details of bit operation, see Register 4-3

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{\text{RI}}$:** `RESET` Instruction Flag bit
For details of bit operation, see Register 4-3

bit 3 **$\overline{\text{TO}}$:** Watchdog Time-out Flag bit
For details of bit operation, see Register 4-3

bit 2 **$\overline{\text{PD}}$:** Power-down Detection Flag bit
For details of bit operation, see Register 4-3

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit
For details of bit operation, see Register 4-3

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
For details of bit operation, see Register 4-3

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR reset	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

TABLE 8-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0	bit0	TTL	Input/output or analog input.
RA1/AN1	bit1	TTL	Input/output or analog input.
RA2/AN2/VREF-	bit2	TTL	Input/output or analog input or VREF-.
RA3/AN3/VREF+	bit3	TTL	Input/output or analog input or VREF+.
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0. Output is open drain type.
RA5/$\overline{SS}$/AN4/LVDIN	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input, or low voltage detect input.
OSC2/CLKO/RA6	bit6	TTL	OSC2 or clock output or I/O pin.

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

TABLE 8-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS
PORTA	—	RA6	RA5	RA4	RA3	RA2	RA1	RA0	`--0x 0000`	`--0u 0000`
LATA	—	Latch A Data Output Register							`--xx xxxx`	`--uu uuuu`
TRISA	—	PORTA Data Direction Register							`--11 1111`	`--11 1111`
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	`--0- 0000`	`--0- 0000`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by PORTA.

TABLE 8-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit0	ST/TTL[(1)]	Input/output port pin or parallel slave port bit0.
RD1/PSP1	bit1	ST/TTL[(1)]	Input/output port pin or parallel slave port bit1.
RD2/PSP2	bit2	ST/TTL[(1)]	Input/output port pin or parallel slave port bit2.
RD3/PSP3	bit3	ST/TTL[(1)]	Input/output port pin or parallel slave port bit3.
RD4/PSP4	bit4	ST/TTL[(1)]	Input/output port pin or parallel slave port bit4.
RD5/PSP5	bit5	ST/TTL[(1)]	Input/output port pin or parallel slave port bit5.
RD6/PSP6	bit6	ST/TTL[(1)]	Input/output port pin or parallel slave port bit6.
RD7/PSP7	bit7	ST/TTL[(1)]	Input/output port pin or parallel slave port bit7.

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

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 8-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	`xxxx xxxx`	`uuuu uuuu`
LATD	LATD Data Output Register								`xxxx xxxx`	`uuuu uuuu`
TRISD	PORTD Data Direction Register								`1111 1111`	`1111 1111`
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			`0000 -111`	`0000 -111`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

9.0 TIMER0 MODULE

The Timer0 module has the following features:

- Software selectable as an 8-bit or 16-bit timer/ counter
- Readable and writable
- Dedicated 8-bit software programmable prescaler
- Clock source selectable to be external or internal
- Interrupt-on-overflow from FFh to 00h in 8-bit mode and FFFFh to 0000h in 16-bit mode
- Edge select for external clock

Figure 9-1 shows a simplified block diagram of the Timer0 module in 8-bit mode and Figure 9-2 shows a simplified block diagram of the Timer0 module in 16-bit mode.

The T0CON register (Register 9-1) is a readable and writable register that controls all the aspects of Timer0, including the prescale selection.

REGISTER 9-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
`1` = Enables Timer0
`0` = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
`1` = Timer0 is configured as an 8-bit timer/counter
`0` = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
`1` = Transition on T0CKI pin
`0` = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: Timer0 Source Edge Select bit
`1` = Increment on high-to-low transition on T0CKI pin
`0` = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
`1` = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
`0` = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2:0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
`111` = 1:256 prescale value
`110` = 1:128 prescale value
`101` = 1:64 prescale value
`100` = 1:32 prescale value
`011` = 1:16 prescale value
`010` = 1:8 prescale value
`001` = 1:4 prescale value
`000` = 1:2 prescale value

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR reset	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

10.1 Timer1 Operation

Timer1 can operate in one of these modes:

- As a timer
- As a synchronous counter
- As an asynchronous counter

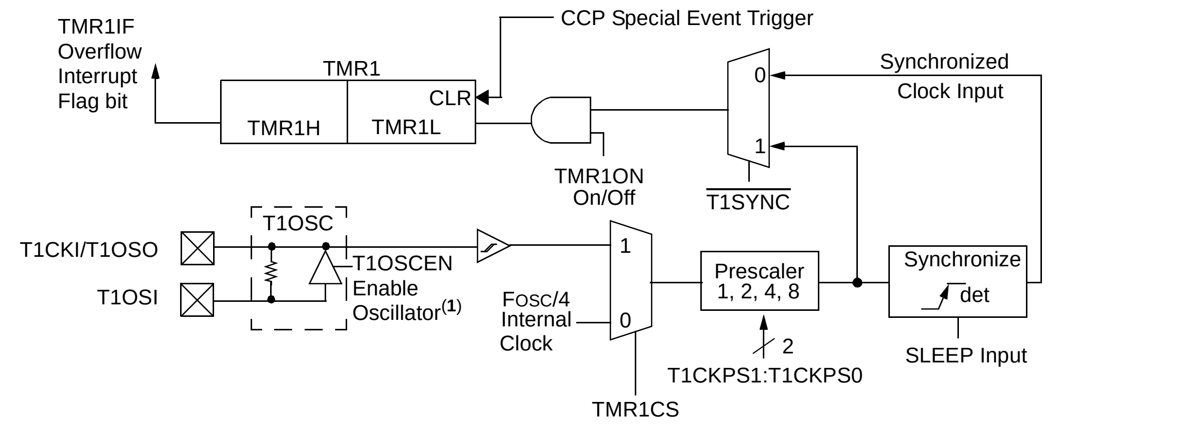
The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

When TMR1CS = 0, Timer1 increments every instruction cycle. When TMR1CS = 1, Timer1 increments on every rising edge of the external clock input or the Timer1 oscillator, if enabled.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored.

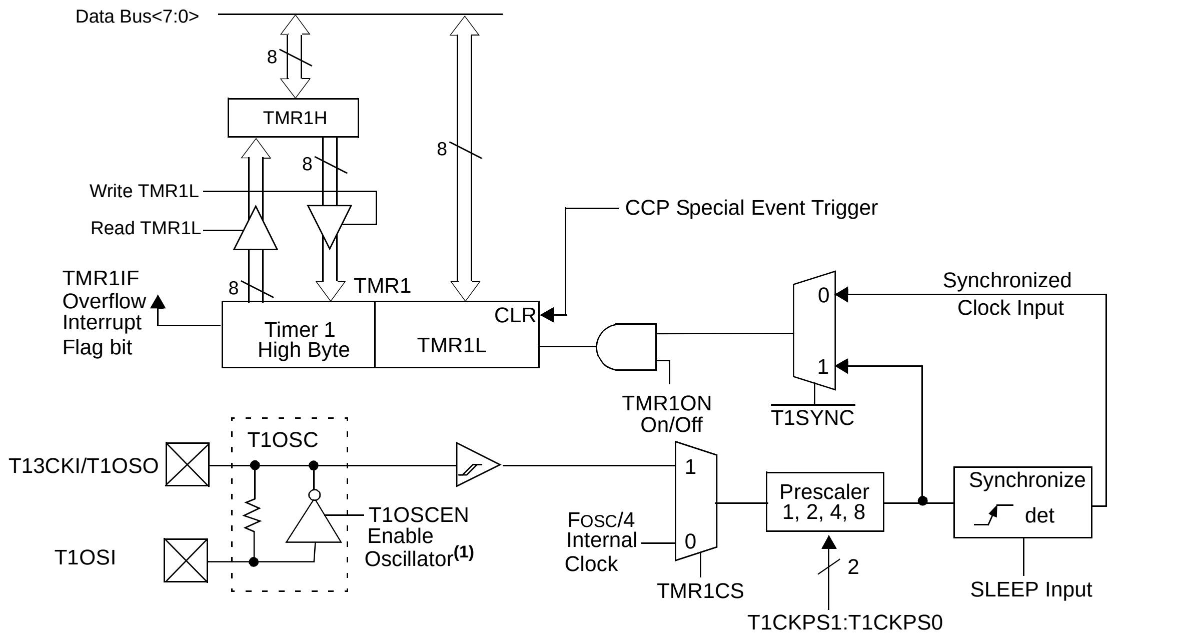
Timer1 also has an internal "RESET input". This RESET can be generated by the CCP module (Section 13.0).

FIGURE 10-1: TIMER1 BLOCK DIAGRAM



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

FIGURE 10-2: TIMER1 BLOCK DIAGRAM: 16-BIT READ/WRITE MODE



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

NOTES:

TABLE 15-5: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 40 MHz			Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
9.6	9.77	-1.70	255	9.615	+0.16	129	9.615	+0.16	103	9.615	+0.16	64
19.2	19.23	-0.16	129	19.230	+0.16	64	19.230	+0.16	51	18.939	-1.36	32
38.4	38.46	-0.16	64	37.878	-1.36	32	38.461	+0.16	25	39.062	+1.7	15
57.6	58.14	-0.93	42	56.818	-1.36	21	58.823	+2.12	16	56.818	-1.36	10
115.2	113.64	+1.38	21	113.63	-1.36	10	111.11	-3.55	8	125	+8.51	4
250	250.00	0	9	250	0	4	250	0	3	NA	—	—
625	625.00	0	3	625	0	1	NA	—	—	625	0	0
1250	1250.00	0	1	1250	0	0	NA	—	—	NA	—	—

BAUD RATE (K)	Fosc = 7.16MHz			Fosc = 5.068 MHz			Fosc = 4 MHz			Fosc = 3.579545 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
9.6	9.520	-0.83	46	9.6	0	32	NA	—	—	9.727	+1.32	22
19.2	19.454	+1.32	22	18.645	-2.94	16	1.202	+0.17	207	18.643	-2.90	11
38.4	37.286	-2.90	11	39.6	+3.12	7	2.403	+0.13	103	37.286	-2.90	5
57.6	55.930	-2.90	7	52.8	-8.33	5	9.615	+0.16	25	55.930	-2.90	3
115.2	111.860	-2.90	3	105.6	-8.33	2	19.231	+0.16	12	111.86	-2.90	1
250	NA	—	—	NA	—	—	NA	—	—	223.72	-10.51	0
625	NA	—	—	NA	—	—	NA	—	—	NA	—	—
1250	NA	—	—	NA	—	—	NA	—	—	NA	—	—

BAUD RATE (K)	Fosc = 1 MHz			Fosc = 32.768 kHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
9.6	8.928	-6.99	6	NA	—	—
19.2	20.833	+8.51	2	NA	—	—
38.4	31.25	-18.61	1	NA	—	—
57.6	62.5	+8.51	0	NA	—	—
115.2	NA	—	—	NA	—	—
250	NA	—	—	NA	—	—
625	NA	—	—	NA	—	—
1250	NA	—	—	NA	—	—

The value that is in the ADRESH/ADRESL registers is not modified for a Power-on Reset. The ADRESH/ADRESL registers will contain unknown data after a Power-on Reset.

After the A/D module has been configured as desired, the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as an input. To determine acquisition time, see Section 16.1. After this acquisition time has elapsed, the A/D conversion can be started. The following steps should be followed for doing an A/D conversion:

1. Configure the A/D module:
   - Configure analog pins, voltage reference and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D conversion clock (ADCON0)
   - Turn on A/D module (ADCON0)
2. Configure A/D interrupt (if desired):
   - Clear ADIF bit
   - Set ADIE bit
   - Set GIE bit
3. Wait the required acquisition time.
4. Start conversion:
   - Set GO/$\overline{\text{DONE}}$ bit (ADCON0)
5. Wait for A/D conversion to complete, by either:
   - Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared

   OR

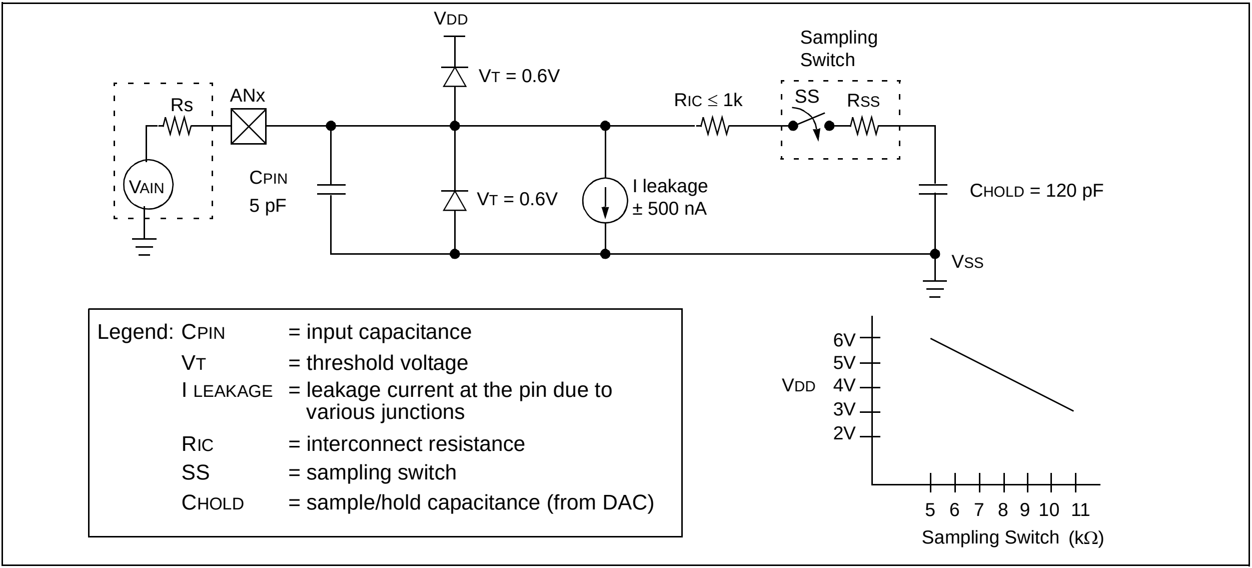
   - Waiting for the A/D interrupt
6. Read A/D Result registers (ADRESH/ADRESL); clear bit ADIF if required.
7. For next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as T_{AD}. A minimum wait of $2T_{AD}$ is required before next acquisition starts.

16.1 A/D Acquisition Requirements

For the A/D converter to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the input channel voltage level. The analog input model is shown in Figure 16-2. The source impedance (R_S) and the internal sampling switch (R_{SS}) impedance directly affect the time required to charge the capacitor C_{HOLD}. The sampling switch (R_{SS}) impedance varies over the device voltage (V_{DD}). The source impedance affects the offset voltage at the analog input (due to pin leakage current). **The maximum recommended impedance for analog sources is 2.5 kΩ**. After the analog input channel is selected (changed), this acquisition must be done before the conversion can be started.

Note:	When the conversion is started, the holding capacitor is disconnected from the input pin.

FIGURE 16-2: ANALOG INPUT MODEL



ADDWFC — ADD WREG and Carry bit to f

Syntax:	[*label*] ADDWFC f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(WREG) + (f) + (C) $\rightarrow$ dest
Status Affected:	N,OV, C, DC, Z
Encoding:	`0010` `00da` `ffff` `ffff`
Description:	Add WREG, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in data memory location 'f'. If ‘a’ is 0, the Access Bank will be selected. If ‘a’ is 1, the BSR will not be overridden.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `ADDWFC    REG, 0, 1`

Before Instruction

```
Carry bit= 1
REG    =   0x02
WREG   =   0x4D
```

After Instruction

```
Carry bit= 0
REG    =   0x02
WREG   =   0x50
```

ANDLW — AND literal with WREG

Syntax:	[*label*] ANDLW k
Operands:	$0 \le k \le 255$
Operation:	(WREG) .AND. k $\rightarrow$ WREG
Status Affected:	N,Z
Encoding:	`0000` `1011` `kkkk` `kkkk`
Description:	The contents of WREG are ANDed with the 8-bit literal 'k'. The result is placed in WREG.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to WREG

Example: `ANDLW    0x5F`

Before Instruction

```
WREG   =   0xA3
```

After Instruction

```
WREG   =   0x03
```

BTG Bit Toggle f

Syntax:	[*label*] BTG f,b[,a]
Operands:	$0 \le f \le 255$ $0 \le b < 7$ $a \in [0,1]$
Operation:	$(\overline{f<b>}) \rightarrow f<b>$
Status Affected:	None
Encoding:	`0111` `bbba` `ffff` `ffff`
Description:	Bit 'b' in data memory location 'f' is inverted. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `BTG     PORTC,  4, 0`

Before Instruction:

```
PORTC   =   0111 0101 [0x75]
```

After Instruction:

```
PORTC   =   0110 0101 [0x65]
```

BOV Branch if Overflow

Syntax:	[*label*] BOV n
Operands:	$-128 \le n \le 127$
Operation:	if overflow bit is ’1’ $(PC) + 2 + 2n \rightarrow PC$
Status Affected:	None
Encoding:	`1110` `0100` `nnnn` `nnnn`
Description:	If the Overflow bit is ’1’, then the program will branch. The 2’s complement number ’2n’ is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE      BOV  Jump`

Before Instruction

```
PC          =   address (HERE)
```

After Instruction

```
If Overflow=   1;
    PC      =   address (Jump)
If Overflow=   0;
    PC      =   address (HERE+2)
```

GOTO — Unconditional Branch

Syntax:	[*label*] GOTO k
Operands:	$0 \le k \le 1048575$
Operation:	$k \rightarrow$ PC<20:1>
Status Affected:	None

Encoding:

1st word (k<7:0>)	`1110`	`1111`	$k_7$kkk	kkkk$_0$
2nd word(k<19:8>)	`1111`	k_{19}kkk	kkkk	kkkk$_8$

Description: `GOTO` allows an unconditional branch anywhere within entire 2 Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. `GOTO` is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	No operation	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `GOTO THERE`

After Instruction

```
PC  =   Address (THERE)
```

INCF — Increment f

Syntax:	[*label*] INCF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f) + 1 $\rightarrow$ dest
Status Affected:	C,DC,N,OV,Z
Encoding:	`0010` `10da` `ffff` `ffff`

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `INCF    CNT, 1, 0`

Before Instruction

```
CNT    =    0xFF
Z      =    0
C      =    ?
DC     =    ?
```

After Instruction

```
CNT    =    0x00
Z      =    1
C      =    1
DC     =    1
```

SUBWFB Subtract WREG from f with Borrow

Syntax:	[*label*] SUBWFB f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) – (WREG) – ($\overline{C}$) → dest
Status Affected:	N,OV, C, DC, Z
Encoding:	0101 | 10da | ffff | ffff
Description:	Subtract WREG and the carry flag (borrow) from register 'f' (2's complement method). If 'd' is 0, the result is stored in WREG. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example 1: `SUBWFB REG, 1, 0`

Before Instruction
```
REG     =   0x19   (0001 1001)
WREG    =   0x0D   (0000 1101)
C       =   1
```
After Instruction
```
REG     =   0x0C   (0000 1011)
WREG    =   0x0D   (0000 1101)
C       =   1
Z       =   0
N       =   0    ; result is positive
```

Example 2: `SUBWFB REG, 0, 0`

Before Instruction
```
REG     =   0x1B    (0001 1011)
WREG    =   0x1A    (0001 1010)
C       =   0
```
After Instruction
```
REG     =   0x1B   (0001 1011)
WREG    =   0x00
C       =   1
Z       =   1    ; result is zero
N       =   0
```

Example 3: `SUBWFB REG, 1, 0`

Before Instruction
```
REG     =   0x03   (0000 0011)
WREG    =   0x0E   (0000 1101)
C       =   1
```
After Instruction
```
REG     =   0xF5   (1111 0100)
                 ; [2's comp]
WREG    =   0x0E   (0000 1101)
C       =   0
Z       =   0
N       =   1    ; result is negative
```

SWAPF Swap f

Syntax:	[*label*] SWAPF f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f<3:0>) → dest<7:4>, (f<7:4>) → dest<3:0>
Status Affected:	None
Encoding:	0011 | 10da | ffff | ffff
Description:	The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `SWAPF REG, 1, 0`

Before Instruction
```
REG     =   0x53
```
After Instruction
```
REG     =   0x35
```

21.3 AC (Timing) Characteristics

21.3.1 TIMING PARAMETER SYMBOLOGY

The timing parameter symbols have been created following one of the following formats:

1. TppS2ppS		3. TCC:ST	(I^2C specifications only)
2. TppS		4. Ts	(I^2C specifications only)
T			
F	Frequency	T	Time

Lowercase letters (pp) and their meanings:

pp			
cc	CCP1	osc	OSC1
ck	CLKOUT	rd	$\overline{\text{RD}}$
cs	$\overline{\text{CS}}$	rw	$\overline{\text{RD}}$ or $\overline{\text{WR}}$
di	SDI	sc	SCK
do	SDO	ss	$\overline{\text{SS}}$
dt	Data in	t0	T0CKI
io	I/O port	t1	T1CKI
mc	$\overline{\text{MCLR}}$	wr	$\overline{\text{WR}}$

Uppercase letters and their meanings:

S			
F	Fall	P	Period
H	High	R	Rise
I	Invalid (Hi-impedance)	V	Valid
L	Low	Z	Hi-impedance
I^2C only			
AA	output access	High	High
BUF	Bus free	Low	Low

TCC:ST (I^2C specifications only)

CC			
HD	Hold	SU	Setup
ST			
DAT	DATA input hold	STO	STOP condition
STA	START condition		

FIGURE 21-15: EXAMPLE SPI SLAVE MODE TIMING (CKE = 1)

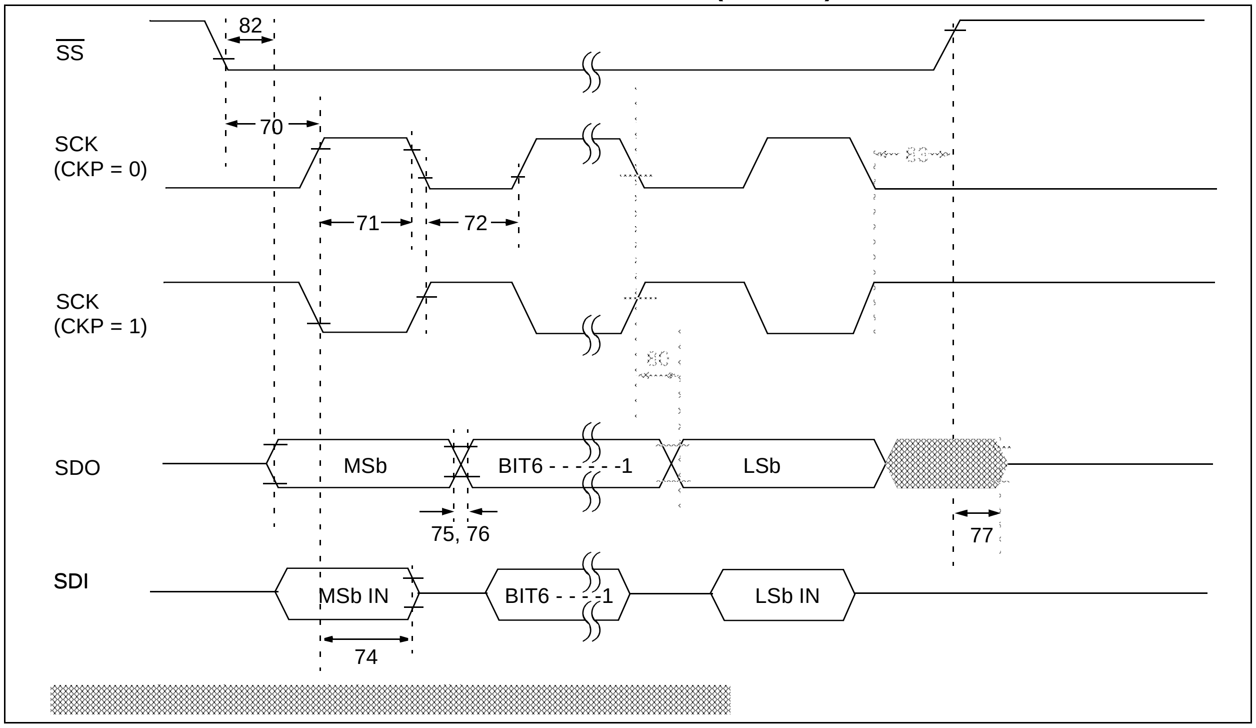


TABLE 21-14: EXAMPLE SPI SLAVE MODE REQUIREMENTS (CKE = 1)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}$↓ to SCK↓ or SCK↑ input		TCY	—	ns	
71	TscH	SCK input high time (Slave mode)	Continuous	1.25TCY + 30	—	ns	
71A			Single Byte	40	—	ns	**(Note 1)**
72	TscL	SCK input low time (Slave mode)	Continuous	1.25TCY + 30	—	ns	
72A			Single Byte	40	—	ns	**(Note 1)**
73A	TB2B	Last clock edge of Byte1 to the first clock edge of Byte2		1.5TCY + 40	—	ns	**(Note 2)**
74	TscH2diL, TscL2diL	Hold time of SDI data input to SCK edge		100	—	ns	
75	TdoR	SDO data output rise time	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX		45	ns	
76	TdoF	SDO data output fall time		—	25	ns	
77	TssH2doZ	$\overline{SS}$↑ to SDO output hi-impedance		10	50	ns	
78	TscR	SCK output rise time (Master mode)	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
79	TscF	SCK output fall time (Master mode)		—	25	ns	
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	
82	TssL2doV	SDO data output valid after $\overline{SS}$↓ edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	
83	TscH2ssH, TscL2ssH	$\overline{SS}$ ↑ after SCK edge		1.5TCY + 40	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter # 71A and # 72A are used.

ISBN: 9781620769676