Table des matières du chapitre 9

Chapter 9: PIC16F84 Microcontroller

1 Features of the PIC16F84 Microcontroller

The PIC16F84 is an 8-bit RISC architecture microcontroller. It belongs to the PIC family known as the Mid-Line series. There are three PIC families:

- **Base-Line**: Instructions are encoded in 12 bits.
- **Mid-Line**: Instructions are encoded in 14 bits.
- **High-End**: Instructions are encoded in 16 bits.

The PIC16F84 is equipped with:

1. An eight-level stack with a width of 13 bits and multiple internal and external interrupt sources.
2. Two separate buses, an instruction bus (14 bits) and a data bus (8 bits), based on the Harvard architecture.
3. Single-cycle execution for all instructions except branch instructions, which require two cycles.
4. A two-stage instruction pipeline allowing all instructions to execute in a single cycle, except for branch instructions (jumps), which take two cycles.
5. An instruction set comprising 35 instructions (reduced instruction set).
6. A maximum external clock frequency of 10 MHz.
7. An 18-pin Dual In-line Package (DIP) form factor.

Table 9.1: General Characteristics of the PIC16F84

	Feature	**PIC16F84**	**PIC 16CR84**
Memory	**Program memory**	**1K × 14 bits (flash)**	**1K × 14 bits (ROM)**
	Data memory (Byte)	**68 × 8 bits**	**68 × 8 bits**
	Data memory of EEPROM	**64**	**64**
	Stack	**8 × 14 bits**	**8 × 14 bits**
Peripherals	**Timer module**	**Timer 0**	**Timer 0**
Specifications	**Interrupt source**	**4**	**4**
	I/O pins	**13**	**13**
	Operating voltage	**2V ~ 6V**	**2V ~ 6V**
	Total number of pins	**18**	**18**
	Number of I/O ports	**PORT A and PORT B**	**PORT A and PORT B**
Clock	**Clock (max)**	**10 MHz**	**10 MHz**

CR: Refers to a microcontroller where the program memory is of the ROM type.
F: Refers to a microcontroller where the program memory is of the Flash type.

PIC microcontrollers with Flash memory (F) allow the same microcontroller to be used for both prototyping and production. They are reprogrammable, enabling the code to be updated without removing the microcontroller from the electronic board.

2 Internal Architecture of the PIC16F84

The PIC16F84 is based on a RISC architecture and employs the Harvard architecture. The microcontroller features separate address buses for program memory and data memory. This separation allows program and data memory to have different word sizes. Data words are 8 bits wide, while machine instruction codes are 14 bits wide. The program memory data bus is referred to as the **instruction bus**.

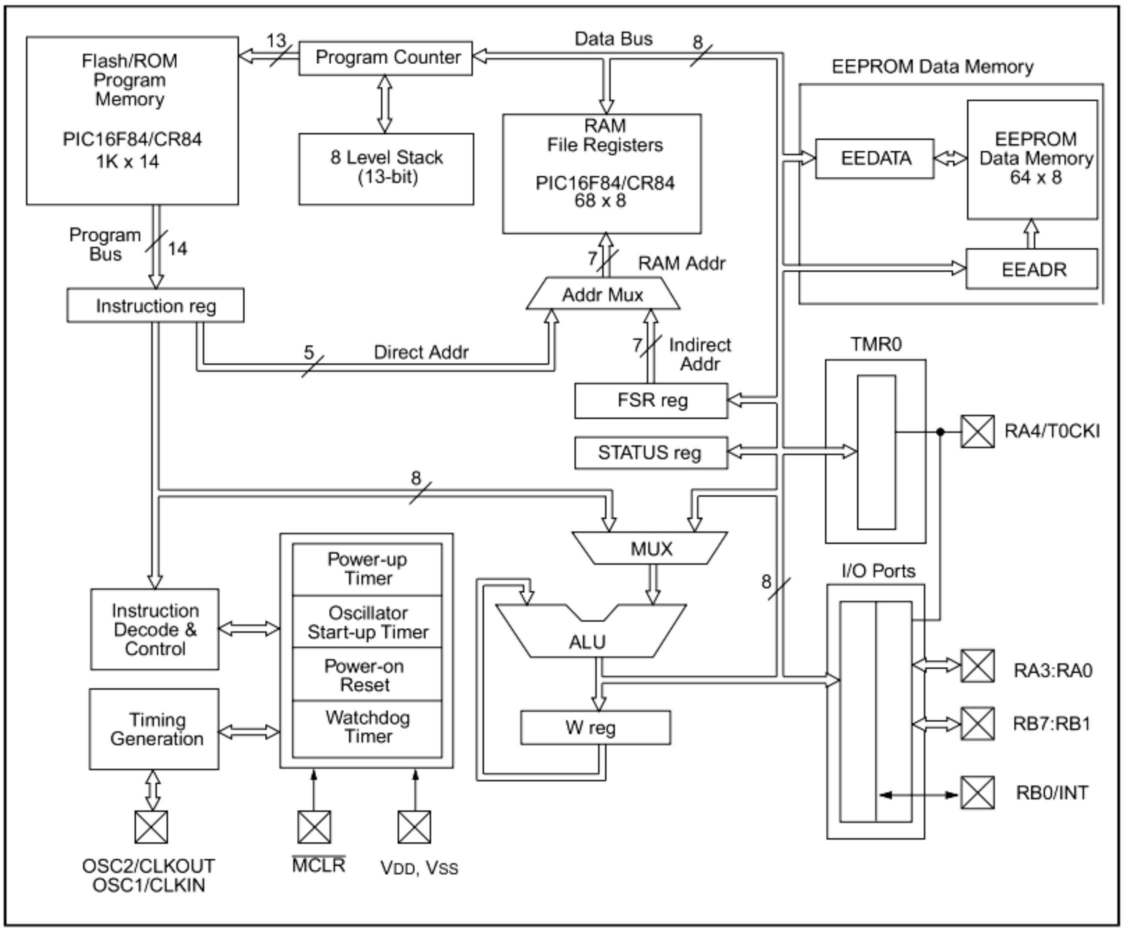
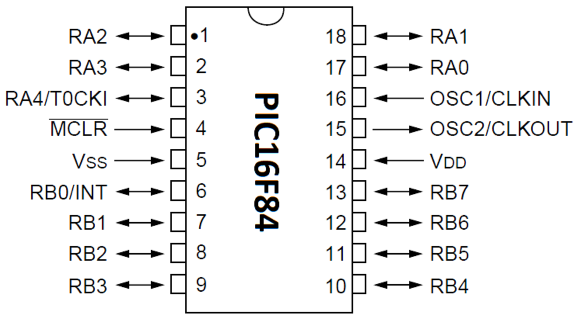


Figure 9.1: Internal Architecture of the PIC16F84 Microcontroller

3 Description of the PIC16F84 Pins



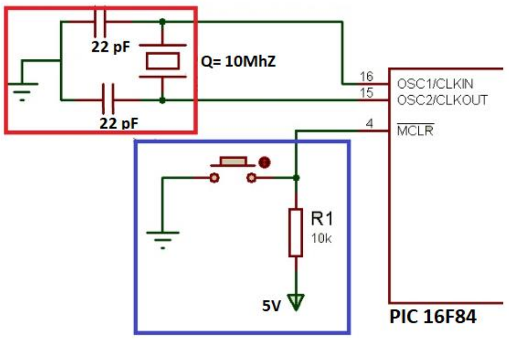


Figure 9.2: Pinout of the PIC16F84

Figure 9.3: Reset and Oscillator Circuits for the PIC

Power Supply Pins

Pin 5 and Pin 14 must be connected to the negative and positive terminals of the power supply, respectively.

Reset Pin

This is Pin 14, referred to as MCLR (Master Clear). It is used to reset the PIC program whenever it is held at a low voltage level (0V).

Clock Pins

Pins 15 and 16 must be connected to the quartz oscillator. The higher the frequency of the quartz used, the faster the processor operates.

Input/Output Ports

The PIC16F84 has 13 GPIO (General Purpose Input/Output) pins distributed across two ports (PORTA and PORTB). These pins can be independently configured as digital inputs or outputs. Additionally, each pin can source or sink a maximum current of 25 mA per pin.
The table below describes all the pins of the PIC16F84.

Table 9.2: Description of the PIC16F84 Pins

Pin Name	Pin Number	Type	Description
OSC1/CLKIN	16	Input	External clock source input.
OSC2/CLKOUT	15	Output	External clock source output.
MCLR	4	Input	(Master Clear) Reset input, active when at a low level.
VSS	5	Power	Ground reference (0V).
VDD	14	Power	Positive power supply.
PORT A			
RA0	17	I/O	Bidirectional: Can be configured as input or output.
RA1	18	I/O	Bidirectional.
RA2	1	I/O	Bidirectional.
RA3	2	I/O	Bidirectional.
RA4/T0CKI	3	I/O	Bidirectional / Can also be selected as the input clock for the TMR0 timer/counter.
PORT B			
RB0/INT	6	I/O	Bidirectional / Can also be selected as the external interrupt pin.
RB1	7	I/O	Bidirectional.
RB2	8	I/O	Bidirectional.
RB3	9	I/O	Bidirectional.
RB4	10	I/O	Bidirectional / Interrupt on pin change.
RB5	11	I/O	Bidirectional / Interrupt on pin change.
RB6	12	I/O	Bidirectional / Interrupt on pin change.
RB7	13	I/O	Bidirectional / Interrupt on pin change.

Designation: **E**: Input, **S**: Output, **A**: Power Supply

The pins of Port A and Port B are bidirectional.

4 Instruction Execution Cycle

Each instruction cycle (Tcy) consists of four Q cycles (Q1-Q4). The Q cycle corresponds to the microcontroller's oscillator cycle (TOSC). The following diagram illustrates the relationship between the Q cycles and the instruction cycle.
The four Q cycles that make up an instruction cycle (Tcy) can be summarized as follows:

1. **Q1**: Instruction fetch cycle.
2. **Q2**: Instruction decode cycle.
3. **Q3**: Data processing cycle.
4. **Q4**: Result write cycle.

Each instruction details the specific Q cycle operations for that instruction.

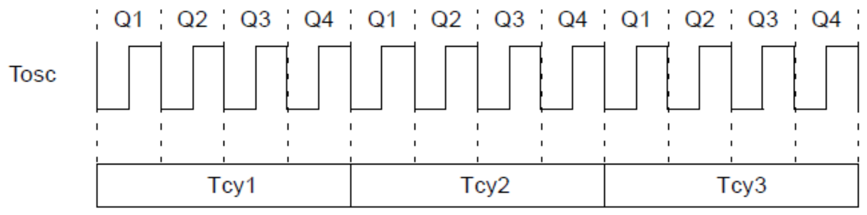


Figure 9.4: Instruction Execution Cycle

PC: The Program Counter, also known as the instruction pointer.

Cycle Time

The PIC microcontroller divides the external clock frequency by 4 to obtain the instruction cycle frequency.

$F_{cycle} = \frac{F_{osc}}{4}$

$$T_{cycle} = T_{osc} \times 4$$

T_{osc}: Oscillator Period (External Clock)

If the oscillator frequency is 10 MHz, its period is calculated as **$1/10^7$ s = 0,1 µs = 100 ns.**

The cycle time is = **100ns** × **4** = **400ns**

If the clock frequency is 4 MHz, its period is **$1/4{\cdot}10^6$ s = 250 ns**, and the cycle time is 1000ns = 1µs.

Note: All instructions execute in a single cycle, except for branch instructions.

Program Memory and Stack Organization

The PIC16FXX has a 13-bit program counter capable of addressing a program memory space of 8K x 14. For the PIC16F84, The first 1K x 14 (0000h-03FFh). The reset vector is located at 0000h and the interrupt vector is at 0004h.

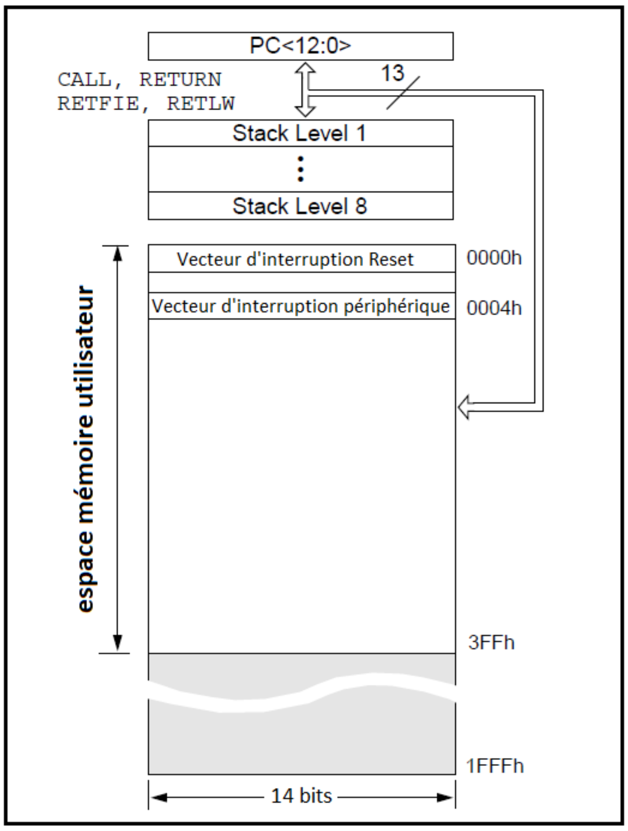

PC<12:0>
CALL, RETURN
RETFIE, RETLW
13
Stack Level 1
Stack Level 8
Vecteur d'interruption Reset
0000h
Vecteur d'interruption périphérique
0004h
espace mémoire utilisateur
3FFh
1FFFh
14 bits

Figure 9.5: Organization of Program Memory and the Stack

Program Counter (Instruction Pointer)

The program counter (PC) is 13 bits wide. The low byte of the PC is the **PCL register**, which is both readable and writable. The high byte of the PC (**PC<12:8>**) is not directly readable or writable and is derived from the **PCLATH register**.

The **PCLATH register** (PC latch high) acts as a holding register for **PC<12:8>**. The contents of PCLATH are transferred to the high byte of the program counter whenever the PC is loaded with a new value. This occurs during a **CALL**, **GOTO**, or a write to the PCL register. The high bits of the PC are loaded from PCLATH, as illustrated in the figure below.

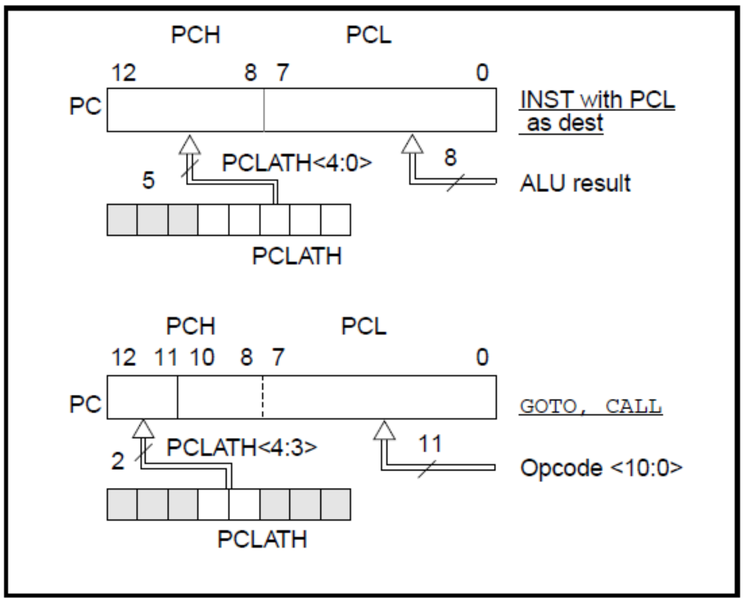


Figure 9.6: Loading the Program Counter (PC) in Different Situations

Data Memory Organization

The data memory is divided into two areas (see figure below). The first area consists of **Special Function Registers (SFRs)**, which control the microcontroller's operation. The second area consists of **General Purpose Registers (GPRs)**, used for general storage.
The data memory is organized into two banks containing both the general-purpose registers and the special function registers. Bank selection requires the use of control bits located in the **STATUS register**.
Bank selection is managed using the **PR0** and **PR1** bits in the **STATUS register**.

- **Bank 0** is selected by setting **PR0 = 0** and **PR1 = 0** in the STATUS register.
- **Bank 1** is selected by setting **PR0 = 1** and **PR1 = 0** in the STATUS register.

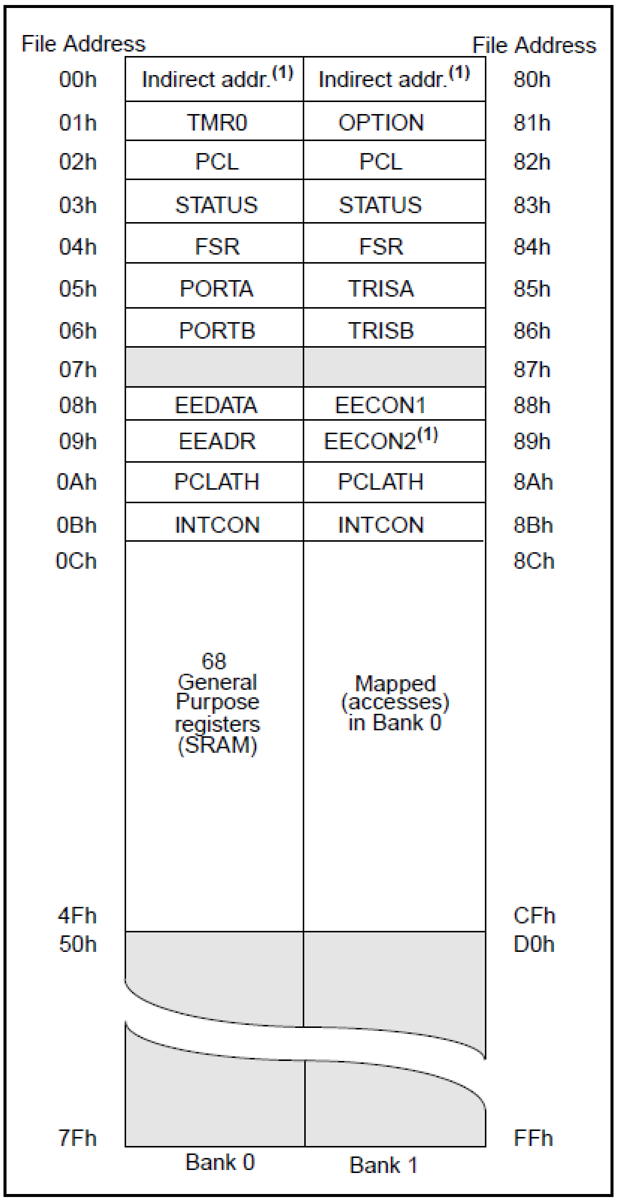
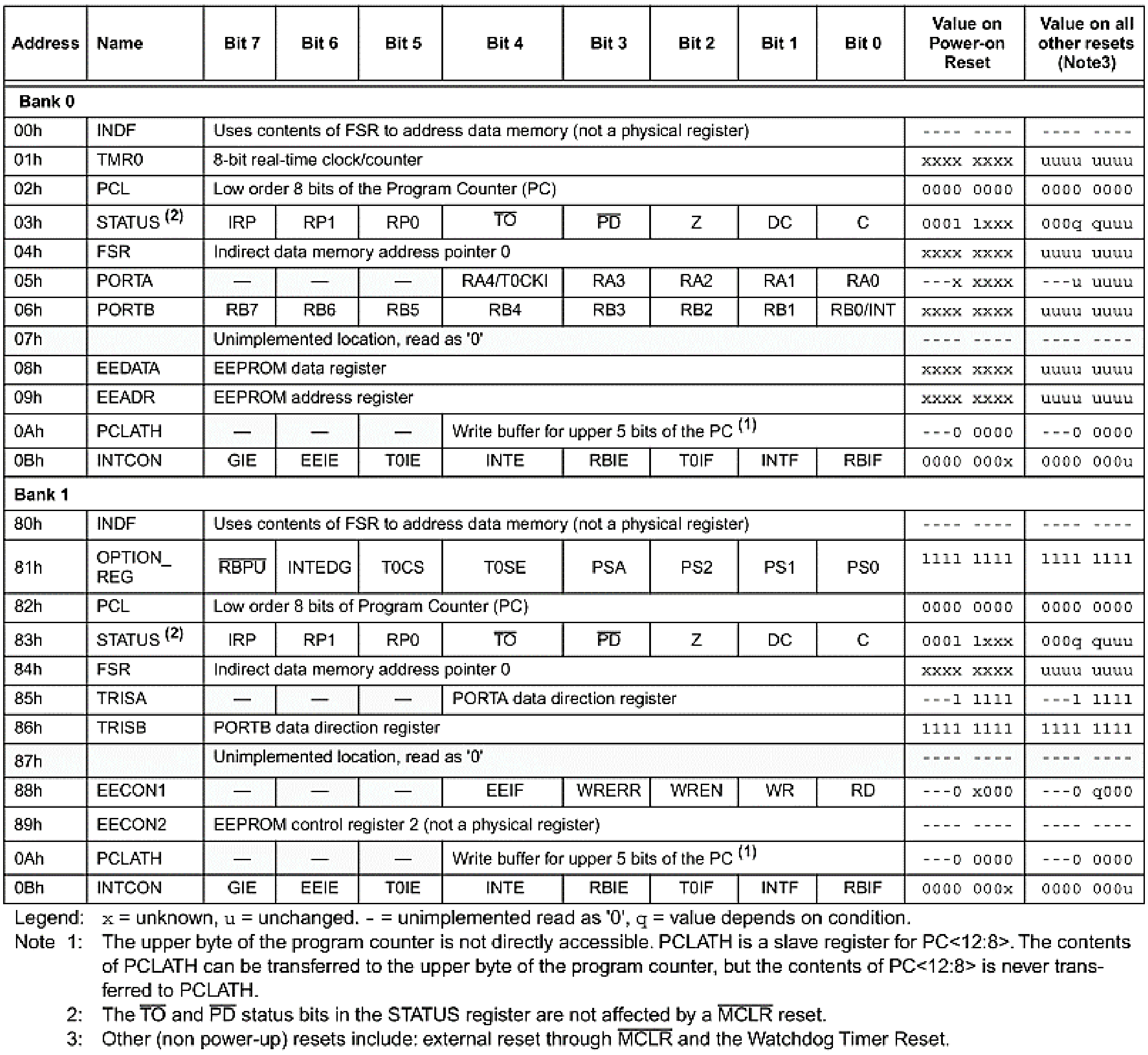


Figure 9.7. Organization of Program and Data Memory

▢ : Unused memory area, read as '0'.
(1) : The indirect address does not physically exist.

4.1.1 Overview of Memory Registers in the PIC16F84



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	T0IE	INTE	RBIE	T0IF	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)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	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.

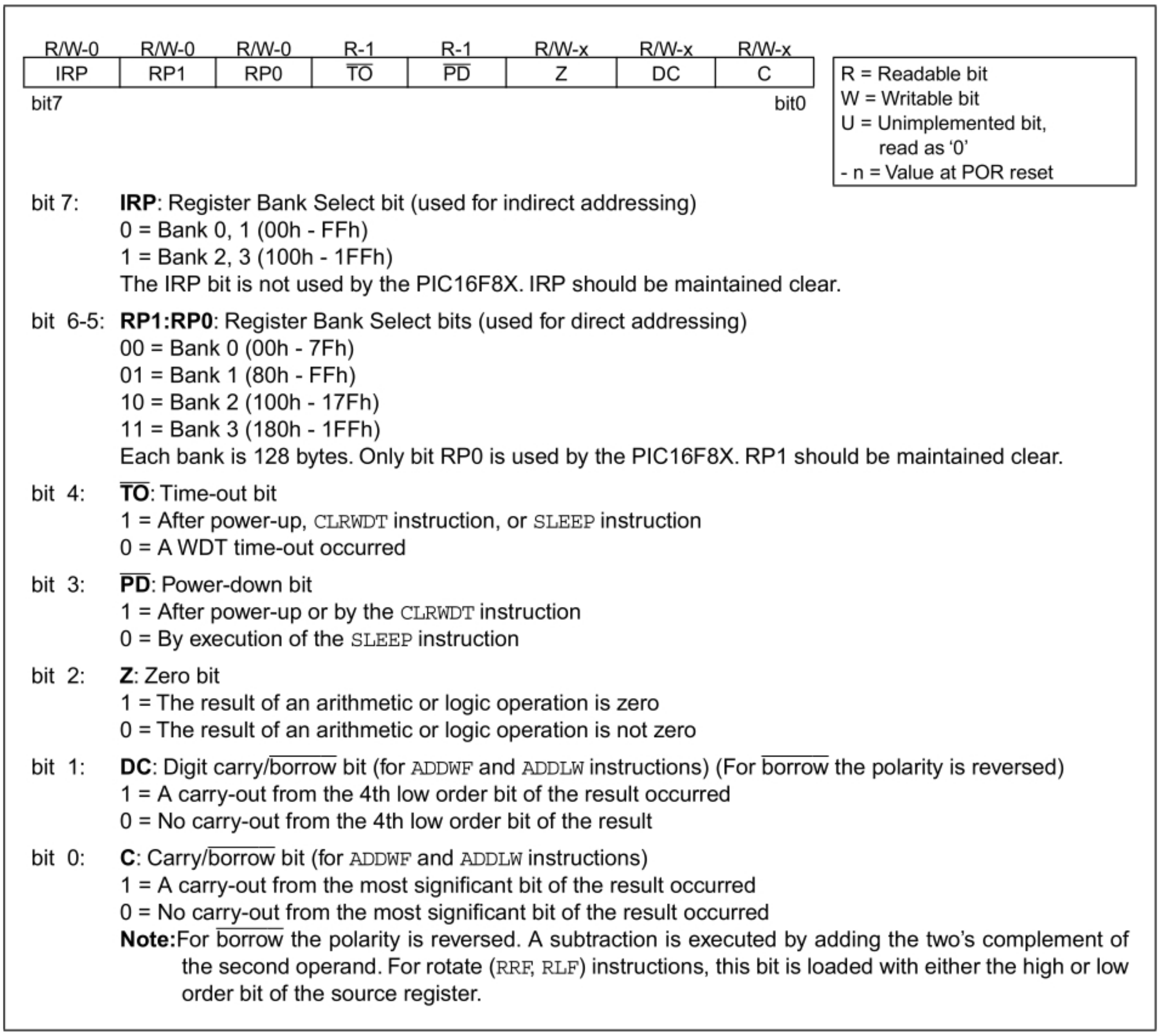
2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

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

4.1.2 Status Register (STATUS) (ADDRESS 03h, 83h)

The status register, known as **STATUS**, located at address **03h** in **bank 0**, contains the status of certain operations performed by the processor (C, DC, and Z). Additionally, the **TO** and **PD** bits are read-only. The **PR0** and **PR1** bits are used to select data banks in the data memory.

The **STATUS** register has a copy in **bank 1** at address **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

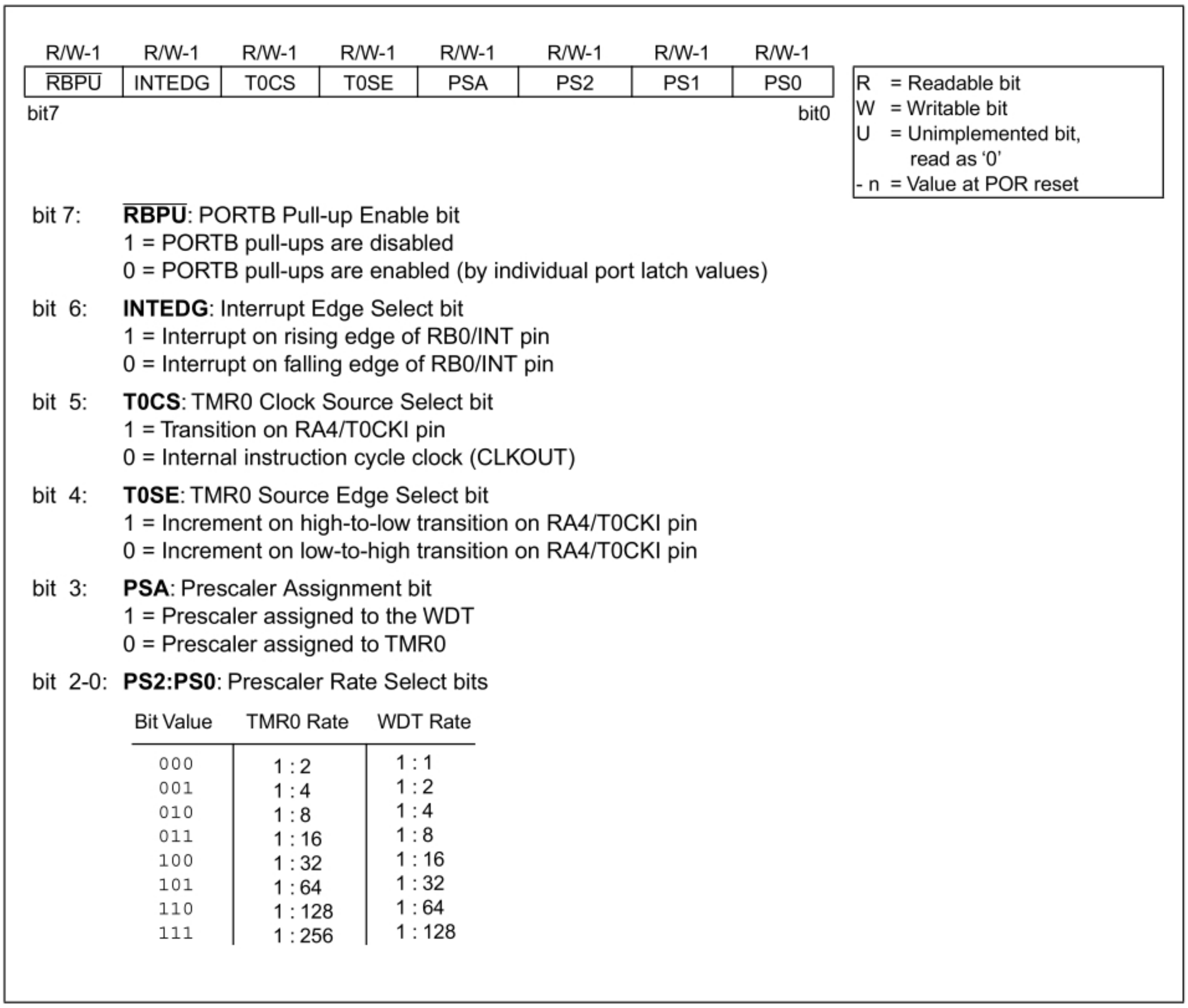
bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

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

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

4.1.3 OPTION_REG REGISTER (ADDRESS 81h)

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.



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

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

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

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

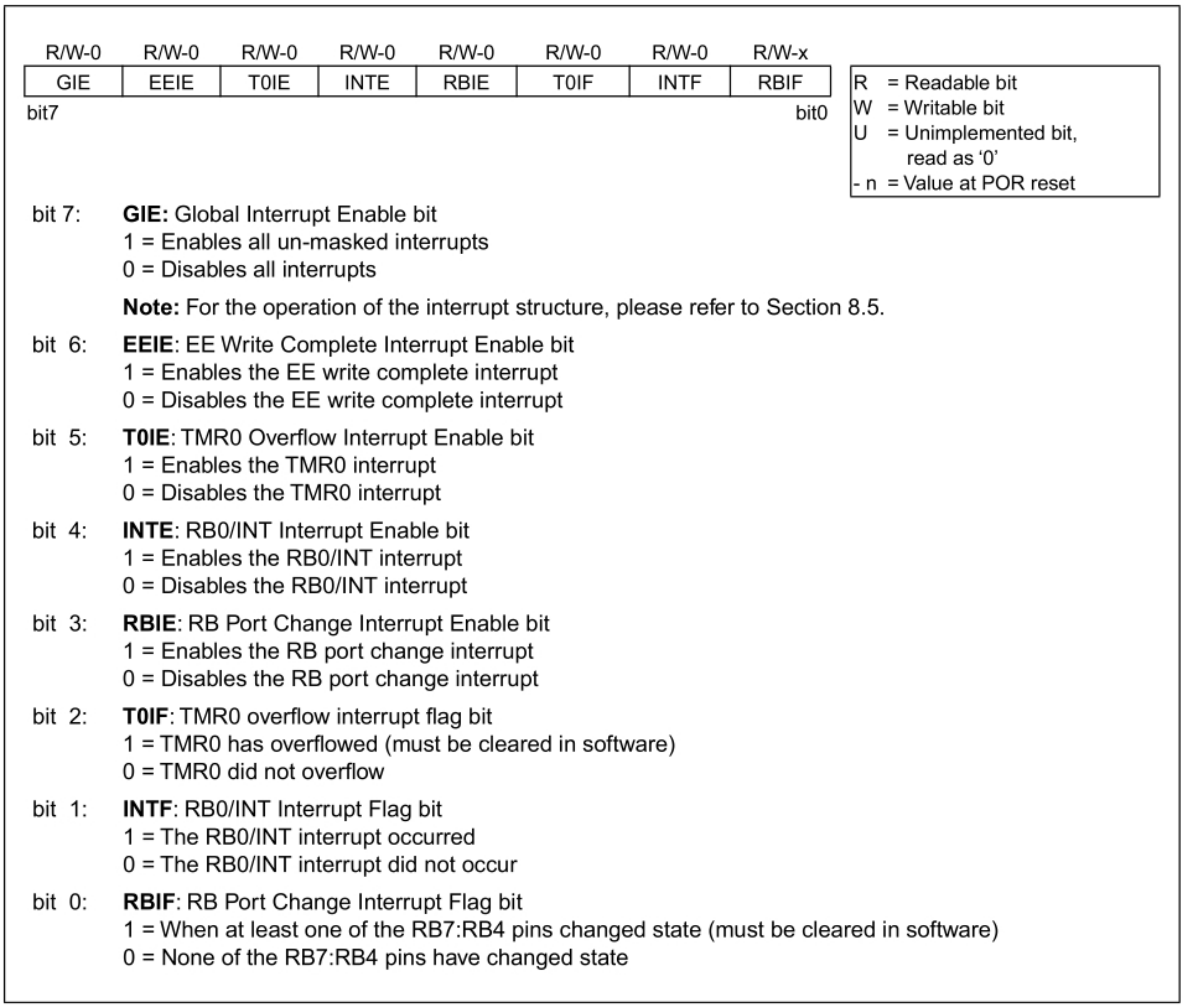
bit 3: **PSA**: Prescaler Assignment bit
1 = Prescaler assigned to the WDT
0 = Prescaler assigned to TMR0

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

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

4.1.4 INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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



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

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

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

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

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

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

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

5 Addressing Modes

Data transfer instructions allow the following operations:

- Move an **immediate value (L)** to the accumulator (W), using the instruction MOVLW.
- Move the **contents of the accumulator (W)** to a memory location, using the instruction MOVWF.
- Move a **memory location value** to the accumulator (W), using the instruction MOVFW.

The entire data memory is accessible either **directly**, using the direct address of each memory register, or **indirectly**, via the File Select Register (FSR). Indirect addressing uses the current values of the **RP1:RP0** bits in the status register to access the banked areas of data memory.

5.1 Immediate Addressing

The operand appears directly in the instruction and is a constant value (also referred to as a **Literal** in English). Example: MOVLW 0XC4 ; This instruction loads the accumulator W (working register) with the hexadecimal value C4H.

Note: The PIC16F84 microcontroller only handles 8-bit data.

5.2 Direct Addressing

In direct addressing, the desired memory bank (bank 0 or bank 1) must first be selected, followed by specifying the memory address for the instruction to execute.
Example:

```
BCF    STATUS, 5    ; Select bank 0 (RP1 = 0 and RP0 = 0)
MOVF   0X6, 0       ; Load the content of memory register at address 6H into W.
```

In this case, address 06H corresponds to the **PORTB** register in the data memory.

5.3 Indirect Addressing

The **INDF** register is not a physical register. It contains the content of the memory location pointed to by the **FSR** register. The **FSR** register is a memory pointer that holds the address of the memory register.
Example:
The memory register 0CH contains the value 22H, and the memory register 0DH contains the value 33H. Both belong to **bank 0**.

```
BCF    STATUS, 5    ; Select bank 0
BCF    STATUS, 6
MOVLW   0X0C        ; Initialize the pointer
MOVWF   FSR         ; FSR is the memory pointer, now contains 0CH
MOVF    INDF, 0     ; Transfer the content of memory register at 0CH into W (W=22H)
INCF    FSR         ; Increment FSR: FSR ← FSR + 1, so FSR = 0DH
MOVF    INDF, 0     ; Transfer the content of memory register at 0DH into W (W=33H)
```

6 Input/Output Ports

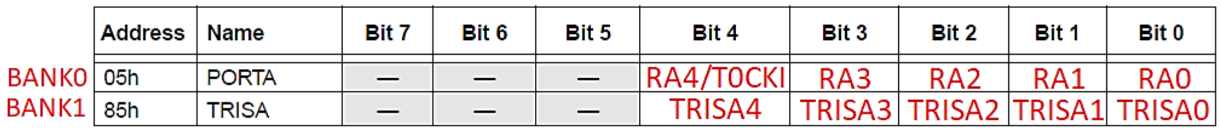
The PIC16F84 has two input/output ports: **PORTA** and **PORTB**, referred to as PortA and PortB. Output values on the ports are stored in memory, while values read from the ports are not stored.

Setting a pin **n** of the PORT to '1' in output mode corresponds to a voltage of 5V on pin **n**.

6.1 PORTA

PORTA has 5 bidirectional input/output pins. The **PORTA register**, located at address **05h** in bank 0, can be configured for input (read) or output (write). Each bit in this register corresponds to one pin.

Table 9.3: Overview of Registers Associated with PORTA Operations



	Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BANK0	05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0
BANK1	85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

The **TRISA register**, located at address **85h** in bank 1, allows selecting the direction of each pin (input or output):

- Each bit set to **1** in TRISA configures the corresponding pin as an **input**.
- Each bit set to **0** in TRISA configures the corresponding pin as an **output**.

The **RA4 pin** can also serve as a counting input for **timer0**.

Example:
To configure all PORTA pins as inputs, load the value FFH into the TRISA register:

```
BSF     STATUS, RP0    ; Select Bank 1
MOVLW   0xFF           ; W ← FFH
MOVWF   TRISA          ; TRISA ← W
```

If you want to configure **RA0** and **RA1** as inputs, **RA2** and **RA3** as outputs, and **RA4** as an input:

```
BCF     STATUS, RP0    ; Select Bank 0, as PORTA is located in Bank 0.
CLRF    PORTA          ; Initialize PORTA pins as outputs with '0'.
BSF     STATUS, RP0    ; Select Bank 1 to configure the pin directions.
MOVLW   B'00010011'    ; Configure RA<1:0> as input, RA<3:2> as output, and RA4 as input.
MOVWF   TRISA          ; TRISA ← W
```

6.2 PORTB

PORTB has 8 bidirectional input/output pins. The **PORTB register**, located at address **06h** in bank 0, can be configured for input (read) or output (write). Each bit in this register corresponds to one pin.

Table 9.4: Overview of Registers Associated with PORTB Operations

	Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BANK0	06h	PORTB	**RB7**	**RB6**	**RB5**	**RB4**	**RB3**	**RB2**	**RB1**	**RB0**
BANK1	86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
BANK1	81h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

The **TRISB register**, located at address **86h** in bank 1, is used to select the direction of each pin (input or output):

- Each bit set to **1** in TRISB configures the corresponding pin as an **input**.
- Each bit set to **0** in TRISB configures the corresponding pin as an **output**.

Example:

```
BCF     STATUS, RP0    ; Select Bank 0, as the PORTB register is located in Bank 0.
CLRF    PORTB          ; Initialize PORTB.
BSF     STATUS, RP0    ; Select Bank 1.
MOVLW   B'11001111'    ; Value used to initialize the direction of PORTB.
MOVWF   TRISB          ; Configure RB<3:0> as input, RB<5:4> as output, and RB<7:6> as input.
```

7 TIMER0 Module

The PIC16F84 features a single 8-bit Timer module, unlike other PIC microcontrollers in the same mid-range family, such as the PIC16F877, which includes three timers (see the previous chapter). The primary function of the Timer is counting (essentially acting as a counter).

The table below summarizes the registers associated with the operation of the TIMER0 module.

Table 9.5: Overview of Registers Associated with TIMER0 Operations

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h	TMR0								
0Bh	INTCON	**GIE**	EEIE	**T0IE**	INTE	RBIE	**T0IF**	INTF	RBIF
81h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**
85h	TRISA	—	—	—	**TRISA4**	TRISA3	TRISA2	TRISA1	TRISA0

Note: The bits in red are associated with the operation of the TIMER.

7.1 Operating Modes of the Timer Module

The Timer module has two operating modes: **timer mode** or **counter mode**. The selection between these modes is made using bit 5 of the **OPTION_REG** register, called **T0CS** (Timer0 Clock Source Select bit):

- **T0CS = 0**: Operates in timer mode.
- **T0CS = 1**: Operates in counter mode.

7.1.1 Pulse Counter Mode

In this mode, the Timer counts pulses received on the **RA4/TOKI** pin. Since the **TMR0** register is an 8-bit memory register, it can count up to 255 pulses. If this value is exceeded, it resets to 0. Reading the **TMR0** register provides the number of pulses received on the **RA4/TOKI** pin.

In this mode, you can specify whether the counting occurs on the rising or falling edge of the pulse. This is determined by bit 4 of the **OPTION_REG** register, called **T0SE** (Timer0 Source Edge Select bit):

- **T0SE = 0**: Counting occurs on the rising edge, when the input (RA4/TOKI) transitions from 0 to 1.
- **T0SE = 1**: Counting occurs on the falling edge, when the input (RA4/TOKI) transitions from 1 to 0.

7.1.2 Timer Mode (Time Counter Mode)

In this mode, the Timer counts the PIC's clock cycles, effectively measuring time. When the **TMR0** register overflows (i.e., transitions from **FFH** to **00H**), the **T0IF** flag in the **INTCON** register is set to 1.

Overflow detection can be done in two ways:

- **Polling**: The program checks the **T0IF** bit to detect the overflow of **TMR0**.
- **Interrupt**: The timer interrupt can be enabled by setting the **T0IE** bit (TMR0 Overflow Interrupt Enable) to 1. When **T0IF** is set to 1, the interrupt occurs. The **T0IF** bit must be cleared by the Timer0 interrupt service routine before re-enabling this interrupt.

Note: The **TMR0 interrupt** cannot wake the processor from sleep mode, as the Timer stops functioning in sleep mode.

Example of polling:

```
CLRF   TMR0         ; Start counting after 2 cycles
BCF    INTCON, T0IF  ; Clear the overflow flag
loop:
BTFSS  INTCON, T0IF  ; Test if the counter has overflowed
GOTO   loop         ; No, wait for overflow
...
XXX                 ; Continue the program
```

Suppose you want to wait for 100 increments. In this case, load **TMR0** with a value such that the overflow occurs after 100 increments.

Example:

```
MOVLW  156          ; Load 156
```

```
MOVWF  TMR0        ; Initialize TMR0
BCF    INTCON, T0IF  ; Clear the flag
loop:
BTFSS  INTCON, T0IF  ; Test if the counter has overflowed
GOTO   loop        ; No, wait for overflow
...
XXX               ; Yes, proceed: 100 events elapsed (256-156=100)
```

7.1.3 Configuring Operating Modes of the TIMER Module

The figure below shows the functional block diagram of Timer 0.

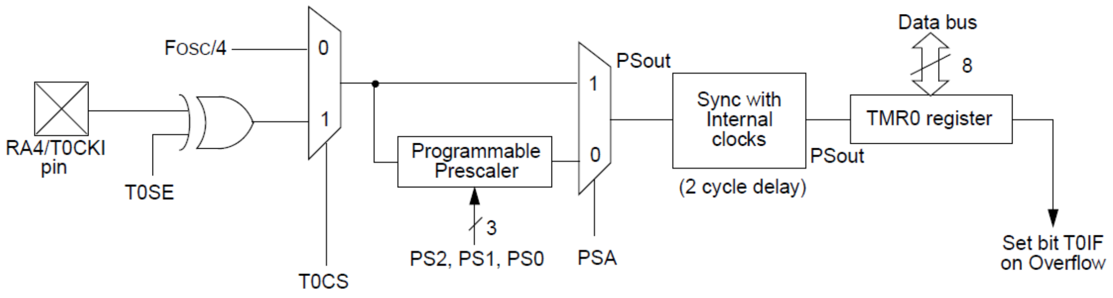


Figure 9.8: Functional Block Diagram of Timer0

The **timer mode** is selected by setting the **T0CS** bit to 0 (located in the **OPTION_REG**). In timer mode, the **TMR0** register increments with each clock cycle (without a prescaler).

The **counter mode** is selected by setting the **T0CS** bit to 1. In this mode, the **TMR0** register increments on each rising or falling edge of the **RA4/T0CKI** pin. The edge for incrementing is determined by the **T0SE** bit in the **OPTION_REG** register:

- **T0SE = 0**: TMR0 increments on the rising edge.
- **T0SE = 1**: TMR0 increments on the falling edge.

The **prescaler** is shared between the Timer0 module and the Watchdog Timer (WDT). The assignment of the prescaler is programmatically controlled by the **PSA** control bit in **OPTION_REG<3>**:

- **PSA = 0**: The prescaler is assigned to Timer0.
- **PSA = 1**: The prescaler is assigned to the Watchdog Timer (WDT).

The prescaler rate is determined by the table below:

PS2 PS1 PS0	**TMR0 Division Rate**	**WDT Division Rate**
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

Example:
Suppose the external clock frequency is 4 MHz (the oscillator is equipped with a 4 MHz quartz crystal). In this case, the internal cycle frequency is:

$$F_{cycle} = \frac{F_{osc}}{4} = \frac{4 \cdot 10^6}{4} = 1\,MHz.$$

Thus, the internal cycle period is: $T_{cycle} = 1\,\mu s$.

If we use **Timer0** in its timer function and in interrupt mode, an interrupt will occur every **256µs**.

If we want to blink an LED at a frequency of **1Hz**, we need a delay of **500ms**. To achieve this, we can use a **prescaler** to slow down the Timer0's time base.

- The prescaler allows the Timer0 register to increment more slowly, thereby increasing the time required to generate an interrupt.
  $\frac{T}{2} = 500\,ms = 500 \times 1000 = 500000\,\mu s$

8 Managing Interrupts on the PIC16F84

The PIC16F84 has four possible interrupt sources. The events that can trigger an interrupt are as follows:

- **RB0/INT**: An interrupt can be generated when the **RB0** pin (also called the INTerrupt pin), configured as an input, detects a change in the applied signal level.
- **TMR0**: The **TIMER0 module** can generate an interrupt when the **TMR0 register overflows** (i.e., when its content transitions from **FFH** to **00H**).
- **PORTB**: An interrupt can be generated when there is a change in the signal level on any of the **RB4 to RB7 pins**. The interrupt will be effective for all four pins collectively or none at all.
- **EEPROM**: This interrupt can be generated when a write operation to an internal EEPROM cell is completed.

8.1 Enabling Peripheral Interrupts

To enable one or more interrupts, you must first allow interrupts globally by setting the **GIE (Global Interrupt Enable)** bit in **INTCON<7>** to **1**. Then, enable only the specific interrupts of interest by setting their corresponding enable bits to **1**.

When a peripheral requests an interrupt, the processor invokes the appropriate **Interrupt Service Routine (ISR)** to determine how to handle the interrupt.

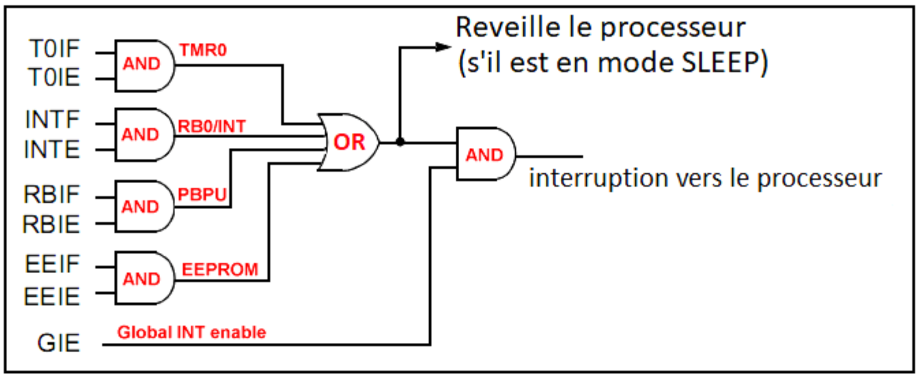


Figure 9.9: Interrupt Management Diagram of the PIC16F84

- **INTE bit (INTCON<4>)**: Enables the interrupt on the **RB0/INT** pin.
- **RBIE bit (INTCON<3>)**: Enables the "RB" interrupt.
- **T0IE bit (INTCON<5>)**: Enables the **Timer0** interrupt.
- **EEIE bit (INTCON<6>)**: Enables the interrupt triggered at the end of an EEPROM write operation.

Before enabling an interrupt, the flag that indicates the occurrence of the interrupt must be cleared (set to **0**):

- **INTF bit (INTCON<1>)**: Signals an interrupt on the **RB0/INT** pin.
- **RBIF bit (INTCON<0>)**: Signals an interrupt on pins **RB4..7**.
- **T0IF bit (INTCON<2>)**: Signals the **Timer0** interrupt.
- **EEIF bit (EECON1<4>)**: Signals the end of an EEPROM write operation.

	7	6	5	4	3	2	1	0
INTCON Register (Bank0)	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

	7	6	5	4	3	2	1	0
EECON1 Register (Bank1)	-	-	-	EEIF	WRERR	WREN	WR	RD

Figure 9.10: Register Responsible for Interrupt Management in the PIC16F84

8.2 Handling an Interrupt by the Microcontroller

When an interrupt occurs, the main program is interrupted as follows:

1. The processor completes the execution of the current instruction in the main program and pushes the **Program Counter (PC)** value onto the stack.
2. The processor jumps to address **0004H** in the program memory (the interrupt vector is located at address **H'0004'**).
3. The programmer must save the **STATUS** and **W** registers into temporary registers in RAM.
4. The program calls the interrupt subroutine (ISR) using the **CALL** instruction.
5. Once the ISR is completed, the programmer must clear the flag that signaled the interrupt.
6. The programmer must restore the **STATUS** and **W** registers from RAM.
7. The programmer signals the end of the interrupt using the **RETFIE** (Return from Interrupt) instruction. When this instruction is executed, the **PC** register content is restored from the stack, and the microcontroller resumes execution of the program it had interrupted.

The figure below illustrates the various steps involved in executing an interrupt subroutine.

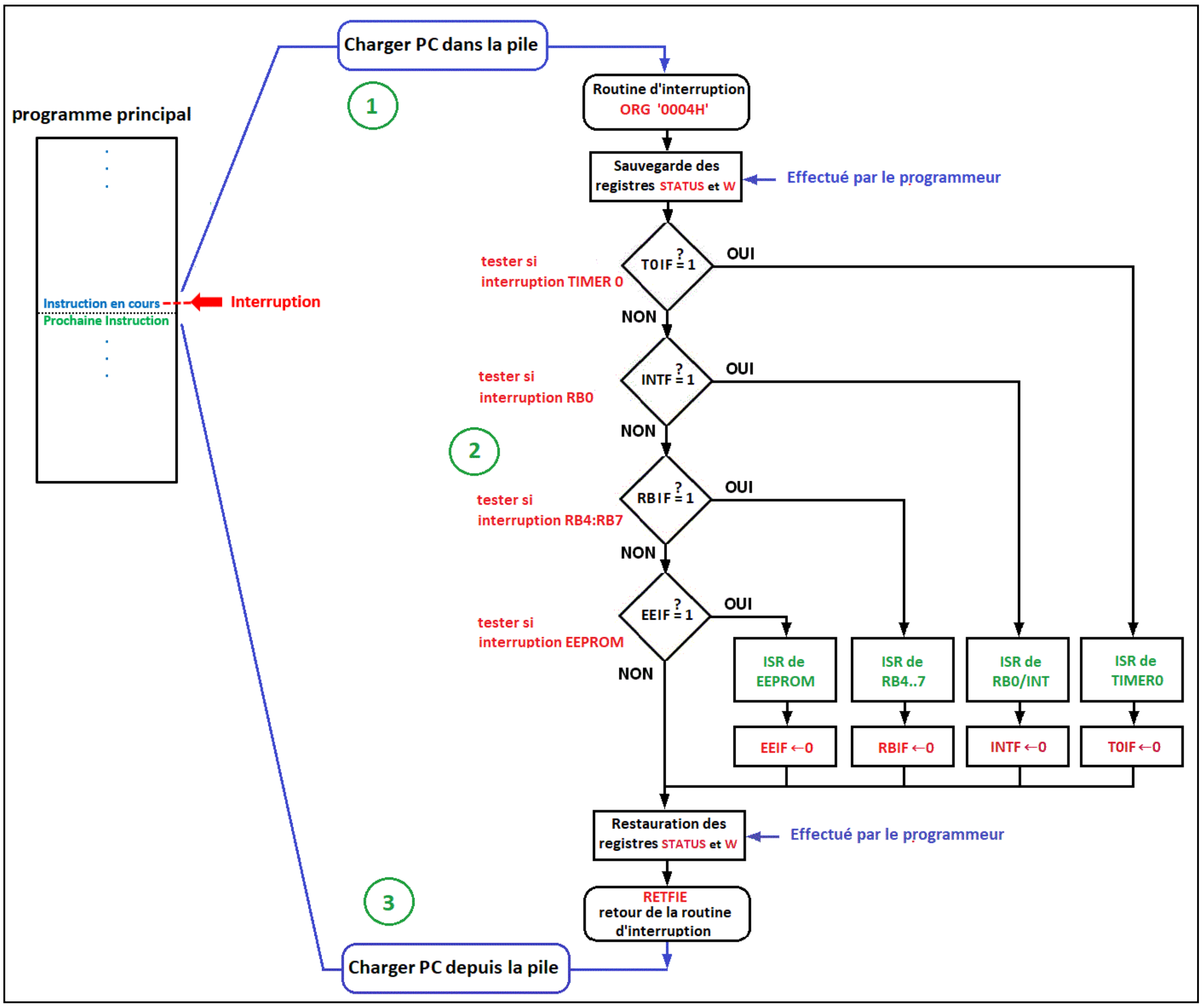

Charger PC dans la pile
1
programme principal
Instruction en cours
Prochaine Instruction
Interruption
Routine d'interruption ORG '0004H'
Sauvegarde des registres STATUS et W
Effectué par le programmeur
tester si interruption TIMER 0
T0IF ≟ 1
OUI
NON
tester si interruption RB0
INTF ≟ 1
2
tester si interruption RB4:RB7
RBIF ≟ 1
tester si interruption EEPROM
EEIF ≟ 1
ISR de EEPROM
ISR de RB4..7
ISR de RB0/INT
ISR de TIMER0
EEIF ←0
RBIF ←0
INTF ←0
T0IF ←0
Restauration des registres STATUS et W
Effectué par le programmeur
RETFIE retour de la routine d'interruption
3
Charger PC depuis la pile

Figure 9.11: Flowchart for Interrupt Handling in the PIC16F84

8.3 Assembly Program for Handling PIC16F84 Interrupts

```
ORG 0x004           ; Interrupt vector address
; Save W and STATUS registers
   MOVWF W_TEMP       ; Save W register
   SWAPF STATUS, W    ; Swap STATUS into W
   MOVWF STATUS_TEMP  ; Save swapped STATUS

; CHECK TIMER 0 INTERRUPT
   BTFSC INTCON, T0IE ; Check if Timer0 interrupt is enabled
   BTFSS INTCON, T0IF ; If yes, check if Timer0 interrupt is active
   GOTO test_RB0      ; No, go to the next test
   CALL ISR_TIMER0    ; Yes, handle Timer0 interrupt
   BCF INTCON, T0IF   ; Clear Timer0 interrupt flag
   GOTO restore_registre ; End of interrupt

; CHECK RB0 INTERRUPT
test_RB0
   BTFSC INTCON, INTE ; Check if RB0 interrupt is enabled
   BTFSS INTCON, INTF ; If yes, check if RB0 interrupt is active
```

```
    GOTO test_RB4_RB7  ; No, go to the next test
    CALL ISR_RB0       ; Yes, handle RB0 interrupt
    BCF INTCON, INTF   ; Clear RB0 interrupt flag
    GOTO restore_registre ; End of interrupt

; CHECK RB4..RB7 INTERRUPT
test_RB4_RB7
    BTFSC INTCON, RBIE ; Check if RB4..RB7 interrupt is enabled
    BTFSS INTCON, RBIF ; If yes, check if RB4..RB7 interrupt is active
    GOTO test_EEPROM   ; No, go to the next test
    CALL ISR_RB4_7     ; Yes, handle RB4..RB7 interrupt
    BCF INTCON, RBIF   ; Clear RB4..RB7 interrupt flag
    GOTO restore_registre ; End of interrupt

; CHECK EEPROM WRITE COMPLETION INTERRUPT
test_EEPROM
    BSF STATUS, RP0    ; Switch to Bank 1
    BTFSC INTCON, EEIE ; Check if EEPROM interrupt is enabled
    BTFSS EECON1, EEIF ; If yes, check if EEPROM interrupt is active
    GOTO restore_registre ; Restore registers
    CALL ISR_EEPROM    ; Handle EEPROM interrupt

restore_registre
    SWAPF STATUS_TEMP, W ; Restore old STATUS into W
    MOVWF STATUS         ; Restore STATUS
    SWAPF W_TEMP, F      ; Restore W register
    SWAPF W_TEMP, W
    RETFIE               ; Return from interrupt

; INTERRUPT SERVICE ROUTINES FOR EACH PERIPHERAL
ISR_TIMER0
    RETURN               ; End of Timer0 interrupt

ISR_RB0
    RETURN               ; End of RB0/INT interrupt

ISR_RB4_7
    RETURN               ; End of RB4..RB7 interrupt

ISR_EEPROM
    BCF EECON1, EEIF     ; Clear EEPROM interrupt flag
    BCF STATUS, RP0      ; Switch to Bank 0
    RETURN               ; End of EEPROM interrupt
```

Remarks

- An interrupt cannot interrupt another interrupt routine.
- If another interrupt occurs during the execution of an interrupt routine, it is ignored and will only be handled after the current routine finishes, when the microcontroller resumes the main program.

Saving and Restoring Context During Interrupts

When an interrupt subroutine is called, only the **Program Counter (PC)** value is saved on the stack. It is essential to save the content of the **W accumulator** and the **STATUS register** before executing the tasks in the

Interrupt Service Routine (ISR). After executing the ISR, the **W** and **STATUS** registers must be restored. The tasks can be outlined as follows:

1. Store the **W register** into **W_TEMP**.
2. Store the **STATUS register** into **STATUS_TEMP**.
3. Execute the interrupt service routine code.
4. Restore the **STATUS register**.
5. Restore the **W register**.

Example: Assembly program for saving the **STATUS** and **W** registers in RAM.

```
MOVWF   W_TEMP          ; Save W into W_TEMP
SWAPF   STATUS, W       ; Swap STATUS into W
MOVWF   STATUS_TEMP     ; Save STATUS into STATUS_TEMP
:
:                ; Perform interrupt service tasks
:
SWAPF   STATUS_TEMP, W  ; Swap STATUS_TEMP into W
MOVWF   STATUS          ; Restore W into STATUS
SWAPF   W_TEMP, F       ; Swap W_TEMP with itself
SWAPF   W_TEMP, W       ; Restore W register
```

Note: The **SWAP** instruction is used because it performs the transfer without affecting the flags in the **STATUS** register.

Exercise 1 :

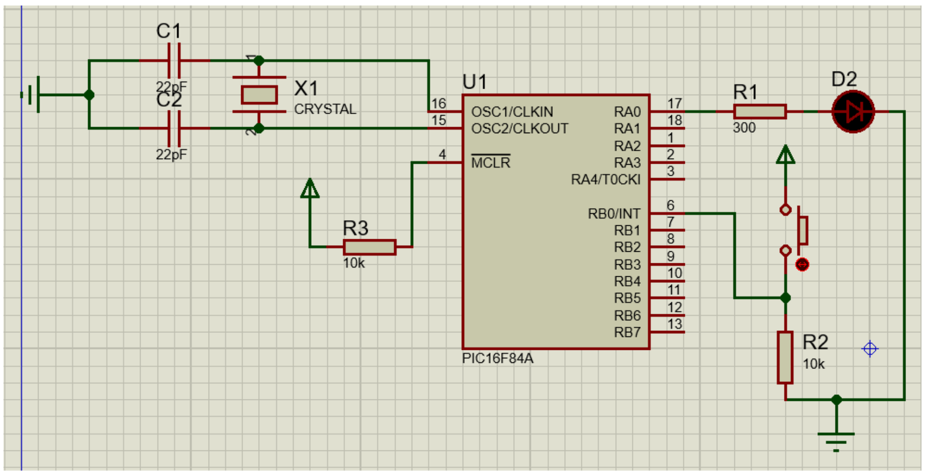


Figure 9.12: Circuit Diagram

The circuit above contains a push button connected to the **RB0** pin of the PIC16F84 and an LED connected to the **RA0** pin.

Task:

Write a program to turn on the LED when the button is pressed.

Solution for Exercise 1:

First, configure the direction of the microcontroller pins:

- **RB0** as input.
- **RA0** as output.

```
LIST      p=16F84A                ; Processor definition
#include <p16F84A.inc>            ; Constant definitions
```

```
            __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
;=============================================================================
;                   ASSIGNMENTS
;=============================================================================
OPTIONVAL           EQU       H'0000'             ; Option register value

#DEFINE led                   PORTA,0             ; LED
#DEFINE bouton      PORTB,0             ; Push button
;=============================================================================
;           PROGRAM VARIABLE DECLARATIONS
;=============================================================================
            CBLOCK 0x00C                ; Start of data area in RAM

            cmpt1 : 1                   ; Memory variable declaration cmpt1

            ENDC                                  ; End of data area
;=============================================================================
;    RESET STARTUP
;=============================================================================
            org        0x000                      ; Start address after reset
            goto    init                ; Jump to initialization

init
            clrf        PORTA
            clrf        PORTB                     ; Set PORTB outputs to 0

            bsf                   STATUS,RP0                ; Select bank 1
            movlw       OPTIONVAL                           ; Load configuration for OPTION register
            movwf       OPTION_REG                          ; Initialize OPTION register
    movlw B'11111111'                   ; Configure PORTB as input
    movwf TRISB
    clrf      TRISA
            bcf                   STATUS,RP0                            ; Select bank 0
            goto        start                               ; Jump to main program
;=============================================================================
;    MAIN PROGRAM
;=============================================================================
start
            btfss       bouton          ; Check if button is pressed
            goto    eteindre        ; No, turn off LED
            bsf                   led                 ; Yes, turn on LED
            goto    start           ; Loop back

eteindre
            bcf         led             ; Turn off LED
            goto        start                               ; Loop back
            END                                       ; End of program directive
```

8.4 Sleep Mode (SLEEP Mode)

This mode minimizes energy consumption in battery-powered applications. When the device is put to sleep, the program halts until it is awakened. Sleep mode is activated using the **SLEEP** instruction.

9 Instruction Set

The PIC16F84 features a set of 35 instructions, each encoded in 14 bits. The table below lists the mnemonics for these instructions along with a brief explanation of their functions.

Operands can be of several types:

Table 9.6: Instruction Set of the PIC16F84

Field	Description
f	Memory address of registers (register file address) from 00 to 7F
W	Working register (accumulator).
b	Bit address within an 8-bit file register (0 to 7).
d	**d** Destination selection: **d = 0**: Save the result in W. **d = 1**: Save the result in f.
k	Literal field (8 or 11 bits) constant value
TOS	Top of Stack (the top of the stack)
L	Literal, meaning an immediate value (constant).
( )	The content
→	Assign to
< >	Used to denote one or more bits of a register
PC	Program Counter (instruction pointer)

9.1 Data Transfer Instructions

No	Mnemonic	Description	Affected flags
1	**MOVF f,d**	**Transfer the content of f into (f or W)** Si d=0, (f) → W Si d=1, (f) → f	**Z**
2	**MOVWF f**	**Transfer the content of W into f** (W) → f	None
3	**MOVLW k**	**Load the literal k into W ,** 0 ≤ k ≤ 255 k → W	None
4	**SWAPF f, d**	**Swap the 4-bit groups of f** Si d=0, (f<3:0>) → W<7:4> et (f<7:4>) → W<3:0> Si d=1, (f<3:0>) → f<7:4> et (f<7:4>) → f<3:0>	None

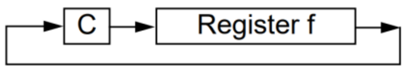
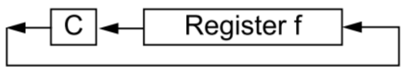
9.2 Arithmetic Instructions

No	Mnemonic	Description	Affected flags
5	**ADDWF f, d**	**Add W to F** If d=0 , (W) + (f) → (W) If d=1 , (W) + (f) → (f)	**C, DC, Z**
6	**ADDLW k**	**Add W to k ,** 0 ≤ k ≤ 255 literal	**C, DC, Z**

		If d=0, k + (W) → W	
7	**INCF f, d**	**Increment f** If d=0, (f) + 1 → W If d=1, (f) + 1 → f	**Z**
8	**SUBWF f, d**	**Subtract W from f** If d=0, (f) - (W) → W If d=1, (f) - (W) → f	**Z**
9	**SUBLW k**	**Subtract W from K ,** 0 ≤ k ≤ 255 literal If d=0, k - (W) → W If d=1, k - (W) → f	**C, DC, Z**
10	**DECF f, d**	**Décremente f** If d=0, (f) - 1 → W If d=1, (f) - 1 → f	**Z**

9.3 Logical Instructions

No	Mnemonic	Description	Affected flags
11	**ANDWF f, d**	**W and F** If d=0, (W) AND (f) → (W) If d=1, (W) AND (f) → (f)	**Z**
12	**ANDLW k**	**K and W ,** 0 ≤ k ≤ 255 (W).AND. (k) → W If d=0, (W) AND k → (W) If d=1, (W) AND k → (f)	**Z**
13	**IORWF f, d**	**W OR f** (W).OR. (k) → W If d=0, (W) OR k → (W) If d=1, (W) OR k → (f)	**Z**
14	**IORLW k**	**K OR W ,** 0 ≤ k ≤ 255 (W) OR k → W	**Z**
15	**XORWF f, d**	**W XOR F** If d=0, (W) XOR (f) → (W) If d=1, (W) XOR (f) → (f)	**Z**
16	**XORLW k**	**W XOR k ,** 0 ≤ k ≤ 255 If d=0, (W) XOR k → (W) If d=1, (W) XOR k → (f)	**Z**
17	**COMF f, d**	1's Complement of **f** If d = 0 , 1's Complement (f) → W If d = 1 , 1's Complement (f) → f	**Z**
18	**CLRW**	**Clear W** 00h → W 1 → Z	**Z**
19	**CLRF f**	**Clear f** 00h → f 1 → Z	**Z**
20	**BCF f, b**	**Clear the bit at position b of f** 0 → f<b>	None
21	**BSF f, b**	**Set the bit at position b of f to '1'** 1 → f<b>	None

22	**RRF f, d**	Rotate f one position to the right through the Carry Flag (CF) If d = 0, f after rotation → W If d = 1, f after rotation → f 	**C**
23	**RLF f, d**	Rotate f one position to the left through the Carry Flag (CF) - If d = 0, f after rotation → W - If d = 1, f after rotation → f 	**C**

9.4 Branch Instructions

No	**Mnemonic**	**Description**	**Affected flags**
24	**DECFSZ f, d**	Decrement **f** and skip if zero Si d=0, (f) - 1 → W; jump if resultat = 0 Si d=1, (f) - 1 → f; jump if resultat = 0	None
25	**INCFSZ f, d**	Increment **f** and skip if zero Si d=0, (f) + 1 → W; jump if resultat = 0 Si d=1, (f) + 1 → f; jump if resultat = 0	None
26	**BTFSC f, b**	Test bit **b** of **f** and skip if '0' Sauter Si (f<b>) = 0	None
27	**BTFSS f, b**	Test bit **b** of **f** and skip if '1' Sauter Si (f<b>) = 1	None
28	**CALL k**	Save return address, load PC with **k** (PC)+ 1→ TOS k → PC<10 :0> (PCLATH<4 :3>) → PC<12 :11>	None
29	**GOTO k**	Jump to address **k** (9 bits!).	None
30	**RETFIE**	Return from interrupt.	None
31	**RETURN**	Return from subroutine.	None
32	**RETLW k**	Load **k** into W and return.	None

9.5 Control Instructions

No	**Mnemonic**	**Description**	**Affected flags**
33	**NOP**	No operation.	None
34	**SLEEP**	Stops the processor.	None
35	**CLRWDT**	Resets the watchdog timer.	None