MECE336 Microprocessors I
Timer/Counter

Dr. Kurtuluş Erinç Akdoğan

kurtuluserinc@cankaya.edu.tr

Course Webpage: http://MECE336.cankaya.edu.tr

ÇANKAYA ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

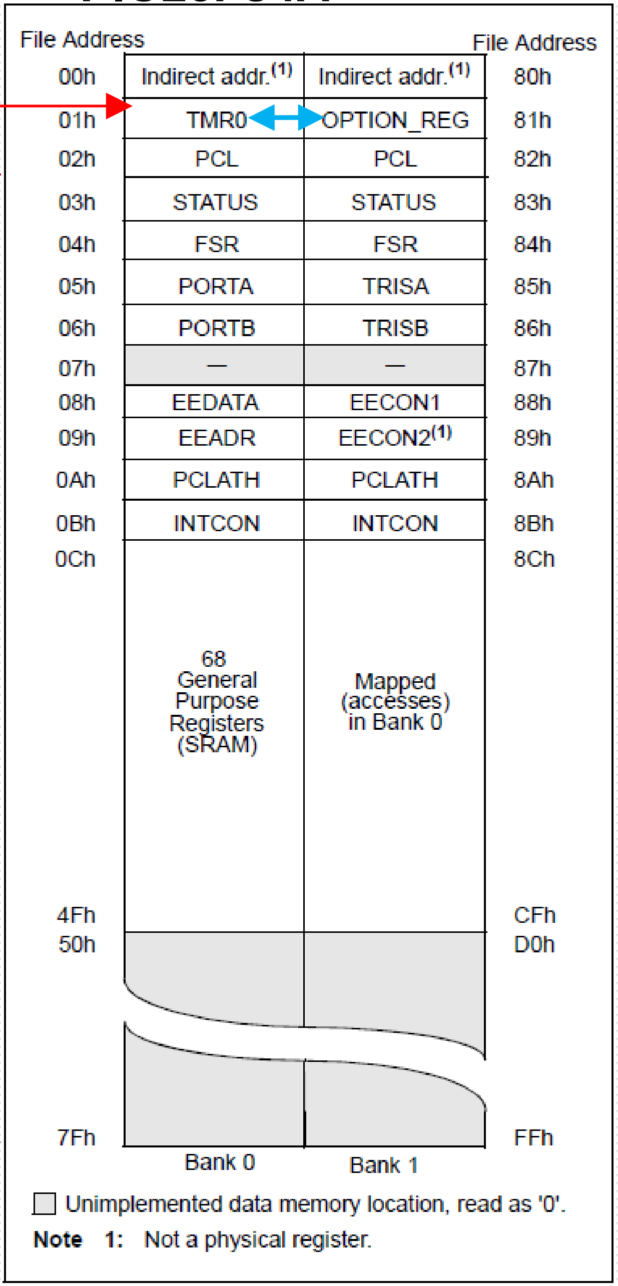
PIC16F84 TIMER PROGRAMMING

- The PIC16F84 has two timers depending on the family member.
  - **Timer 0**
  - **Watchdog timer (WDT).**
- They can be used either as timers to generate a time delay or as counters to count events happening outside the microcontroller.
- Every timer needs a clock pulse to tick. The clock source can be internal or external.
- If we use the internal clock source, then 1/4th of the frequency of the crystal oscillator on the OSC1 (Fosc/4) pin is fed into the timer. Therefore it is used for time delay generation and for that reason is called a timer.
- By choosing the external clock option, we feed pulses through one of the PIC16’s pins: this is called a counter.

TMR0

- TMR0 is an 8-bit special function register in the RAM. It has the following features;
- Timer0 can operate as a 8-bit timer or as a 8-bit counter.
- Readable and writable
- Timer 0 is configurable, controlled by a number of bits that appear in the OPTION register.
- Internal or external clock can be selected
- Edge can be selected for external clock (rising or falling edge)
- 8-bit software programmable prescaler
- Interrupt occurs when TMR0 counts from h'FF' to h'00' (Timer overflow interrupt)

REGISTER FILE MAP - PIC16F84A

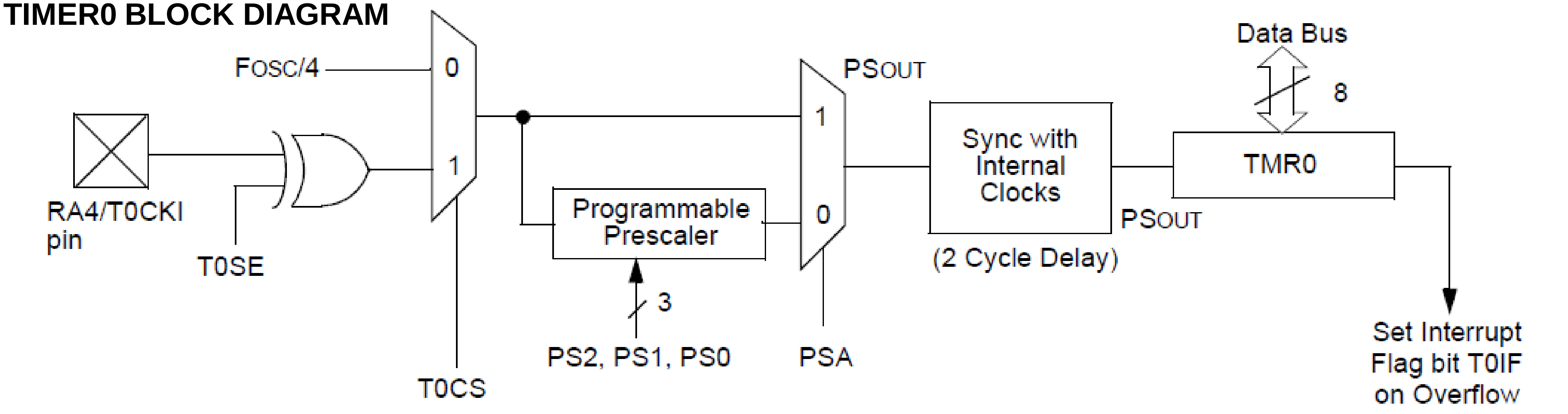


File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch – 4Fh	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch – CFh
50h – 7Fh			D0h – FFh

▨ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

TIMER0 BLOCK DIAGRAM



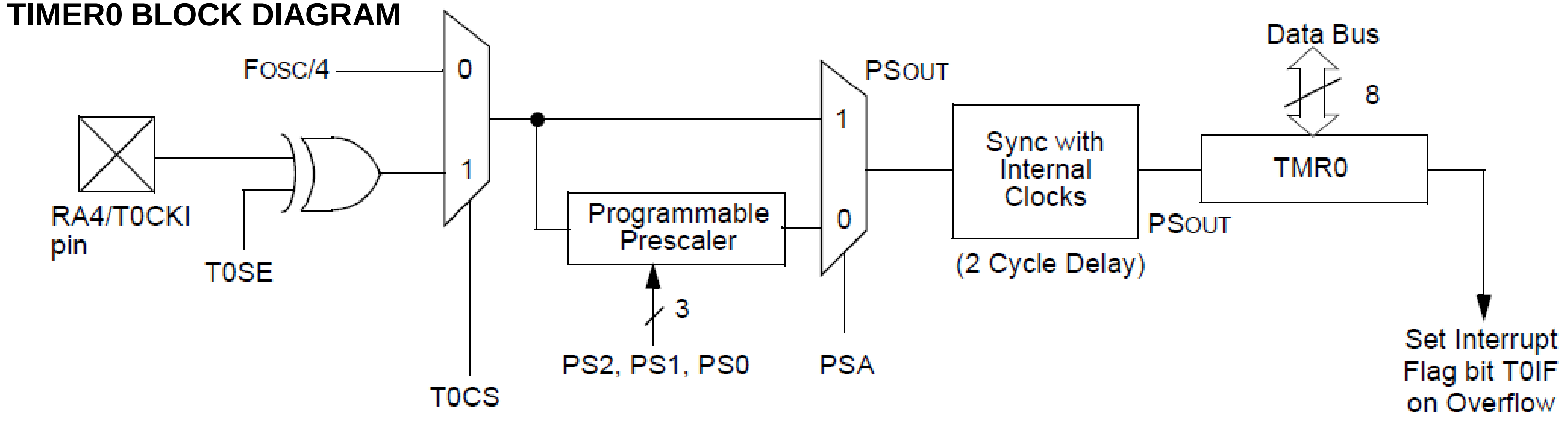
Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

OPTION register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

- **T0CS** controls the sources of the clock input to the **TMR0** counter whether **RA4** pin or internal instruction cycle frequency, labelled Fosc/4.
- Timer mode is selected by clearing bit **T0CS** (**OPTION_REG**<5>). In Timer mode, the Timer0 module will increment every instruction cycle.
- Counter mode is selected by setting bit **T0CS** (**OPTION_REG**<5>). In Counter mode, **Timer0** will increment, either on every rising or falling edge of pin **RA4/T0CKI**.
- The incrementing edge is determined by the **Timer0 Source Edge Select** bit, **T0SE** (OPTION_REG<4>). Clearing bit **T0SE** selects the rising edge.

TIMER0 BLOCK DIAGRAM



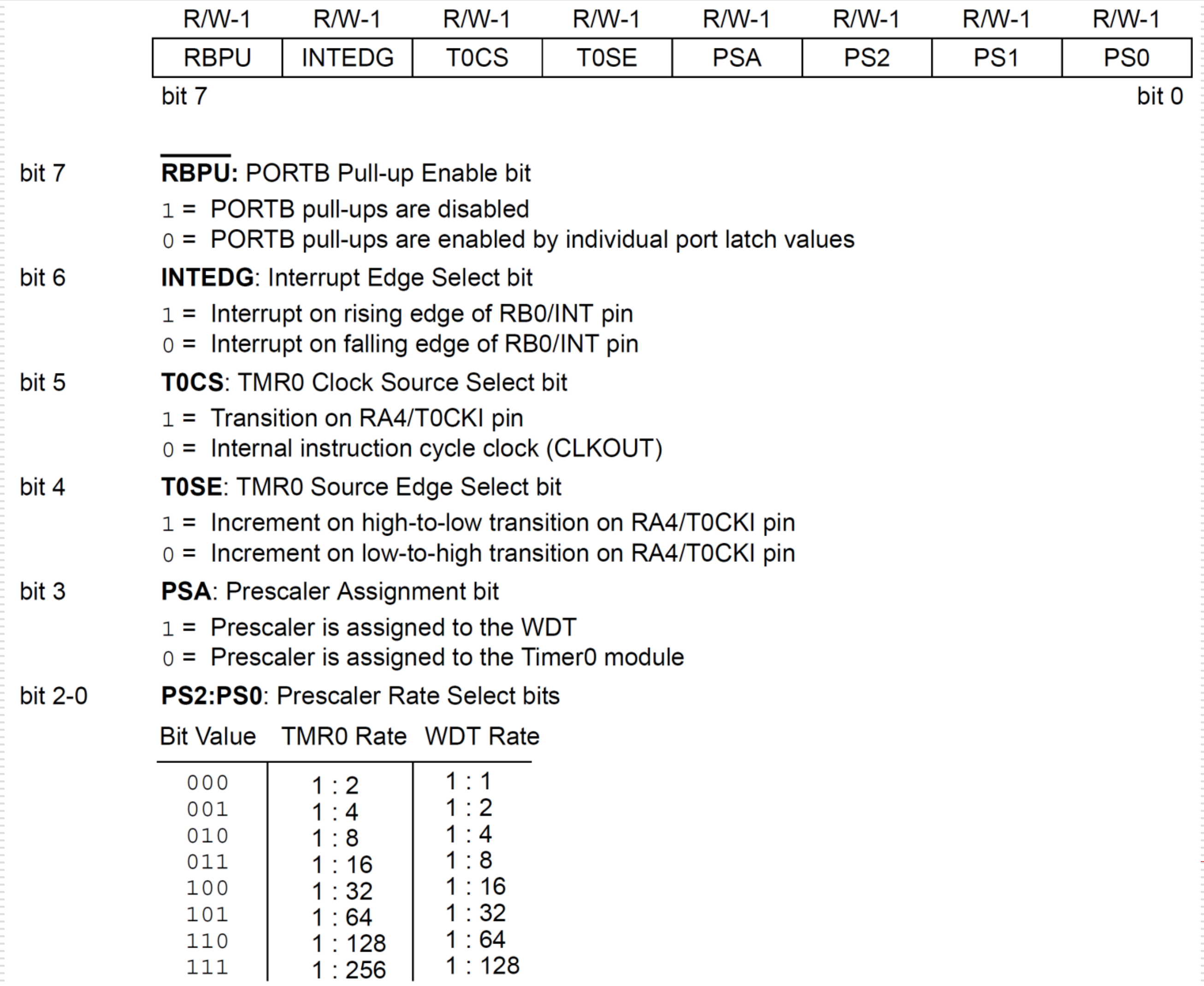
Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

OPTION register

bit 7							bit 0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

- The output of the first multiplexer branches before reaching a second multiplexer. This selects either a direct path or the path taken through a programmable **prescaler**.
- The choice is controlled by bit **PSA** of the Option register. If PSA is set to 0, then the prescaler is assigned to the Timer 0.
- The prescaler itself is controlled by bits PS2, PS1 and PS0 of the Option register. They allow a choice of frequency divisions of the incoming clock signal.
- The output of the second multiplexer is synchronised with the internal clock, before becoming the input to the actual counter.
- When the counter overflows, it sets the timer overflow flag, one of the PIC microcontroller’s four interrupt sources.

OPTION 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
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

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 is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

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

-The PSA and PS2:PS0 bits (OPTION_REG<3:0>) determine the prescaler assignment and prescale ratio.

-Clearing bit PSA will assign the prescaler to the Timer0 module.

-When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

-If TMR0=1/2, TMR0 increases every 2 instruction cycle. If TMR0=1/128, TMR0 increases every 128 instruction cycle.

Example

- If prescaler value is b’000’, what is the increment period of TMR0 and maximum interrupt delay? (Oscillator frequency 4 MHz)

SOLUTION

- internal frequency=4MHz/4=1MHz
- ICT (Instruction cycle time)=1/1MHz=1us
- Prescaler value=000, TMR0_rate=1/2
- IP(Increment period)=2x1μs=2μs
- Timer overflow Interrupt occurs when TMR0 counts from h’FF’ to h’00’. There are 256 numbers between h’00’ to h’FF’ for maximum interrupt delay.
- ID(Interrupt delay)=IPx256=2usx256=512us

Example:
Use Timer 0 as a counter

- For an electronic ping-pong circuit, right paddle is used as the counter input, continuously displaying the current value on the LEDs connected to Port B.

- To configure Timer 0, we'll need to select its external input, i.e. **T0CS=1**.

- Due to less likelihood of bounce, rising edge (switch release) is selected for input by setting **T0SE=0**.

- Exact number of switch presses will be counted so by setting PSA=1, **WatchDog Timer (WDT)** will be able to use prescaler. Hence the values of PS2, PS1 and PS0 do not matter.

- Rest of the bits of Option register are set to 0 since they don't matter.

-A final value for the Option register setting is thus 00101000B.

```
;*******************************************************************
;cntr_demo                                     Counter Demonstration
;This program demos Timer 0 as counter, using ping-pong hardware
;TJW 15.4.05                                          Tested 15.4.05
;*******************************************************************
;Clock freq 800kHz approx (RC osc.)
;Port A 4    right paddle (ip) Counter input.
;       2    "out of play" led (op)
;Port B 7-0  "play" leds (all op)
;Interrupts not used
;Config Word: RC oscillator, WDT off, PU timer on, code protect off
;
      #include p16f84A.inc
;
             org   00
; Initialise
             bsf   status,rp0   ;select memory bank 1
             movlw B'00011000'
             movwf trisa        ;port A according to above pattern
             movlw 00
             movwf trisb        ;all port B bits output
             movlw B'00101000'  ;set up TMR0 for external input, +ve edge,
                                                            ;no prescale

             movwf TMR0         ;as we are in Bank 1, this addresses OPTION
             bcf   status,rp0   ;select bank 0
;
             movlw 04     ;switch on "out of play" led to show power is on
             movwf porta
loop         movf  TMR0,0 ;Continuously display Timer 0 on Port B
             movwf portb
             goto  loop
             end
```

Hardware-generated delays

- We have used **software-generated delays** to time how long the LEDs are to be illuminated.
- This is **only acceptable in simple programs**, as in software-generated delays the CPU is doing nothing useful during the whole of the delay.
- Now that we have a **counter/timer** at our disposal, we can use it to generate the delay and let CPU be free.
- This seems quite simple, but a small problem presents itself: how do we know when the delay period is up?
- If we have to keep checking the timer value, then we will have made little progress. This is where the **'interrupt on overflow'** comes into its own.
- If things are set up so that an interrupt is generated as the delay ends, then we have a powerful means of creating efficient delays.

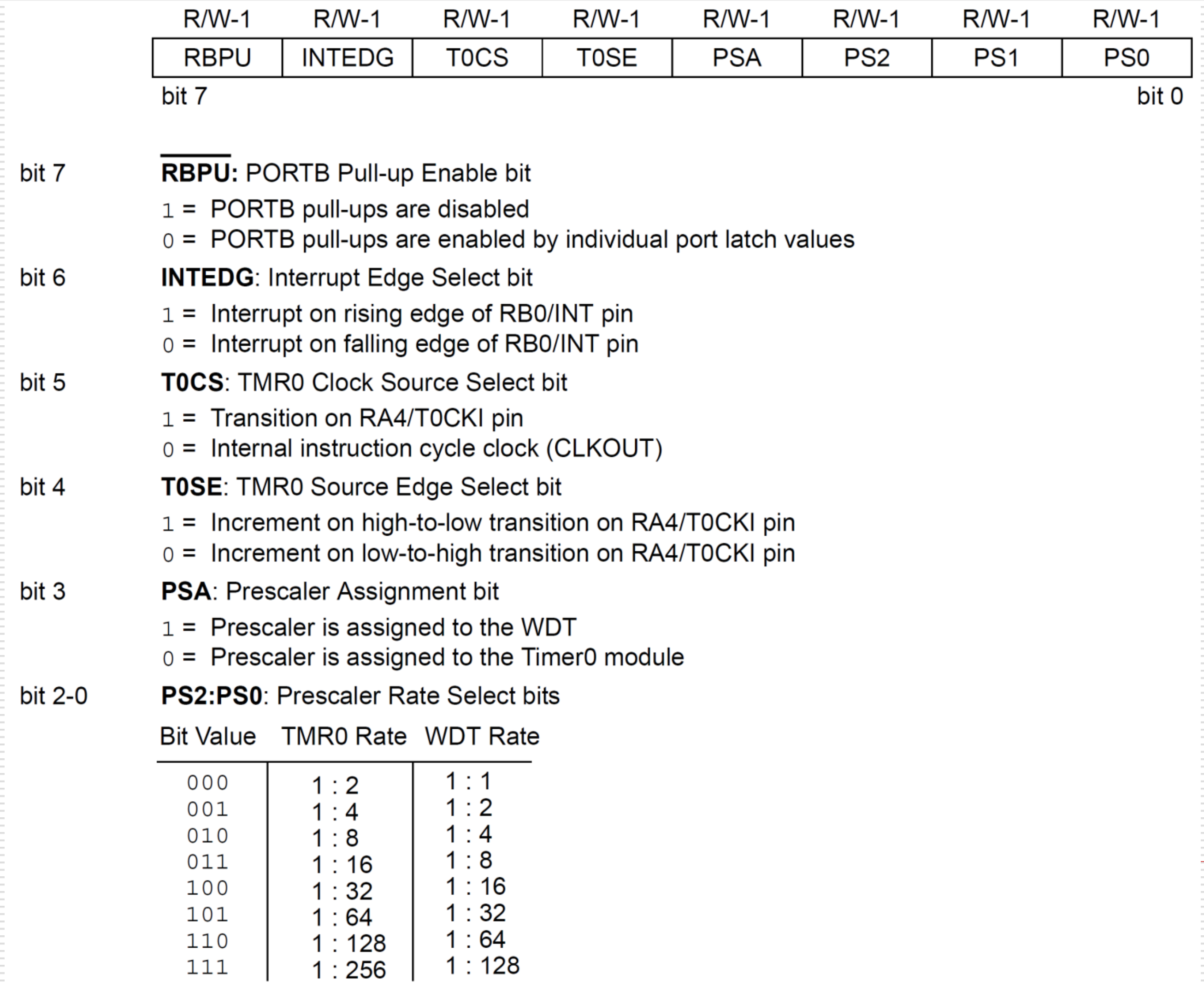
Hardware-generated delays

- As a first step, let’s replace the 5 ms software delay subroutine with a delay controlled by Timer 0.
- The internal clock is approximately 800 kHz and the instruction cycle rate (Fosc/4) is therefore 200 kHz, or a period of 5 μs.
- Now with this clock frequency, Timer 0 would count up to its maximum value (255) in 255x5μs, or 1275μs, and would overflow on the next cycle, i.e. after 1280μs.
- We can, however, make use of the prescaler here. If the incoming signal is divided by 4 (i.e. PS2, PS1, PS0 set to 001), then Timer 0 will overflow after 256x4x5 μs, or 5.120μs. This is very close to the 5 ms we’re looking for, but it’s not quite exact.
- Although the ping-pong program does not need accurate timing, suppose we genuinely needed a delay very close to 5 ms?
- Let us divide the incoming clock by 8 instead of 4, which gives a divided frequency of 25 kHz, or a period of 40 μs.
- Now 125 Timer 0 input cycles will cause a delay of 40x125μs, or 5.00 ms, which is exactly our target.
- If we arrange for this prescaling, and at the start of each delay pre-load Timer 0 with 256-125=131, then an exact delay, terminated by the interrupt on overflow, is possible.

Example

- Write a code that produces 5ms delay using timer module and interrupt to switch on a LED connected to PORTB.

00000010 value in OPTION REGISTER



R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

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 is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

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

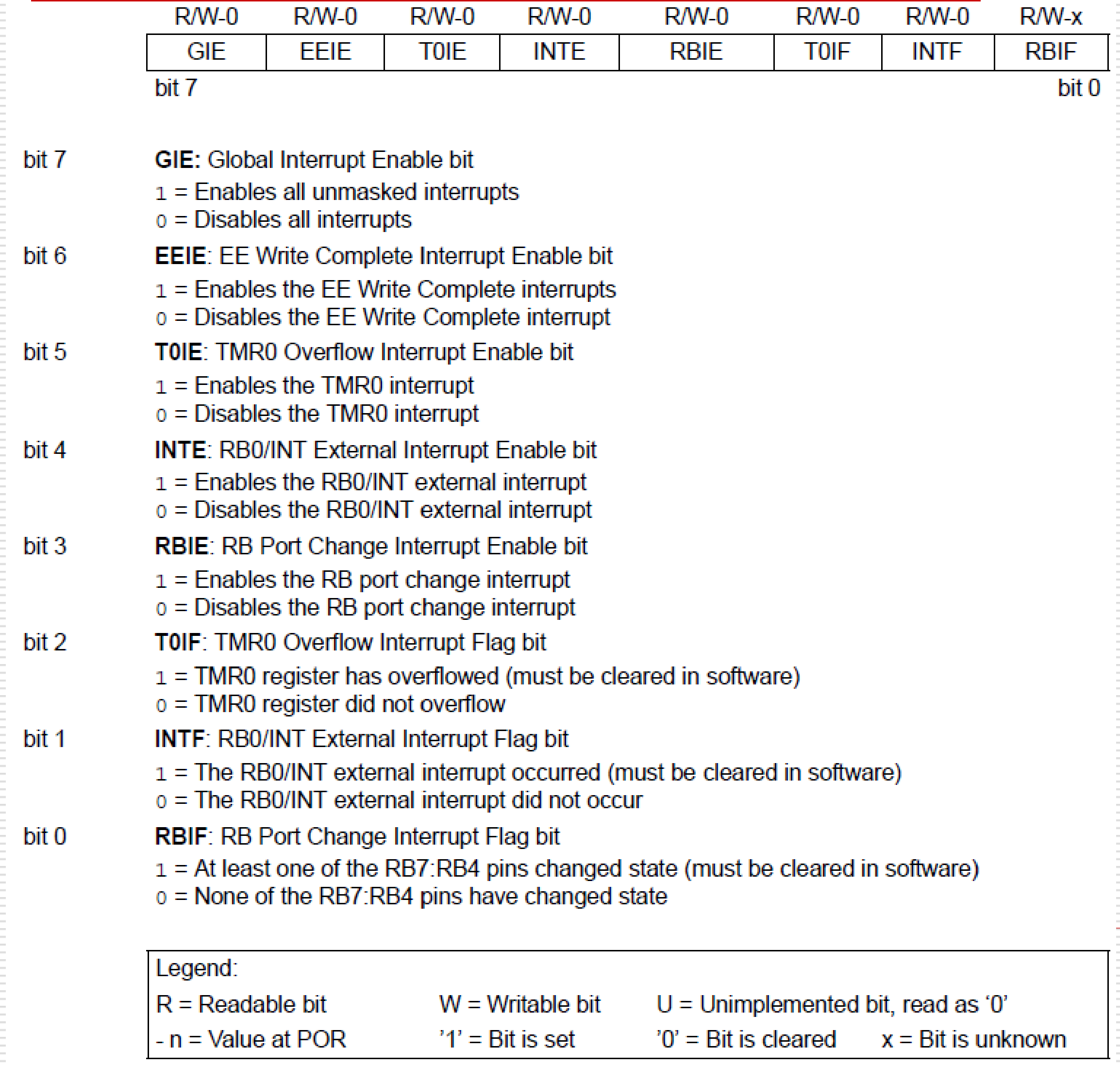
-The PSA and PS2:PS0 bits (OPTION_REG<3:0>) determine the prescaler assignment and prescale ratio.

-Clearing bit PSA will assign the prescaler to the Timer0 module.

-When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

-If TMR0=1/2, TMR0 increases every 2 instruction cycle. If TMR0=1/128, TMR0 increases every 128 instruction cycle.

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	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
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 External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

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

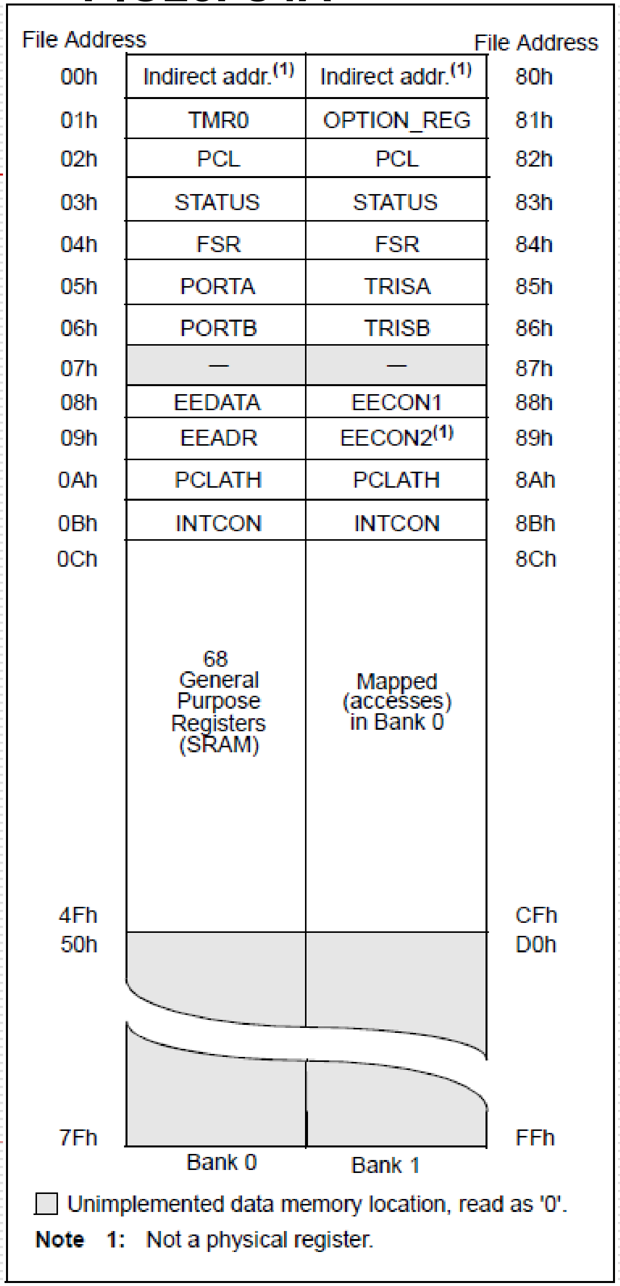
bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = 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

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

REGISTER FILE MAP - PIC16F84A



File Address			File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	—	—	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch – 4Fh	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch – CFh
50h – 7Fh	▨	▨	D0h – FFh
	Bank 0	Bank 1	

▨ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

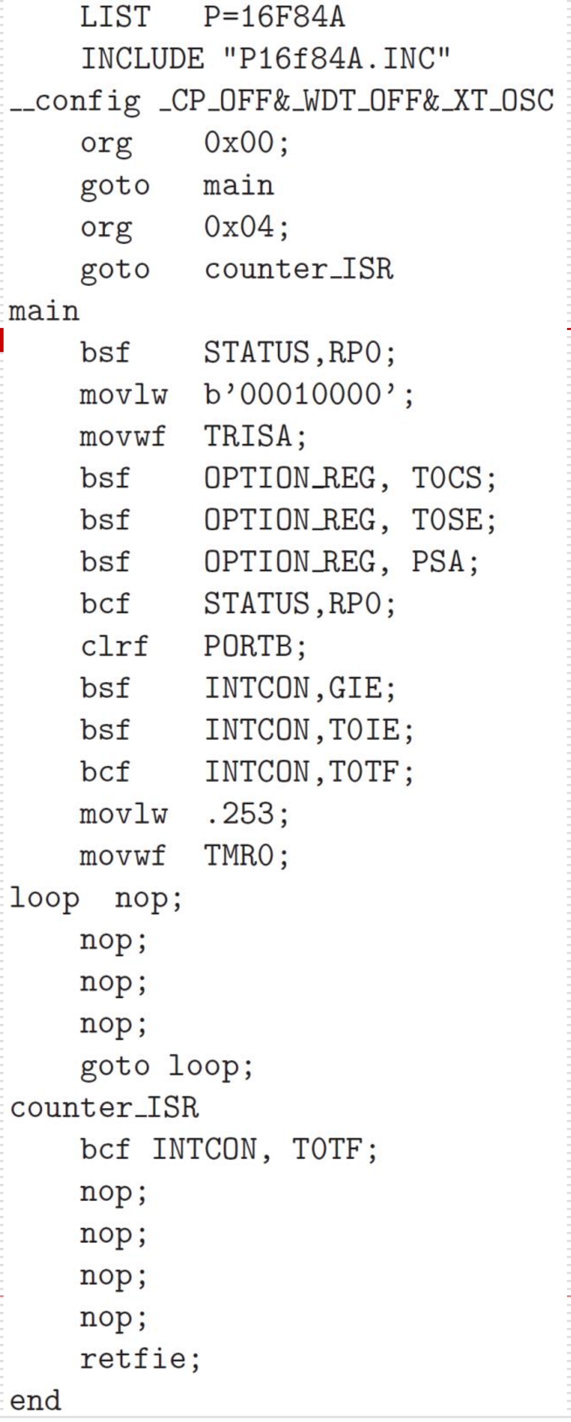
Example

- This includes both the initialization section and the revised delay subroutine.
- Interrupts are not enabled and the subroutine determines when the delay is complete by testing the overflow interrupt flag.
- The advantage to the programmer is that timing is now achieved by manipulating the Timer 0 settings, rather than by adjusting the software routine.
- The 'interrupt on overflow' has not been enabled, as it would in this instance offer little advantage.
- In a more demanding program, however, the interrupt could be enabled and the time spent in the delay used to undertake other CPU activities.

```
...
;Initialise
      org   0010
start bsf   status,5     ;select memory bank 1
      movlw B'00011000'
      movwf trisa        ;port A according to above pattern
      movlw 00
      movwf trisb        ;all port B bits op
      movlw B'00000010'  ;set up TMR0 for internal input, prescale by 8
      movwf TMR0         ;as we are in Bank 1, this addresses OPTION
      bcf   status,5     ;select bank 0
...
...
;introduces delay of 5ms approx
      delay5 movlw D'131'        ;preload counter, so that 125 cycles, each
                                 ;of 40us, occur before timer overflow
      movwf TMR0
del1  btfss intcon,2             ;test for Timer Overflow flag
      goto del1                  ;loop if not set
      bcf intcon,2               ;clear Timer Overflow flag
      return
```

Example

- Explain what the program does.
- Modify the program such that
  - the timer starts counting from 253 after the interrupt subroutine
  - the prescaler with rate 1 : 2 is used.



```
    LIST    P=16F84A
    INCLUDE "P16f84A.INC"
__config _CP_OFF&_WDT_OFF&_XT_OSC
    org     0x00;
    goto    main
    org     0x04;
    goto    counter_ISR
main
    bsf     STATUS,RP0;
    movlw   b'00010000';
    movwf   TRISA;
    bsf     OPTION_REG, TOCS;
    bsf     OPTION_REG, TOSE;
    bsf     OPTION_REG, PSA;
    bcf     STATUS,RP0;
    clrf    PORTB;
    bsf     INTCON,GIE;
    bsf     INTCON,TOIE;
    bcf     INTCON,TOTF;
    movlw   .253;
    movwf   TMR0;
loop  nop;
    nop;
    nop;
    nop;
    goto loop;
counter_ISR
    bcf INTCON, TOTF;
    nop;
    nop;
    nop;
    nop;
    retfie;
end
```

Example

- To generate 1.28ms interrupt delay, what will be the first number of the TMR0.(Fosc=4MHz, Prescaler=110)

SOLUTION

- Internal frequency=4MHz/4=1MHz
- ICT (Instruction cycle time)=1/1MHz=1µs
- For 1.28ms ID, 1.28ms/1 µs=1280 instruction.
- Prescaler=110; TMR0 increases every 128 instruction cycle. 1280 /128=10 ,
- To create 1.28ms interrupt delay, TMR0 should count 10 number.
- Timer overflow Interrupt occurs when TMR0 count from h’FF’ to h’00’. 256-10=246;First number of the TMR0=246.

Example

- Assume that the oscillator frequency is 4Mhz. Configure the Timer 0 such that an interrupt occurs after approximately 65.5msec.
- Write a program that
  - increments PORTB every 65.5msec
  - clears PORTB and TMR0 if the button at pin RA2 is pressed.