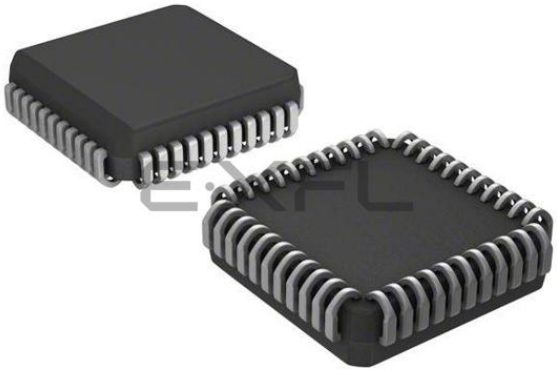


Welcome to E-XFL.COM

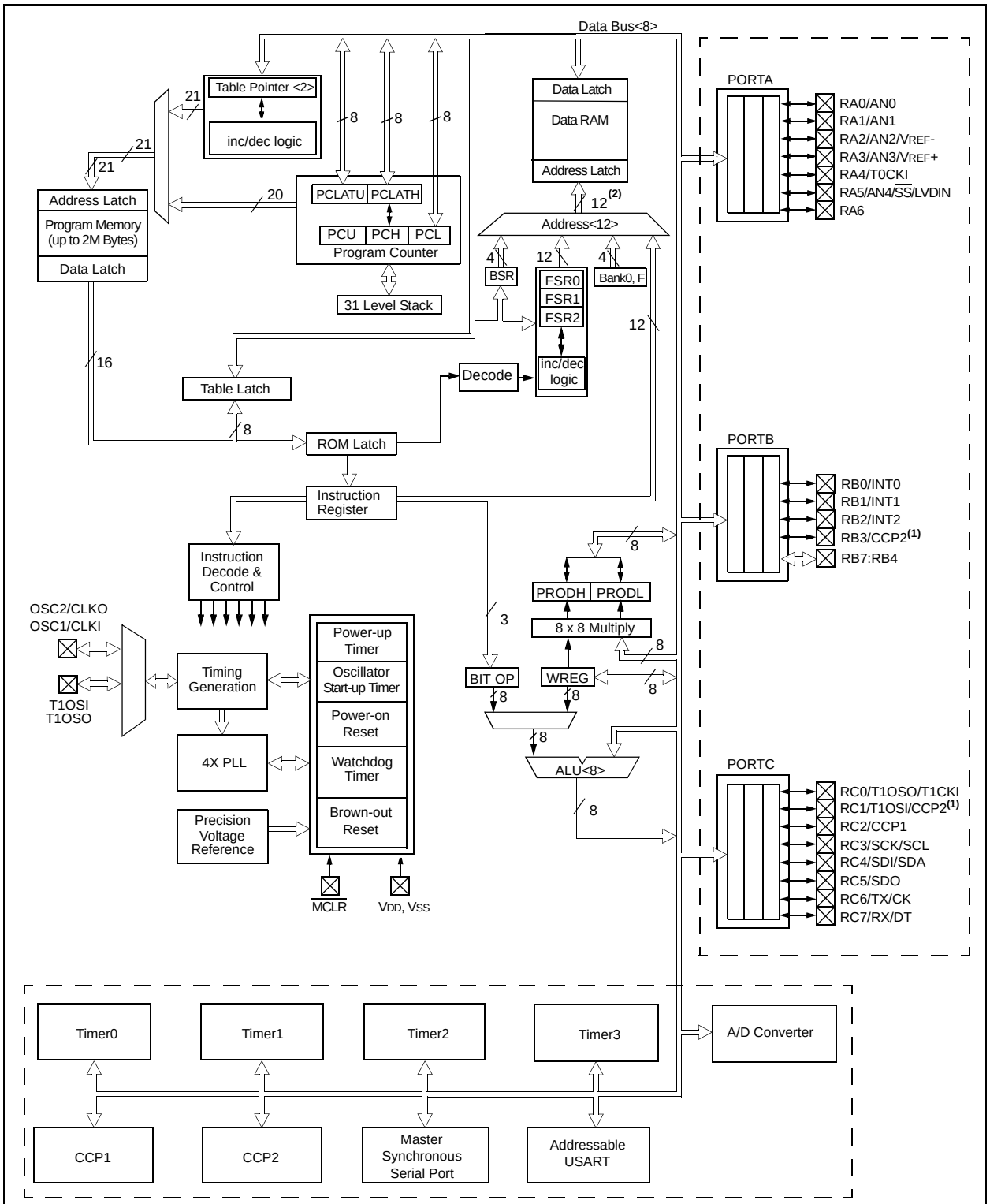
What is "Embedded - Microcontrollers"?

"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-LCC (J-Lead)
Supplier Device Package	44-PLCC (16.59x16.59)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18c442t-e-l

FIGURE 1-1: PIC18C2X2 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 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset RESET Instruction Stack Resets	Wake-up via WDT or Interrupt
TOSU	242	442	252	452	---0 0000	---0 0000	---0 uuuu(3)
TOSH	242	442	252	452	0000 0000	0000 0000	uuuu uuuu(3)
TOSL	242	442	252	452	0000 0000	0000 0000	uuuu uuuu(3)
STKPTR	242	442	252	452	00-0 0000	00-0 0000	uu-u uuuu(3)
PCLATU	242	442	252	452	---0 0000	---0 0000	---u uuuu
PCLATH	242	442	252	452	0000 0000	0000 0000	uuuu uuuu
PCL	242	442	252	452	0000 0000	0000 0000	PC + 2(2)
TBLPTRU	242	442	252	452	--00 0000	--00 0000	--uu uuuu
TBLPTRH	242	442	252	452	0000 0000	0000 0000	uuuu uuuu
TBLPTRL	242	442	252	452	0000 0000	0000 0000	uuuu uuuu
TABLAT	242	442	252	452	0000 0000	0000 0000	uuuu uuuu
PRODH	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
PRODL	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
INTCON	242	442	252	452	0000 000x	0000 000u	uuuu uuuu(1)
INTCON2	242	442	252	452	1111 -1-1	1111 -1-1	uuuu -u-u(1)
INTCON3	242	442	252	452	11-0 0-00	11-0 0-00	uu-u u-uu(1)
INDF0	242	442	252	452	N/A	N/A	N/A
POSTINC0	242	442	252	452	N/A	N/A	N/A
POSTDEC0	242	442	252	452	N/A	N/A	N/A
PREINC0	242	442	252	452	N/A	N/A	N/A
PLUSW0	242	442	252	452	N/A	N/A	N/A
FSR0H	242	442	252	452	---- 0000	---- 0000	---- uuuu
FSR0L	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
WREG	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF1	242	442	252	452	N/A	N/A	N/A
POSTINC1	242	442	252	452	N/A	N/A	N/A
POSTDEC1	242	442	252	452	N/A	N/A	N/A
PREINC1	242	442	252	452	N/A	N/A	N/A
PLUSW1	242	442	252	452	N/A	N/A	N/A

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: See Table 3-2 for RESET value for specific condition.

5: Bit 6 of PORTA, LATA, and TRISA are enabled in ECIO and RCIO oscillator modes only. In all other oscillator modes, they are disabled and read ’0’.

6: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ Reset.

7: Bit 6 of PORTA, LATA and TRISA are not available on all devices. When unimplemented, they are read as ’0’.

REGISTER 4-1: STKPTR REGISTER

R/C-0	R/C-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STKFUL	STKUNF	—	SP4	SP3	SP2	SP1	SP0
bit 7							bit 0

bit 7(1) **STKFUL**: Stack Full Flag bit
1 = Stack became full or overflowed
0 = Stack has not become full or overflowed

bit 6(1) **STKUNF**: Stack Underflow Flag bit
1 = Stack underflow occurred
0 = Stack underflow did not occur

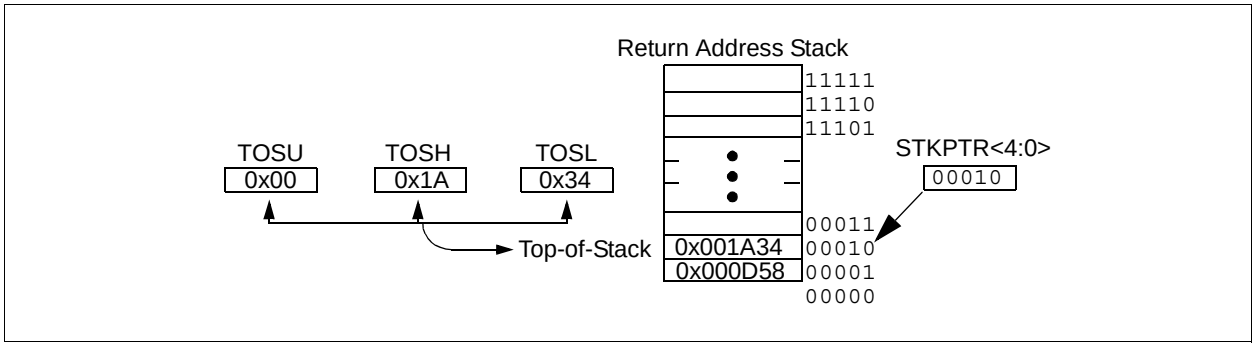
bit 5 **Unimplemented**: Read as '0'

bit 4-0 **SP4:SP0**: Stack Pointer Location bits

Note 1: Bit 7 and bit 6 can only be cleared in user software or by a POR.

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

FIGURE 4-3: RETURN ADDRESS STACK AND ASSOCIATED REGISTERS



4.2.3 PUSH AND POP INSTRUCTIONS

Since the Top-of-Stack (TOS) is readable and writable, the ability to push values onto the stack and pull values off the stack, without disturbing normal program execution, is a desirable option. To push the current PC value onto the stack, a PUSH instruction can be executed. This will increment the stack pointer and load the current PC value onto the stack. TOSU, TOSH and TOSL can then be modified to place a return address on the stack.

The ability to pull the TOS value off of the stack and replace it with the value that was previously pushed onto the stack, without disturbing normal execution, is achieved by using the POP instruction. The POP instruction discards the current TOS by decrementing the stack pointer. The previous value pushed onto the stack then becomes the TOS value.

4.2.4 STACK FULL/UNDERFLOW RESETS

These resets are enabled by programming the STVREN configuration bit. When the STVREN bit is disabled, a full or underflow condition will set the appropriate STKFUL or STKUNF bit, but not cause a device RESET. When the STVREN bit is enabled, a full or underflow will set the appropriate STKFUL or STKUNF bit and then cause a device RESET. The STKFUL or STKUNF bits are only cleared by the user software or a POR Reset.

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.

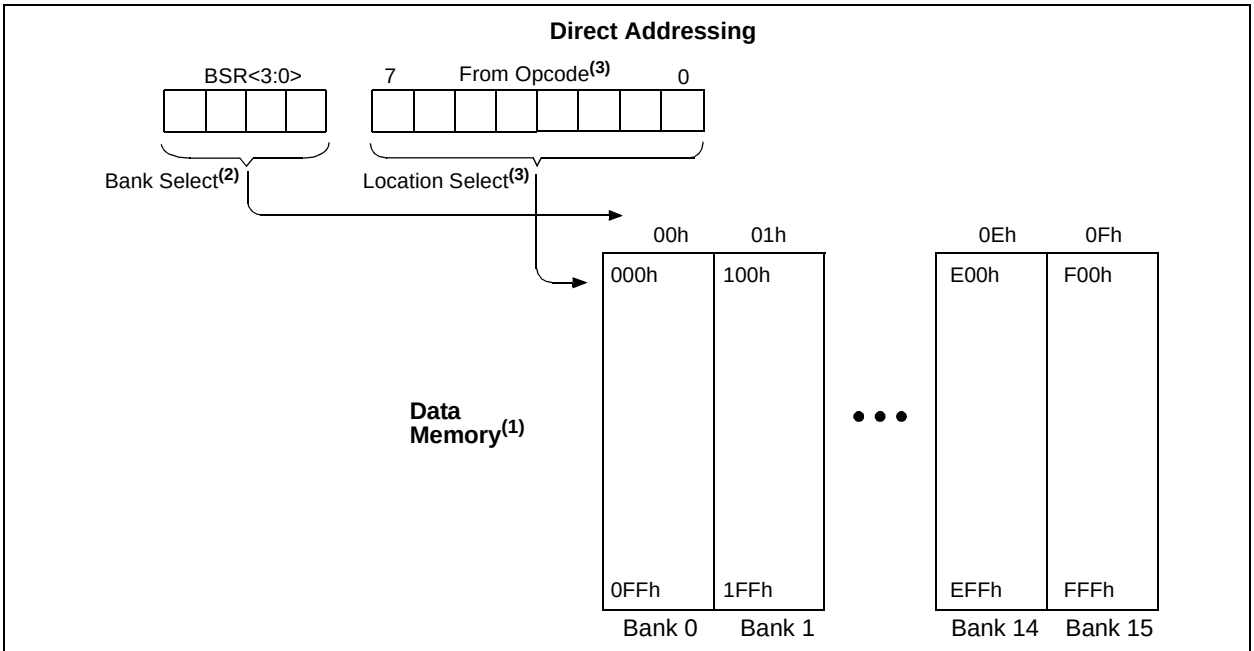
If the currently selected bank is not implemented, any read will return all '0's and all writes are ignored. The STATUS register bits will be set/cleared as appropriate for the instruction performed.

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.

5.0 TABLE READS/TABLE WRITES

Enhanced devices have two memory spaces: the program memory space and the data memory space. The program memory space is 16-bits wide, while the data memory space is 8 bits wide. Table Reads and Table Writes have been provided to move data between these two memory spaces through an 8-bit register (TABLAT).

The operations that allow the processor to move data between the data and program memory spaces are:

- Table Read (TBLRD)
- Table Write (TBLWT)

Table Read operations retrieve data from program memory and place it into the data memory space. Figure 5-1 shows the operation of a Table Read with program and data memory.

Table Write operations store data from the data memory space into program memory. Figure 5-2 shows the operation of a Table Write with program and data memory.

Table operations work with byte entities. A table block containing data is not required to be word aligned, so a table block can start and end at any byte address. If a Table Write is being used to write an executable program to program memory, program instructions will need to be word aligned.

FIGURE 5-1: TABLE READ OPERATION

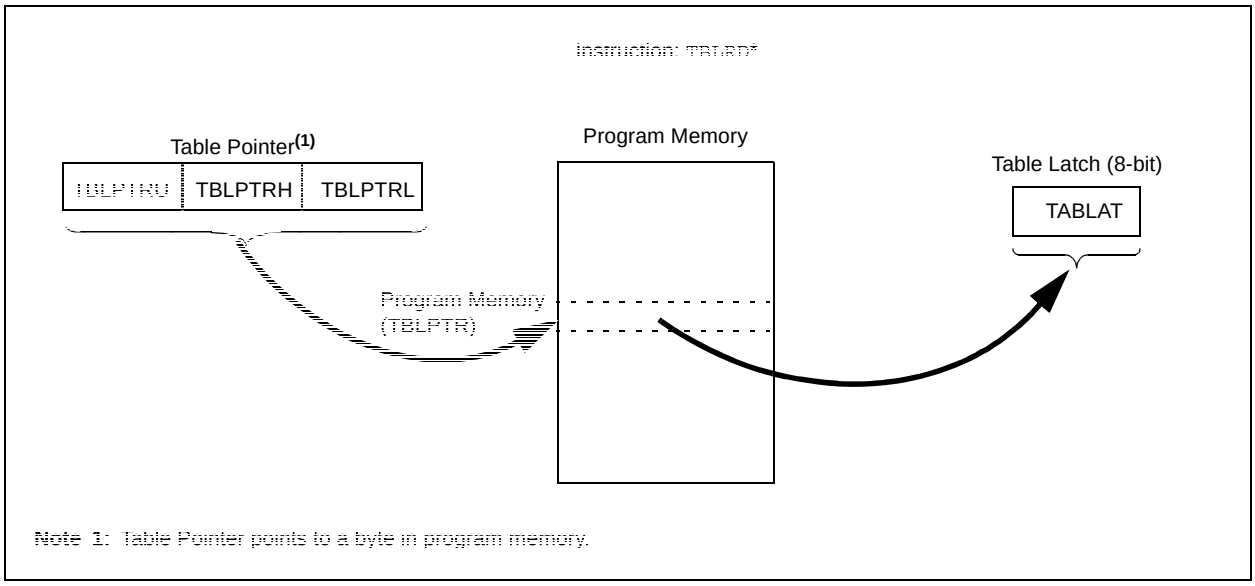
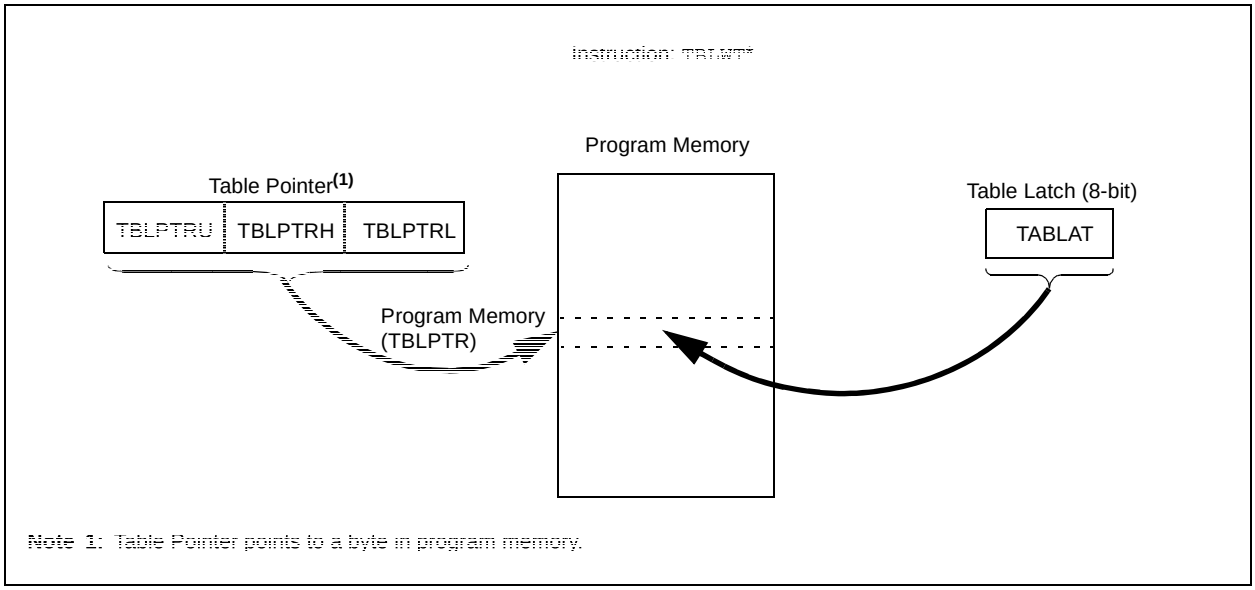


FIGURE 5-2: TABLE WRITE OPERATION



5.1 Control Registers

Several control registers are used in conjunction with the TBLRD and TBLWT instructions. These include the:

- TBLPTR registers
- TABLAT register
- RCON register

5.1.1 RCON REGISTER

The LWRT bit specifies the operation of Table Writes to internal memory when the VPP voltage is applied to the MCLR pin. When the LWRT bit is set, the controller continues to execute user code, but long Table Writes are allowed (for programming internal program memory) from user mode. The LWRT bit can be cleared only by performing either a POR or MCLR Reset.

REGISTER 5-1: RCON REGISTER (ADDRESS: FD0h)

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
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
1 = Enable TBLWT to internal program memory
0 = Disable TBLWT to internal program memory.

Note: Only cleared on a POR or MCLR Reset.
This bit has no effect on TBLWTs to external program memory.

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

bit 4 **$\overline{RI}$:** RESET Instruction Flag bit
1 = No RESET instruction occurred
0 = A RESET instruction occurred

bit 3 **$\overline{TO}$:** Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 2 **$\overline{PD}$:** Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 1 **$\overline{POR}$:** Power-on Reset Status bit
1 = No Power-on Reset occurred
0 = A Power-on Reset occurred (must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
1 = No Brown-out Reset or POR Reset occurred
0 = A Brown-out Reset or POR Reset occurred
(must be set in software after a Brown-out Reset occurs)

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

FIGURE 7-1: INTERRUPT LOGIC

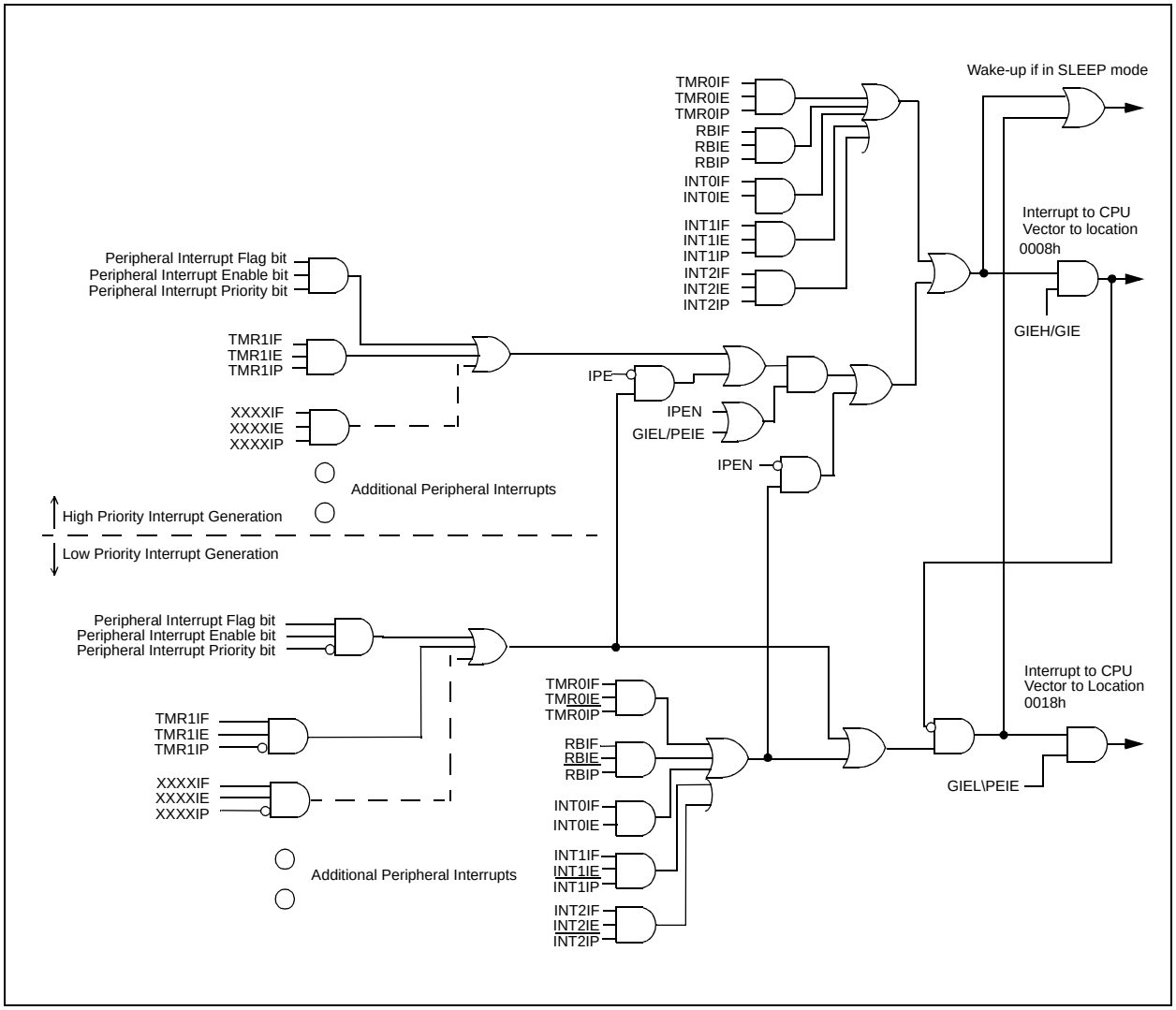


TABLE 8-1: PORTA FUNCTIONS

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.

8.3 PORTC, TRISC and LATC Registers

PORTC is an 8-bit wide, bi-directional port. The corresponding Data Direction Register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin).

Note:	On a Power-on Reset, these pins are configured as digital inputs.

The Data Latch register (LATC) is also memory mapped. Read-modify-write operations on the LATC register reads and writes the latched output value for PORTC.

PORTC is multiplexed with several peripheral functions (Table 8-5). PORTC pins have Schmitt Trigger input buffers.

When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

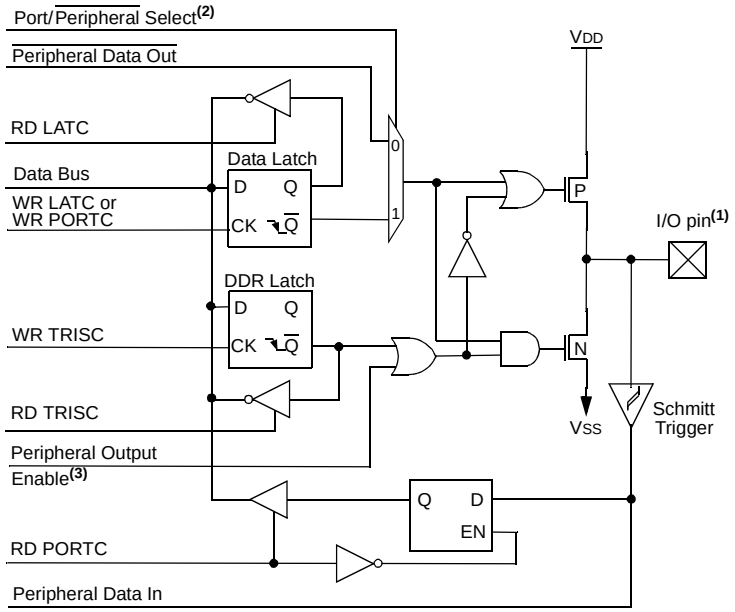
The pin override value is not loaded into the TRIS register. This allows read-modify-write of the TRIS register, without concern due to peripheral overrides.

RC1 is normally configured by the configuration bit CCP2MX as the default peripheral pin for the CCP2 module (default/erased state, CCP2MX = '1').

EXAMPLE 8-3: INITIALIZING PORTC

```
CLRF    PORTC    ; Initialize PORTC by
                 ; clearing output
                 ; data latches
CLRF    LATC     ; Alternate method
                 ; to clear output
                 ; data latches
MOVLW 0xCF       ; Value used to
                 ; initialize data
                 ; direction
MOVWF TRISC      ; Set RC<3:0> as inputs
                 ; RC<5:4> as outputs
                 ; RC<7:6> as inputs
```

FIGURE 8-7: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE)

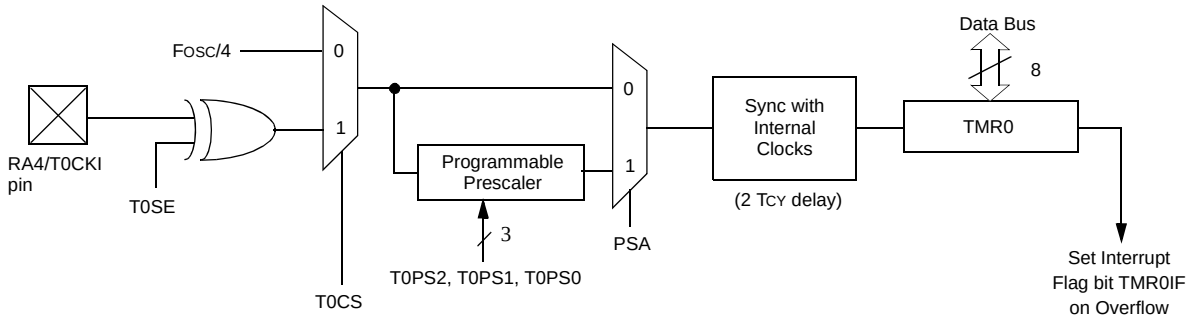


Note 1: I/O pins have diode protection to VDD and VSS.

2: Port/Peripheral select signal selects between port data (input) and peripheral output.

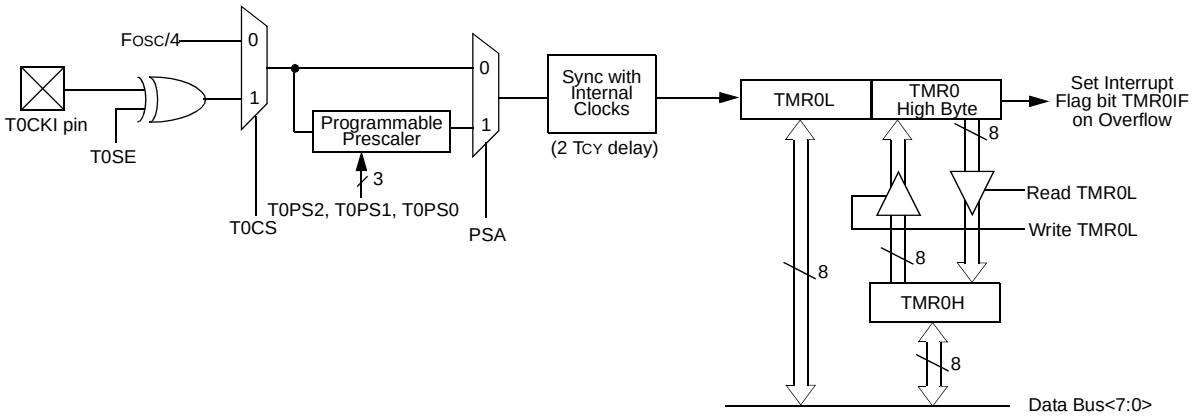
3: Peripheral Output Enable is only active if peripheral select is active.

FIGURE 9-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

FIGURE 9-2: TIMER0 BLOCK DIAGRAM IN 16-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

NOTES:

REGISTER 18-5: CONFIGURATION REGISTER 3 HIGH (CONFIG3H: BYTE ADDRESS 300005h)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/P-1
—	—	—	—	—	—	—	CCP2MX
bit 7							bit 0

bit 7-1 **Unimplemented:** Read as ’0’

bit 0 **CCP2MX:** CCP2 Mux bit

1 = CCP2 input/output is multiplexed with RC1

0 = CCP2 input/output is multiplexed with RB3

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 18-6: CONFIGURATION REGISTER 4 LOW (CONFIG4L: BYTE ADDRESS 300006h)

U-0	U-0	U-0	U-0	U-0	U-0	R/P-1	R/P-1
—	—	—	—	—	—	Reserved	STVREN
bit 7							bit 0

bit 7-2 **Unimplemented:** Read as ’0’

bit 1 **Reserved:** Maintain this bit set

bit 0 **STVREN:** Stack Full/Underflow Reset Enable bit

1 = Stack Full/Underflow will cause RESET

0 = Stack Full/Underflow will not cause RESET

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

MOVFF **Move f to f**

Syntax: [*label*] MOVFF f_s,f_d

Operands: $0 \le f_s \le 4095$
$0 \le f_d \le 4095$

Operation: $(f_s) \rightarrow f_d$

Status Affected: None

Encoding:

1st word (source)	1100	ffff	ffff	$ffff_s$
2nd word (destin.)	1111	ffff	ffff	$ffff_d$

Description: The contents of source register 'f_s' are moved to destination register 'f_d'. Location of source 'f_s' can be anywhere in the 4096 byte data space (000h to FFFh), and location of destination 'f_d' can also be anywhere from 000h to FFFh.
Either source or destination can be WREG (a useful special situation).
MOVFF is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

The MOVFF instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.

Words: 2

Cycles: 2 (3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f' (src)	Process Data	No operation
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: MOVFF REG1, REG2

Before Instruction
REG1 = 0x33
REG2 = 0x11

After Instruction
REG1 = 0x33,
REG2 = 0x33

MOVLB **Move literal to low nibble in BSR**

Syntax: [*label*] MOVLB k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow BSR$

Status Affected: None

Encoding:

0000	0001	kkkk	kkkk

Description: The 8-bit literal 'k' is loaded into the Bank Select Register (BSR).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: MOVLB 5

Before Instruction
BSR register = 0x02

After Instruction
BSR register = 0x05

RRNCF **Rotate Right f (no carry)**

Syntax: [*label*] RRNCF f [,d [,a]

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

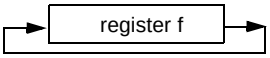
Operation: (f<n>) → dest<n-1>,
(f<0>) → dest<7>

Status Affected: N,Z

Encoding:

0100	00da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the right. 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' 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:
```
RRNCF   REG, 1, 0
```
Before Instruction
```
REG     =   1101 0111
```
After Instruction
```
REG     =   1110 1011
```

Example 2:
```
RRNCF   REG, 0, 0
```
Before Instruction
```
WREG    =   ?
REG     =   1101 0111
```
After Instruction
```
WREG    =   1110 1011
REG     =   1101 0111
```

SETF **Set f**

Syntax: [*label*] SETF f [,a]

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: FFh → f

Status Affected: None

Encoding:

0110	100a	ffff	ffff

Description: The contents of the specified register are set to FFh. 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 register 'f'

Example:
```
SETF    REG,1
```
Before Instruction
```
REG     =   0x5A
```
After Instruction
```
REG     =   0xFF
```

SUBWFB **Subtract WREG from f with Borrow**

Syntax: [*label*] SUBWFB f [,d [,a]

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

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: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

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
```

TBLRD **Table Read**

Syntax: [*label*] TBLRD (*; *+; *-; +*)

Operands: None

Operation:
if TBLRD *,
(Prog Mem (TBLPTR)) → TABLAT;
TBLPTR - No Change;
if TBLRD *+,
(Prog Mem (TBLPTR)) → TABLAT;
(TBLPTR) +1 → TBLPTR;
if TBLRD *-,
(Prog Mem (TBLPTR)) → TABLAT;
(TBLPTR) -1 → TBLPTR;
if TBLRD +*,
(TBLPTR) +1 → TBLPTR;
(Prog Mem (TBLPTR)) → TABLAT;

Status Affected: None

Encoding:

0000	0000	0000	10nn nn=0 * =1 *+ =2 *- =3 +*

Description: This instruction is used to read the contents of Program Memory (P.M.). To address the program memory, a pointer called Table Pointer (TBLPTR) is used.
The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 Mbyte address range.

TBLPTR[0] = 0: Least Significant Byte of Program Memory Word

TBLPTR[0] = 1: Most Significant Byte of Program Memory Word

The TBLRD instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read Program Memory)	No operation	No operation (Write TABLAT)

TBLRD **Table Read (cont'd)**

Example 1:
```
TBLRD  *+ ;
```
Before Instruction
```
TABLAT              =   0x55
TBLPTR              =   0x00A356
MEMORY(0x00A356)    =   0x34
```
After Instruction
```
TABLAT              =   0x34
TBLPTR              =   0x00A357
```

Example 2:
```
TBLRD  +* ;
```
Before Instruction
```
TABLAT              =   0xAA
TBLPTR              =   0x01A357
MEMORY(0x01A357)    =   0x12
MEMORY(0x01A358)    =   0x34
```
After Instruction
```
TABLAT              =   0x34
TBLPTR              =   0x01A358
```

FIGURE 21-1: PIC18CXX2 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL, EXTENDED)

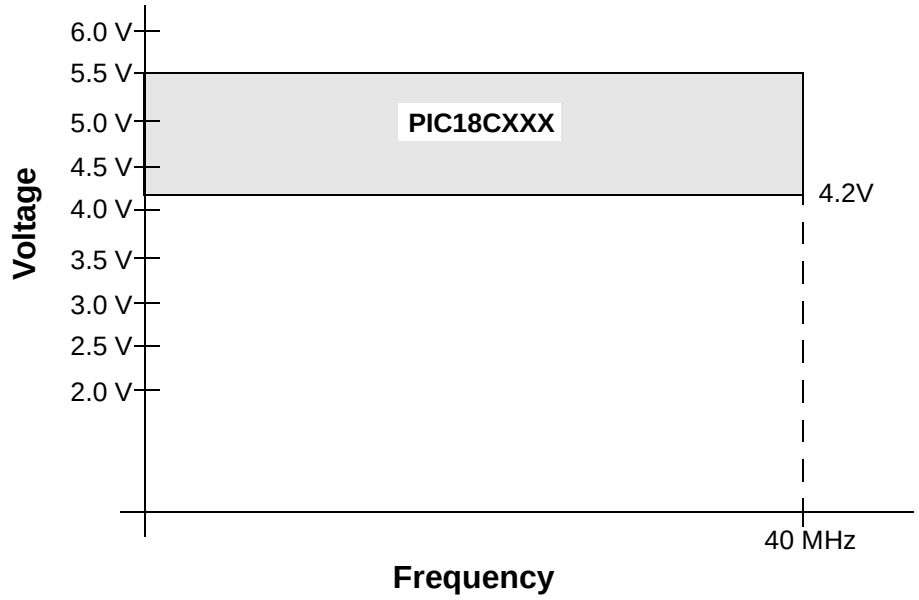
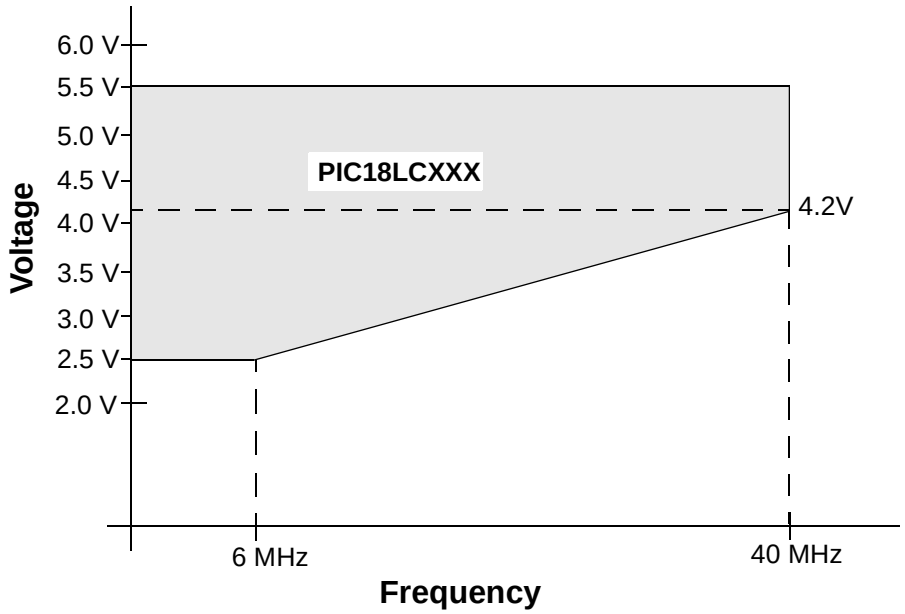


FIGURE 21-2: PIC18LCXX2 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)



F_{MAX} = (20.0 MHz/V) ($V_{DDAPPMIN}$ - 2.5 V) + 6 MHz

Note: $V_{DDAPPMIN}$ is the minimum voltage of the PIC® device in the application.

FIGURE 22-3: TYPICAL IDD vs. FOSC OVER VDD (HS/PLL MODE)

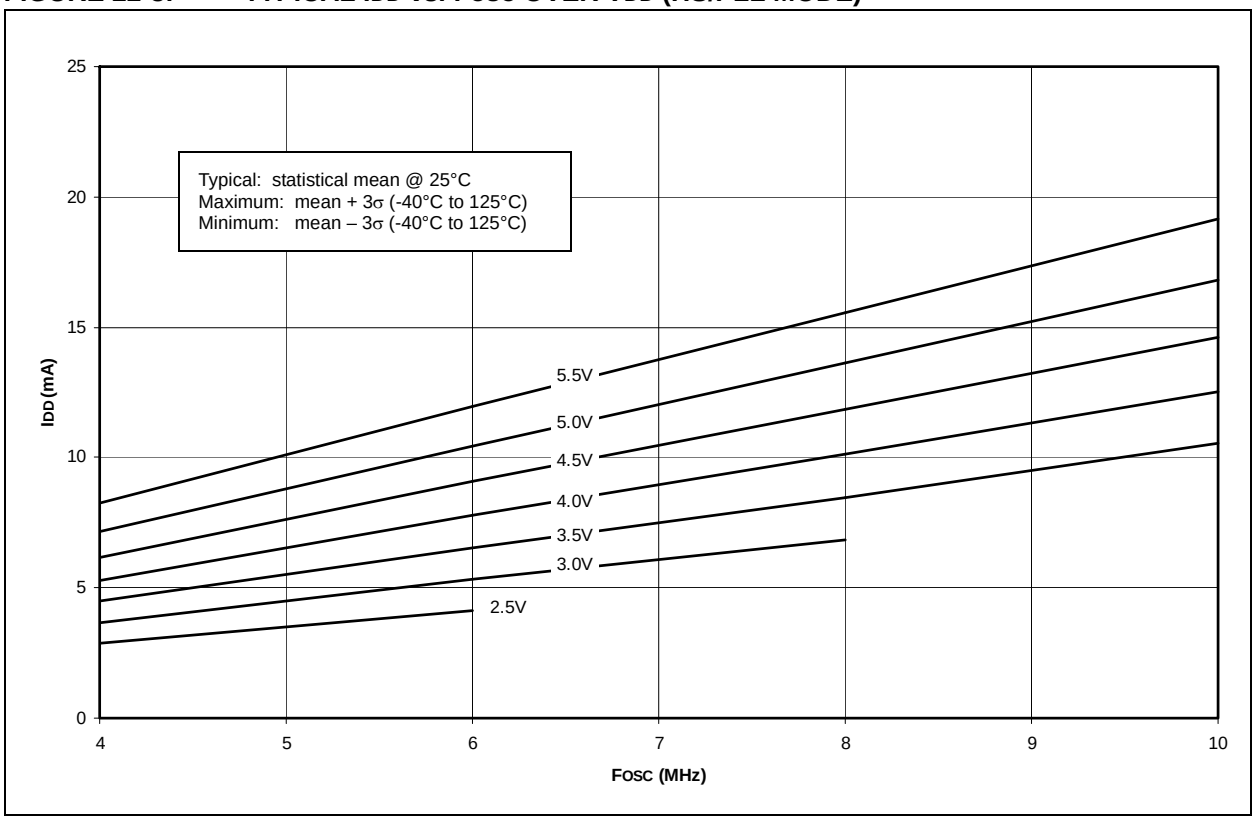


FIGURE 22-4: MAXIMUM IDD vs. FOSC OVER VDD (HS/PLL MODE)

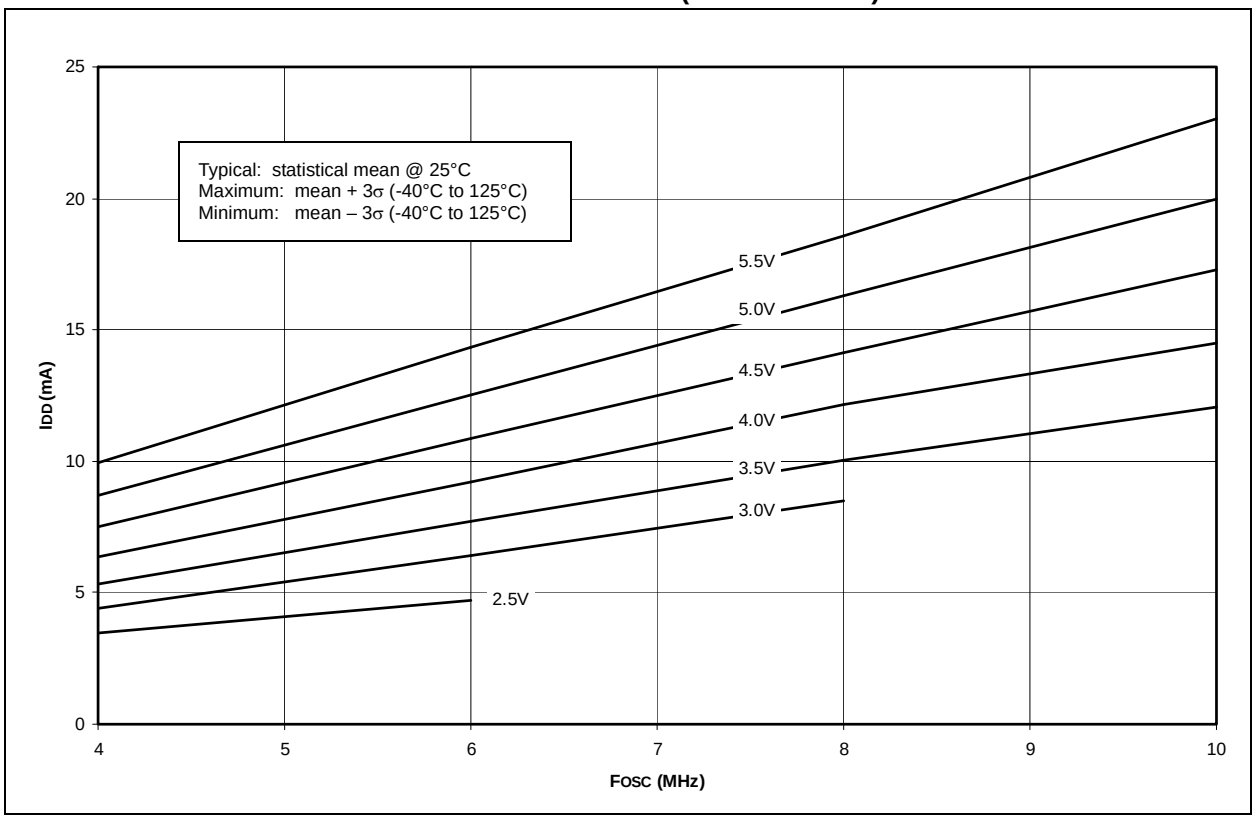


FIGURE 22-5: TYPICAL IDD vs. FOSC OVER VDD (XT MODE)

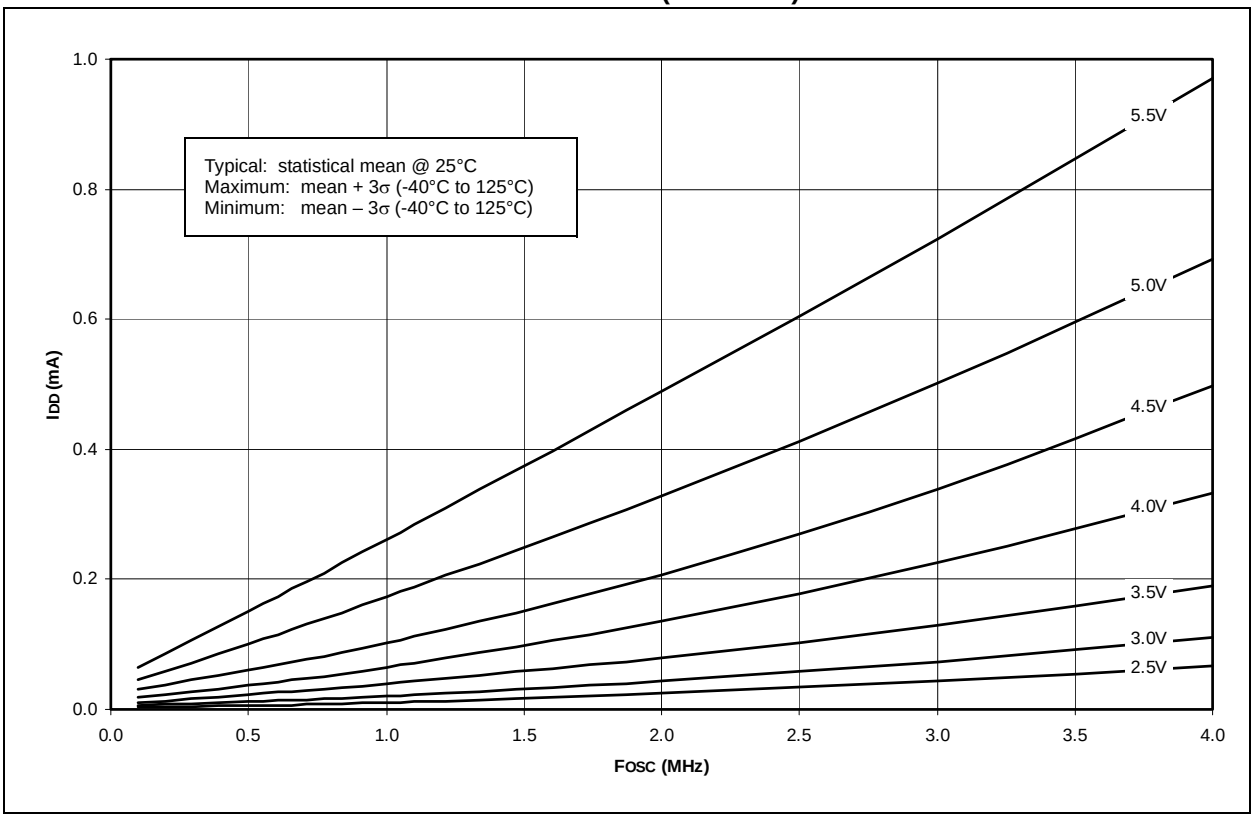


FIGURE 22-6: MAXIMUM IDD vs. FOSC OVER VDD (XT MODE)

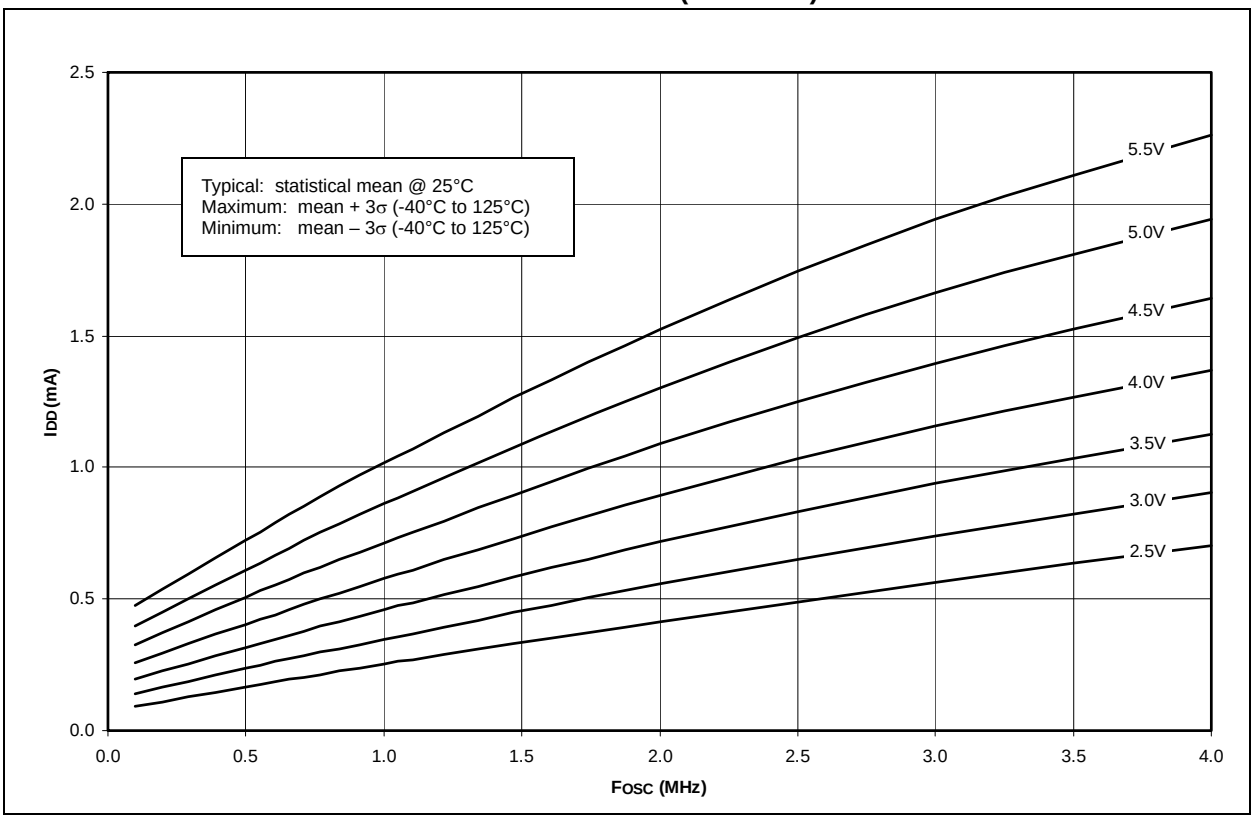


FIGURE 22-19: ΔILVD VS. VDD OVER TEMPERATURE (LVD ENABLED, VLVD = 4.5V - 4.78V)

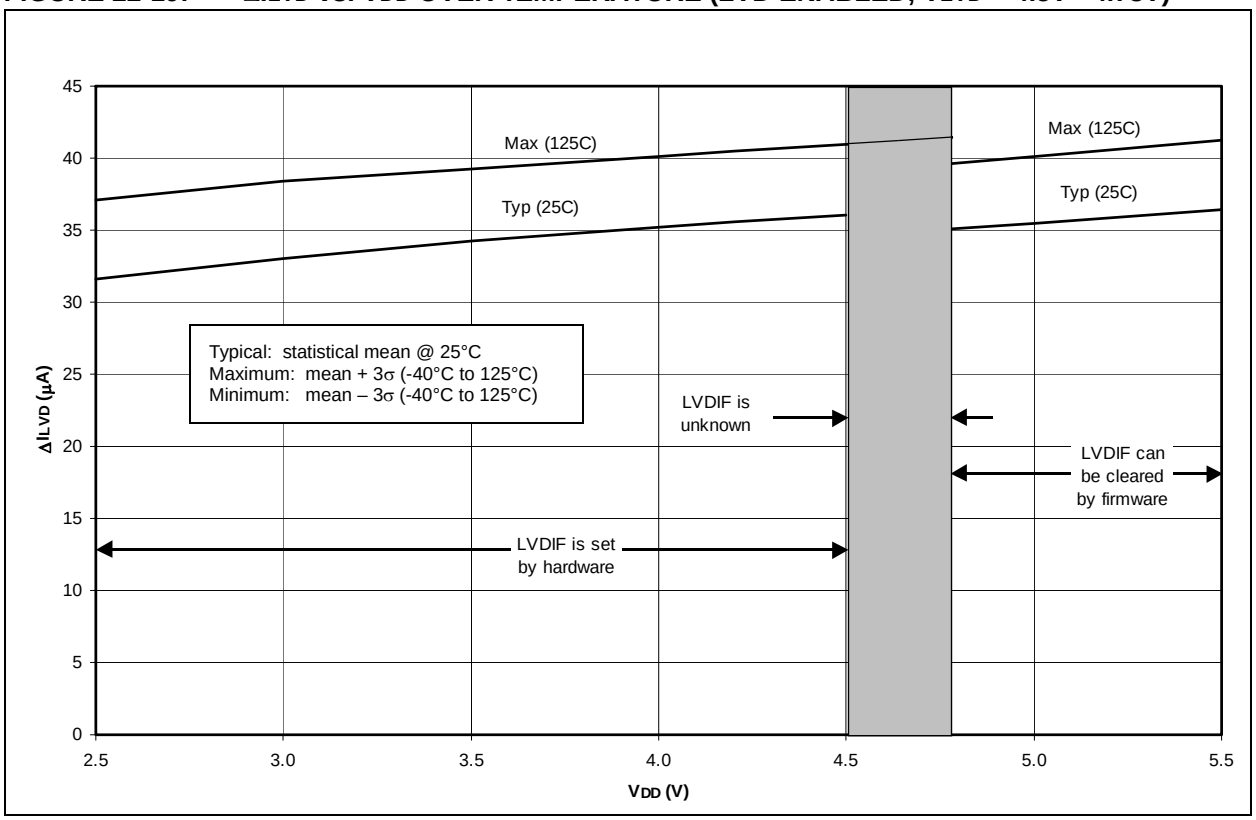


FIGURE 22-20: TYPICAL, MINIMUM AND MAXIMUM VOH VS. IOH (VDD = 5V, -40°C TO +125°C)