EEC 417/517
Embedded Systems
Cleveland State University

Lab 4

Macros, Context Saving,

Pulse Width Modulation (PWM),

The Capture/Compare/PWM (CCP) Module

Dan Simon
Rick Rarick

Summer 2018

Lab 4 Outline

1. **Macros**
2. Saving Context During Interrupts
3. Pulse Width Modulation
4. Lab 4 Setup for PWM
5. Capture, Compare, PWM (CCP) Module
6. Lab 4 Setup for CCP

Macros

1. Macro: a user-named assembler directive.

2. Defines a block of instructions.

3. The assembler inserts the instructions in program memory at the location where the macro is called.

4. Often used where the same code is repeated in a program.

Macro Example

A macro called PortCout that makes all of the PORTC pins outputs.

```
PortCout      macro
              banksel      TRISC
              movlw        0x00
              movwf        TRISC
              banksel      PORTA
              endm
```

Macro Example

Macro definition

MPLAB IDE Editor

lab04a.asm

```
121
122     clrf        Timer0Flag  ; Initialize Timer0Flag to 0
123
124 ;*****************************************************************
125 ; Main Routine
126 ;*****************************************************************
127
128 MAIN
129
130 PortCout    macro
131             banksel     TRISC
132             movlw       0x00
133             movwf       TRISC
134             banksel     PORTA
135             endm
136
137     banksel     ADCON0      ; ADCON0: Bank 0
138
139     bsf         ADCON0, GO  ; Start A/D conversion
140
141 PortCout
142
143
144 WaitForConversion
145
146     btfss       PIR1, ADIF  ; Wait for conversion to complete
147                             ; PIR1: Bank 0
148
149     goto        WaitForConversion
150
151     ; Get the duty cycle
152
153     bcf         PIR1, ADIF  ; Clear the A/D interrupt flag
154
155     movf        ADRESH, W   ; Get A/D result. ADRESH: Bank 0
156
157     movwf       DutyCycle   ; Copy A/D result to DutyCycle
158
```

Program Memory

Line	Address	Opcode	Label	Disassembly
1	0000	0000		NOP
2	0001	2805		GOTO INIT
3	0002	3FFF		
4	0003	3FFF		
5	0004	2823		GOTO Int_Svc_Rtn
6	0005	30E0	INIT	MOVLW 0xe0
7	0006	008B		MOVWF INTCON
8	0007	0187		CLRF PORTC
9	0008	3041		MOVLW 0x41
10	0009	009F		MOVWF ADCON0
11	000A	1683		BSF STATUS, 0x5
12	000B	1303		BCF STATUS, 0x6
13	000C	3083		MOVLW 0x83
14	000D	0081		MOVWF TMR0
15	000E	0187		CLRF PORTC
16	000F	300E		MOVLW 0xe
17	0010	009F		MOVWF ADCON0
18	0011	1283		BCF STATUS, 0x5
19	0012	1303		BCF STATUS, 0x6
20	0013	01A1		CLRF Timer0Flag
21	0014	1283	MAIN	BCF STATUS, 0x5
22	0015	1303		BCF STATUS, 0x6
23	0016	151F		BSF ADCON0, 0x2
24	0017	1683		BSF STATUS, 0x5
25	0018	1303		BCF STATUS, 0x6
26	0019	3000		MOVLW 0
27	001A	0087		MOVWF PORTC
28	001B	1283		BCF STATUS, 0x5
29	001C	1303		BCF STATUS, 0x6
30	001D	1F0C	WaitForConver	BTFSS PIR1, 0x6
31	001E	281D		GOTO WaitForConversion

Opcode Hex | Machine | Symbolic

Macro call

Macro instructions

Why PORTC ?

Lab 4 Outline

1. Macros
2. **Saving Context During Interrupts**
3. Pulse Width Modulation
4. Lab 4 Setup for PWM
5. Capture, Compare, PWM (CCP) Module
6. Lab 4 Setup for CCP

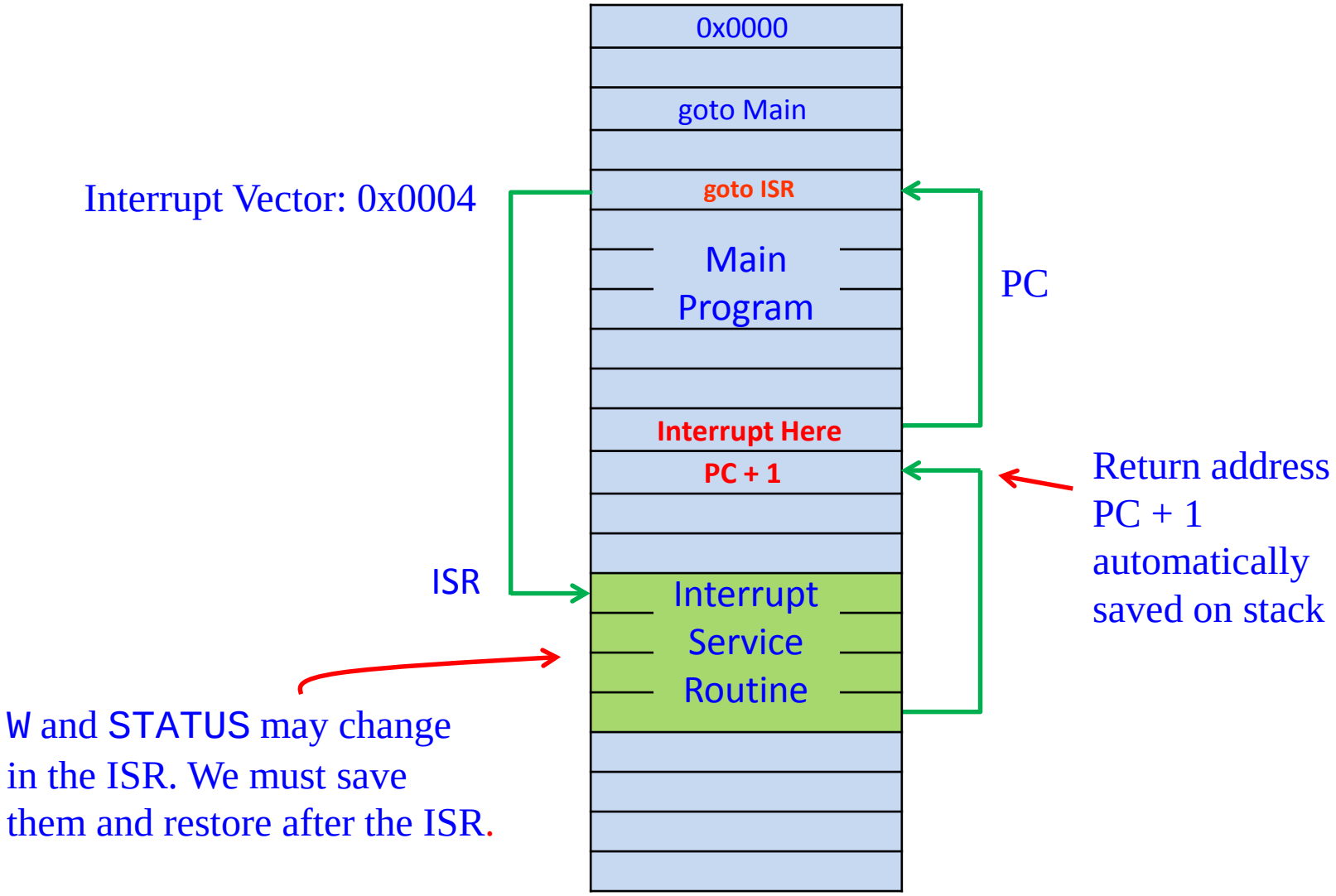
Saving Context During Interrupts

1. Interrupts occur at unpredictable times during program execution.

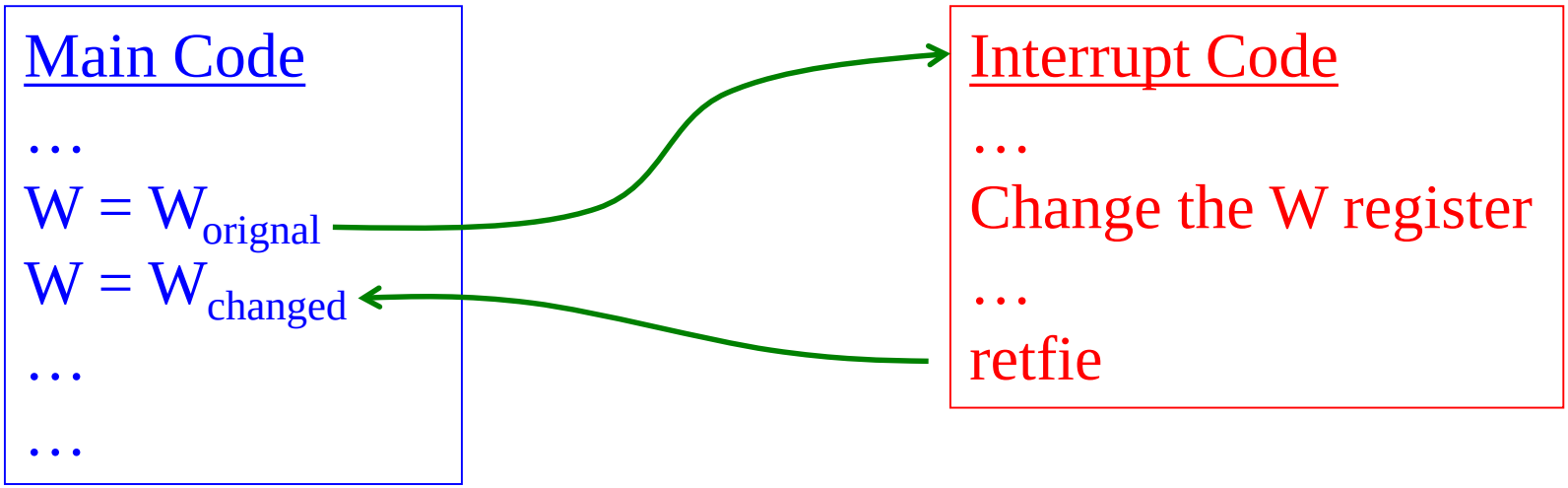
2. When an interrupt occurs, the current contents of the `W` and `STATUS` registers (and possibly others) may be changed within the interrupt service routine.

3. We need to save `W` and `STATUS` and restore them after the interrupt so that the CPU can continue executing instructions with W and `STATUS` as they were before the interrupt.

Saving Context During Interrupts



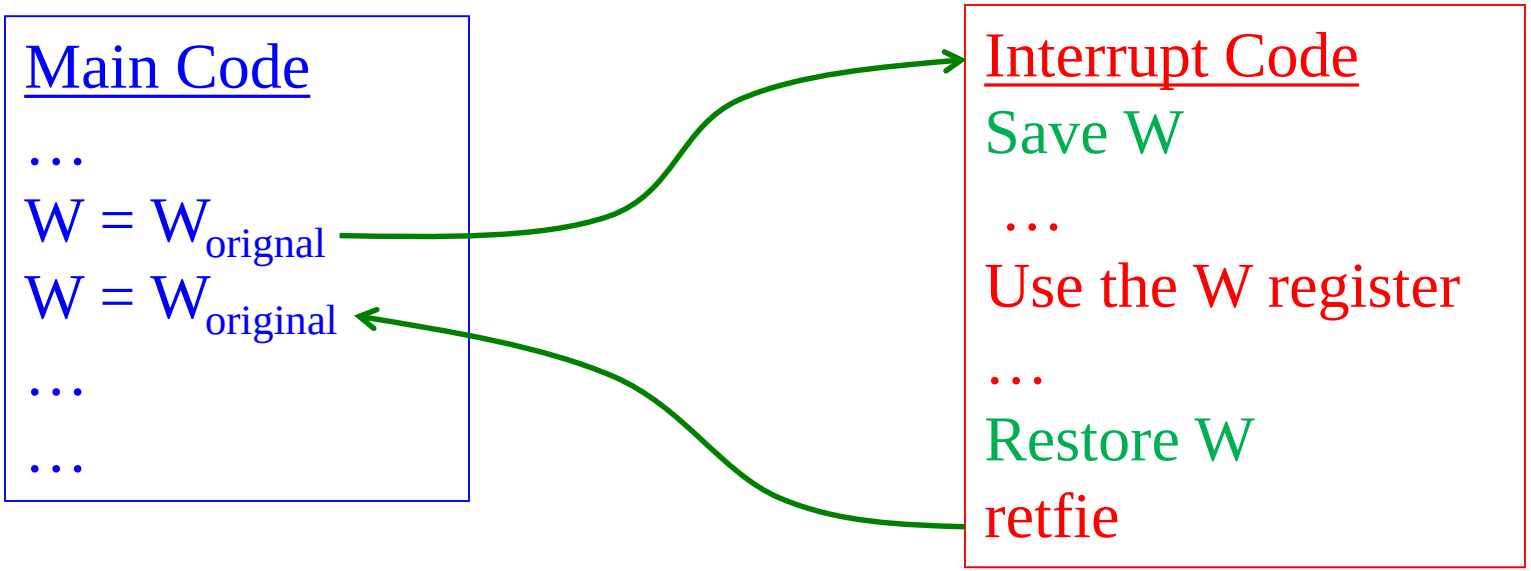
Saving Context During Interrupts



Problem: The main code may not work correctly because the W register changed. This an example of a "concurrency problem."

Solution: Save important registers (save context) at the start of each interrupt, and restore them (restore context) at the end of each interrupt.

Saving Context During Interrupts

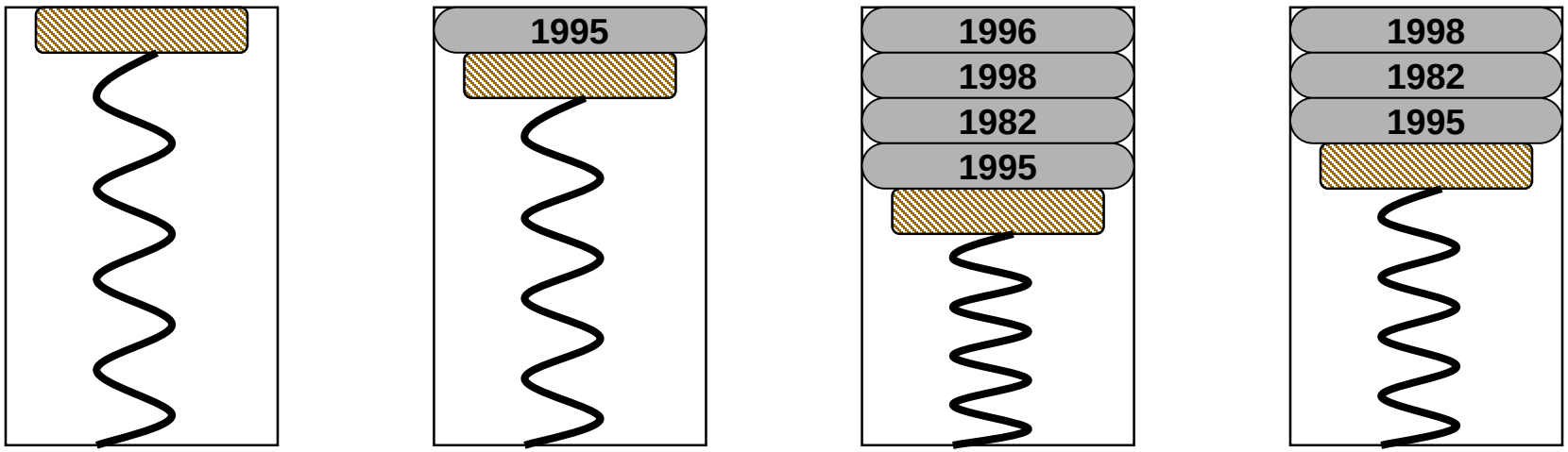


Other registers may also need to be saved, especially STATUS and PCLATH.

Software Stacks, Push and Pop Macros: A Physical Analogy -- Coin Storage

Push 1995 quarter onto top of stack

Pop 1996 quarter off top of stack



Initial State

After One Push

After Three More Pushes

After One Pop

Last quarter in is first quarter out (LIFO).

Stacks, Push and Pop Macros:

1. A **stack** is a last-in, first-out (LIFO) data storage structure.

2. Access to the data is controlled by two macros: "Push" and "Pop".

3. The "Push" macro is used to save data in memory.

4. The "Pop" macro is used to retrieve the data that was saved when the "Push" operation was executed.

5. The above is a **software concept** and is not the same as the hardware stack in the PIC architecture.

6. The hardware stack is used to save the Program Counter (PC) during subroutine calls and interrupts.

Example: Push and Pop Macros

```
cblock          0x20            ; Bank 0 assignments
                WTemp           ; Create temporary variables
                StatusTemp
endc
```

Save W and STATUS Register:

```
push macro
     banksel    WTemp           ; Select WTemp bank (Bank 0)
     movwf      WTemp           ; Copy W to WTemp
     movf       STATUS, W       ; Copy STATUS to W
     movwf      StatusTemp      ; Copy W to StatusTemp
     endm
```

Restore W and STATUS Register:

```
pop  macro
    banksel     StatusTemp      ; Select the StatusTemp (Bank 0)
    movf        StatusTemp, W   ; Copy StatusTemp to W
    movwf       STATUS          ; Copy W to STATUS
    movf        WTemp, W        ; Copy WTemp to W
    endm
```

But there is a problem with the push macro . . .

Example: Push and Pop Macros

Problem: The `banksel` directive might change the `STATUS` register before we save it in the StatusTemp register..

How can we avoid using the `banksel` directive?

Push and Pop Macros from previous slide

```
push macro
     banksel     WTemp            ; Select WTemp bank (Bank 0)
     movwf       WTemp            ; Copy W to WTemp
     movf        STATUS, W        ; Copy STATUS to W
     movwf       StatusTemp       ; Copy W to StatusTemp
     endm
```

Saving Context

Solution: Use common memory.

1. `0x70 - 0x7F` are common or shared memory across all banks.

2. This means that if you write to memory location `0x70`, you automatically write to `0xF0`, `0x170` and `0x1F0`.

3. `0x70`, `0xF0`, `0x170` and `0x1F0` are reserved for the debugger, so we use `0x71` and `0x72`.

4. Define `WTemp` at data memory address `0x71` and `StatusTemp` at address `0x72`.

5. We don't need to use `banksel`.

	File Address		File Address		File Address		File Address
Indirect addr.(*)	00h	Indirect addr.(*)	128 80h	Indirect addr.(*)	256 100h	Indirect addr.(*)	384 180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h	General Purpose Register 16 Bytes	110h	General Purpose Register 16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
	20h 32		A0h 160		120h 288		1A0h 416
General Purpose Register 96 Bytes	6Fh	General Purpose Register 80 Bytes	EFh	General Purpose Register 80 Bytes	16Fh	General Purpose Register 80 Bytes	1EFh
WTemp	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
StatusTemp	7Fh		FFh		17Fh		1FFh
Bank 0	127	Bank 1	255	Bank 2	383	Bank 3	511

Saving Context

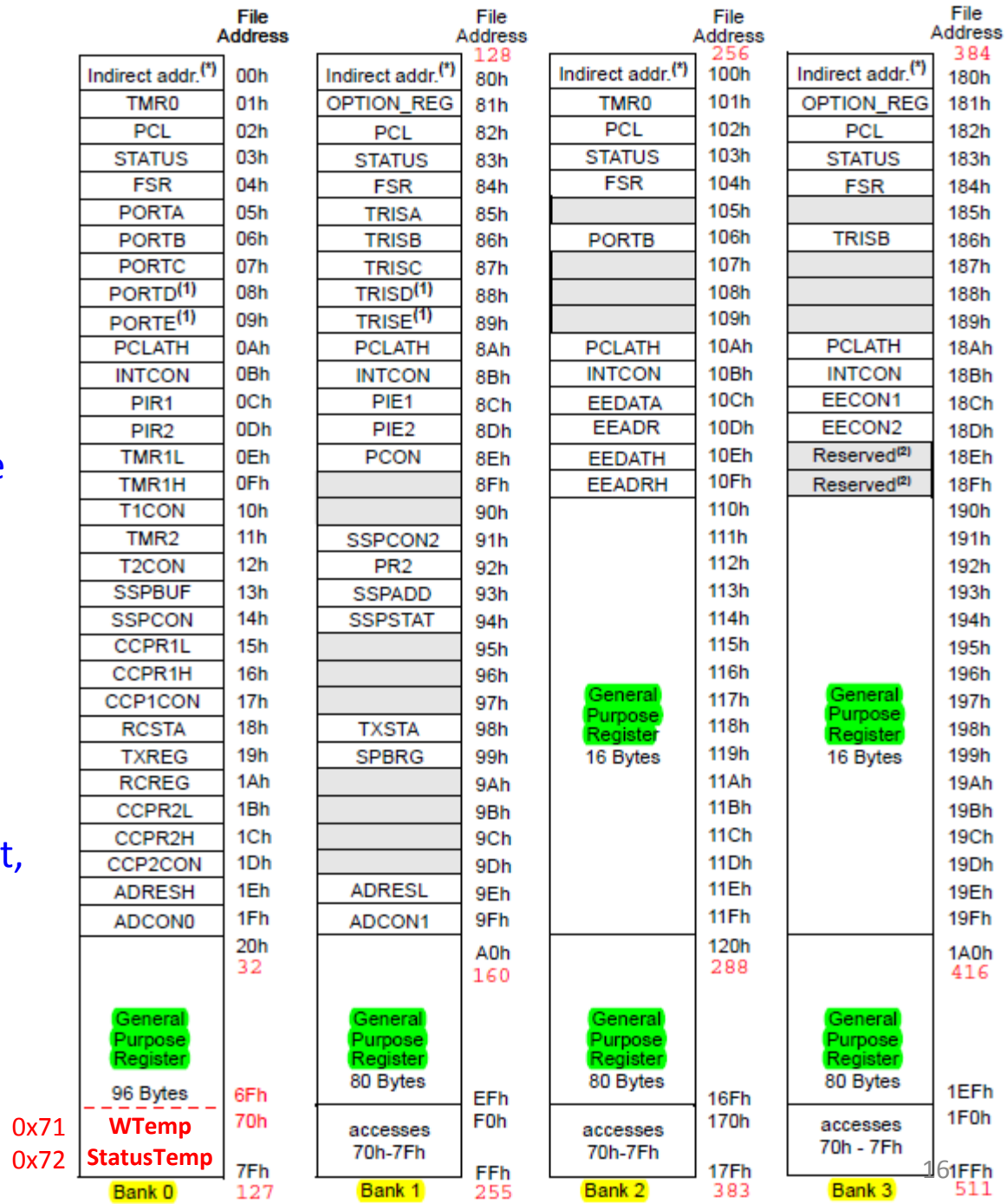
1. Code:

```
cblock  0x71
        WTemp
        StatusTemp
endc
```

2. We must be careful not to use the memory at the same relative address in the other banks:

   `0xF1, 0x171, and 0x1F1`
   `0xF2, 0x172, and 0x1F2`

3. See Page 130 in the data sheet, and Page 8-11 in the Mid-Range MCU Family Reference Manual for more information.



Example: Push and Pop Macros

```
cblock          0x71            ; Common memory assignments
                WTemp           ; Create temporary variables
                StatusTemp
endc
```

Save W and STATUS Register:

```
push macro
     movwf      WTemp           ; Copy W to WTemp
     movf       STATUS, W       ; Copy STATUS to W
     movwf      StatusTemp      ; Copy W to StatusTemp
     endm
```

Restore W and STATUS Register:

```
pop  macro
     movf       StatusTemp, W   ; Copy StatusTemp to W
     movwf      STATUS          ; Copy W to STATUS
     movf       WTemp, W        ; Copy WTemp to W
     endm
```

But there is a **second** problem with the push macro . . .

Example: Push and Pop Macros

```
push macro
      movwf       WTemp            ; Copy W to WTemp
      movf        STATUS, W        ; Copy STATUS to W
      movwf       StatusTemp       ; Copy W to StatusTemp
      endm
```

Problem:

The `movf STATUS, W` instruction might change the STATUS Z-bit.

Solution:

We need to avoid using `movf`. We will use a new instruction, `swapf`, which does not change the Z-bit.

MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2

`swapf` Instruction

New instruction: `swapf  f, d`

Example: `swapf  STATUS, F`

STATUS =

b7	b6	b5	b4	b3	b2	b1	b0

High nibble Low nibble

Swap nibbles

STATUS =

b3	b2	b1	b0	b7	b6	b5	b4

High nibble Low nibble

Example: Save and restore with common memory and `swapf`

Note: `movwf` and `swapf` do not affect the `STATUS` Z-bit.

```
; Assume STATUS = 1111 0000,  W = 1001 0110

push macro
     movwf     WTemp          ;      WTemp = 1001 0110
     swapf     STATUS, W      ;          W = 0000 1111
     movwf     StatusTemp     ; StatusTemp = 0000 1111
     endm
```

Interrupt

```
...
movlw   B'10101010'          ;      W = 1010 1010
movwf   STATUS               ; STATUS = 1010 1010
...
```

```
pop     macro                ;      W = 1010 1010
        swapf   StatusTemp, W ;      W = 1111 0000
        movwf   STATUS       ; STATUS = 1111 0000
        swapf   WTemp, F     ;  WTemp = 0110 1001
        swapf   Wtemp, W     ;      W = 1001 0110
        endm
```

Save and restore context using common memory and `swapf`.

```
cblock          0x71            ; Common memory assignments
                WTemp           ; 0x71: Temporary W variable
                StatusTemp      ; 0x72: Temporary STATUS variable
endc
```

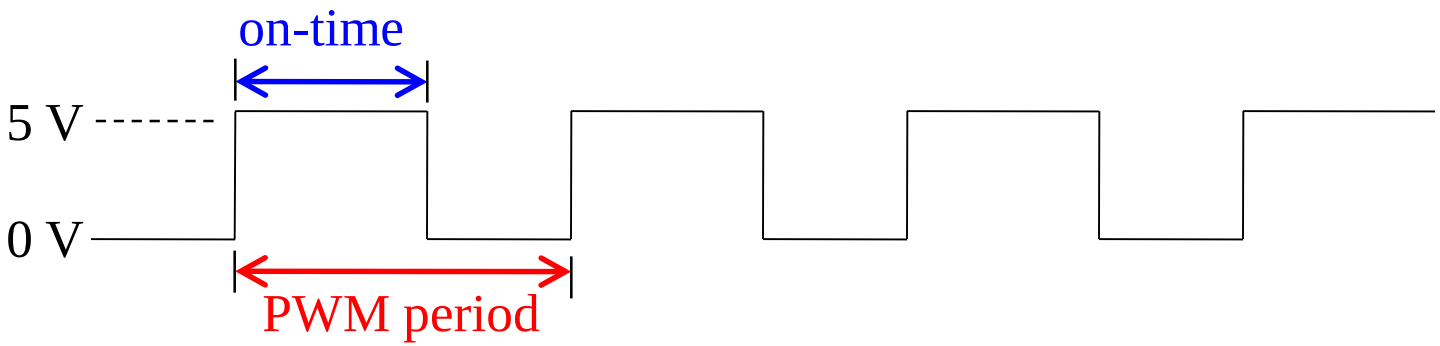
```
push macro
     movwf      WTemp           ; Copy W to WTemp in common memory
     swapf      STATUS, W       ; Swap STATUS nibbles and save in W
     movwf      StatusTemp      ; Copy STATUS to StatusTemp in common
     endm                       ; memory.
```

```
pop  macro
    swapf   StatusTemp,  W      ; Unswap StatusTemp register into W
    movwf   STATUS              ; Copy W into STATUS
                                ; (Sets bank to original state)
    swapf   WTemp,       F      ; Swap WTemp nibbles into WTemp
    swapf   WTemp,       W      ; Unswap WTemp into W
    endm
```

Lab 4 Outline

1. Macros
2. Saving Context During Interrupts
3. **Pulse Width Modulation**
4. Lab 4 Setup for PWM
5. Capture, Compare, PWM (CCP) Module
6. Lab 4 Setup for CCP

Pulse Width Modulation (PWM)



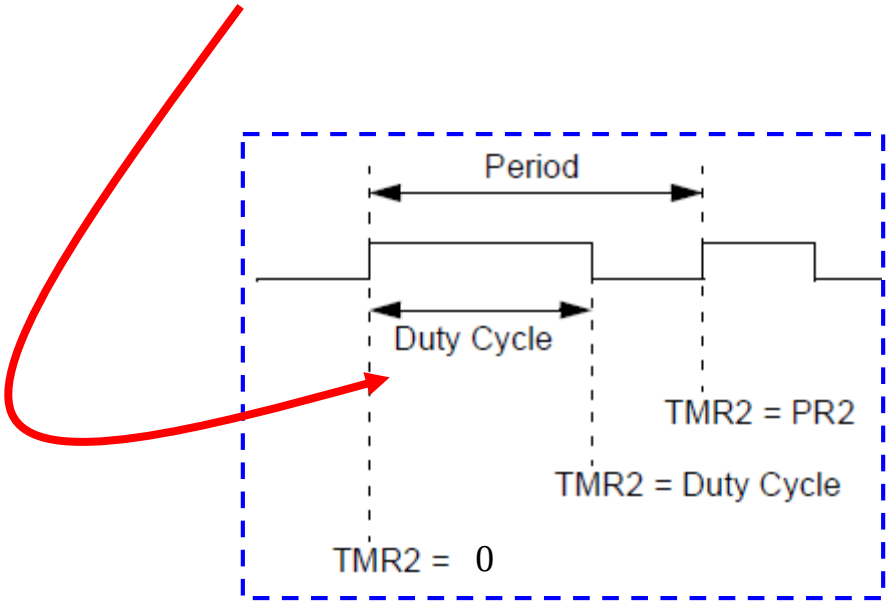
1. PWM is an efficient way of providing variable amounts of electrical power to a device by switching power fully on or fully off.

2. Average output votage = (on time / period) × 5 V

PWM Duty Cycle

The term "duty cycle" is used loosely in the datasheet:

1. Definition: **duty cycle** = on-time / PWM period (e. g., DC = 0.75 = 75%).
2. In the datasheet, “duty cycle” = on-time (in sec, e.g., DC = 200 μs).
3. In the datasheet, “duty cycle” = on-time (in timer ticks, e. g., DC = 51).

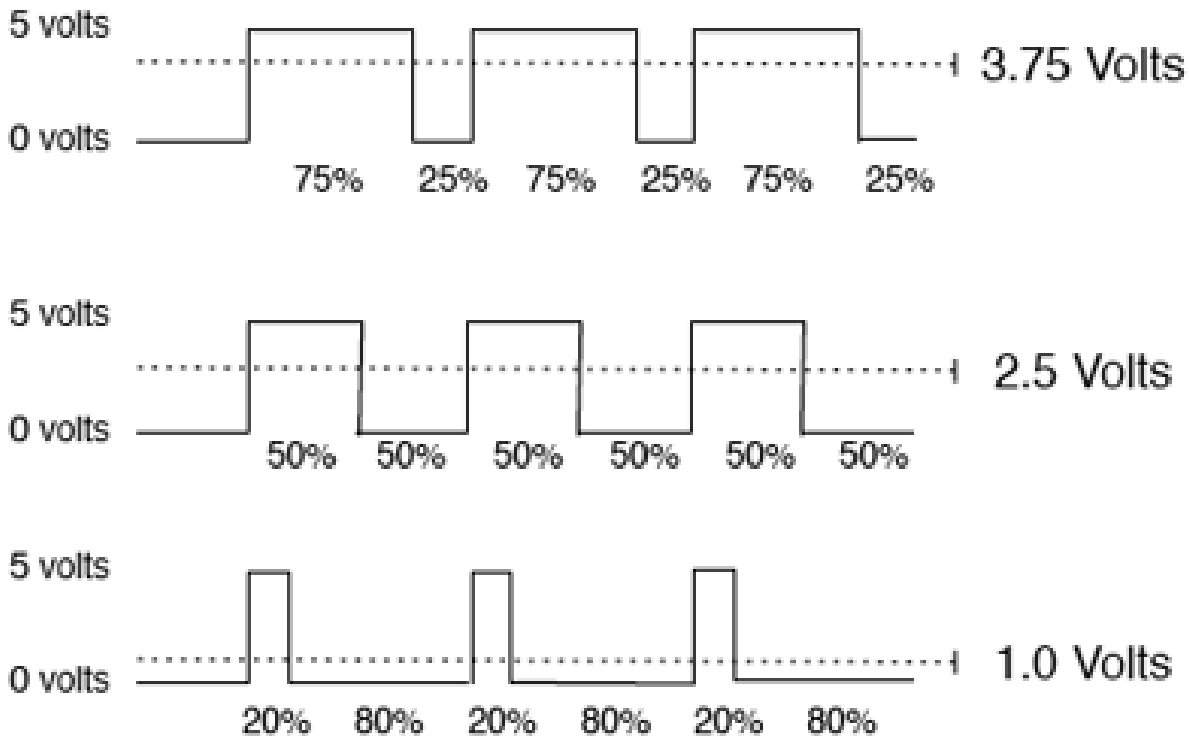


Data sheet, p. 58

Pulse Width Modulation

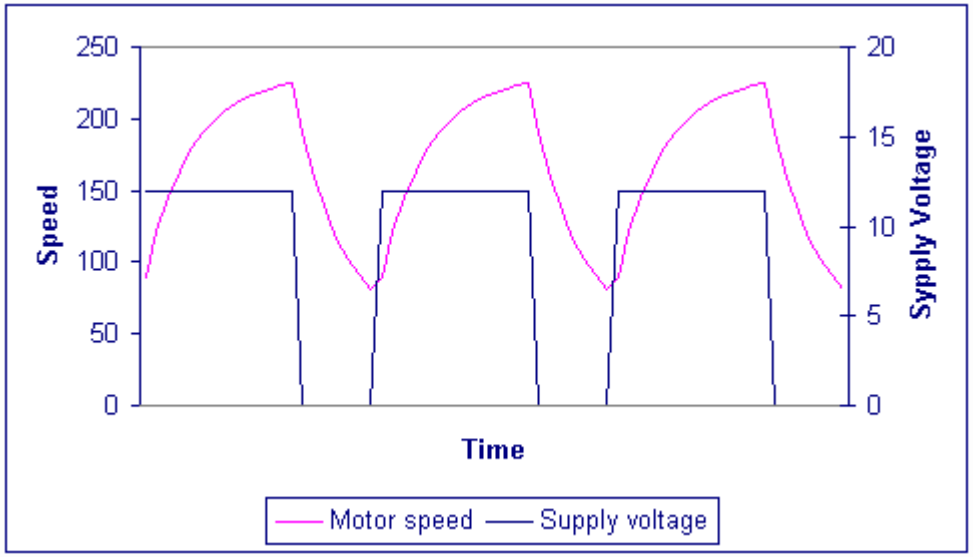
Average output = (on time / period) * 5 V

Duty Cycle = 75% →



Pulse Width Modulation

The **desired modulation frequency** depends on the response characteristics of the device.



Speed response of DC motor (large ripple).

If the supply voltage is switched fast enough, the motor won't have time to change speed much because of rotational inertia, friction, and load, so the speed will be approximately constant (small ripple).

Lab 4 Outline

1. Macros
2. Saving Context During Interrupts
3. Pulse Width Modulation
4. **Lab 4 Setup for PWM**
5. Capture, Compare, PWM (CCP) Module
6. Lab 4 Setup for CCP

lab04a
Programming a PWM signal

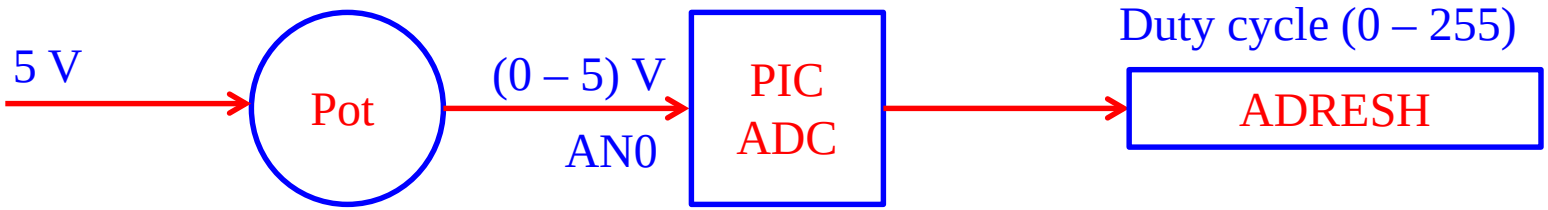
1. Pulse width modulate the LEDs on PORTC based on the AN0 analog voltage input.

2. The duty cycle is controlled by the voltage on AN0 from the pot. The LEDs will dim and brighten as the analog voltage on AN0 changes between 0 and 5 volts.

3. Timer0 interrupts are used to turn the PORTC output on and off at the appropriate duty cycle to achieve the PWM.

Lab04a (PWM)

Digital Duty Cycle



Example: If ADC input = 1volt, then

DutyCycle = ADRESH = (1/5) × (255) = 51

Lab04a (PWM)

Main Routine

```
;*****************************************************************
; Main Routine
;*****************************************************************

MAIN
    bsf         ADCON0, GO   ; Start A/D conversion

WaitForConversion

    btfss       PIR1, ADIF   ; Wait for conversion to complete

    goto        WaitForConversion

    ; Get the duty cycle

    bcf         PIR1, ADIF   ; Clear the A/D interrupt flag

    movf        ADRESH, W    ; Get A/D result

    movwf       DutyCycle    ; Copy A/D result to DutyCycle

    goto        MAIN         ; Repeat

;*****************************************************************
```

DutyCycle = ADRESH

Lab04a (PWM)
Timer0 Control Register Setup

Setup Timer0:

```
movlw     B'10000011'      ; TMR0 prescaler = 1:16
                           ; TMR0 counts one tick per 16
movwf     OPTION_REG       ; instruction cycles
```

1. Timer0 interrupts on rollover every
   $16 \times 256 = 4096$ instruction cycles or $4096 \times 1.085\ \mu s = 4.44$ ms.

2. We can use this as the PWM period, so the PWM frequency will be 1/4.44 ms = 225 Hz.

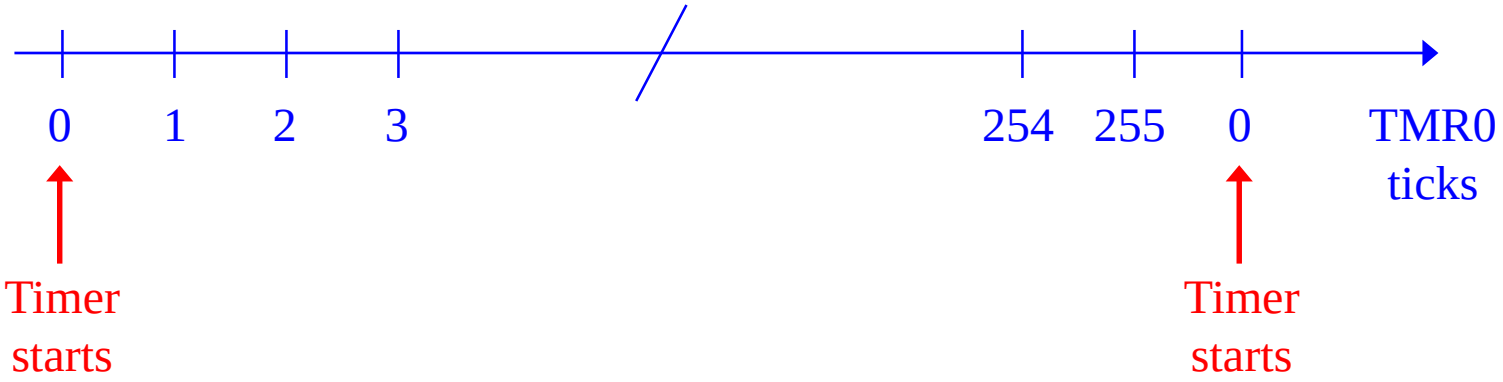
3. This is fast enough so that the human eye will not see the LEDs flicker.

Pulse Width Modulation

Digital PWM:	Duty Cycle = 0 - 255 (0% - 100%)
	Period = 256 (ticks)

TMR0

256 ticks (periods) from rollover to rollover



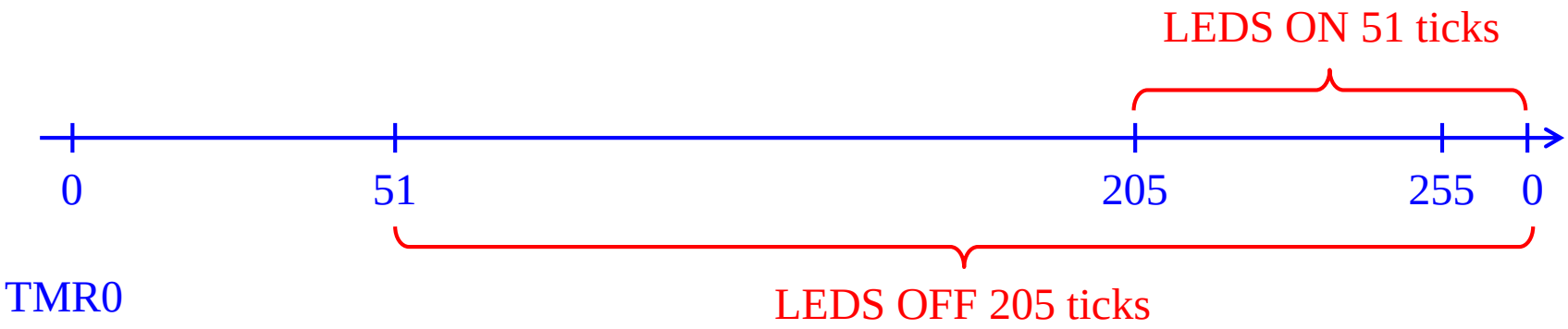
Lab04a (PWM)
Pulse Toggling using Timer0

1. Example: If DutyCycle = 51, the LEDs will be ON for 51 Timer0 ticks, and OFF for 256 – 51 = 205 ticks.

2. Each time Timer0 rolls over, an interrupt occurs and TMR0 = 0.

3. If we set TMR0 = 205 after an interrupt, and turn the LEDs ON, then the LEDs will remain on for 51 ticks until the timer rolls over.

4. After the rollover, if we set TMR0 = 51, and turn the LEDs OFF, then the LEDs will remain off for 205 ticks until the timer rolls over again.



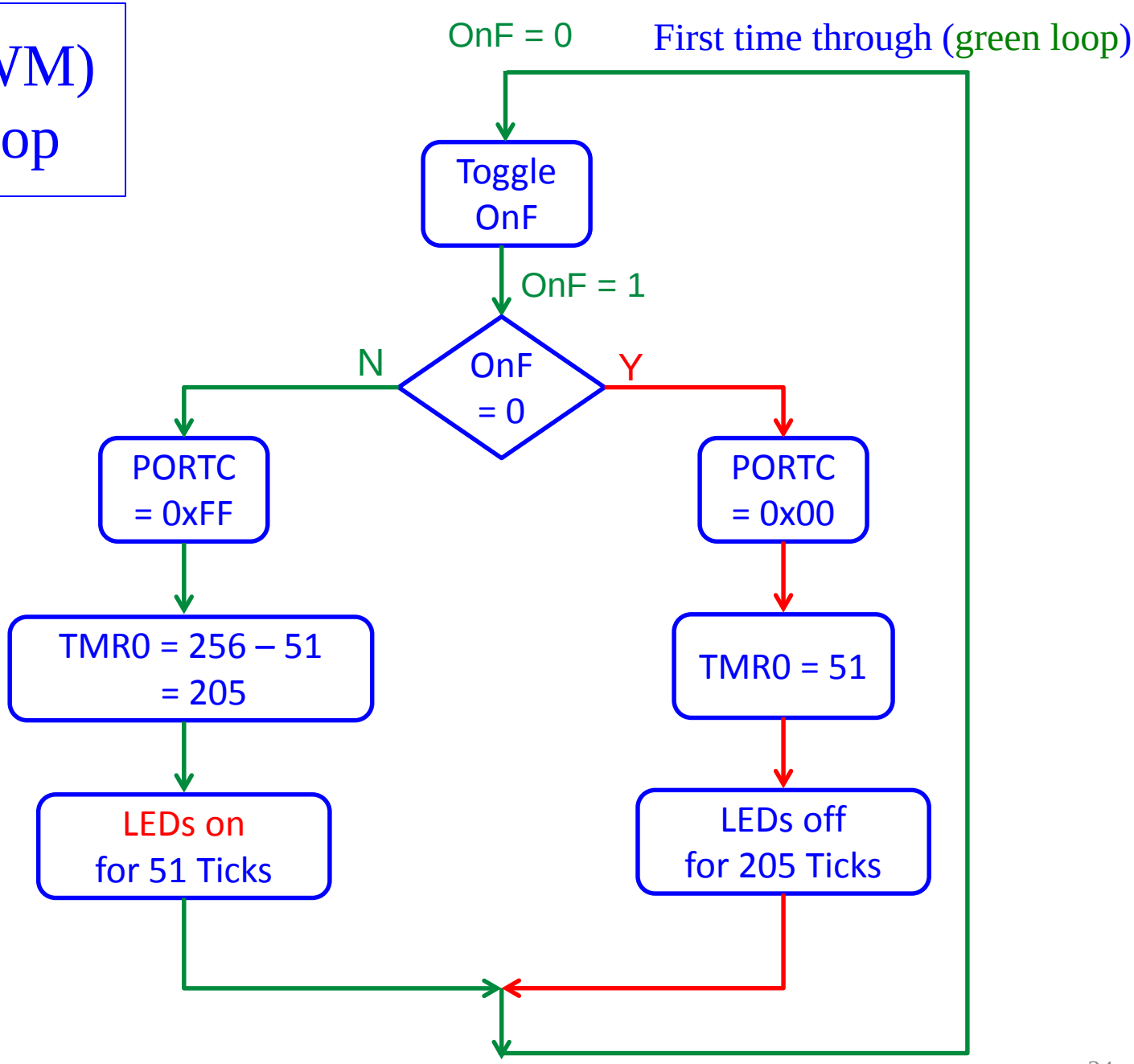
Lab04a (PWM) Toggle Loop

OnF: "OnFlag"

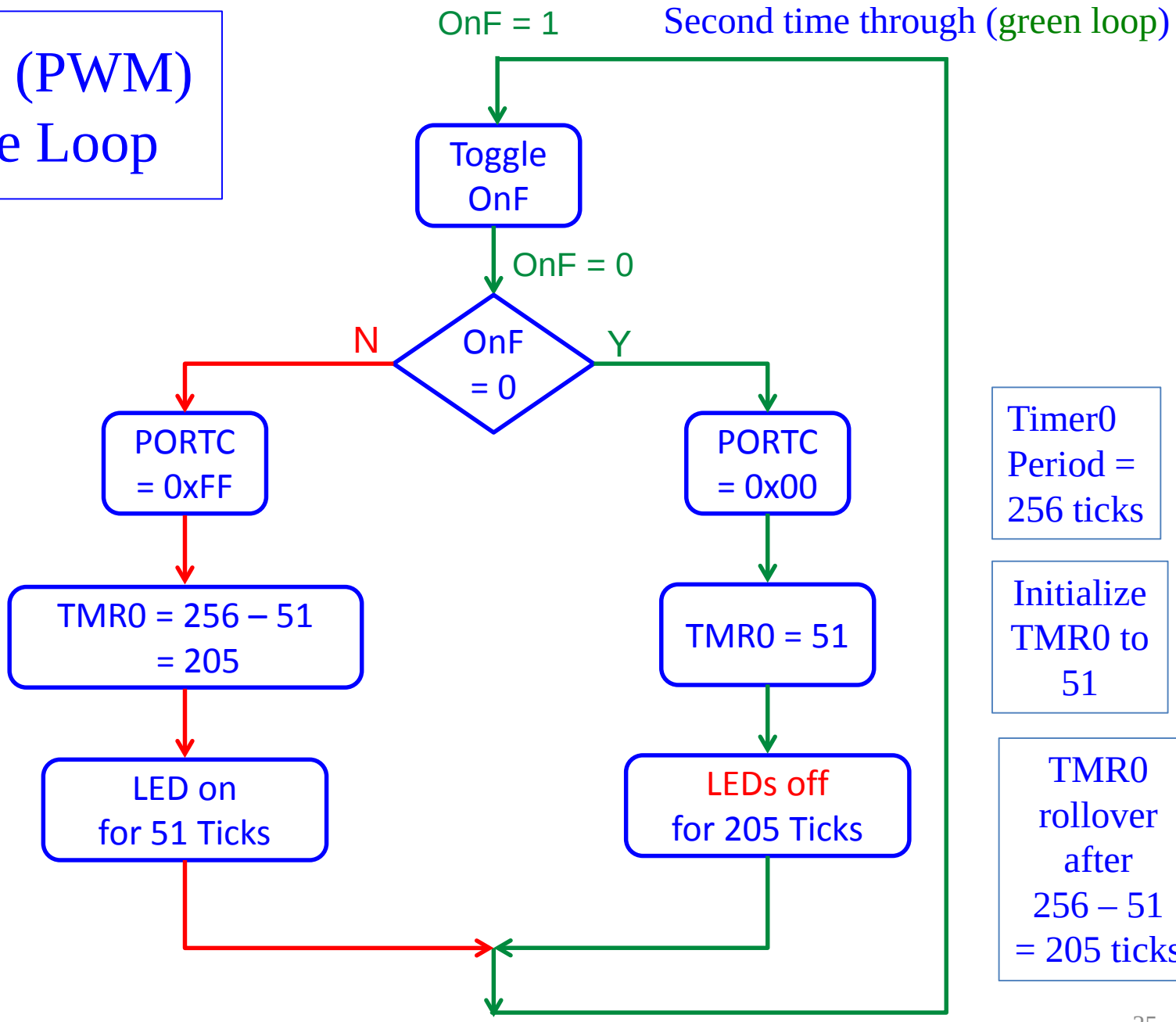
Timer0 Period = 256 ticks

Initialize TMR0 to 205

TMR0 rollover after 256 – 205 = 51 ticks



Lab04a (PWM) Toggle Loop



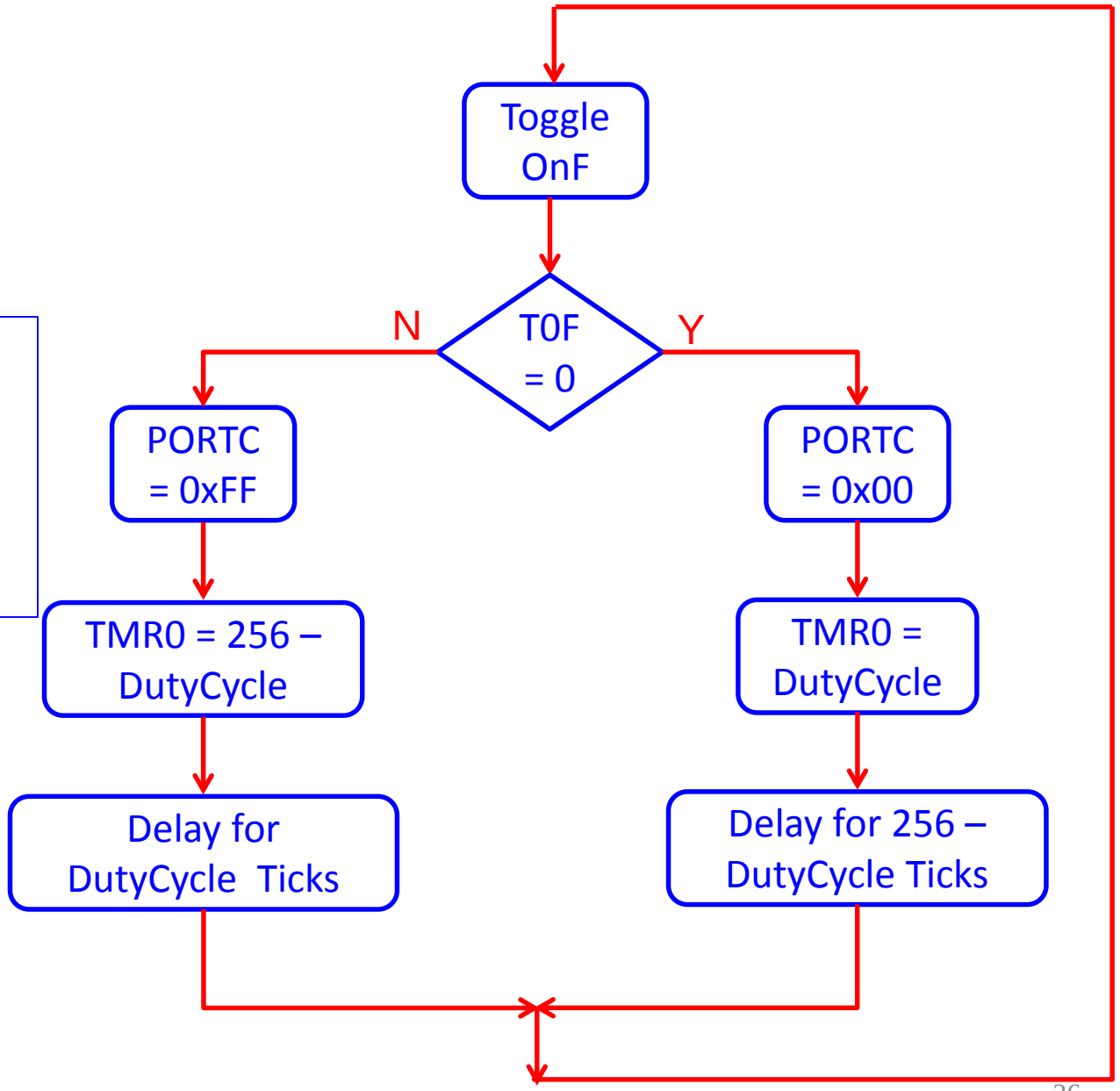
Timer0 Period = 256 ticks

Initialize TMR0 to 51

TMR0 rollover after 256 – 51 = 205 ticks

Lab04a Toggle Loop for a general duty cycle

1 complete cycle equals 2 loops: once through left loop, once through right loop.



Toggle Subroutine

We will need the `sublw` instruction: subtract L – W

SUBLW	Subtract W from Literal
Syntax:	[*label*] SUBLW k
Operands:	$0 \le k \le 255$
Operation:	k - (W) → (W)
Status Affected:	C, DC, Z
Description:	The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.

L – W → W

```
movlw .51      ; W = 51
sublw .0       ; W = L – W = 0 – 51 = 256 - 51 = 205
               ; Note 0 = 256 mod(256)
```

```
movlw .0       ; W = 0
sublw .51      ; W = L – W = 51 - 0 = 51
```

Toggle Subroutine

First time through the Toggle routine: OnFlag = 0x00 (INIT)

```
Toggle
    bcf     INTCON, T0IF
    comf    OnFlag, F       ; Toggle: OnFlag = 0xFF
    clrw                    ; Set W = 0
    btfsc   OnFlag, 0       ; Do not skip the next instruction
    movlw   0xFF            ; W = 0xFF = 1111 1111
    movwf   PORTC           ; PORTC = 0xFF (LEDs on)
    movf    DutyCycle,  W   ; W = DutyCycle = 51 (assuming ADC
                            ; input = 1 V)
    btfsc   OnFlag, 0       ; Do not skip the next instruction
    sublw   0x00            ; W = L – W = 0 – 51 = -51 mod(256) = 205
    movwf   TMR0            ; TMR0 = 205 (TMR0 starts at 205)
    goto    Poll_Int_Flags  ; Poll T0IF until Timer0 interrupt
```

Summary:

1. Turn on PORTC LEDs
2. Wait for 256 – 205 = 51 TMR0 ticks

Toggle Subroutine

Second time through the Toggle routine: OnFlag = 0xFF

```
Toggle
    bcf     INTCON,  T0IF
    comf    OnFlag, F       ; Toggle: OnFlag = 0x00
    clrw                    ; W = 0
    btfsc   OnFlag, 0       ; Skip the next instruction
    movlw   0xFF
    movwf   PORTC           ; PORTC = W = 0 (LEDs off)
    movf    DutyCycle,  W   ; W = DutyCycle = 51 (assuming ADC
                            ; input = 1 V)
    btfsc   OnFlag, 0       ; Skip the next instruction
    sublw   0x00
    movwf   TMR0            ; TMR0 = W = 51 (TMR0 starts at 51)
    goto    Poll_Int_Flags  ; Poll T0IF until Timer0 interrupt
```

Summary:

1. Turn off PORTC LEDs
2. Wait for 256 – 51 = 205 TMR0 ticks

Lab 4 Outline

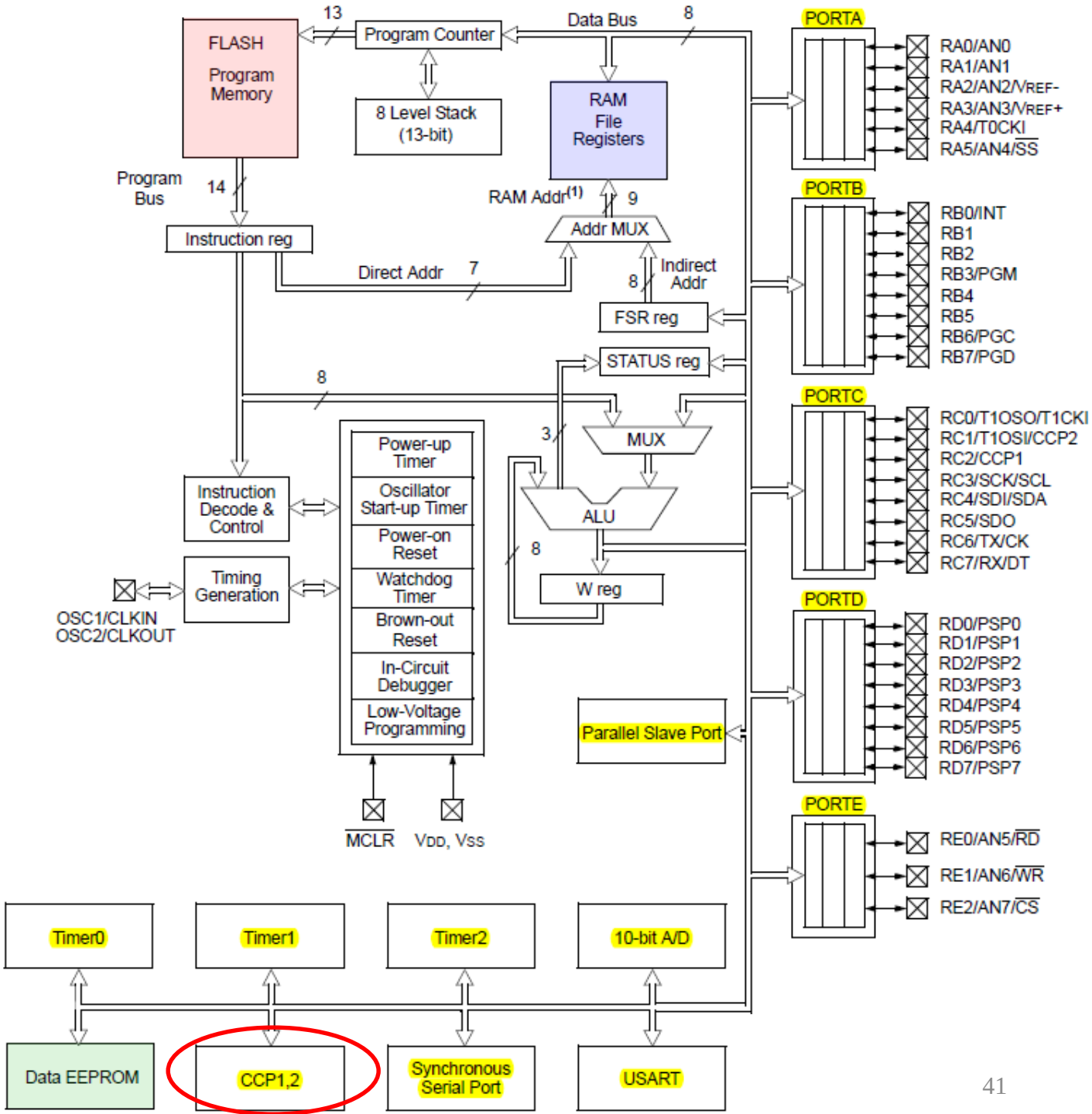
1. Macros
2. Saving Context During Interrupts
3. Pulse Width Modulation
4. Lab 4 Setup for PWM
5. **Capture, Compare, PWM (CCP) Module**
6. Lab 4 Setup for CCP

PIC 16F877 Architecture

$2^{13} = 8192$
program memory addresses
(Flash EEPROM)

$2^{9} = 512$
data memory addresses
(SRAM)

$2^{8} = 256$
data memory addresses
(EEPROM)

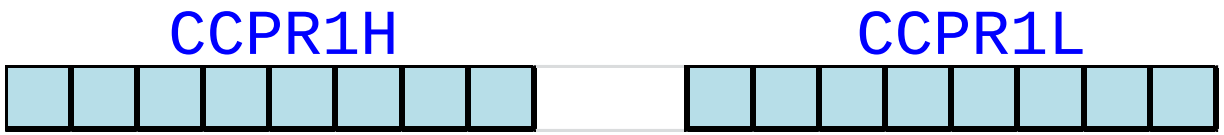


Data sheet p. 6

Capture/Compare/PWM (CCP) module

1. There are two CCP modules in the PIC (CCP1 and CCP2).

2. Each contains a 16-bit register (2 concatenated 8-bit registers `CCPR1H:CCPR1L`).



3. The CCP modules can operate in three different modes:

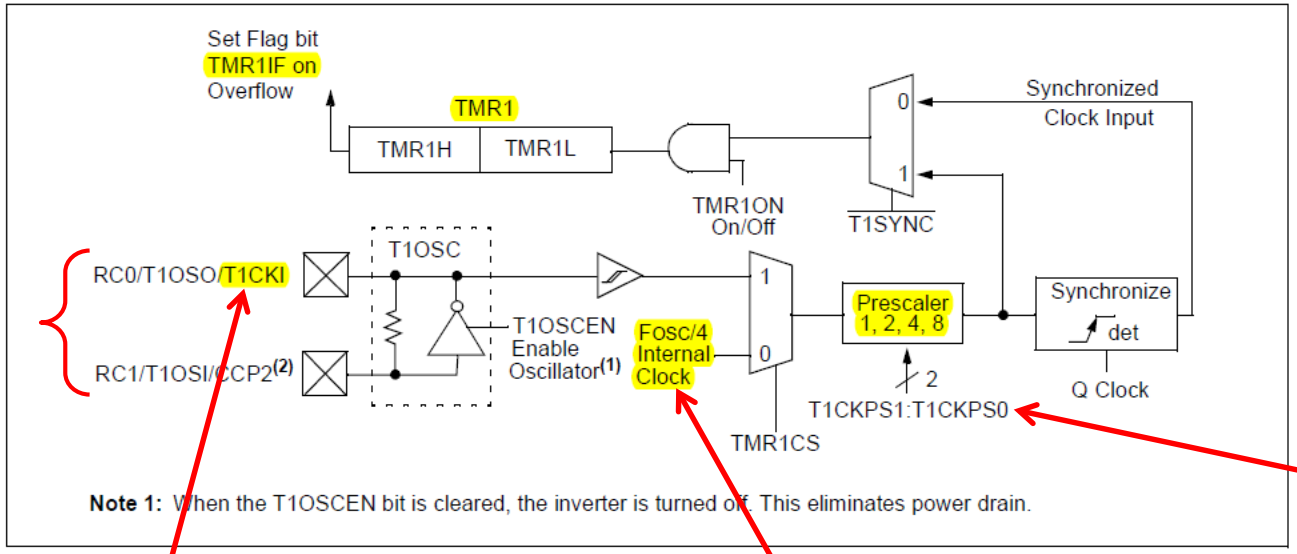
   - 16-bit Capture mode is used to time external events.
   - 16-bit Compare mode is used as a timer.
   - 10-bit PWM mode is used to generate PWM signals.

4. We will only look at the Compare and PWM modes.

Timer1 Block Diagram

16-bit timer
0000h to FFFFh
0 – 65535

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



We will not use

T1CON

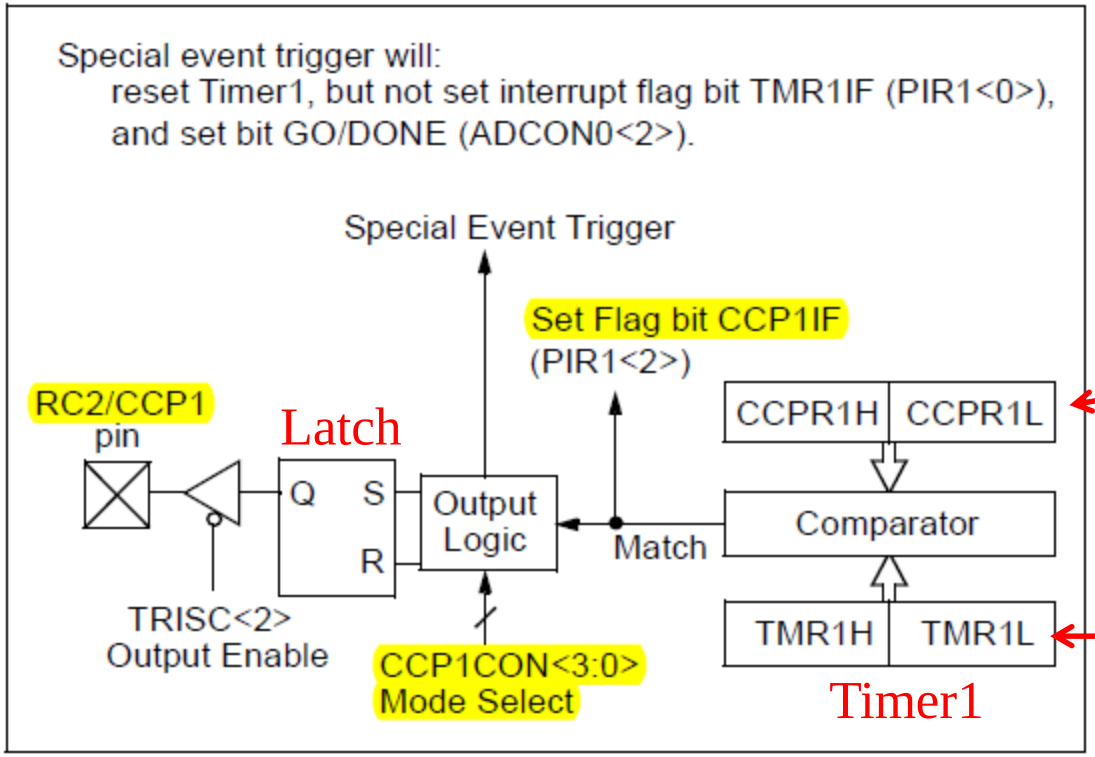
External stimulus: When used in the capture mode as a counter on RC0, increments every rising edge.

Internal stimulus: When used in the compare mode as a timer, increments every instruction cycle.

Datasheet p. 52

CCP1 - Compare Mode (Used as 16-bit Timer1)

CCP1



Used as Period Register

Match when

`TMR1H:TMR1L = CPR1H:CCPR1L`

Fosc/4

In decimal

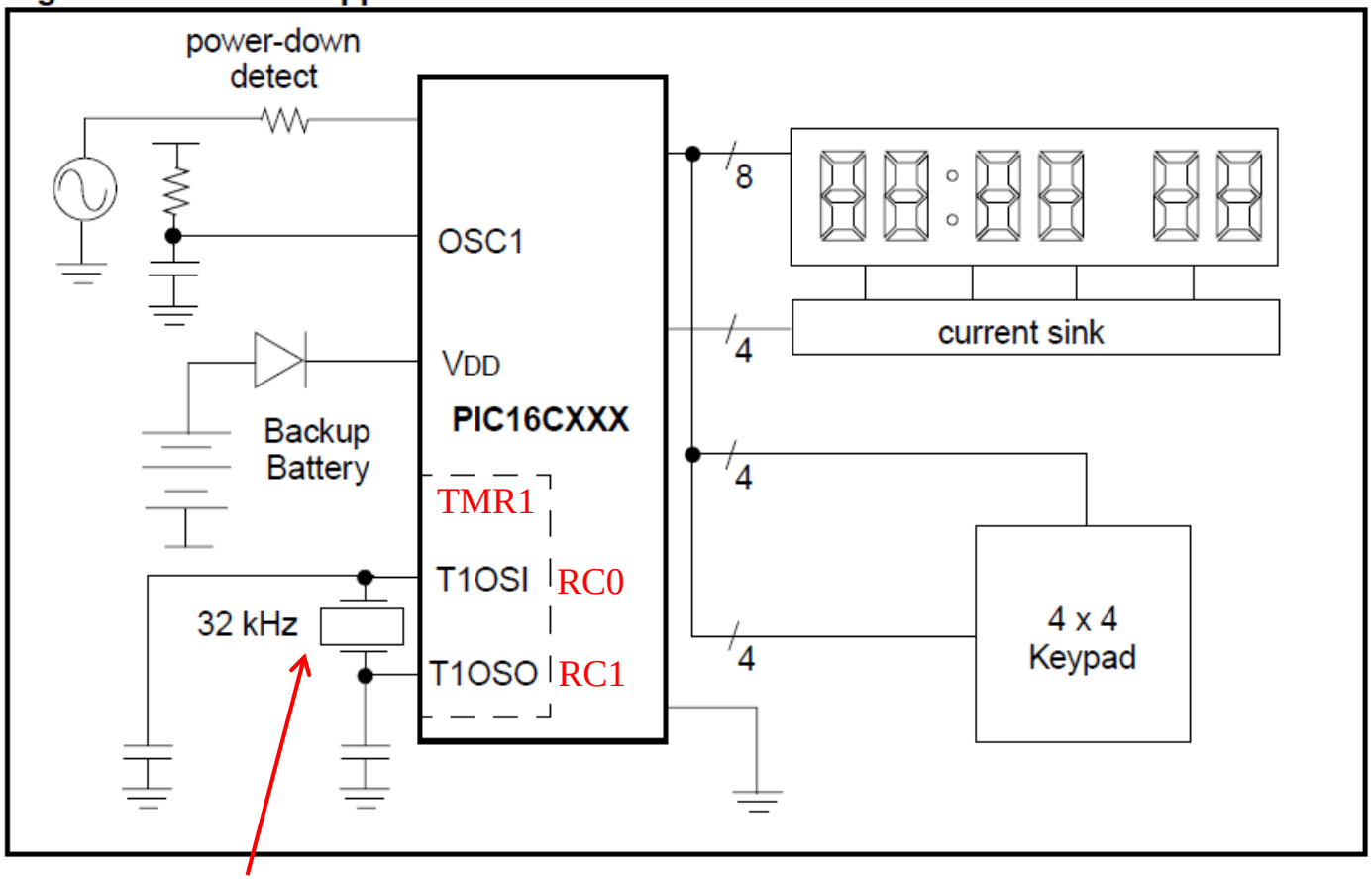
CCP1 interrupt period = (4 Tosc) (T1 Prescale) (CCPR1H:CCPR1L)

Datasheet p. 60

Timer1 with External Oscillator

Real-Time Clock Application

Figure 12-2: Timer1 Application



32 kHz Crystal Oscillator - 3.6864 MHz / 32 kHz = 115 times slower

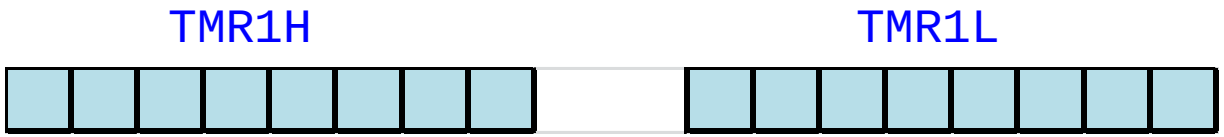
T1OSI: Timer1 Oscillator input

Mid-Range Manual, p. 12-8

Compare Mode (CCP1)

1. Compare mode uses Timer1

2. `TMR1H:TMR1L` forms a 16-bit timer/counter and increments every Prescale instruction cycles



3. When `TMR1H:TMR1L = CCPR1H:CPR1L`

   - CCP1 interrupt (if enabled)
   - Set the CCP1IF interrupt flag
   - Set or clear the CCP1 pin (RC2)
   - Reset TMR1H : TMR1L

4. Same for CCP2

CCP Module

Uses Timer1

CCP Control Registers

	File Address		File Address		File Address		File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h	General Purpose Register 16 Bytes	110h	General Purpose Register 16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
	20h 32		A0h 160		120h 288		1A0h 416
General Purpose Register 96 Bytes	6Fh	General Purpose Register 80 Bytes	EFh	General Purpose Register 80 Bytes	16Fh	General Purpose Register 80 Bytes	1EFh
	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Bank 0	127	Bank 1	255	Bank 2	383	Bank 3	511

Lab 4 Outline

1. Macros
2. Saving Context During Interrupts
3. Pulse Width Modulation
4. Lab 4 Setup for PWM
5. Capture, Compare, PWM (CCP) Module
6. **Lab 4 Setup for CCP**

Lab04c

1. Uses the compare function of the CCP module to toggle the RC2 output.

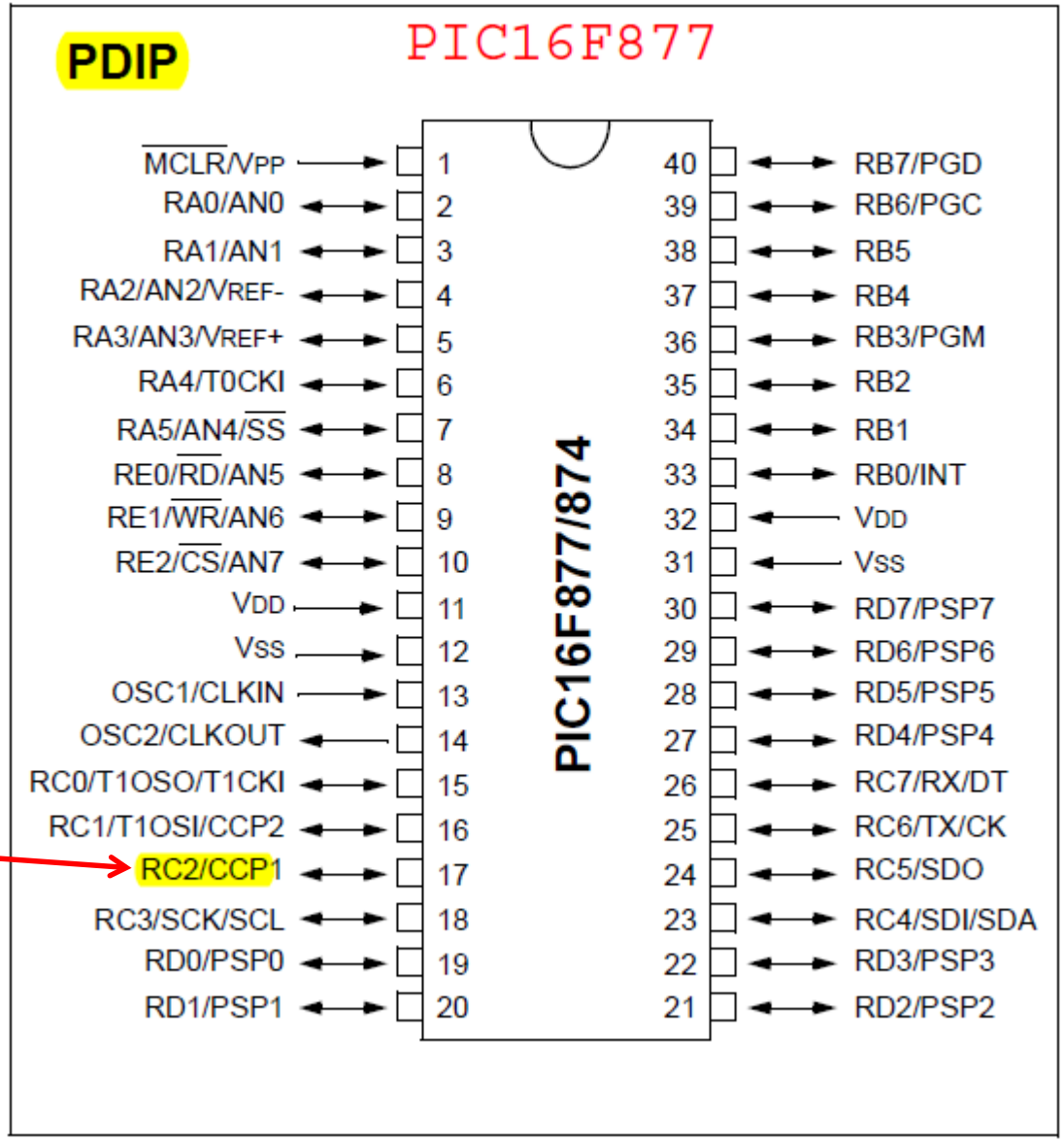
2. Toggles the CCP1/RC2 pin every 1/2 second.

3. The variable OnFlag specifies if the CCP1 output will be set high or low at the next CCP1 interrupt.

lab04c:

Toggle CCP1 output on RC2 to blink the third LED.

Pin Diagram



Compare Mode

REGISTER 8-1: CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS: 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

lab04c toggles these modes

Output on RC2

Datasheet p. 58

Lab04c

```
INIT

    banksel TRISC        ; Set register access to bank 1
    movlw   B'11111011'  ; Set up RC2 as output for LED
    movwf   TRISC
    bsf     INTCON, GIE  ; Enable the CCP1 interrupt
    bsf     INTCON, PEIE
    bsf     PIE1, CCP1IE
    banksel PORTC        ; Set register access back to bank 0
    bsf     PORTC, 2     ; Set RC2 = 1 (LED on)
    movlw   B'00110001'  ; Enable Timer1 with a 1:8 prescale
    movwf   T1CON
    movlw   0xF4
    movwf   CCPR1H       ; Set Timer1 match to occur after 0xF424 =
                         ; 62500 ticks (62500 x 8 = 500000 cycles =
    movlw   0x24         ; 500000 x  1.085 µs = 0.543 sec)
    movwf   CCPR1L       ;
    movlw   B'00001001'  ; Clear RC2/CCP1 on match (CCP1 interrupt)
    movwf   CCP1CON
    clrf    OnFlag       ; OnFlag = 0x00
    return
```

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

T1CON = 0011 0001

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit

1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit

When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit

1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON**: Timer1 On bit

1 = Enables Timer1
0 = Stops Timer1

Datasheet p. 51

Compare Mode

Before first CCP1 interrupt: OnFlag = 0x00, RC2 = 1. After interrupt: Go to Toggle, RC2 = 0

```
Toggle
    bcf     PIR1, CCP1IF  ; Clear the interrupt flag
    comf    OnFlag, F     ; Toggle: OnFlag = 0xFF
    clrf    TMR1H         ; Reset TMR1
    clrf    TMR1L         ; TMR1H:TMR1L = 0x0000
    movlw   B'00001001'   ; RC2 = 0 on next interrupt
    btfsc   OnFlag, 0     ; Do not skip the next instruction
    movlw   B'00001000'   ; RC2 = 1 on next interrupt
    movwf   CCP1CON       ; CCP1CON = 0000 1000
    goto    Poll          ; Poll for a CCP1 interrupt
```

1. When the program starts, RC2 = 1 (from INIT)

2. After 1/2 second, we get the first CCP1 interrupt, which sets RC2 = 0 because CCP1CON = 0000 1001 (from INIT).

3. The Toggle routine is entered and CCP1CON is set to 0000 1000

Compare Mode

Second CCP1 interrupt: goto Toggle routine: OnFlag = 0xFF, RC2 = 1 (after second interrupt)

```
Toggle
   bcf    PIR1, CCP1IF  ; Clear the interrupt flag
   comf   OnFlag, F     ; Toggle: OnFlag = 0x00
   clrf   TMR1H         ; Reset TMR1
   clrf   TMR1L         ; TMR1H:TMR1L = 0x0000
   movlw  B'00001001'   ; RC2 = 0 on next interrupt
   btfsc  OnFlag, 0     ; Skip the next instruction
   movlw  B'00001000'   ; RC2 = 1 on the next interrupt
   movwf  CCP1CON       ; CCP1CON = 0000 1001
   goto   Poll          ; Poll for a CCP1 interrupt
```

1. When we get the second interrupt, RC2 = 0 for the past 1/2 second.

2. This interrupt causes RC2 to be set (RC2 = 1).

CCP in PWM Mode
Next Lab

End of Lab 4