

PIC16F8X

PIC16CXX devices contain an 8-bit ALU and working register. The ALU is a general purpose arithmetic unit. It performs arithmetic and Boolean functions between data in the working register and any register file.

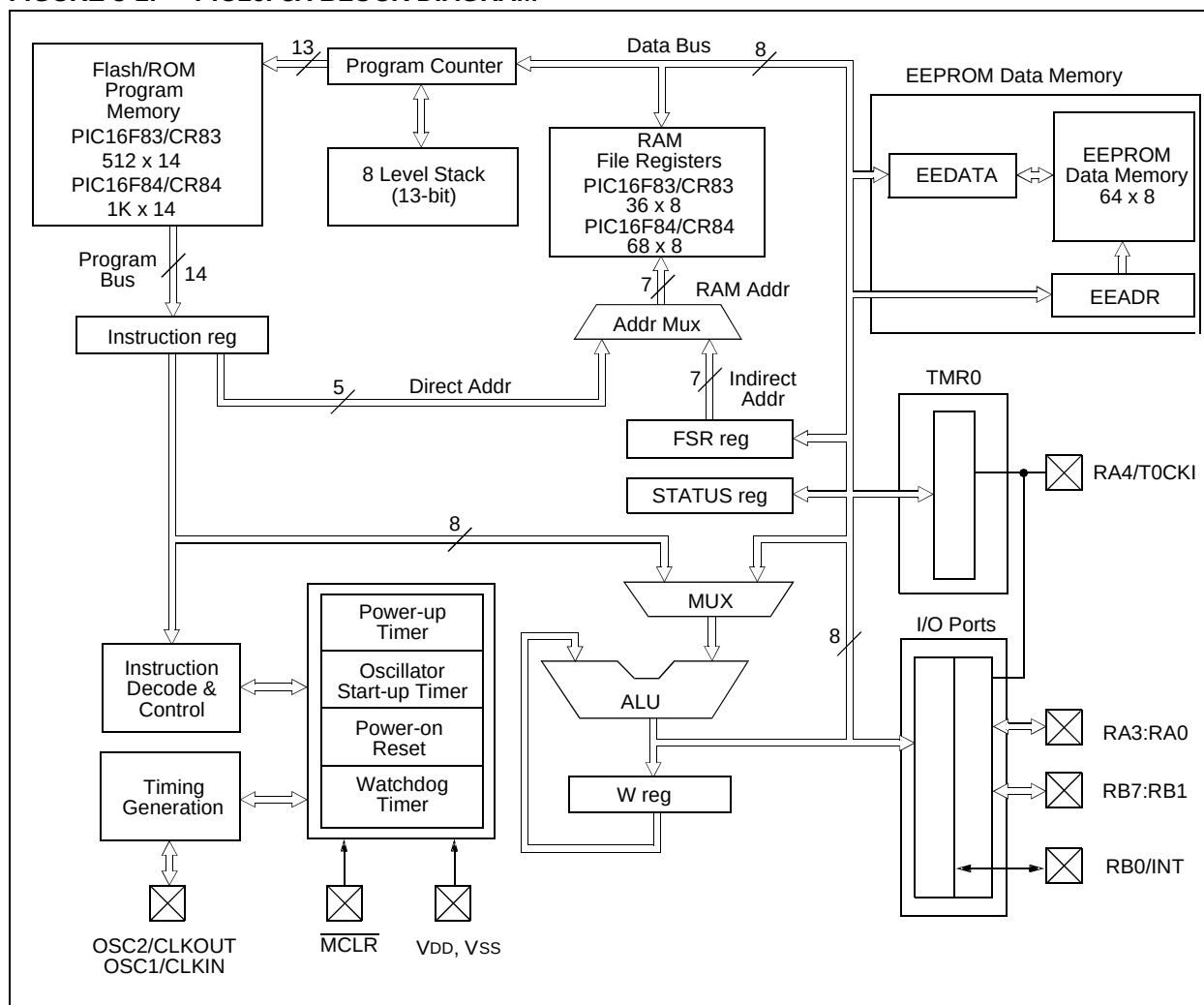
The ALU is 8-bits wide and capable of addition, subtraction, shift and logical operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature. In two-operand instructions, typically one operand is the working register (W register), and the other operand is a file register or an immediate constant. In single operand instructions, the operand is either the W register or a file register.

The W register is an 8-bit working register used for ALU operations. It is not an addressable register.

Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. The C and DC bits operate as a borrow and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

A simplified block diagram for the PIC16F8X is shown in Figure 3-1, its corresponding pin description is shown in Table 3-1.

FIGURE 3-1: PIC16F8X BLOCK DIAGRAM



PIC16F8X

3.1 Clocking Scheme/Instruction Cycle

The clock input (from OSC1) is internally divided by four to generate four non-overlapping quadrature clocks namely Q1, Q2, Q3 and Q4. Internally, the program counter (PC) is incremented every Q1, the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow is shown in Figure 3-2.

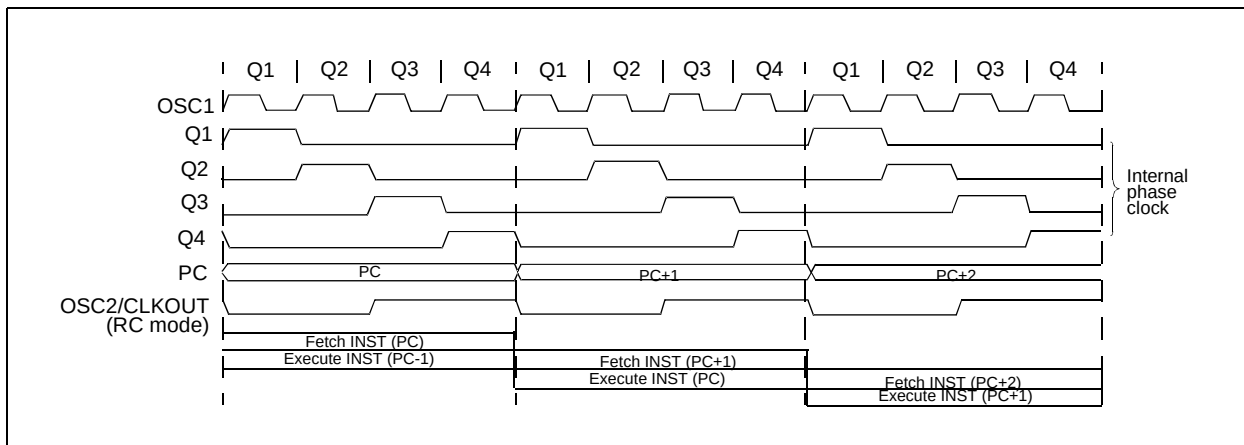
3.2 Instruction Flow/Pipelining

An "Instruction Cycle" consists of four Q cycles (Q1, Q2, Q3 and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g., GOTO) then two cycles are required to complete the instruction (Example 3-1).

A fetch cycle begins with the Program Counter (PC) incrementing in Q1.

In the execution cycle, the fetched instruction is latched into the "Instruction Register" in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

FIGURE 3-2: CLOCK/INSTRUCTION CYCLE

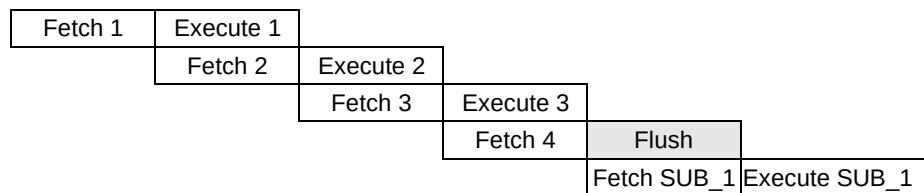


EXAMPLE 3-1: INSTRUCTION PIPELINE FLOW

```

1. MOVLW 55h
2. MOVWF PORTB
3. CALL SUB_1
4. BSF PORTA, BIT3

```



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

4.0 MEMORY ORGANIZATION

There are two memory blocks in the PIC16F8X. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the "core" are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

The data memory area also contains the data EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write. The 64 bytes of data EEPROM memory have the address range 0h-3Fh. More details on the EEPROM memory can be found in Section 7.0.

4.1 Program Memory Organization

The PIC16FXX has a 13-bit program counter capable of addressing an 8K x 14 program memory space. For the PIC16F83 and PIC16CR83, the first 512 x 14 (0000h-01FFh) are physically implemented (Figure 4-1). For the PIC16F84 and PIC16CR84, the first 1K x 14 (0000h-03FFh) are physically implemented (Figure 4-2). Accessing a location above the physically implemented address will cause a wrap-around. For example, for the PIC16F84 locations 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, and 1C20h will be the same instruction.

The reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 4-1: PROGRAM MEMORY MAP AND STACK - PIC16F83/CR83

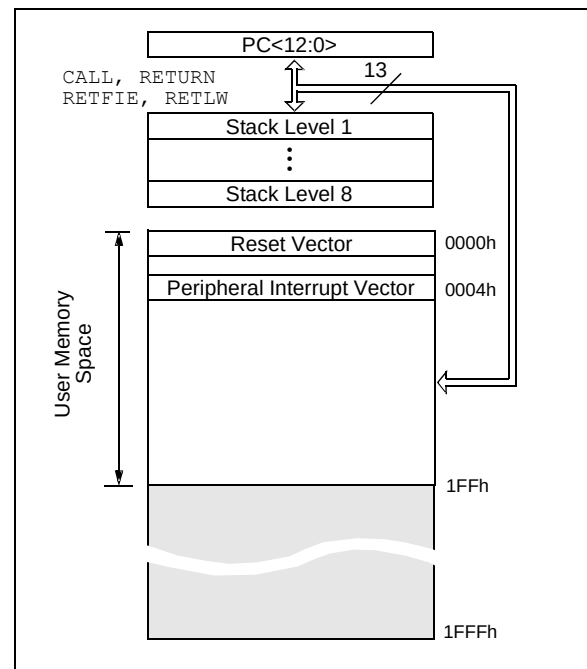
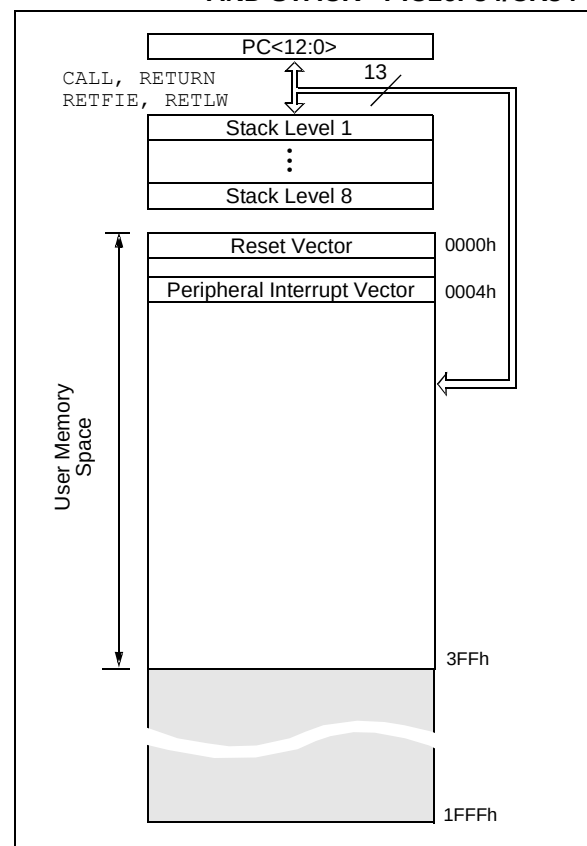


FIGURE 4-2: PROGRAM MEMORY MAP AND STACK - PIC16F84/CR84



4.2 Data Memory Organization

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 4-1 and Figure 4-2 show the data memory map organization.

Instructions `MOVWF` and `MOVF` can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 4.5). Indirect addressing uses the present value of the RP1:RP0 bits for access into the banked areas of data memory.

Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

4.2.1 GENERAL PURPOSE REGISTER FILE

All devices have some amount of General Purpose Register (GPR) area. Each GPR is 8 bits wide and is accessed either directly or indirectly through the FSR (Section 4.5).

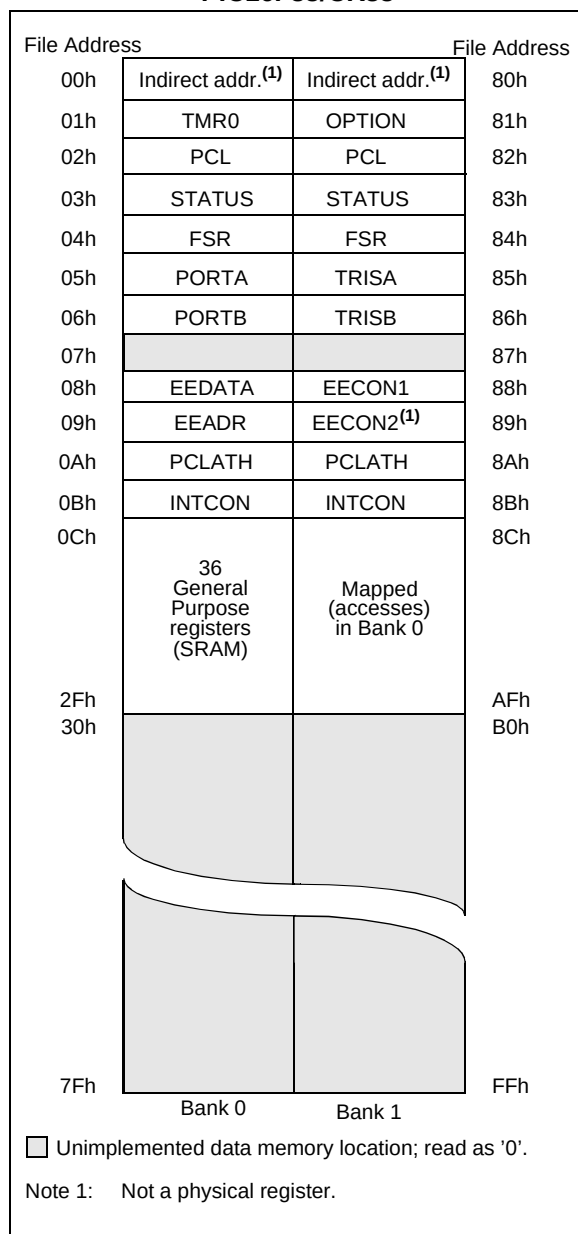
The GPR addresses in bank 1 are mapped to addresses in bank 0. As an example, addressing location 0Ch or 8Ch will access the same GPR.

4.2.2 SPECIAL FUNCTION REGISTERS

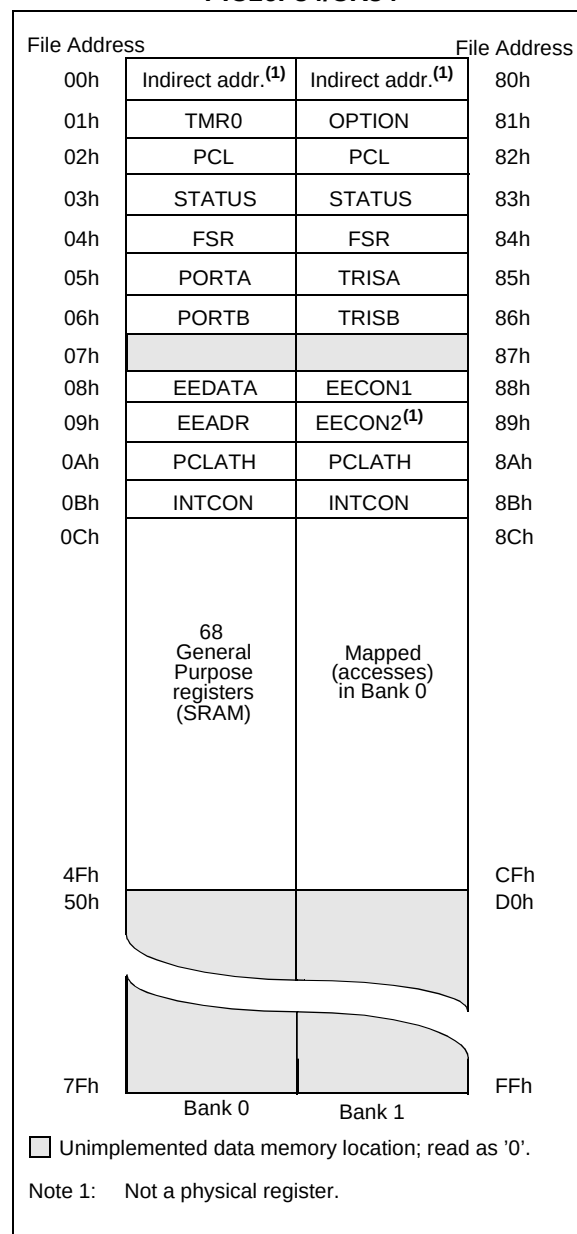
The Special Function Registers (Figure 4-1, Figure 4-2 and Table 4-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

**FIGURE 4-1: REGISTER FILE MAP -
PIC16F83/CR83**



**FIGURE 4-2: REGISTER FILE MAP -
PIC16F84/CR84**



PIC16F8X

TABLE 4-1 REGISTER FILE SUMMARY

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 (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- --	---- --
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS ⁽²⁾	IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- --	---- --
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC ⁽¹⁾					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- --	---- --
81h	OPTION_REG	\overline{RBPU}	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS ⁽²⁾	IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- --	---- --
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- --	---- --
8Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC ⁽¹⁾					---0 0000	---0 0000
8Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u

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

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The \overline{TO} and \overline{PD} status bits in the STATUS register are not affected by a \overline{MCLR} reset.

3: Other (non power-up) resets include: external reset through \overline{MCLR} and the Watchdog Timer Reset.

4.2.2.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the \overline{TO} and \overline{PD} bits are not writable. 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).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 9-2) because these instructions do not affect any status bit.

Note 1: The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16F8X and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.

Note 2: The C and DC bits operate as a borrow and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

Note 3: When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic.

FIGURE 4-1: STATUS REGISTER (ADDRESS 03h, 83h)

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/borrow bit (for `ADDWF` and `ADDLW` instructions) (For 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/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 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.

PIC16F8X

4.2.2.2 OPTION_REG REGISTER

The OPTION_REG register is a readable and writable register which contains various control bits to configure the TMR0/WDT prescaler, the external INT interrupt, TMR0, and the weak pull-ups on PORTB.

Note: When the prescaler is assigned to the WDT (PSA = '1'), TMR0 has a 1:1 prescaler assignment.

FIGURE 4-1: OPTION_REG REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBP _U	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

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

bit 7: **RBP_U**: 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

4.2.2.3 INTCON REGISTER

The INTCON register is a readable and writable register which contains the various enable bits for all interrupt sources.

Note: Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

FIGURE 4-1: INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit7							bit0

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

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: **TOIE:** 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

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

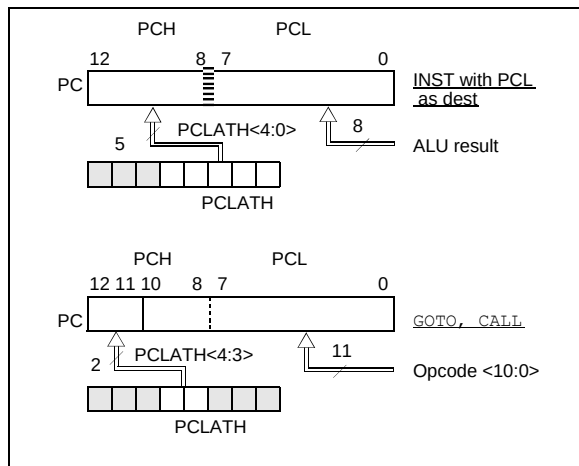
bit 1: **INTF:** RB0/INT Interrupt Flag bit
1 = The RB0/INT interrupt occurred
0 = The RB0/INT interrupt did not occur

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

4.3 Program Counter: PCL and PCLATH

The Program Counter (PC) is 13-bits wide. The low byte is the PCL register, which is a readable and writable register. The high byte of the PC (PC<12:8>) is not directly readable nor writable and comes from the PCLATH register. The PCLATH (PC latch high) register is a holding register for PC<12:8>. The contents of PCLATH are transferred to the upper byte of the program counter when the PC is loaded with a new value. This occurs during a `CALL`, `GOTO` or a write to PCL. The high bits of PC are loaded from PCLATH as shown in Figure 4-1.

FIGURE 4-1: LOADING OF PC IN DIFFERENT SITUATIONS



4.3.1 COMPUTED GOTO

A computed GOTO is accomplished by adding an offset to the program counter (`ADDWF PC`). When doing a table read using a computed GOTO method, care should be exercised if the table location crosses a PCL memory boundary (each 256 word block). Refer to the application note "Implementing a Table Read" (AN556).

4.3.2 PROGRAM MEMORY PAGING

The PIC16F83 and PIC16CR83 have 512 words of program memory. The PIC16F84 and PIC16CR84 have 1K of program memory. The `CALL` and `GOTO` instructions have an 11-bit address range. This 11-bit address range allows a branch within a 2K program memory page size. For future PIC16F8X program memory expansion, there must be another two bits to specify the program memory page. These paging bits come from the PCLATH<4:3> bits (Figure 4-1). When doing a `CALL` or a `GOTO` instruction, the user must ensure that these page bits (PCLATH<4:3>) are programmed to the desired program memory page. If a `CALL` instruction (or interrupt) is executed, the entire 13-bit PC is "pushed" onto the stack (see next section). Therefore,

manipulation of the PCLATH<4:3> is not required for the return instructions (which "pops" the PC from the stack).

Note: The PIC16F8X ignores the PCLATH<4:3> bits, which are used for program memory pages 1, 2 and 3 (0800h - 1FFFh). The use of PCLATH<4:3> as general purpose R/W bits is not recommended since this may affect upward compatibility with future products.

4.4 Stack

The PIC16FXX has an 8 deep x 13-bit wide hardware stack (Figure 4-1). The stack space is not part of either program or data space and the stack pointer is not readable or writable.

The entire 13-bit PC is "pushed" onto the stack when a `CALL` instruction is executed or an interrupt is acknowledged. The stack is "popped" in the event of a `RETURN`, `RETLW` or a `RETFIE` instruction execution. PCLATH is not affected by a push or a pop operation.

Note: There are no instruction mnemonics called push or pop. These are actions that occur from the execution of the `CALL`, `RETURN`, `RETLW`, and `RETFIE` instructions, or the vectoring to an interrupt address.

The stack operates as a circular buffer. That is, after the stack has been pushed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

If the stack is effectively popped nine times, the PC value is the same as the value from the first pop.

Note: There are no status bits to indicate stack overflow or stack underflow conditions.

4.5 Indirect Addressing: INDF and FSR Registers

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

EXAMPLE 4-1: INDIRECT ADDRESSING

- Register file 05 contains the value 10h
- Register file 06 contains the value 0Ah
- Load the value 05 into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR = 06)
- A read of the INDF register now will return the value of 0Ah.

Reading INDF itself indirectly (FSR = 0) will produce 00h. Writing to the INDF register indirectly results in a no-operation (although STATUS bits may be affected).

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 4-2.

EXAMPLE 4-2: HOW TO CLEAR RAM USING INDIRECT ADDRESSING

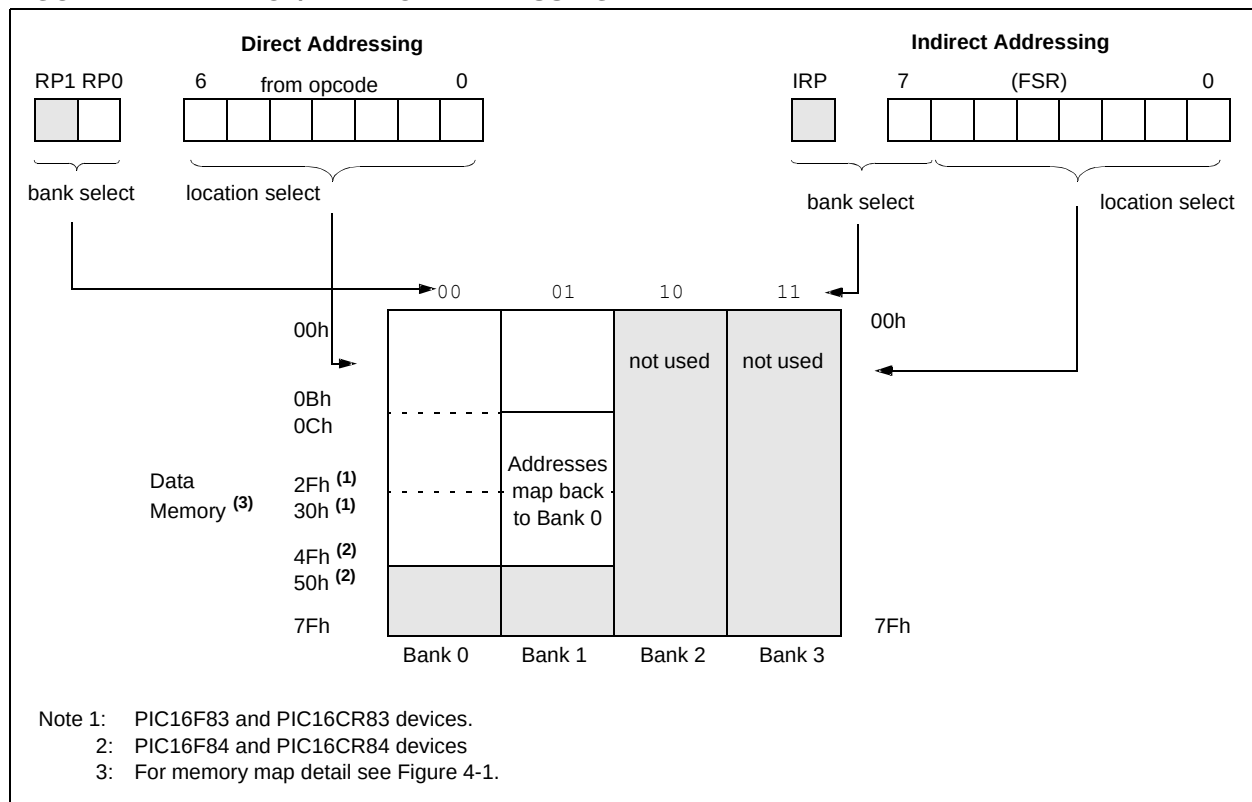
```

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
    
```

An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 4-1. However, IRP is not used in the PIC16F8X.

FIGURE 4-1: DIRECT/INDIRECT ADDRESSING



5.0 I/O PORTS

The PIC16F8X has two ports, PORTA and PORTB. Some port pins are multiplexed with an alternate function for other features on the device.

5.1 PORTA and TRISA Registers

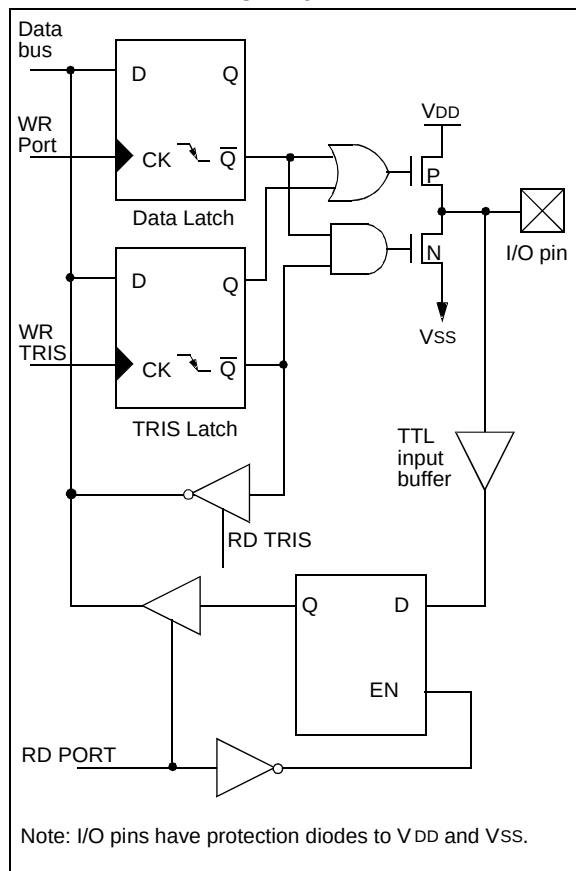
PORTA is a 5-bit wide latch. RA4 is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers. All pins have data direction bits (TRIS registers) which can configure these pins as output or input.

Setting a TRISA bit (=1) will make the corresponding PORTA pin an input, i.e., put the corresponding output driver in a hi-impedance mode. Clearing a TRISA bit (=0) will make the corresponding PORTA pin an output, i.e., put the contents of the output latch on the selected pin.

Reading the PORTA register reads the status of the pins whereas writing to it will write to the port latch. All write operations are read-modify-write operations. So a write to a port implies that the port pins are first read, then this value is modified and written to the port data latch.

The RA4 pin is multiplexed with the TMR0 clock input.

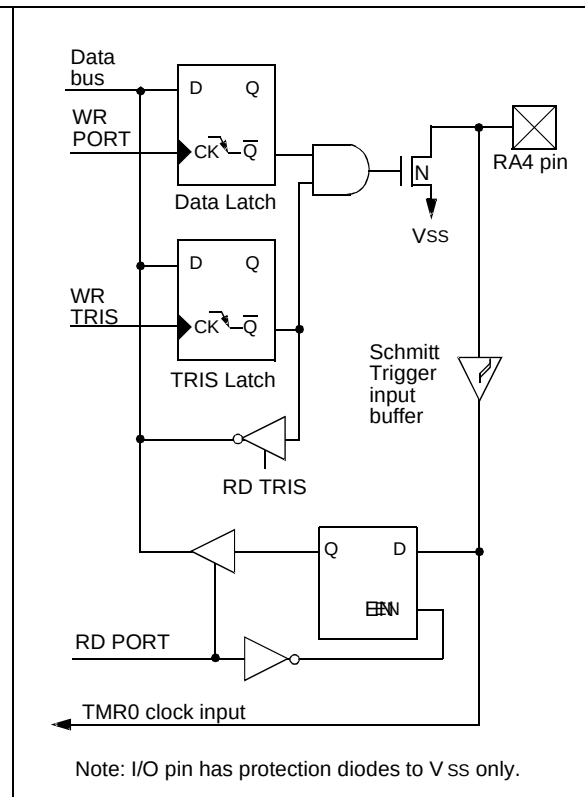
FIGURE 5-1: BLOCK DIAGRAM OF PINS RA3:RA0



EXAMPLE 5-1: INITIALIZING PORTA

```
CLRF    PORTA      ; Initialize PORTA by
                  ; setting output
                  ; data latches
BSF     STATUS, RP0 ; Select Bank 1
MOVLW   0x0F        ; Value used to
                  ; initialize data
                  ; direction
MOVWF   TRISA       ; Set RA<3:0> as inputs
                  ; RA4 as outputs
                  ; TRISA<7:5> are always
                  ; read as '0'.
```

FIGURE 5-2: BLOCK DIAGRAM OF PIN RA4



5.2 PORTB and TRISB Registers

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. A '1' on any bit in the TRISB register puts the corresponding output driver in a hi-impedance mode. A '0' on any bit in the TRISB register puts the contents of the output latch on the selected pin(s).

Each of the PORTB pins have a weak internal pull-up. A single control bit can turn on all the pull-ups. This is done by clearing the $\overline{\text{RBP}}\text{U}$ (OPTION_REG<7>) bit. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The pins value in input mode are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of the pins are OR'ed together to generate the RB port change interrupt.

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

- Read (or write) PORTB. This will end the mismatch condition.
- Clear flag bit RBIF.

A mismatch condition will continue to set the RBIF bit. Reading PORTB will end the mismatch condition, and allow the RBIF bit to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on these four pins allow easy interface to a key pad and make it possible for wake-up on key-depression (see AN552 in the Embedded Control Handbook).

Note 1: For a change on the I/O pin to be recognized, the pulse width must be at least T_{CY} ($4/f_{OSC}$) wide.

The interrupt on change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

FIGURE 5-3: BLOCK DIAGRAM OF PINS RB7:RB4

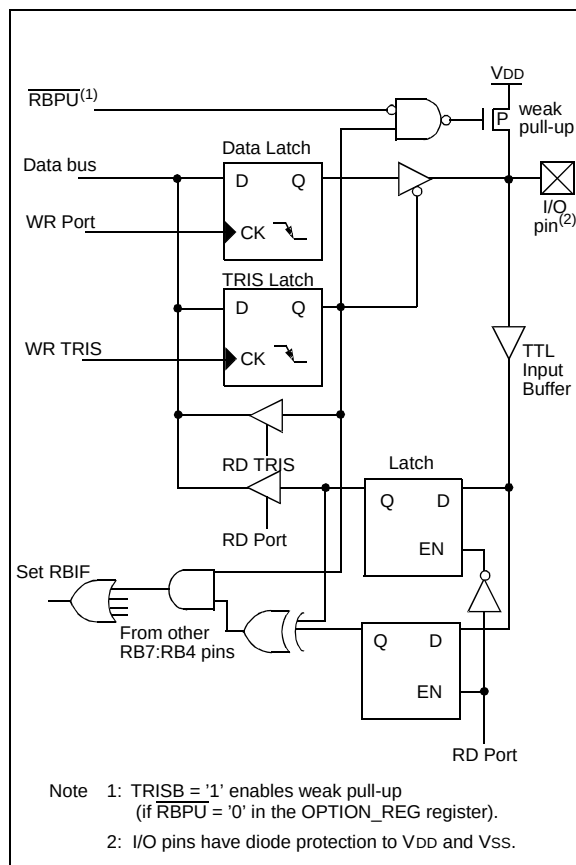
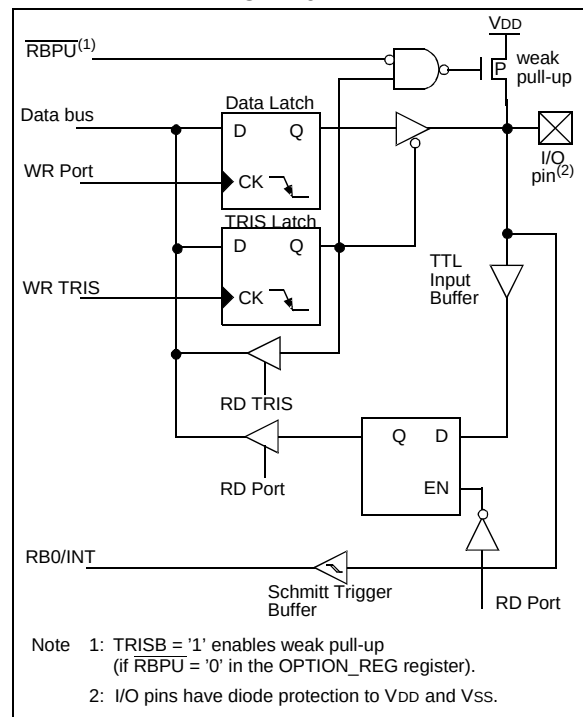


FIGURE 5-4: BLOCK DIAGRAM OF PINS RB3:RB0



PIC16F8X

EXAMPLE 5-1: INITIALIZING PORTB

```
CLRF    PORTB        ; Initialize PORTB by
                        ; setting output
                        ; data latches
BSF     STATUS, RP0   ; Select Bank 1
MOVLW   0xCF          ; Value used to
                        ; initialize data
                        ; direction
MOVWF   TRISB         ; Set RB<3:0> as inputs
                        ; RB<5:4> as outputs
                        ; RB<7:6> as inputs
```

TABLE 5-3 PORTB FUNCTIONS

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.
RB4	bit4	TTL	Input/output pin (with interrupt on change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output pin (with interrupt on change). Internal software programmable weak pull-up.
RB6	bit6	TTL/ST ⁽²⁾	Input/output pin (with interrupt on change). Internal software programmable weak pull-up. Serial programming clock.
RB7	bit7	TTL/ST ⁽²⁾	Input/output pin (with interrupt on change). Internal software programmable weak pull-up. Serial programming data.

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

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.

TABLE 5-4 SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

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
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION_REG	$\overline{\text{RBP}}\text{U}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

8.9 Interrupts

The PIC16F8X has 4 sources of interrupt:

- External interrupt RB0/INT pin
- TMR0 overflow interrupt
- PORTB change interrupts (pins RB7:RB4)
- Data EEPROM write complete interrupt

The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also contains the individual and global interrupt enable bits.

The global interrupt enable bit, GIE (INTCON<7>) enables (if set) all un-masked interrupts or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in INTCON register. Bit GIE is cleared on reset.

The “return from interrupt” instruction, `RETFIE`, exits interrupt routine as well as sets the GIE bit, which re-enable interrupts.

The RB0/INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flags are contained in the INTCON register.

When an interrupt is responded to; the GIE bit is cleared to disable any further interrupt, the return address is pushed onto the stack and the PC is loaded with 0004h. For external interrupt events, such as the RB0/INT pin or PORTB change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency depends when the interrupt event occurs (Figure 8-17). The latency is the same for both one and two cycle instructions. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid infinite interrupt requests.

Note 1: Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

FIGURE 8-16: INTERRUPT LOGIC

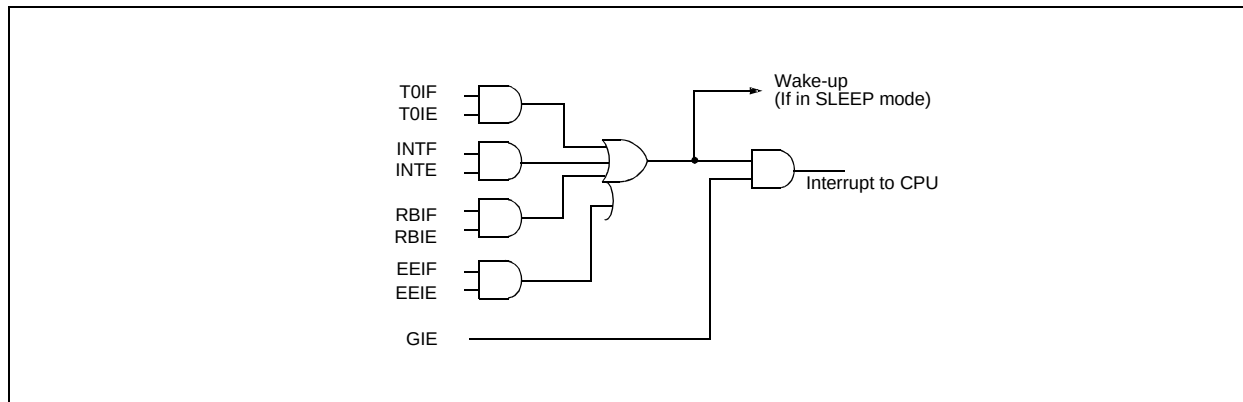
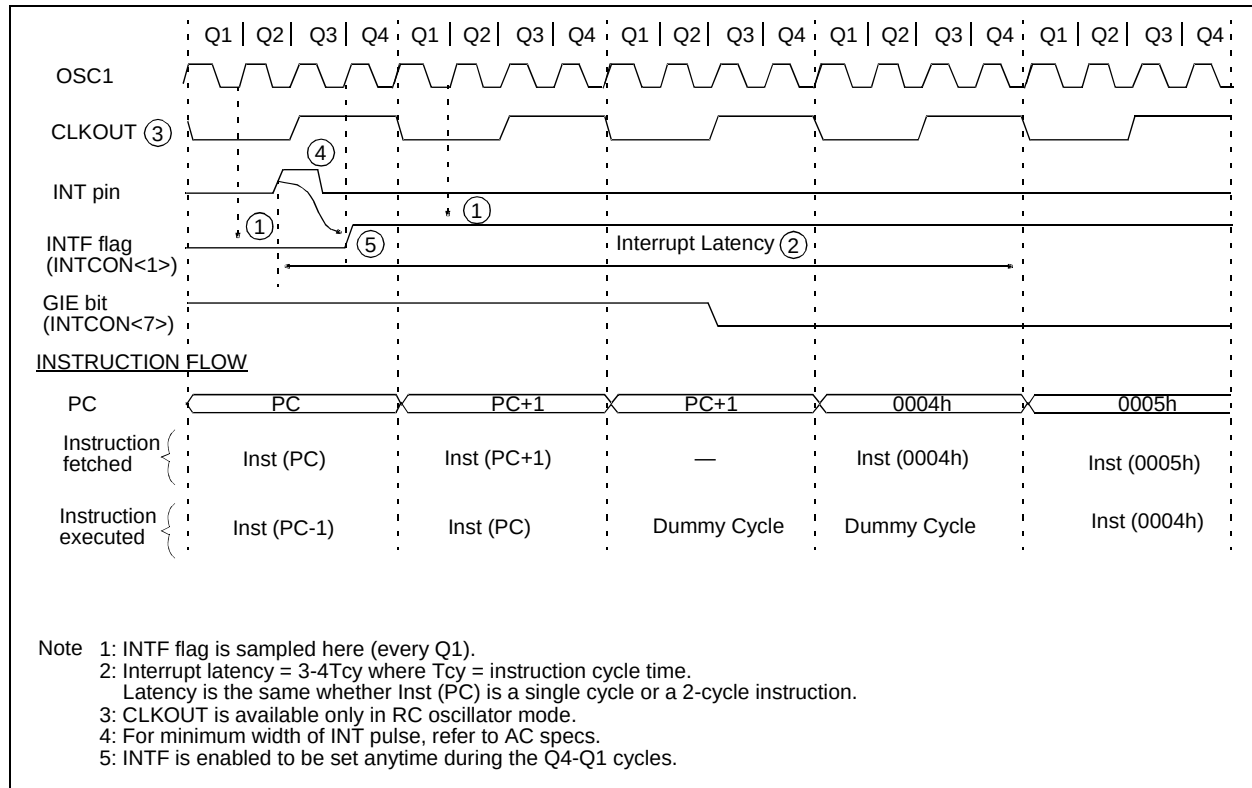


FIGURE 8-17: INT PIN INTERRUPT TIMING



8.9.1 INT INTERRUPT

External interrupt on RB0/INT pin is edge triggered: either rising if INTEDG bit (OPTION_REG<6>) is set, or falling, if INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, the INTF bit (INTCON<1>) is set. This interrupt can be disabled by clearing control bit INTE (INTCON<4>). Flag bit INTF must be cleared in software via the interrupt service routine before re-enabling this interrupt. The INT interrupt can wake the processor from SLEEP (Section 8.12) only if the INTE bit was set prior to going into SLEEP. The status of the GIE bit decides whether the processor branches to the interrupt vector following wake-up.

8.9.2 TMR0 INTERRUPT

An overflow (FFh → 00h) in TMR0 will set flag bit T0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>) (Section 6.0).

8.9.3 PORT RB INTERRUPT

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit RBIE (INTCON<3>) (Section 5.2).

Note 1: For a change on the I/O pin to be recognized, the pulse width must be at least Tcy wide.

9.0 INSTRUCTION SET SUMMARY

Each PIC16CXX instruction is a 14-bit word divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The PIC16CXX instruction set summary in Table 9-2 lists **byte-oriented**, **bit-oriented**, and **literal and control** operations. Table 9-1 shows the opcode field descriptions.

For **byte-oriented** instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven bit constant or literal value.

TABLE 9-1 OPCODE FIELD DESCRIPTIONS

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
TO	Time-out bit
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)

The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 μ s. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 μ s.

Table 9-2 lists the instructions recognized by the MPASM assembler.

Figure 9-1 shows the general formats that the instructions can have.

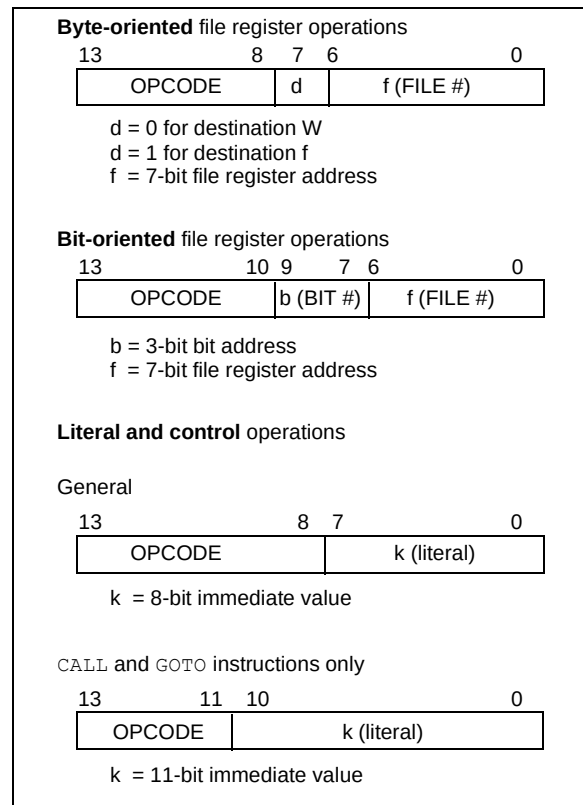
Note: To maintain upward compatibility with future PIC16CXX products, do not use the **OPTION** and **TRIS** instructions.

All examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

FIGURE 9-1: GENERAL FORMAT FOR INSTRUCTIONS



dings verändert auch der Befehl MOVF f,d dieses Flag. Mehr dazu unter Kapitel 4.1.

Mnemonic		Beschreibung	Befehls- takte	Flag
Byteorientierte Befehle				
ADDWF	f,d	Addiert W und Inhalt von f	1	C, DC, Z
ANDWF	f,d	UND-Verknüpfung von W und Inhalt von f	1	Z
CLRF	f	löscht den Inhalt von f	1	Z
CLRW		löscht das W-Register	1	Z
COMF	f,d	Bildet Komplement vom Inhalt von f	1	Z
DECF	f,d	Dekrementiert den Inhalt von f	1	Z
DECFSZ	f,d	Dek. f und überspringt bei 0 den nächsten Befehl	1 (2)	
INCF	f,d	Inkrementiert den Inhalt von f	1	Z
INCFSZ	f,d	Ink. f und überspringt bei 0 den nächsten Befehl	1 (2)	
IORWF	f,d	ODER-Verknüpfung von W und Inhalt von f	1	Z
MOVF	f,d	Holt den Wert aus Adresse f	1	Z
MOVWF	f	Schreibt Inhalt von W nach Adresse f	1	
NOP		Leerbefehl	1	
RLF	f,d	Rotiert den Inhalt von f nach links durchs Carry	1	C
RRF	f,d	Rotiert den Inhalt von f nach rechts durchs Carry	1	C
SUBWF	f,d	Subtrahiert W vom Inhalt der Adresse f	1	C, DC, Z
SWAPF	f,d	Vertauscht die beiden Halbbytes in Adresse f	1	
XORWF	f,d	EXCLUSIV-ODER von W und f	1	Z
Bitorientierte Befehle				
BCF	f,b	Löscht Bit b in Adresse f	1	
BSF	f,b	Setzt Bit b in Adresse f	1	
BTFSC	f,b	Testet Bit b an Adresse f und springt, wenn es 0 ist	1 (2)	
BTFSS	f,b	Testet Bit b an Adresse f und springt, wenn es 1 ist	1 (2)	
Literal und Steuerbefehle				
ADDLW	k	Addiert Literal k zum W-Register	1	C, DC, Z
ANDLW	k	Literal k wird UND verknüpft mit dem W-Register	1	Z
CALL	k	Unterprogrammaufruf an Adresse k	2	
CLRWDT		löscht den Watchdog	1	\overline{TO} , \overline{PD}
GOTO	k	Sprung zur Adresse k	2	
IORLW	k	Literal wird mit dem W-Register ODER verknüpft	1	\overline{Z}
MOVLW	k	Lädt das Literal ins W-Registers	1	
RETFIE		Rückkehr aus der Interruptroutine, setzt GIE	2	
RETLW	k	Rückkehr aus einem Unterprogramm, lädt k in W	2	
RETURN		Rückkehr aus einem Unterprogramm	2	
SLEEP		Geht in den Stand By Modus	1	\overline{TO} , \overline{PD}
SUBLW	k	Subtrahiert W vom Literal k	1	C, DC, Z
XORLW	k	EXCLUSIV ODER Verknüpfung von W und k	1	Z

Tabelle 1: Befehlsübersicht

In der zweitletzten Spalte werden die benötigten Befehlstakte angegeben. Übli-

*****PROGRAMME*****

16-Bit-Zähler + Addierer

;16-Bit_Increment -> 2Variablen

;WertL & WerthH

incf WertL

btfsc status,zero ;Wenn 1 dann nicht springen

incf WerthH

;16-Bit_Addition

movf Wert1L,w

addwf Wert2L ;Addiere die beiden unteren 8 Bit

btfsc status,carry ;Wenn es Überlauf gab, dann Höheres Byte +1

incf Wert1H,w

addwf Wert2H ;Wieder zusammenaddieren

*****Binaercodierer_ohne_Interrupt*****

;*****

; binarycounter.src

;*****

device 16f84

;Symbole definieren

status equ 3

zero equ 2

rp0 equ 5 ;bitadresse

trisa equ 5 ;fileadresse

trisb equ 6

porta equ 5

portb equ 6

pcl equ 2

wert equ 12 ;entspricht 0ch

alterwert equ 13

counter equ 14

org 0

;erster Befehl an Adresse 0, alle weiteren hinten dran

;Einsprung beim Einschalten (Power on)

cold

bsf status,rp0 ;auf Bank 1 umschalten

movlw 0

movwf trisb ;PortB wird komplett als Ausgang

geschaltet

bcf trisa,3 ;RA3 wird Ausgang

(Carry)

bcf status,rp0 ;zurück auf Bank 0

;definieren von alterwert mit aktuellem wert an RA0

	movf	porta,w	;Porta lesen
	andlw	0000001b	
	movwf	alterwert	
;Hauptschleife			
loop	clrf	counter	;Reset und Startwert
	clrf	portb	
loop1			
;reset aktiv?	btfss	porta,1	;Reseteingang (0=Reset)
	goto	loop	
;inhibit aktiv?			
	btfsc	porta,2	;Inhibit-Eingang
	goto	loop1	
;Takteingang lesen			
	movf	porta,w	;Porta komplett einlesen
	andlw	1	;nur RA0 ist von
Interesse			
	xorwf	alterwert,w	;wenn beide gleich, keine Flanke
	btfsc	status,zero	;beide gleich, Zero gesetzt
	goto	loop1	;nichts passiert
	movlw	1	
	xorwf	alterwert	;beinhaltet neuen Pegel an RA0
	btfss	alterwert,0	
	goto	loop1	
;richtige Flanke gefunden			
	bcf	porta,3	;Carryausgang
zurücksetzen			
	incf	counter	;Zähler erhöhen
	movlw	0fh	
	andwf	counter,w	
	xorlw	10	;Vergleich untere 4 Bits
mit 10			
	btfss	status,zero	
	goto	ausgabe	
	movlw	6	
	addwf	counter	
	movlw	0a0h	
	xorwf	counter,w	
	btfss	status,zero	
	goto	ausgabe	
	clrf	counter	
	bsf	porta,3	
;ausgabe auf 7-Segmentanzeige			
ausgabe			
	movf	counter,w	;zuerst rechte Stelle anzeigen
	call	convert	;in 7-Segmentanzeige umsetzen
	movwf	portb	
	bcf	porta,4	;Digit einschalten
	bsf	porta,4	;Digit wieder
ausschalten			

```

        swapf    counter,w           ;jetzt linke Stelle anzeigen
        call     convert
        movwf    portb
        bcf      porta,5
        bsf      porta,5
        goto     loop1

;setzt Binärzahl in Bitmuster für 7-Segmentanzeige um.
convert
        andlw    15
        addwf    pcl                 ;w= offset der zum PCL addiert
wird.
        retlw    3fh                 ;0
        retlw    06h                 ;1
        retlw    5bh                 ;2
        retlw    4fh                 ;3
        retlw    66h                 ;4
        retlw    6dh                 ;5
        retlw    7dh                 ;6
        retlw    07h                 ;7
        retlw    7fh                 ;8
        retlw    6fh                 ;9
        retlw    0                   ;ungültig
        retlw    0
        retlw    0
        retlw    0
        retlw    0
        retlw    0

        end

;*****
;*****Interrupts*****
;*****
device    16f84
intcon    equ    0bh
inte      equ    4
intf      equ    1
gie       equ    7
status    equ    3
rp0       equ    5
option    equ    1
intedg    equ    6
counter   equ    0ch
;startadresse für Programm
org       0
cold
        goto    main
        nop
        nop
        nop

isr                                           ;Interrupt
springt immer an Adresse 4
        btfss   intcon,intf                ;War es ein RB0-Signal

```

```

        goto    intend          ;Nein, Fehler
        bcf     intcon,intf     ;Interrupt    RB0 wird
bearbeitet
        incf    counter        ;Zähler wird erhöht
intend
        retfie                  ;Return from Interrupt
enable

main
;Interrupts initialisieren
        bsf     intcon,inte     ;RB0-Interrupt
freischalten
        bcf     intcon,intf     ;Interruptflag
löschen
        bsf     status,rp0      ;auf Bank 1
schalten
        bcf     option,intedg   ;auf Fallende Flanke
prüfen
        bcf     status,rp0      ;zurück auf Bank
0 schalten
        bsf     intcon,gie      ;Globales
Interrupt
enable Bit
loop
        goto    loop;

```

*****Binarycounter with
Interrupt*****

```

;*****
; binarycounter.src
;*****
        device 16f84

;Symbole definieren
status      equ      3
zero        equ      2
rp0         equ      5      ;bitadresse
trisa       equ      5      ;fileadresse
trisb       equ      6
porta       equ      5
portb       equ      6
pcl         equ      2
intcon      equ      0bh
inte        equ      4
intf        equ      1
gie         equ      7
option      equ      1
intedg      equ      6
counter     equ      13h

        org 0
;erster Befehl an Adresse 0, alle weiteren hinten dran

;Einsprung beim Einschalten (Power on)
cold

```

```

                                goto    main
                                nop
                                nop
                                nop
isr                                ;Interrupt
springt immer an Adresse 4
                                btfss   intcon,intf        ;War es ein RB0-Signal
                                goto     intend             ;Nein, Fehler
                                bcf      intcon,intf         ;Interrupt      RB0 wird
bearbeitet
                                incf     counter            ;Zähler wird erhöht
intend
                                retfie                      ;Return from Interrupt
enable

                                bsf      status,rp0         ;auf Bank 1 umschalten
                                movlw    0
                                movwf    trisb             ;PortB wird komplett als Ausgang
geschaltet
                                bcf      trisa,3             ;RA3 wird Ausgang
(Carry)
                                bcf      status,rp0         ;zurück auf Bank 0

main
;Interrupts initialisieren
                                bsf      intcon,inte        ;RB0-Interrupt
freischalten
                                bcf      intcon,intf        ;Interruptflag
löschen
                                bsf      status,rp0         ;auf Bank 1
schalten
                                bcf      option,intedg      ;auf Fallende Flanke
prüfen
                                bcf      status,rp0         ;zurück auf Bank
0 schalten
                                bsf      intcon,gie         ;Globales
Interrupt enable Bit
;Hauptschleife
loop
                                clrf     counter            ;Reset und Startwert
                                clrf     portb
loop1
;reset aktiv?
                                btfss   porta,1            ;Reseteingang (0=Reset)
                                goto     loop
;inhibit aktiv?
                                btfsc   porta,2            ;Inhibit-Eingang
                                goto     loop1
                                xorwf    counter,w
                                btfss   status,zero
                                goto     ausgabe

```

```

        clrf    counter
        bsf     porta,3

;ausgabe auf 7-Segmentanzeige
ausgabe
        movf    counter,w           ;zuerst rechte Stelle anzeigen
        call    convert             ;in 7-Segmentanzeige umsetzen
        movwf   portb
        bcf     porta,4             ;Digit einschalten
        bsf     porta,4             ;Digit wieder
ausschalten
        swapf   counter,w           ;jetzt linke Stelle anzeigen
        call    convert
        movwf   portb
        bcf     porta,5
        bsf     porta,5
        goto    loop1

```

;setzt Binärzahl in Bitmuster für 7-Segmentanzeige um.

```

convert
        andlw   15
        addwf   pcl                 ;w= offset der zum PCL addiert
wird.
        retlw   01111110b           ;0
        retlw   00001100b           ;1
        retlw   10110110b           ;2
        retlw   10011110b           ;3
        retlw   11001100b           ;4
        retlw   11011010b           ;5
        retlw   11111010b           ;6
        retlw   00001110b           ;7
        retlw   11111110b           ;8
        retlw   11011110b           ;9
        retlw   0                   ;ungültig
        retlw   0
        retlw   0
        retlw   0
        retlw   0
        retlw   0

```

end

*****Wuerfel*****

```

CLRF    counter           ;Counter auf 0
bsf     status,rp0        ;Auf Bank 1 setzen
movlw   0
movwf   trisb             ;PortB auf Output
bcf     trisa,1           ;PortA -> RA1 -> 0
bcf     status,rp0        ;wieder auf Bank 1
goto    HP                ;Sprung in Hauptprogramm

```

HP

```

movlw   1
movwf   counter

```



```

loop
    btfsc    porta,0
    goto     Ausgabe
    incf     counter
    movlw    7
    xorwf    counter,w        ;->speichern in w, wenn 7 -> Dann zeroflag = 1
    btfsc    status, zero
    goto     HP

```

```

Ausgabe
    movf     counter, w
    call     tabelle
    movwf    Portb
    movf     counter, w
    call     tabelle2
    movwf    PortA
    call     HP

```

```

tabelle
    addwf    pcl                ;counter wird auf Programmcounter addiert
    nop                                ;bei 0 -> Nop, da nie 0 übertragen wird
    retlw    0                    ;1
    retlw    00010001b            ;2
    retlw    00010001b            ;3
    retlw    01010101b            ;4
    retlw    01010101b            ;5
    retlw    11011101b            ;6

```

```

tabelle2
    addwf    pcl
    nop
    retlw    2
    retlw    0
    retlw    2
    retlw    0
    retlw    2
    retlw    0

```

*****Eingangsimpuls zwischen 1ms und 5ms*****

```

warteauflow
    btfsc    porta,0
    goto     warteauflow        ;startsequenz für impuls

```

```

wartehigh
    btfss    porta,0
    goto     wartehigh

```

```

erhoehe
    incf     dauer                ;Ab jetzt Messen
    btfsc    porta,0              ;4 Takte      (8-Bitzähler zählt bis
512microsek
    goto     erhoehe              ;Ausgangsschleife muss genau gleich lang
sein

```

```

        movlw    8                                ;Für 8x
        movwf    schleife
        bsf      porta,1
L5      movf      dauer,w                          ;+1*
        movwf    counter                          ;+1*
dekrement
        decf     counter                            ;+1 Schleife{
        btfss   status,zero                        ;+1 (2)
        goto    dekrement                          ;+2          }
        decfsz  schleife                            ;+1*
        goto    L5                                ;+2*
        bcf     porta,1                            ;_____ Ausgang auf 0
        goto    wartehigh                          ;5*8 = 40 microsec fehler ( warum 5? ->
*

```

Kegel-Aufgabe

```

start
        bsf     status,rp0                        ;auf Bank1
        movlw   256                               ;ProtB auf Input(alle 1)
        movwf   trisb                             ;06h
        movlw   17                               ;PortA input
        movwf   trisa
        bcf     status, rp0                       ;wieder auf Bank 0
        goto    HP

HP
        call   unterprog
        goto   HP

unterprog
        clrf    counter                            ;counter auf 0
wartehigh
        btfsc   porta,0
        goto    wartehigh
        btfsc   porta,4
        incf    counter
        movlw   8
        movwf   loopcounter

pruefe
        btfsc   portb,0                            ;Schleife 8x
        incf    counter                            ;durchlaufen
        rrf     portb                             ;Bit 0 prüfen
        decfsz  loopcounter                        ;Nach rechts
        goto    pruefe                            ;rotieren
        movf    counter,w
        return

```

*****Lauflicht*****

device 16f84

```

trisa      equ      5          ;fileadresse
trisb      equ      6
porta      equ      5
portb      equ      6
status     equ      3
rp0        equ      5          ;bitadresse
alterwert  equ      13

```

```

org      0

```

```

cold

```

```

bsf      status,rp0          ;auf Bank1
movlw    0
movwf    trisb              ;Portb auf output
movlw    255
movwf    trisa              ;Porta Bit 0 auf Input
bcf      status,rp0          ;wieder auf Bank0

```

```

HP

```

```

bcf      status,0          ;CarryBit auf 0
movlw    1
movwf    alterwert
movlw    128
movwf    portb

```

```

teste

```

```

movf     porta,w
movwf    alterwert
btfss    porta,0          ;Schalter geschlossen, dann überspringe
goto     teste

```

```

movf     porta,w          ;Porta komplett einlesen
andlw    1                ;nur RA0 ist von Interesse
xorwf    alterwert,w      ;wenn beide gleich, keine Flanke
btfsc    status,zero      ;beide gleich, Zero gesetzt
goto     rotiere          ;nichts passiert
goto     HP

```

```

rotiere

```

```

rrf      portb            ;rotiere nach rechts
btfss    status,0
goto     HP

```

```

end

```

```

*****Flankenerkennung*****

```

```

COLD
;Initialisieren der Ports RA und RB

```

```

        BSF STATUS,RP0           ;auf Bank 1 umschalten, dort sind die
Tris-Register
        CLRF TRISB               ;Port RB ist komplett als Ausgang
geschaltet
        BCF TRISA,DIGIT0        ;RA,2 muss Ausgang sein (Ansteuerung der Digits)
        BCF TRISA,DIGIT1        ;gleiches gilt für RA,3
        BCF STATUS,RP0         ;wieder auf Bank 0 zurückschalten
;initialisieren der Variablen
        BSF PORTA,DIGIT0        ;low aktives Digitselekt
        BSF PORTA,DIGIT1        ;dito
        MOVLW 0                 ;auch in entsprechende Arbeitsvariable
übernehmen
        MOVWF DIGITNR           ;aus dieser wird später das aktive Digit
abgeleitet
        MOVF PORTA,W            ;Taster (an Port RA) einlesen und Pegel merken
        ANDLW 2                 ;ungültige Bits ausblenden (= auf 0
setzen)
        MOVWF APEGEL            ;Pegel hier merken
LOOP1
;Zähler auf 0, wird immer angesprungen, wenn Reset-Taster gedrückt ist (= Pegel
0)
        CLRF COUNTER            ;Flankenzähler auf 0 setzen
LOOP2
;Hauptschleife wird immer zyklisch durchlaufen
        CALL DISPLAY            ;Anzeige bedienen, Counter anzeigen
;prüfen, ob Reset-Taster gedrückt ist
        BTFSS PORTA, RESET      ;nicht gedrückt = Pegel 1
        GOTO LOOP1              ;wenn gedrückt (Pegel = 0) dann zurück
zu LOOP1
;prüfe, ob eine Flanke (Pegelwechsel) an Taster erzeugt wurde
        MOVF PORTA,W            ;kompletten Port RA einlesen
        ANDLW 2                 ;nur Info des Taster weiterverarbeiten
        MOVWF NPEGEL            ;als neuer Pegel abspeichern
        XORWF APEGEL,W          ;<>0 wenn Pegelwechsel vorliegt
        BTFSC STATUS,ZERO       ;Wenn Erg.=0, dann Zero = 1 => kein Pegelwechsel
        GOTO LOOP2
;Flanke wurde entdeckt
        MOVF NPEGEL,W           ;neuer Pegel in APEGEL speichern
        MOVWF APEGEL
;prüfe, um welche Flanke es sich handelt.
        BTFSS NPEGEL,TASTER     ;wenn 1, dann steigende Flanke, bei 0 = fallende
Flanke
        GOTO LOOP2              ;keine Wirkung, da steigende Flanke
(Taster wird losgel.)
;steigende Flanke gefunden
        INCF COUNTER            ;zähle diese steigende Flanke
;hier kann man noch den BCD-Zähler implementieren, ansonsten zählt COUNTER eben
binär
        GOTO LOOP2              ;zurück zur Hauptschleife
;Die Anzeigeroutine ist als Unterprogramm (GOSUB / RETURN) realisiert
DISPLAY
        MOVF COUNTER,W          ;niederwertige Stelle?
        BTFSC DIGITNR,0         ;0 wenn niederwertige Stelle, sonst 1 (low
aktiv)

```

```

        SWAPF COUNTER,W           ;es wird höherwertige Stelle ausgegeben
        CALL CONVERT              ;Wert in W in Bitmuster für 7-Segmente umsetzen
        MOVWF PORTB
        BTFSC DIGITNR,0
        GOTO DISP1
DISP0
        BCF PORTA,DIGIT0
        BSF PORTA,DIGIT0
        GOTO DISP3
DISP1 BCF PORTA,DIGIT1
        BSF PORTA,DIGIT1
DISP3
        INCF DIGITNR
        RETURN
;Konvertierungstabelle, setzt eine Zahl in das entsprechende Bitmuster für
7-Segmente um
CONVERT
        ANDLW 15 ;nur untere 4 Bits gültig, damit 16 Tabelleneinträge
        ADDWF PCL ;W = Offset für Tab.-zugriff (Sprung auf entsp. RETLW)
        RETLW 3FH
        RETLW 06H
        RETLW 5BH
        RETLW 4FH
        RETLW 66H
        RETLW 6DH
        RETLW 7DH
        RETLW 07H
        RETLW 7FH
        RETLW 6FH
        RETLW 77H
        RETLW 7CH
        RETLW 39H
        RETLW 5EH
        RETLW 79H
        RETLW 71H
        RETLW 00

```

*****Lauflicht_blinken*****

```

                device 16f84

trisa          equ            5           ;fileadresse
trisb          equ            6
porta          equ            5
portb          equ            6
status         equ            3
rp0            equ            5           ;bitadresse
dauer          equ            13
counter        equ            12
zero           equ            2

                org            0

```

cold

```
    bsf          status,rp0          ;auf Bank1
    movlw        0
    movwf        trisb                ;Portb auf output
    movlw        255
    movwf        trisa                ;Porta Bit 0 auf Input
    bcf          status,rp0          ;wieder auf Bank0
```

HP

```
    movlw        128
    movwf        dauer
    movlw        128
    movwf        portb
    btfss        porta,0
    goto         HP
```

blinken

```
    movf         dauer,w              ;+1*
    movwf        counter              ;+1*
    movlw        63
    movlw        porta
```

dekrement

```
    bcf          porta
    movlw        64
    xorwf        counter,w
    btfss        status,zero          ;wenn ungleich
    goto         next1
    movlw        31
    movwf        porta
```

next1

```
    movlw        32
    xorwf        counter,w
    btfss        status,zero          ;wenn ungleich
    goto         next2
    movlw        15
    movwf        porta
```

next2

```
    movlw        16
    xorwf        counter,w
    btfss        status,zero          ;wenn ungleich
    goto         next3
    movlw        7
    movwf        porta
```

next3

```
    movlw        8
    xorwf        counter,w
    btfss        status,zero          ;wenn ungleich
    goto         next4
    movlw        3
    movwf        porta
```

next4

```

        movlw    4
        xorwf    counter,w
        btfss    status,zero        ;wenn ungleich
        goto     next5
        movlw    1
        movwf    porta

```

```

next5
        decf     counter            ;+1 Schleife{
        btfss    status,zero        ;+1 (2)
        goto     dekrement          ;+2          }
        decfsz   schleife           ;+1*
        goto     blinken            ;+2*
        goto     HP

```

end

;Einfacher-> Wir zählen einen Counter bis 63 und geben ihn direkt aus
cold

```

        bsf      status,rp0        ;auf Bank1
        movlw    0
        movwf    trisb              ;Portb auf output
        movlw    255
        movwf    trisa              ;Porta Bit 0 auf Input
        bcf      status,rp0        ;wieder auf Bank0

```

```

HP
        bcf      counter
        movlw    128
        movwf    portb
        btfss    porta,0
        goto     HP

```

```

erhoehe
        btfss    porta,0
        goto     HP
        movf     counter,w
        movwf    portb
        incf     counter            ;Ab jetzt Messen
        btfss    counter,6          ;4 Takte      (8-Bitzähler zählt bis
512microsek
        goto     erhoehe            ;Ausgangsschleife muss genau gleich lang
sein
        goto     hp
End

```

*****Ampel*****

```

;Programm: Ampelsteuerung
device 16F84
;Variablen und Konstantendefinitionen
trisa EQU 5

```

```

porta EQU 5
trisb EQU 6
portb EQU 6
status EQU 3
rp0 EQU 5
zero EQU 2
carry EQU 0
;äußere Zustände = Ampelanzeige
azustand0 EQU 00001001b
azustand1 EQU 00001011b
azustand2 EQU 00001100b
azustand3 EQU 00001010b
azustand4 EQU 00001001b
azustand5 EQU 00011001b
azustand6 EQU 00100001b
azustand7 EQU 00010001b
azustand8 EQU 00001001b
;Festlegen des Programmstarts
ORG 0
cold
state1
BSF status, rp0 ;Initialisierung
CLRF trisrb ;Port B auf Ausgang
BCF status, rp0
state2
MOVLW azustand0 ;äußerer Zustand 0 an Port ausgeben
MOVWF portb
CALL wait30s ;warte 30 Sekunden
state3
MOVLW azustand1 ;rot-gelb für (1) ausgeben
MOVWF portb
CALL wait3s ;warte 3 Sekunden
state4
MOVLW azustand2 ;grün für (1) ausgeben
MOVWF portb
CALL wait120s ;warte 2 Minuten
state5
MOVLW azustand3 ;gelb für (1) ausgeben
MOVWF portb
CALL wait3s
state6
MOVLW azustand4 ;rot für (1) ausgeben
MOVWF portb
CALL wait2s ;gesamte Verkehr ruht 2 Sekunden
state7
MOVLW azustand5 ;rot-gelb für (2) ausgeben.
MOVWF portb
CALL wait3s
state8
MOVLW azustand6 ;grün für (2)
MOVWF portb
CALL wait60s ;Hier nur 1 min Grünphase
state9
MOVLW azustand7 ;gelb an (2)

```



```

MOVWF portb
CALL wait3s
state10
MOVWL azustand8 ;rot an (2) ausgeben
MOVWF portb
state11
BTFSS einschalter ;Anlage noch eingeschaltet
GOTO state13
CALL wait2s
state12 MOVLW azustand1 ;rot-gelb für (1)
MOVWF portb
CALL wait3s
GOTO state4
state13
GOTO state13 ;Ampeln bleiben rot bis Betriebs-
;spannung weggenommen wird.

*****Delay*****
    device 16f84
;Symbole definieren
delaycnt    equ    0ch

delay166
    movlw    52
    movwf    delaycnt
delay166_a
    decfsz    delaycnt
    goto     delay166_a
    return

delay625
    movlw    206
    movwf    delaycnt
delay625_a
    decfsz    delaycnt
    goto     delay625_a
    return

warte_null
    btfss    rb,4            ;Nulldurchgang?
    goto     warte_null
;Nulldurchgang gefunden
    call     delay166
;Schalter einlesen
    movf     portb,w         ;Schalter = 0?
    andlw    15              ;Obere 4 Bits auf 0 setzen
    btfsc    status,zero
    goto     warte_null

    movwf    loopcnt

loop
    call     delay625

```

```
    decfsz    loopcnt  
    goto     loop
```

```
;TRIAC einschalten
```

```
    bsf      rb,5  
    call     delay166  
    bcf      rb,5  
    goto     warte_null
```