11002 GS2

Getting Started with PIC® MCU Mid-Range

Architecture, Instruction Set and Assembly Language Programming

Class Objective

When you finish this class you will:

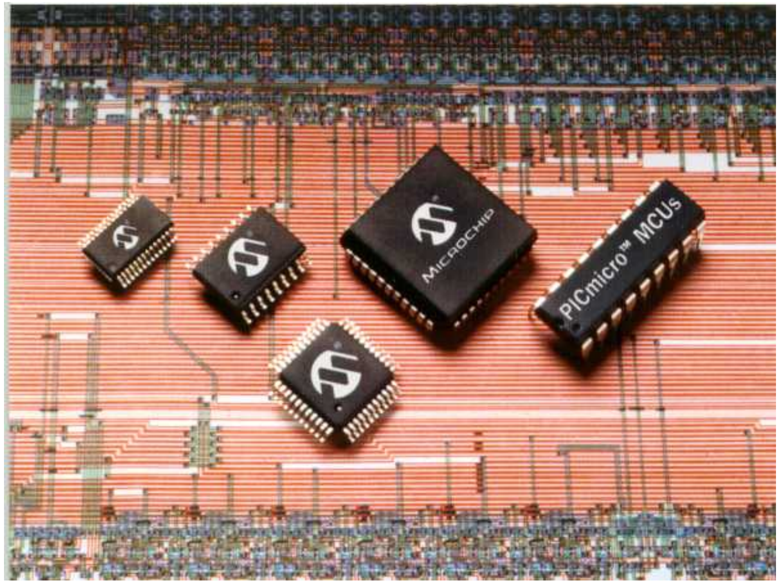
- Understand the basics of the inner workings of a PIC16
- Understand most instructions
- Understand memory organization
- Understand how to write simple programs

Agenda

- Architecture Basics
- Instruction Set Overview
- Memory Organization and Addressing Modes
- Special Features
- Hands-on Exercises

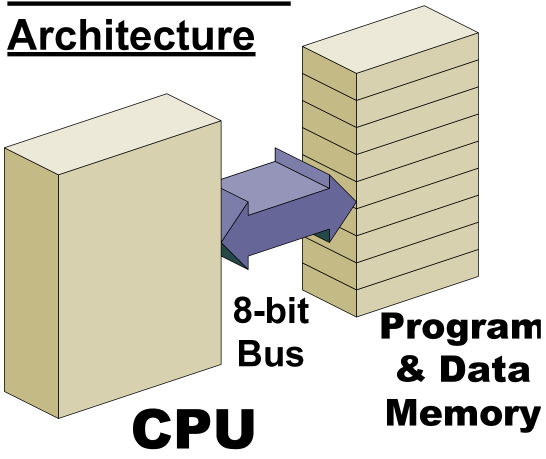
Architecture

- **The high performance of the PIC® microcontroller can be attributed to the following architectural features:**
  - Harvard Architecture
  - Instruction Pipelining
  - Large Register File
  - Single Cycle Instructions
  - Single Word Instructions
  - Long Word Instructions
  - Reduced Instruction Set
  - Orthogonal Instruction Set

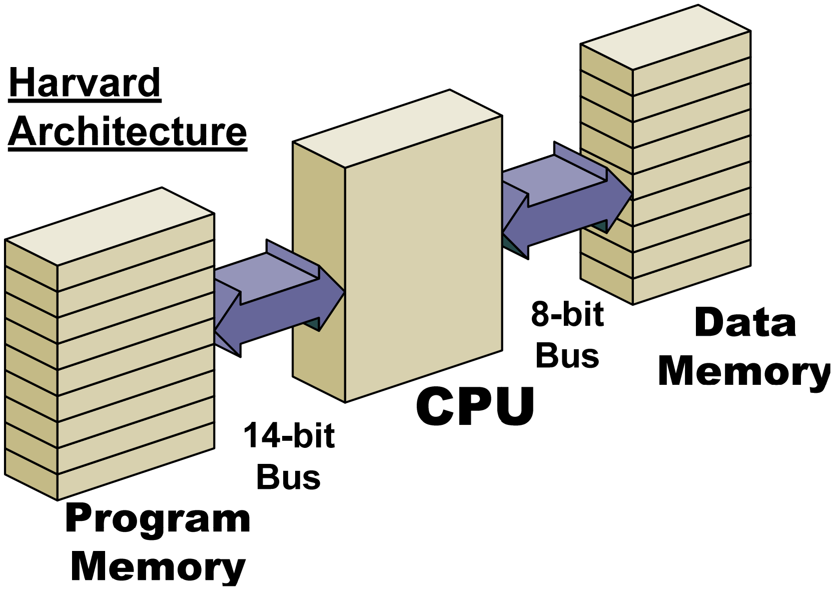


Harvard Architecture

Von Neumann Architecture



Harvard Architecture



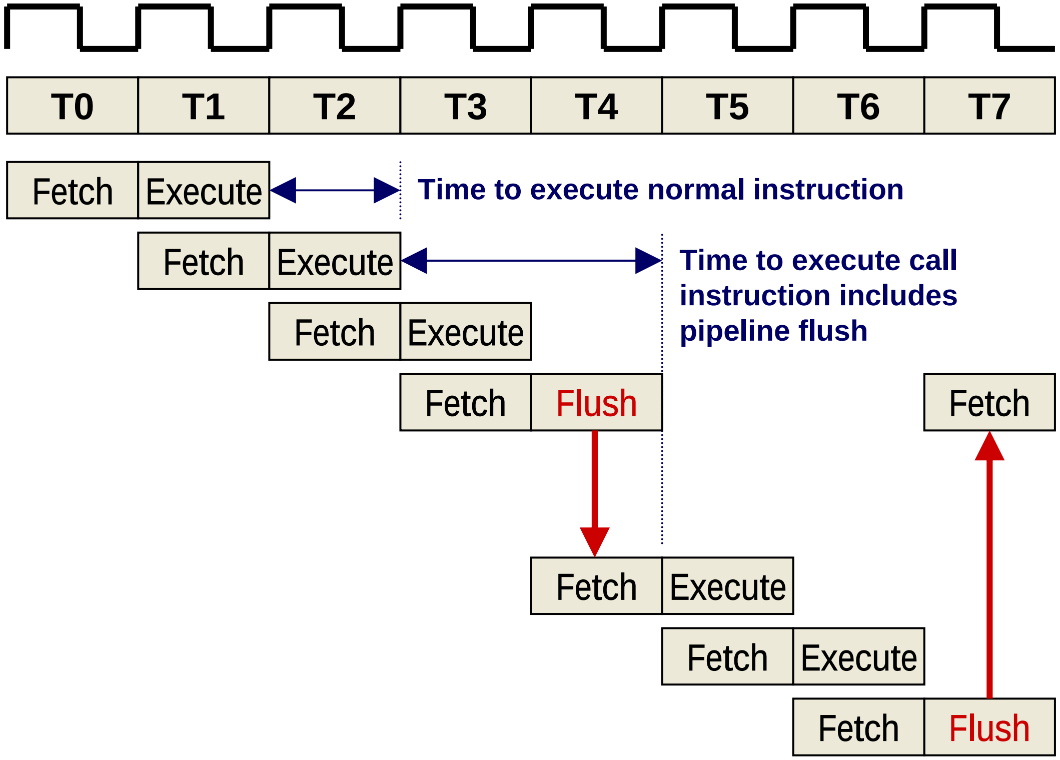
- **Von Neumann Architecture:**
  - Fetches instructions and data from a single memory space
  - Limits operating bandwidth
- **Harvard Architecture:**
  - Uses two separate memory spaces for program instructions and data
  - Improved operating bandwidth
  - Allows for different bus widths

Instruction Pipelining

- **Instruction fetch is overlapped with execution of previously fetched instruction**

```
1   MAIN movlw  0x05
2        movwf  REG1
3        call   SUB1
4        addwf  REG2
          ⋮
51  SUB1 movf   PORTB,w
52       return
53  SUB2 movf   PORTC,w
54       return
```

Instruction Cycles



T0 T1 T2 T3 T4 T5 T6 T7

Fetch Execute

Time to execute normal instruction

Fetch Execute

Time to execute call instruction includes pipeline flush

Fetch Execute

Fetch Flush

Fetch

Fetch Execute

Fetch Execute

Fetch Flush

Instruction Pipelining

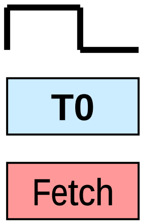
Pre-Fetched Instruction

```
movlw 0x05
```

Executing Instruction

```
-
```

Instruction Cycles



Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call    SUB1
4         addwf   REG2
              ⋮
51  SUB1  movf    PORTB,w
52        return
53  SUB2  movf    PORTC,w
54        return
```

Instruction Pipelining

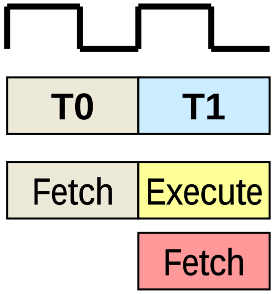
Pre-Fetched Instruction

```
movwf REG1
```

Executing Instruction

```
movlw 0x05
```

Instruction Cycles



Example Program

```
 1  MAIN movlw  0x05
 2       movwf  REG1
 3       call   SUB1
 4       addwf  REG2


51  SUB1 movf   PORTB,w
52       return
53  SUB2 movf   PORTC,w
54       return
```

Instruction Pipelining

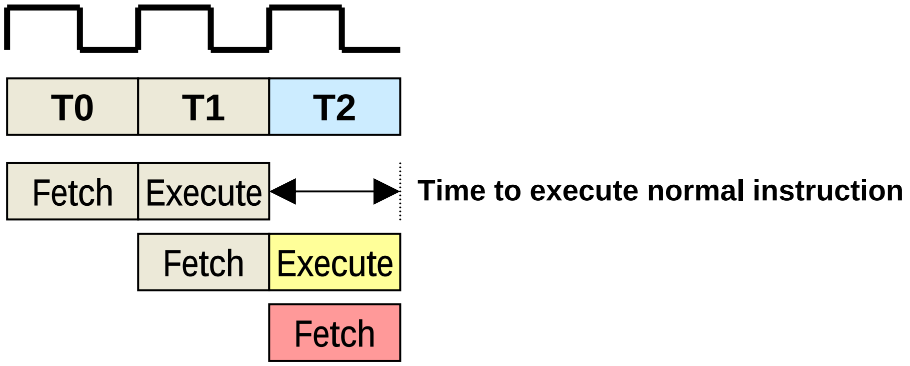
Pre-Fetched Instruction

```
call SUB1
```

Executing Instruction

```
movwf REG1
```

Instruction Cycles



T0	T1	T2
Fetch	Execute	
	Fetch	Execute
		Fetch

Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call    SUB1
4         addwf   REG2

51  SUB1  movf    PORTB,w
52        return
53  SUB2  movf    PORTC,w
54        return
```

Instruction Pipelining

Pre-Fetched Instruction

```
addwf  REG2
```

Executing Instruction

```
call  SUB1
```

Instruction Cycles

T0	T1	T2	T3
Fetch	Execute		
	Fetch	Execute	
		Fetch	Execute
			Fetch

Example Program

```
 1  MAIN movlw   0x05
 2       movwf   REG1
 3       call    SUB1
 4       addwf   REG2

51  SUB1 movf    PORTB,w
52       return
53  SUB2 movf    PORTC,w
54       return
```

Instruction Pipelining

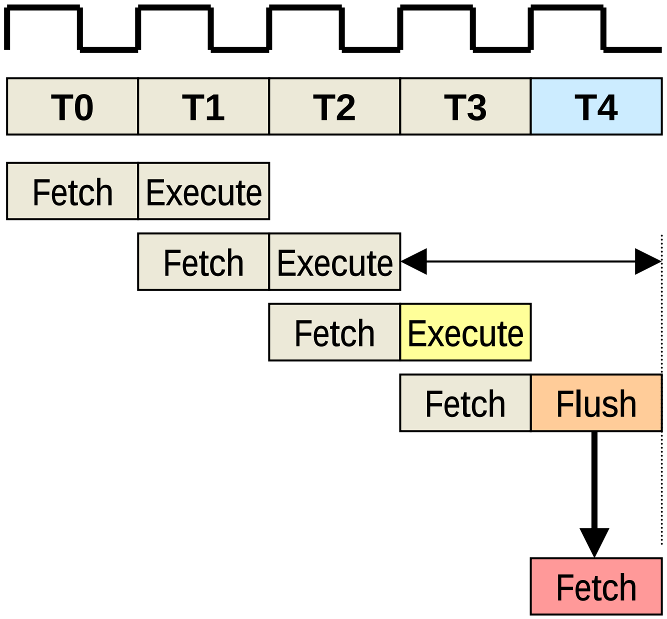
Pre-Fetched Instruction

```
movf PORTB,w
```

Executing Instruction

```
call SUB1
```

Instruction Cycles



Time to execute call instruction includes pipeline flush

Example Program

```
 1  MAIN movlw   0x05
 2       movwf   REG1
 3       call    SUB1
 4       addwf   REG2

51  SUB1 movf    PORTB,w
52       return
53  SUB2 movf    PORTC,w
54       return
```

Instruction Pipelining

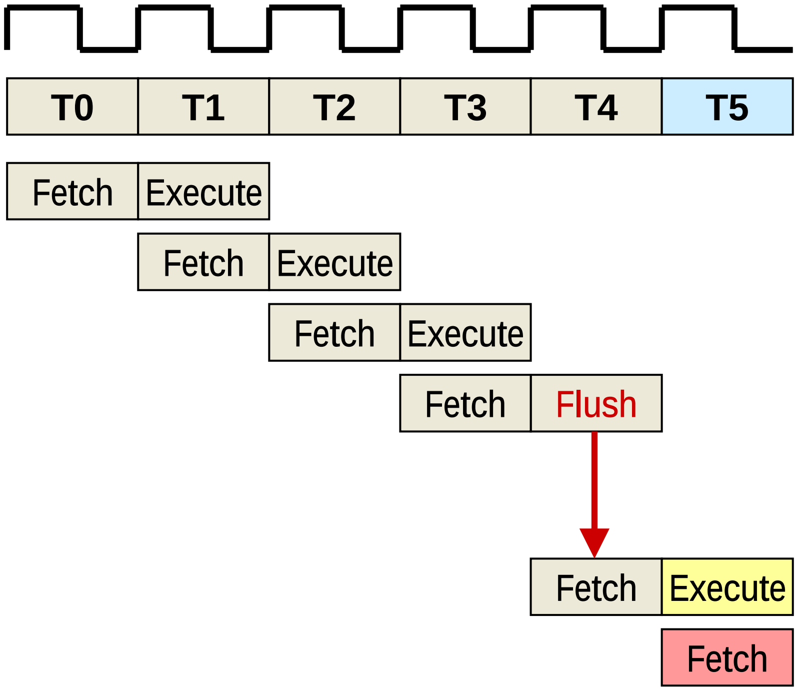
Pre-Fetched Instruction

`return`

Executing Instruction

`movf PORTB,w`

Instruction Cycles



T0	T1	T2	T3	T4	T5
Fetch	Execute				
	Fetch	Execute			
		Fetch	Execute		
			Fetch	Flush	
				Fetch	Execute
					Fetch

Example Program

```
 1  MAIN movlw  0x05
 2       movwf  REG1
 3       call   SUB1
 4       addwf  REG2

51  SUB1 movf   PORTB,w
52       return
53  SUB2 movf   PORTC,w
54       return
```

University of Microchip
UofM
MASTERs 2007

Instruction Pipelining

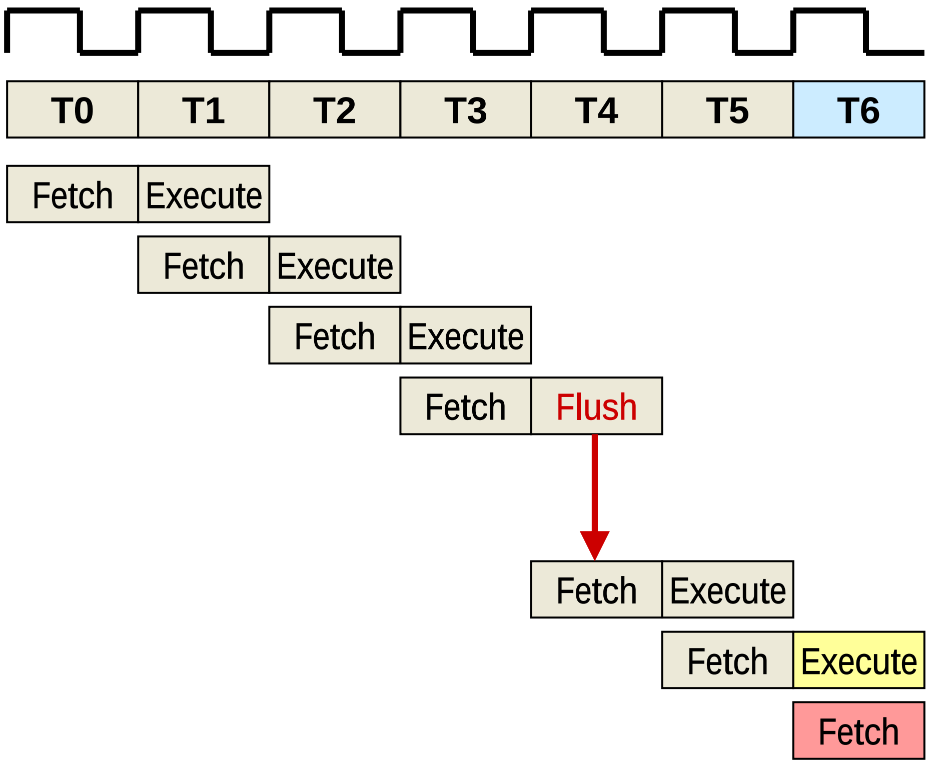
Pre-Fetched Instruction

```
movf PORTC,w
```

Executing Instruction

```
return
```

Instruction Cycles



T0	T1	T2	T3	T4	T5	T6
Fetch	Execute					
	Fetch	Execute				
		Fetch	Execute			
			Fetch	Flush		
				Fetch	Execute	
					Fetch	Execute
						Fetch

Example Program

```
 1  MAIN movlw  0x05
 2       movwf  REG1
 3       call   SUB1
 4       addwf  REG2

51  SUB1 movf   PORTB,w
52       return
53  SUB2 movf   PORTC,w
54       return
```

Instruction Pipelining

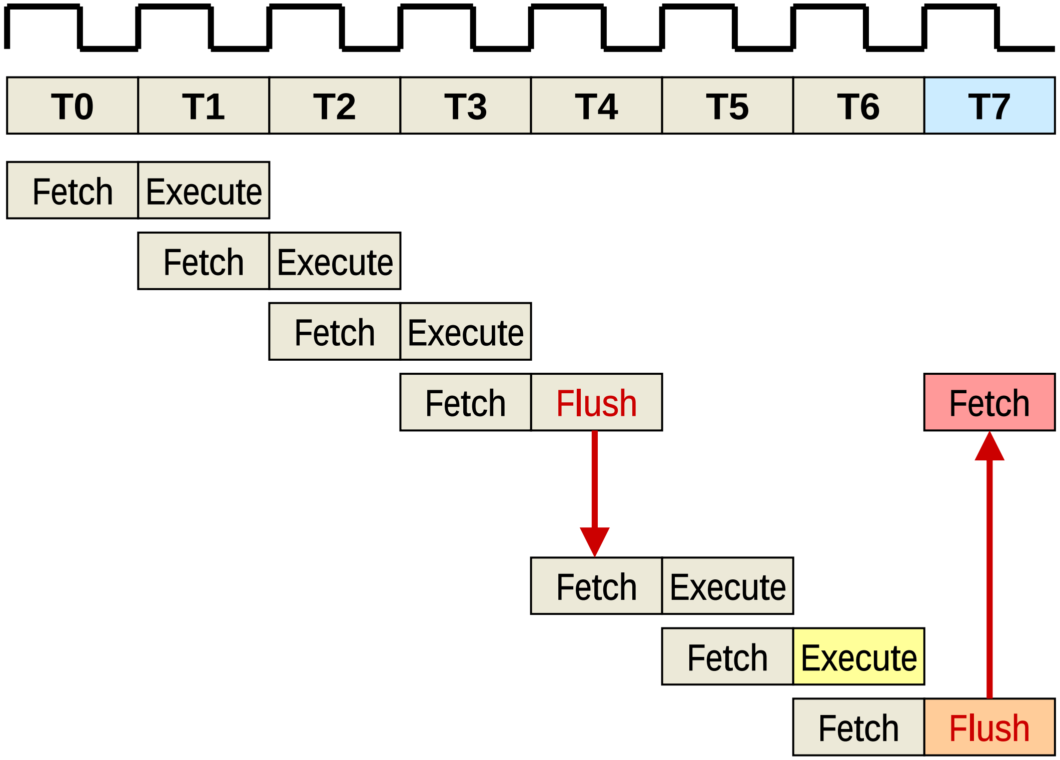
Pre-Fetched Instruction

`addwf REG2`

Executing Instruction

`return`

Instruction Cycles



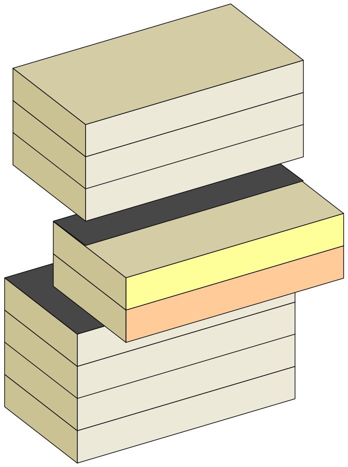
T0	T1	T2	T3	T4	T5	T6	T7
Fetch	Execute						
	Fetch	Execute					
		Fetch	Execute				
			Fetch	Flush			Fetch
				Fetch	Execute		
					Fetch	Execute	
						Fetch	Flush

Example Program

```
1   MAIN  movlw   0x05
2         movwf   REG1
3         call    SUB1
4         addwf   REG2

51  SUB1  movf    PORTB,w
52        return
53  SUB2  movf    PORTC,w
54        return
```

Long Word Instruction

8-bit Program Memory



8-bit Instruction on typical 8-bit MCU

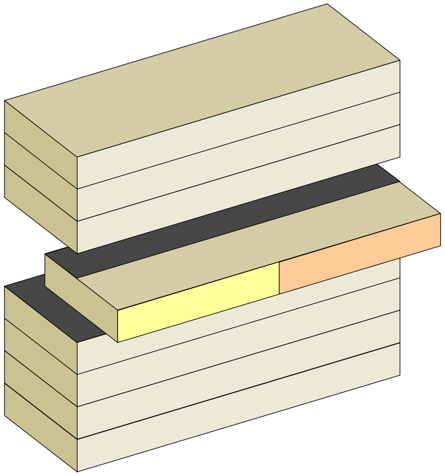
Example: Freescale 'Load Accumulator A':
• 2 Program Memory Locations
• 2 Instruction Cycles to Execute

ldaa #k

1	0	0	0	0	1	1	0
k	k	k	k	k	k	k	k

- **Limits Bandwidth**
- **Increases Memory Size Requirements**

14-bit Program Memory



14-bit Instruction on PIC16 8-bit MCU

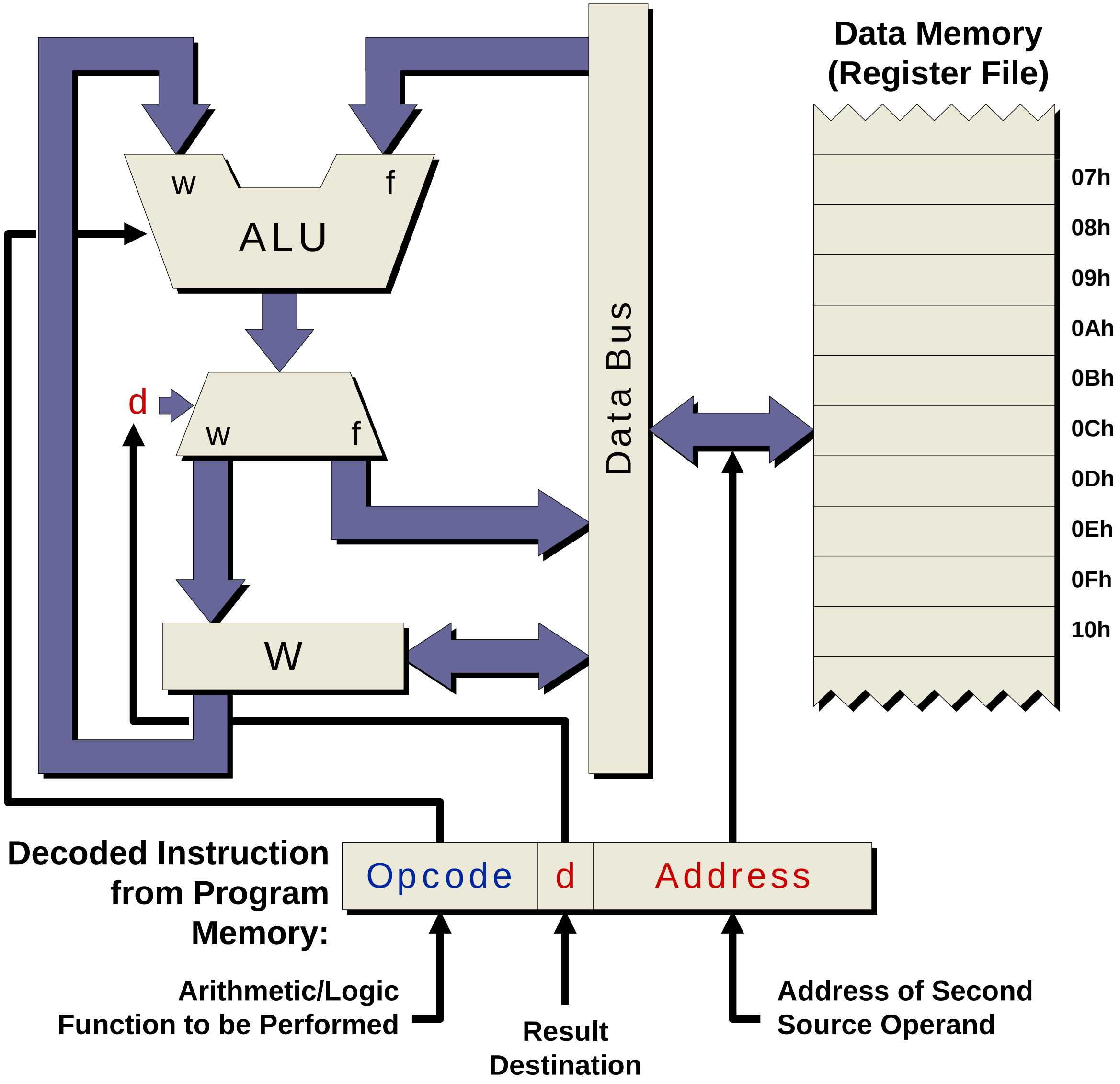
Example: 'Move Literal to Working Register'
• 1 Program Memory Location
• 1 Instruction Cycle to Execute

movlw k

1	1	0	0	0	0	k	k	k	k	k	k	k	k

- **Separate busses allow different widths**
- **2k x 14 is roughly equivalent to 4k x 8**

Register File Concept



Decoded Instruction from Program Memory:

Arithmetic/Logic Function to be Performed

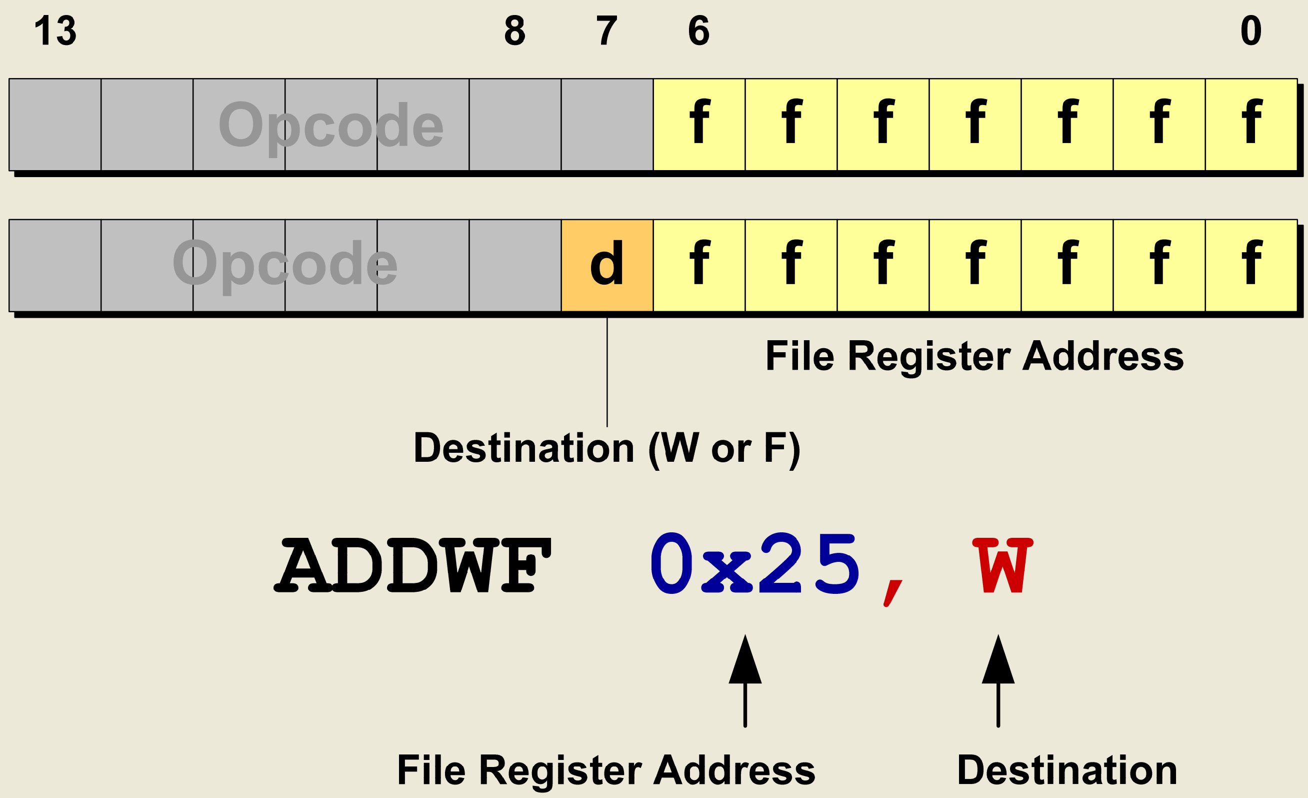
Result Destination

Address of Second Source Operand

- **<u>Register File Concept</u>: All of data memory is part of the register file, so any location in data memory may be operated on directly**
- **All peripherals are mapped into data memory as a series of registers**
- **<u>Orthogonal Instruction Set</u>: ALL instructions can operate on ANY data memory location**
- **The Long Word Instruction format allows a directly addressable register file**

Instruction Set Overview

Byte Oriented Operations



13					8	7	6						0
Opcode							f	f	f	f	f	f	f
Opcode						d	f	f	f	f	f	f	f

File Register Address

Destination (W or F)

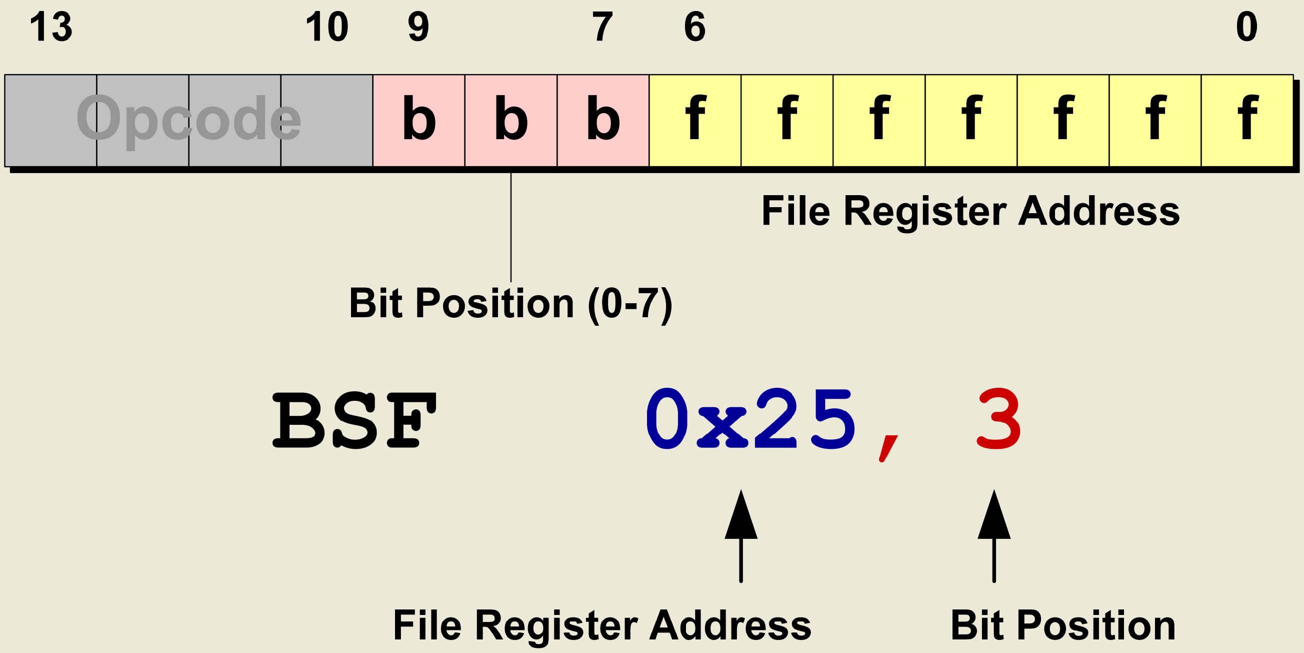
```
ADDWF  0x25, W
```

File Register Address

Destination

Instruction Set Overview

Bit Oriented Operations

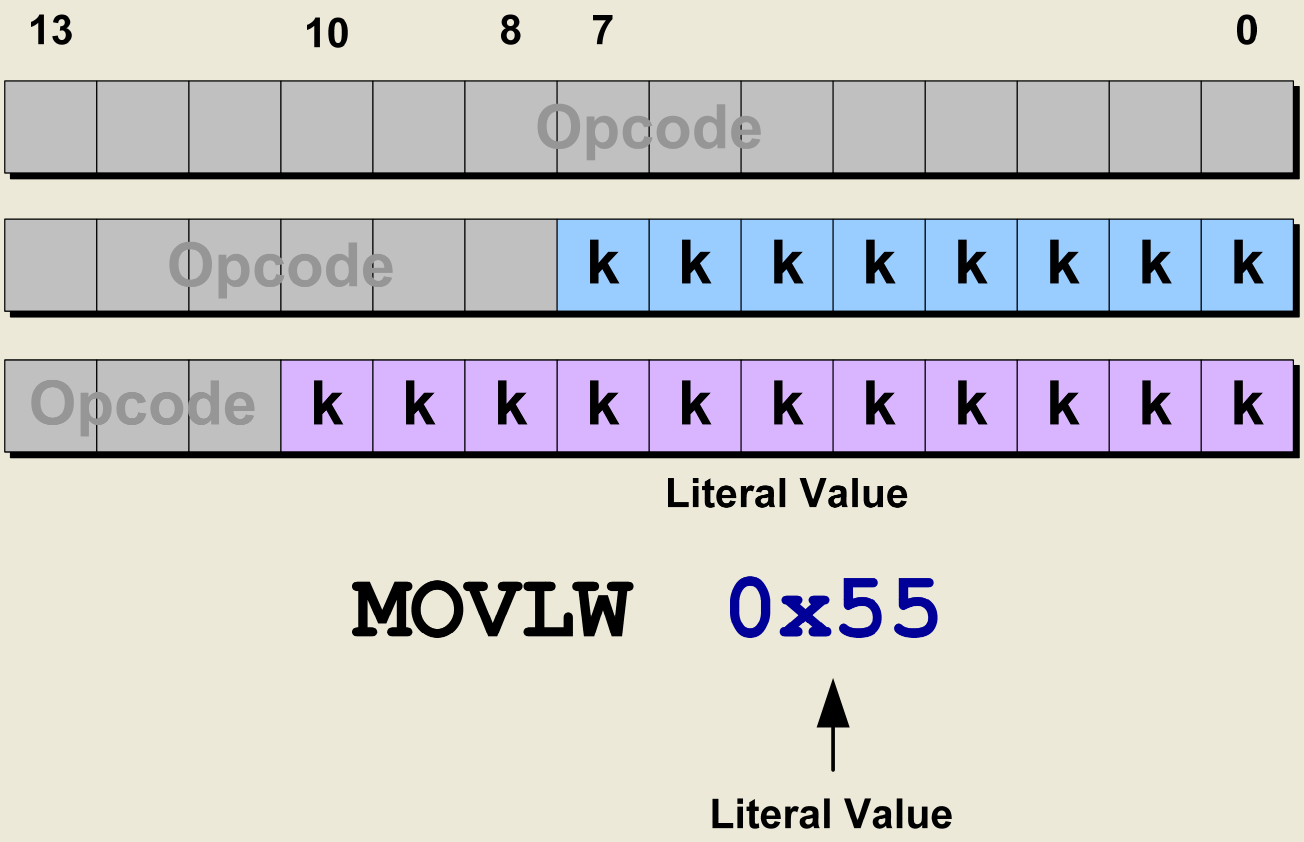


13			10	9		7	6						0
Opcode				b	b	b	f	f	f	f	f	f	f

```
BSF    0x25, 3
```

Instruction Set Overview

Literal and Control Operations



13 10 8 7 0

Opcode

Opcode k k k k k k k k

Opcode k k k k k k k k k k k

Literal Value

MOVLW 0x55

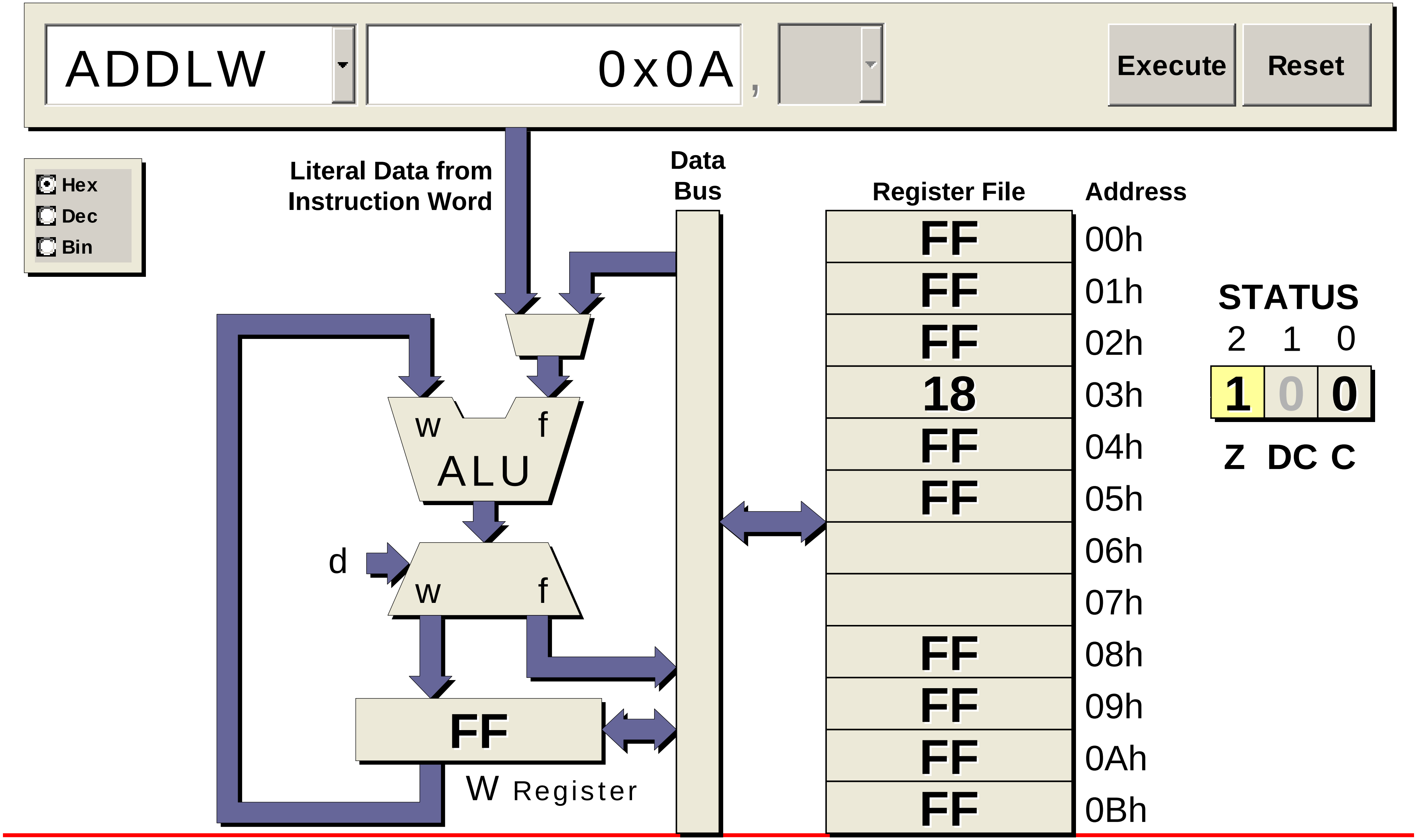
Literal Value

PIC16 Instruction Set

Byte Oriented Operations		
addwf	f,d	Add W and f
andwf	f,d	AND W with f
clrf	f	Clear f
clrw	-	Clear W
comf	f,d	Complement f
decf	f,d	Decrement f
decfsz	f,d	Decrement f, Skip if 0
incf	f,d	Increment f
incfsz	f,d	Increment f, Skip if 0
iorwf	f,d	Inclusive OR W with f
movf	f,d	Move f
movwf	f	Move W to f
nop	-	No Operation
rlf	f,d	Rotate Left f through Carry
rrf	f,d	Rotate Right f through Carry
subwf	f,d	Subtract W from f
swapf	f,d	Swap nibbles in f
xorwf	f,d	Exclusive OR W with f

Bit Oriented Operations		
bcf	f,b	Bit Clear f
bsf	f,b	Bit Set f
btfsc	f,b	Bit Test f, Skip if Clear
btfss	f,b	Bit Test f, Skip if Set
Literal and Control Operations		
addlw	k	Add literal and W
andlw	k	AND literal with W
call	k	Call subroutine
clrwdt	-	Clear Watchdog Timer
goto	k	Go to address
iorlw	k	Inclusive OR literal with W
movlw	k	Move literal to W
retfie	-	Return from interrupt
retlw	k	Return with literal in W
return	-	Return from Subroutine
sleep	-	Go into standby mode
sublw	k	Subtract W from literal
xorlw	k	Exclusive OR literal with W

PIC16 Visual Interpreter



ADDLW ▾ | 0x0A , | ▾ | Execute | Reset

● Hex
○ Dec
○ Bin

Register File	Address
FF	00h
FF	01h
FF	02h
18	03h
FF	04h
FF	05h
	06h
	07h
FF	08h
FF	09h
FF	0Ah
FF	0Bh

STATUS

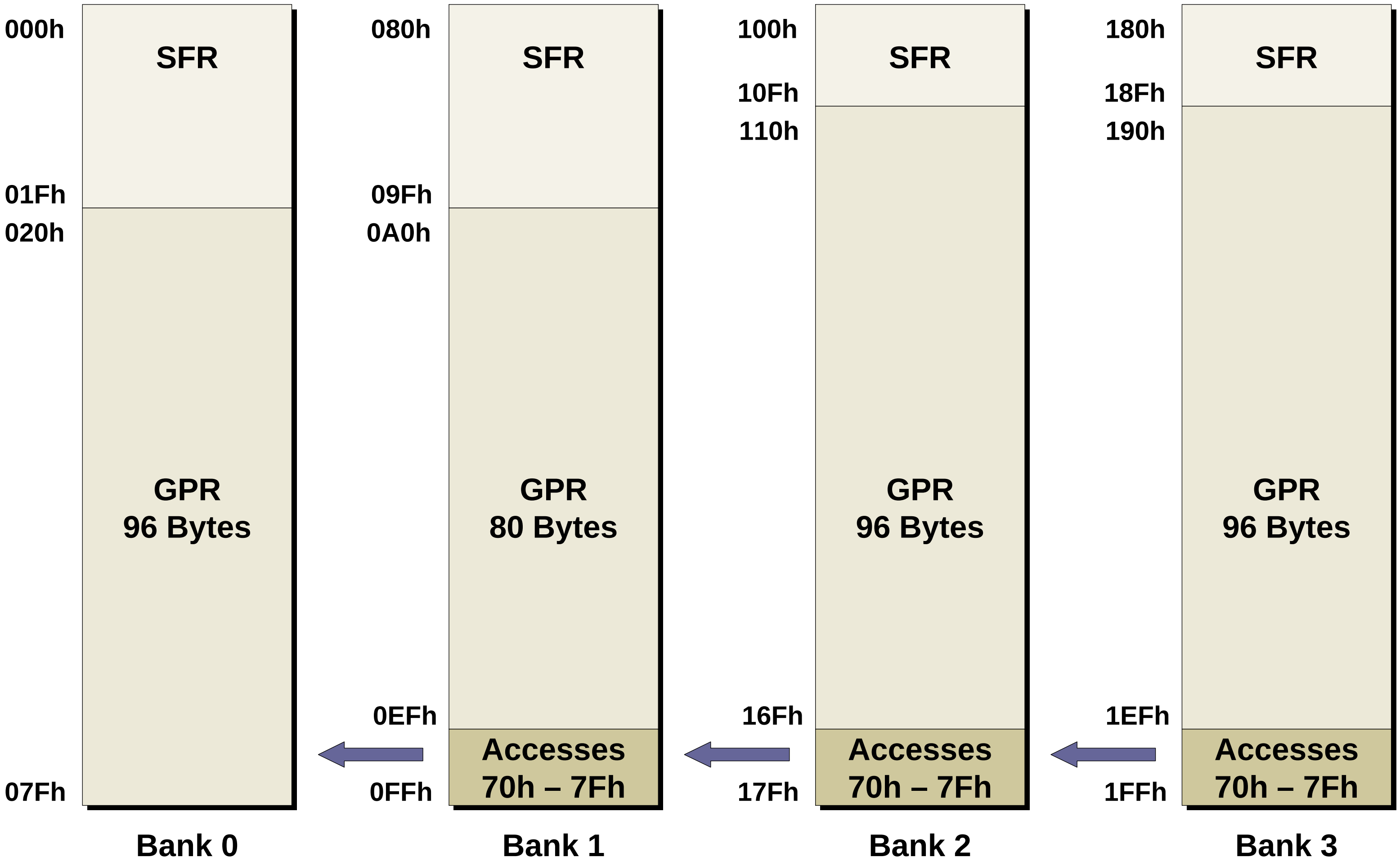
2	1	0
1	0	0
Z	DC	C

W Register: FF

Data Memory Organization

PIC16F876/877 Register File Map

368 Bytes of General Purpose RAM Plus Special Function Registers



Data Memory Organization

	Bank 0		Bank 1		Bank 2		Bank 3
000	INDF	080	INDF	100	INDF	180	INDF
001	TMR0	081	OPTION_REG	101	TMR0	181	OPTION_REG
002	PCL	082	PCL	102	PCL	182	PCL
003	STATUS	083	STATUS	103	STATUS	183	STATUS
004	FSR	084	FSR	104	FSR	184	FSR
005	PORTA	085	TRISA	105		185	
006	PORTB	086	TRISB	106	PORTB	186	TRISB
007	PORTC	087	TRISC	107		187	
008	PORTD	088	TRISD	108		188	
009	PORTE	089	TRISE	109		189	
00A	PCLATH	08A	PCLATH	10A	PCLATH	18A	PCLATH
00B	INTCON	08B	INTCON	10B	INTCON	18B	INTCON
00C	PIR1	08C	PIE1	10C	EEDATA	18C	EECON1
00D	PIR2	08D	PIE2	10D	EEADR	18D	EECON2

Device Specific Registers

STATUS Register

IRP	**RP1**	**RP0**	$\overline{\textbf{TO}}$	$\overline{\textbf{PD}}$	**Z**	**DC**	**C**
bit 7							bit 0

IRP: Register Bank Select (used for Indirect addressing)

0 = Bank 0, 1   1 = Bank 2, 3

RP1:RP0: Register Bank Select Bits (used for direct addressing)

00 = Bank 0, 01 = Bank 1, 10 = Bank 2, 11 = Bank 3

$\overline{\textbf{TO}}$**:** Time-out bit

0 = A WDT time-out occurred

$\overline{\textbf{PD}}$**:** Power-down bit

0 = SLEEP instruction executed

Z: Zero bit

1 = Result of arithmetic operation is zero

DC: Digit cary / borrow bit

1 = Carry out of 4th low order bit occurred / No borrow occurred

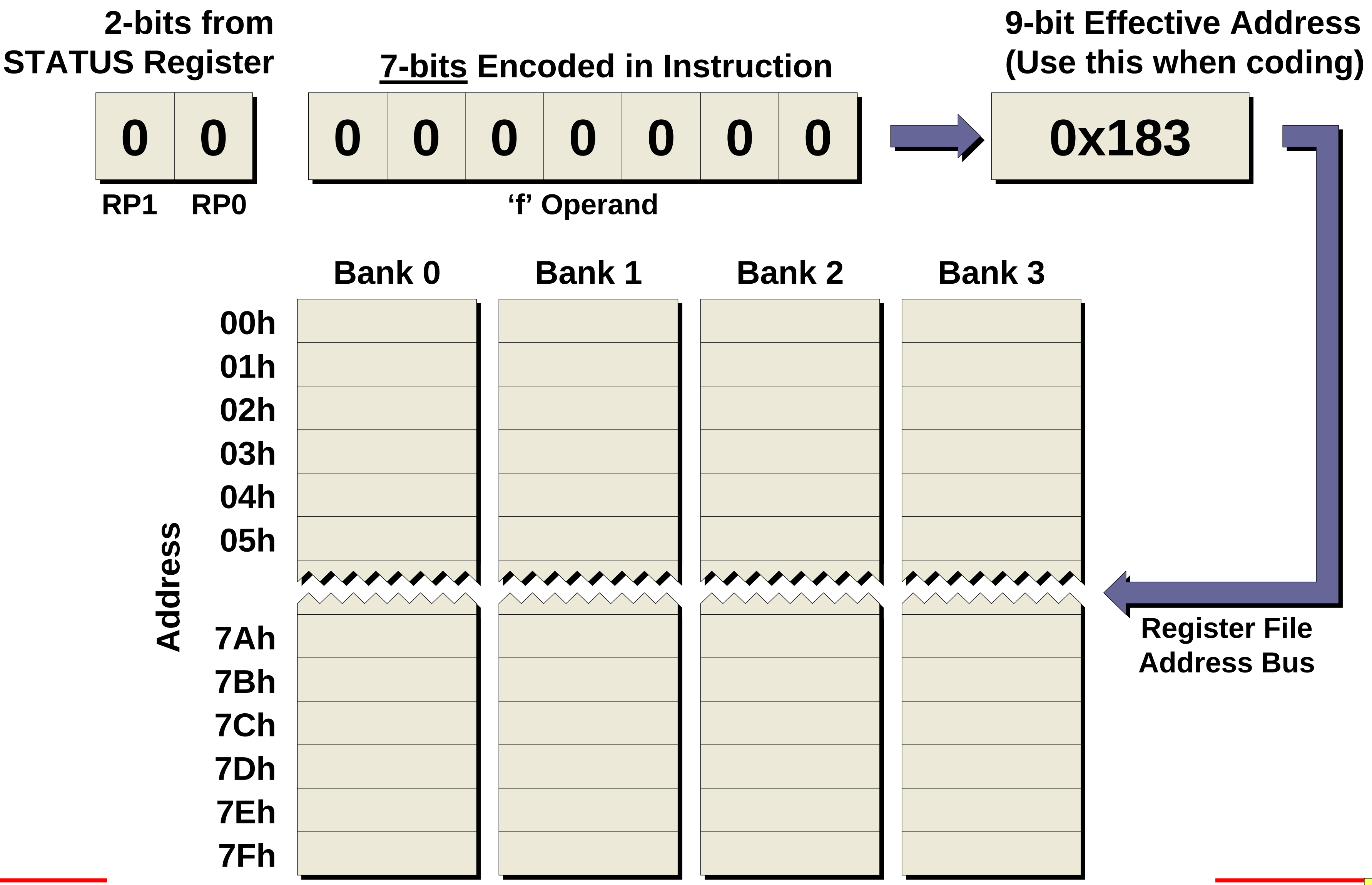
C: Carry / borrow bit

1 = Carry out of MSb occurred / No borrow occurred

PIC16 Addressing Modes

- **Data Memory Access:**
  - Direct `addwf <data_address>, <d>`
  - Indirect `addwf INDF, <d>`
  - Immediate (Literal) `movlw <constant>`
- **Program Memory Access:**
  - Absolute `goto <program_address>`
  - Relative `addwf PCL,f`

Register Direct Addressing



2-bits from STATUS Register

7-bits Encoded in Instruction

9-bit Effective Address (Use this when coding)

Register Direct Addressing

Example: Initialize bits 0-3 of PORTB as outputs

W Register:

F0

9-Bit Effective Address:

RP1	RP0	7-bits from Instruction						
0	0	0	0	0	0	0	0	0

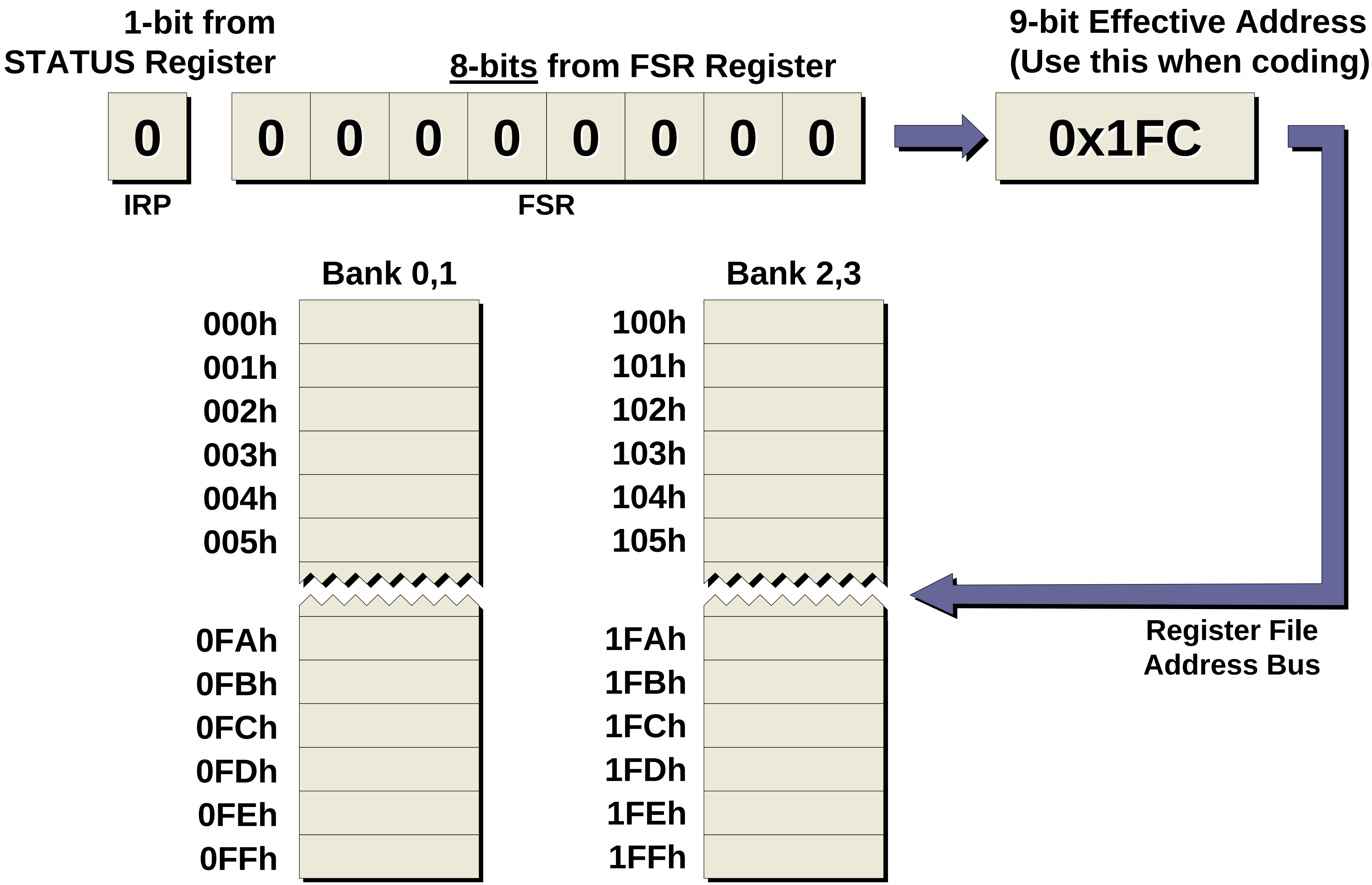
```
bsf    STATUS,RP0
movlw  b'11110000'
movwf  TRISB
bcf    STATUS,RP0
clrf   PORTB
```

Register File

Address	Bank 0	Bank 1	Address
INDF: 00h	FF	FF	80h : INDF
TMR0: 01h	FF	FF	81h : OPTION
PCL : 02h	FF	FF	82h : PCL
STATUS: 03h	38	38	83h : STATUS
FSR: 04h	FF	FF	84h : FSR
PORTA: 05h	FF	FF	85h : TRISA
PORTB: 06h	FF	FF	86h : TRISB
PORTC: 07h	FF	FF	87h : TRISC
20h	FF	FF	A0h
21h	FF	FF	A1h
22h	FF	FF	A2h
23h	FF	FF	A3h

○ Bin ○ Dec ● Hex

Register Indirect Addressing



1-bit from STATUS Register

8-bits from FSR Register

9-bit Effective Address (Use this when coding)

0

0 0 0 0 0 0 0 0

0x1FC

IRP

FSR

Bank 0,1

000h
001h
002h
003h
004h
005h
0FAh
0FBh
0FCh
0FDh
0FEh
0FFh

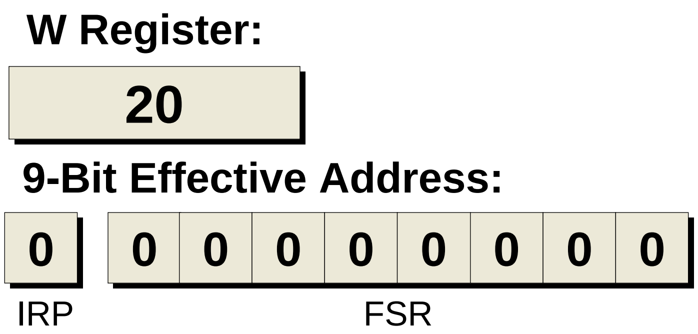
Bank 2,3

100h
101h
102h
103h
104h
105h
1FAh
1FBh
1FCh
1FDh
1FEh
1FFh

Register File Address Bus

Register Indirect Addressing

Example: Clear all RAM locations from 20h to 7Fh



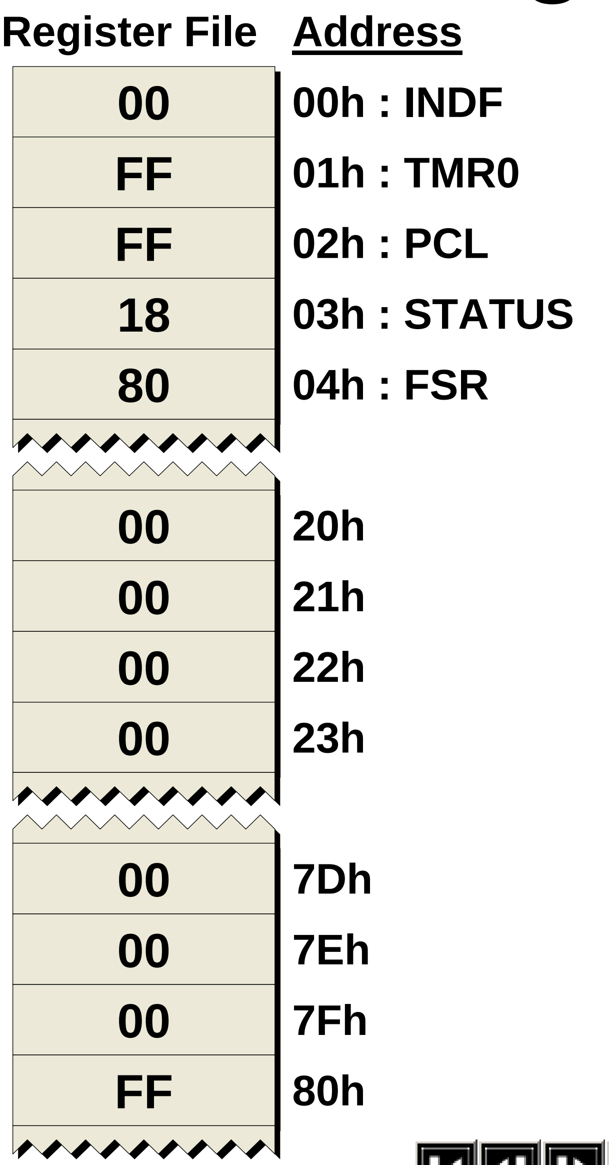
W Register:

20

9-Bit Effective Address:

IRP	FSR							
0	0	0	0	0	0	0	0	0

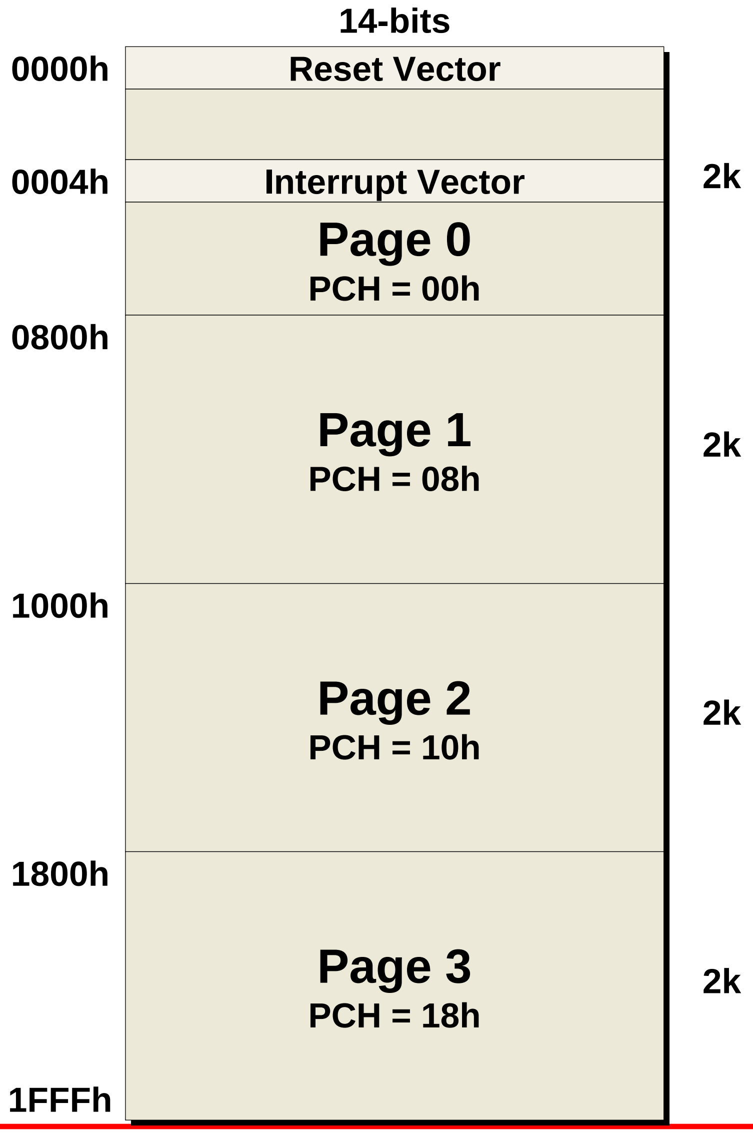
```
        bcf     STATUS,IRP
        movlw   0x20
        movwf   FSR
LOOP    clrf    INDF
        incf    FSR,f
        btfss   FSR,7
        goto    LOOP
        <next instruction>
```



Register File	Address
00	00h : INDF
FF	01h : TMR0
FF	02h : PCL
18	03h : STATUS
80	04h : FSR
00	20h
00	21h
00	22h
00	23h
00	7Dh
00	7Eh
00	7Fh
FF	80h

Program Memory Organization

- Program memory is divided into four 2k×14 pages
- Required to maintain single word/single cycle execution
- Paging is only a concern when using the call or goto instructions, or when directly modifying the program counter



14-bits

0000h Reset Vector

0004h Interrupt Vector

Page 0
PCH = 00h
2k

0800h

Page 1
PCH = 08h
2k

1000h

Page 2
PCH = 10h
2k

1800h

Page 3
PCH = 18h
2k

1FFFh

Program Counter

	PCH					PCL							
	12	11	10	9	8	7	6	5	4	3	2	1	0
Program Counter	0	0	0	0	0	0	0	0	0	0	0	0	0

- **13-bit PC can access up to 2^{13} = 8192 words**
- **Contains address of <u>NEXT</u> instruction (pipelining)**
- **Lower byte accessible in data memory as PCL**
- **Upper byte indirectly accessible via PCLATH**
- **Runs freely across page boundaries**
- **Events that modify PC out of sequence:**
  - Interrupts
  - Instructions: CALL, GOTO, RETURN, RETLW, RETFIE
  - Any instruction that uses the PCL register as an operand

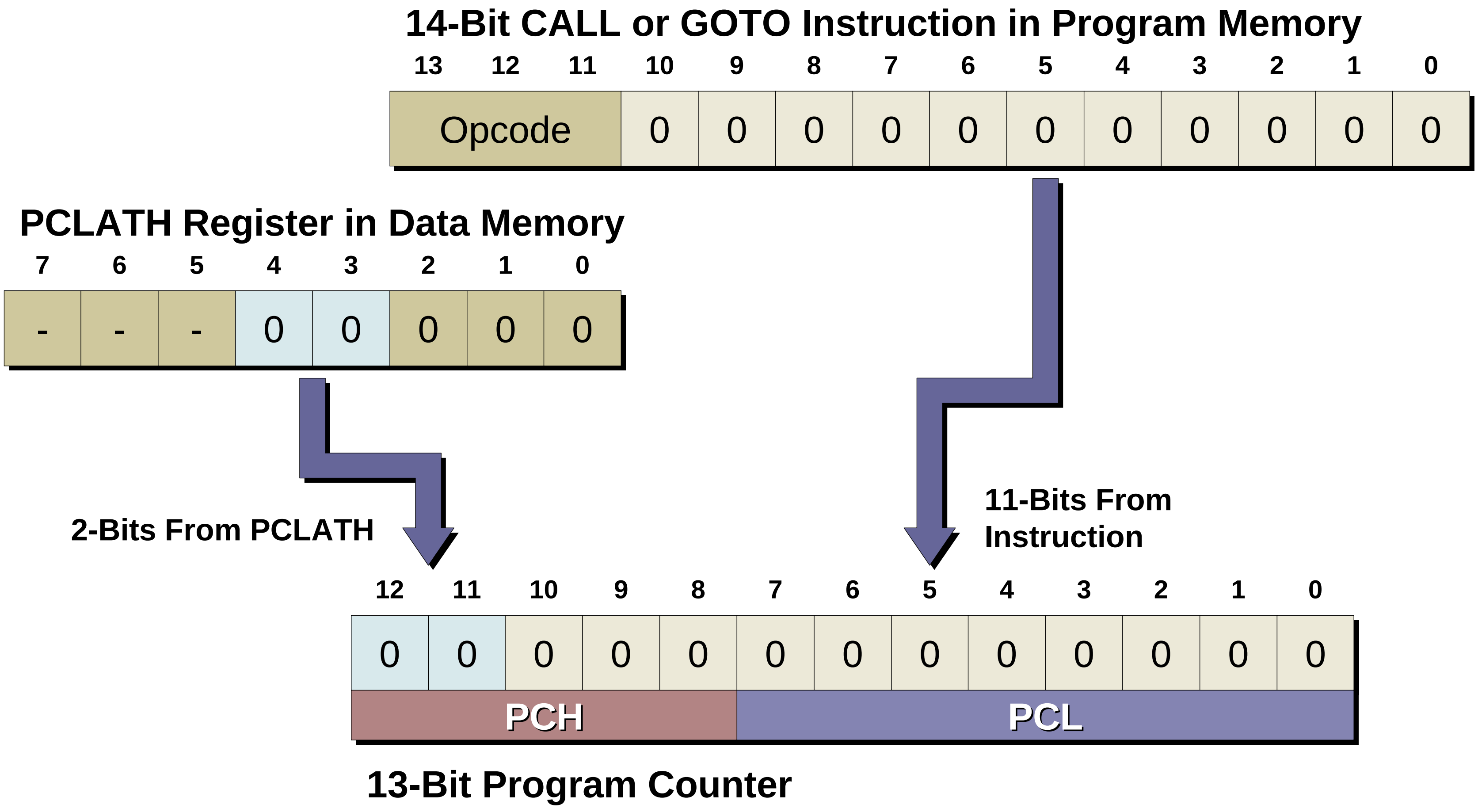
PC Absolute Addressing

CALL and GOTO Instructions:

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode			0	0	0	0	0	0	0	0	0	0	0

- **PC Absolute Addressing (Program Memory)**
  - Jump to another program memory location out of PC sequence
  - Call a subroutine
- **Used by the CALL and GOTO instructions**
  - 11-bits of the required 13 address bits are encoded in the instruction
  - 2 additional bits will come from the PCLATH register
- **Used when performing Computed Goto operation**
  - Address to jump to is calculated by the program
  - Computed address is written directly into the Program Counter

PC Absolute Addressing



14-Bit CALL or GOTO Instruction in Program Memory

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode			0	0	0	0	0	0	0	0	0	0	0

PCLATH Register in Data Memory

7	6	5	4	3	2	1	0
-	-	-	0	0	0	0	0

2-Bits From PCLATH

11-Bits From Instruction

12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0
PCH					PCL							

13-Bit Program Counter

PC Absolute Addressing

PCLATH Register

7	6	5	4	3	2	1	0
-	-	-	0	0	0	0	0

CALL Instruction in Program Memory

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode			0	0	0	0	0	0	0	0	0	0	0

W Register

FF

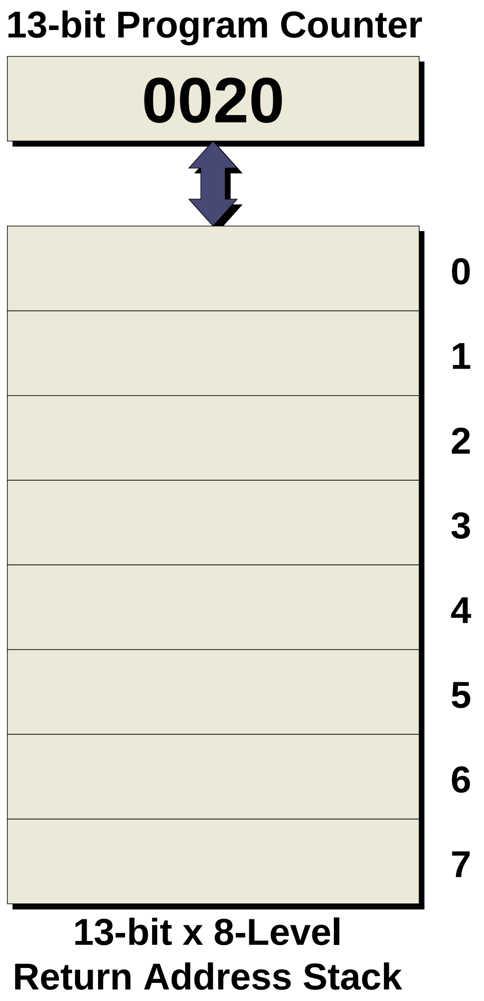
Program Counter - PCH:PCL

-	0	0	0	0	0	0	0	0	0	0	0	0	0

```
              org 0x0020
              movlw       HIGH MySubroutine
              movwf       PCLATH
              call        MySubroutine
              …
              org 0x1250
MySubroutine  <do something useful>
              …
              return
```

CALL / RETURN Stack

```
0020                 movlw  HIGH   MySub1
0021                 movwf  PCLATH
0022                 call   MySub1
0023                 call   MySub4
0024                 bsf    PORTB,7
 …                   …
1000   MySub1        bsf    PORTB,0
1001                 call   MySub2
1002                 return
1003   MySub2        bsf    PORTB,1
1004                 call   MySub3
1005                 return
1006   MySub3        bsf    PORTB,2
1007                 return
1008   MySub4        bsf    PORTB,3
1009                 call   MySub2
100A                 return
```

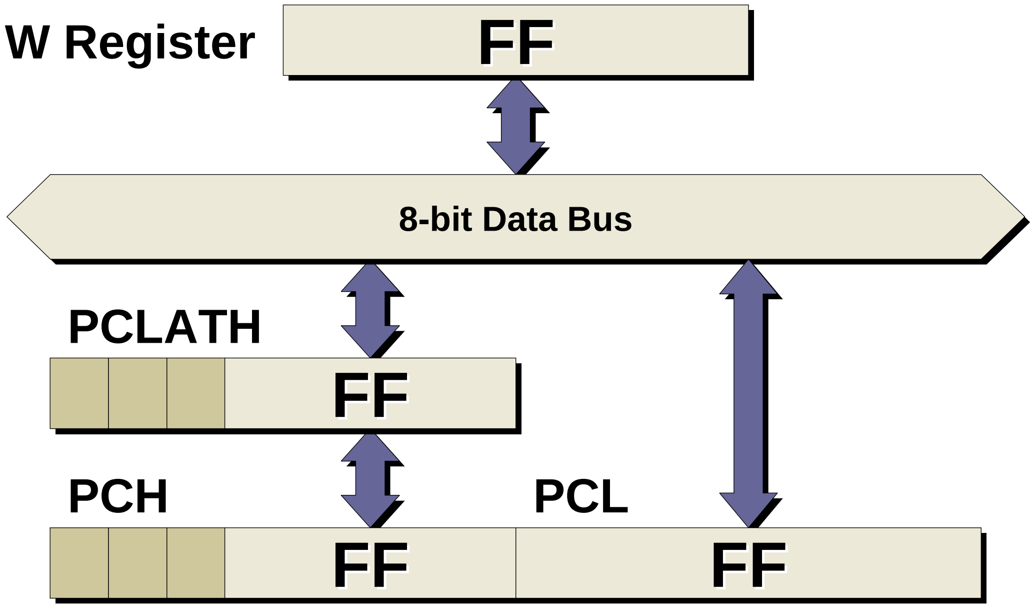


PC Relative Addressing

To write to PC:

❶ Write high byte to PCLATH

❷ Write low byte to PCL (PCH will be loaded with value from PCLATH)



W Register FF

8-bit Data Bus

PCLATH FF

PCH FF

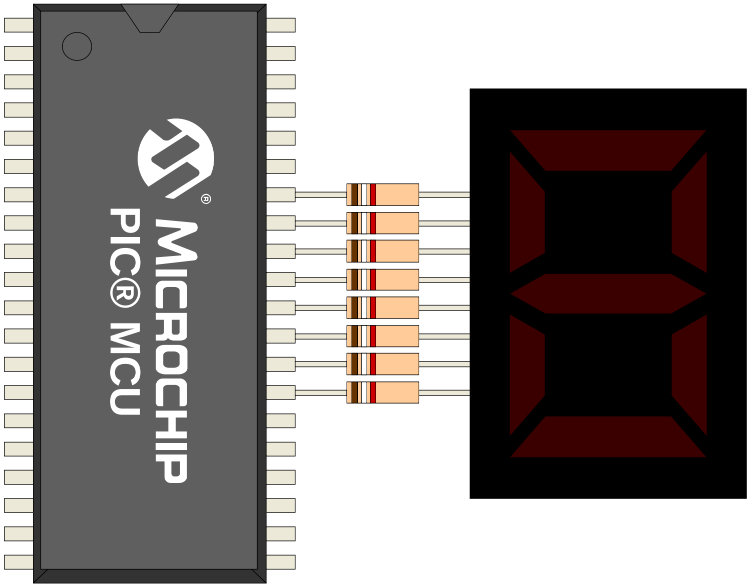
PCL FF

```
movlw     HIGH 0x1250
movwf     PCLATH
movlw     LOW 0x1250
movwf     PCL
```

PC Relative Addressing: Lookup Table

Example: Use a lookup table with relative addressing to retrieve the bit pattern to display a digit on a 7-segment LED

```
      ORG    0x0020              ;Page 0
      movlw  HIGH SevenSegDecode
      movwf  PCLATH
      movlw  .5
      call   SevenSegDecode
      movwf  PORTB
      …
      ORG    0x1800              ;Page 3
SevenSegDecode:
      addwf  PCL,f
      retlw  b’00111111’   ;0
      retlw  b’00000110’   ;1
      retlw  b’01011011’   ;2
      retlw  b’01001111’   ;3
      retlw  b’01100110’   ;4
      retlw  b’01101101’   ;5
      retlw  b’01111101’   ;6
      retlw  b’00000111’   ;7
      retlw  b’01111111’   ;8
      retlw  b’01101111’   ;9
```



Special Features Overview

Configuration Word

CP	**-**	**DEBUG**	**WRT1**	**WRT0**	**CPD**	**LVP**	**BOREN**	**-**	**-**	**PWRTEN**	**WDTEN**	**FOSC1**	**FOSC0**
bit 1													bit 0

- **Located in program memory space, outside the reach of the program counter**
- **Used to setup device options:**
  - Code Protection
  - Oscillator Mode
  - Watchdog Timer
  - Power Up Timer
  - Brown Out Reset
  - Low Voltage Programming
  - Flash Program Memory Write
- **Only readable at program time on most PIC16 devices**

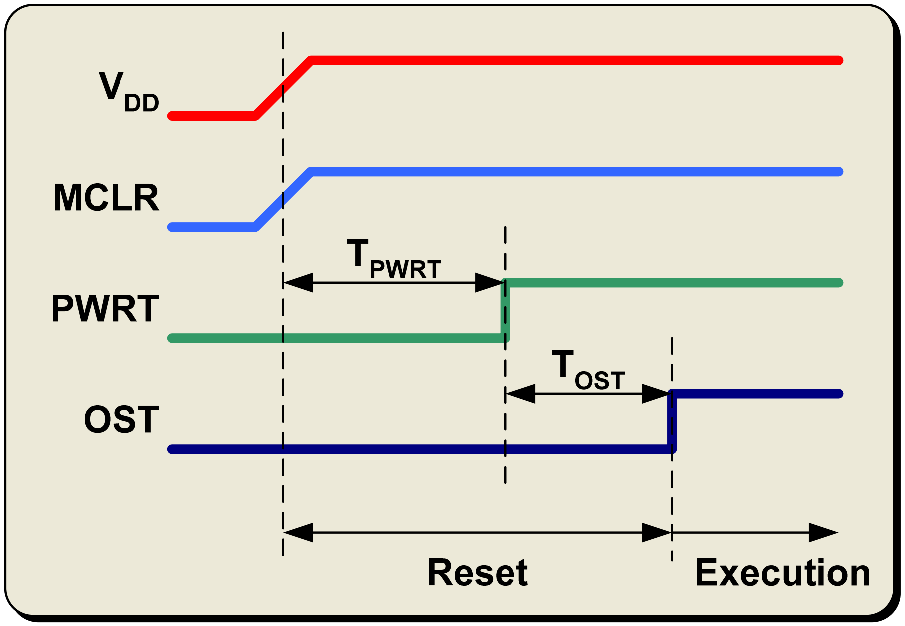
PIC16 Oscillator Options

XT	**Standard frequency crystal oscillator**	**100kHz - 4MHz**
HS	**High frequency crystal oscillator**	**4MHz - 20MHz**
LP	**Low frequency crystal oscillator**	**5kHz - 200kHz**
RC	**External RC oscillator**	**DC - 4MHz**
INTRC	**Internal RC oscillator**	**4 or 8 MHz ± 2%**

- **Selectable clock options provide greater flexibility for the designer:**
  - LP Oscillator designed to draw least amount of current
  - RC or INTRC provide ultra low cost oscillator solution
  - XT optimized for most commonly used oscillator frequencies
  - HS optimized to drive high frequency crystals or resonators
- **Speed ranges are guidelines only**

POR, OST, PWRT

- **POR: Power On Reset**
  - With MCLR tied to V_{DD}, a reset pulse is generated when V_{DD} rise is detected
- **PWRT: Power Up Timer**
  - Device is held in reset for 72ms (nominal) to allow V_{DD} to rise to an acceptable level (after POR only)
- **OST: Oscillator Start-up Timer**
  - Holds device in reset for 1024 <u>cycles</u> to allow crystal or resonator to stabilize in frequency and amplitude; not active in RC modes; used only after POR or Wake Up from SLEEP



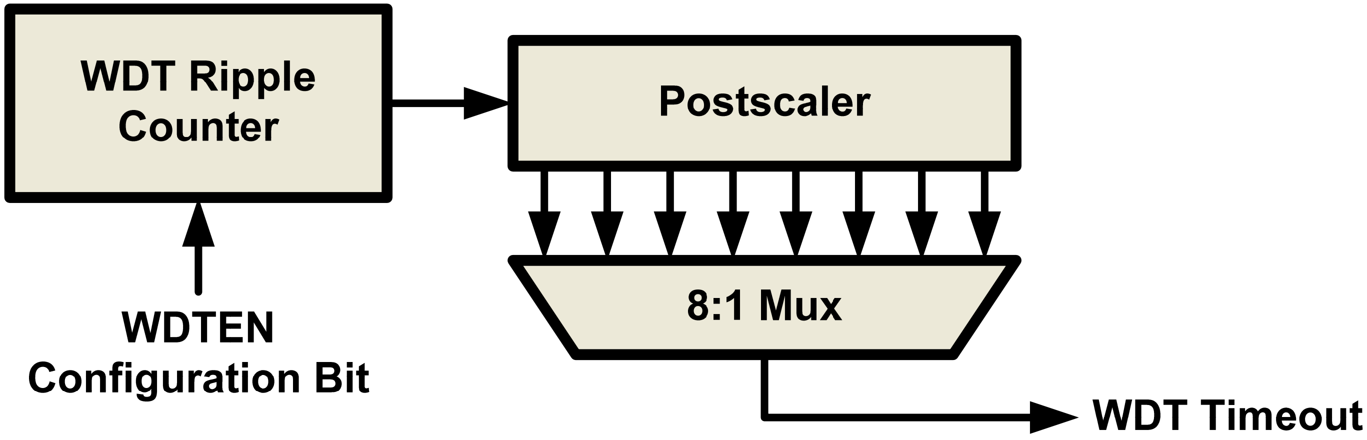
Sleep Mode

- **The processor can be put into a power-down mode by executing the SLEEP instruction**
  - System oscillator is stopped
  - Processor status is maintained (static design)
  - Watchdog timer continues to run, if enabled
  - Minimal supply current is drawn - mostly due to leakage (0.1 - 2.0µA typical)

Events that wake processor from sleep	
MCLR	**Master Clear Pin Asserted (pulled low)**
WDT	**Watchdog Timer Timeout**
INT	**INT Pin Interrupt**
TMR1	**Timer 1 Interrupt (or also TMR3 on PIC18)**
ADC	**A/D Conversion Complete Interrupt**
CMP	**Comparator Output Change Interrupt**
CCP	**Input Capture Event**
PORTB	**PORTB Interrupt on Change**
SSP	**Synchronous Serial Port (I^2C™ Mode) Start / Stop Bit Detect Interrupt**
PSP	**Parallel Slave Port Read or Write**

Watchdog Timer

- Helps recover from software malfunction
- Uses its own free-running on-chip RC oscillator
- WDT is cleared by CLRWDT instruction
- Enabled WDT cannot be disabled by software
- WDT overflow resets the chip
- Programmable timeout period: 18ms to 3.0s typical
- Operates in SLEEP; on time out, wakes up CPU



BOR –Brown Out Reset

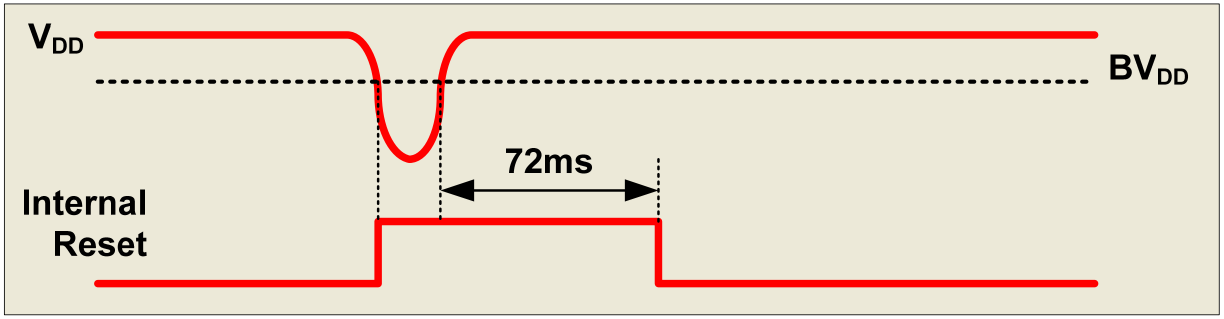
- When voltage drops below a particular threshold, the device is held in reset
- Prevents erratic or unexpected operation
- Eliminates need for external BOR circuitry

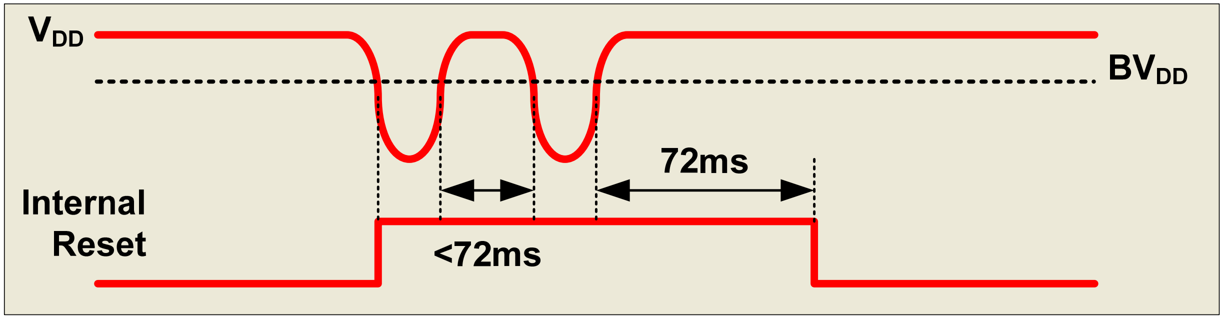
PBOR – Programmable Brown Out Reset

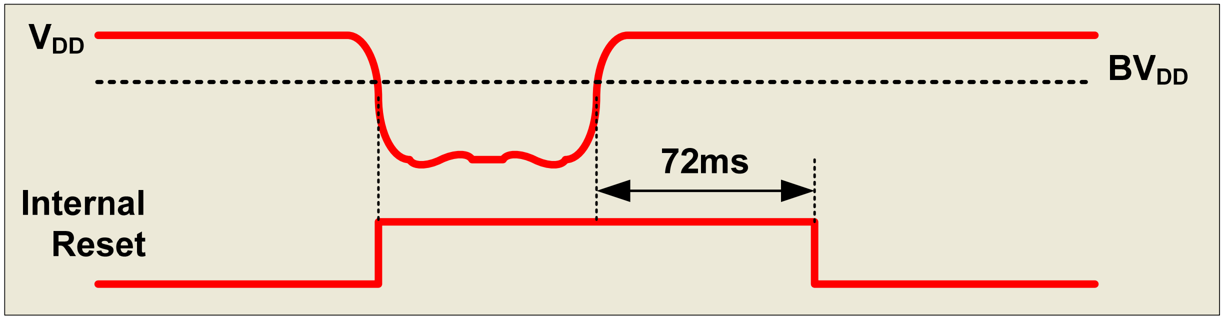
- **Configuration Option (set at program time)**
  - Cannot be enabled / disabled in software
- **Four selectable BV_{DD} trip points:**
  - 2.5V – Minimum V_{DD} for OTP PIC® MCUs
  - 2.7V
  - 4.2V
  - 4.5V
- **For other thresholds, use an external supervisor (MCP1xx, MCP8xx/TCM8xx, or TC12xx)**

(P)BOR – Brown Out Reset

- Holds PIC® MCU in reset until ~72ms after V_{DD} rises back above threshold

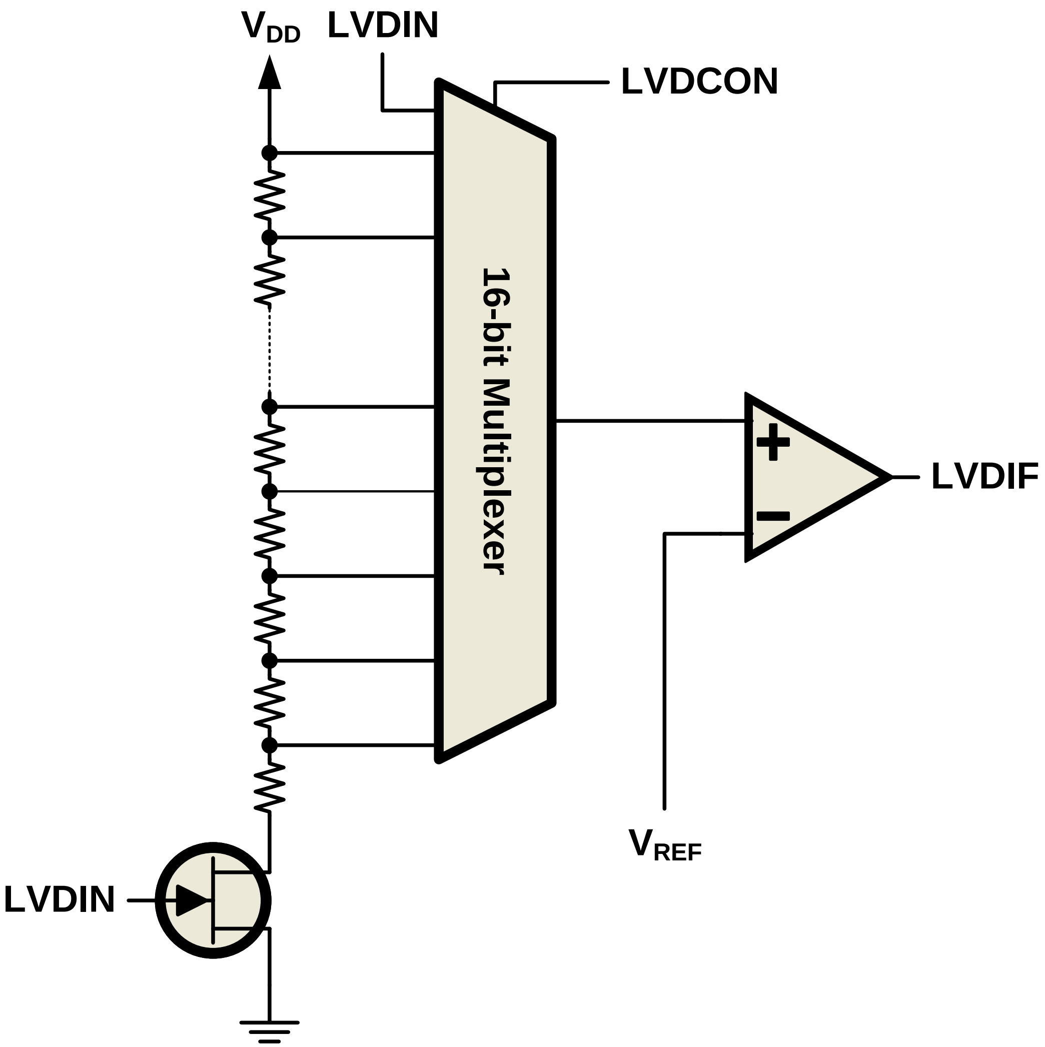






PLVD – Programmable Low Voltage Detect

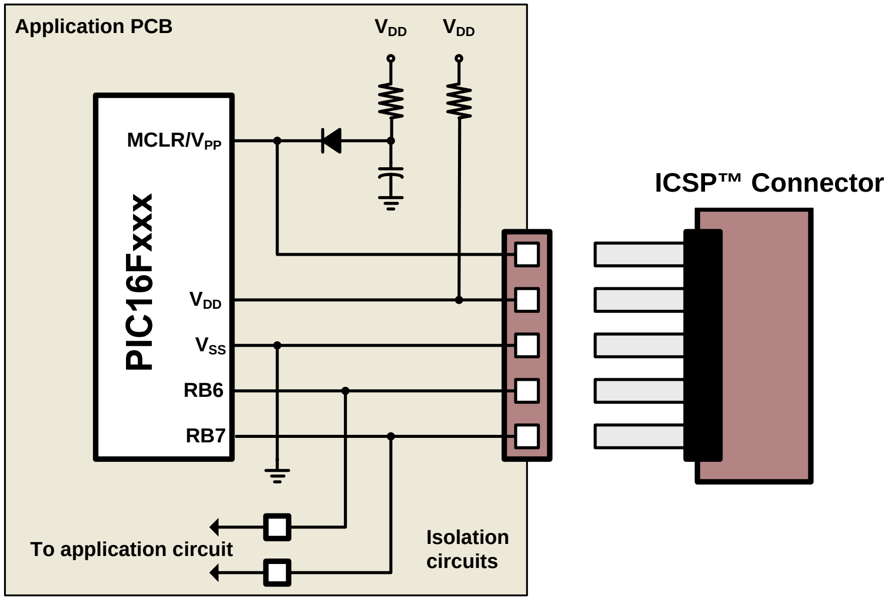
- **Early warning before brown out**
- **16 selectable trip points:**
  - 1.8V up to 4.5V in 0.1 to 0.2V steps
  - External analog input
- **Internal V_{REF}**



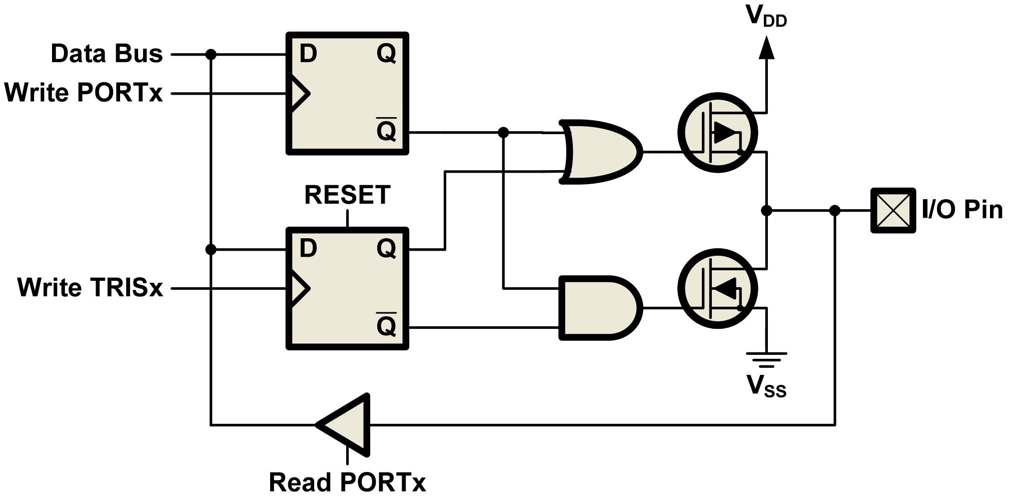
In-Circuit Serial Programming™

- **Only two pins required for programming**
- **Convenient for In-System Programming of**
  - Calibration Data
  - Serialization Data
- **Supported by MPLAB® PM3 & ICD2**

Pin	Function
V_{PP}	Programming Voltage = 13V
V_{DD}	Supply Voltage
V_{SS}	Ground
RB6	Clock Input
RB7	Data I/O & Command Input

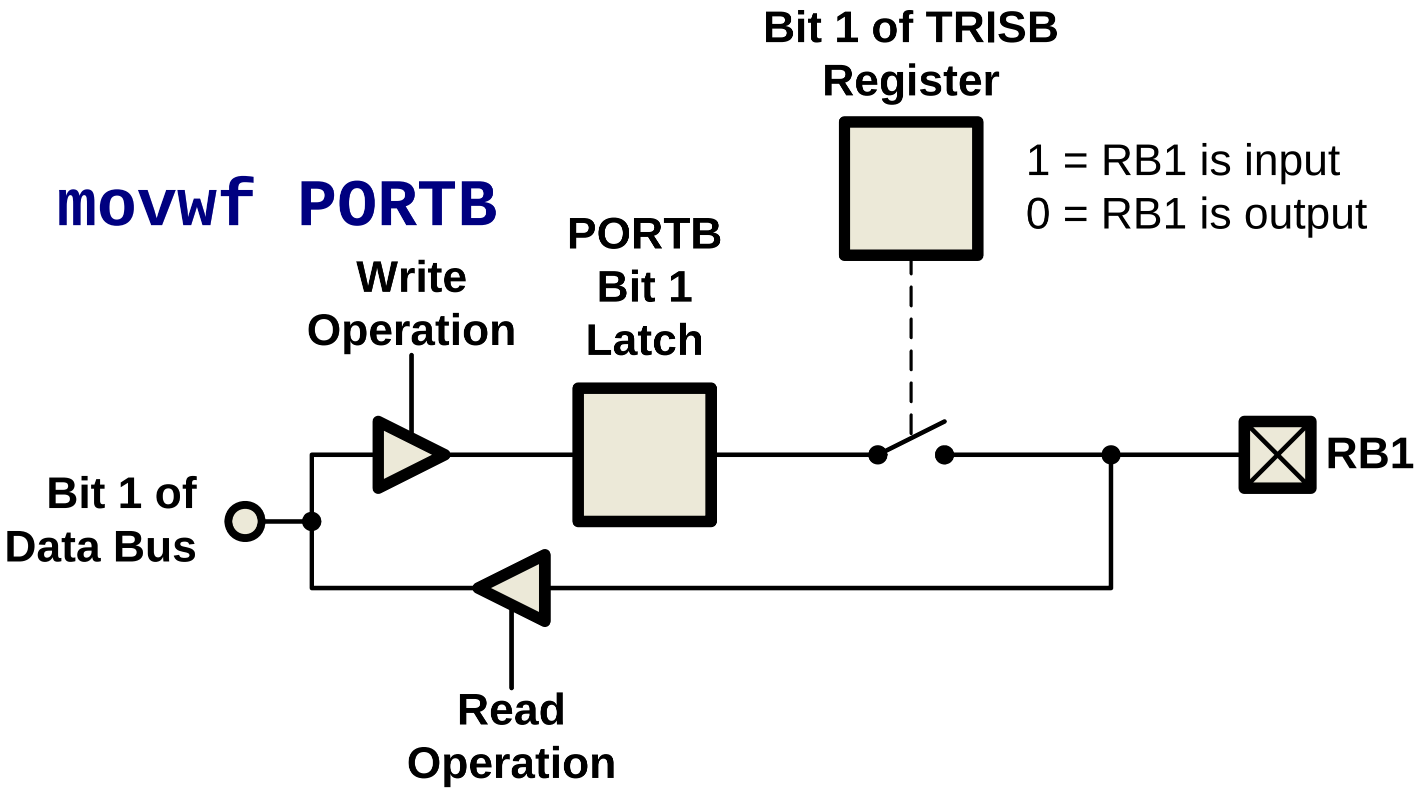


I/O Ports



- High Drive Capability
- Can directly drive LEDs
- Direct, single cycle bit manipulation
- Each pin has individual direction control under software
- All pins have ESD protection diodes
- Pin RA4 is usually open drain
- All I/O pins default to inputs (high impedance) on startup
- All pins multiplexed with analog functions default to analog inputs on startup

I/O Pin Conceptual Diagram



Bit 1 of TRISB Register

1 = RB1 is input
0 = RB1 is output

movwf PORTB

Write Operation

PORTB Bit 1 Latch

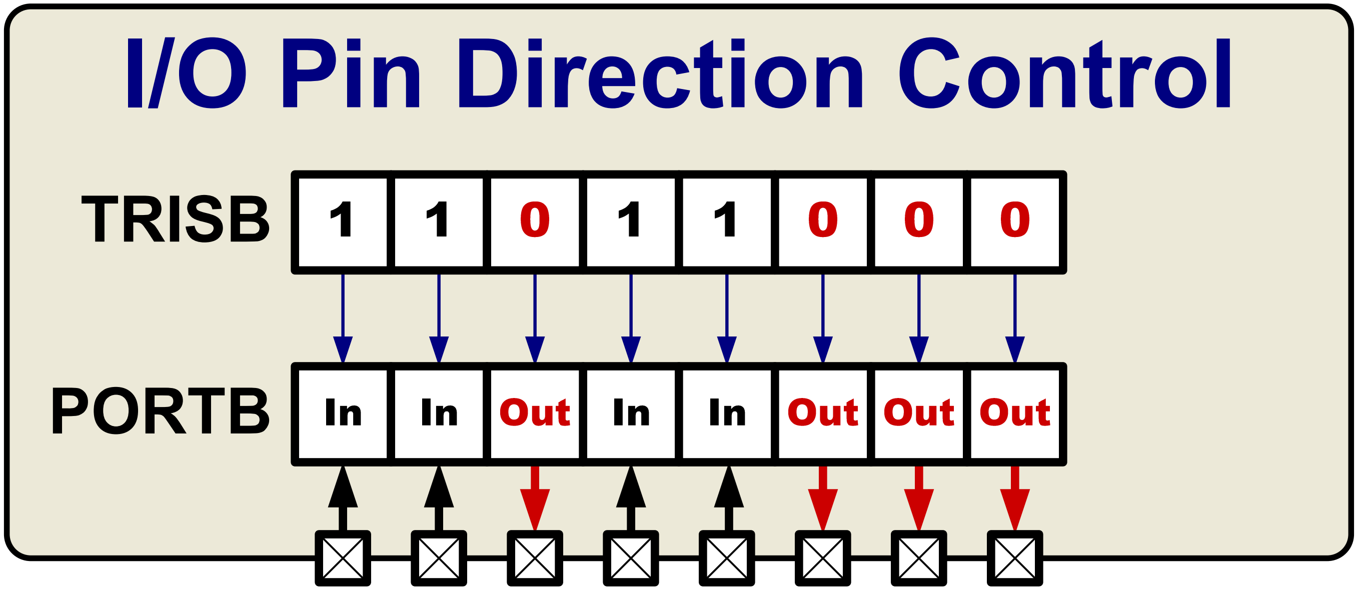
RB1

Bit 1 of Data Bus

Read Operation

movf PORTB,w

I/O Ports



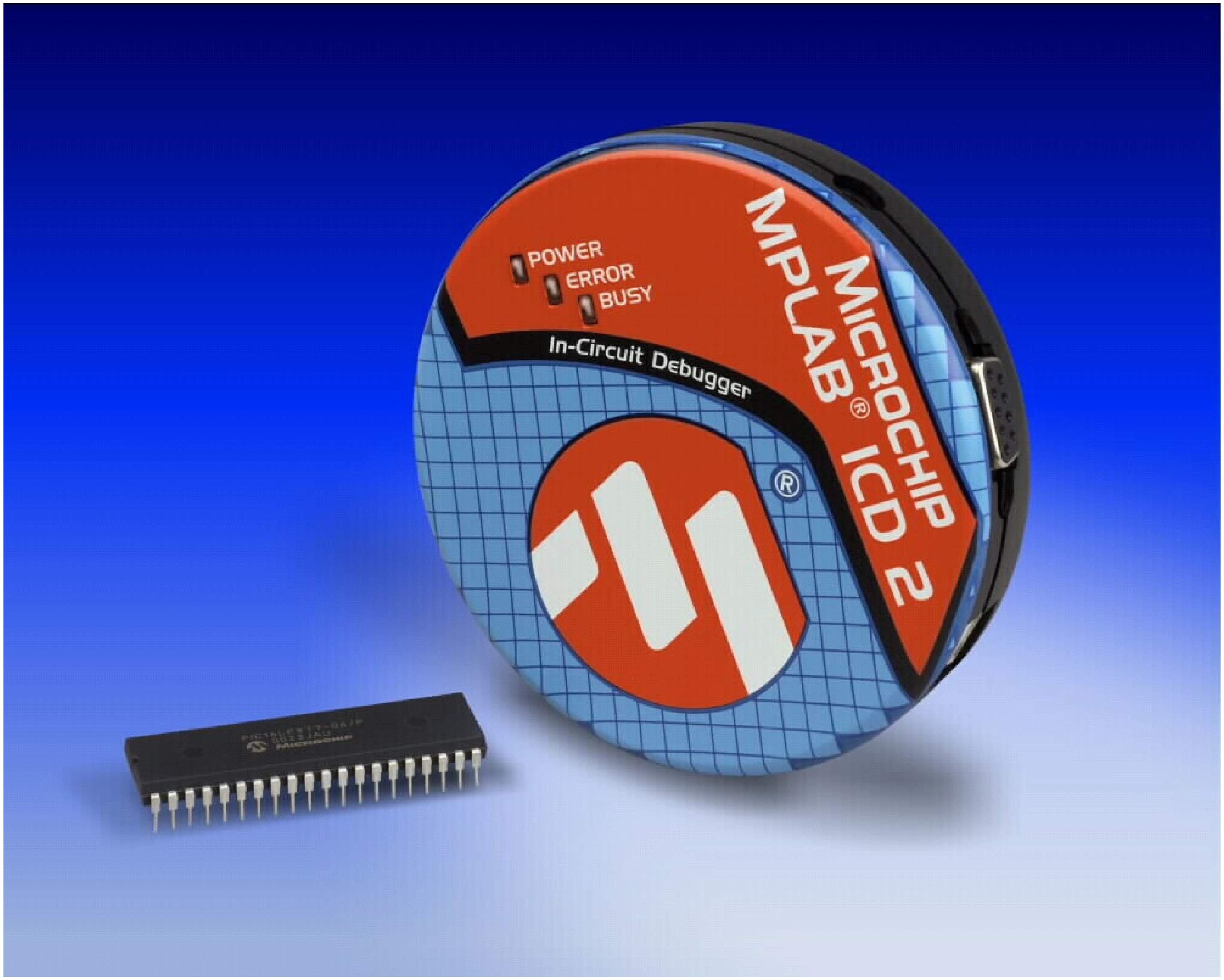
I/O Pin Direction Control

TRISB	1	1	0	1	1	0	0	0
PORTB	In	In	Out	In	In	Out	Out	Out

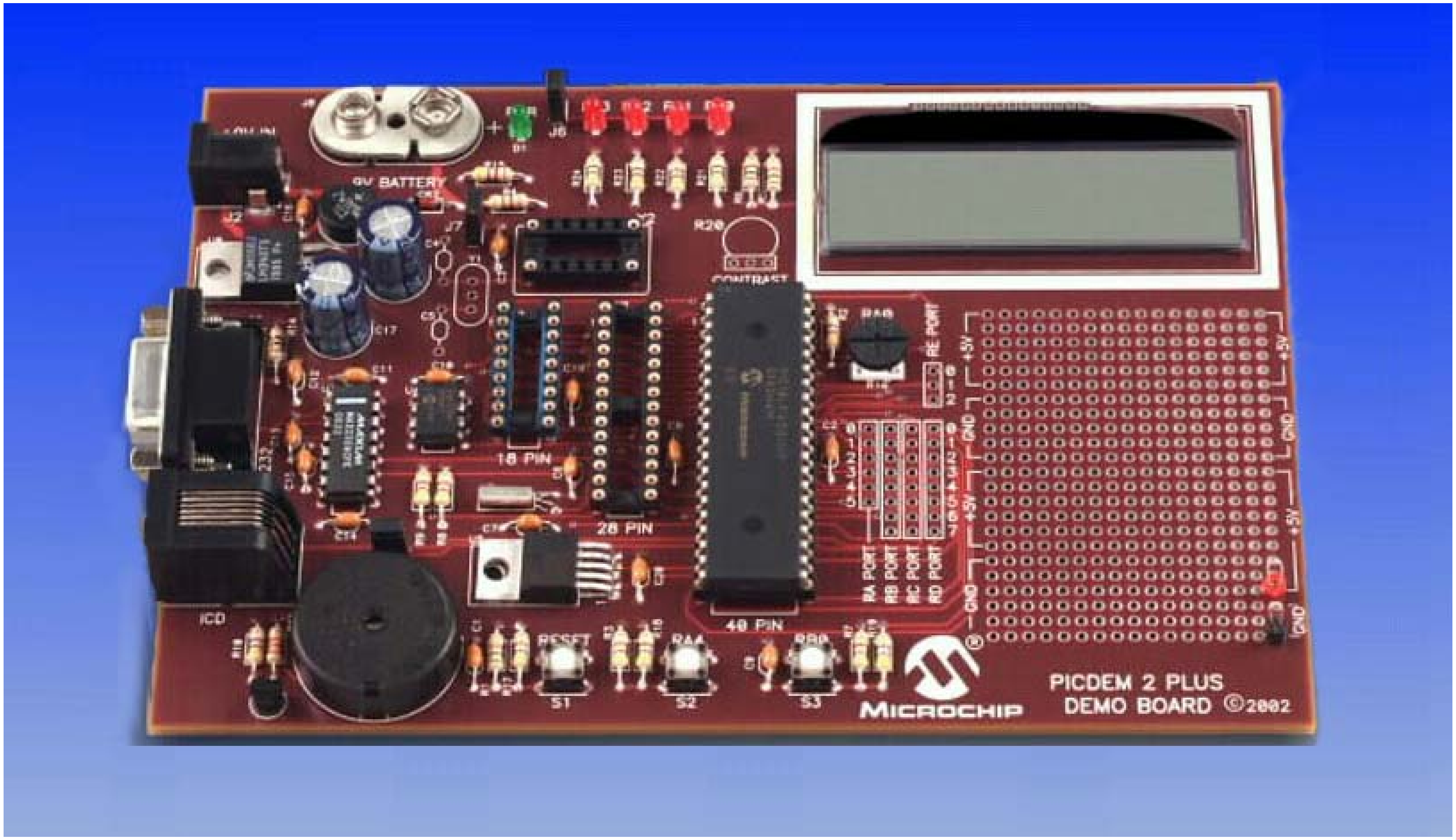
- Bit n in TRISx controls the data direction of Bit n in PORTx
- 1 = Input, 0 = Output

Hands-on Exercises

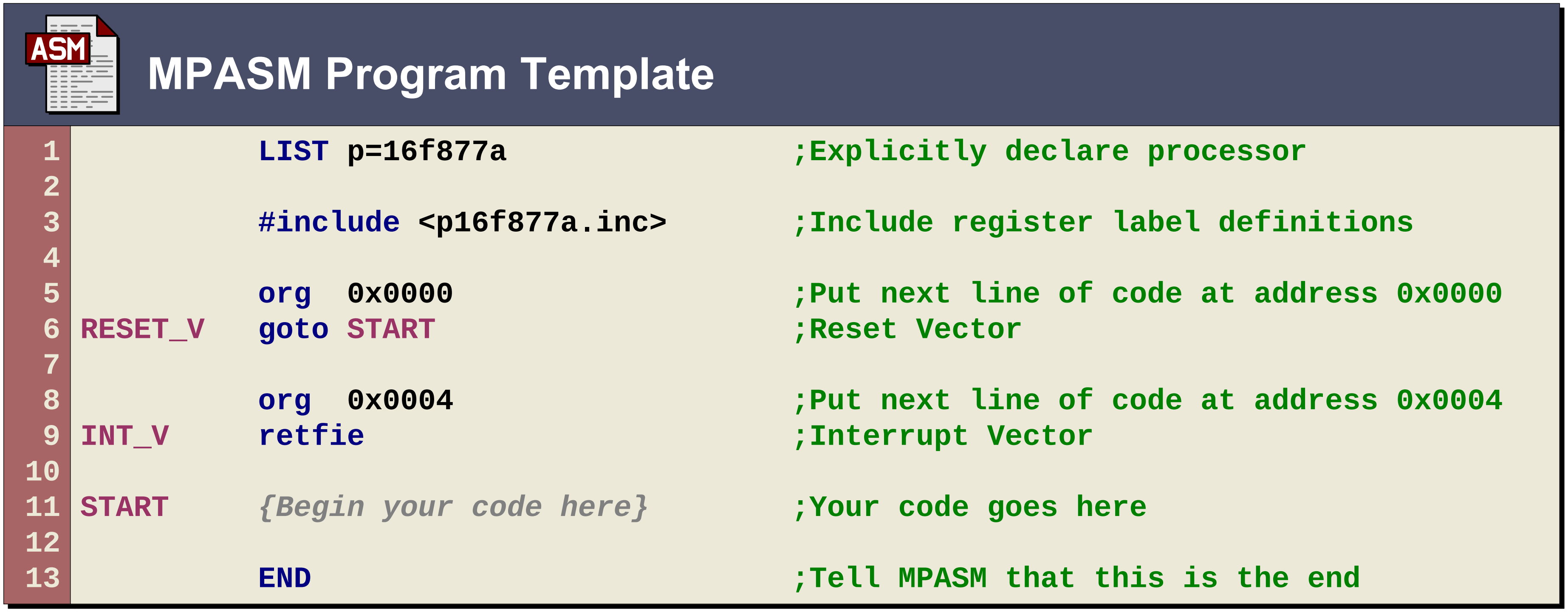
MPLAB® ICD2



PICDEM™ 2 Plus Board



MPASM™ Assembler Template



MPASM Program Template

```
 1             LIST p=16f877a                ;Explicitly declare processor
 2
 3             #include <p16f877a.inc>       ;Include register label definitions
 4
 5             org  0x0000                   ;Put next line of code at address 0x0000
 6 RESET_V     goto START                    ;Reset Vector
 7
 8             org  0x0004                   ;Put next line of code at address 0x0004
 9 INT_V       retfie                        ;Interrupt Vector
10
11 START       {Begin your code here}        ;Your code goes here
12
13             END                           ;Tell MPASM that this is the end
```

- **If not using interrupts, lines 8 and 9 could be omitted**
- **The labels in the left column may be anything you want; these are just examples**

Specifying the Radix

- By default, MPASM™ assembler expects numbers in hexadecimal
- Default can be changed through IDE or by adding r=hex or r=dec as a parameter to the LIST directive:

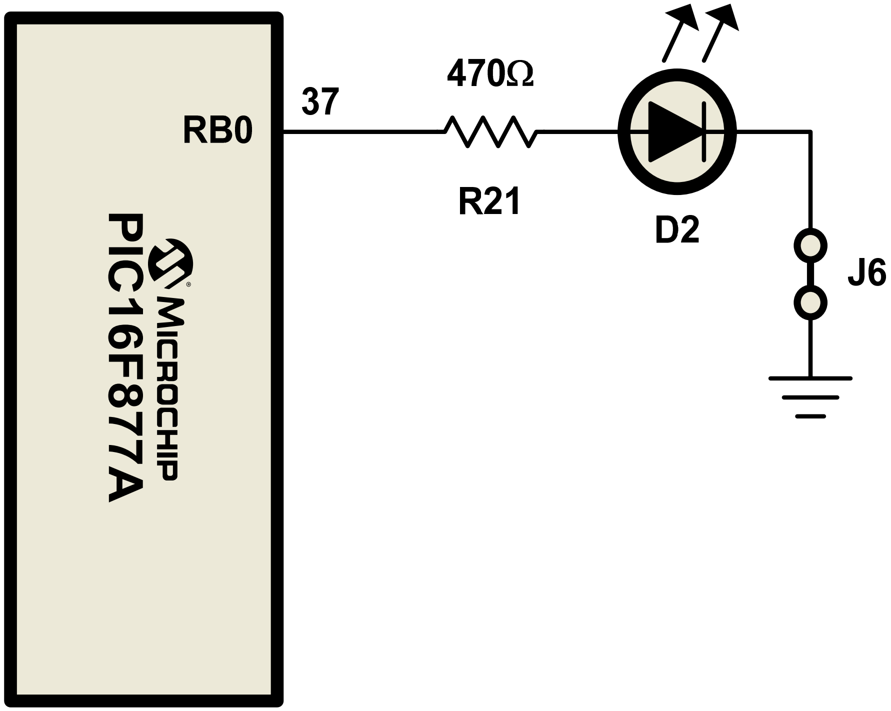
```
LIST   p=16f877a, r=dec
```

- Good programming practice suggests that a number's radix be specified explicitly:

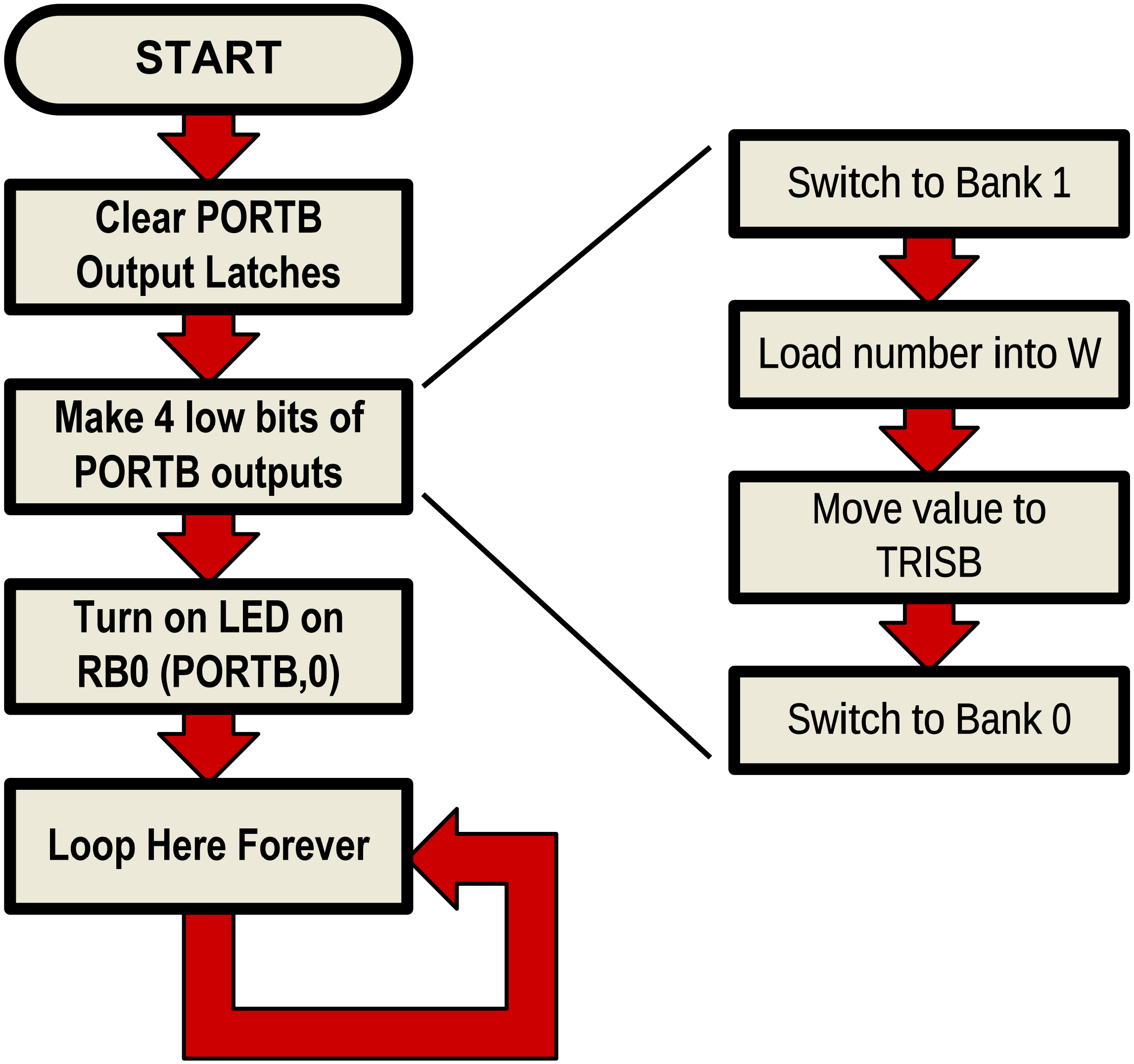
Radix	MPASM Syntax
Binary	b'10101010'
Decimal	d'25' or .25
Hexadecimal	h'2A' or 0x2A

Lab 1: The Task

- Turn on LED connected to bit 0 of PORTB (RB0)



Lab 1: Program Structure



1 Instruction

4 Instructions

1 Instruction

1 Instruction: `goto $`
(Go to self)

Lab 1: Template

Lab 1: “Hello, world!” for Microcontrollers

```
          LIST p=16f877a

          #include <p16f877a.inc>

          org     0x0000
RESET_V   goto    START           ;Reset Vector

START     {1st Instruction}       ;Clear PORTB output latches
          {2nd Instruction}       ;Switch to bank 1
          {3rd Instruction}       ;Load value to make lower 4 bits outputs
          {4th Instruction}       ;Move value to TRISB
          {5th Instruction}       ;Switch to bank 0
          {6th Instruction}       ;Turn on LED on RB0

          goto $                  ;Loop here forever

          END
```

Lab 1: Solution

Lab 1: “Hello, world!” for Microcontrollers

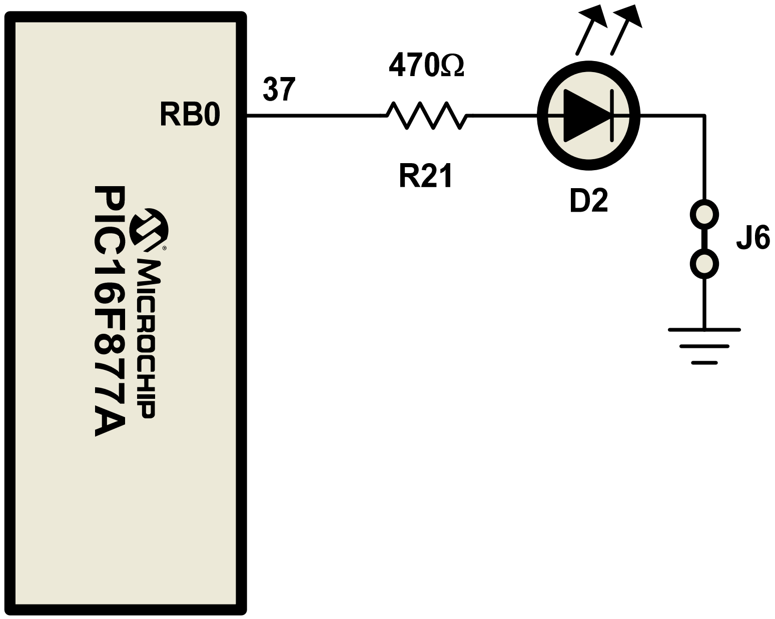
```
            LIST p=16f877a

            #include <p16f877a.inc>

            org     0x0000
RESET_V     goto    START           ;Reset Vector

START       clrf    PORTB           ;Clear PORTB output latches
            bsf     STATUS,RP0      ;Switch to bank 1
            movlw   b’11110000'     ;Load value to make lower 4 bits outputs
            movwf   TRISB           ;Move value to TRISB
            bcf     STATUS,RP0      ;Switch to bank 0
            bsf     PORTB,0         ;Turn on LED on RB0

            goto $                  ;Loop here forever

            END
```

Lab 1: Results

- **You have learned:**
  - How to program a device and run the code using the MPLAB® ICD2
  - How to configure an I/O port
  - How to manipulate I/O pins
  - How to code an infinite loop (the equivalent of while(1) in C)

Lab 2: The Task

- Make the LED connected to bit 0 of PORTB (RB0) blink



Lab 2: The Task

- A delay is required to make the blinking slow enough for the human eye
- At 4MHz, one instruction executes in 1µs
- A 16-bit software counter is sufficient to implement the delay

Naming Registers/Constants

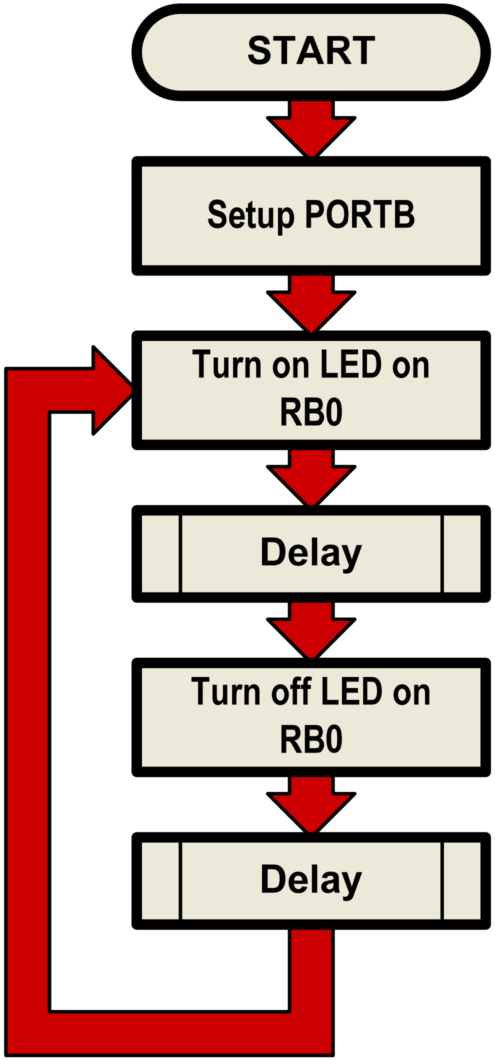
- **Equate Method:**

```
MyReg0 equ 0x20  ;MyReg0 = 0x20
MyReg1 equ 0x21  ;MyReg1 = 0x21
MyReg2 equ 0x23  ;MyReg2 = 0x23
```

- **Constant Block Method:**

```
CBLOCK 0x20
  MyReg0             ;MyReg0 = 0x20
  MyReg1: 2          ;MyReg1 = 0x21
  MyReg2             ;MyReg2 = 0x23
ENDC
```

Lab 2: Program Structure



Taken from Lab 1

1 Instruction

1 Instruction
Subroutine Call

1 Instruction

1 Instruction
Subroutine Call

Lab 2: Program Structure

Delay Subroutine

Hint: Use **decfsz**

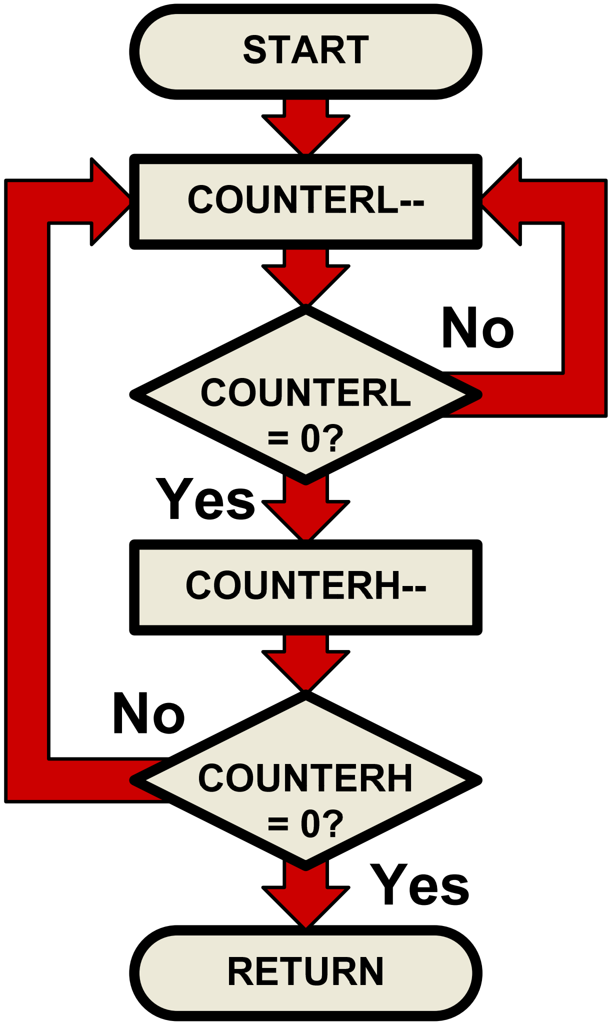
1 Instruction

1 Instruction

1 Instruction

1 Instruction

1 Instruction



Lab 2: Template – Part 1

Lab 2: Blinking LED

```
          LIST p=16f877a

          #include <p16f877a.inc>

          cblock 0x020
            COUNTERL
            COUNTERH
          endc

          org     0x0000
RESET_V   goto    START         ;Reset Vector

START     clrf    PORTB         ;Clear PORTB output latches
          bsf     STATUS,RP0    ;Switch to bank 1
          movlw   b’11110000'   ;Load value to make lower 4 bits outputs
          movwf   TRISB         ;Move value to TRISB
          bcf     STATUS,RP0    ;Switch to bank 0

;CONTINUED ON NEXT SLIDE
```

Lab 2: Template – Part 2

Lab 2: Blinking LED - Continued

```
LOOP        {1st Instruction}         ;Turn on LED on RB0
            {2nd Instruction}         ;Call delay routine
            {3rd Instruction}         ;Turn off LED on RB0
            {4th  Instruction}        ;Call delay routine
            {5th  Instruction}        ;Repeat main loop

DELAY       {6th  Instruction}        ;Decrement COUNTERL
            {7th  Instruction}        ;If not zero, keep decrementing COUNTERL
            {8th  Instruction}        ;Decrement COUNTERH
            {9th  Instruction}        ;If not zero, decrement COUNTERL again
            {10th  Instruction}       ;Return to main routine

            END
```

Lab 2: Solution – Part 1

Lab 2: Blinking LED

```
          LIST p=16f877a

          #include <p16f877a.inc>

          cblock 0x020
            COUNTERL
            COUNTERH
          endc

          org    0x0000
RESET_V   goto   START         ;Reset Vector

START     clrf   PORTB         ;Clear PORTB output latches
          bsf    STATUS,RP0    ;Switch to bank 1
          movlw  b’11110000'   ;Load value to make lower 4 bits outputs
          movwf  TRISB         ;Move value to TRISB
          bcf    STATUS,RP0    ;Switch to bank 0

;CONTINUED ON NEXT SLIDE
```

Lab 2: Solution – Part 2

Lab 2: Blinking LED - Continued

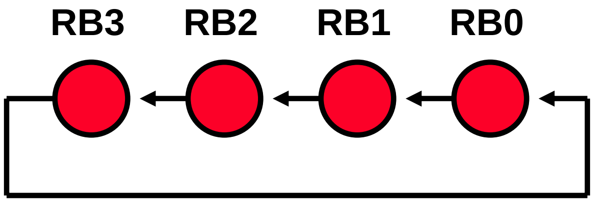
```
LOOP      bsf     PORTB,0     ;Turn on LED on RB0
          call    DELAY       ;Call delay routine
          bcf     PORTB,0     ;Turn off LED on RB0
          call    DELAY       ;Call delay routine
          goto    LOOP        ;Repeat main loop

DELAY     decfsz  COUNTERL    ;Decrement COUNTERL
          goto    DELAY       ;If not zero, keep decrementing COUNTERL
          decfsz  COUNTERH    ;Decrement COUNTERH
          goto    DELAY       ;If not zero, decrement COUNTERL again
          return

          END
```

Lab 2: Results

- **You have learned:**
  - How to define register labels
  - How to implement a loop
  - How to implement software delays
  - How to use a “skip” instruction
  - How to call a subroutine

Lab 3: The Task

- **Using one of the rotate instructions, “move” the illuminated LED across the lower 4 bits of PORTB. When it reaches one side, send it back to the start.**



Lab 3: Program Structure

Same setup code from Lab 1

1 Instruction

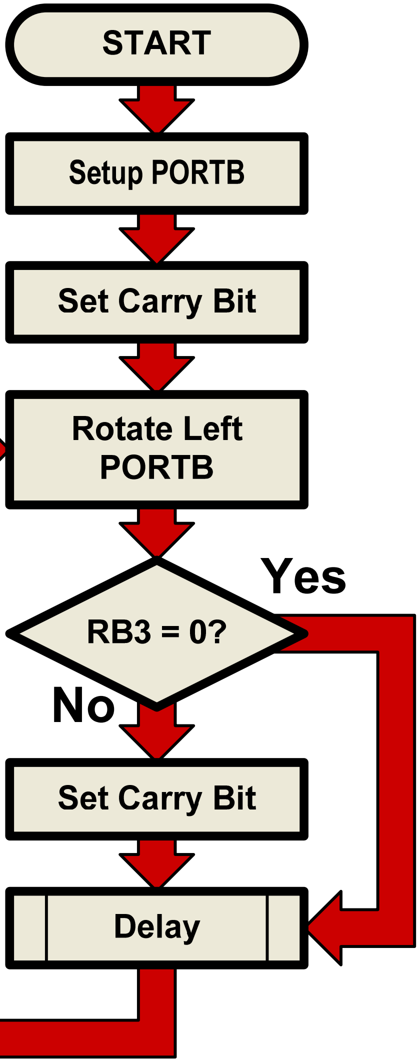
1 Instruction

1 Instruction

1 Instruction

1 Instruction

Call same subroutine from Lab 2



Rember: The rotate instructions operate on 9-bits, with the Carry bit in the STATUS register as the 9th bit

Lab 3: Template – Part 1

Lab 3: Rotating LED

```
          LIST p=16f877a

          #include <p16f877a.inc>

          cblock 0x020
            COUNTERL
            COUNTERH
          endc

          org    0x0000
RESET_V   goto   START        ;Reset Vector

START     clrf   PORTB        ;Clear PORTB output latches
          bsf    STATUS,RP0   ;Switch to bank 1
          movlw  b'11110000'  ;Load value to make lower 4 bits outputs
          movwf  TRISB        ;Move value to TRISB
          bcf    STATUS,RP0   ;Switch to bank 0

;CONTINUED ON NEXT SLIDE
```

Lab 3: Template – Part 2

Lab 3: Rotating LED - Continued

```
            {1st Instruction}      ;Set carry bit for initial rotate
LOOP        {2nd Instruction}      ;Rotate PORTB to left
            {3rd Instruction}      ;Call delay routine
            {4th  Instruction}     ;Is the LED on RB3 (PORTB,3) on?
            {5th  Instruction}     ;If yes, set the Carry bit
            {6th  Instruction}     ;Repeat main loop

DELAY       decfsz  COUNTERL       ;Decrement COUNTERL
            goto    DELAY          ;If not zero, keep decrementing COUNTERL
            decfsz  COUNTERH       ;Decrement COUNTERH
            goto    DELAY          ;If not zero, decrement COUNTERL again
            return                 ;Return to main subroutine

            END
```

Lab 3: Solution – Part 1

Lab 3: Rotating LED

```
          LIST p=16f877a

          #include <p16f877a.inc>

          cblock 0x020
            COUNTERL
            COUNTERH
          endc

          org     0x0000
RESET_V   goto    START         ;Reset Vector

START     clrf    PORTB         ;Clear PORTB output latches
          bsf     STATUS,RP0    ;Switch to bank 1
          movlw   b’11110000'   ;Load value to make lower 4 bits outputs
          movwf   TRISB         ;Move value to TRISB
          bcf     STATUS,RP0    ;Switch to bank 0

;CONTINUED ON NEXT SLIDE
```

Lab 3: Solution – Part 2

Lab 3: Rotating LED - Continued

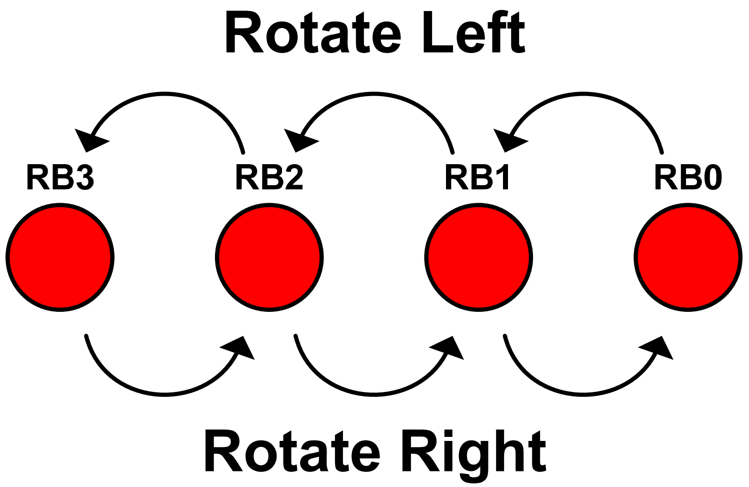
```
          bsf    STATUS,C     ;Set carry bit for initial rotate
LOOP      rlf    PORTB,f      ;Rotate PORTB to left
          call   DELAY        ;Call delay routine
          btfsc  PORTB,3      ;Is the LED on RB3 (PORTB,3) on?
          bsf    STATUS,C     ;If yes, set the Carry bit
          goto   LOOP         ;Repeat main loop

DELAY     decfsz COUNTERL     ;Decrement COUNTERL
          goto   DELAY        ;If not zero, keep decrementing COUNTERL
          decfsz COUNTERH     ;Decrement COUNTERH
          goto   DELAY        ;If not zero, decrement COUNTERL again
          return              ;Return to main subroutine

          END
```

Lab 3: Results

- **You have learned:**
  - How to use the rotate instructions
  - How to use the bit test & skip instructions

Lab 4: The Task

- Same as Lab 3, but this time make the direction of rotation change when the LED is rotated to either end



Lab 4: Program Structure

Same setup code from Lab 1

`bsf STATUS,C`

1 Instruction

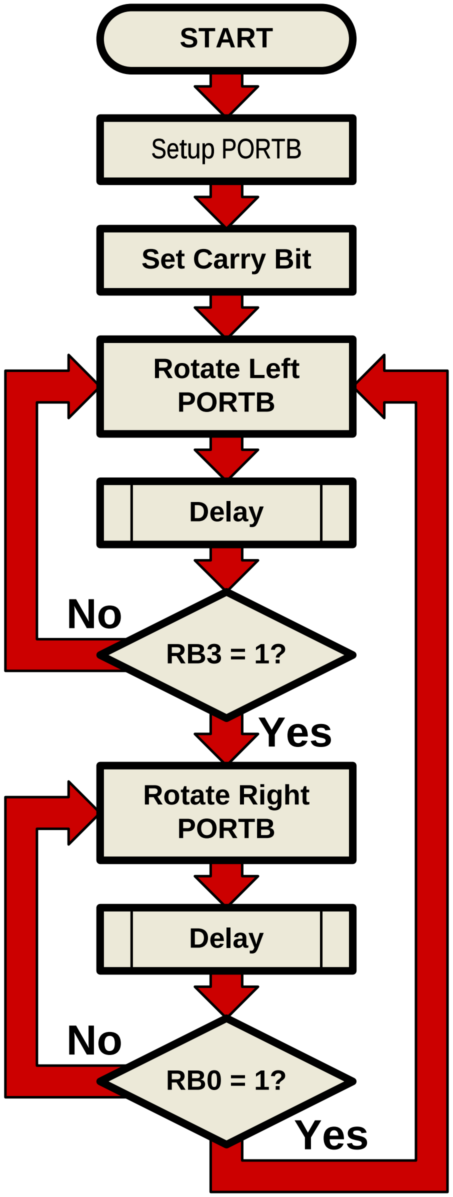
1 Instruction Subroutine Call

2 Instructions

1 Instruction

1 Instruction Subroutine Call

2 Instructions



Lab 4: Template

- **Setup is identical to Lab 3 up to the LOOP**

Lab 3: Rotating LED - Continued

```
          bsf     STATUS,C    ;Set carry bit for initial rotate

LEFT      {1st Instruction}   ;Rotate PORTB to left
          {2nd Instruction}   ;Call delay routine
          {3rd Instruction}   ;Is the LED on RB3 (PORTB,3) on?
          {4th Instruction}   ;if no, rotate left again

RIGHT     {5th Instruction}   ;Rotate PORTB to right
          {6th Instruction}   ;Call delay routine
          {7th Instruction}   ;Is the LED on RB0 (PORTB,0) on?
          {8th Instruction}   ;if no, rotate right again
          {9th Instruction}   ;if yes, rotate left

DELAY     decfsz  COUNTERL    ;Decrement COUNTERL
          goto    DELAY       ;If not zero, keep decrementing COUNTERL
          decfsz  COUNTERH    ;Decrement COUNTERH
          goto    DELAY       ;If not zero, decrement COUNTERL again
          return              ;Return to main subroutine

          END
```

Lab 4: Solution

Lab 3: Rotating LED - Continued

```
          bsf    STATUS,C     ;Set carry bit for initial rotate

LEFT      rlf    PORTB,f      ;Rotate PORTB to left
          call   DELAY        ;Call delay routine
          btfss  PORTB,3      ;Is the LED on RB3 (PORTB,3) on?
          goto   LEFT         ;if no, rotate left again

RIGHT     rrf    PORTB,f      ;Rotate PORTB to right
          call   DELAY        ;Call delay routine
          btfss  PORTB,0      ;Is the LED on RB0 (PORTB,0) on?
          goto   RIGHT        ;if no, rotate right again
          goto   LEFT         ;if yes, rotate left

DELAY     decfsz COUNTERL     ;Decrement COUNTERL
          goto   DELAY        ;If not zero, keep decrementing COUNTERL
          decfsz COUNTERH     ;Decrement COUNTERH
          goto   DELAY        ;If not zero, decrement COUNTERL again
          return              ;Return to main subroutine

          END
```

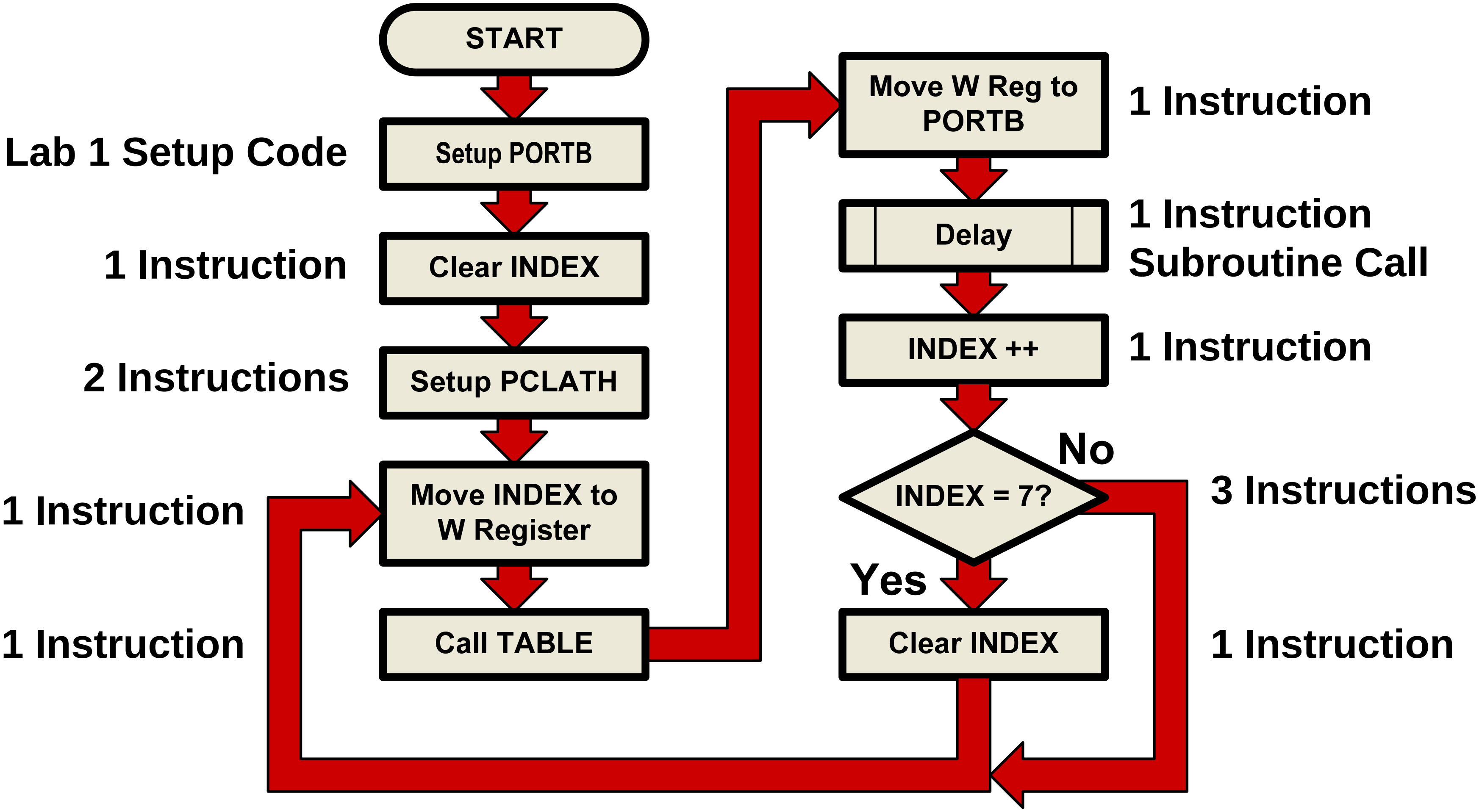
Lab 4: Results

- **You have learned:**
  - How to make decisions in software and take different courses of action

Lab 5: The Task

- Use a lookup table to obtain the bit pattern to be displayed on PORTB

Lab 5: Program Structure



Lab 5: Template – Part 1

Lab 5: Lookup Table

```
          LIST p=16f877a

          #include <p16f877a.inc>

          cblock 0x020
            COUNTERL
            COUNTERH
          endc

          org     0x0000
RESET_V   goto    START         ;Reset Vector

START     clrf    PORTB         ;Clear PORTB output latches
          bsf     STATUS,RP0    ;Switch to bank 1
          movlw   b’11110000'   ;Load value to make lower 4 bits outputs
          movwf   TRISB         ;Move value to TRISB
          bcf     STATUS,RP0    ;Switch to bank 0
          {1st Instruction}     ;Clear index into table
          {2nd Instruction}     ;Load W with high byte of TABLE address
          {3rd Instruction}     ;Move W to PCLATH

;CONTINUED ON NEXT SLIDE
```

UNIVERSITY OF MICROCHIP
UofM
MASTERs 2007

Lab 5: Template – Part 2

Lab 3: Lookup Table - Continued

```
LOOP      {4th Instruction}          ;Move INDEX to W
          {5th Instruction}          ;Call TABLE
          {6th Instruction}          ;Move W to PORTB
          {7th Instruction}          ;Call delay
          {8th Instruction}          ;Increment INDEX
          {9th Instruction}          ;Load W with 0x07
          {10th Instruction}         ;Subtract W from INDEX, result in W
          {11th Instruction}         ;Is Z bit in STATUS set?
          {12th Instruction}         ;if yes, clear INDEX
          {13th Instruction}         ;Repeat loop

DELAY     decfsz  COUNTERL           ;Decrement COUNTERL
          goto    DELAY              ;If not zero, keep decrementing COUNTERL
          decfsz  COUNTERH           ;Decrement COUNTERH
          goto    DELAY              ;If not zero, decrement COUNTERL again
          return                     ;Return to main subroutine

;CONTINUED ON NEXT SLIDE
```

Lab 5: Template – Part 3

Lab 3: Lookup Table - Continued

```
TABLE       addwf   PCL,f         ;Add offset to program counter
            retlw   b’00000001'   ;Table entry 0
            retlw   b’00000011'   ;Table entry 1
            retlw   b’00000111'   ;Table entry 2
            retlw   b’00001111'   ;Table entry 3
            retlw   b’00001110'   ;Table entry 4
            retlw   b’00001100'   ;Table entry 5
            retlw   b’00001000'   ;Table entry 6

            END
```

Lab 5: Solution – Part 1

Lab 5: Lookup Table

```
          LIST p=16f877a

          #include <p16f877a.inc>

          cblock 0x020
            COUNTERL
            COUNTERH
          endc

          org     0x0000
RESET_V   goto    START         ;Reset Vector

START     clrf    PORTB         ;Clear PORTB output latches
          bsf     STATUS,RP0    ;Switch to bank 1
          movlw   b’11110000'   ;Load value to make lower 4 bits outputs
          movwf   TRISB         ;Move value to TRISB
          bcf     STATUS,RP0    ;Switch to bank 0
          clrf    INDEX         ;Clear index into table
          movlw   HIGH TABLE    ;Load W with high byte of TABLE address
          movwf   PCLATH        ;Move W to PCLATH

;CONTINUED ON NEXT SLIDE
```

Lab 5: Solution – Part 2

Lab 3: Lookup Table - Continued

```
LOOP      movf    INDEX,w      ;Move INDEX to W
          call    TABLE        ;Call TABLE
          movwf   PORTB        ;Move W to PORTB
          call    DELAY        ;Call delay
          incf    INDEX,f      ;Increment INDEX
          movlw   0x07         ;Load W with 0x07
          subwf   INDEX,w      ;Subtract W from INDEX, result in W
          btfsc   STATUS,Z     ;Is Z bit in STATUS set?
          clrf    INDEX        ;if yes, clear INDEX
          goto    LOOP    ;Repeat loop

DELAY     decfsz  COUNTERL     ;Decrement COUNTERL
          goto    DELAY        ;If not zero, keep decrementing COUNTERL
          decfsz  COUNTERH     ;Decrement COUNTERH
          goto    DELAY        ;If not zero, decrement COUNTERL again
          return               ;Return to main subroutine

;CONTINUED ON NEXT SLIDE
```

Lab 5: Solution – Part 3

Lab 3: Lookup Table - Continued

```
TABLE       addwf   PCL,f         ;Add offset to program counter
            retlw   b’00000001'   ;Table entry 0
            retlw   b’00000011'   ;Table entry 1
            retlw   b’00000111'   ;Table entry 2
            retlw   b’00001111'   ;Table entry 3
            retlw   b’00001110'   ;Table entry 4
            retlw   b’00001100'   ;Table entry 5
            retlw   b’00001000'   ;Table entry 6

            END
```

Lab 5: Results

- **You have learned:**
  - How to implement a lookup table
  - How to retrieve data from a lookup table
  - How to call a subroutine on another page
  - How to perform a computed goto

Summary

- PIC16 Architecture
- PIC16 Instruction Set
- PIC16 Memory Organization
- Simple Programming Techniques

References

- PIC® MCU Mid-Range Family Reference Manual (DS33023A) *Microchip Technology*
- Programming and Customizing PICmicro Microcontrollers *by Myke Predko*
- Design with PIC® Microcontrollers *by John B. Peatman*

References

- **123 PIC® Microcontroller Experiments for the Evil Genius by Myke Predko**

Thank You

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoq, KeeLoq logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.