PIC Microcontroller and Embedded Systems

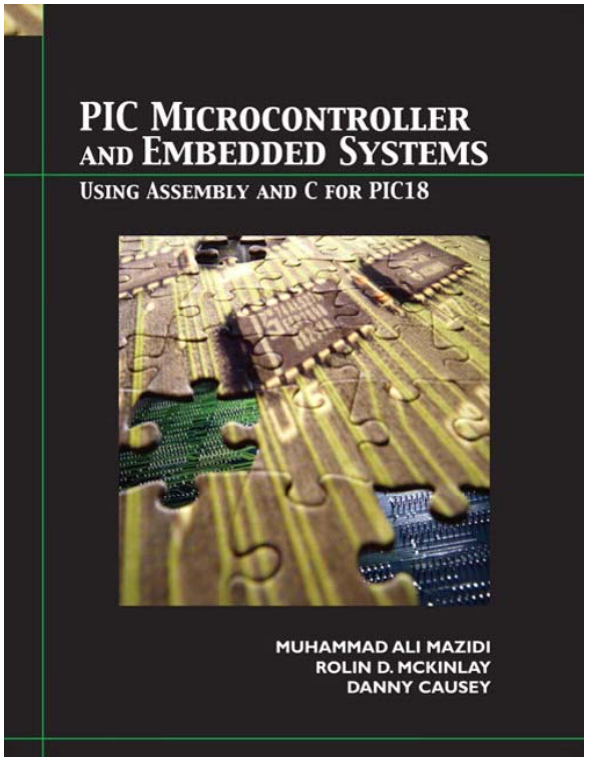
Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 1: The PIC Microcontrollers: History and Features

- Microcontroller and Embedded Processors
- Overview of the PIC18 Family



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

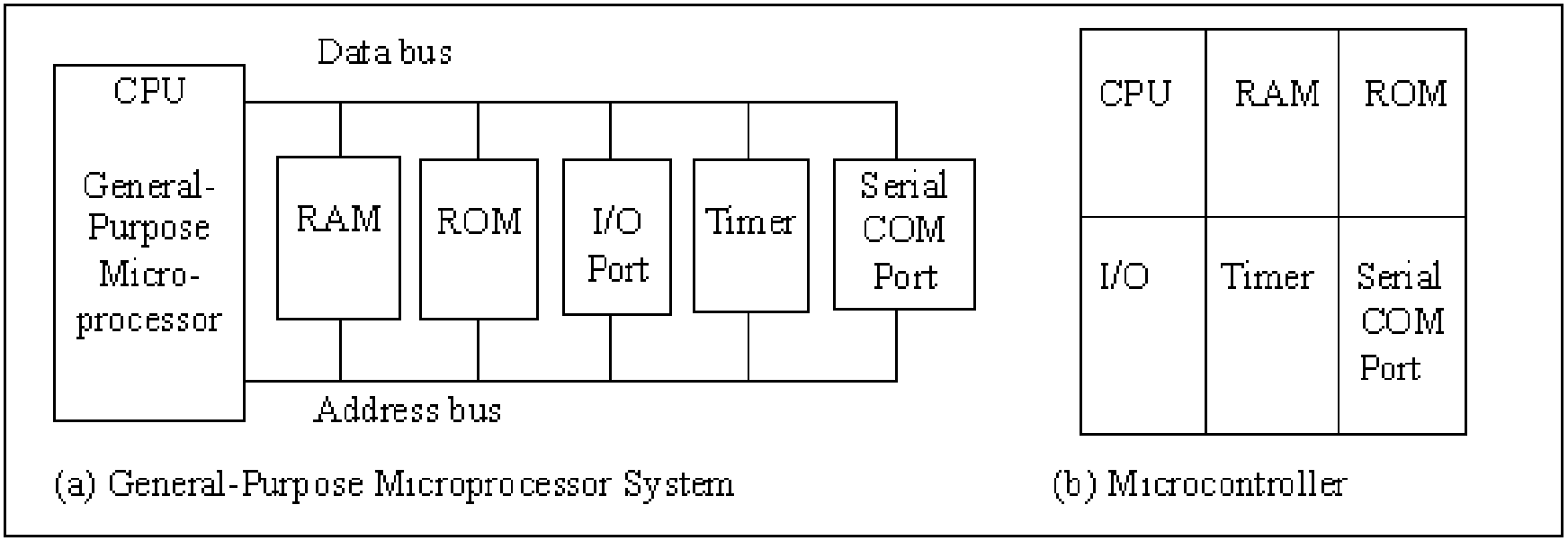
Objective

- Compare and contrast uP and uC
- Describe the advantages of uC
- Explain the concept of ES
- Describe criteria for considering a uC
- Compare and contrast the various of the PIC Family
- Compare the PIC with uC offered by others

Microcontroller and Embedded Processors

- Microcontroller VS General purpose uP
- uC for embedded systems
- X86 PC Embedded Application

Figure 1-1. Microprocessor System Contrasted With Microcontroller System



Choosing a uController

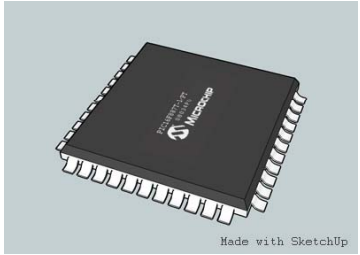
- The major 8-bit
  - Freescale Semiconductor's (formerly Motorola) 68HC08/68HC11
  - Intel's 8051
  - Atmel's AVR
  - Zilog's Z8
  - PIC from Microchip Technology

Criteria for Choosing uController

1. Meeting the computing needs of the task at hand efficiency and cost effectively
2. Availability of SW and HW development tools
   - Compilers
   - Assemblers
   - Debuggers
   - Emulators
3. Wide availability and reliable source

Criteria for Choosing uController

- Meeting the computing needs of the task at hand efficiency and cost effectively
  - Determine its type, 8-bit,16-bit or 32-bit
  - Speed
  - Packaging (40-Pin or QFP)
  - Power consumption
  - The amount of RAM and ROM
  - The number of I/O pins and the timer
  - Cost per unit
  - Ease of upgrade.



uC Data width

- 8-bit Microcontrollers
  - PIC10, PIC12, PIC14
  - PIC16, PIC17, PIC18
- 16-bit Microcontrollers
  - PIC24F, PIC24H
- 32-bit Microcontrollers
  - PIC32
- 16-bit Digital Signal Controllers
  - dsPIC30, dsPIC33F

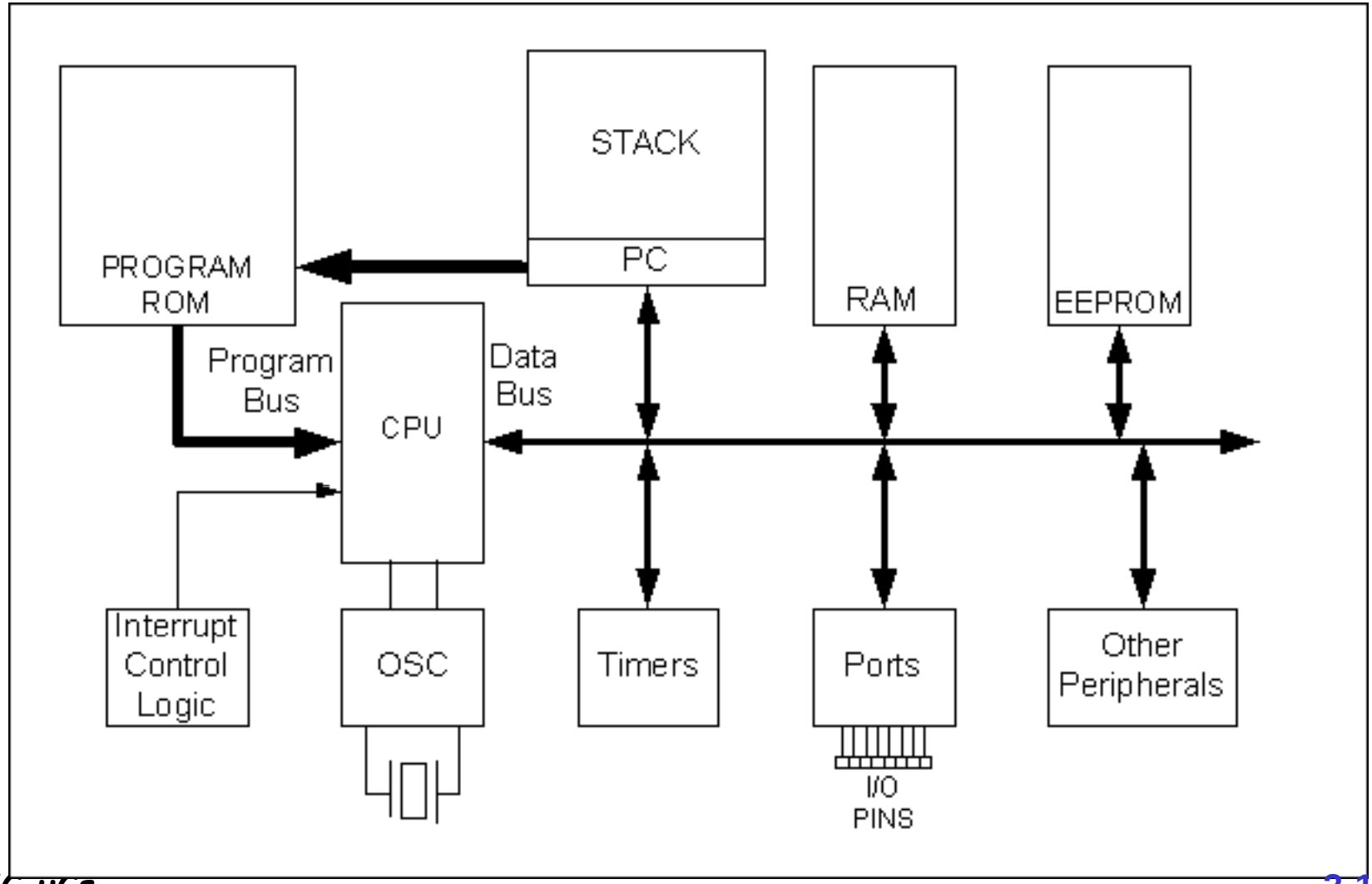
Overview of the PIC18 Family

- An 8-bit uController called PIC is introduces in 1989 by Microchip Technology Corporation
- It includes
  - Small Data Ram
  - Few bytes of Rom
  - One timer
  - I/O ports

PIC 18 Feathers

- RISC Architecture
- On-chip program, Code, ROM
- Data EEPROM
- Timers
- ADC
- USART
- I/O Ports

Figure 1-2. Simplified View of a PIC Microcontroller



PIC18 Features

- RISC Architecture
- On chip Code ROM and Data RAM, Data EEPROM
- Timers
- ADC
- USART
- I/O ports

Figure 1-3. PIC18 Block Diagram

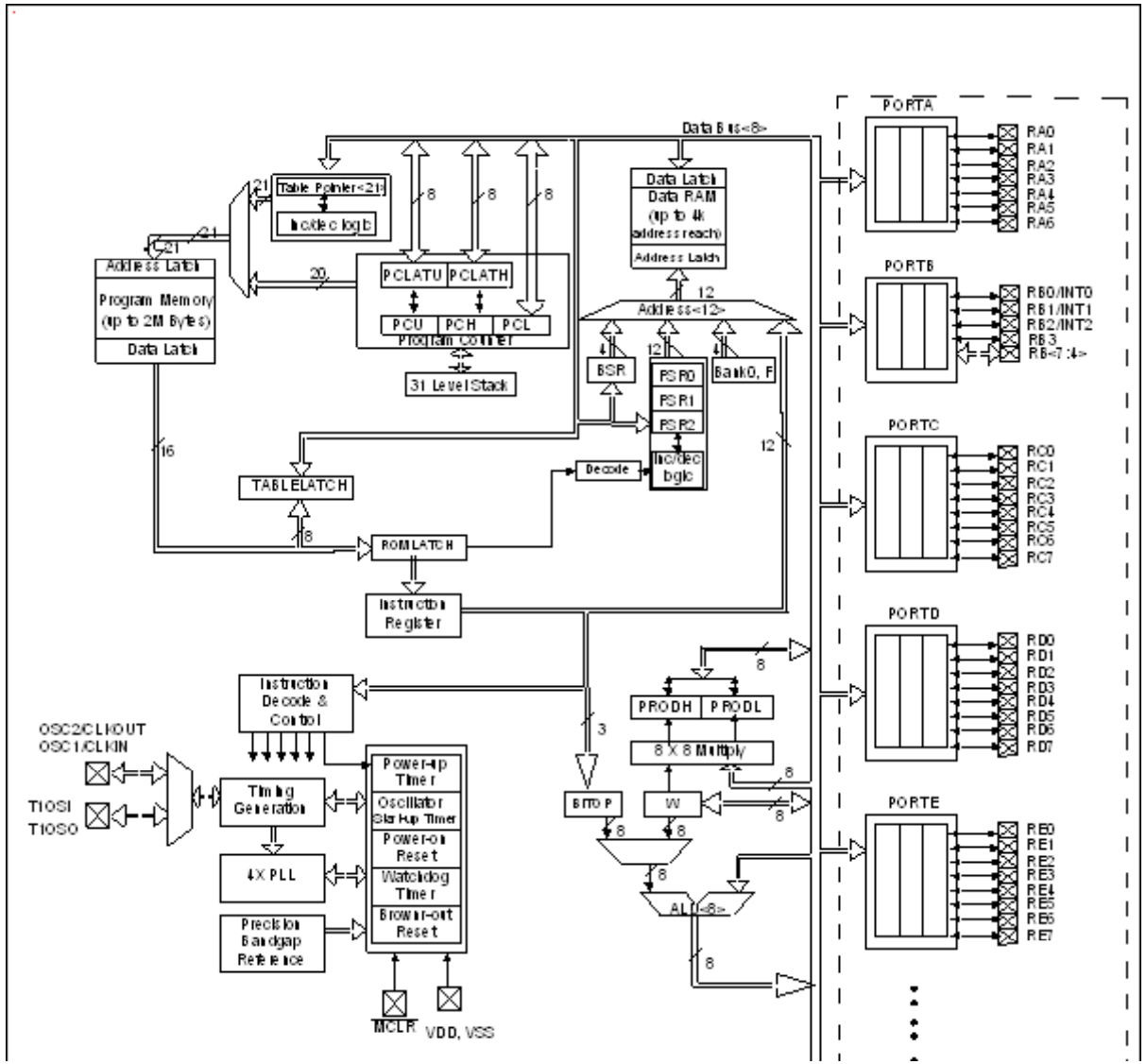
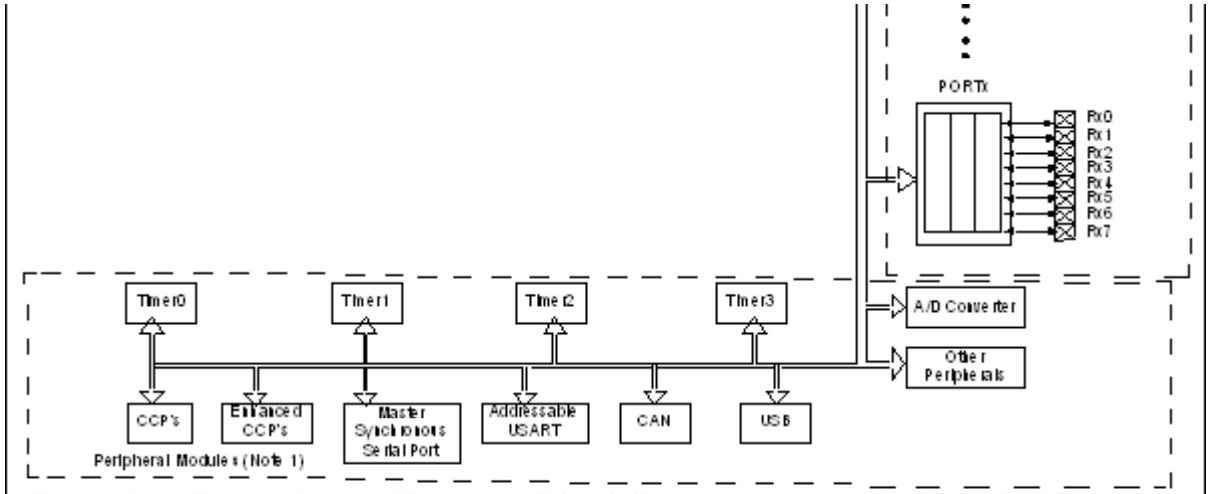


Figure 1-3. PIC18 Block Diagram (continued)



Note 1: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

Figure 1-4. PIC16 Block Diagram

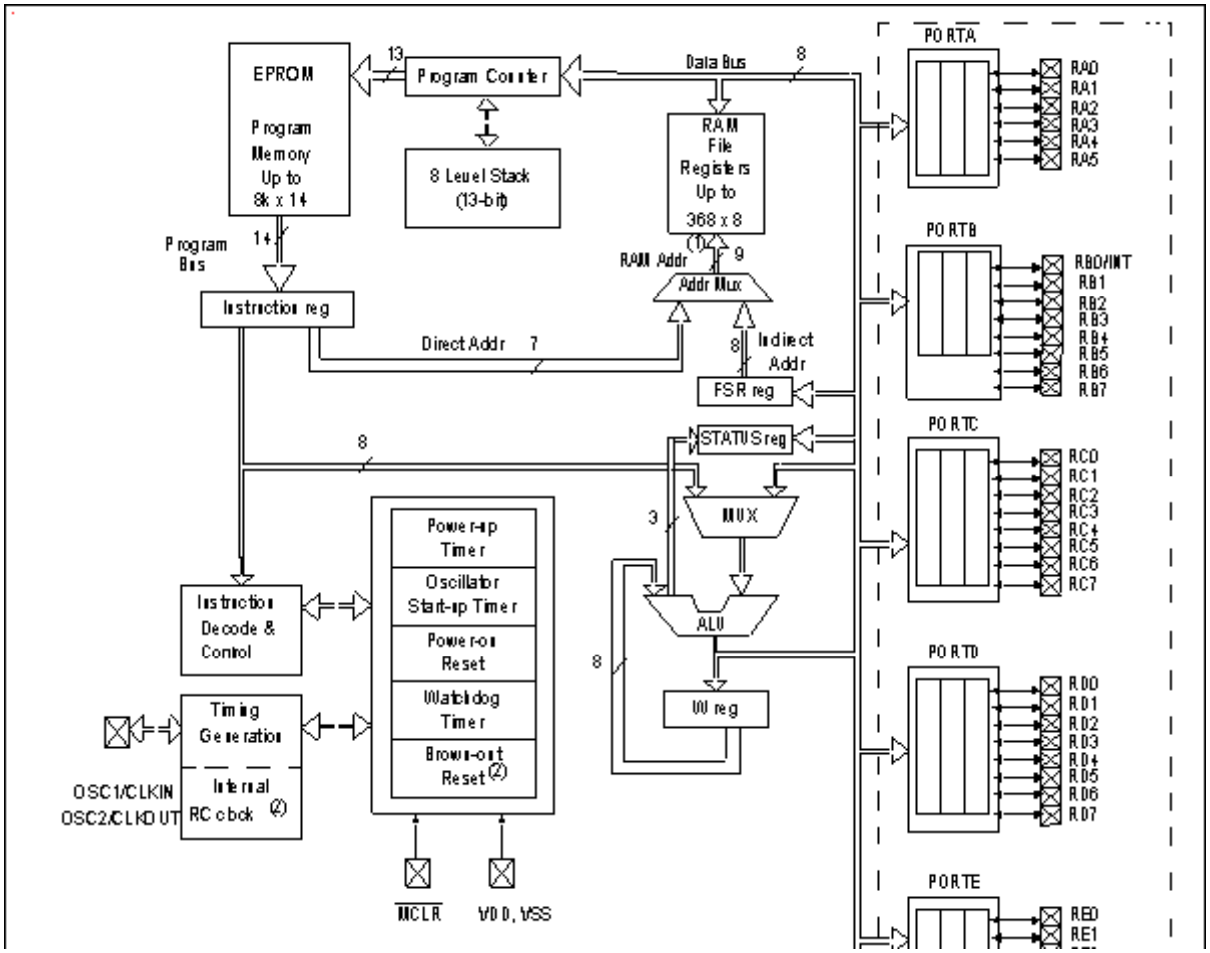
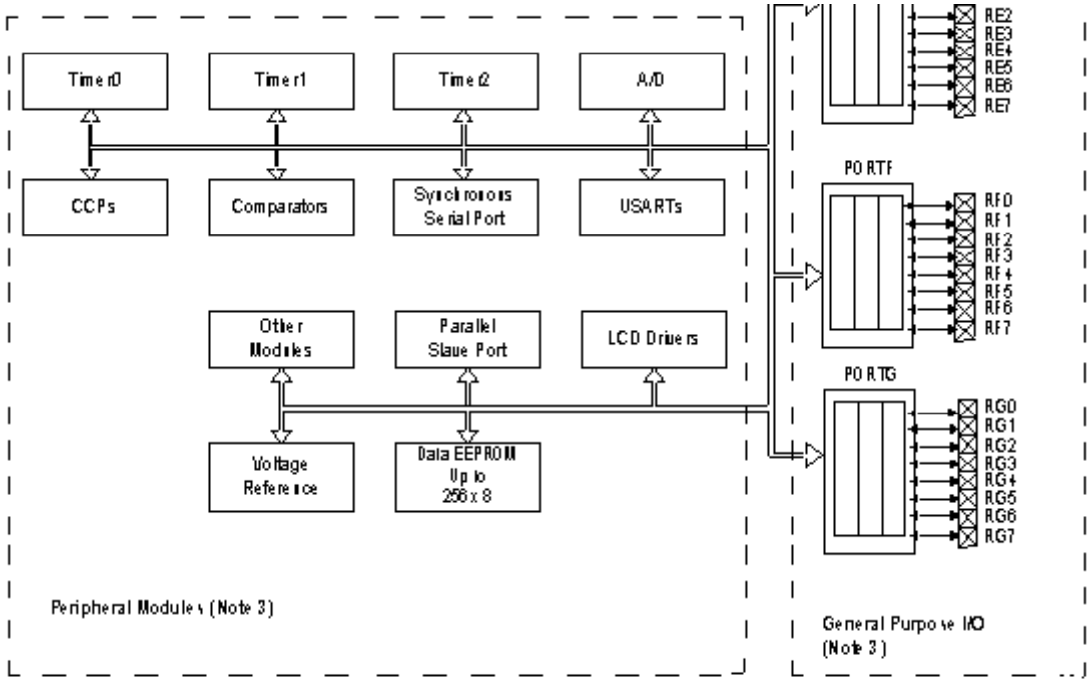
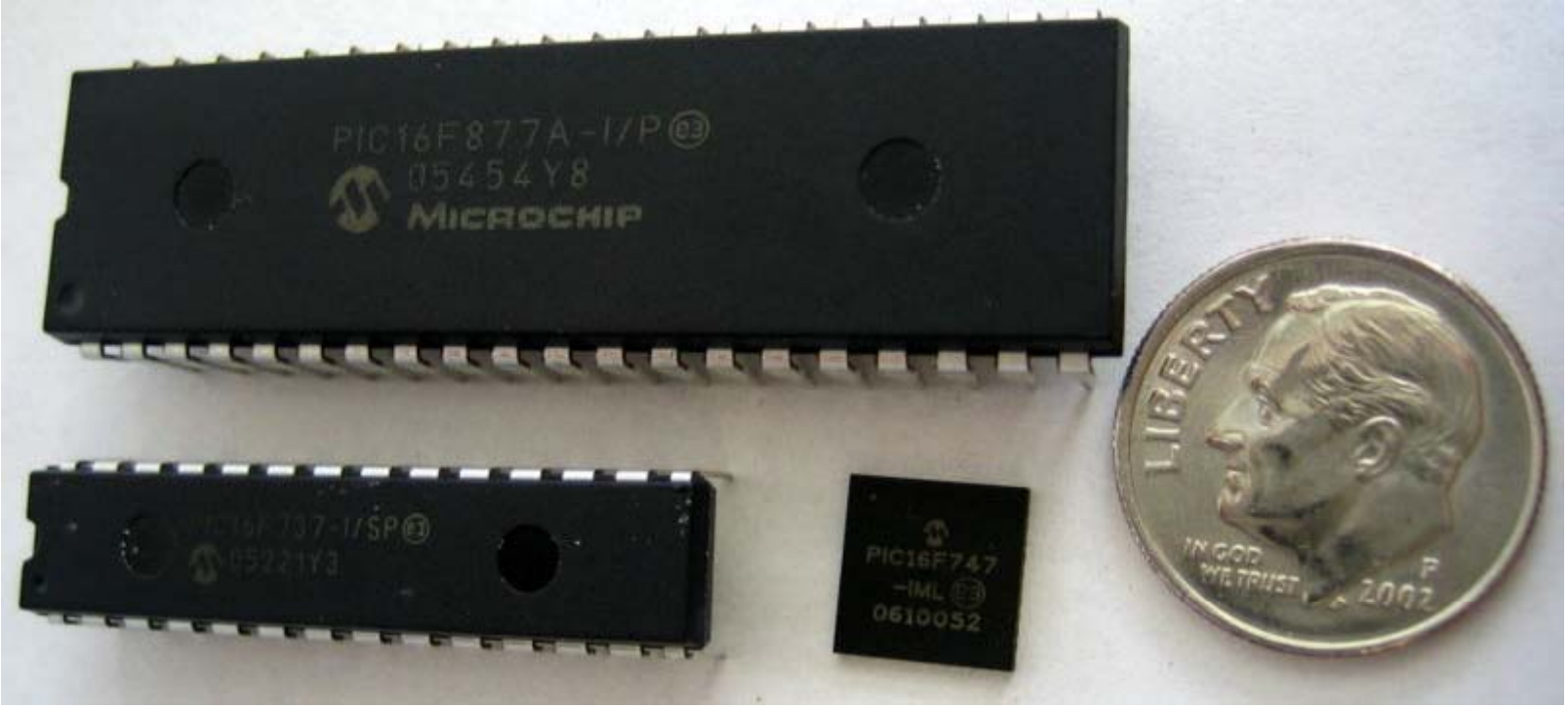


Figure 1-4. PIC16 Block Diagram (continued)



Note 1: The high order bits of the Direct Address for the RAM are from the STATUS register.
2: Not all devices have this feature, please refer to device data sheet.
3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

uC


PIC16F877A-I/P
05454Y8
MICROCHIP
PIC16F747
-IML
0610052
LIBERTY
IN GOD WE TRUST
2002

PIC uC program ROM

- PIC exists in terms of different speed and the amount of on-chip RAM/ROM
- Compatibility is restricted as far as the instructions are concerns.

PIC uC Program ROM

- PIC 18 can support up to 2MB
- Generally, they come with 4KB – 128KB
- Available in flash, OTP, UV-EPROM, and masked.

UV-EPROM

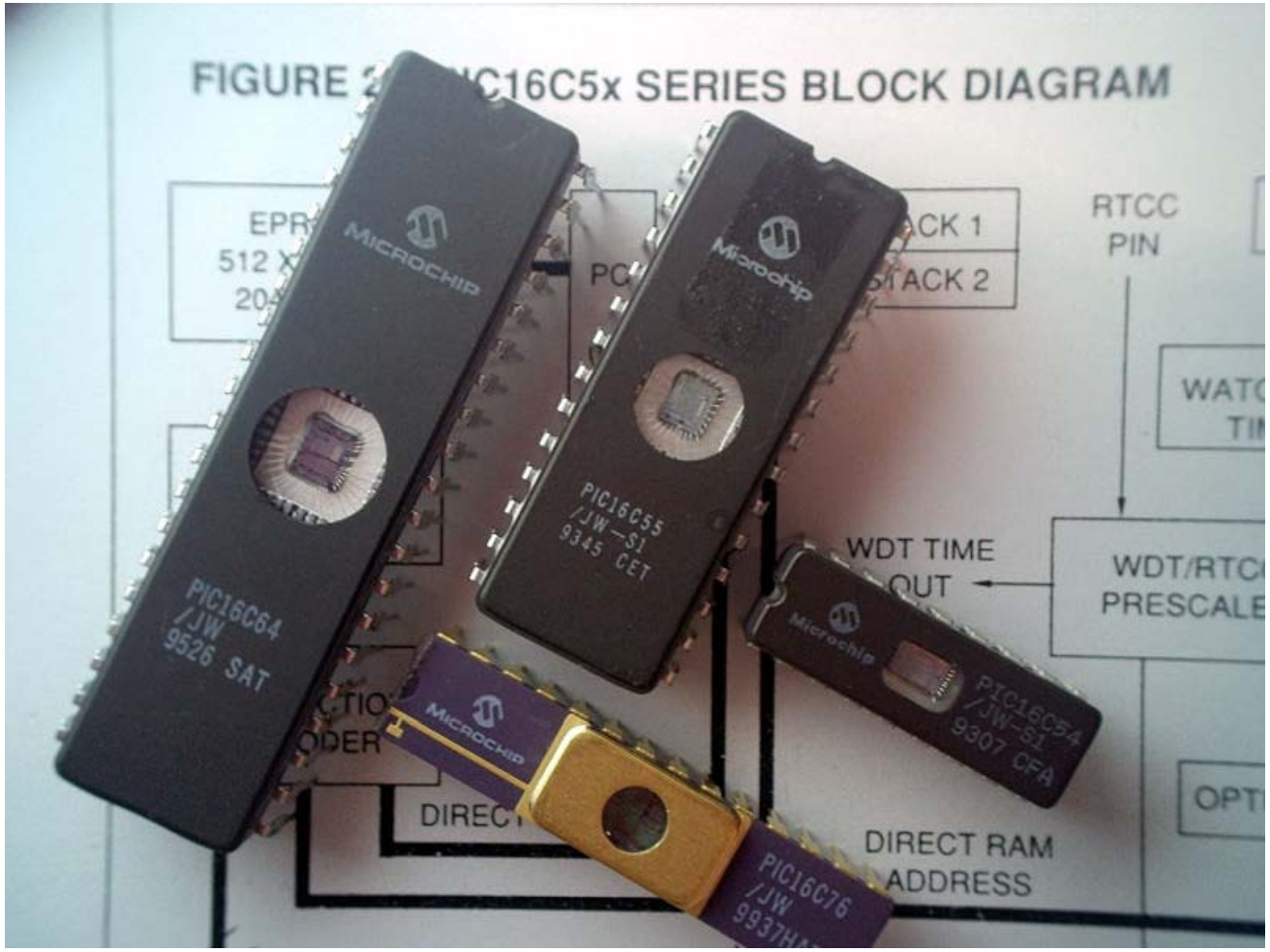


FIGURE 2
C16C5x SERIES BLOCK DIAGRAM
RTCC
PIN
WDT TIME
OUT
WDT/RTC
PRESCALE
DIRECT RAM
ADDRESS
MICROCHIP
PIC16C64
/JW
9526 SAT
PIC16C55
/JW-S1
9345 CET
PIC16C54
/JW-S1
9307 CFA
PIC16C76
/JW

PIC18Fxxxx with flash

- Used for product development

PIC18Cxxxx and Masked PIC

- OTP
  - One time programmable
  - C indicates the OTP RPM
  - Used for mass production
  - Cheaper
- Masked
  - program will be burned into the PIC chip during the fabrication process

PIC uC data RAM and EEPROM

- Max. 4096 Bytes (4 kB) of data RAM space.
- Data RAM space has two components
  - Varied GPR, General Purpose RAM
    - For read/write and data manipulation
    - Divided into banks of 256 B
  - Fixed SFR, Special Function Registers
- Some of PICs have a small amount of EEPROM
  - Used for critical data storing

PIC18 Microcontroller Family

Product	Program Memory Type	Program Memory Bytes	Data Memory RAM Bytes	Data Memory EEPROM Bytes	I/O Ports	ADC 10-bit	MSSP	USART	Other	CCP/ PWM	Timers 8/16-bit	Packages	Pins
PIC18F1220	FLASH	4K	256	256	16	7	—	1	6x PMM	1	1/3	DIP, SOIC, SSOP, QFN	18
PIC18F1320	FLASH	8K	256	256	16	7	—	1	6x PMM	1	1/3	DIP, SOIC, SSOP, QFN	18
PIC18F2220	FLASH	4K	512	256	23	10	I²C/SPI	1	6x PMM	2	1/3	DIP, SOIC	28
PIC18F2320	FLASH	8K	512	256	23	10	I²C/SPI	1	6x PMM	2	1/3	DIP, SOIC	28
PIC18C242	OTP	16K	512	—	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC	28
PIC18C252	OTP	32K	1536	—	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC	28
PIC18F242	FLASH	16K	512	256	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC, SSOP	28
PIC18F252	FLASH	32K	1536	256	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC, SSOP	28
PIC18F258	FLASH	32K	1536	256	22	5	I²C/SPI	1	CAN 2.0B	1	1/3	DIP, SOIC	28
PIC18F4220	FLASH	4K	512	256	34	13	I²C/SPI	1	6x PMM	2	1/3	DIP, TQFP, QFN	40/44
PIC18F4320	FLASH	8K	512	256	34	13	I²C/SPI	1	6x PMM	2	1/3	DIP, TQFP, QFN	40/44
PIC18C442	OTP	16K	512	—	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18C452	OTP	32K	1536	—	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F442	FLASH	16K	512	256	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F452	FLASH	32K	1536	256	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F458	FLASH	32K	1536	256	33	5	I²C/SPI	1	CAN 2.0B	1	1/3	DIP, PLCC, TQFP	40/44
PIC18C601	—	ROMless	1536	—	31	8	I²C/SPI	1	—	2	1/3	PLCC, TQFP	64/68
PIC18C658	OTP	32K	1536	—	52	12	I²C/SPI	1	CAN 2.0B	2	1/3	PLCC, TQFP	64/68
PIC18F6520	FLASH	32K	2048	1024	52	12	I²C/SPI	2	—	5	2/3	TQFP	64
PIC18F6620	FLASH	64K	3840	1024	52	12	I²C/SPI	2	—	5	2/3	TQFP	64
PIC18F6720	FLASH	128K	3840	1024	52	12	I²C/SPI	2	—	5	2/3	TQFP	64
PIC18C801	—	ROMless	1536	—	42	12	I²C/SPI	1	—	2	1/3	PLCC, TQFP	80/84
PIC18C858	OTP	32K	1536	—	68	16	I²C/SPI	1	CAN 2.0B	2	1/3	PLCC, TQFP	80/84
PIC18F8520	FLASH	32K	2048	1024	68	16	I²C/SPI	2	EMA	5	2/3	TQFP	80
PIC18F8620	FLASH	64K	3840	1024	68	16	I²C/SPI	2	EMA	5	2/3	TQFP	80
PIC18F8720	FLASH	128K	3840	1024	68	16	I²C/SPI	2	EMA	5	2/3	TQFP	80

Abbreviation: ADC = Analog-to-Digital Converter; CCP = Capture/Compare/PWM; I²C = Inter-Integrated Circuit Bus; PMM = Power Managed Mode; PWM = Pulse Width Modulation; SPI = Serial Peripheral Interface; USART = Universal Synchronous/Asynchronous Receiver/Transmitter

PIC uC peripherals

- CAN- (Controller Area Network),
- LIN- (Local Interconnect Network),
- USB- (Universal Serial Bus),
- I^2C- (Inter-Integrated Circuit),
- SPI- (Serial Peripheral Interface),
- Seriel or Ethernet Interface
- ADC - Analog Digital Converter
- USART- Universal Synchronous Asynchronous Receiver Transmitter

Chapter 1: Summary

- We have Compared between uP and uC
- We have described the advantages of uC
- We have given a simple introduction for PIC18

Next:
PIC Architecture and assembly language programming.

PIC Microcontroller and Embedded Systems

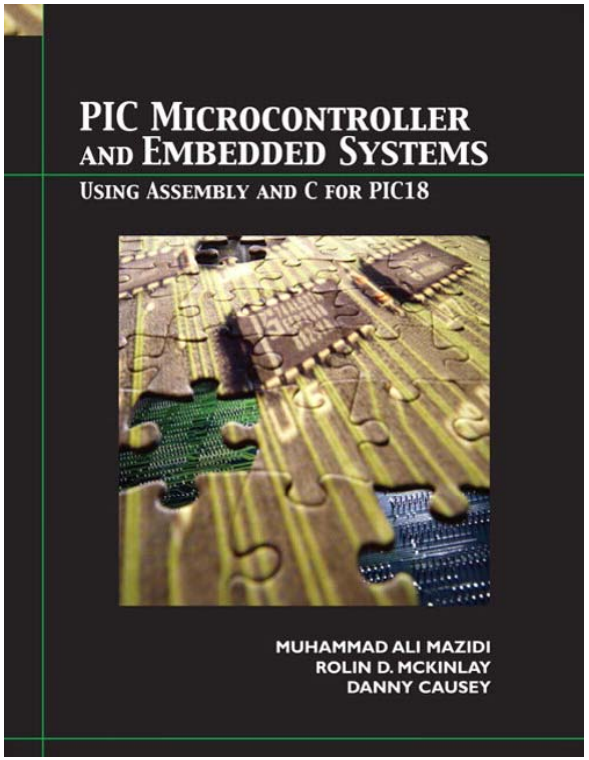
Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 2: PIC Architecture And Assembly Language Programming.

- The WREG Register
- The PIC File Register
- Using instruction with the default access bank



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

Outline

- PIC Status Register
- PIC data format and directive
- Intro. To PIC assembly language
- Assembling and linking a PIC program
- The Program Counter and program ROM space in the PIC
- RISC Architecture in the PIC
- Viewing Register and memory with MPLAB simulator

Objective

- Examine the data RAM fileReg of the PIC uC
- Manipulate data using the WREG & MOVE
- Perform simple operations such ADD and fileReg using and access bank in the PIC uC
- Explain the purpose of the status reg
- Discuss data RAM memory space allocation in the PIC uC
- List SFRs of the PIC uC
- Describe PIC data types and directives

The WREG Register

- Many registers for arithmetic and logic operation.
- The WREG (WORking Register) Register is one of the most widely used registers of the PIC
  - 8-bit register → any data larger than 8 bits must be broken into 8-bits chunks before it is processed.
  - There is only one .

D7	D6	D5	D4	D3	D2	D1	D0

MOVLW

- Moves 8-bit data into WREG
  - MOVLW k; move literal value k into WREG
- Example
  - MOVLW 25H
  - MOVLW A5H

- Is the following code correct?
  - MOVLW 9H
  - MOVLW A23H

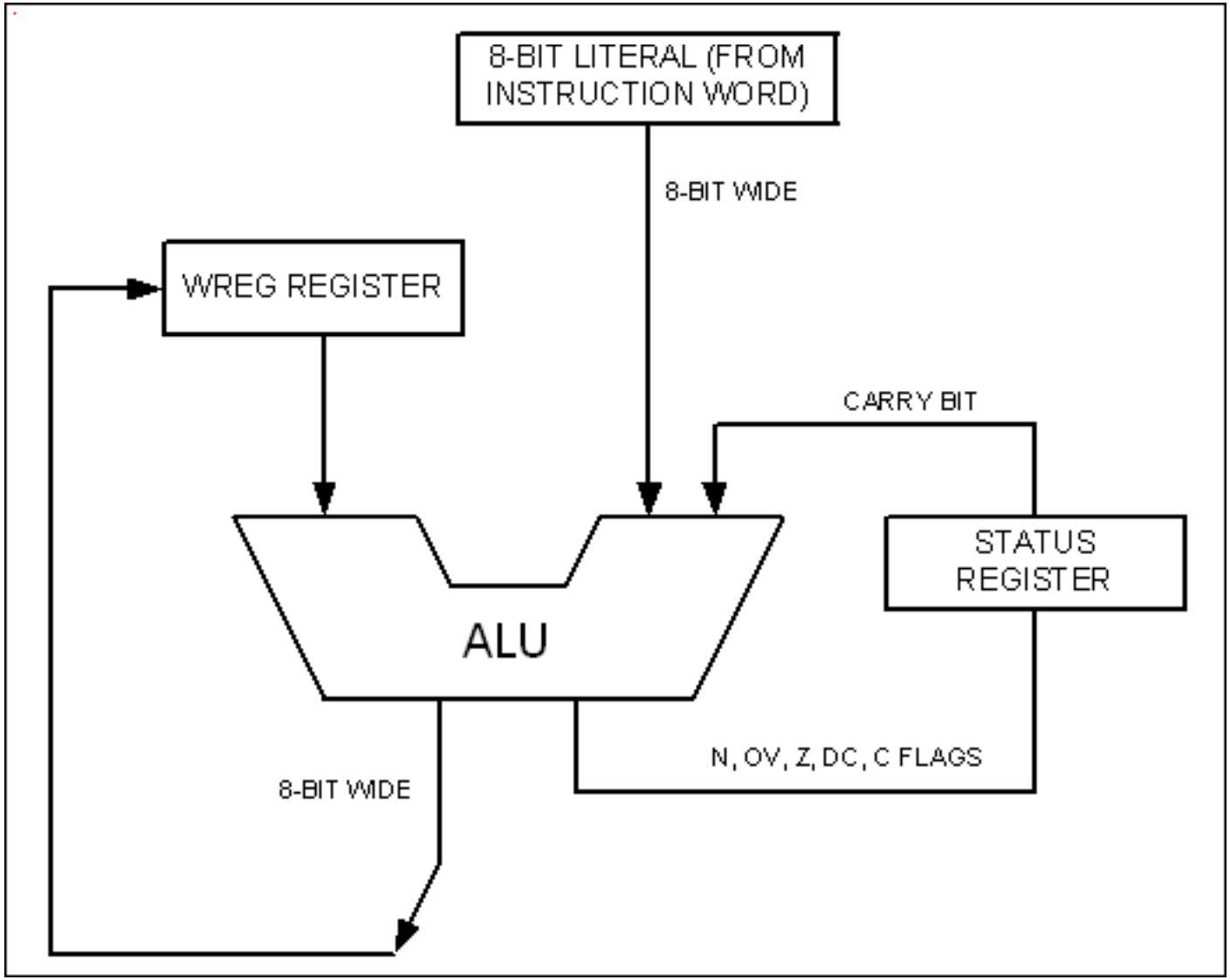
ADDLW

- ADDLW k; Add literal value k to WREG (k +WREG)

- Example:
  - MOVLW 12H
  - ADDLW 16H
    - ADDKW 11H
    - ADDLW 43H

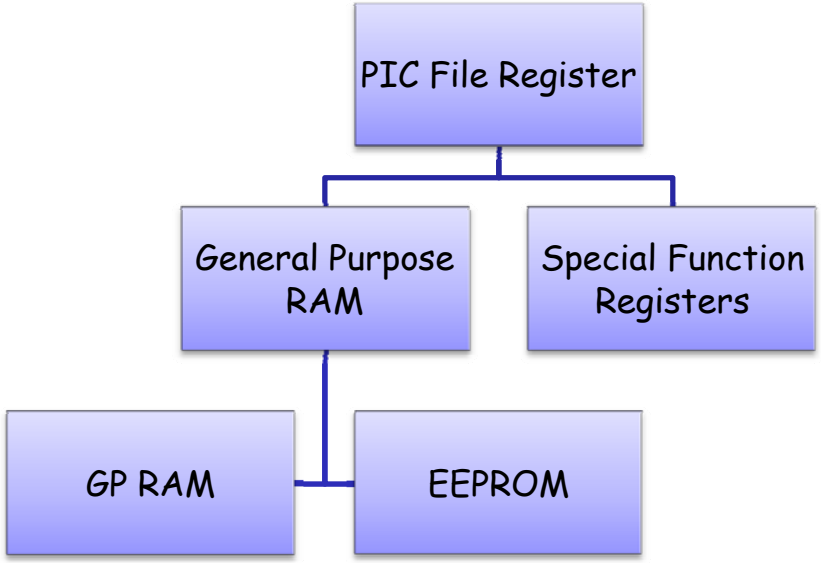
	0	0	0	1	0	0	1	0
;	0	0	1	0	1	0	0	0
;	0	0	1	1	1	0	0	1
;	0	1	1	1	1	1	0	0

Figure 2-1. PIC WREG and ALU Using Literal Value

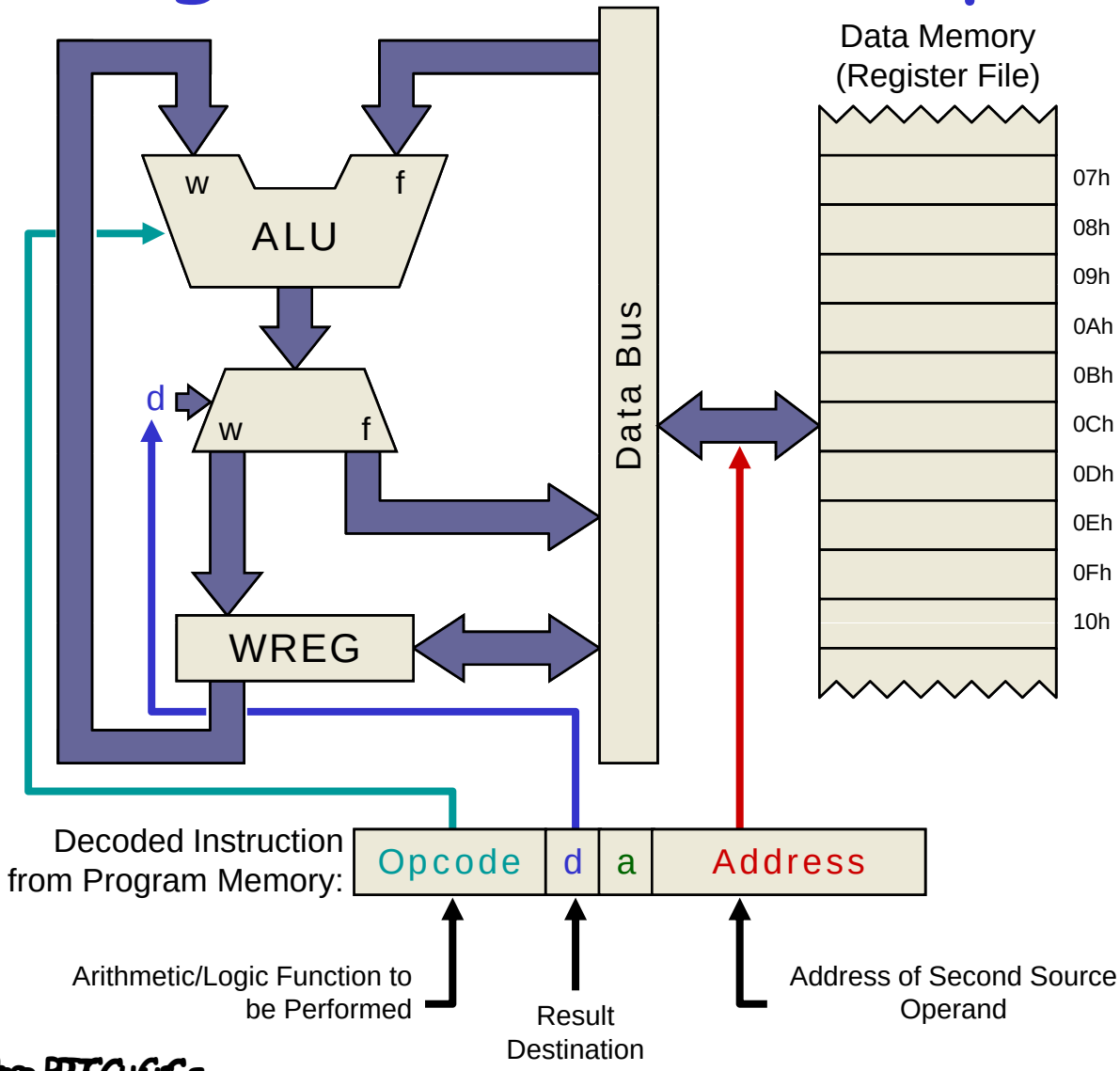


The PIC File Register

- It is the data memory.
  - Read/Write → Static RAM
  - Used for data storage, scratch pad and registers for internal use and function
  - 8-bit width

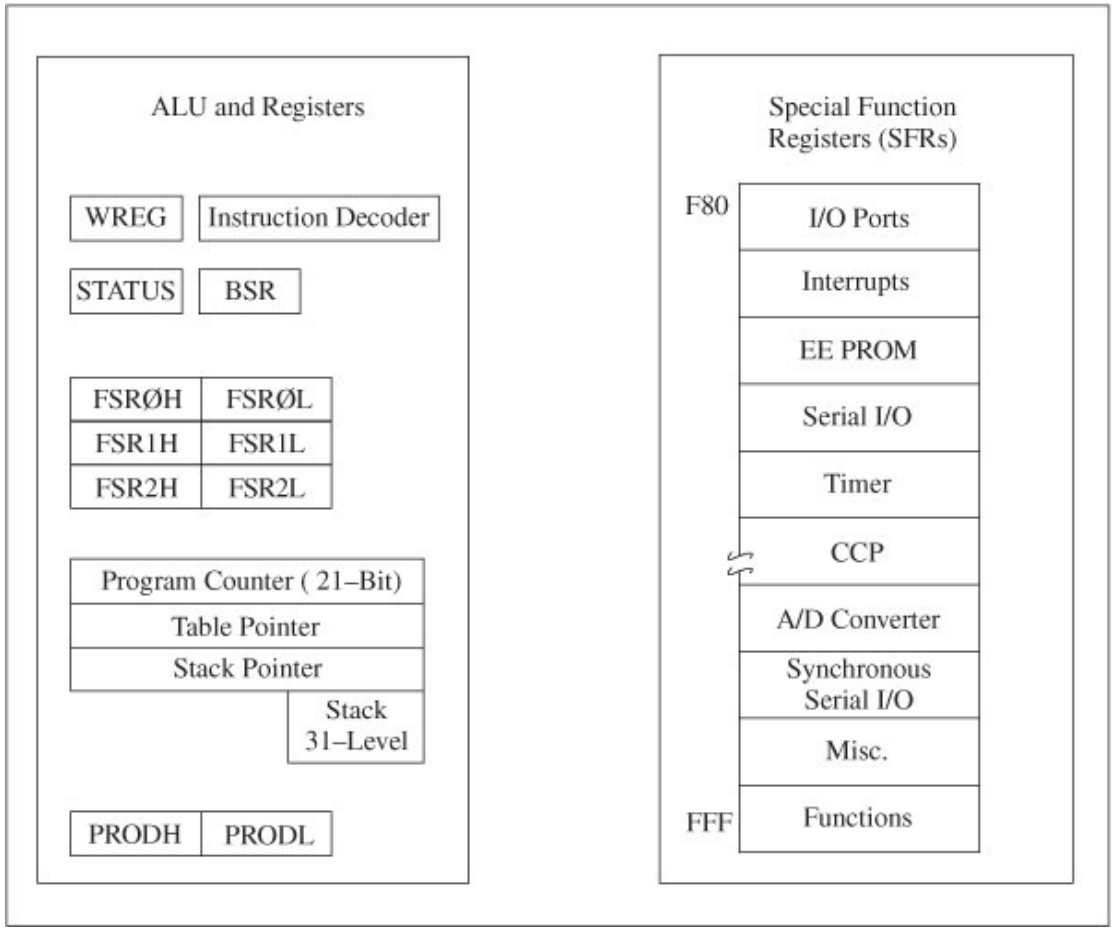


Register File Concept



- Register File Concept: 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
- Orthogonal Instruction Set: ALL instructions can operate on ANY data memory location

PIC18F Programming Model



Special Function Registers

- dedicated to specific functions such as ALU status, timers, serial communication, I/O ports, ADC,…
- The function of each SFR is fixed by the CPU designer at the time of design
  - it is used for control of the microcontroller or peripheral
- 8-bit registers
- Their numbers varies from one chip to another.

General Purpose RAM

- Group of RAM locations
- 8-bit registers
- Larger than SFR
  - Difficult to manage them by using Assembly language
  - Easier to handle them by C Compiler.

- The microchip website provides the data RAM size, which is the same as GPR size.

File Register Size

	File Register (Bytes)	=	SFR (Bytes)	+	GPR (Bytes)
PIC12F508	32		7		25
PIC16F84	80		12		68
PIC18F1220	512		256		256
PIC18F452	1792		256		1536
PIC18F2220	768		256		512
PIC18F458	1792		256		1536
PIC18F8722	4096		158		3938
PIC18F4550	2048		160		1888

Figure 2-2. File Registers of PIC12, PIC16, and PIC18

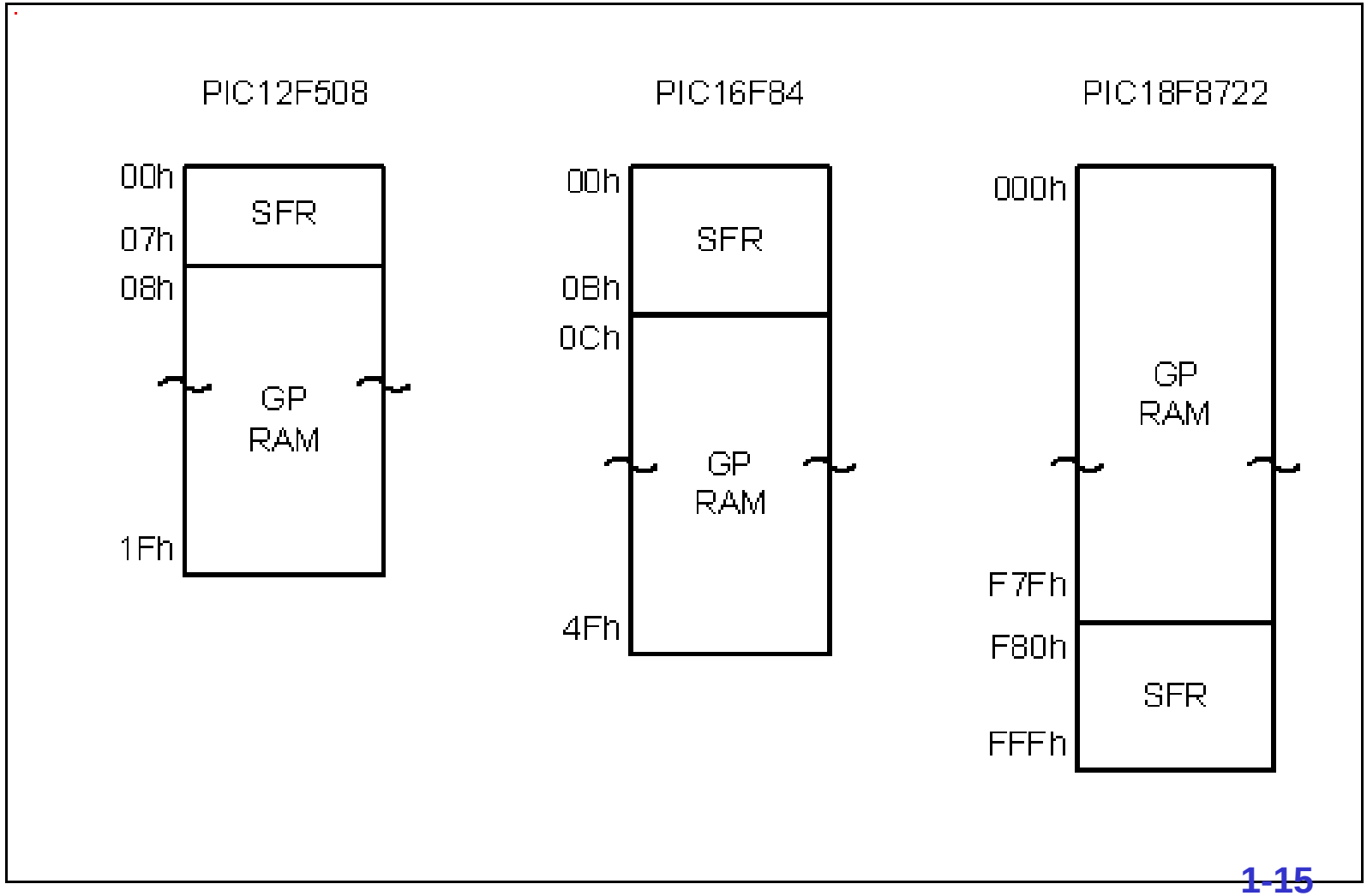
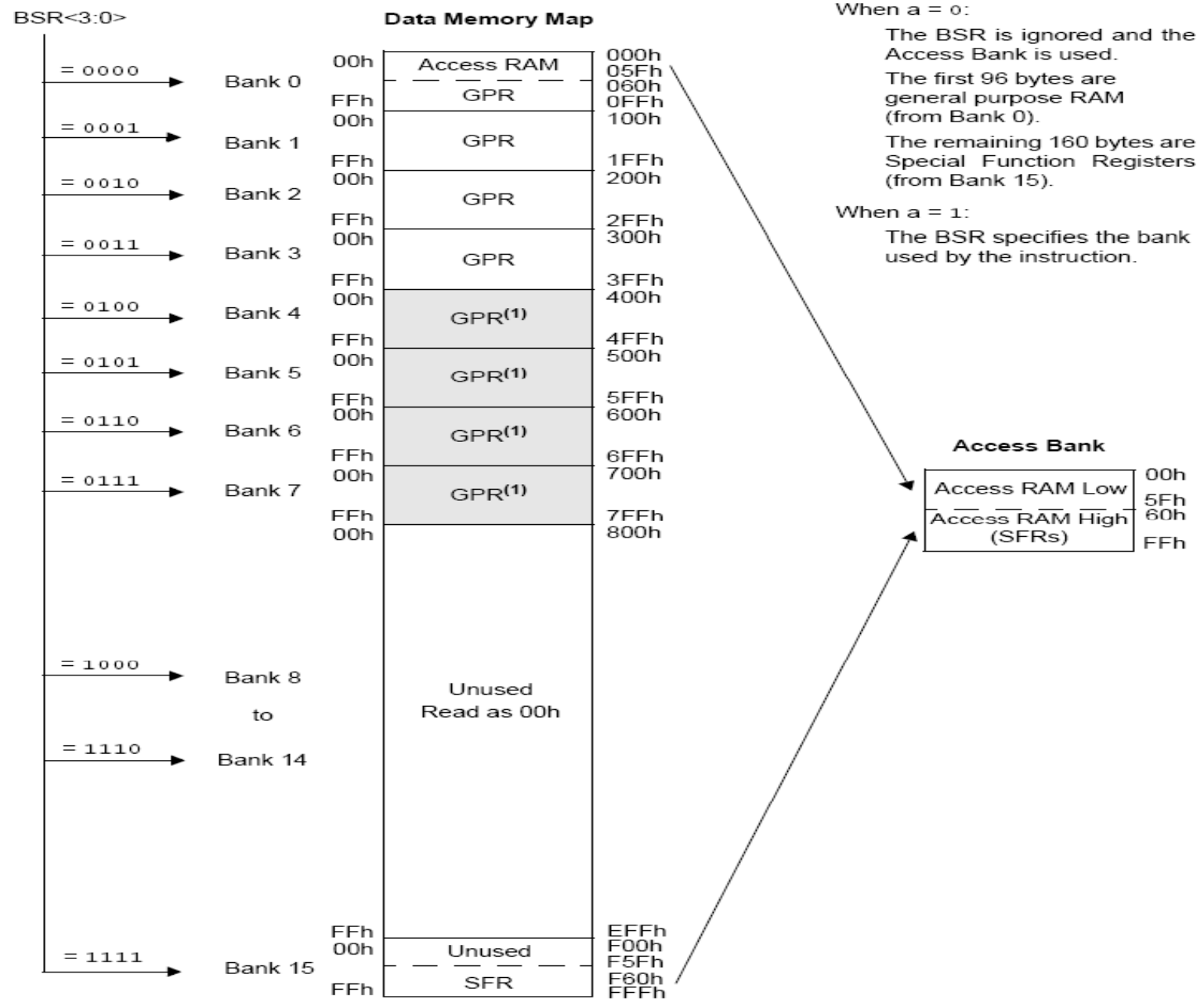


FIGURE 5-5: DATA MEMORY MAP FOR PIC18F2455/2550/4455/4550 DEVICES



Note 1: These banks also serve as RAM buffer for USB operation. See **Section 5.3.1 "USB RAM"** for more information.

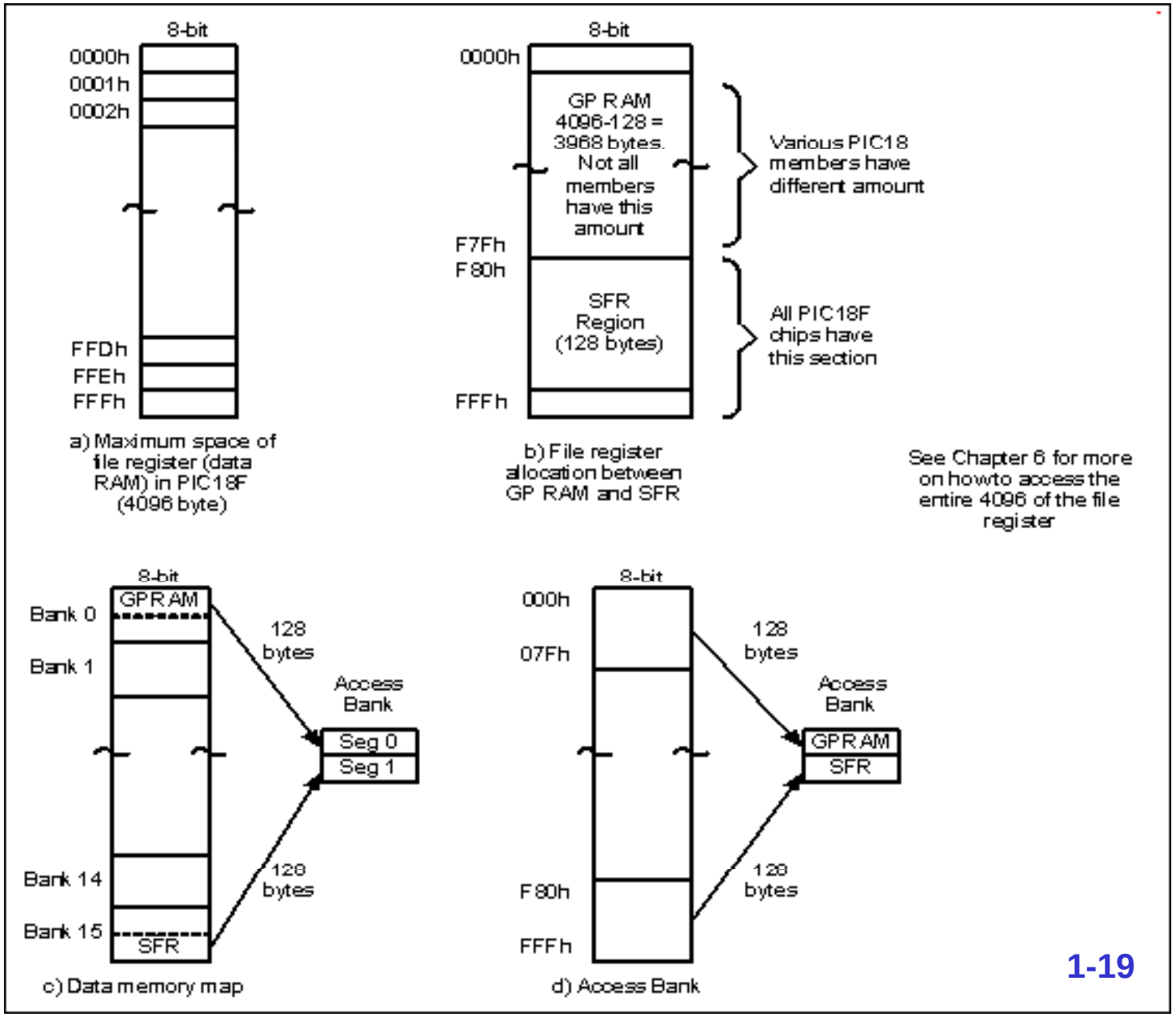
GPRAM VS. EEPROM

- An add-on memory
- Can be zero size

File Register and access bank in the PIC18

- The PIC18 Family can have a max. of 4096 Bytes.
- The File Register
  - has addresses of 000- FFFH
  - divided into 256-byte banks
  - Max. 16 banks (How?)
- At least there is one bank
  - Known as default access bank.
- Bank switching is a method used to access all the banks

Figure 2-3. File Register for PIC18 Family



Access bank in the PIC18

- It is 256-Byte bank.
- Divided into equal two discontinuous sections (each 128 B).
  - GP RAM, from 0 to 7FH
  - SFR, from F80H to FFFH

Figure 2-4. SFRs of the PIC18 Family.

F80h	PORTA	FA0h	PIE2	FC0h	----	FE0h	BSR
F81h	PORTB	FA1h	PIR2	FC1h	ADCON1	FE1h	FSR1L
F82h	PORTC	FA2h	IPR2	FC2h	ADCON0	FE2h	FSR1H
F83h	PORTD	FA3h	----	FC3h	ADRESL	FE3h	PLUSW1 *
F84h	PORTE	FA4h	----	FC4h	ADRESH	FE4h	PREINC1 *
F85h	----	FA5h	----	FC5h	SSPCON2	FE5h	POSTDEC1 *
F86h	----	FA6h	----	FC6h	SSPCON1	FE6h	POSTINC1 *
F87h	----	FA7h	----	FC7h	SSPSTAT	FE7h	INDF1 *
F88h	----	FA8h	----	FC8h	SSPADD	FE8h	WREG
F89h	LATA	FA9h	----	FC9h	SSPBUF	FE9h	FSR0L
F8Ah	LATB	FAAh	----	FCAh	T2CON	FEAh	FSR0H
F8Bh	LATC	FABh	RCSTA	FCBh	PR2	FEBh	PLUSW0 *
F8Ch	LATD	FACh	TXSTA	FCCh	TMR2	FECh	PREINC0 *
F8Dh	LATE	FADh	TXREG	FCDh	T1CON	FEDh	POSTDEC0 *
F8Eh	----	FAEh	RCREG	FCEh	TMR1L	FEEh	POSTINC0 *
F8Fh	----	FAFh	SPBRG	FCFh	TMR1H	FEFh	INDF0 *
F90h	----	FB0h	----	FD0h	RCON	FF0h	INTCON3
F91h	----	FB1h	T3CON	FD1h	WDTCON	FF1h	INTCON2
F92h	TRISA	FB2h	TMR3L	FD2h	LVDCON	FF2h	INTCON
F93h	TRISB	FB3h	TMR3H	FD3h	OSCCON	FF3h	PRODL
F94h	TRISC	FB4h	----	FD4h	----	FF4h	PRODH
F95h	TRISD	FB5h	----	FD5h	T0CON	FF5h	TABLAT
F96h	TRISE	FB6h	----	FD6h	TMR0L	FF6h	TBLPTRL
F97h	----	FB7h	----	FD7h	TMR0H	FF7h	TBLPTRH
F98h	----	FB8h	----	FD8h	STATUS	FF8h	TBLPTRU
F99h	----	FB9h	----	FD9h	FSR2L	FF9h	PCL
F9Ah	----	FBAh	CCP2CON	FDAh	FSR2H	FFAh	PCLATH
F9Bh	----	FBBh	CCPR2L	FDBh	PLUSW2 *	FFBh	PCLATU
F9Ch	----	FBCh	CCPR2H	FDCh	PREINC2 *	FFCh	STKPTR
F9Dh	PIE1	FBDh	CCP1CON	FDDh	POSTDEC2 *	FFDh	TOSL
F9Eh	PIR1	FBEh	CCPR1L	FDEh	POSTINC2 *	FFEh	TOSH
F9Fh	IPR1	FBFh	CCPR1H	FDFh	INDF2 *	FFFh	TOSU

* - These are not physical registers.

Using instruction with the default access bank

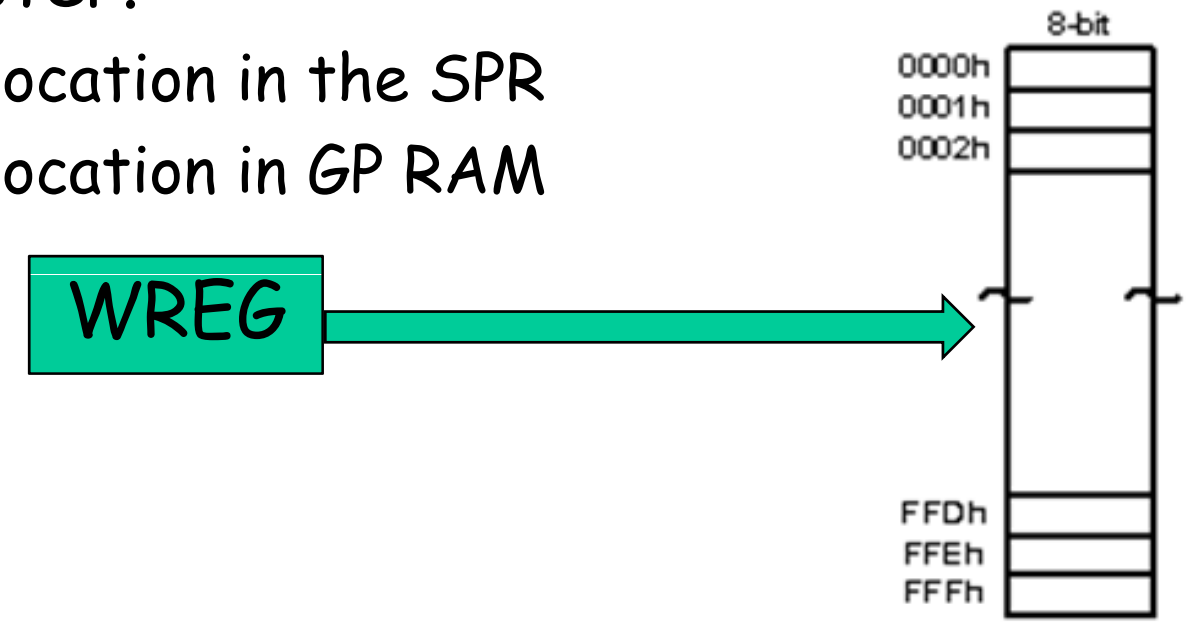
- We need instruction to access other locations in the file register for ALU and other operations.
  - MOVWF
  - COMF
  - DECF
  - MOVF
  - MOVFF

MOVWF instruction

- F indicates for a file register

  MOVWF Address

- It tells the CPU to copy the source register, WREG, to a destination in the file register.
  - A location in the SPR
  - A location in GP RAM



Example 2-1

- MOVLW 99H
- MOVWF 12H
- MOVLW 85H
- MOVWF 13H
- MOVLW 3FH
- MOVWF 14H
- MOVLW 63H
- MOVWF 15H
- MOVLW 12H
- MOVWF 16H

WRFG

99

85

3F

63

12

Address	Data
012H	
013H	
014H	
015H	
016H	

Address	Data
012H	99
013H	85
014H	3F
015H	63
016H	12

Note

- We cannot move literal values directly into the general purpose RAM location in the PIC18. They must be moved there via WREG.

ADDWF

- Adds together the content of WREG and a file register location

  ADDWF File Reg. Address, D

- The result will be placed in either the WREG or in the file register location
  - D indicates the destination bit
- If D=0 or (D=w)
  - The result will be placed in the WREG
- If D=1 or (D=f)
  - The result will be placed in the file register

Example 2-2

- State the content of file register location and WREG after the following program
- MOVLW 0
- MOVWF 12H
- MOVLW 22H
- ADDWF 12H, F
- ADDWF 12H, F
- ADDWF 12H, F
- ADDWF 12H, F

0

22

Address	Data
012H	0
013H	
014H	
015H	
016H	

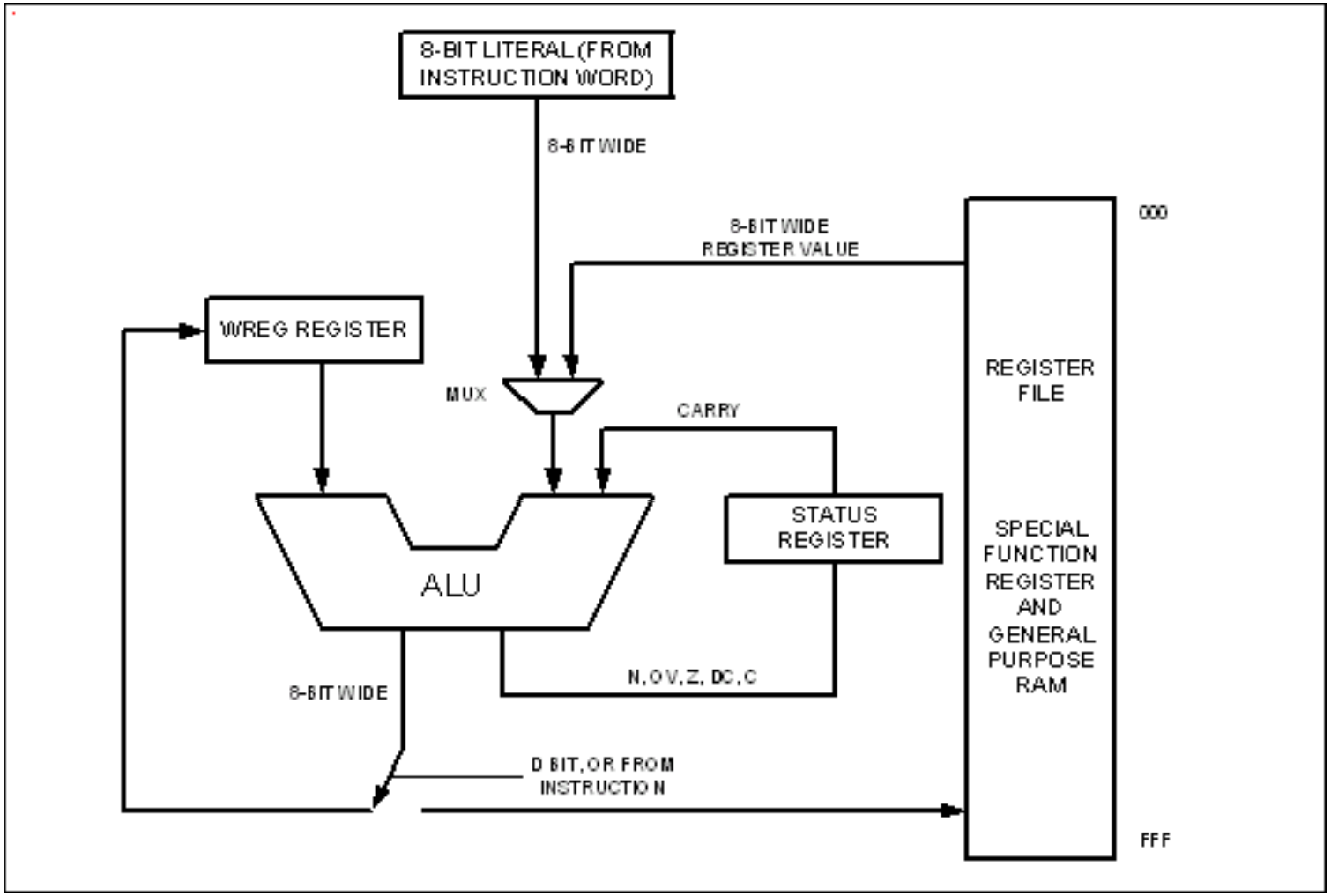
Example 2-3

- State the content of file register location and WREG after the following program
- MOVLW 0
- MOVWF 12H
- MOVLW 22H
- ADDWF 12H, F
- ADDWF 12H, W
- ADDWF 12H, W
- ADDWF 12H, W

0
22
44
66
88

Address	Data
012H	22
013H	
014H	
015H	
016H	

Figure 2-5. WREG, fileReg, and ALU in PIC18



COMF instruction

COMF File Reg. Address, D

- It tells the CPU to complement the content of fileReg and places the results in WREG or in fileReg.
-

Example 2-4

- Write a simple program to toggle the SFR of Port B continuously forever.

Solution

55

	MOVLW	55H
	MOVWF	PORTB
B1	COMF	PORTB, F
	GOTO	B1

Address	Data
F81H	55H
F82H	
F83H	

Address	Data
F81H	AAH
F82H	
F83H	

DECF instruction

DECF File Reg. Address, D

- It tells the CPU to decrement the content of fileReg and places the results in WREG or in fileReg.
- Example:
  - MOVLW 3
  - MOVWF 20H
  - DECF 20H, F
  - DECF 20H, F
  - DECF 20H, F

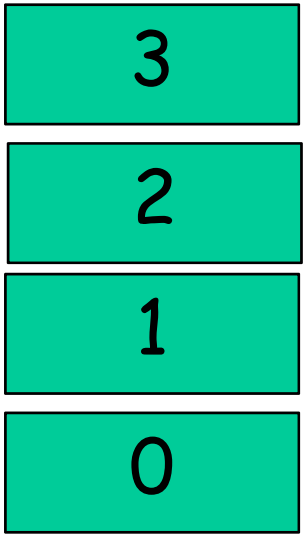
3

Address	Data
012H	~~3~~ 2
013H	
014H	
015H	
016H	

DECF instruction

DECF File Reg. Address, D

- It tells the CPU to decrement the content of fileReg and places the results in WREG or in fileReg.
- Example:
  - MOVLW 3
  - MOVWF 20H
  - DECF 20H, w
  - DECF 20H, w
  - DECF 20H, w



Address	Data
012H	3
013H	
014H	
015H	
016H	

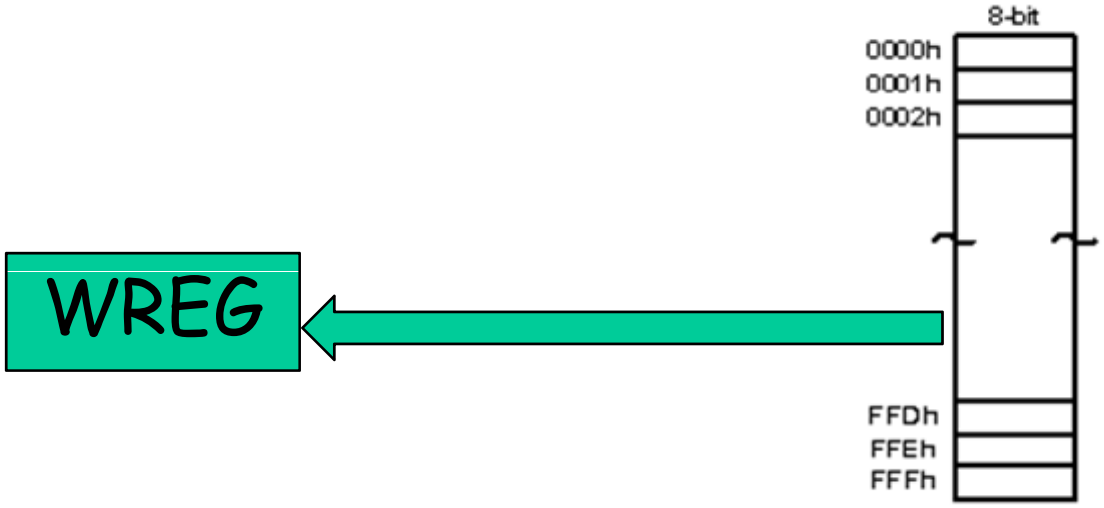
MOVF instruction

MOVF File Reg. Address, D

- It is intended to perform MOVFW
  - MOVFW isn't existed
- If D=0
  - Copies the content of fileReg (from I/O pin) to WREG
- If D=1
  - The content of the fileReg is copied to itself. (why?)

MOVF instruction

- MOVF File Reg. Address, 0



Example 2-5

- Write a simple program to get data from the SFRs of Port B and send it the SFRs of PORT C continuously.

Solution

XX

Address	Data
F81H	XX
F82H	XX
F83H	

```
AGAIN  MOVF  PORTB, W
       MOVWF PORTC
       GOTO  AGAIN
```

Example 2-6

- Write a simple program to get data from the SFRs of Port B Add the value 5 to it and send it the SFRs of PORT C
- Solution
- MOVF PORTB,W
- ADDLW 05H
- MOVWF PORTC

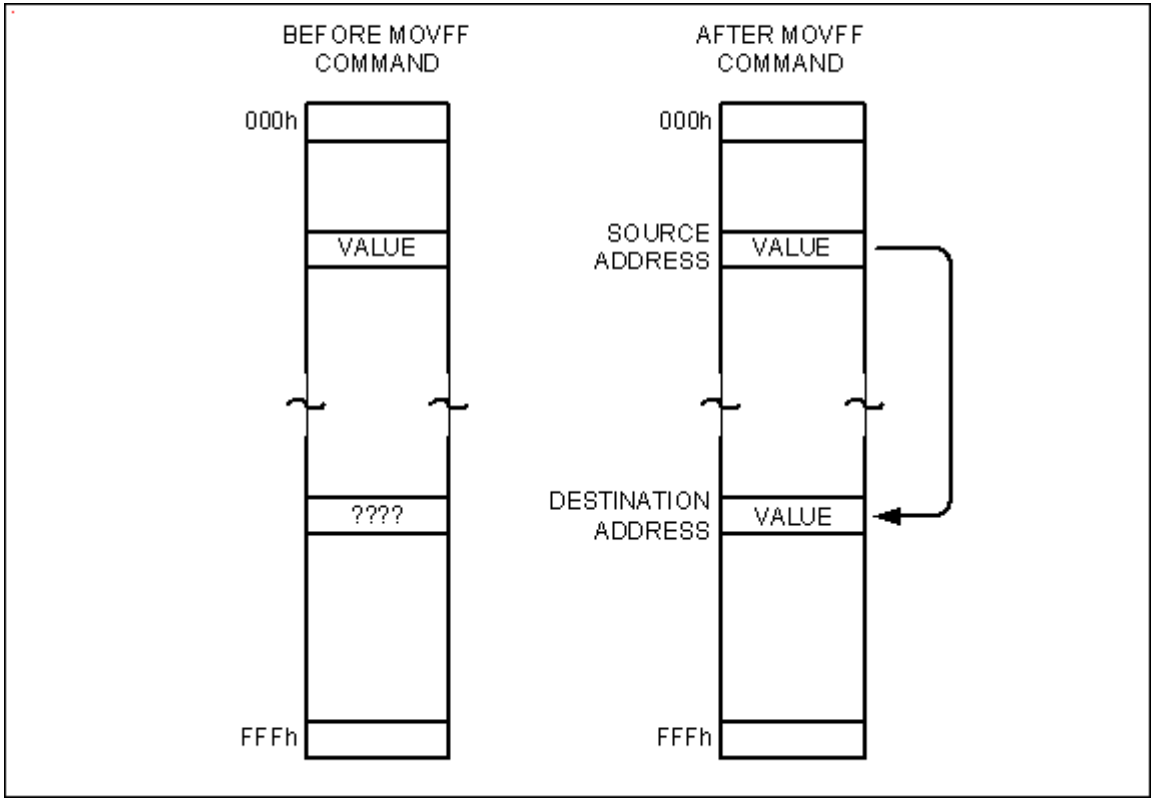
5A

Address	Data
F81H	55H
F82H	5AH
F83H	

MOVFF instruction

- It copies data from one location in FileReg to another location in FileReg.

MOVFF Source FileReg, destination FileReg



Example 2-7

- Write a simple program to get data from the SFRs of Port B and send it the SFRs of PORT C continuously.

Solution

XX

Address	Data
F81H	XX
F82H	XX
F83H	

- AGAIN MOVFF PORTB, PORTC
- GOTO AGAIN

PIC Status Register

- To indicate arithmetic conditions
- It is a 8-bit register
  - Five bits are used
- D0: C Carry Flag
- D1: DC Digital Carry Flag
- D2: Z Zero Flag
- D3: OV Overflow Flag
- D4: N Negative Flag

Figure 2-7. Bits of Status Register

D7							D0
X	X	X	N	OV	Z	DC	C

C – Carry flag

DC – Digital Carry flag

Z – Zero flag

OV – Overflow flag

N – Negative flag

X – D5, D6, and D7 are not implemented,
and reserved for future use.

Example 2-8

- Show the status of the C, DC, Z flags after the following addition instruction
- MOVLW 38H
- ADDLW 2FH
- Solution
- 38H + 2FH = 67H → WREG=67H

  C=0

  DC=1

  Z=0

Example 2-9

- Show the status of the C, DC, Z flags after the following addition instruction
- MOVLW 9CH
- ADDLW 64H
- Solution
- 9CH + 64H = 100H → WREG= 00H

  C=1

  DC=1

  Z=1

Instruction That Affect Flag Bits

Mnemonic		Beschreibung	Status
Transferoperationen			
MOVF	f, d, a	Move f (d=0: f ⇒ W; d=1: f ⇒ f)	Z, N
MOVFF	f_s, f_D	Move f_s to f_D	
MOVWF	f, a	Move W to f (W ⇒ f)	
MOVLW	k	Move literal to W (k ⇒ W)	
MOVLB	k	Move literal to BSR<3:0>	
LFSR	n, k	Move literal to FSRn	
SWAPF	f, d, a	Swap nibbles in f	
PUSH		Increment STKPTR<4:0>, TOS = PC+2	
POP		Decrement STKPTR<4:0>	

Instruction That Affect Flag Bits

Logische Operationen			
ANDWF	f, d, a	AND W with f	Z, N
ANDLW	k	AND literal with W	Z, N
IORWF	f, d, a	Incl. OR W with f	Z, N
IORLW	k	Incl. OR literal with W	Z, N
XORWF	f, d, a	Excl. OR W with f	Z, N
XORLW	k	Excl. OR literal with W	Z, N
CLRF	f, a	Clear f	Z
SETF	f, a	Set f (f = FFH)	
COMF	f, d, a	Compement f	Z, N

Instruction That Affect Flag Bits

Arithmetische Operationen			
ADDWF	f, d, a	Add W and f	alle
ADDWFC	f, d, a	Add W, f and C	alle
ADDLW	k	Add literal and W	alle
SUBFWB	f, d, a	Subtract f from W (W - f) with Burrow	alle
SUBWF	f, d, a	Subtract W from f (f – W)	alle
SUBWFB	f, d, a	Subtract W from f (f – W) with Burrow	alle
SUBLW	k	Subtract W from literal (k – W)	alle
DECF	f, d, a	Decrement f (f – 1)	alle
INCF	f, d, a	Increment f (f + 1)	alle
MULWF	f, a	Multiply WREG with f	
MULLW	f, a	Multiply literal with WREG	
DAW		Decimal Adjust W (BCD-Operation)	C
Schiebeoperationen			
RLCF	f, d, a	Rotate Left f through Carry	C, Z, N
RLNCF	f, d, a	Rotate Left f without Carry	Z, N
RRCF	f, d, a	Rotate Right f through Carry	C, Z, N
RRNCF	f, d, a	Rotate Right f without Carry	Z, N

Flag Bits and Decision Making

BC	k	Branch relative if Carry
BNC	k	Branch relative if Not Carry
BN	k	Branch relative if Negative
BNN	k	Branch relative if Not Negative
BOV	k	Branch relative if Overflow
BNOV	k	Branch relative if Not Overflow
BZ	k	Branch relative if Zero
BNZ	k	Branch relative if Not Zero

PIC Data Format and Directives

- There is one data type
  - 8 bits
  - It is the job of the programmer to break down data larger 8 bits
- Data type can be positive or negative
- Data format are
  - Hex (default in PIC) 12 or 0x12 or H'12' or 12H
  - Binary B'00010010'
  - Decimal .12 or D'12'
  - ASCII A'c' or a'c'

Assembler Directives

- What is the difference between Instruction and Directives?
- EQU
  - Defines a constant or fixed address
- SET
  - Defines a constant or fixed address
  - Maybe reassigned later
- ORG (Origin)
- END
- LIST

Rules for labels in A.L.

- Unique name
- Alphabetic letters
  - Upper, lower, digits (0-9),special char. (? . @_ $)
- The first letter must be Alphabetic letters
- Not a reserved word

Introduction to PIC Assembly Language

- Difficult for us to deal with the machine code (0s and 1s)
- Assembly Language provide
  - Mnemonic: codes and abbreviations that are easy to remember
  - Faster programming and less prone error
  - LLL (why?)
  - Programmer must know all Reg. …etc.
- Assembler is used to translate the assembly code into machine code (object code)

Structure of Assembly Language

- Series of lines
  - Instruction
  - Directives
- Consists of four field

[label] mnemonic [operands] [;commands]

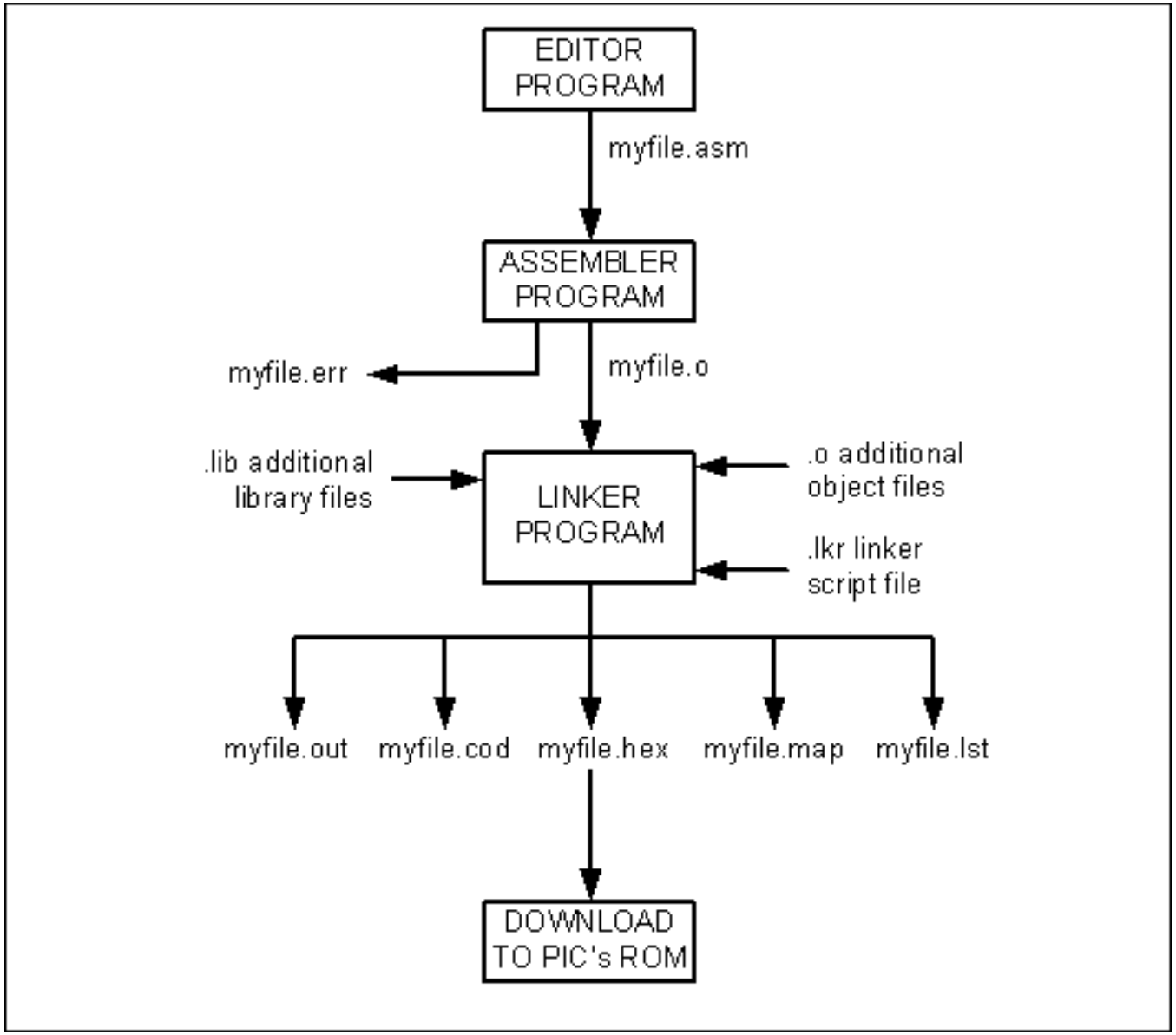
- Label: refer to line by code (certain length)
- Mnemonic and operands are task that should be executed.
  - Directive don't generate any machine code and used by assembler

Sample of Assembly Language Program

```
SUM   EQU 10H ;RAM loc 10H fro SUM
      ORG 0H; start at address 0
      MOVLW 25H ; WREG = 25
      ADDLW 0x34 ;add 34H to WREG=59H
      ADDLW 11H ;add 11H to WREG=6AH
      ADDLW D'18' ; W = W+12H=7CH
      ADDLW 1CH ; W = W+1CH=98H
      ADDLW b'00000110' ; W = W+6H=9EH
      MOVWF SUM ;save the result in SUM location
HERE  GOTO HERE ;stay here forever
      END       ; end of asm source file
```

Assembling and Linking A PIC Program

Figure 2-8. Steps to Create a Program



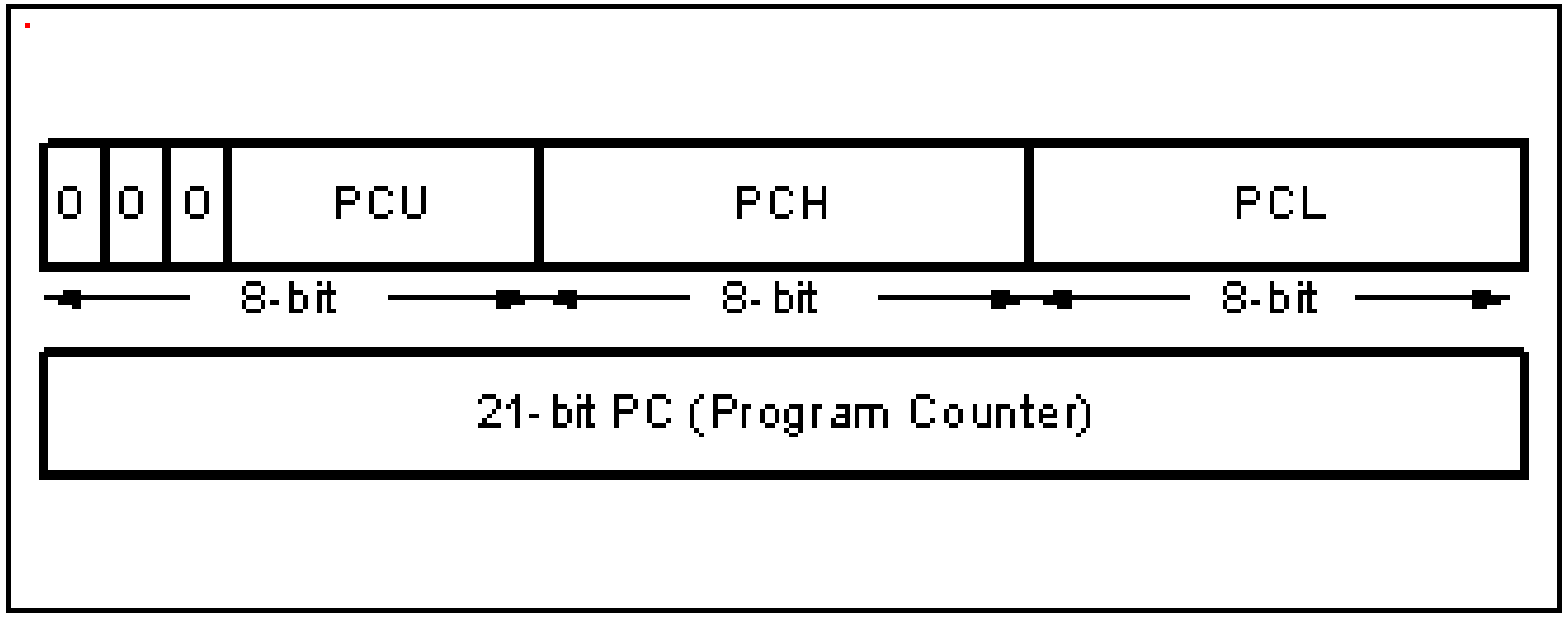
List File

```
MPASM  5.30.01                          S.ASM   4-17-2009  23:53:01        PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

  00000010            00001 SUM     EQU 10H ;RAM loc 10H fro SUM
000000                00002                 ORG 0H; start at address 0
000000 0E25           00003                 MOVLW 25H ; WREG = 25
000002 0F34           00004                 ADDLW 0x34 ;add 34H to WREG=59H
000004 0F11           00005                 ADDLW 11H ;add 11H to WREG=6AH
000006 0F12           00006                 ADDLW D'18' ; W = W+12H=7CH
000008 0F1C           00007                 ADDLW 1CH ; W = W+1CH=98H
00000A 0F06           00008                 ADDLW b'00000110' ; W = W+6H=9EH
00000C 6E10           00009                 MOVWF SUM ;save the result in SUM location
00000E EF07 F000      00010 HERE GOTO HERE ;stay here forever
                      00011                 END             ; end of asm source file

MPASM  5.30.01                          S.ASM   4-17-2009  23:53:01        PAGE  2


SYMBOL TABLE
  LABEL                             VALUE

HERE                              0000000E
SUM                               00000010
__18F4550                         00000001
__DEBUG                           1
```

The Program Counter and Program ROM Space in the PIC

- Program Counter (PC) is used by the CPU to point to the address of the next instruction to be executed
- The wider the program counter, more the memory locations can be accessed
  - PIC16 has 14 bits (8K)
  - PIC18 has 21 bits (2M)
  - 8051 has 16 bits (64K)

Figure 2-9. Program Counter in PIC18



PIC18 Microcontroller Family

Product	Program Memory Type	Program Memory Bytes	Data Memory RAM Bytes	Data Memory EEPROM Bytes	I/O Ports	ADC 10-bit	MSSP	USART	Other	CCP/ PWM	Timers 8/16-bit	Packages	Pins
PIC18F1220	FLASH	4K	256	256	16	7	—	1	6x PMM	1	1/3	DIP, SOIC, SSOP, QFN	18
PIC18F1320	FLASH	8K	256	256	16	7	—	1	6x PMM	1	1/3	DIP, SOIC, SSOP, QFN	18
PIC18F2220	FLASH	4K	512	256	23	10	I²C/SPI	1	6x PMM	2	1/3	DIP, SOIC	28
PIC18F2320	FLASH	8K	512	256	23	10	I²C/SPI	1	6x PMM	2	1/3	DIP, SOIC	28
PIC18C242	OTP	16K	512	—	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC	28
PIC18C252	OTP	32K	1536	—	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC	28
PIC18F242	FLASH	16K	512	256	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC, SSOP	28
PIC18F252	FLASH	32K	1536	256	23	5	I²C/SPI	1	—	2	1/3	DIP, SOIC, SSOP	28
PIC18F258	FLASH	32K	1536	256	22	5	I²C/SPI	1	CAN 2.0B	1	1/3	DIP, SOIC	28
PIC18F4220	FLASH	4K	512	256	34	13	I²C/SPI	1	6x PMM	2	1/3	DIP, TQFP, QFN	40/44
PIC18F4320	FLASH	8K	512	256	34	13	I²C/SPI	1	6x PMM	2	1/3	DIP, TQFP, QFN	40/44
PIC18C442	OTP	16K	512	—	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18C452	OTP	32K	1536	—	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F442	FLASH	16K	512	256	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F452	FLASH	32K	1536	256	34	8	I²C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F458	FLASH	32K	1536	256	33	5	I²C/SPI	1	CAN 2.0B	1	1/3	DIP, PLCC, TQFP	40/44
PIC18C601	—	ROMless	1536	—	31	8	I²C/SPI	1	—	2	1/3	PLCC, TQFP	64/68
PIC18C658	OTP	32K	1536	—	52	12	I²C/SPI	1	CAN 2.0B	2	1/3	PLCC, TQFP	64/68
PIC18F6520	FLASH	32K	2048	1024	52	12	I²C/SPI	2	—	5	2/3	TQFP	64
PIC18F6620	FLASH	64K	3840	1024	52	12	I²C/SPI	2	—	5	2/3	TQFP	64
PIC18F6720	FLASH	128K	3840	1024	52	12	I²C/SPI	2	—	5	2/3	TQFP	64
PIC18C801	—	ROMless	1536	—	42	12	I²C/SPI	1	—	2	1/3	PLCC, TQFP	80/84
PIC18C858	OTP	32K	1536	—	68	16	I²C/SPI	1	CAN 2.0B	2	1/3	PLCC, TQFP	80/84
PIC18F8520	FLASH	32K	2048	1024	68	16	I²C/SPI	2	EMA	5	2/3	TQFP	80
PIC18F8620	FLASH	64K	3840	1024	68	16	I²C/SPI	2	EMA	5	2/3	TQFP	80
PIC18F8720	FLASH	128K	3840	1024	68	16	I²C/SPI	2	EMA	5	2/3	TQFP	80

Abbreviation: ADC = Analog-to-Digital Converter; CCP = Capture/Compare/PWM; I²C = Inter-Integrated Circuit Bus; PMM = Power Managed Mode; PWM = Pulse Width Modulation; SPI = Serial Peripheral Interface; USART = Universal Synchronous/Asynchronous Receiver/Transmitter

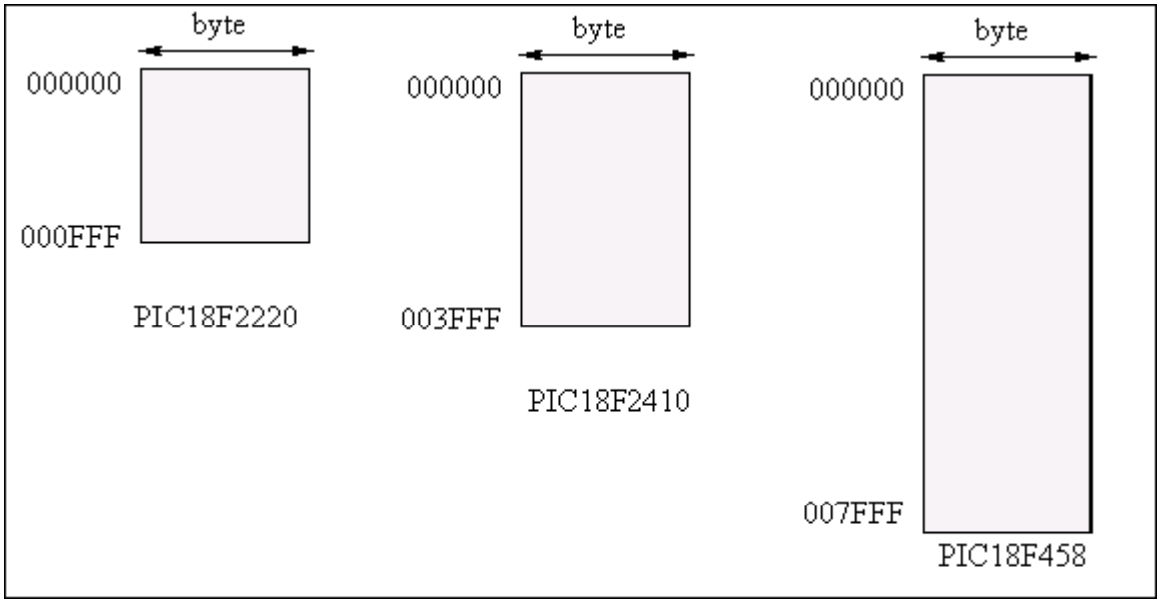
Example 2-11

- Find the ROM Memory Address of each of the following PIC chips:

a) PIC18F2220

b) PIC18F2410

c) PIC18F458



Powering UP

At what address does the CPU wake up when power applied?

- The uC wakes up at memory address 0000
- The PC has the value 0000
- ORG directive put the address of the first op code at the memory location 0000

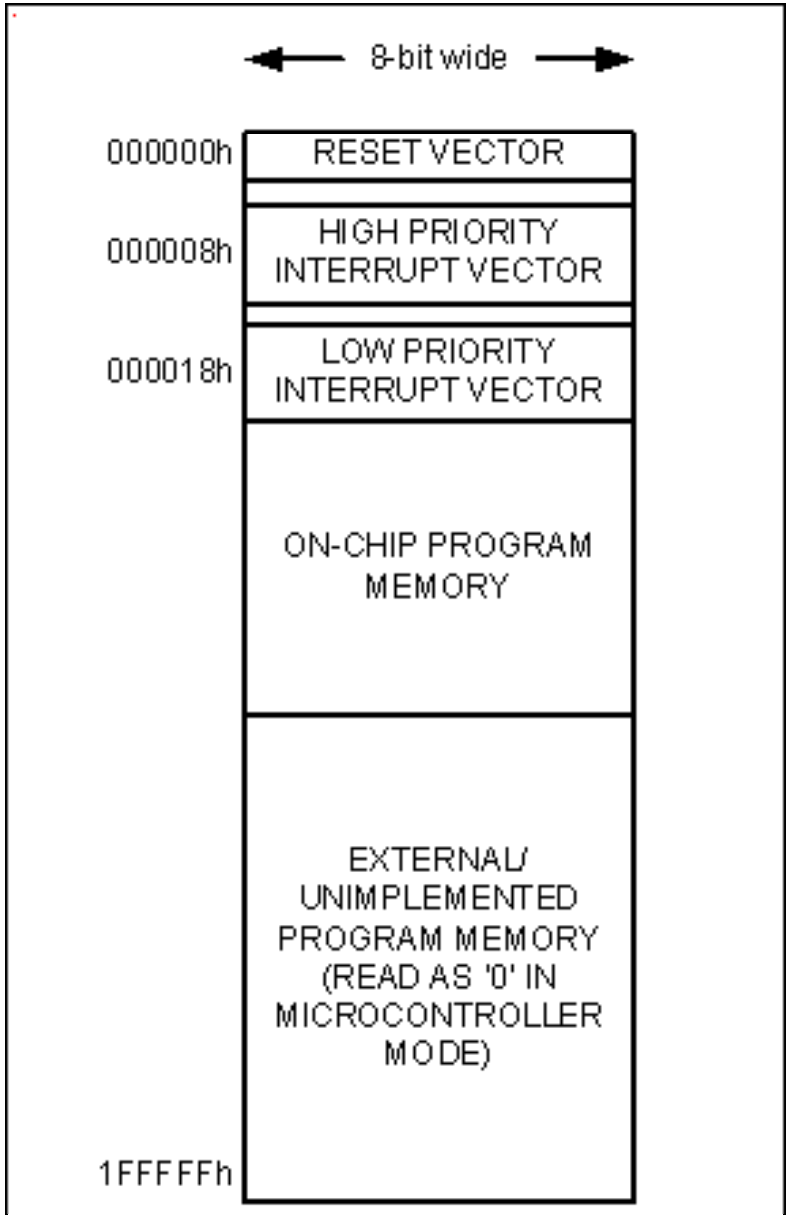


Figure 2-11. PIC18 Program ROM Space

Placing Code in program ROM

```
LOC   OBJECT CODE       LINE  SOURCE TEXT

  00000010              00001 SUM      EQU 10H
000000                  00002          ORG 0H
000000 0E25             00003          MOVLW 25H
000002 0F34             00004          ADDLW 0x34
000004 0F11             00005          ADDLW 11H
000006 0F12             00006          ADDLW D'18'
000008 0F1C             00007          ADDLW 1CH
00000A 0F06             00008          ADDLW b'00110'
00000C 6E10             00009          MOVWF SUM
00000E EF07 F000        00010 HERE     GOTO HERE
                        00011                  END
```

Program Memory

Program Memory

Line	Address	Opcode	Label	Disassembly
1	0000	0E25		MOVLW 0x25
2	0002	0F34		ADDLW 0x34
3	0004	0F11		ADDLW 0x11
4	0006	0F12		ADDLW 0x12
5	0008	0F1C		ADDLW 0x1c
6	000A	0F06		ADDLW 0x6
7	000C	6E10		MOVWF SUM, ACCESS
8	000E	EF07	HERE	GOTO HERE
9	0010	F000		NOP
10	0012	FFFF		NOP
11	0014	FFFF		NOP
12	0016	FFFF		NOP
13	0018	FFFF		NOP
14	001A	FFFF		NOP
15	001C	FFFF		NOP
16	001E	FFFF		NOP
17	0020	FFFF		NOP
18	0022	FFFF		NOP

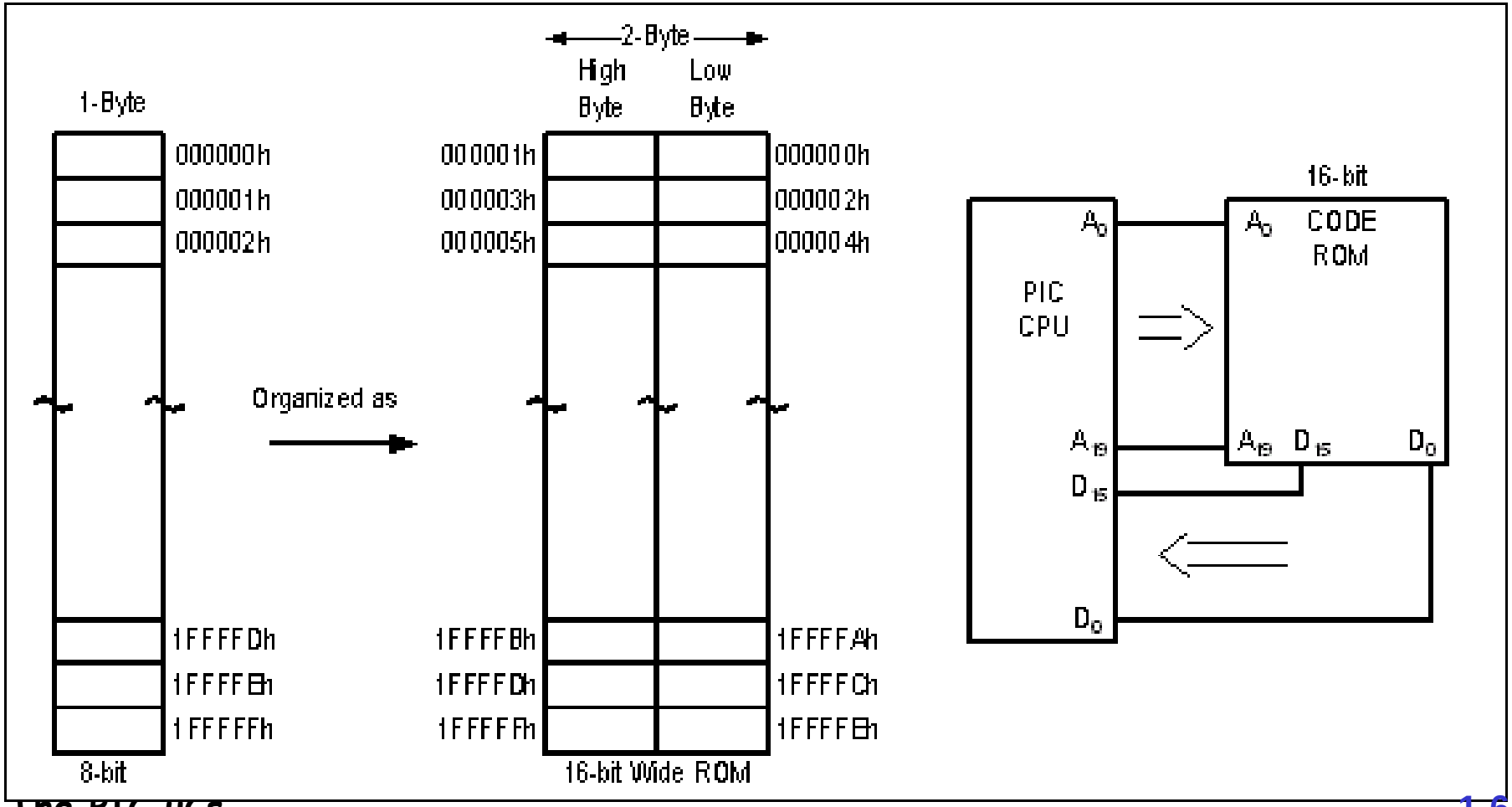
Opcode Hex | Machine | Symbolic

All instructions are 2Byte except the GOTO, which has 4-Byte

Program ROM Width for the PIC18

- Byte addressable: each location holds only one byte
  - CPU with 8-Bit will fetch one byte a time
  - Increasing the data bus will bring more information
- Solution: Data bus between CPU and ROM can be similar to traffic lanes on the highway
- The wide of Data path is 16 bit
  - Increase the processing power
  - Match the PIC18 instruction → single cycle

Figure 2-12. Program ROM Width for the PIC18



Little endian VS big endian war

- The low byte goes to the low memory location
- The high byte goes to the high memory location
- Intel uP and many uCs use little endian

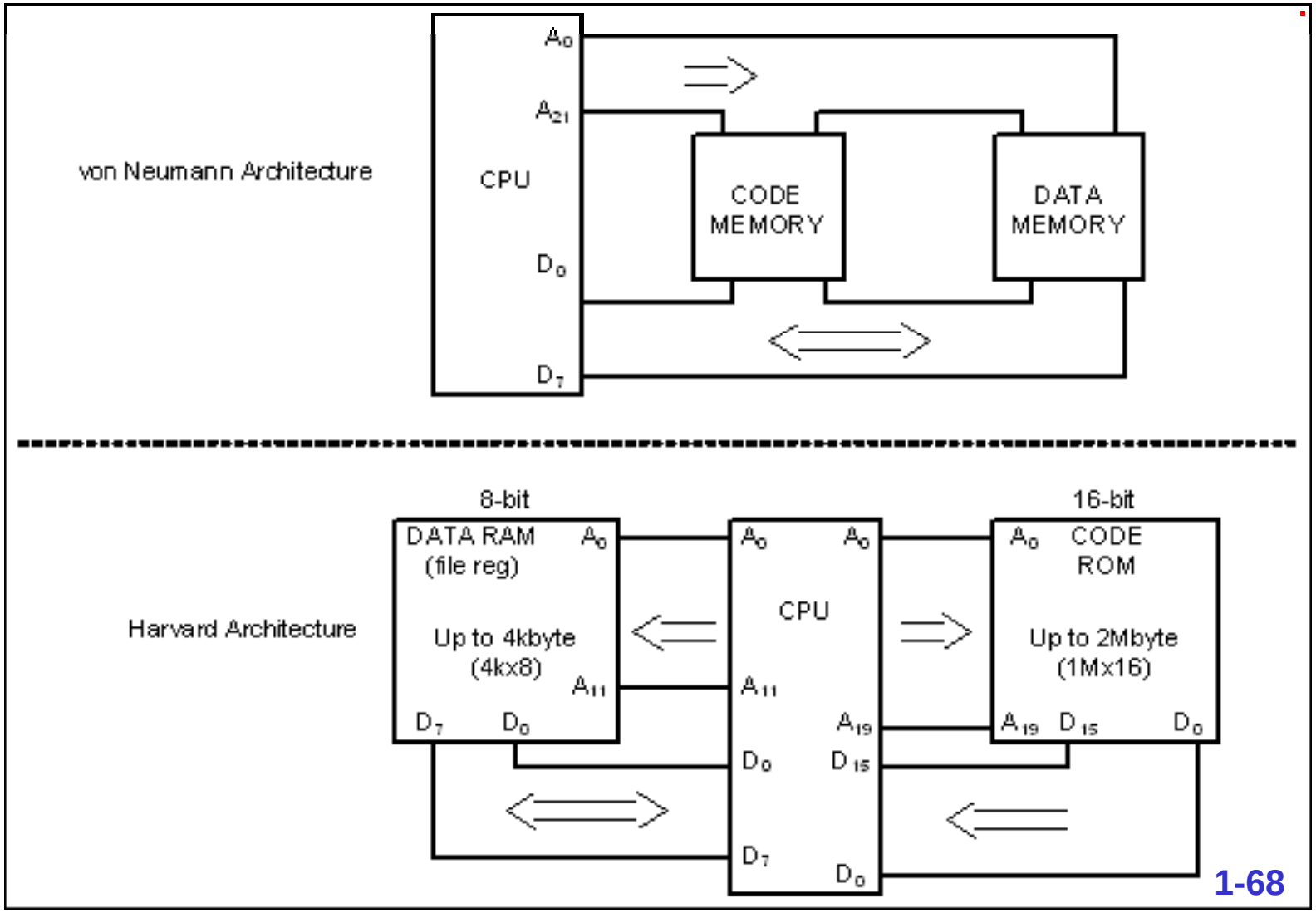
Figure 2-13. PIC18 Program ROM Contents for Program 2-1 List File

WORD ADDRESS	HIGH BYTE	LOW BYTE
000000h	0Eh	25h
000002h	0Fh	34h
000004h	0Fh	11h
000006h	0Fh	12h
000008h	0Fh	1Ch
00000Ah	0Fh	06h
00000Ch	6Eh	10h
00000Eh	EFh	07h
000010h	F0h	00h

Harvard Architecture in the PIC

- Von Neumann Architecture: uses the same bus for accessing both the code and data memory.
  - Slow down the processing speed
  - Get in each other's way
- Harvard Architecture: uses separate buses for accessing the code and data memory.
  - Inexpensive for a chip

Figure 2-14. von Neumann vs. Harvard Architecture



Instruction size of the PIC18

- PIC Instructions are 2-Byte or 4-Byte
- The first seven or eight bits represents the op-code
- Most of PIC18 instructions are 2-Byte
  - MOVLW 0000 1110 kkkk kkkk (0E XX)
  - ADDLW 0000 1111 kkkk kkkk (0F XX)
  - MOVWF 0110 111a ffff ffff (6E XX or 6F XX)
    - A specifies the default access bank if it is 0 and if a = 1 we have to use bank switching

Instruction size of the PIC18

- 4-Byte instructions include
  - MOVFF (move data within RAM, which is 4k)
    - 1100 ssss ssss ssss ($0 \leq fs \leq FFF$)
    - 1111 dddd dddd dddd ($0 \leq fd \leq FFF$)
  - GOTO (the code address bus width is 21, which is 2M)
    - 1110 1111 $k_7$kkk kkkk$_0$
    - 1111 k_{19}kkk kkkk kkkk$_8$

RISC Architecture in the PIC

- To increase the processing power of the CPU
  1. Increase the clock frequency of the chip
  2. Use Harvard architecture
  3. change the internal architecture of the CPU and use what is called RISC architecture

RISC Architecture in the PIC

- **RISC**
- Simple and Small instruction set
- Regular and fixed instruction format
- Simple address modes
- Pipelined instruction execution --> 95% executed in one cycle

- **CISC**
- Complex and large instruction set
- Irregular instruction format
- Complex address modes
- May also pipeline instruction execution

RISC Architecture in the PIC

- **RISC**
- Provide large number of CPU registers
- Separated data and program memory
- Most operations are register to register
- Take shorter time to design and debug

- **CISC**
- Provide smaller number of CPU registers
- Combined data and program memory
- Most operations can be register to memory
- Take longer time to design and debug

Viewing Register and memory with MPLAB Simulater

Figure 2-15. SFR Window in MPLAB Simulator

Special Function Registers

Address	SFR Name	Hex	Decimal	Binary	Char
0F80	PORTA	00	0	00000000	.
0F81	PORTB	00	0	00000000	.
0F82	PORTC	00	0	00000000	.
0F83	PORTD	00	0	00000000	.
0F84	PORTE	00	0	00000000	.
0F89	LATA	00	0	00000000	.
0F8A	LATB	00	0	00000000	.
0F8B	LATC	00	0	00000000	.
0F8C	LATD	00	0	00000000	.
0F8D	LATE	00	0	00000000	.
0F92	TRISA	00	0	00000000	.
0F93	TRISB	00	0	00000000	.
0F94	TRISC	00	0	00000000	.
0F95	TRISD	00	0	00000000	.
0F96	TRISE	00	0	00000000	.
0F9D	PIE1	00	0	00000000	.
0F9E	PIR1	00	0	00000000	.
0F9F	IPR1	00	0	00000000	.
0FA0	PIE2	00	0	00000000	.
0FA1	PIR2	00	0	00000000	.
0FA2	IPR2	00	0	00000000	.

Figure 2-16. File Register (Data RAM) Window in MPLAB Simulator

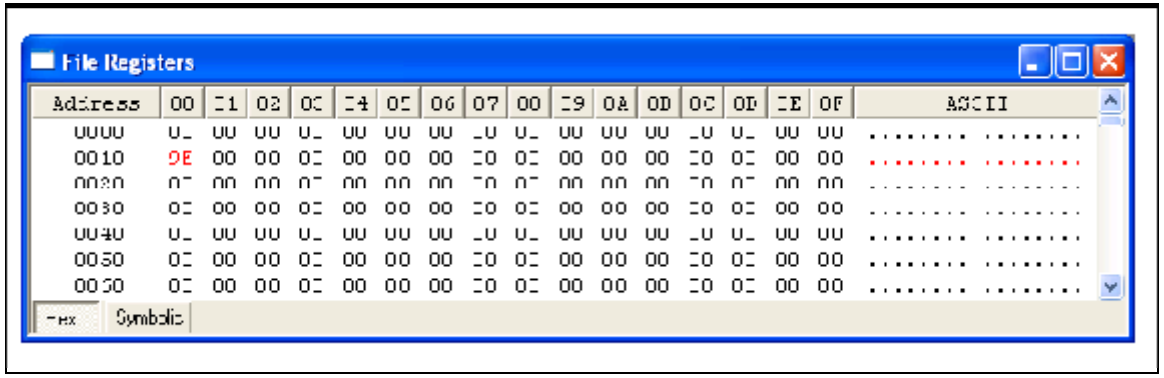
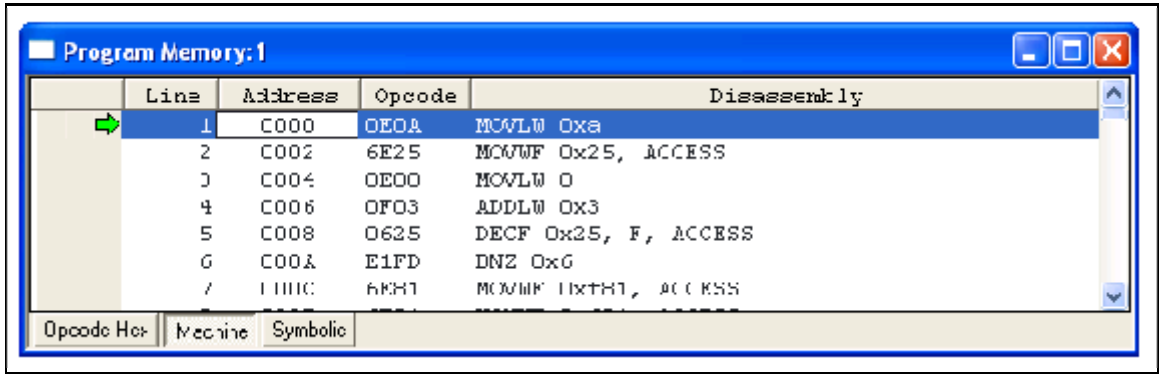


Figure 2-17. Program (Code) ROM Window in MPLAB Simulator



Chapter 2: Summary

- Sample PIC18 Instructions
  - Move, add, subtract

Next:

Branch, Call and Time Delay Loo

PIC Microcontroller and Embedded Systems

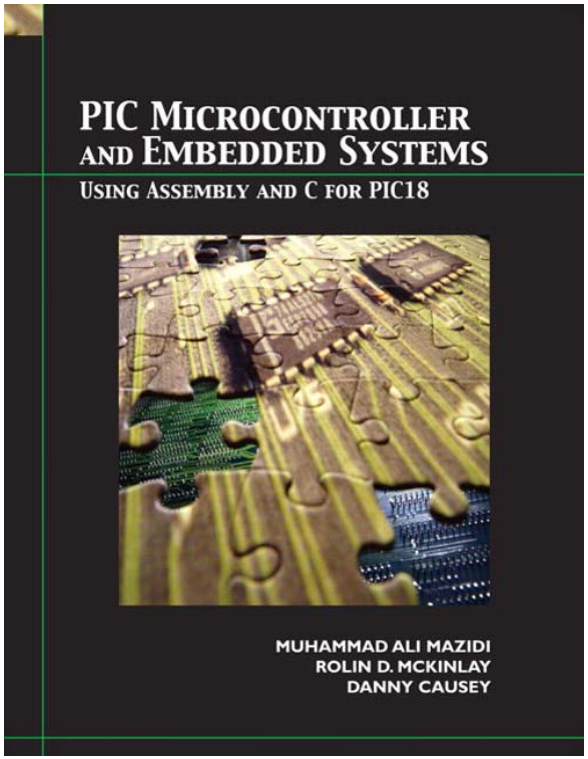
Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 3: Branch, Call and Time Delay Loop

- Branch instruction and looping
- Call instruction and stack
- PIC18 Time Delay and instruction pipeline



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

Objective

- Code PIC Assembly language instructions to create loops and conditional branch instructions
- Code Goto instructions for unconditional jump

Branch instructions and looping

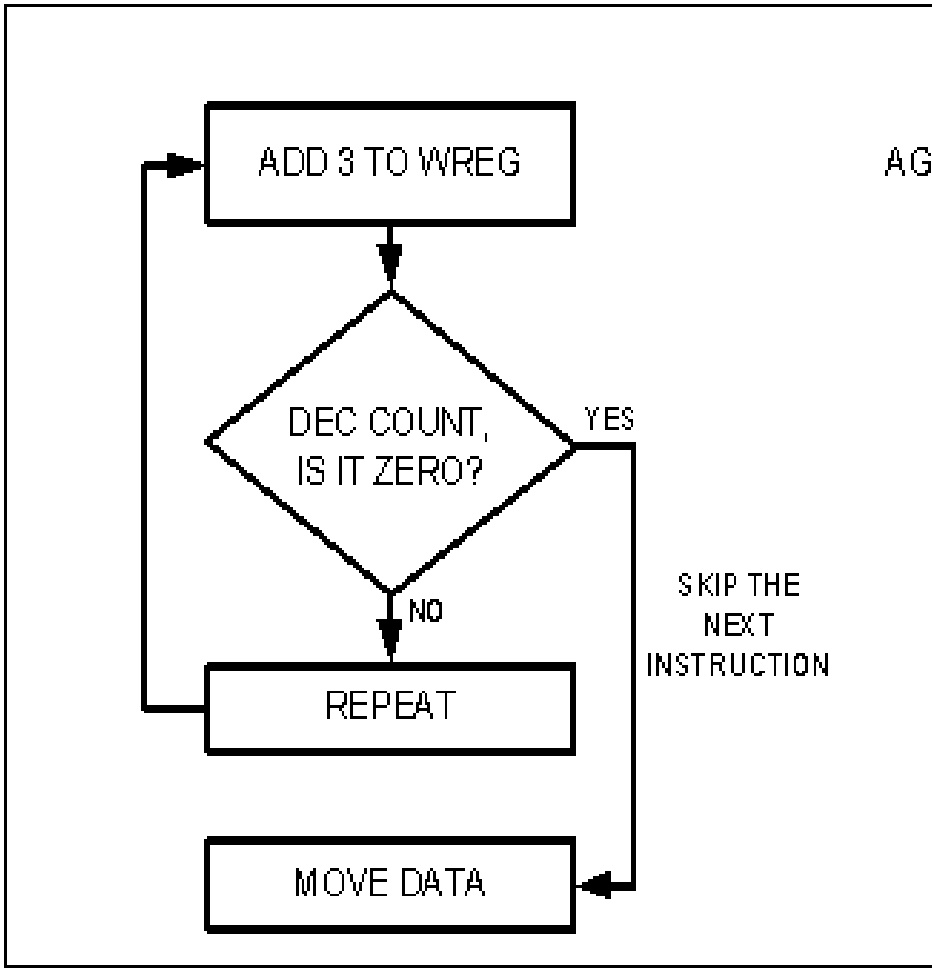
- Looping in PIC
- Loop inside loop
- Other conditional jumps
- All conditional branches are short jumps
- Calculating the short branch address
- Unconditional branch instruction

Looping in PIC

- Repeat a sequence of instructions or a certain number of times
- Two ways to do looping
  - Using DECFSZ instruction
  - Using BNZ\BZ instructions

DECFSZ instruction

- Decrement file register, skip the next instruction if the result is equal 0
- DECFSZ fileRef, d
- GOTO instruction follows DECFSZ



Example 3-1

- Write a program to
  a) Clear WREG
  b) Add 3 to WREG ten times and place the result in SFR PORTB

- Solution

```
COUNT    EQU 0x25
         MOVLW  d'10'
         MOVWF  COUNT
         MOVLW  0
AGAIN    ADDLW  3
         DECFSZ COUNT,F
         GOTO   AGAIN
         MOVWF  PORTB
```

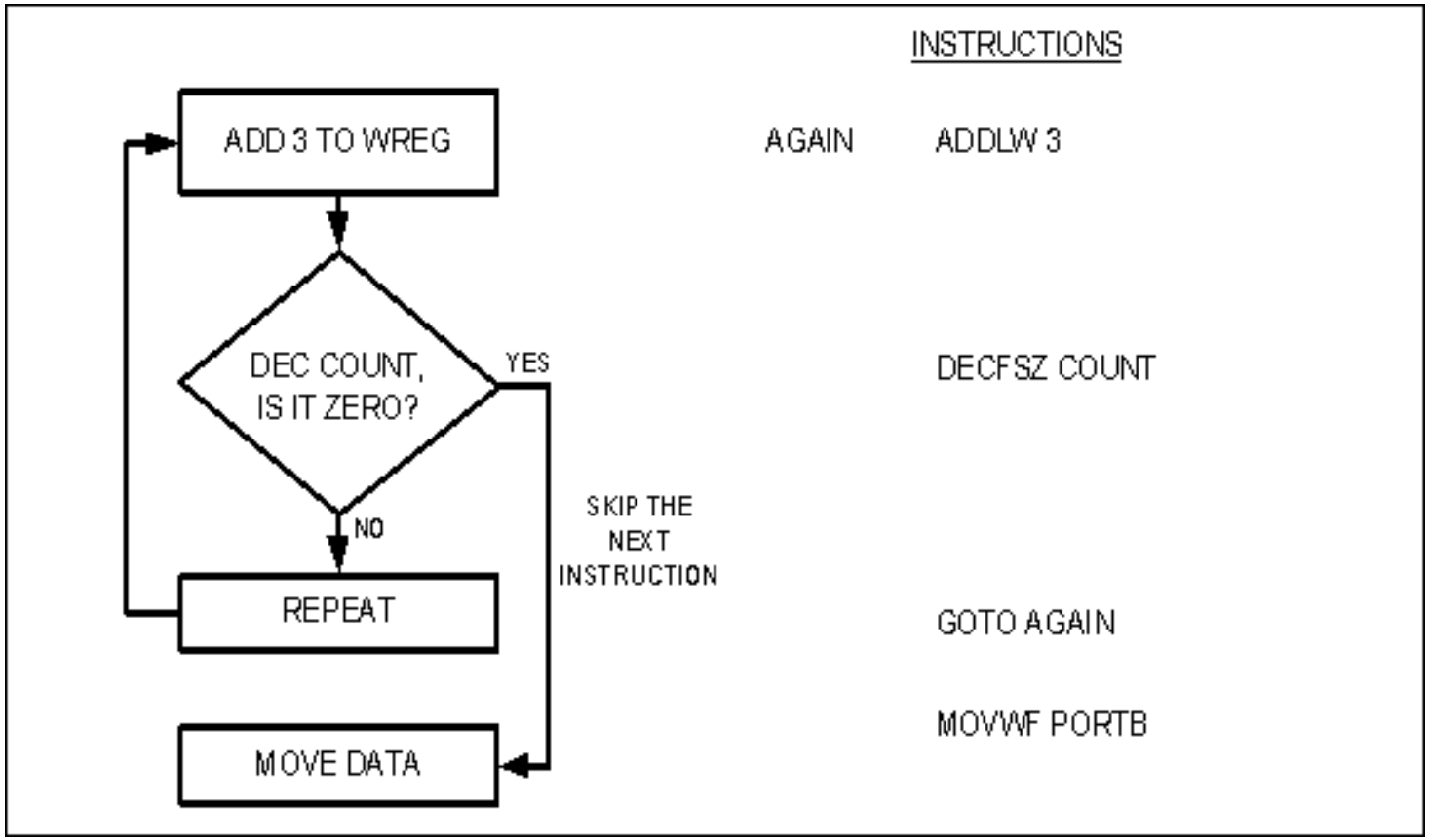


Figure 3-1. Flowchart for the DECFSZ Instruction

Using BNZ\BZ instructions

- Supported by PIC18 families
  - Early families such as PIC16 and PIC12 doesn't support these instruction
- These instructions check the status flag

```
Back        ...................
            ...................
            DECF  fileReg, f
            BNZ   Back
```

Example 3-2

- Write a program to
  a) Clear WREG
  b) Add 3 to WREG ten times and place the result in SFR PORTB

- Solution

```
COUNT    EQU 0x25
         MOVLW  d'10'
         MOVWF  COUNT
         MOVLW  0
AGAIN    ADDLW  3
         DECF   COUNT,F
         BNZ    AGAIN
         MOVWF  PORTB
```

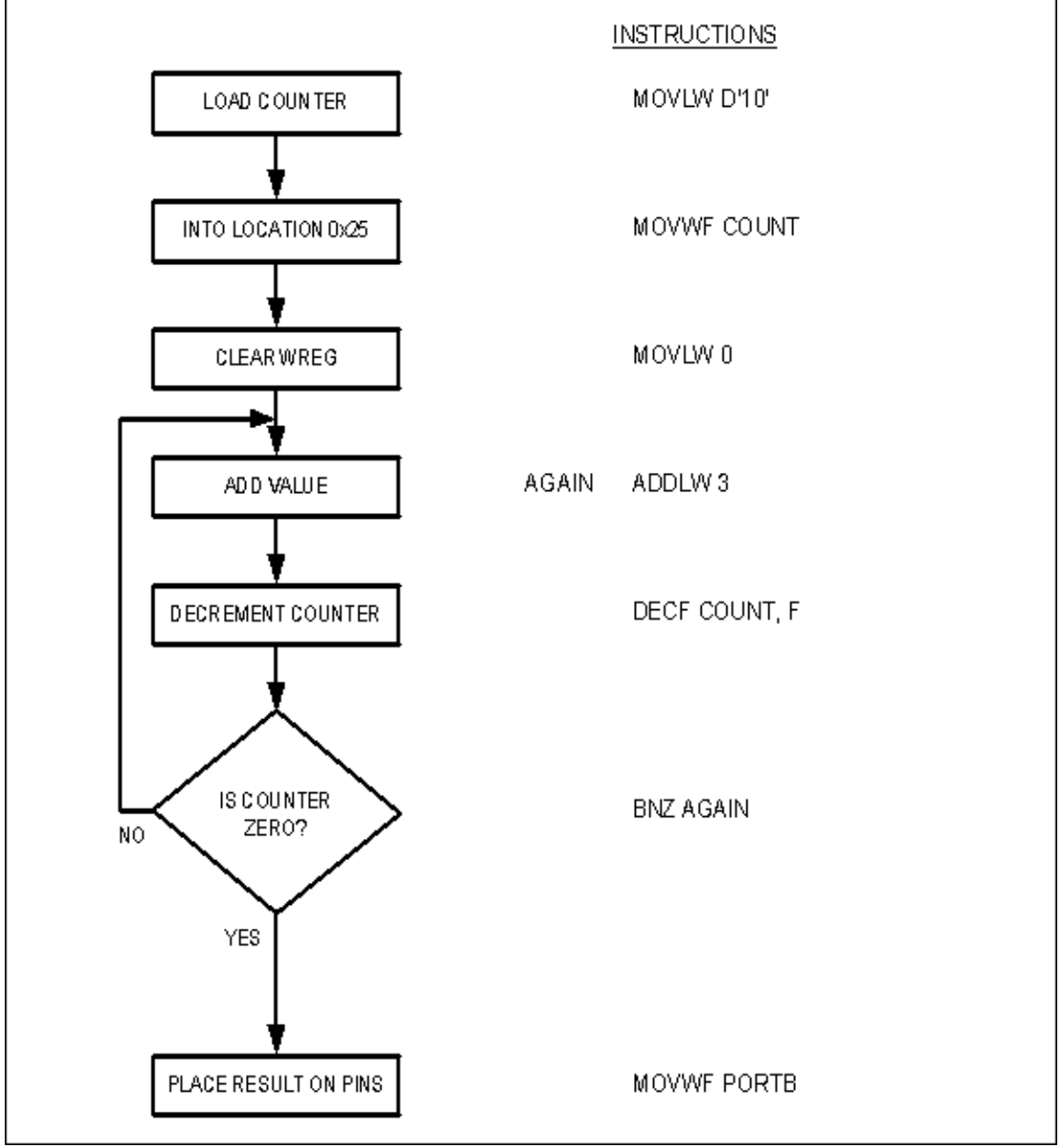


Figure 3-2. Flowchart for Example 3-2

Example 3-3

- What is the maximum number of times that the loop can be repeated?
- All locations in the FileReg are 8-bit
- The max. loop size is 255 time

Loop inside a loop

- Write a program to

a) Load the PORTB SFR register with the value 55H

b) Complement PORTB **700** times

Solution

```
R1 EQU 0x25
R2 EQU 0x26
COUNT_1 EQU d'10'
COUNT_2 EQU d'70'
```

Solution

```
        MOVLW 0x55
        MOVWF PORTB
        MOVLW COUNT_1
        MOVWF R1
LOP_1   MOVLW COUNT_2
        MOVWF R2
LOP_2   COMPF PORTB, F
        DECF R2, F
        BNZ LOP_2
        DECF R1, F
        BNZ LOP_1
```

Address	Data
25H (R1)	**10**
26H (R2)	**70**
…	
…	
F81H (PORTB)	**55**

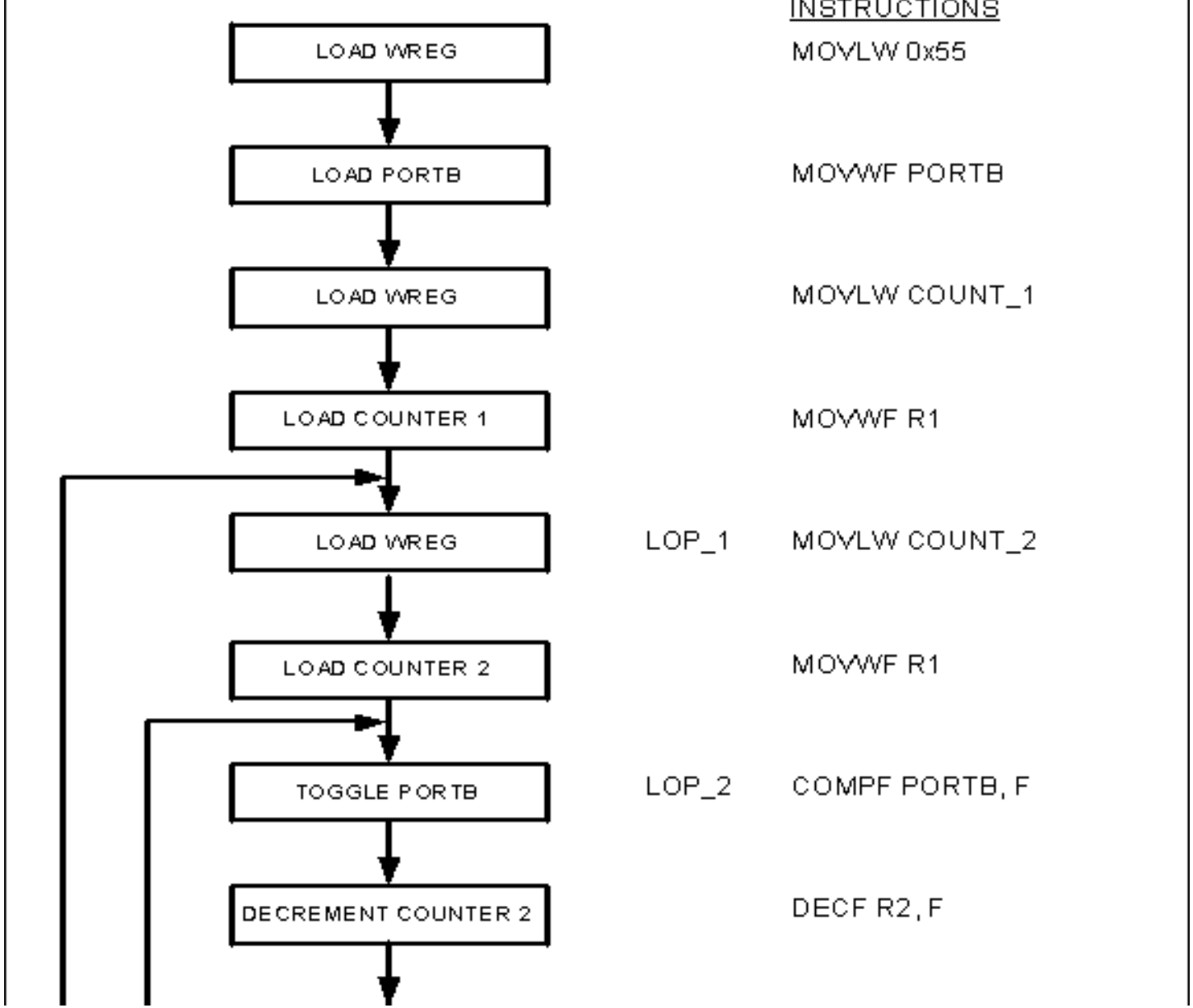


Figure 3-3. Flowchart

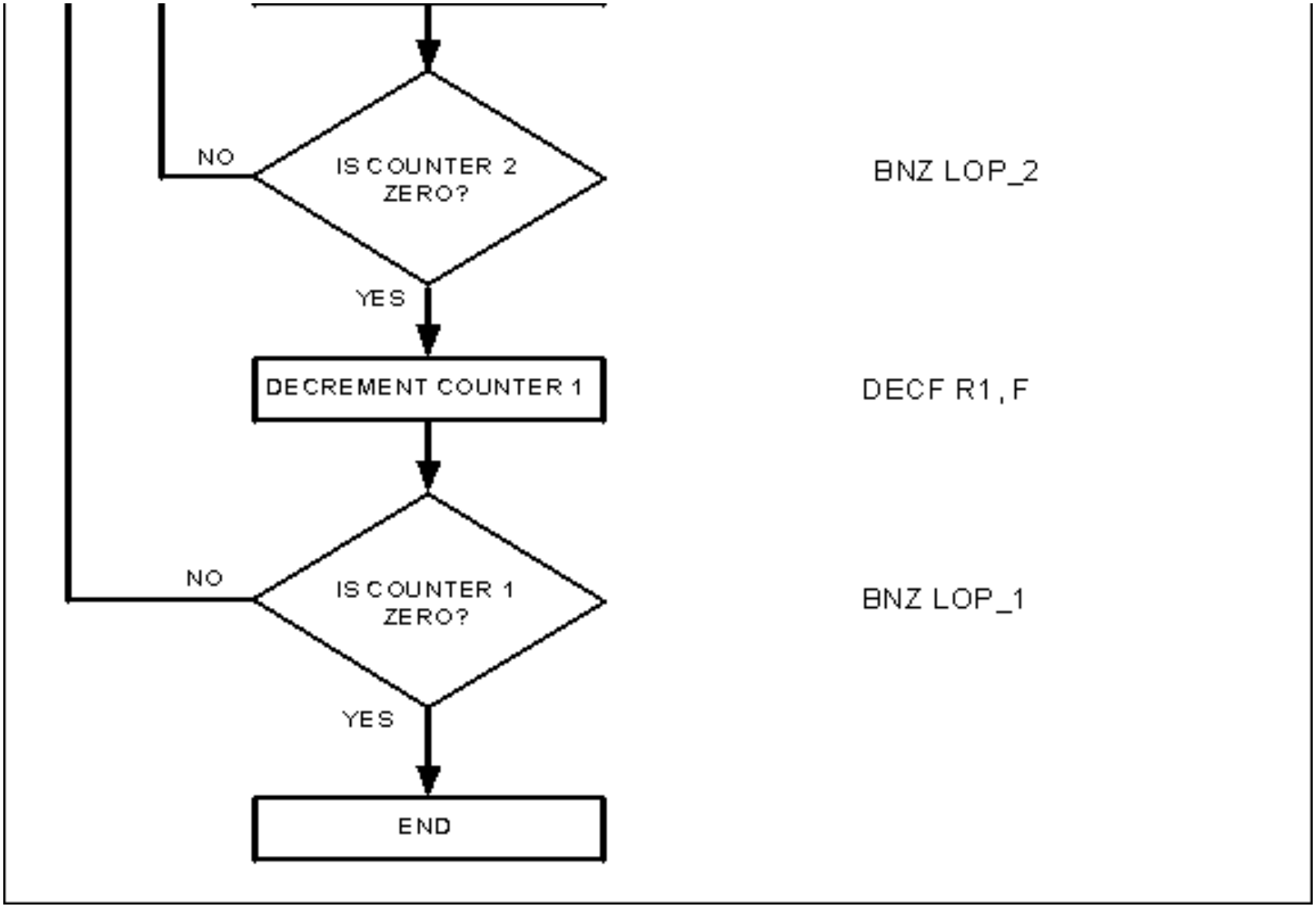


Figure 3-3. (continued)

Other conditional jumps

- All of the 10 conditional jumps are 2-byte instructions
- They requires the target address
  - 1 byte address (short branch address)
  - Relative address
- Recall: MOVF will affect the status Reg.
- In the BZ instruction, the Z flag is checked. If it is high, that is equal 1, it jumps to the target address.

Flag Bits and Decision Making

BC	k	Branch relative if Carry
BNC	k	Branch relative if Not Carry
BN	k	Branch relative if Negative
BNN	k	Branch relative if Not Negative
BOV	k	Branch relative if Overflow
BNOV	k	Branch relative if Not Overflow
BZ	k	Branch relative if Zero
BNZ	k	Branch relative if Not Zero

Example 3-5

- Write a program to determine if the loc. 0x30 contains the value 0. if so, put 55H in it.
- Solution:
- MYLOC EQU 0x30
- MOVF MYLOC, F
- BNZ NEXT
- MOVLW 0x55
- MOVWF MYLOC
- NEXT ...

Example 3-6

- Find the sum of the values 79H, F5H, and E2H. Put the sum in fileReg loc. 5H and 6H.

Address	Data
5H (Low-Byte)	0
6H (High-Byte)	0
…	

79

79+F5 = 16E

Address	Data
5H (Low-Byte)	0
6H (High-Byte)	1

6E

6E+E2 = 150

Address	Data
5H (Low-Byte)	50
6H (High-Byte)	2
…	

50

Solution

```
L_Byte EQU 0x5
H_Byte EQU 0x6
       ORG 0h
       MOVLW 0x0
       MOVWF H_Byte
       ADDLW 0x79
       BNC   N_1
       INCF  H_Byte,F
N_1    ADDLW 0xF5
       BNC   N_2
       INCF  H_Byte,F
N_2    ADDLW 0xE2
       BNC   OVER
       INCF  H_Byte,F
OVER   MOVWF L_Byte
       END
```

Example 3-7

```
000000 0E00    00004        MOVLW   0x0
000002 6E06    00005        MOVWF   H_Byte
000004 0F79    00006        ADDLW   0x79
000006 E301    00007        BNC     N_1
000008 2A06    00008        INCF    H_Byte,F
00000A 0FF5    00009  N_1   ADDLW   0xF5
00000C E301    00010        BNC     N_2
00000E 2A06    00011        INCF    H_Byte,F
000010 0FE2    00012  N_2   ADDLW   0xE2
000012 E301    00013        BNC     OVER
000014 2A06    00014        INCF    H_Byte,F
000016 6E05    00015  OVER  MOVWF   L_Byte
```

Example 3-7

Program Memory

Line	Address	Opcode	Label	Disassembly
1	0000	0E00		MOVLW 0
2	0002	6E06		MOVWF H_Byte, ACCESS
3	0004	0F79		ADDLW 0x79
4	0006	E301		BNC N_1
5	0008	2A06		INCF H_Byte, F, ACCESS
6	000A	0FF5	N_1	ADDLW 0xf5
7	000C	E301		BNC N_2
8	000E	2A06		INCF H_Byte, F, ACCESS
9	0010	0FE2	N_2	ADDLW 0xe2
10	0012	E301		BNC OVER
11	0014	2A06		INCF H_Byte, F, ACCESS
12	0016	6E05	OVER	MOVWF L_Byte, ACCESS
13	0018	FFFF		NOP
14	001A	FFFF		NOP
15	001C	FFFF		NOP
16	001E	FFFF		NOP
17	0020	FFFF		NOP
18	0022	FFFF		NOP
19	0024	FFFF		NOP

The

Example 3-8

Program Memory

Line	Address	Opcode	Label	Disassemb
1	0000	0E0A		MOVLW 0xa
2	0002	6E25		MOVWF COUNT, ACCESS
3	0004	0E00		MOVLW 0
4	0006	0F03	AGAIN	ADDLW 0x3
5	0008	0625		DECF COUNT, F, ACCESS
6	000A	E1FD		BNZ AGAIN
7	000C	6E81		MOVWF PORTB, ACCESS
8	000E	FFFF		NOP
9	0010	FFFF		NOP
10	0012	FFFF		NOP
11	0014	FFFF		NOP
12	0016	FFFF		NOP
13	0018	FFFF		NOP

Question?

Which is better, to use BNZ along with DECF or DCFSNZ??

Unconditional branch instruction

- Control is transferred unconditionally to the target location (at ROM)
- Tow unconditional branches
  - GOTO
  - BRA

1110	1111	k_7kkk	$kkkk_0$
1111	$k_{19}kkk$	kkkk	$kkkk_8$

$$0 \leq k \leq FFFFF$$

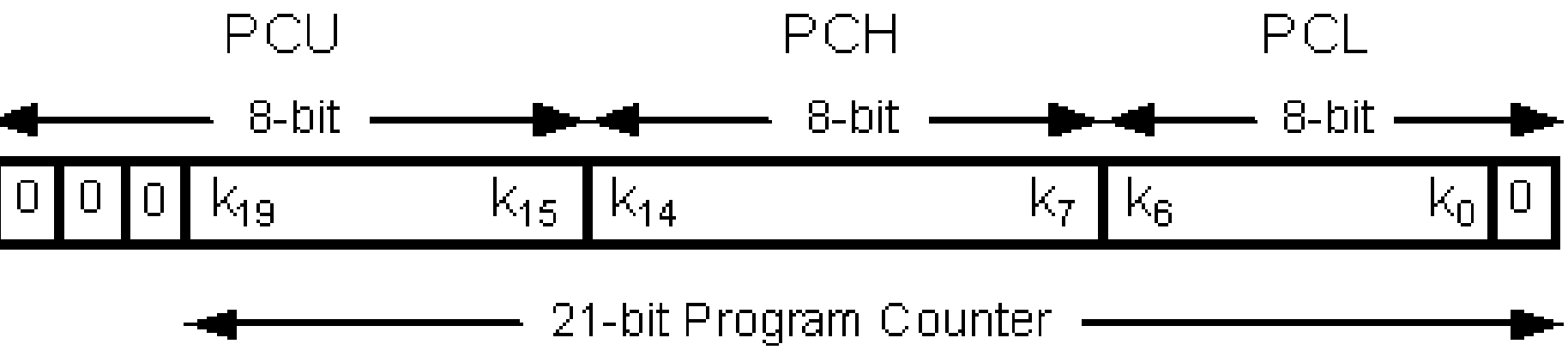


Figure 3-4. GOTO Instruction

BRA Instruction

1110	0nnn	nnnn	nnnn

$-1024 \leq n \leq 1023$

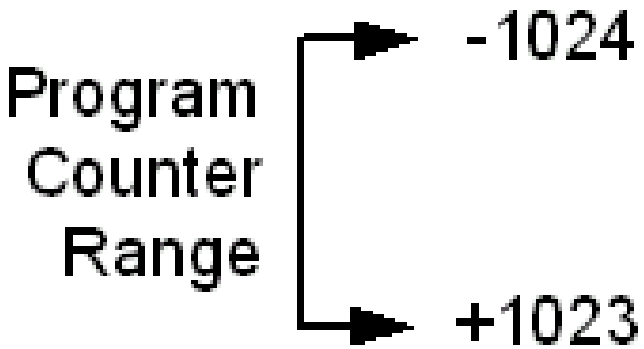


Figure 3-5. BRA (Branch Unconditionally
Instruction Address Range

BRA Instruction

Program Memory

Line	Address	Opcode	Label	Disassembly
1	0000	0E0A		MOVLW 0xa
2	0002	6E25		MOVWF COUNT, ACCESS
3	0004	D001		BRA AGAIN
4	0006	0E00		MOVLW 0
5	0008	0F03	AGAIN	ADDLW 0x3
6	000A	0625		DECF COUNT, F, ACCESS
7	000C	E1FD		BNZ AGAIN
8	000E	6E81		MOVWF PORTB, ACCESS
9	0010	D7FB		BRA AGAIN
10	0012	FFFF		NOP
11	0014	FFFF		NOP
12	0016	FFFF		NOP

Forward jump

Backward jump

GOTO to itself

- Label and $ can be used to keep uC busy (jump to the same location)
- HERE GOTO HERE
- GOTO $

- OVER BRA OVER
- BRA $

PIC18 Call instruction

Section 3-2

Call instruction

Call a subrutine

Call

4-byte instruction

Long Call

Rcall

2-byte instruction

Relative Call

CALL Instruction

1110	110S	k_7kkk	$kkkk_0$
1111	$k_{19}kkk$	kkkk	$kkkk_8$

$0 \leq k \leq FFFFF$

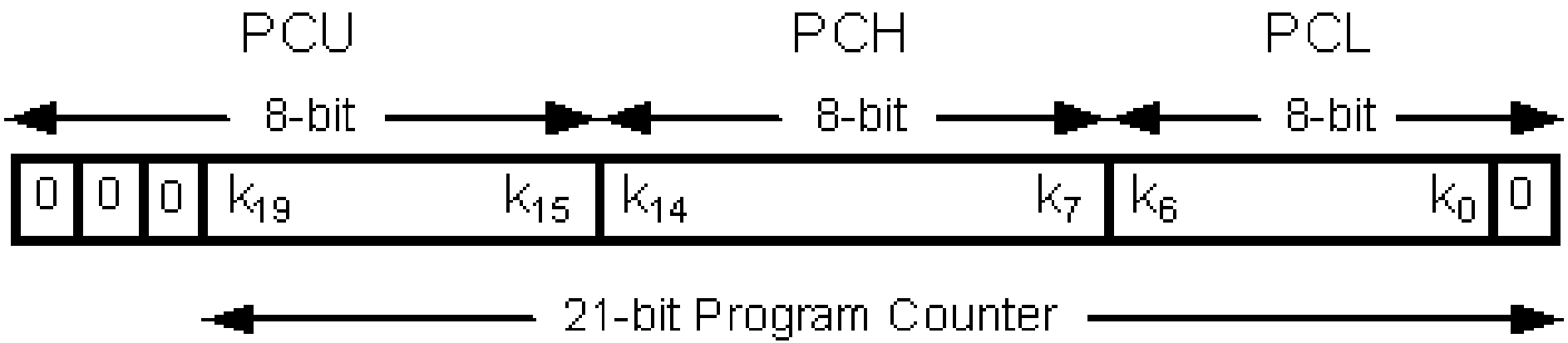


Figure 3-6. CALL Instruction

CALL Instruction

- Control is transferred to subroutine
- Current PC value, the instruction address just below the CALL instruction, is stored in the stack
  - push onto the stack
- Return instruction is used to transfer the control back to the caller,
  - the previous PC is popped from the stack

Stack and Stack Pointer (SP)

- Read/Write Memory
- Store the PC Address
  - 21-bit (000000 to 1FFFFF)
- 5-bit stack, total of 32 locations
- SP points to the last used location of the stack
  - Location 0 doesn't used
  - Incremented pointer

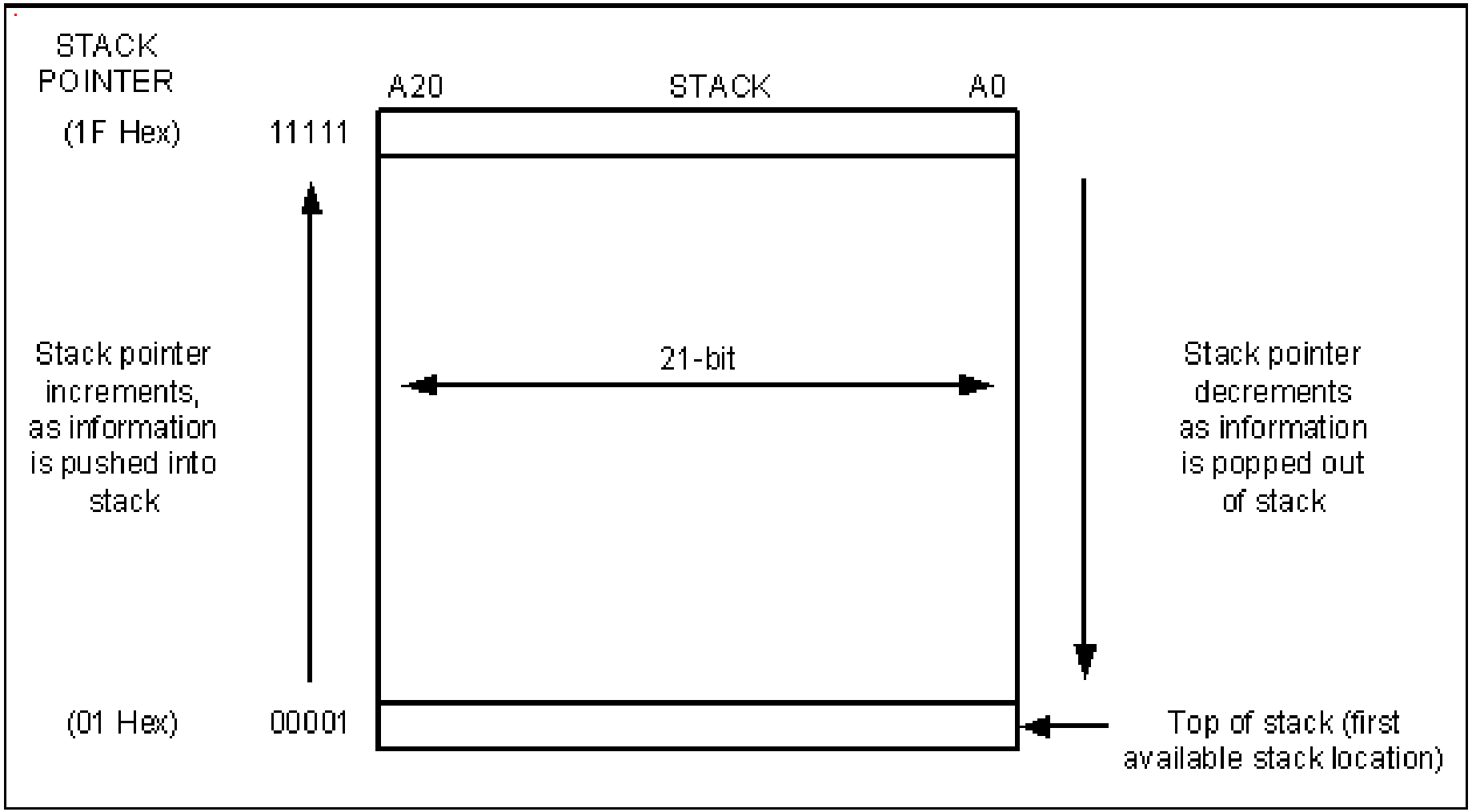


Figure 3-7. PIC Stack 31 × 21

Return from Subroutine

- The stack is popped and the top of the stack (TOS)is loaded into the program counter.
- If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR.
- If 's' = 0, no update of these registers occurs (default).

0000	0000	0001	001s

Example 3-9

- Toggle all bits of to SFR register of PORTB by sending to it values 55H and AAH continuously. Put a delay in between issuing of data to PORTB .
- Analyze the stack for the CALL instructions

Solution

```
MYREG     EQU  0x08
PORTB     EQU 0x0F8
          ORG 0
BACK      MOVLW  0x55
          MOVWF  PORTB
          CALL   DELAY
          MOVLW  0xAA
          MOVWF  PORTB
          CALL   DELAY
          GOTO   BACK
```

```
          ORG    20H
DELAY     MOVLW      0xFF
          MOVWF MYREG
AGAIN       NOP
            NOP
            DECF  MYREG, F
            BNZ  AGAIN
        RETURN
        END
```

Example 3-10

Address	Data
4	
3	
2	
1	000008

Program Memory

Line	Address	Opcode	Label	Disasse
1	0000	0E55	BACK	MOVLW 0x55
2	0002	6E81		MOVWF PORTB, ACCESS
3	0004	EC10		CALL DELAY, 0
4	0006	F000		NOP
5	0008	0EAA		MOVLW 0xaa
6	000A	6E81		MOVWF PORTB, ACCESS
7	000C	EC10		CALL DELAY, 0
8	000E	F000		NOP
9	0010	EF00		GOTO BACK
10	0012	F000		NOP
11	0014	FFFF		NOP
12	0016	FFFF		NOP
13	0018	FFFF		NOP
14	001A	FFFF		NOP
15	001C	FFFF		NOP
16	001E	FFFF		NOP
17	0020	0EFF	DELAY	MOVLW 0xff
18	0022	6E08		MOVWF MYREG, ACCESS
19	0024	0000	AGAIN	NOP
20	0026	0000		NOP
21	0028	0608		DECF MYREG, F, ACCESS
22	002A	E1FC		BNZ AGAIN
23	002C	0012		RETURN 0
24	002E	FFFF		NOP
25	0030	FFFF		NOP

```
;MAIN program calling subroutines
            ORG   0
MAIN        CALL  SUBR_1
            CALL  SUBR_2
            CALL  SUBR_3

HERE        BRA   HERE        ;stay here
;————————end of MAIN
;
SUBR_1      ....
            ....
            RETURN
;————————end of subroutine 1
;
SUBR_2      ....
            ....
            RETURN
;————————end of subroutine 2

SUBR_3      ....
            ....
            RETURN
;————————end of subroutine 3
            END               ;end of the asm file
```

Figure 3-8. PIC Assembly Main Program That Calls Subroutines

RCALL (Relative Call)

- 2-Byte instruction
- The target address must be within 2K
  - 11 bits of the 2 Byte is used
- Save a number of bytes.

1101	1nnn	nnnn	nnnn

Example 3-12

Program Memory

Line	Address	Opcode	Label	Disassembly
1	0000	0E55		MOVLW 0x55
2	0002	6E81	BACK	MOVWF PORTB, ACCESS
3	0004	D802		RCALL DELAY
4	0006	1E81		COMF PORTB, F, ACCESS
5	0008	D7FC		BRA BACK
6	000A	0EFF	DELAY	MOVLW 0xff
7	000C	6E08		MOVWF MYREG, ACCESS
8	000E	0000	AGAIN	NOP
9	0010	0000		NOP
10	0012	0608		DECF MYREG, F, ACCESS
11	0014	E1FC		BNZ AGAIN
12	0016	0012		RETURN 0
13	0018	FFFF		NOP
14	001A	FFFF		NOP
15	001C	FFFF		NOP
16	001E	FFFF		NOP
17	0020	FFFF		NOP
18	0022	FFFF		NOP
19	0024	FFFF		NOP

Opcode Hex | Machine | Symbolic

PIC18 Time Delay and instruction pipeline

Section 3-3

Delay Calculating for PIC18

- Two factors can affect the accuracy of the delay

1. The duration of the clock period, which is function of the Crystal freq
   - Connected to OSC! And OSC2
2. The instruction cycle duration
   - Most of the PIC18 instructions consumes 1 cycle
     - Use Harvard Architecture
     - Use RISC Architecture
     - Use the pipeline concept between fetch and execute.

Non-pipeline	fetch 1	exec 1	fetch 2	exec 2	fetch 3	exec 3

Figure 3-9. Pipeline vs. Non-pipeline

PIC multistage pipeline

- Superpipeline is used to speed up execution.
- The process of executing instructions is split into small steps
- Limited to the slowest step

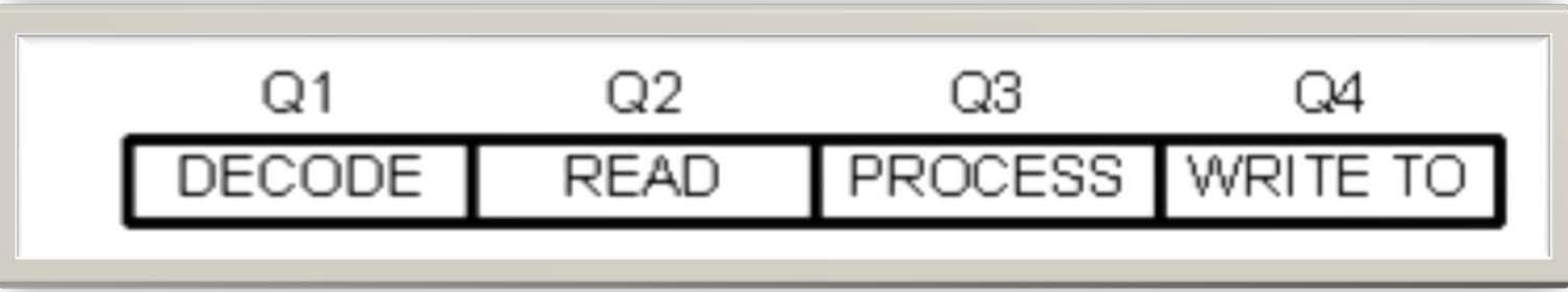


Figure 3-10. Pipeline Activity After the Instruction Has Been Fetched

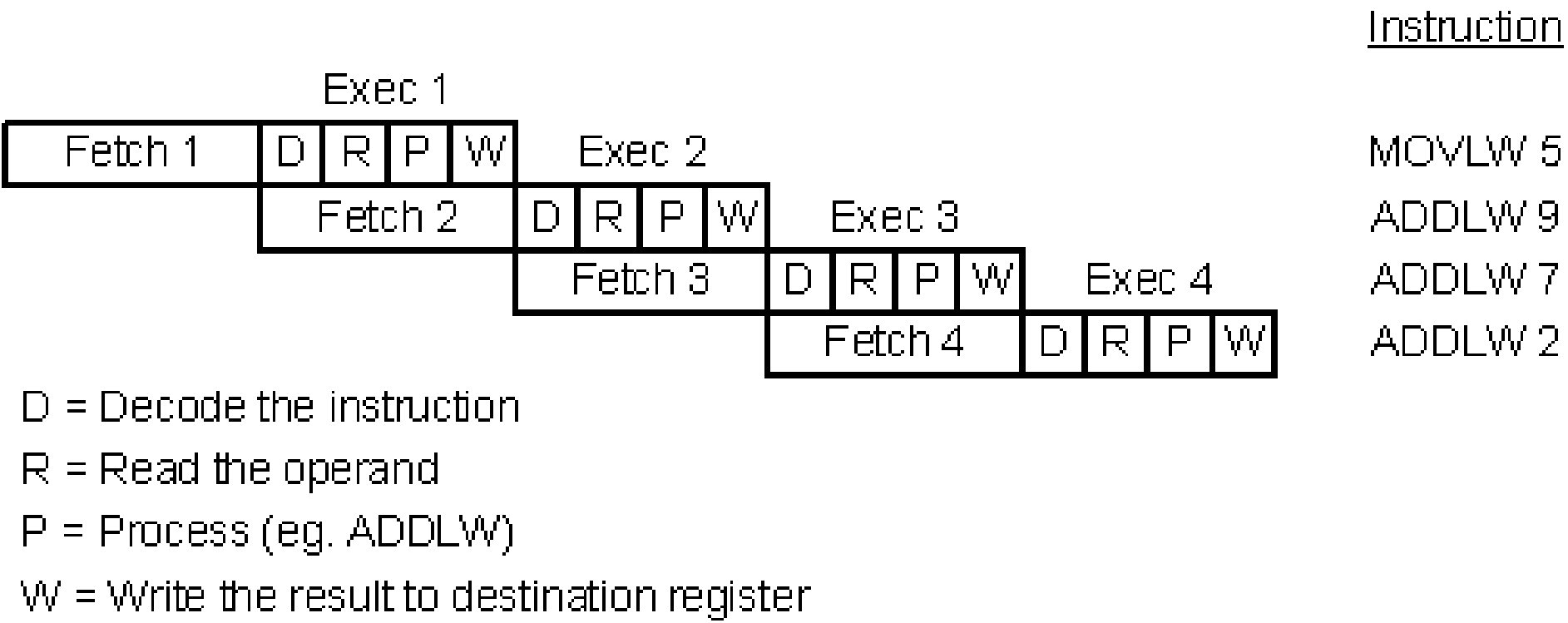


Figure 3-11. Pipeline Activity for Both Fetch and Execute

Instruction Cycle time for the PIC

- What is the Instruction Cycle ?
- Most instructions take one or tow cycles
  - BTFSS can take up to 3 cycles
- Instruction Cycle depends on the freq. of oscillator
- Clock source: Crystal oscillator and on-chip circuitry
- One instruction cycle consists of four oscillator period.

Example 3-14

- Find the period of the instruction cycle you chose 4 MHz crystal? And what is required time for fetching an instruction?
- Solution
- 4 MHz/4 =1 MHz
- Instruction Cycle = 1/1MHz = 1 usec
- Fetch cycle = 4 * 1 usec = 4 usec

Branch penalty

- Queue is needed for prefetched instruction
- If the prefetched instruction is incorrect, the CPU must flush the memory. When??

If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Branch penalty

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOVLW 55h	Fetch 1	Execute 1				
2. MOVWF PORTB		Fetch 2	Execute 2			
3. BRA SUB_1			Fetch 3	Execute 3		
4. BSF PORTA, BIT3 (Forced NOP)				Fetch 4	Flush (NOP)	
5. Instruction @ address SUB_1					Fetch SUB_1	Execute SUB_1

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

BTFSC and BTFSS

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example 3-15

- Find how long it take to execute each of the following instructions for a PIC18 with 4 MHz
- MOVLW
- ADDLW
- DECF
- NOP
- MOVWF

CALL

GOTO

BNZ

Delay calculation for PIC18 Example 3-16

- Find the size of the delay in the following program if the crystal freq. is 4MHz.

```
DELAY   MOVLW   0xFF
        MOVWF   MYREG
AGAIN   NOP
        NOP
        DECF    MYREG, F
        BNZ     AGAIN
        RETERN
```

Example 3-17

```
MYREG   EQU 0x08
ORG 0
BACK    MOVLW 0x55
MOVWF   PORTB
CALL    DELAY
MOVLW   0xAA
MOVWF   PORTB
CALL    DELAY
GOTO    BACK

ORG 300H
DELAY   MOVLW 0xFA
MOVWF   MYREG
AGAIN   NOP
        NOP
        NOP
DECF    MYREG, F
BNZ AGAIN
RETURN
```

Example 3-20

```
R2      EQU 0x2
R3      EQU 0x3
R4      EQU 0x4
  MOVLW       0x55
  MOVWF       PORTB
BACK
CALL DELAY_500MS
  COMF    PORTB
  GOTO    BACK
```

```
DELAY_500MS
        MOVLW       D'20'
        MOVWF               R4
BACK
        MOVLW D'100'
        MOVWF       R3
AGAIN
   MOVLW D'250'
        MOVWF       R2
HERE NOP
         NOP
        DECF   R2, F
        BNZ         HERE
        DECF   R3, F
        BNZ         AGAIN
        DECF   R4, F
        BNZ         BACK
        RETURN
```

Chapter 3: Summary

- Looping in PIC Assembly language is performed using an instruction to decrement a counter and to jump to the top of the loop if the counter is not zero.
- Assembly language includes conditional and unconditional, and call instructions.
- PIC18 uses Superpipeline is used to speed up execution.

Next: Chapter 4
PIC I/O Port Programming

PIC Microcontroller and Embedded Systems

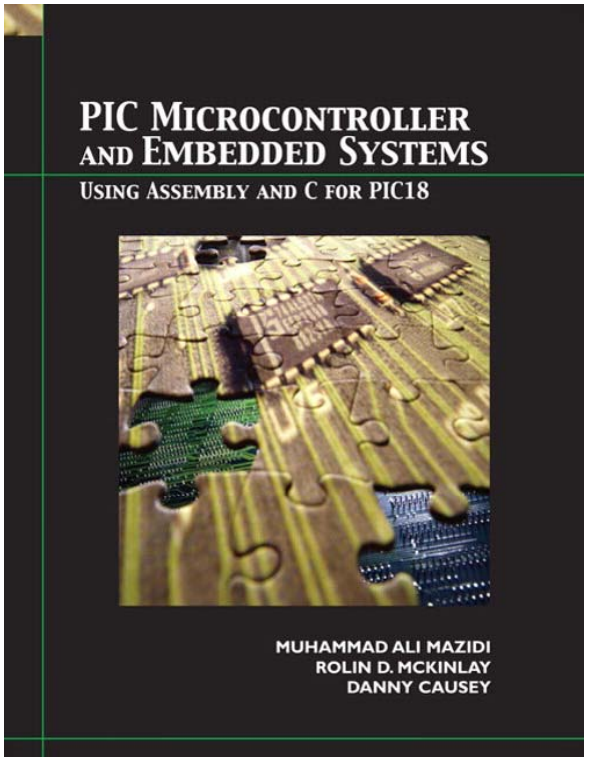
Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 4: PIC I/O Port Programming

- I/O Port Programming in PIC18
- I/O Bit Manipulation Programming



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

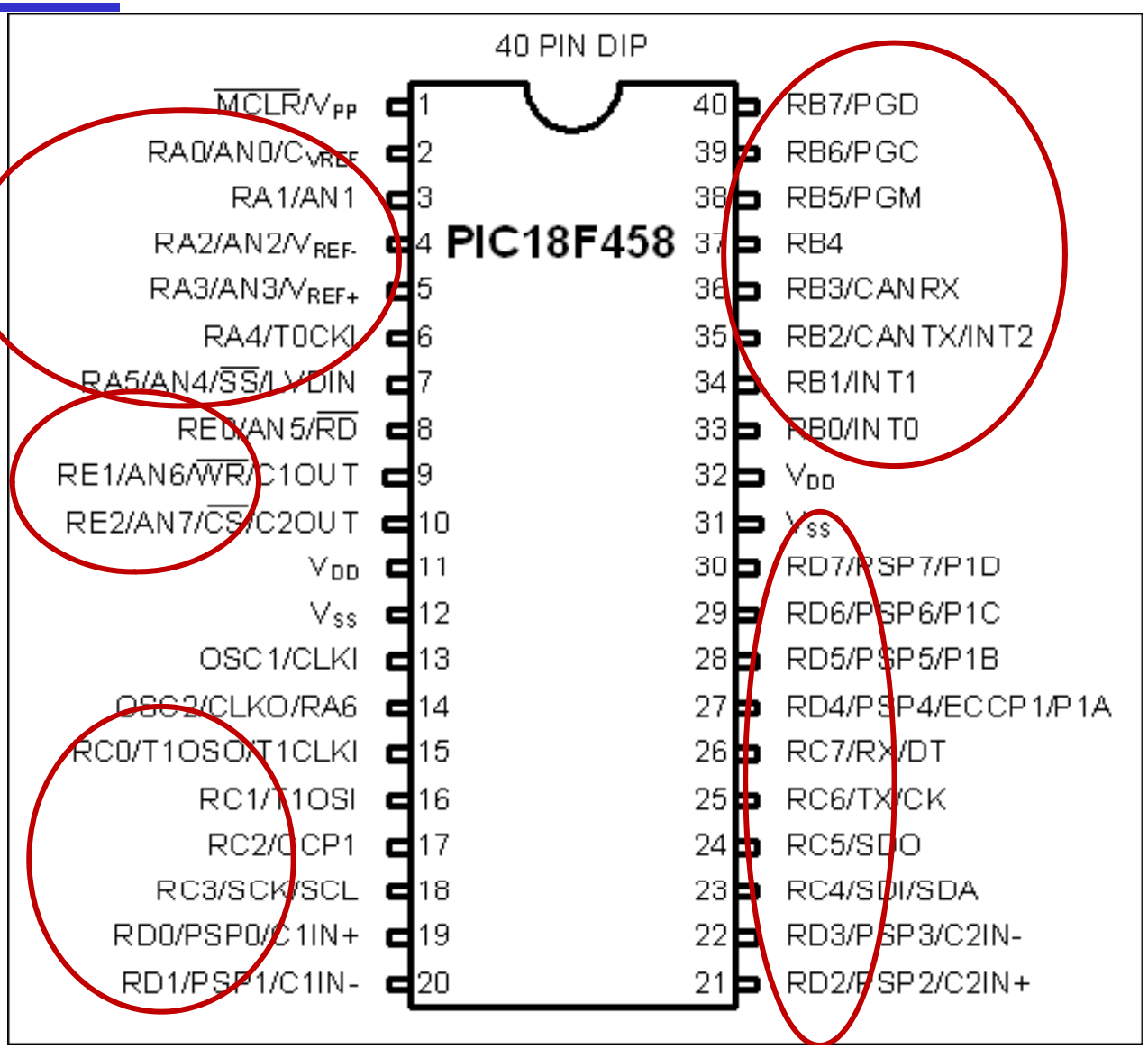
Objective

- List all the ports of the PIC18
- Describe the dual role of PIC18 pins
- Code Assembly to use ports for input or output
- Code PIC instructions for I/O handling
- Code I/O bit-manipulation Programs for PIC
- Explain the bit addressability of PIC ports

I/O Port Programming in PIC18

- PIC18 has many ports
  - Depending on the family member
  - Depending on the number of pins on the chip
  - Each port can be configured as input or output.
    - Bidirectional port
  - Each port has some other functions
    - Such as timer , ADC, interrupts and serial communication
  - Some ports have 8 bits, while others have not

Figure 4-1. PICF458 Pin Diagram



Pins	Add	18-pin	28-pin	40-pin	64-pin	80-pin
Chip		**PIC18F1220**	**PIC18F2220**	**PIC18F458**	**PIC18F6525**	**PIC18F8525**
PORT A	F80H	X	X	X	X	X
PORT B	F81H	X	X	X	X	X
PORT C	F82H		X	X	X	X
PORT D	F83H			X	X	X
PORT E	F84H			X	X	X
PORT F	F85H				X	X
PORT G	F86H				X	X
PORT H					X	X
PORT J					X	X
PORT K						X
PORT L						X

I/O SFR

- Each port has three registers for its operation:
  - TRIS register (Data Direction register)
    - If the corresponding bit is 0 → Output
    - If the corresponding bit is 1 → Input
  - PORT register (reads the levels on the pins of the device)
  - LAT register (output latch)
- The Data Latch (LAT) register is useful for read-modify-write operations on the value that the I/O pins are driving.

I/O SFR

- PIC18F458 has 5 ports
- Upon reset, all ports are configured as input
- TRISx register has 0FFH

Pins	Address
PORT A	F80H
PORT B	F81H
PORT C	F82H
PORT D	F83H
PORT E	F84H
LATA	F89H
LATB	F8AH
LATC	F8BH
LATD	F8CH
LATE	F8DH
TRISA	F92H
TRISB	F93H
TRISC	F94H
TRISD	F95H
TRISE	F96H

Figure 4-2. CMOS States for P and N Transistors

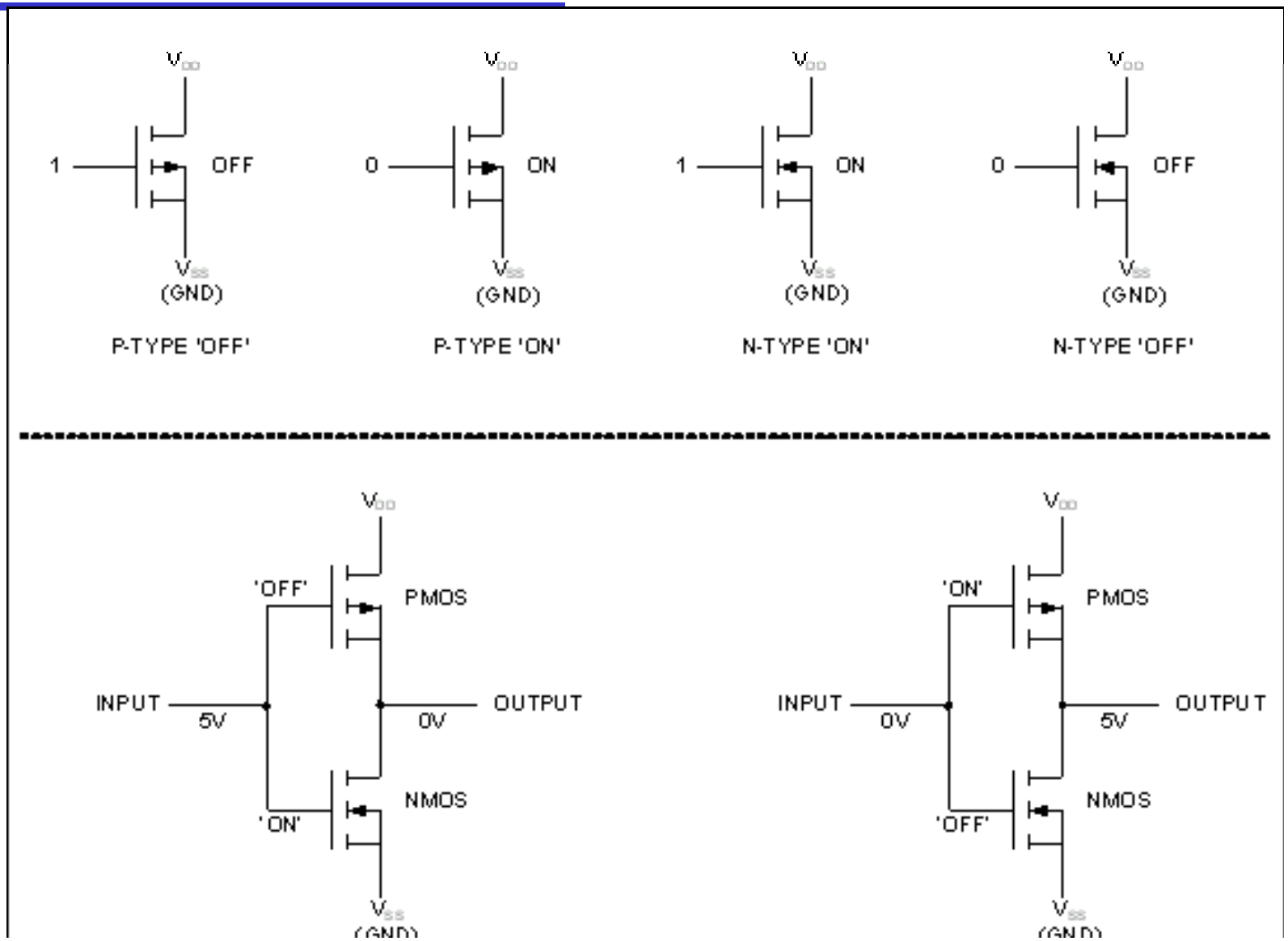


Figure 4-3. Outputting (Writing) 0 to a Pin in the PIC18

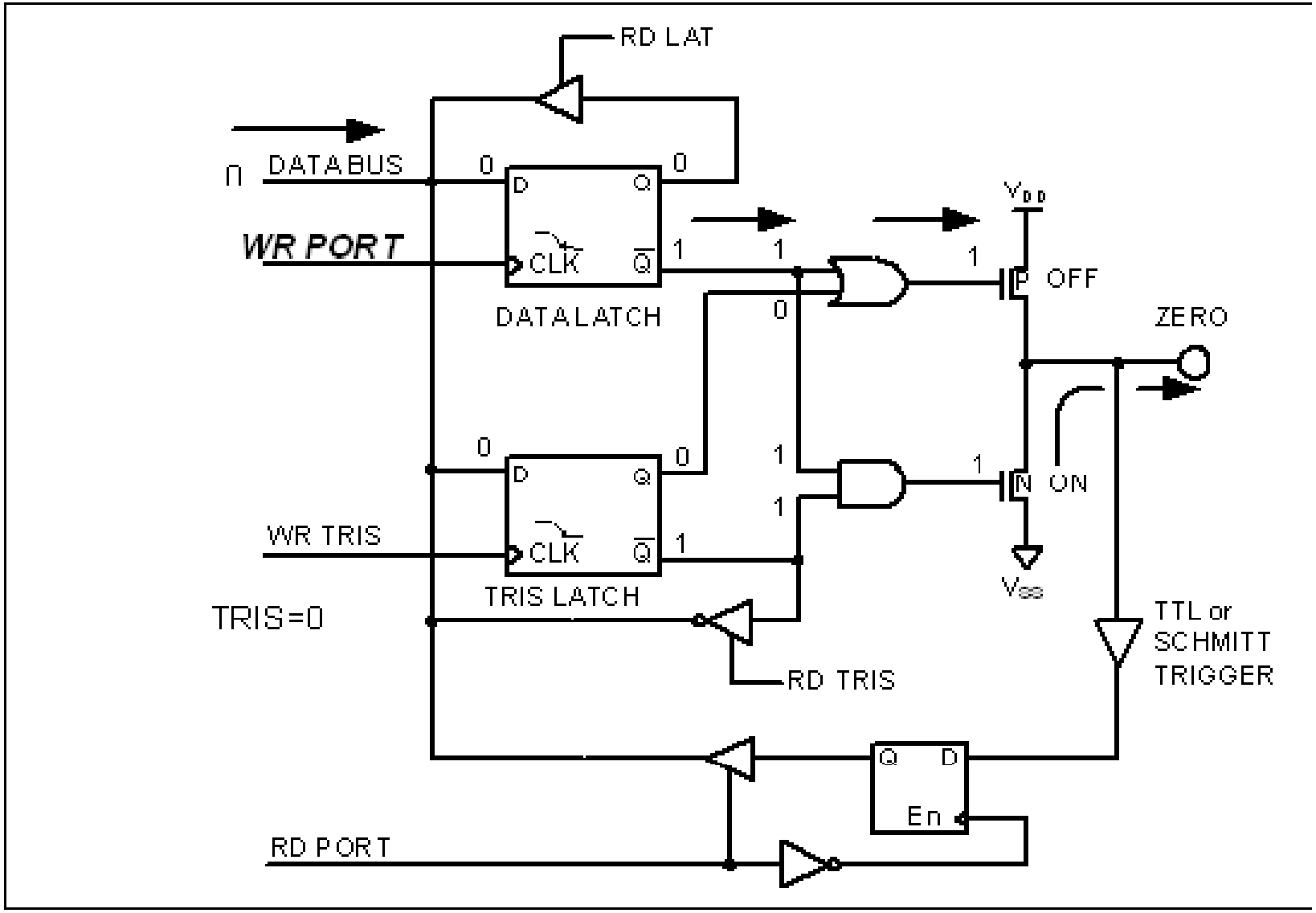


Figure 4-4. Outputting (Writing) 1 to a Pin in the PIC18

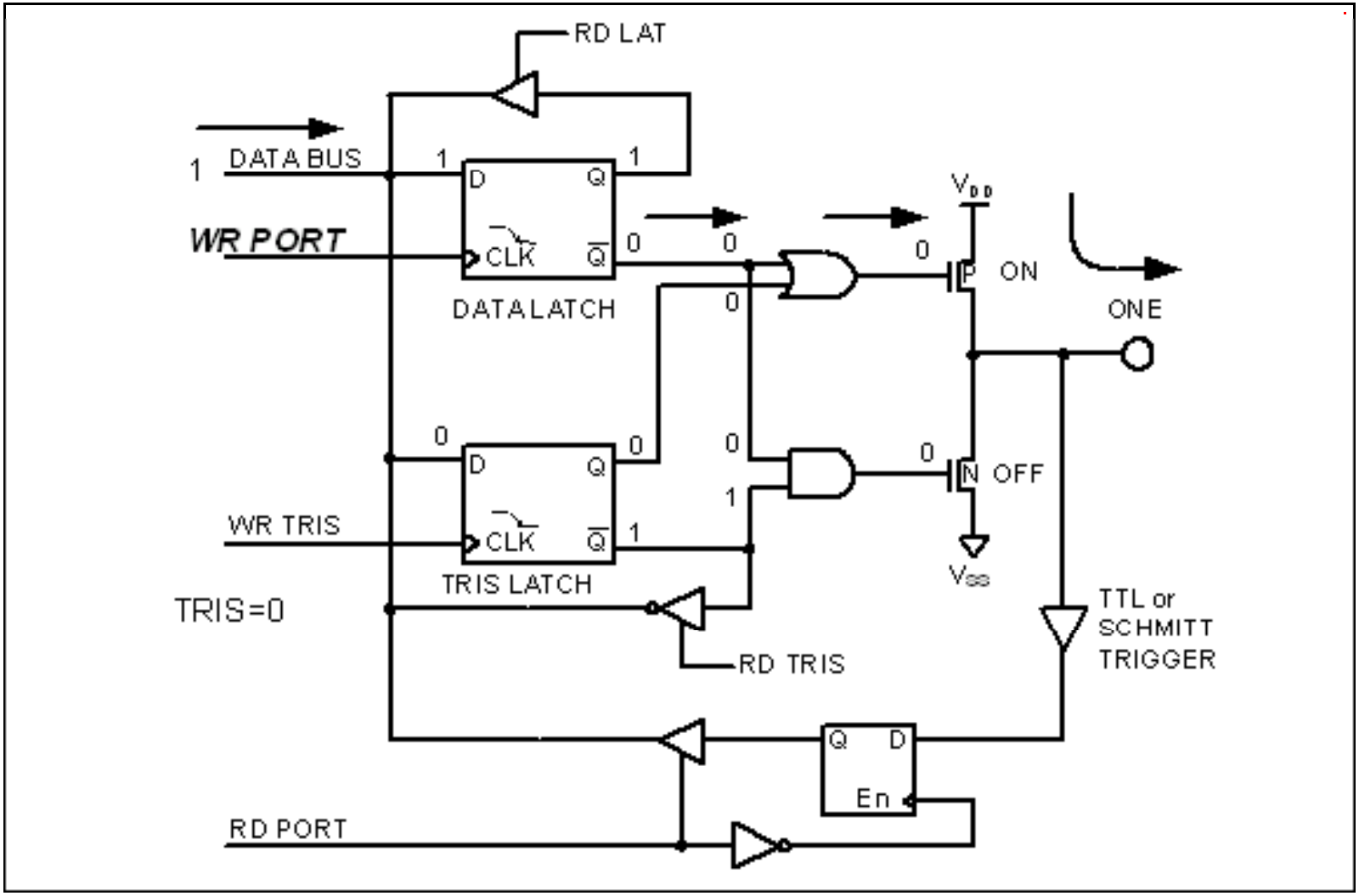


Figure 4-5. Inputting (Reading) 0 from a Pin in the PIC18

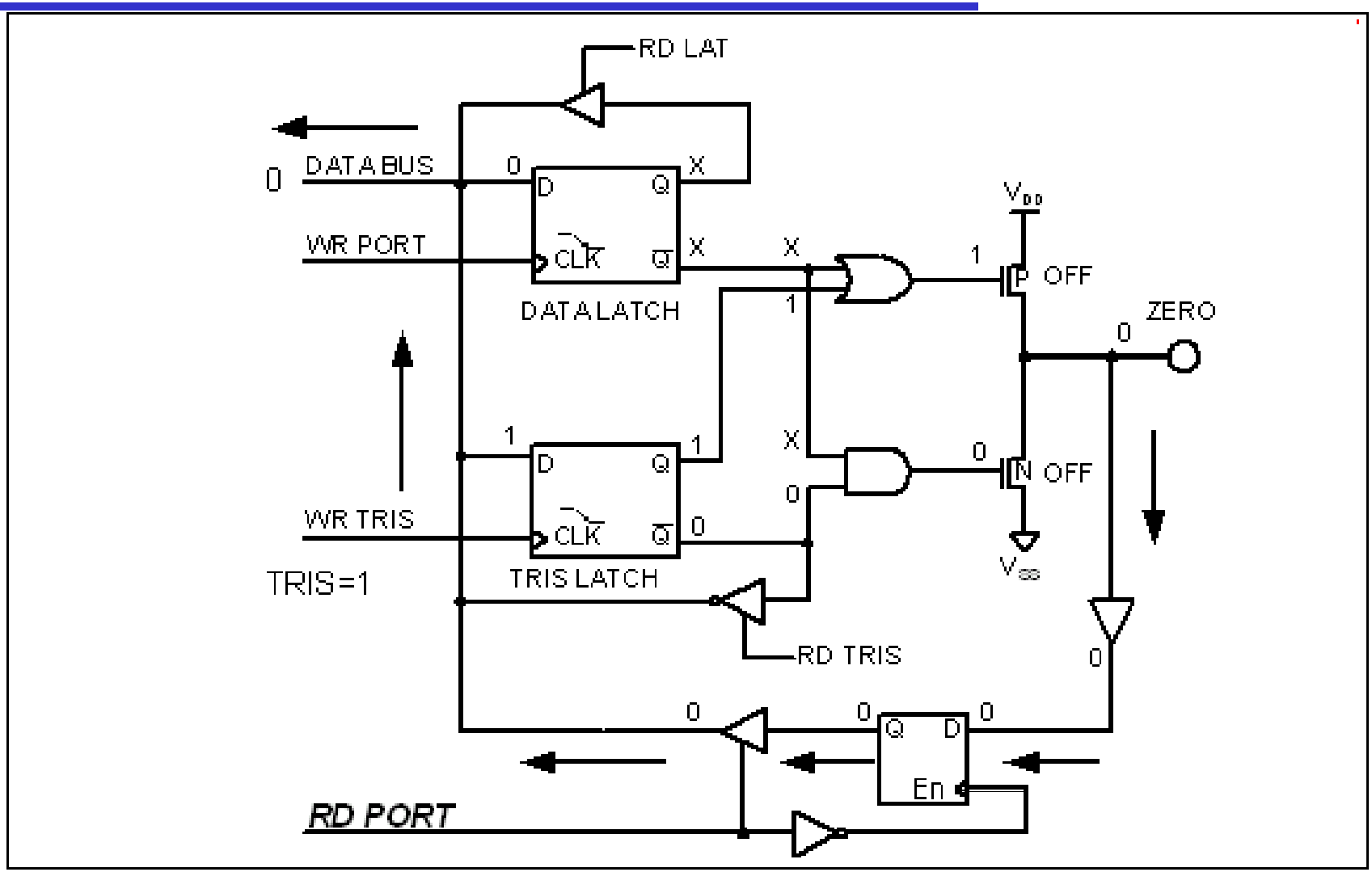
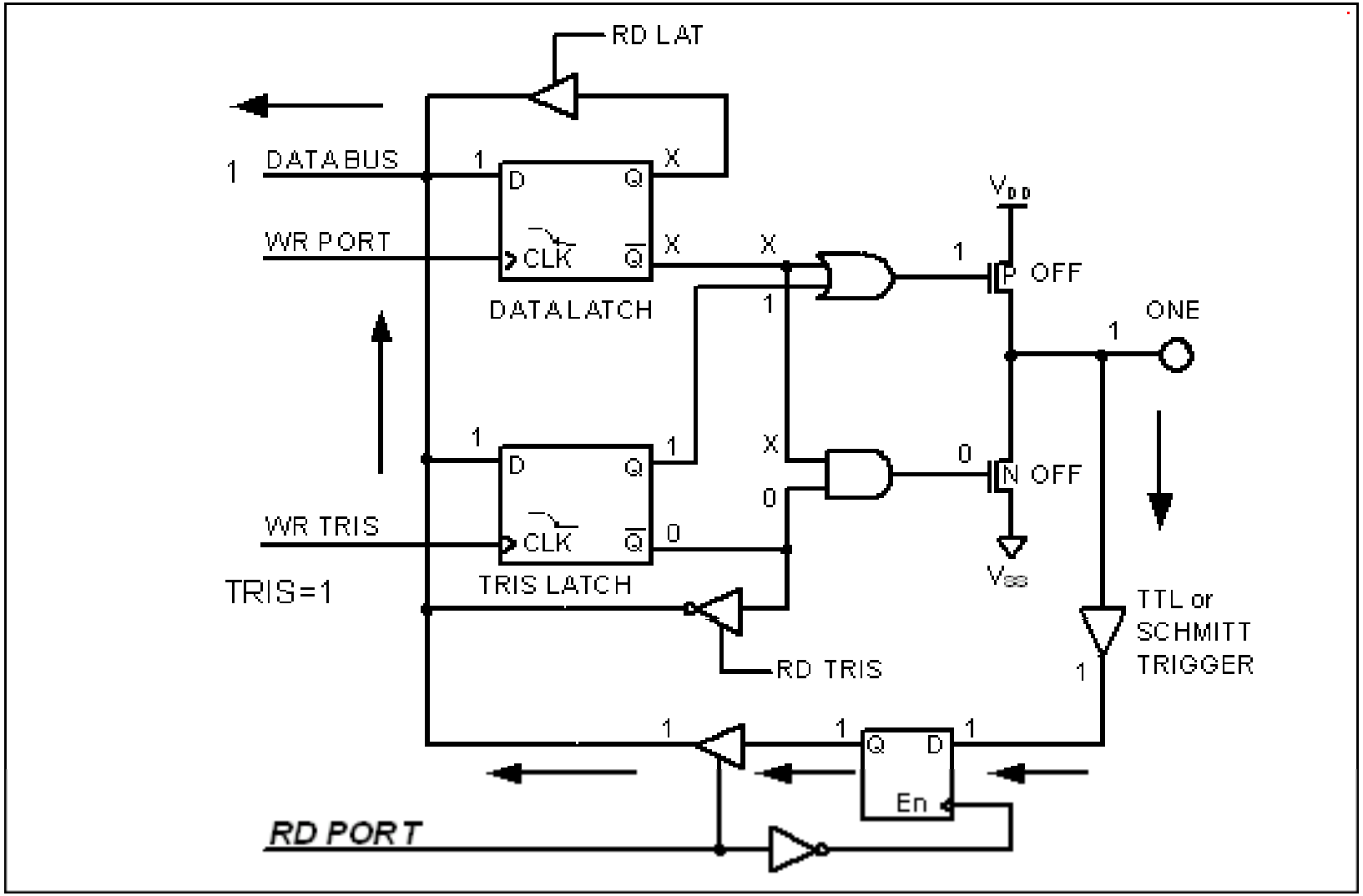


Figure 4-6. Inputting (Reading) 1 from a Pin in the PIC18



Port A

- PORTA is a 7-bit wide, bidirectional port.
  - Sometimes A6 is not available. why?
- The corresponding Data Direction register is TRISA.
- Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input
- Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output
- On a Power-on Reset, these pins are configured as inputs and read as '0'.

Example 1

```
BACK    MOVLW  0x55
        MOVWF  PORTA
        CALL   DELAY
        MOVLW  0xAA
        MOVWF  PORTA
        CALL   DELAY
        GOTO   BACK
```

```
        MOVLW  B'00000000'
        MOVWF  TRISA
BACK    MOVLW  0x55
        MOVWF  PORTA
        CALL   DELAY
        MOVLW  0xAA
        MOVWF  PORTA
        CALL   DELAY
        GOTO   BACK
```

Example 2

```
MYREG EQU 0x20
MOVLW  B'11111111'
MOVWF  TRISA
MOVF PORTA, w
MOVWF MYREG
```

PORT B, PORT C, PORT D and PORT E

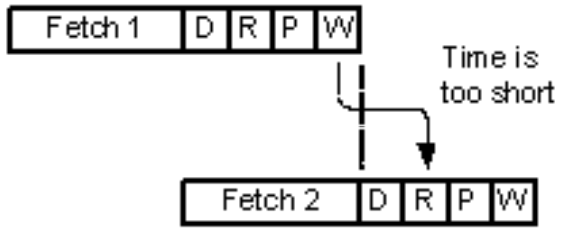
- PORTB is 8 pins
- PORTC is 8 pins
- PORTD is 8 pins
- PORTE is 3 pins

Read followed by write operation

- Be careful
  - Don't have a two I/O operations one right after the other.
- Data Dependency
  - A NOP is needed to make that data is written into WREG before it read for outputting to PortB.

```
CLRF      TRISB
SETF      TRISC
L4 MOVF   PORTC,W
MOVWF     PORTB
BRA L4
```

Figure 4-7. Pipeline for Read Followed by Write I/O

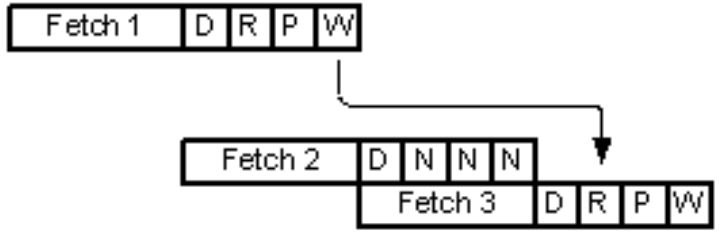


INSTRUCTION

MOVF PORTC,W ;Read PORTC into WREG

MOVWF PORTB ;Write WREG to PORTB

The RAW (Read – After – Write) for two consecutive instructions.



INSTRUCTION

MOVF PORTC,W

NOP ;Bubble in Pipeline

MOVWF PORTB

N = No Operation

D = Decode the instruction

Two Solutions

Solution1

```
CLRF      TRISB
SETF      TRISC
L4 MOVF   PORTC,W
NOP
MOVWF     PORTB
BRA L4
```

Solution2

```
CLRF      TRISB
SETF      TRISC
L4 MOVFF PORTC, PORTB
BRA L4
```

MOVFF is 4-byte instruction.

Example 4-1

Write a test program for the PIC18 chip to toggle all the bits of PORTB, PORTC and PORTD every 0.25 of a second. (suppose that there is a 4 MHz)

```
list P=PIC18F458
#include P18F458.INC
R1 equ 0x07
R2 equ 0x08
ORG 0
    CLRF    TRISB
    CLRF    TRISC
    CLRF    TRISD
    MOVLW   0x55
    MOVWF   PORTB
    MOVWF   PORTC
    MOVWF   PORTD
```

Solution

```
L3    COMF    PORTB,F
      COMF    PORTC,F
      COMF    PORTD,F
      CALL    QDELAY
      BRA  L3
```

```
QDELAY
   MOVLW D'200'
   MOVWF R1
D1    MOVLW D'250'
   MOVWF R2
D2    NOP
   NOP
   DECF R2, F
   BNZ D2
   DECF R1, F
   BNZ D1
   RETURN
   END
```

I/O Bit Manipulation Programming

- I/O ports and bit-addressability
- Monitoring a single bit
- Reading a single bit

Section 4-2

I/O ports and bit-addressability

PORT A	PORT B	PORT C	PORT D	PORT E	PORT Bit
RA0	RB0	RC0	RD0	RE0	D0
RA1	RB1	RC1	RD1	RE1	D1
RA2	RB2	RC2	RD2	RE2	D2
RA3	RB3	RC3	RD3		D3
RA4	RB4	RC4	RD4		D4
RA5	RB5	RC5	RD5		D5
	RB6	RC6	RD6		D6
	RB7	RC7	RD7		D7

Bit Oriented Instruction for PIC18

Instruction		Function
BSF	fileReg, bit	Bit Set File Register
BCF	fileReg, bit	Bit Clear File Register
BTG	fileReg, bit	Bit Toggl File Register
BTFSC	fileReg, bit	Bit Test File Register, skip if clear
BTFSS	fileReg, bit	Bit Test File Register, skip if set

Example 4-2

- A LED is connected to each pin of port D. Write a program to turn on each LED from pin D0 to D4.

```
CLRF TRISD
BSF  PORTD,0
CALL DELAY
BSF  PORTD,1
CALL DELAY
BSF  PORTD,2
CALL DELAY
BSF  PORTD,3
CALL DELAY
BSF  PORTD,4
CALL DELAY
```

Example 4-3

- Write the following programs

A. Create a square wave of 50% duty cycle on bit 0 of C

Solution 1

```
        BCF     TRISC,0
HERE
        BSF     PORTC,0
        CALL    DELAY
        BCF     PORTC,0
        CALL    DELAY
        BRA     HERE
```

How many byte are used?

Example 4-3

Solution 2

- Write the following programs

A. Create a square wave of 50% duty cycle on bit 0 of C

```
BCF   TRISC,0
BACK
    BTF     PORTC,0
    CALL    DELAY
    BRA     BACK
```

How many byte are used?

Example 4-4

- Write a program to perform the following:

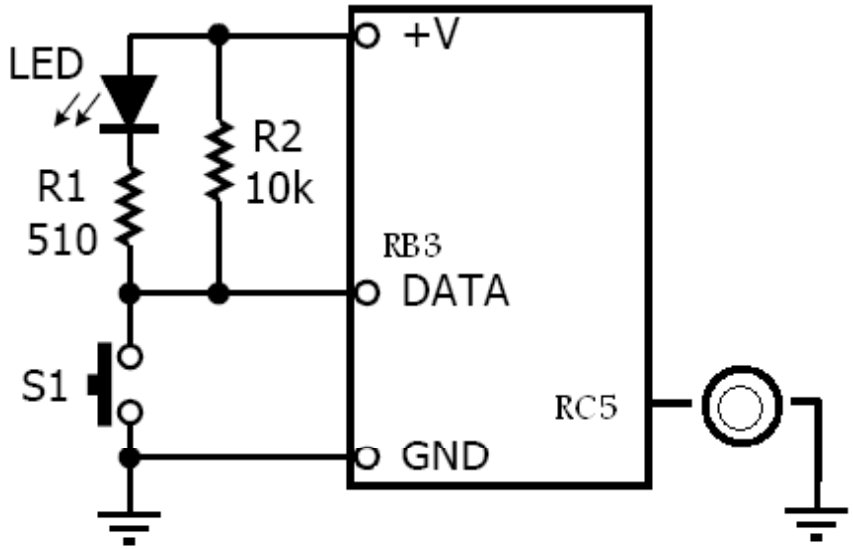
a) Keep monitoring the RB2 bit until it becomes HIGH (1)

b) When RB2 becomes HIGH, write value 45H to portC and send a HIGH to LOW plus to RD3

```
        BSF     TRISB,2
        CLRF    TRISC
        BCF     PORTD,3
        MOVLW   0x45
AGAIN
        BTFSS   PORTB,2
        BRA     AGAIN
        MOVWF   PORTC
        BSF     PORTD,3
        CALL    DELAY
        BCF     PORTD,3
```

Example 4-5

- Bit RB3 is an input and represents the condition of a door alarm.
- Whenever it goes LOW, send a HIGH-to-LOW pulse to RC5 to turn on a buzzer



Solution

```
        BSF     TRISB,3
        BCF     TRISC,5
HERE
        BTFSC   PORTB,3
        BRA     HERE
        BSF     PORTC,5
        BCF     PORTC,5
        CALL    DELAY
        BRA     HERE
```

Reading a single bit Examole4-8

- A switch is connected to pin RB0 and a LED to pin RB7. Write a program to read the status of SW and send it to the LED

```
BSF TRISB,0
BCF  TRISB,7
AGAIN
BTFSS PORTB,0
GOTO OVER
BSF PORTB,7
GOTO AGAIN
OVER
BCF PORTB,7
GOTO AGAIN
```

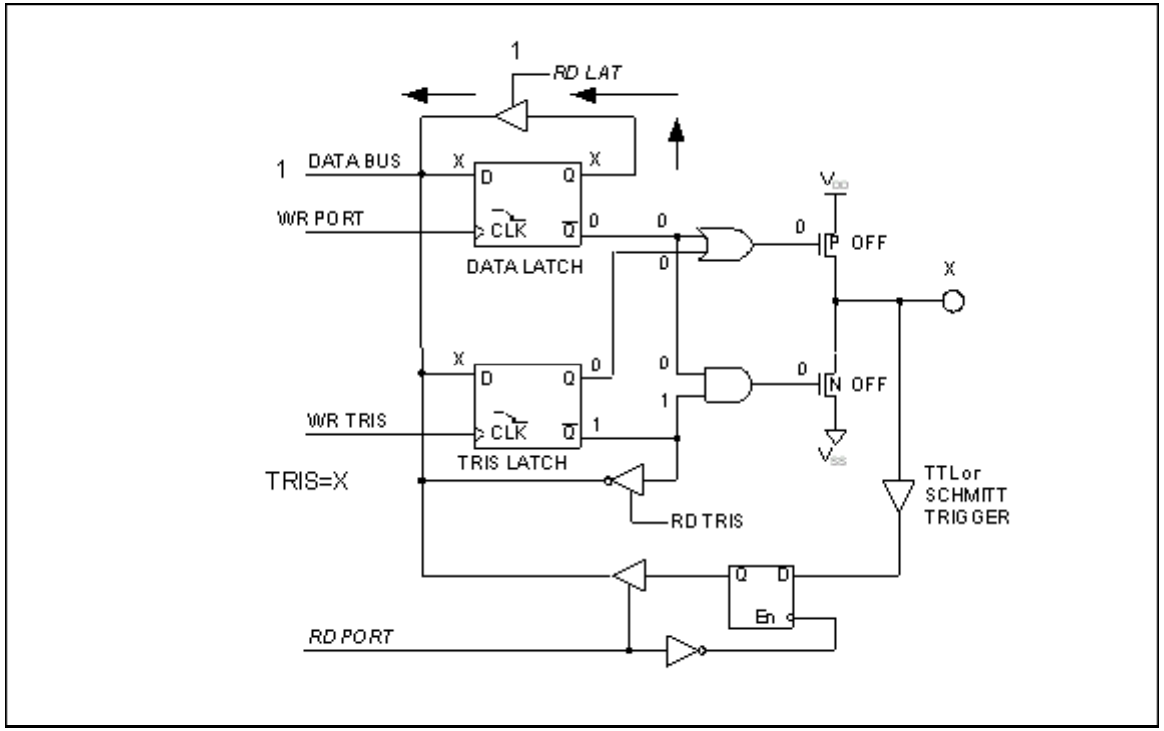
Reading input pins VS. LATx port

- There are two possibilites to read port's value
  - Through reading the status of the input pin
  - Through reading the internal latch of the LAT register.
    - Some instructions do that
    - The action is

    1. The instruction read the latch instead of the pin
    2. Execute the instruction
    3. Write back the result to the Latch
    4. The data on the pins are changed only if the TRISx bits are cleared.

Instruction		Function
ADDWF	fileReg,d	Add WREG from f
BSF	fileReg,bit	Bit Set fileReg
BCF	fileReg,bit	Bit Clear fileReg
COMF	fileReg,d	Complement f
INCF	fileReg,d	Increment F
SUBWF	fileReg,d	Subtruct WREG from f
XORWF	fileReg,d	Exclusive-OR WREG with f

Figure 4-8. LATx Register Role in Reading a Port or Latch



Chapter 4: Summary

- We focused on the I/O Ports of the PIC.
- These ports used for input or output. Programming
- We discussed Bit manipulation instructions

Next:

Arithmetic, logic Instruction and programs

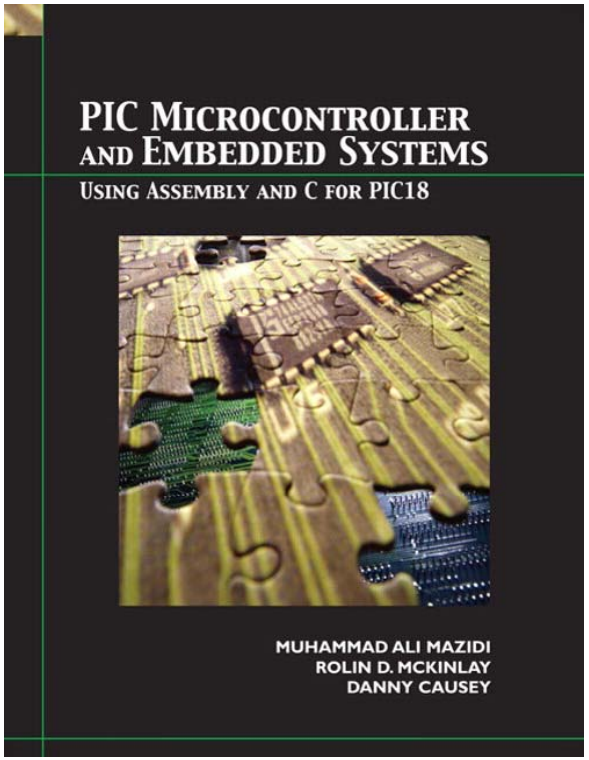
PIC Microcontroller and Embedded Systems

Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 5: Arithmetic, logic Instruction and programs



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

Objective

- Define the range of numbers possible in PIC unsigned data
- Code addition and subtraction instructions for unsigned data
- Perform addition of BCD
- Code PIC unsigned data multiplication instructions and programs for division
- Code PIC Assembly language logic instructions
- Code PIC rotate instructions

Outlines

- Arithmetic Instructions
- Signed Number Concepts and Arithmetic Operations
- Logic and Compare Instructions
- Rotate instruction and data serialization
- BCD and ASCII Conversion

Arithmetic Instructions

- Unsigned numbers are defined as data in which all the bits are used to represent data
  - no bits are set aside for neg. or pos. sign
- Addition of unsigned numbers
  - ADDLW k
  - ADDWF fileReg, d, a
  - ADDWFC (adding two 16-bit numbers)
- What happens to flag register?

Example 5-3

- Add

3CE7H and
3B8DH

Store the sum in fileReg locations 6 and 7, where location 6 should have the lower bye.

```
MOVLW 08DH
MOVWF 0x6
MOVLW 3BH
MOVWF 0x7
MOVLW 0xE7
ADDWF 0x6,F
MOVLW 0x3C
ADDWFC 0x7,F
```

WRFG

00

Address	Data
05H	00
06H	00
07H	00
08H	00
09H	00

BCD Number System

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- We use the digits 0 to 9 in everyd
- Binary Coded Decimal
  - Unpacked BCD
    - The lower 4 bits is just used
    - Requires 1 byte

0000 0010

  - Packed BCD
    - A single byte has two BCD numbers
    - Efficient in storing data

0101 0010

BCD

- What is the result if you add

- To correct the problem, we should add 6.

0x17

0x28

0x3F

0x45

DAW, Decimal Adjust WREG

- Works only with WREG
- Add 6 to the lower or higher nibble if needed
- After execution,
  - If the lower nibble is greater than 9, or if DC = 1, add 0110 to the lower nibble.
  - If the upper nibble is greater than 9, or if C = 1, add 0110 to the upper nibble.
- Doesn't require the use of arithmetic instructions prior the DAW execution

Subtraction of unsigned numbers

- Subtracter circuit is cumbersome. (Why?)
- PIC performs the 2's complement then uses adder circuit to the result.
- Take one Clock Cycle
- There are four sub instructions
  - SUBLW k (k – WREG)
  - SUBWF f d (destination = fileReg – WREG)
- Result may be negative (N=1 and C=1)
  - The result is left in 2's complement

Example 5-5

MOVLW 0x23

SUBLW 0x3F

	0011 1111
+	1101 1101
1	0001 1100

C =1, D7 = N = 0

Example 5-6

- Subtract 4C – 6E?

```
MYREG    EQU 0x20
         MOVLW 0x4C
         MOVWF MYREG
         MOVLW 0x6E
         SUBWF MYREG,W
         BNN   NEXT
         NEGF  WREG
NEXT
         MOVWF MYREG
```

	0100 1100
+	1001 0010
0	1101 1110

C =0, D7 = N = 1

0010 0010

Multiplication of unsigned number

- PIC supports byte-by-byte multiplication
- One of the operand must be in WREG
- After multiplication, the result is stored in PRODH and PRODL (16 bit)
- Example
  - MOVLW 0x25
  - MULLW 0x65

Special Function Registers

Address	SFR Name	Hex
FE6	POSTINC1	--
FE7	INDF1	--
FE8	WREG	0x25
FE9	FSR0	0x0000
FE9	FSR0L	0x00
FEA	FSR0H	0x00
FEB	PLUSW0	--
FEC	PREINC0	--
FED	POSTDEC0	--
FEE	POSTINC0	--
FEF	INDF0	--
FF0	INTCON3	0xC0
FF1	INTCON2	0xF5
FF2	INTCON	0x00
FF3	PROD	0x0E99
FF3	PRODL	0x99
FF4	PRODH	0x0E
FF5	TABLAT	0x00
FF6	TBLPTR	)x000000
FF6	TBLPTRL	0x00
FF7	TBLPTRH	0x00
FF8	TBLPTRU	0x00

Division of unsigned numbers

- There is no single instruction for the division of byte/byte numbers.
- You need to write a program
  - Repeated subtraction
  - The numerator is place in a fileReg
  - Denominator is subtracted from it repeatedly
  - The quotient is the number of times we subtracted
  - The reminder is in fileReg upon completion

Example 5-8

Convert the hexadecimal number FDH, stored in location 0x15, into decimal.

Save the digits in locations 0x22, 0x23 and 0x24

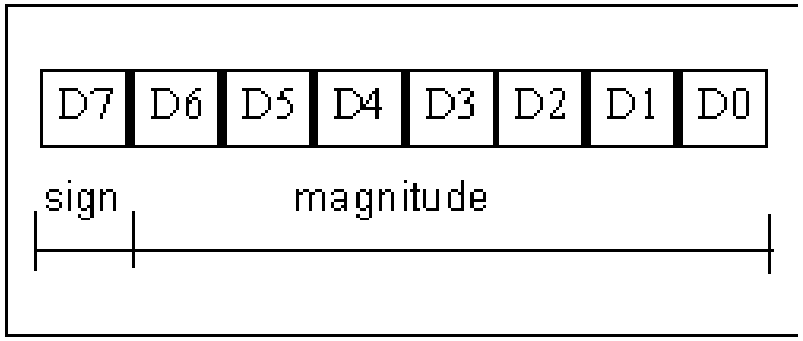
```
#include <P18F458.INC>
NUME  EQU     0x15
QU          EQU 0x20
RMND_L      EQU 0x22
RMND_M      EQU 0x23
RMND_H      EQU 0x24
MYNUM       EQU 0xFD
MYDEN       EQU D'10'
ORG 0H
MOVLW MYNUM
MOVWF NUME
MOVLW MYDEN
CLRF  QU,F
```

It is a Mistake in your book. There is no F

Example 5-8 (2)

```
D_1
   INCF  QU,F
   SUBWF NUME
   BC    D_1
   ADDWF NUME
   DECF  QU,F
MOVFF NUME,RMND_L
MOVFF QU,NUME
   CLRF  QU
D_2
      INCF  QU,F
      SUBWF NUME
      BC    D_2
      ADDWF NUME
      DECF  QU,F
      MOVFF
   NUME,RMND_M
      MOVFF
   QU,RMND_H
HERE
   GOTO HERE
   END
```

Signed Number Concepts and Arithmetic Operations



- The MSB is set aside for the sign (0 or -)
- The rest, 7 bits, are used for the magnitude.
- To convert any 7-bit positive number to negative use the 2's complement
- You have 128 negative numbers and 127 positive numbers

Overflow problem in Signed Number Operations

- An overflow occurs when the result of an operation is too large for the register
- OV flag indicate whether the result is valid or not.
  - If OV = 1, the result is erroneous
- When is the OV flag set?
  - The is a carry from D6 to D7 but no carry out of D7
  - There is a carry from D7 out (C = 1) but no carry from D6 to D7

Examples

	+96	0110 0000
+	+70	0100 0110
+	166	1010 0110 (N=1, OV=1 and sum=-90)

	-128	1000 0000
+	- 2	11 11 11 10
+	166	1 0111 1110 (N=0, OV=1 and sum=126)

Logic and Compare Instructions

- Widely used instructions
  - ANDLW k
  - ANDFW FileReg, d
  - IORLW k
  - IORFW FileReg, d
  - XORLW k
  - XORFW FileReg, d
- Effect only Z and N Flags

Complement Instructions

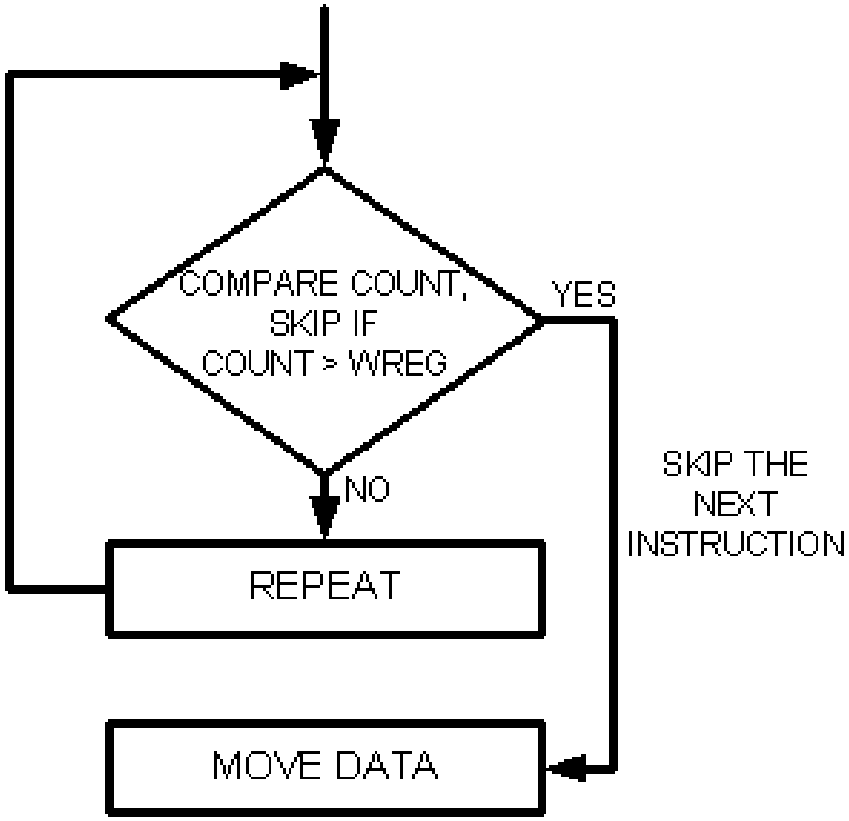
- COMF FileReg,d
  - Takes the 1's complement of a file register
  - Effect only Z and N Flags
- NEGF FileReg
  - Takes the 2's complement of a file register
  - Effect all Flags
- Example
  - MYREG EQU 0x10
  - MOVLW 0x85
  - MOVWF MYREG
  - NEGF MYREG

Compare Instructions

- These instructions take 1/2 cycle(s)

CPFSGT FileReg	**Compare FileReg with WREG, skip if greater than**	**FileREg > WREG**
CPFSEQ FileReg	**Compare FileReg with WREG, skip if equal**	**FileREg = WREG**
CPFSLT FileReg	**Compare FileReg with WREG, skip if less than**	**FileREg < WREG**

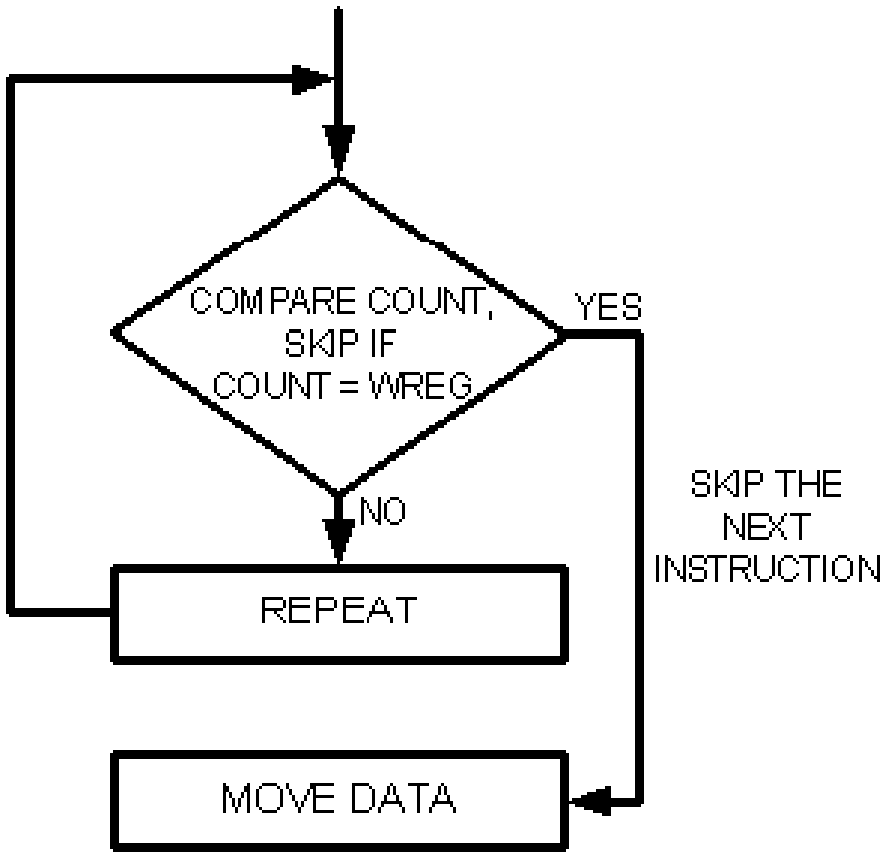
Figure 5-3. Flowchart for CPFSGT



INSTRUCTIONS

AGAIN	CPFSGT COUNT
	GOTO AGAIN
	MOVWF PORTB

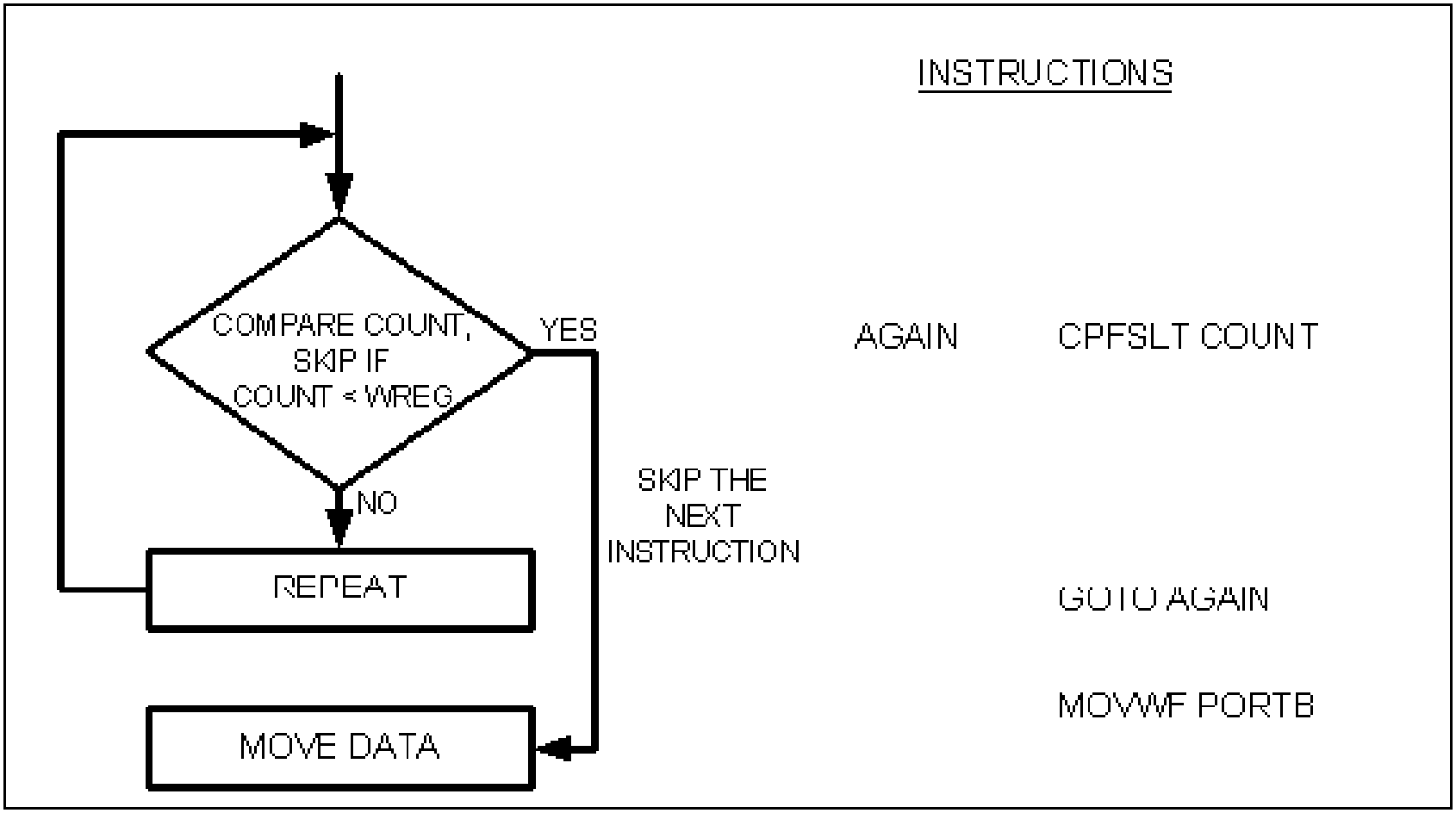
Figure 5-4. Flowchart for CPFSEQ



INSTRUCTIONS

AGAIN	CPFSEQ COUNT
	GOTO AGAIN
	MOVWF PORTB

Figure 5-5. Flowchart for CPFSLT



Example 5-27

- Write code to determine if data on PORTB contains the value 99H. If so, write letter 'y' to PORTC; otherwise, make PORTC='N'

```
        CLRF    TRISC
        MOVLW   A'N'
        MOVWF   PORTC
        SETF    TRISB
        MOVLW   0x99
        CPFSEQ  PORTB
        BRA     OVER
        MOVLW   A'Y'
        MOVWF   PORTC
OVER    ............
```

Rotate instruction and data serialization

- Rotate fileReg **R**ight or **L**eft (no Carry)
  - RRNCF fileRed, d
  - RLNCF fileRed, d
  - affect the N and Z flag
- Rotate **R**ight or **L**eft through Carry flag
  - RRCF fileRed, d
  - RLCF fileRed, d
  - affect the C, N and Z flag

Serializing data

- One of the most widely used applications of the rotate instructions.
  - Take less space on the PCB
- Sending a byte of data, one bit at a time through a single pin of uC.
  - Using the serial port.
  - Using a programming technique to transfer data one bit at a time and control the sequence of data and spaces between them.

Example 5-28

- Write a program to transfer value 41H serially via RB1.
- Put one High at the start and end
- Send LSB

- Solution

```
RCNT      EQU  0x20
MYREG     EQU  0x21
```

```
        BCF  TRISB,1
        MOVLW   0x41
        MOVWF   MYREG
        BCF     STATUS,C
        MOVLW   0x8
        MOVWF   RCNT
        BSF     PORTB,1
AGAIN   RRCF    MYREG,F
        BNC     OVER
        BSF     PORTB,1
        BRA     NEXT
OVER    BCF     PORTB,1
NEXT    DECF    RCNT,F
        BNZ     AGAIN
        BSF     PORTB,1
```

Example 5-29

- Write a program to bring in a byte of data serially via pin RC7 and save it in file register location 0x21
- The byte comes in with the LSB first

```
RCNT     EQU 0x20
MYREG    EQU 0x21
   BSF      TRISC,7
   MOVLW    0x8
   MOVWF    RCNT
AGAIN      BTFSC
         PORTC,7
   BSF      STATUS,C
   BTFSS    PORTC,7
   BCF      STATUS,C
   RRCF     MYREG,F
   DECF     RCNT,F
   BNZ      AGAIN
```

SWAPF

- Swap the lower nibble and the higher nibble

Before		After	
D7-D4	D3-D0	D3-D0	D7-D4

- In the absence of a SWAPF instruction, how would you exchange the nibbles? How many rotate instruction do you need?

BCD and ASCII Conversion

- What is ASCII?
  - What does Keyboard produce when you press any button?
- Real time clock, RTC, provide the time and date in BCD.

BCD and ASCII Codes for digits 0-9

Key	ASCII (hex)	Binary		BCD (unpacked)	
0	30	0011	0000	0000	0000
1	31	0011	0001	0000	0001
2	32	0011	0010	0000	0010
3	33	0011	0011	0000	0011
4	34	0011	0100	0000	0100
5	35	0011	0101	0000	0101
6	36	0011	0110	0000	0110
7	37	0011	0111	0000	0111
8	38	0011	1000	0000	1000
9	39	0011	1001	0000	1001

Packed BCD to ASCII Conversion

- RTC provides the date and the time in packed BCD
- Data must be in ASCII to be displayed on a LCD

packed BCD	***Unpacked BCD***	***ASCII***
29H	**02H & 09H**	**32H and 39H**
0010 1001	0011 0010	0011 0010
	0011 1001	0011 1001

ASCII to Packed BCD Conversion

- Get rid of the high nibble (3)

key	*ASCII*	*Unpacked BCD*	*packed BCD*
4	34	0000 0100	0100 0111 which is 47H
7	37	0000 0111	

Example 5-32

- Assume that register WREG has packed BCD. Write a program to convert packed BCD to two ASCII numbers and place them in in file register locations 6 and 7.

```
BCD_VAL   EQU   0x29
L_ASC     EQU   0x06
H_ASC     EQU   0x07
    MOVLW BCD_VAL
    ANDLW 0x0F
    IORLW 0x30
    MOVWF L_ASC
    MOVLW BCD_VAL
    ANDLW 0xF0
    SWAPF WREG,W
    IORLW 0x30
    MOVWF H_ASC
```

Chapter 5: Summary

- We discussed arithmetic instructions for both signed and unsigned data.
- We defined the logic and compare instructions.
- The rotate and swap instructions are widely used.
- We described BCD and ASCII formats and conversions.

Next:

Bank Switching, Table processing, Macros and Modules

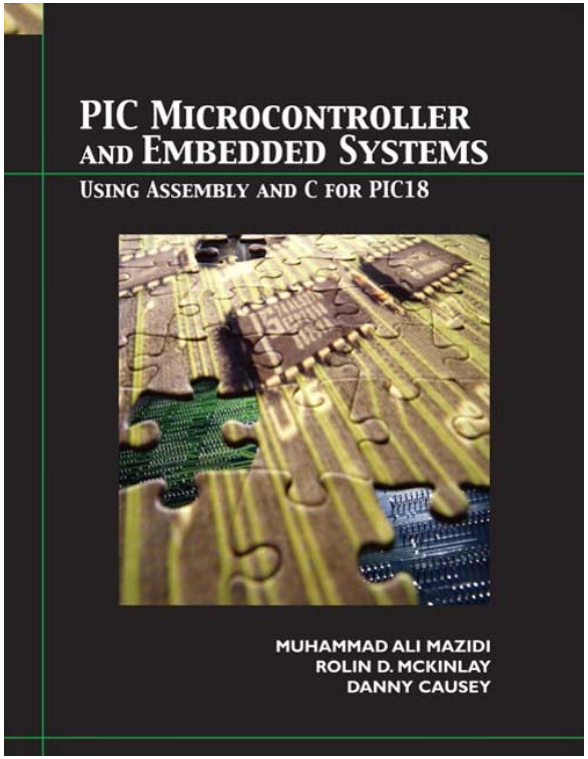
PIC Microcontroller and Embedded Systems

Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 6: Bank Switching, Table processing, Macros and Modules



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

Objective

- List all addressing modes of PIC18 uCs
- Contrast and compare the addressing modes
- Code PIC18 instructions to manipulate a lookup table.
- Access fixed data residing in ROM space.
- Discuss how to create macros and models, and its advantages.
- Discuss how to access the entire 4kB of RAM
- List address for all 16 banks of the PIC18
- Discuss bank switching for the PIC18

Outlines

- Immediate and Direct Addressing mode
- Register indirect Addressing mode
- Lookup table and table processing
- Bit addressability of data RAM
- Bank switching
- Checksum and ASCII subroutines
- Macros and models

Introduction

- Data could be in
  - A register
  - In memory
  - Provided as an immediate values
- PIC18 provides 4 addressing modes
  - Immediate
  - Direct
  - Register indirect
  - Indexed-ROM

Section 6.1: Immediate and Direct Addressing mode

- In immediate addressing mode, the operands comes after the opcode
  - MOVLW 0x25
  - SUBLW D'34'
  - ADDLW 0x86
- In direct addressing mode, the operand data is in a RAM location whose address is known and given as a part of the instruction.

Figure 6-1. MOVFF and MOVWF Direct Addressing Opcode

MOVLW 0X56

MOVWF 0X40

MOVFF 0X40,50H

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A	0B
000	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00
040	56	00	00	00	00	00	00	00	00	00	00	00
050	56	00	00	00	00	00	00	00	00	00	00	00
060	00	00	00	00	00	00	00	00	00	00	00	00

Program Memory

Address	00	02	04	06	08	0A	0C	0E
0000	0E56	6E40	C040	F050	8493	6A94	9683	0E45
0010	A481	D7FE	6E82	8683	9683	9094	6A82	8082
0020	EC23	F000	9082	EC23	F000	D7F9	6A93	6A94
0030	6A95	0E55	6E81	6E82	6E83	1E81	1E82	1E83
0040	EC23	F000	D7FA	0EC8	6E07	0EFA	6E08	0000
0050	0000	0608	E1FC	0607	E1F8	0012	FFFF	FFFF

Figure 6-1. MOVFF and MOVWF Direct Addressing Opcode

MOVLW 0X56

MOVWF 0X40

MOVFF 0X40,50H

0110	111A	ffff	ffff

0 ≤ ffff ffff ≤ FF

A – bank accessed for operation

A = 0, use default access bank

A = 1, use bank pointed to by BSR (Bank Selector Register)

Immediate and Direct Addressing mode

- What is the difference between
  - INCF fileReg, W
  - INCF fileReg, F
- What is the default destination?
- What is the difference between DECFSZ and DECF?
  - Operation
  - Branch

SFR Registers and their addresses

- Can be access by
  - Their name
  - Their address
- Which is easier to remember?

- MOVWF PORTB
- MOVWF 0xF81

Address	Symbol Name	Value
FE8	WREG	0x56
F80	PORTA	0x00
F81	PORTB	0x00
F83	PORTD	0x00
F82	PORTC	0x00
F89	LATA	0x00
F8A	LATB	0x00
F8B	LATC	0x00
F8C	LATD	0x00
F92	TRISA	0x7F
F93	TRISB	0xFF
F94	TRISC	0x00
F95	TRISD	0xFF
	INDF0	.ed Memory
	INDF1	.ed Memory
FE9	FSR0	0x0000
FE9	FSR0L	0x00
FEA	FSR0H	0x00
FE1	FSR1L	0x00
FE2	FSR1H	0x00

SFR Registers and their addresses

Remember

- SFR addresses is started at F80h and the last location has the address FFFh

Notes

- In .lst file, you will see that the SFR names are replaced with their addresses.
- The WREG register is one of the SFR registers and has address FE8h

Section 6.2: Register indirect Addressing mode

- A register is used as a pointer to the data RAM location.
- Three 12-bit Registers are used (from 0 to FFFh)
  - FSR0
  - FSR1
  - FSR2

FSR means file Select register

- Each register is associated with INDFx
- Syntax
  - **LFSR n,data** → LFSR 1,95Eh → needs 2 cycles

Advantages of Register indirect Addressing mode

- It makes accessing data dynamic
- Looping is possible to increment the address
  - Not possible in direct addressing mode
  - Example
    - **INCF FSR2L**

Example 6-2

- Write a program to copy the value 55H into RAM locations 40h to 45h using

A. Direct addressing mode
B. Register indirect addressing mode
C. A loop

Solution A

```
MOVLW   0x55
MOVWF   0x40
MOVWF   0x41
MOVWF   0x42
MOVWF   0x43
MOVWF   0x44
```

Example 6-2 (cont.)

Solution B

```
MOVLW   55H
LFSR    0,0x40
MOVWF   INDF0
INCF    FSR0L,F
MOVWF   INDF0
INCF    FSR0L,F
MOVWF   INDF0
INCF    FSR0L,F
MOVWF   INDF0
INCF    FSR0L,F
MOVWF   INDF0
```

Solution C

```
        COUNT   EQU 0x10
        MOVLW   0x5
        MOVWF   COUNT
        LFSR    0,0x40
        MOVLW   0x55
B1
        MOVWF   INDF0
        INCF    FSR0L,F
        DECF    COUNT,F
        BNZ     B1
```

Auto increment option for FSR

- Normal increment can cause problem since it increments 8-bit
  - INC FSR0L, F

FSR0H	FSR0L
03	FF

- Auto increment and auto decrement solve the problem
  - They doesn't affect the status flag

PIC18 auto increment/decrement of FSRn

Instruction	Function
CLRF INDFn	**After clearing fileReg pointed by FSRn, the FSRn stays the same**
CLRF POSTINCn	**After clearing fileReg pointed by FSRn, the FSRn is incremented (like x++)**
CLRF PREINCn	**The FSRn is incremented, then fileReg pointed to by FSRn is cleared (like ++x)**
CLRF POSTDECn	**After clearing fileReg pointed by FSRn, the FSRn is decremented (like x++)**
CLRF PLUSWn	**Clears fileReg pointed by FSRn + WREG, and FSRn W are unchanged**

Example 6-4

- Write a program to clear 16 RAM location starting at location 60H using Auto increment.
- Note: there are two identical mistakes in your book, pp 202. The right correction is FSR1=60H (not 40H)

Solution

```
COUNTREG EQU 0x10
CNTVAL EQU D'16'
    MOVLW    CNTVAL
    MOVWF    COUNTREG
    LFSR     1,0x60
B3
    CLRF     POSTINC1
    DECF     COUNTREG,F
    BNZ      B3
```

Example 6-5

- Write a program to copy a block of 5 bytes of data from location starting at 30H to RAM locations starting at 60H.

Solution

```
COUNTREG EQU 0x10
CNTVAL EQU D'5'
    MOVLW    CNTVAL
    MOVWF    COUNTREG
    LFSR     0, 0x30
    LFSR     1, 0x60
B3
    MOVF     POSTINC0,W
    MOVWF    POSTINC1
    DECF     COUNTREG,F
    BNZ      B3
```

Example 6-6

- Assume that RAM locations 40-43H have the following hex data. Write a program to add them together and place the result in locations 06 and 07.

Address	Data
040H	7D
041H	EB
042H	C5
043H	5B

Solution

```
COUNTREG EQU 0x20
L_BYTE EQU 0x06
H_BYTE EQU 0x07
CNTVAL EQU 4
MOVLW     CNTVAL
MOVWF     COUNTREG
LFSR  0,0x40
CLRF  WREG
CLRF  H_BYTE
B5    ADDWF POSTINC0, W
BNC   OVER
INCF  H_BYTE,F
OVER DECF     COUNTREG,F
BNZ   B5
MOVWF     L_BYTE
```

Example 6-7

- Write a program to add the following multi-byte BCD numbers and save the result at location 60H.

```
  12896577
+ 23647839
```

Address	Data
030H	77
031H	65
032H	89
033H	12
050H	39
051H	78
052H	64
053H	23

Solution

```
        COUNTREG EQU 0x20
        CNTVAL EQU D'4'
        MOVLW    CNTVAL
        MOVWF    COUNTREG
        LFSR     0,0x30
        LFSR     1,0x50
        LFSR     2,0x60
        BCF STATUS,C
B3      MOVF     POSTINC0,W
        ADDWFC POSTINC1,W
        DAW
        MOVWF    POSTINC2
        DECF     COUNTREG,F
        BNZ      B3
```

Section 6.3: Lookup table and table processing

- Beside instructions, ROM has enough space to store fixed data
- DB directive, which means **Define Byte,** is widely used to allocate ROM program memory in byte-sized chunks
- Use single quotes (´) for a single character or double quotes (") for a string
  - Org 0x500
  - DATA1 DB 0x39
  - DATA2 DB `z´
  - DATA3 DB "Hello All"
- ROM address must be even

Reading table elements in PIC18

- Program conter is 21-bit, which is used to point to any location in ROM space.
- How to fetch data from the code space?
  - Known as a **table processing**: register indirect ROM addressing mode.
  - There are table read and table write instructions

Reading table elements in PIC18

- To read the fixed data byte
  - We need an address pointer: TBLPTR
    - Points to data to be fetched
    - 21 bits as the program counter!!
    - Divided into 3 registers: TBLPTRL, TBLPTRH, TBLPTRU (all parts of SFR)
    - Is there any instruction to load 21 bits (as LFSR)?
  - A register to store the read byte
    - TBLLAT: keeps the data byte once it is fetched into the CPU

Auto increment option for TBLPTR

- Can you use the following instruction
  - INCF TBLPTRL, f
- Cause Problem
- **Example:** Assume that ROM space starting at 250H contains "Embedded System", write a program to send all characters to PORTB one byte at a time

TBLRD*	Table Read	After Read, TBLPRTR stays the sam
TBLRD*+	Table Read with Post-inc	Reads and inc. TBLPTR
TBLRD*-	Table Read with Post-dec	Reads and dec TBLPTR
TBLRD+*	Table Read with pret-inc	Increments TBLPTR and then reads

Example 6.10a

```
RCOUNT EQU 0x20
CNTVAL EQU 0x0F
ORG 0000H
MOVLW 0x50
MOVWF TBLPTRL
MOVLW 0x02
MOVWF TBLPTRH
MOVLW CNTVAL
MOVWF RCOUNT
CLRF TRISB
B6 TBLRD*
MOVFF
   TABLAT,PORTB
INCF TBLPTRL,F
DECF RCOUNT,F
BNZ B6
HERE GOTO HERE

ORG 0x250
MYDATA DB
   "Embedded System"
END
```

Program Memory

Address	00	02	04	06	08	0A	0C	0E	ASCII
01A0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
01B0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
01C0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
01D0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
01E0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
01F0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0200	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0210	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0220	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0230	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0240	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0250	6D45	6562	6464	6465	5320	7379	6574	006D	Embedded System.
0260	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0270	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0280	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0290	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....

Why don't you see the "Embedded System"

Program Memory

	Line	Address	Opcode	Label	Disassembly
	291	0244	FFFF		NOP
	292	0246	FFFF		NOP
	293	0248	FFFF		NOP
	294	024A	FFFF		NOP
	295	024C	FFFF		NOP
	296	024E	FFFF		NOP
B	297	0250	6D45	MYDATA	NEGF 0x45, BANKED
	298	0252	6562		CPFSGT 0x62, BANKED
	299	0254	6464		CPFSGT 0x64, ACCESS
	300	0256	6465		CPFSGT 0x65, ACCESS
	301	0258	5320		MOVF RCOUNT, F, BANKED
	302	025A	7379		BTG 0x79, 0x1, BANKED
	303	025C	6574		CPFSGT 0x74, BANKED
	304	025E	006D		
	305	0260	FFFF		NOP

Example 6.10b

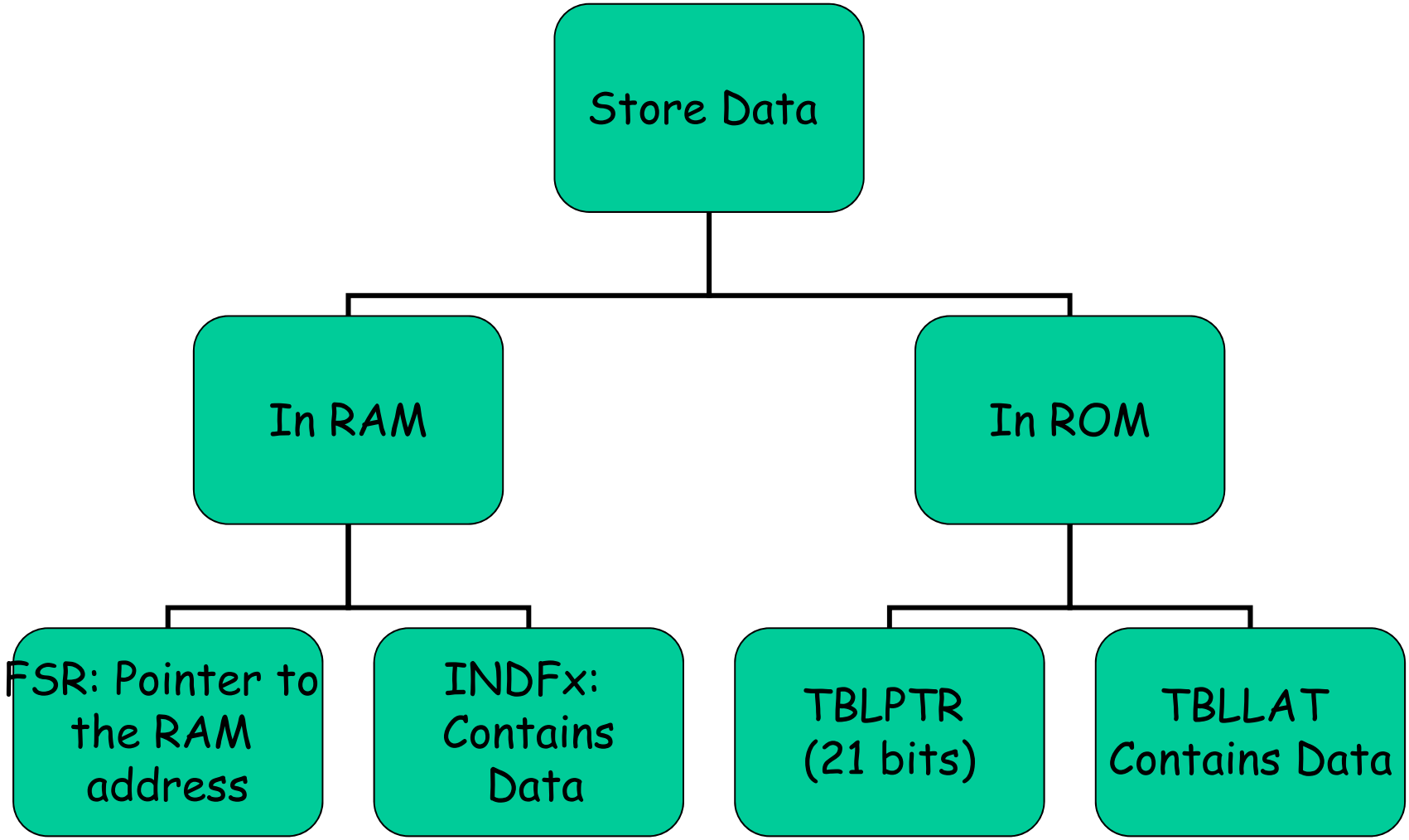
```
ORG 0000H
MOVLW 0x50
MOVWF TBLPTRL
MOVLW 0x02
MOVWF TBLPTRH
CLRF TRISB
B7 TBLRD*
MOVF TABLAT,W
BZ EXIT
MOVWF PORTB
INCF TBLPTRL,F
BRA B7
EXIT GOTO EXIT

ORG 0x250
MYDATA DB "
   Embedded System",0
END
```

Example 6.11: Auto increment

```
ORG 0000H
MOVLW 0x50
MOVWF TBLPTRL
MOVLW 0x02
MOVWF TBLPTRH
CLRF TRISB
B7 TBLRD*+
MOVF TABLAT,W
BZ EXIT
MOVWF PORTB
BRA B7
EXIT GOTO EXIT
```

```
ORG 0x250
MYDATA DB "
   Embedded System",0
END
```

Summery


Store Data
In RAM
In ROM
FSR: Pointer to the RAM address
INDFx: Contains Data
TBLPTR (21 bits)
TBLLAT Contains Data

Look-Up table and RETLW

- Used to access elements of a frequently used with minimum operations
- Example: x^2
- We can use a look-up table instead of calculating the values **WHY?**
- We need to a fixed value to the PCL to index into the look-up table
- RETLW (Return Literal to W) will provide the desired look-up table element in WREG

Example 6-14

- Write a program to get the x valuefrom PORT B and send x^2 to port C.
- Use look-up table instead of a mutliply instruction.
- Use PB3-PB0

```
ORG 0
SETF TRISB
CLRF TRISC
B1 MOVF PORTB,W
ANDLW 0x0F
CALL XSQR_TABLE
MOVWF PORTC
BRA B1
```

```
XSQR_TABLE
MULLW 0x2
MOVFF PRODL, WREG
ADDWF PCL
RETLW D'0'
RETLW D'1'
RETLW D'4'
RETLW D'9'
RETLW D'16'
RETLW D'25'
RETLW D'36'
 RETLW D'49'
RETLW D'64'
RETLW D'81'
END
```

Example 6-14

Program Memory

Address	00	02	04	06	08	0A	0C	0E	ASCII
0000	6893	6A94	5081	0B0F	EC08	F000	6E82	D7FA	.h.j.P.. ....
0010	0D02	CFF3	FFE8	26F9	0C00	0C01	0C04	0C09	.....& ....
0020	0C10	0C19	0C24	0C31	0C40	0C51	FFFF	FFFF	....$.1. @.Q.
0030	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0040	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0050	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0060	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0070	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0080	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
0090	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
00A0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....

Accessing a look-up table in RAM

- Store data in a continue location
- Using FSR as a pointer and the working register as an index
- For example:
  - MOVFF PLUS2 , PortD
  - Will copy data from location pointed by FSR2+WREG into PortD

Example 6-15: X^2

```
ORG 0
MOVLW 0
MOVWF 40H
MOVLW 1
MOVWF 41H
MOVLW 4
MOVWF 42H
MOVLW .9
MOVWF 43H
MOVLW .16
SETF      TRISC
CLRF TRISD
LFSR 2,0x40

B1    MOVF      PORTC,W
   ANDLW B'00000111'
   MOVFF  PLUSW2,PORTD
   BRA      B1
   END
```

Example 6-15

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
040	00	01	04	09	10	19	24	31	40	51	00	00	00	00	00	00	.....$1 @Q.....
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Hex Symbolic

Example 6-15

Program Memory

Line	Address	Opcode	Disassembly
13	0018	0E24	MOVLW 0x24
14	001A	6E46	MOVWF 0x46, ACCESS
15	001C	0E31	MOVLW 0x31
16	001E	6E47	MOVWF 0x47, ACCESS
17	0020	0E40	MOVLW 0x40
18	0022	6E48	MOVWF 0x48, ACCESS
19	0024	0E51	MOVLW 0x51
20	0026	6E49	MOVWF 0x49, ACCESS
21	0028	6894	SETF 0xf94, ACCESS
22	002A	6A95	CLRF 0xf95, ACCESS
23	002C	EE20	LFSR 0x2, 0x40
24	002E	F040	NOP
25	0030	5082	MOVF 0xf82, W, ACCESS
26	0032	0B07	ANDLW 0x7
27	0034	CFDB	MOVFF 0xfdb, 0xf83
28	0036	FF83	NOP
29	0038	D7FB	BRA 0x30
30	003A	FFFF	NOP
31	003C	FFFF	NOP
32	003E	FFFF	NOP
33	0040	FFFF	NOP

Opcode Hex | Machine | Symbolic

Example 6-16

- Write a program to get the x valuefrom PORT B and send x2 +2x +3 to port C.
- Use look-up table instead of a mutliply instruction.
- Use PB3-PB0

```
ORG 0
SETF TRISB
CLRF TRISC
B1 MOVF PORTB,W
ANDLW 0x0F
CALL XSQR_TABLE
MOVWF PORTC
BRA B1
```

```
XSQR_TABLE
MULLW 0x2
MOVFF PRODL, WREG
ADDWF PCL
RETLW D'3'
RETLW D'6'
RETLW D'11'
RETLW D'18'
RETLW D'27'
RETLW D'38'
RETLW D'51,
 RETLW D'66'
RETLW D'83'
RETLW D'102'
END
```

Section 6.4: bit addressability of data RAM

- One of the basic feathers of the PIC18 is the bit addressability of RAM.
  - Bit-addressable instructions
    - Use only direct addressing mode
  - Byte-addressable instructions

Status Register Bit-addressability

- You can access any bit of the status register by their name.
- Examples

  BCF STATUS,C

  BTFSS STATUS, Z

D7							D0
X	X	X	N	OV	Z	DC	C

C – Carry flag

OV – Overflow flag

DC – Digital Carry flag

N – Negative flag

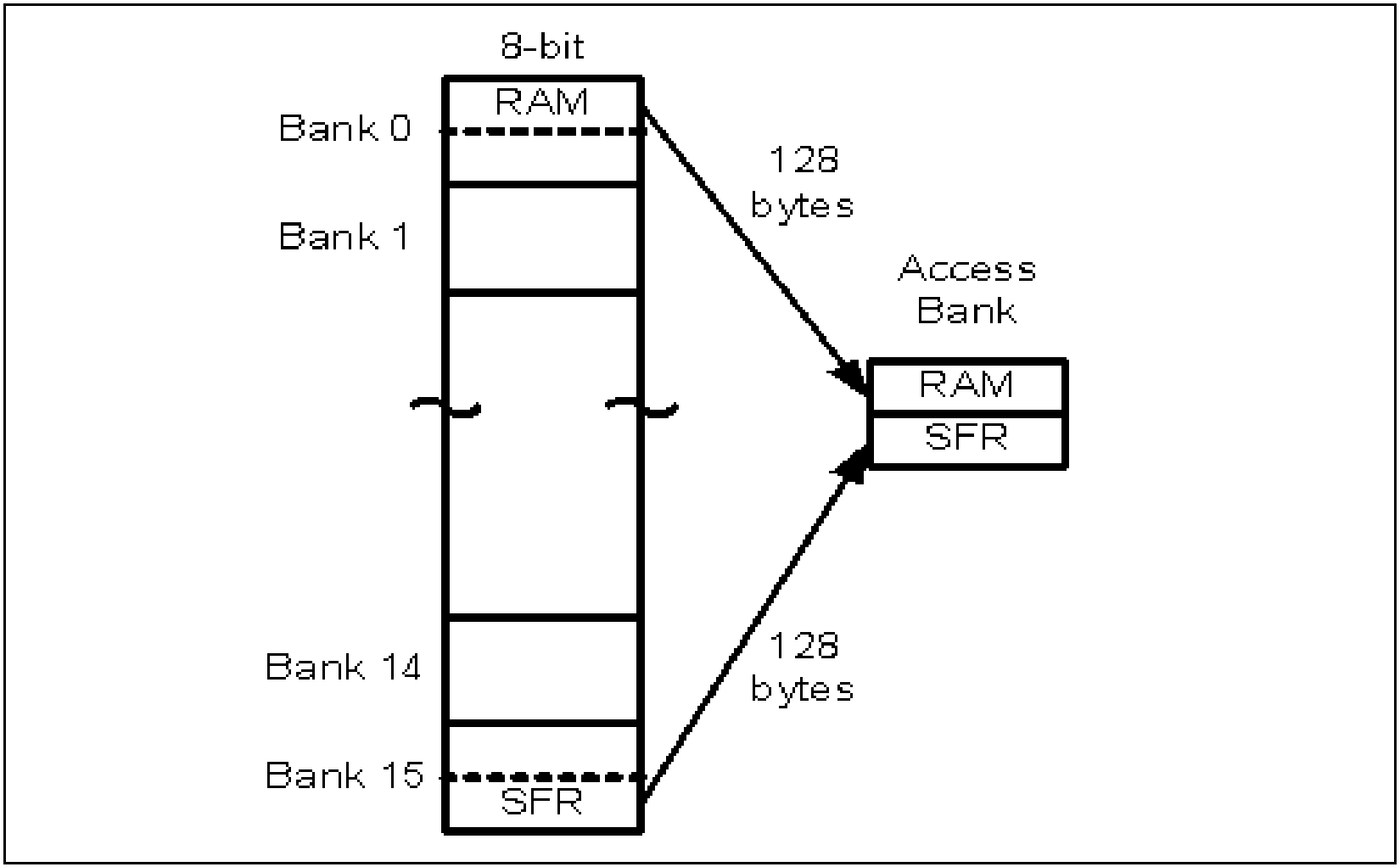
Z – Zero flag

X – D5, D6, and D7 are not implemented, and reserved for future use.

Section 6.5: Bank switching in the PIC18

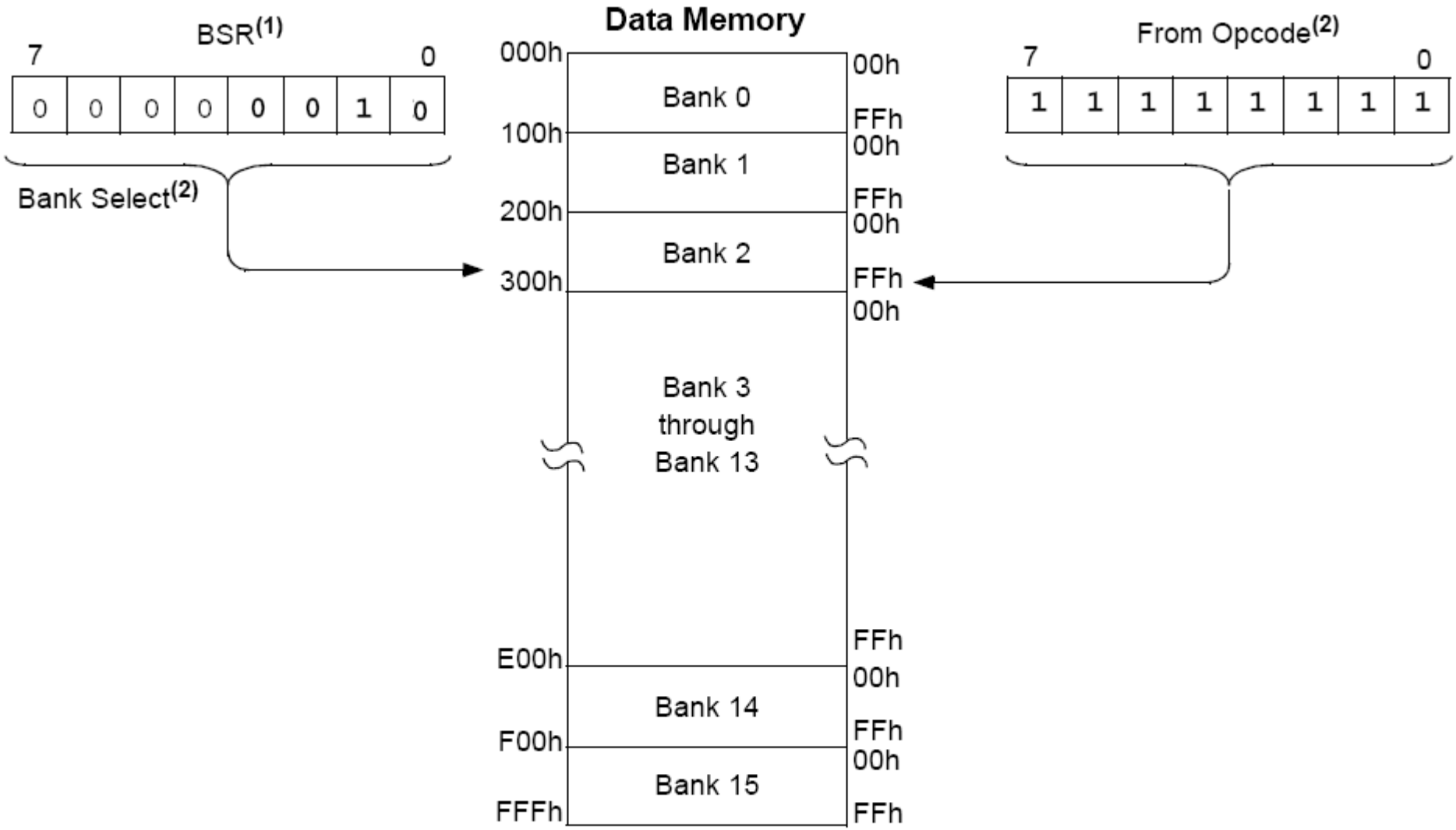
- PIC18 has maximum of 4K of RAM
  - Not all the space used.
  - The fileReg is divided into 16 banks of 256B each
  - Every PIC18 has the access bank (the first 128B of RAM + SFR)
  - Most PIC18 that access the data space in RAM has the ability to access any bank through setting an optional operand, called A
  - Example: **MOVWF myReg, A**
    - If 0 it access the default bank (default)
    - If 1, it uses the bank selection register (BSR) to select the bank

Figure 6-3. Data RAM Registers



The BSR register and bank switching

- It is 8-bit register
  - 4 bits are used → 16 banks
  - Banks 0 (from 00 to FF)
  - Banks 1 (from 100 to 1FF)
  - Banks 2 (from 200 to 2FF)
  - .....
  - Banks F (from F00 to FFF) (includes SFR)
- Upon power-on reset, BSR is equal to 0 (default value)



Note 1: The Access RAM bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

2: The `MOVFF` instruction embeds the entire 12-bit address in the instruction.

A Bit in the Instruction Field for INCF F, D, A

- Two things must be done
1. Load BSR with desired bank
2. Make A = 1 in the instruction itself.

```
MYREG EQU 0x40
MOVLB 0x2
MOVLW 0
MOVWF MYREG,1
INCF MYREG, F, 1
INCF MYREG, F, 1
INCF MYREG, F, 1
```

0010	10DA	ffff	ffff

D – destination for operation

A – bank accessed for operation

$0 \leq f \leq FF$

D = F, destination is fileReg
D = W, destination is WREG

A = 0, use default access bank
A = 1, use bank pointed to by BSR (Bank Selector Register)

Example 6-25

- Write a program to copy the value 55H into RAM locations 340h to 345h using

A. Direct addressing mode

B. A loop

Solution (A)

```
MOVLB  0x3
MOVLW  0x55
MOVWF  0x40, 1
MOVWF  0x41, 1
MOVWF  0x42, 1
MOVWF  0x43, 1
MOVWF  0x44, 1
MOVWF  0x44, 1
```

Example 6-25

- Write a program to copy the value 55H into RAM locations 340h to 345h using
  A. Direct addressing mode
  B. A loop

Solution (B)

```
COUNT EQU 0x10
MOVLB 0x3
MOVLW 0x6
MOVWF COUNT
LFSR  0,0x340
MOVLW 0x55
B1
MOVWF INDF0,0
INCF  FSR0L
DECF  COUNT,F,0
BNZ   B1
```

Mistake in your Textbook

Section 6.6: Checksum and ASCII subroutines

- To ensure the integrity of ROM contents, every system must perform a checksum calculation.
  - Corruption (caused by current surge)
- To calculate the checksum byte
  1. Add the bytes and drop the carries
  2. Take the 2's complement of the total sum
- To perform the checksum operation
  1. Add all bytes, including the checksum byte
  2. The result must be zero, else error

Example 6-29

- Find the checksum byte

```
  25H
+ 62H
+ 3FH
+ 52H
-------
 118H      (Drop the carry bit)
```

The 2's comp. is E8

- Perform the checksum

```
  25H
+ 62H
+ 3FH
+ 52H
+ E8H
-------
 200 (Drop the carry)
```

Example 6-29

- If the second byte 62H has been changed into 22H. Show how the checksum method detects the error.

```
  25H
+ 22H
+ 3FH
+ 52H
+ E8H
 1C0H      (Drop the carry bit)
```

Section 6.6: Checksum

Calculating and Testing Checksum byte

```
AM_ADDR   EQU 40H
COUNTREG  EQU 0x20
CNTVAL    EQU 4
CNTVAL1   EQU 5
    ORG       0
    CALL      COPY_DATA
    CALL      CAL_CHKSUM
    CALL      TEST_CHKSUM
    BRA       $
```

```
ORG 0x500
MYBYTE DB 0x25,
    0x62, 0x3F, 0x52,
    0x00
END
```

Mistake in your Textbook

```
COPY_DATA
    MOVLW   low(MYBYTE)
    MOVWF   TBLPTRL
    MOVLW   high(MYBYTE)
    MOVWF   TBLPTRH
    MOVLW upper(MYBYTE)
    MOVWF   TBLPRTRU
    LFSR    0, RAM_ADDR
C1      TBLRD*+
    MOVF    TABLAT,W
    BZ  EXIT
    MOVWF   POSTINC0
    BRA     C1
EXIT RETURN
```

```
CAL_CHKSUM
    MOVLW     CNTVAL
    MOVWF     COUNTREG
    LFSR      0,RAM_ADDR
    CLRF      WREG
C2  ADDWF     POSTINC0,W
    DECF      COUNTREG,F
    BNZ       C2
    XORLW     0xFF
    ADDLW     1
    MOVWF     POSTINC0
    RETURN
```

```
TEST_CHKSUM
    MOVLW     CNTVAL1
    MOVWF     COUNTREG
    CLRF      TRISB
    LFSR      0,RAM_ADDR
    CLRF      WREG
C3  ADDWF     POSTINC0,W
    DECF      COUNTREG,F
    BNZ       C3
    XORLW     0x0
    BZ   G_1
    MOVLW     'B'
    MOVWF     PORTB
    RETURN
G_1     MOVLW         'G'
    MOVWF     PORTB
    RETURN
```

Section 6.7: Macros and models

- Dividing a program into several models allows us to use models in other application.
  - Reduce time
  - Reduce of errors
- Increase the code size every time it invoked
- MACRO Syntax

Name MACRO dummy1, dummy2 …

……

……

ENDM

Unique

Body

Example:

- Write a delay macro and a MOVLF macro.

```
#include P18F458.INC
NOEXPAND
DELAY_1 MACRO V1,
    TREG
    LOCAL   BACK
    MOVLW V1
    MOVWF TREG
BACK        NOP
    NOP
    NOP
    NOP
    DECF    TREG,F
    BNZ     BACK
    ENDM
```

Local decleration

Example, cont.

```
MOVLF MACRO K, MYREG
    MOVLW K
    MOVWF MYREG
    ENDM

ORG 0
    CLRF      TRISB
OVER      MOVLF
    0x55,PORTB
    DELAY_1 0x200,0x10
    MOVLF   0xAA,PORTB
    DELAY_1 0x200,0x10
    BRA       OVER
END
```

Figure 6-11. List File with NOEXPAND Option for Program 6-4

```
                 00001 ;Program 6-4:toggling Port B using macros
                 00002        #include P18F458.INC
                 00003        NOEXPAND
                 00004 ;---------------sending data to fileReg macro
                 00005 MOVLF MACRO K, MYREG
                 00006        MOVLW    K
                 00007        MOVWF    MYREG
                 00008        ENDM
                 00009
                 00010 ;----------------------------time delay macro
                 00011 DELAY_1 MACRO V1, TREG
                 00012        LOCAL       BACK
                 00013        MOVLW       V1
                 00014        MOVWF       TREG
                 00015 BACK   NOP
                 00016        NOP
                 00017        NOP
                 00018        NOP
                 00019        DECF        TREG,F
                 00020        BNZ         BACK
                 00021        ENDM
                 00022
                 00023 ;----------------------------program starts
000000           00024        ORG         0
000000 6A93      00025        CLRF        TRISB       ;Port B as an output
                 00026 OVER   MOVLF       0x55,PORTB
                 00027        DELAY_1     0x200,0x10
                 00028        MOVLF       0xAA,PORTB
                 00029        DELAY_1     0x200,0x10
00002A D7EB      00030        BRA         OVER
                 00031        END
```

Figure 6-12. List File with EXPAND Option for Program 6-4

```
00001 ;Program 6-4:toggling Port B using macros
00002        #include P18F458.INC
00003        EXPAND
00004 ;---------------sending data to fileReg macro
00005 MOVLF MACRO K, MYREG
00006        MOVLW K
00007        MOVWF MYREG
00008        ENDM
00009
00010 ;-----------------------------time delay macro
00011 DELAY_1 MACRO V1, TREG
00012        LOCAL BACK
00013        MOVLW V1
00014        MOVWF TREG
00015 BACK   NOP
00016        NOP
00017        NOP
00018        NOP
00019        DECF   TREG,F
00020        BNZ    BACK
00021        ENDM
00022
```

Figure 6-12. List File with EXPAND Option for Program 6-4 (cont.)

```
                00023 ;----------------------------program starts
000000          00024        ORG   0
000000  6A93    00025        CLRF  TRISB ;Port B as an output
                00026 OVER   MOVLF 0x55,PORTB
000002  0E55       M         MOVLW 0x55
000004  6E81       M         MOVWF PORTB
                00027 DELAY_1 0x200,0x10
  0000             M         LOCAL BACK
000006  0E00       M         MOVLW 0x200
000008  6E10       M         MOVWF 0x10
00000A  0000       M  BACK   NOP
00000C  0000       M         NOP
00000E  0000       M         NOP
000010  0000       M         NOP
000012  0610       M         DECF  0x10,F
000014  E1FA       M         BNZ   BACK
                00028        MOVLF 0xAA,PORTB
000016  0EAA       M         MOVLW 0xAA
000018  6E81       M         MOVWF PORTB
                00029 DELAY_1 0x200,0x10
  0000             M         LOCAL BACK
00001A  0E00       M         MOVLW 0x200
00001C  6E10       M         MOVWF 0x10
00001E  0000       M  BACK   NOP
000020  0000       M         NOP
000022  0000       M         NOP
000024  0000       M         NOP
000026  0610       M         DECF  0x10,F
000028  E1FA       M         BNZ   BACK
00002A  D7EB    00030        BRA   OVER
                00031        END
```

Chapter 6: Summary

Next: Chapter 9

Arithmetic, logic Instruction and programs

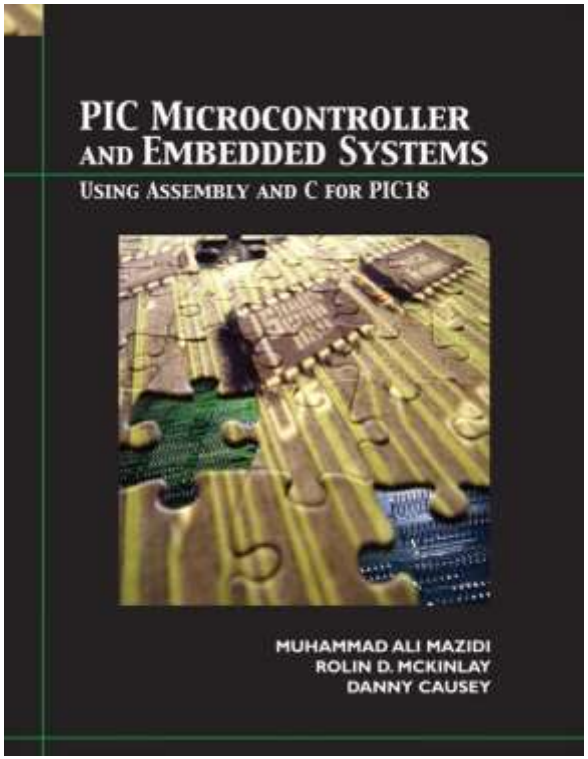
PIC Microcontroller and Embedded Systems

Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 9: PIC18 Timer Programming in Assembly and C



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

Objective

- List the Timers of PIC18 and their associated registers
- Describe the various modes of the PIC18 timers
- Program the PIC18 timers in Assembly to generate time delays
- Program the PIC18 timers in Assembly as event counters

Outlines

- Programming timers 0 and 1
- Counter Programming

Introduction

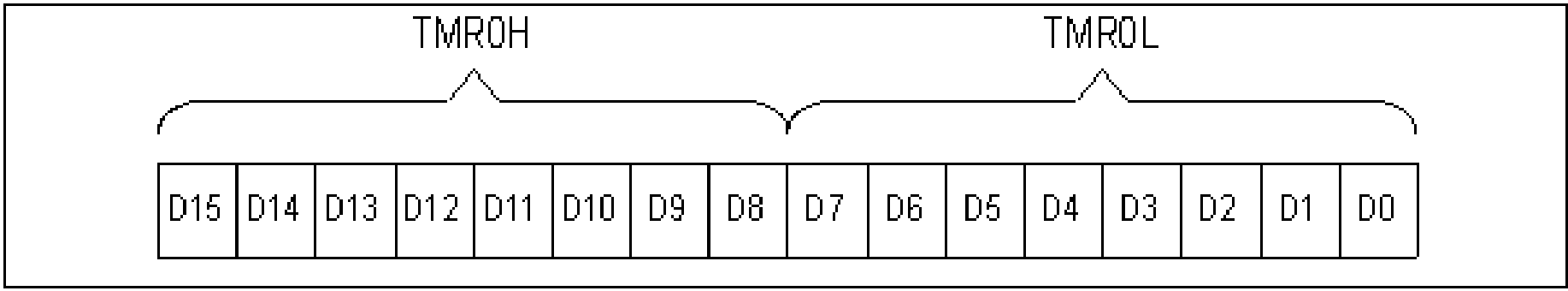
- PIC18 has two to five timers
  - Depending on the family number
- These timers can be used as
  - Timers to generate a time delay
  - Counters to count events happening outside the uC

Section 9.1: Programming timers 0 and 1

- Every timer needs a clock pulse to tick
- Clock source can be
  - **Internal** → 1/4th of the frequency of the crystal oscillator on OSC1 and OSC2 pins (Fosc/4) is fed into timer
  - **External:** pulses are fed through one of the PIC18's pins → Counter
- Timers are 16-bit wide
  - Can be accessed as two separate reg. (TMRxL & TMRxH)
  - Each timer has TCON (timer Control) reg.

Timer0 registers and programming

- TMR0L & TMR0H are 8-bit Reg.
  - MOVWF TMR0L
  - MOVFF TMR0L, PORTB



T0CON Reg

- Determine the timer operations modes
- **Example**
- If T0CON= 0000 1000
  - 16-bit
  - No prescaler
  - Rising edge

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS

TMR0ON	D7	Timer0 ON and OFF control bit 1 = Enable (start) Timer0 0 = Stop Timer0
T08BIT	D6	Timer0 8-bit/16-bit selector bit 1 = Timer0 is configured as an 8-bit timer/counter. 0 = Timer0 is configured as a 16-bit timer/counter.
T0CS	D5	Timer0 clock source select bit 1 = External clock from RA4/T0CKI pin 0 = Internal clock (Fosc/4 from XTAL oscillator)
T0SE	D4	Timer0 source edge select bit 1 = Increment on H-to-L transition on T0CKI pin 0 = Increment on L-to-H transition on T0CKI pin
PSA	D3	Timer0 prescaler assignment bit 1 = Timer0 clock input bypasses prescaler. 0 = Timer0 clock input comes from prescaler output.
T0PS2:T0PS0	D2 D1 D0	Timer0 prescaler selector
	0 0 0 = 1:2	Prescale value (Fosc / 4 / 2)
	0 0 1 = 1:4	Prescale value (Fosc / 4 / 4)
	0 1 0 = 1:8	Prescale value (Fosc / 4 / 8)
	0 1 1 = 1:16	Prescale value (Fosc / 4 / 16)
	1 0 0 = 1:32	Prescale value (Fosc / 4 / 32)
	1 0 1 = 1:64	Prescale value (Fosc / 4 / 64)
	1 1 0 = 1:128	Prescale value (Fosc / 4 / 128)
	1 1 1 = 1:256	Prescale value (Fosc / 4 / 256)

Figure 9-2. T0CON (Timer0 Control) Register

TMR0IF flag bit

- Part of INTCON

					TMR0IF		

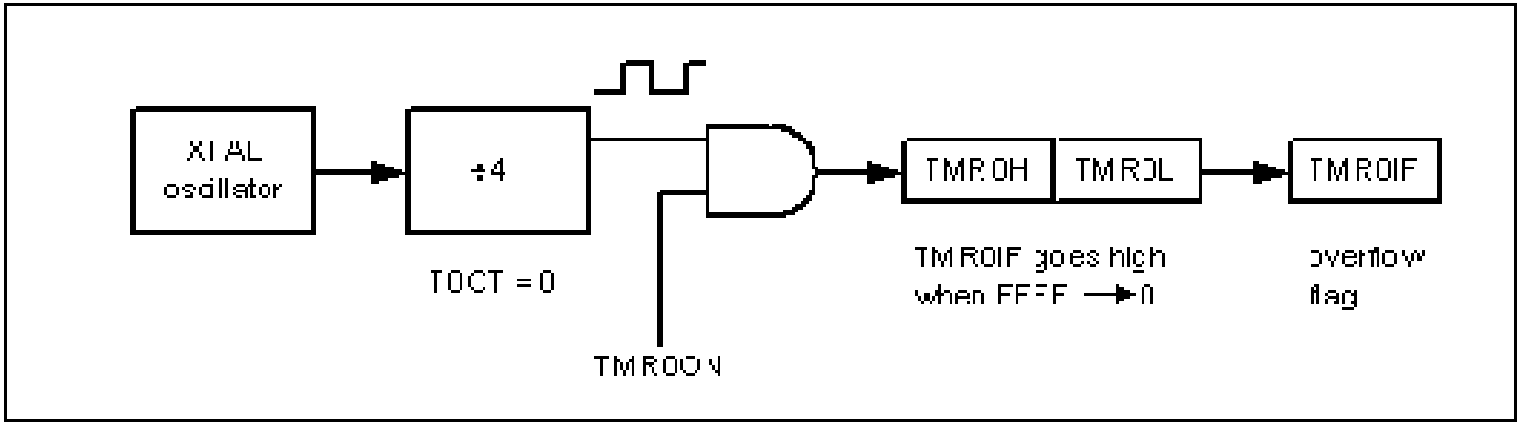
TMR0IF D2 Timer0 interrupt overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (FFFF to 0000, or FF to 00 in 8-bit mode).

The importance of TMR0IF: In 16-bit mode, when TMR0H:TMR0L overflows from FFFF to 0000 this flag is raised. In 8-bit, it is raised when the timer goes from FF to 00. We monitor this flag bit before we reload the TMR0H:TMR0L registers.

The other bits of this register are discussed in Chapter 11.

Figure 9-3. INTCON (Interrupt Control Register) has the TMR0IF Flag

Figure 9-4. Timer0 Overflow Flag



Characteristics and operations of 16-bit mode

1. 16-bit timer, 0000 to FFFFH.
2. After loading TMR0H and TMR0L, the timer must be started.
3. Count up, till it reaches FFFFH, then it rolls over to 0000 and activate TMR0IF bit.
4. Then TMR0H and TMR0L must be reloaded with the original value and deactivate TMR0IF bit.

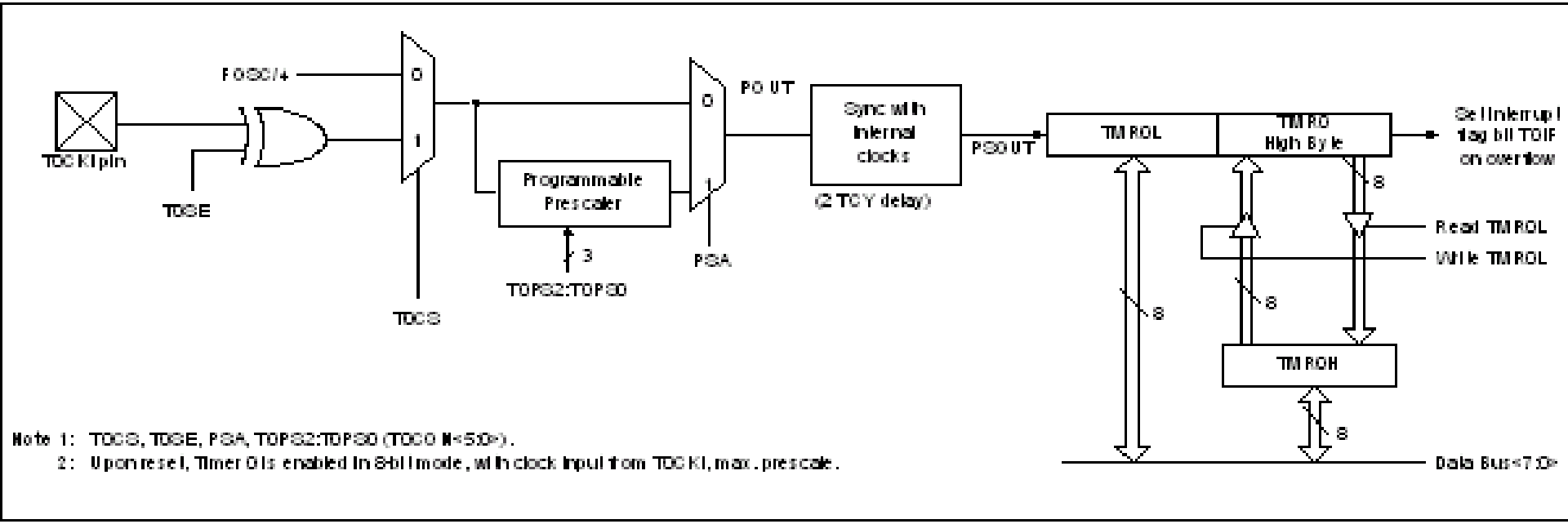
Steps to program Timer0 in 16-bit mode to generate time delay

1. Load the value into the T0CON register
2. Load reg. TMR0H followed by reg. TMR0L with initial value
3. Start the timer with instruction

   BSF T0CON, TMR0ON

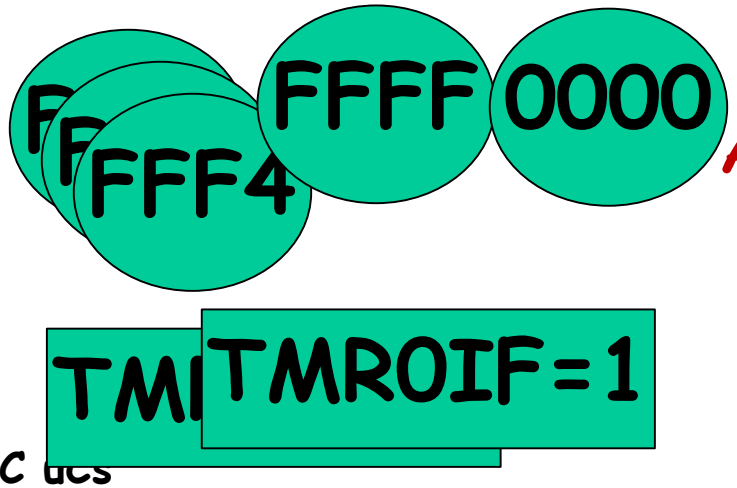
4. Keep monitoring the timer flag (TMR0IF) to see if it is raised.
5. Stop the timer
6. Clear the TMR0IF flag 3
7. Go Back to step 2

Figure 9-5. Timer0 16-bit Block Diagram



Example 9-3

- A square wave of 50% duty cycle on the PORTB.5 is created
- Analyze the program




```
        BCF     TRISB,5
        MOVLW   0x08
        MOVWF   T0CON
HERE
        MOVLW   0xFF
        MOVWF   TMR0H
        MOVLW   0xF2
        MOVWF   TMR0L
        BCF     INTCON, TMR0IF
        BTG     PORTB,5
        BSF     T0CON, TMR0ON
AGAIN
        BTFSS   INTCON, TMR0IF
        BRA     AGAIN
        BCF     T0CON, TMR0ON
        BRA     HERE
```

Example 9-5

- Calculate the frequency of the wave generated on PIN PORTB 5.

```
BCF     TRISB,5
MOVLW   0x08
MOVWF   T0CON
BCF     INTCON,
        TMR0IF
```

```
HERE
        MOVLW   0xFF            1
        MOVWF   TMR0H           1
        MOVLW   -D'48'          1
        MOVWF   TMR0L           1
        CALL    DELAY           2
        BTG     PORTB,5         1
        BRA     HERE            2
DELAY
        BSF     T0CON, TMR0ON   1
AGAIN
        BTFSS   INTCON, TMR0IF  48
        BRA     AGAIN
        BCF     T0CON, TMR0ON   1
        BCF     INTCON, TMR0IF  1
        RETURN                  2
```

Figure 9-6. Timer Delay Calculation for XTAL = 10 MHz with No Prescaler

- General formula for delay calculation
  - T = 4/(10MHz) = 0.4 usecond

(a) in hex	(b) in decimal
(FFFF - YYXX + 1) × 0.4 μs where YYXX are the TMR0H, TMR0L initial values respectively. Notice that YYXX values are in hex.	Convert YYXX values of the TMR0H, TMR0L register to decimal to get a NNNNN decimal number, then (65536 - NNNNN) × 0.4 μs

Example 9-8

- Write a program to generate a square wave with a period of ms on pin PORTB.3 (XALT=10 Mhz)

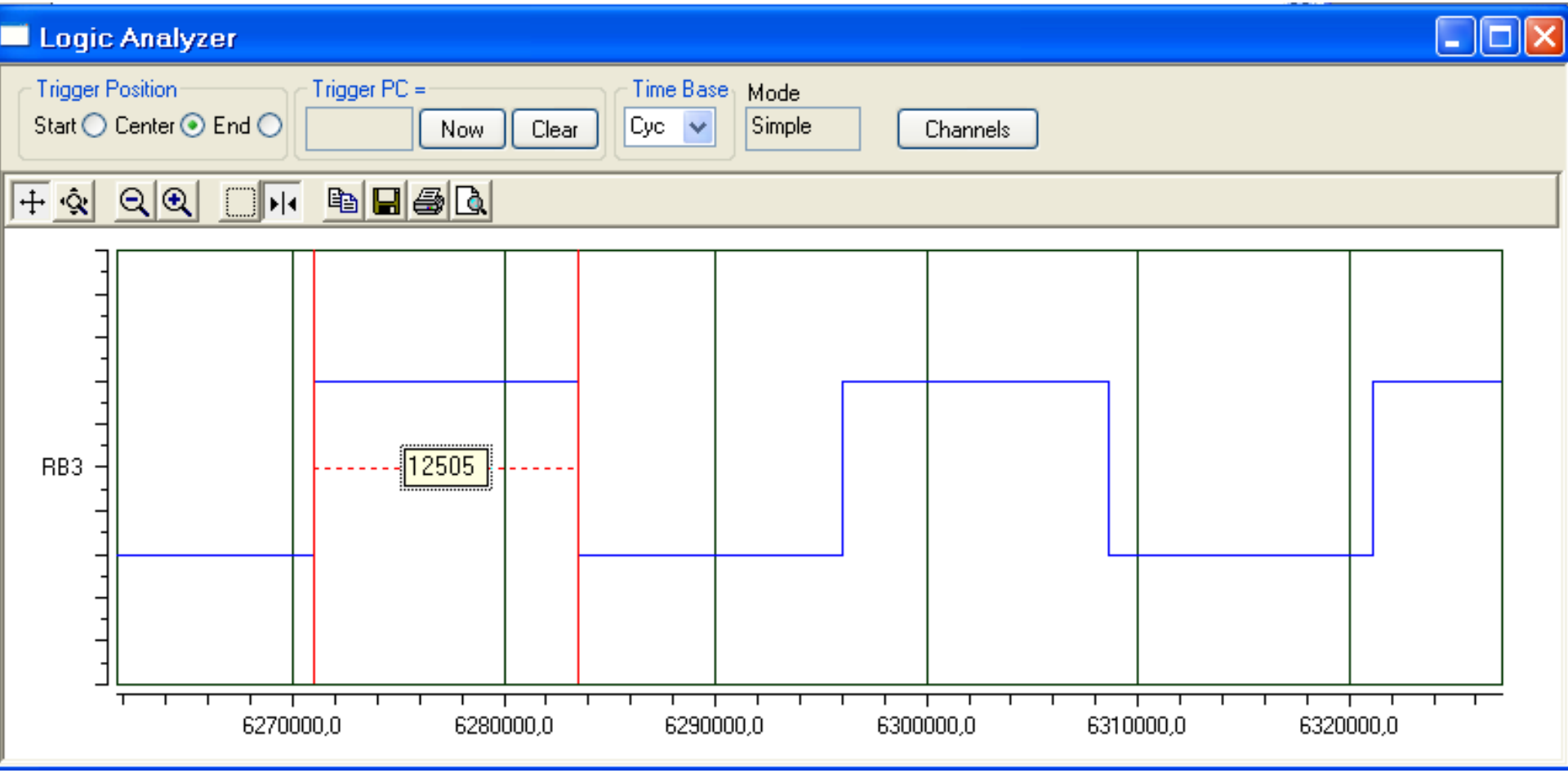
- T = 10 ms
- Time delay = 10ms/2 = 5 ms.
- We need 5ms/0.4us = 12500 clocks
- FFFF - 30D4 +1 =CF2C
- TMR0H = CFH
- TMR0L= 2CH

Example 9-8, Cont.

```
        BCF     TRISB,3
        MOVLW   0x08
        MOVWF   T0CON
HERE
        MOVLW   0xCF
        MOVWF   TMR0H
        MOVLW   0x2C
        MOVWF   TMR0L
        BCF     INTCON,TMR0IF
        CALL    DELAY
        BTG     PORTB,3
        BRA     HERE
```

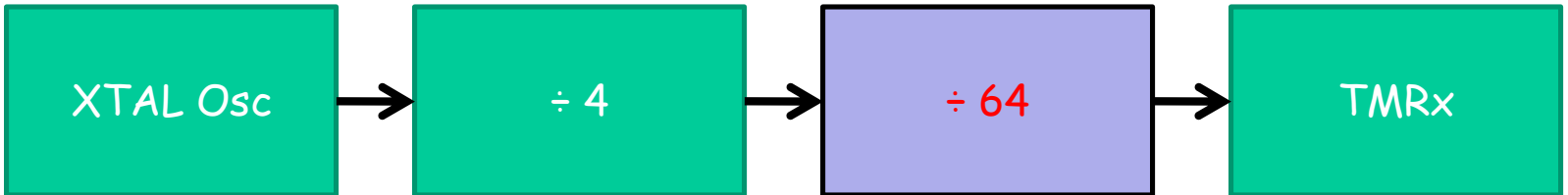
```
DELAY
        BSF     T0CON,TMR0ON
AGAIN
        BTFSS   INTCON,TMR0IF
        BRA     AGAIN
        BCF     T0CON,TMR0ON
        RETURN
```

Example 9-8, Cont.



Prescaler and generating larger delay

- The size of delay depend on
  - The Crystal frequency
  - The timer's 16-bit register.
- The largest timer happens when TMR0L=TMR0H=0
- Prescaler option is used to duplicate the delay by dividing the clock by a factor of 2,4, 8,16, 32,64 ,128,256
  - If T0CON=0000 0101, then T = 4*64/f

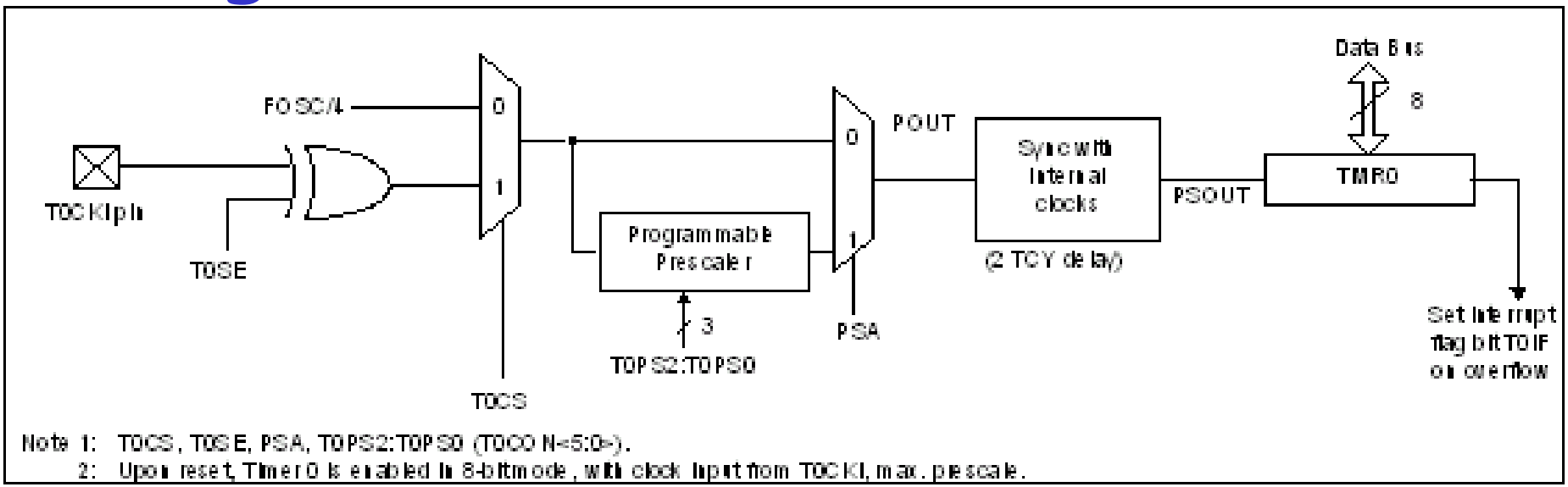


Example 9-13

- Examine the following program and find the time delay in second.
- Assume that XALT = 10 MHz.

```
        BCF     TRISB,2
        MOVLW   0x05
        MOVWF   T0CON
HERE
        MOVLW   0x01
        MOVWF   TMR0H
        MOVLW   0x08
        MOVWF   TMR0L
        BCF     INTCON,TMR0IF
        CALL    DELAY
        BTG     PORTB,2
        BRA     HERE
```

Figure 9-7. Timer0 8-bit Block Diagram



Note 1: T0CS, T0SE, PSA, T0PS2:T0PS0 (T0CON<5:0>).
2: Upon reset, Timer0 is enabled in 8-bit mode, with clock input from T0CKI, max. prescale.

Figure 9-8. Timer1 High and Low Registers

- Can be programmed in 16-bit mode only
- It has 2 bytes named as TMR1L and RMR1H
- It has also T1CON and TMR1IF
- The module incorporates its own low-power oscillator to provide an additional clocking option.
- Used as a low-power clock source for the microcontroller in power-managed operation.

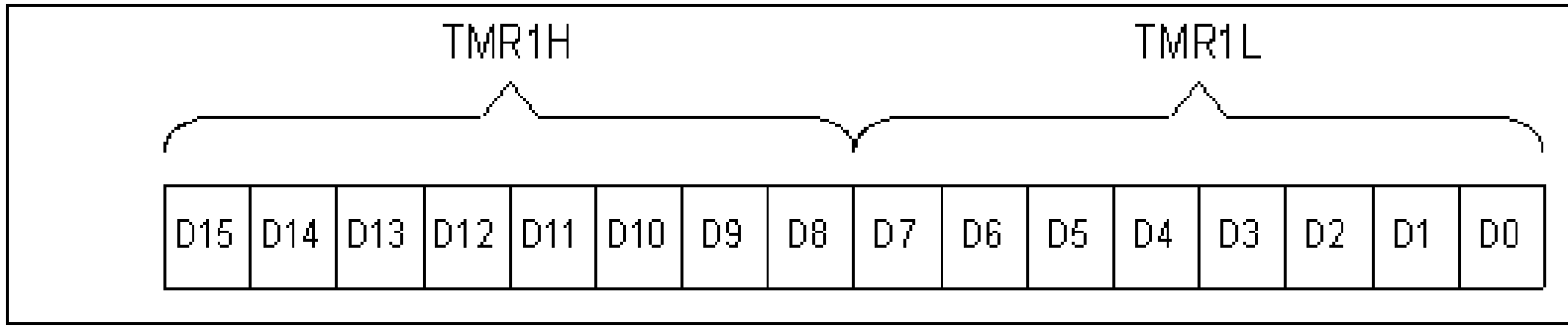


Figure 9-9. Timer1 Block Diagram

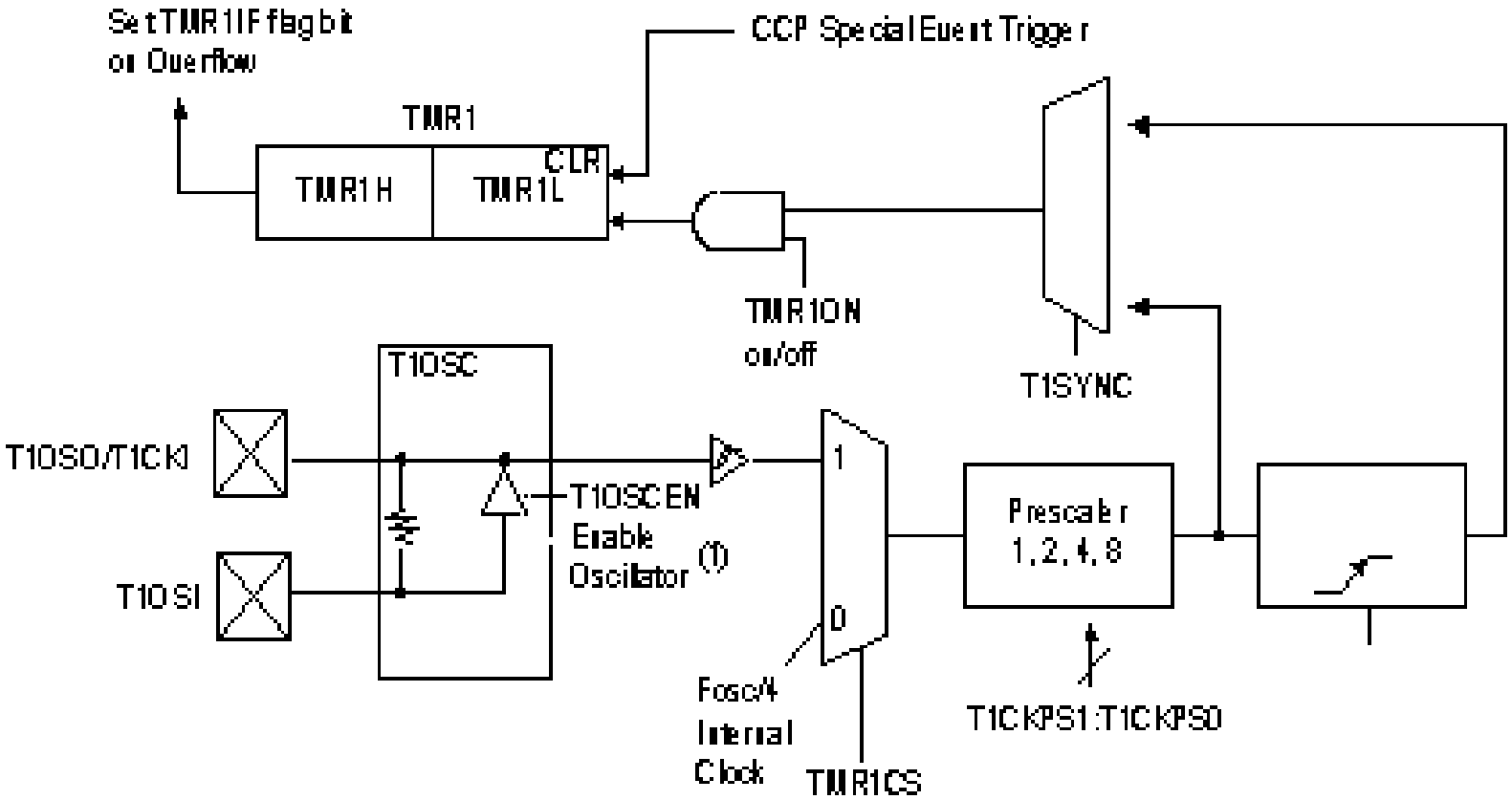


Figure 9-10. T1CON (Timer 1 Control) Register

RD16	---	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

RD16 D7 16-bit read/write enable bit
1 = Timer1 16-bit is accessible in one 16-bit operation.
0 = Timer1 16-bit is accessible in two 8-bit operations.

D6 Not used

T1CKPS2:T1CKPS0 D5 D4 Timer1 prescaler selector
0 0 = 1:1 Prescale value
0 1 = 1:2 Prescale value
1 0 = 1:4 Prescale value
1 1 = 1:8 Prescale value

T1OSCEN D3 Timer1 oscillator enable bit
1 = Timer1 oscillator is enabled.
0 = Timer1 oscillator is shutoff

T1SYNC D2 Timer1 synchronization (used only when TMR1CS = 1 for counter mode to synchronize external clock input)
If TMR1CS = 0 this bit is not used.

TMR1CS D1 Timer1 clock source select bit
1 = External clock from pin RC0/T1CKI
0 = Internal clock (Fosc/4 from XTAL)

TMR1ON D0 Timer1 ON and OFF control bit
1 = Enable (start) Timer1

Figure 9-11. PIR1 (Interrupt Control Register 1) Contains the TMR1IF Flag

							TMR1IF

TMR1IF D1 Timer1 Interrupt overflow flag bit
0 = Timer1 did not overflow.
1 = Timer1 has overflowed (FFFF to 0000).

The importance of TMR1IF: When TMR1H:TMR1L overflows from FFFF to 0000, this flag is raised. We monitor this flag bit before we reload the TMR1H:TMR1L registers.

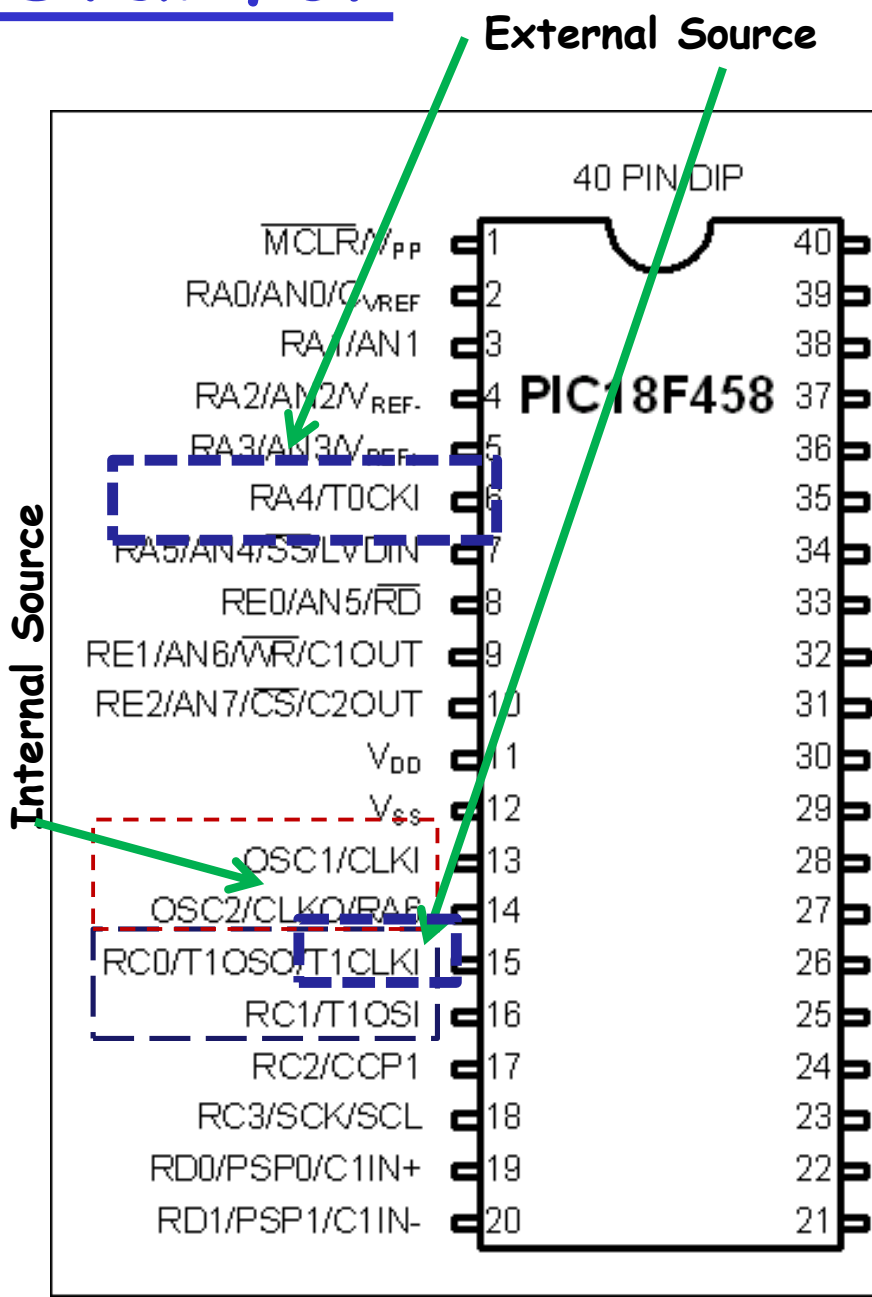
The other bits of this register are discussed in Chapter 11.

SECTION 9.2: Counter Programming

- Used to counts event outside the PIC
  - Increments the TMR0H and TMR0L registers
- T0CS in T0CON reg. determines the clock source,
  - If T0CS = 1, the timer is used as a counter
  - Counts up as pulses are fed from pin RA4 (T0CKI)
  - What does T0CON=0110 1000 mean?
- If TMR1CS=1, the timer 1 counts up as clock pulses are fed into pin RC0

Using external Crystal for Timer1 clock

- Timer1 comes with two options,
  - clock fed into T1CKI
    - T1OSCEN=0
  - Clock from a crystal connected to T1OSI-T1OSO (additional)
    - T1OSCEN=1
    - 32 kHz Crystal is connected
    - Used for saving power during SLEEP mode → doesn't disable Timer1 while the main crystal is shut down



Example 9-23

- Assuming that clock pulses are fed into pin T0CK1, write a program for counter 0 in 8-bit mode to count the pulses and display the state of the TMR0L count on PORTB.

```
        BSF     TRISA,RA
        CLRF    TRISB
        MOVLW   0x68
        MOVWF   T0CON
HERE    MOVLW   0x0
        MOVWF   TMR0L
        BCF     INTCON,TMR0IF
        BSF     T0CON,TMR0ON
AGAIN   MOVFF   TMR0L,PORTB
        BTFSS   INTCON,TMR0IF
        BRA     AGAIN
        BCF     T0CON,TMR0ON
        GOTO    HERE
```

Example 9-24

- Assume that a 1 Hz frequency pulse is connected to input for Timer0(T0CKI)
- Write a program to display counter 0 on PORTB, C and D in decimal.
- Ser the initial value of TMR0L to -60.

```
NUME      EQU 0x00
QU        EQU 0x20
RMND_L    EQU 0x30
RMND_M    EQU 0x31
RMND_H    EQU 0x32
MYDEN     EQU D'10'
   BSF      TRISA,RA4
   MOVLW    0x68
   MOVWF    T0CON
HERE       MOVLW    0x0
   MOVWF    TMR0L
   BCF      INTCON,TMR0I
   BSF      T0CON,TMR0ON
```

Example 9-24

```
AGAIN     MOVF
          TMR0L,W
  CALL
          BIN_ASC_CON
  BTFSS
          INTCON,TMR0IF
  BRA       AGAIN
  BCF     T0CON,TMR0ON
  GOTO      HERE
```

```
BIN_ASC_CON
    MOVFF       PORTB,WREG
    MOVWF       NUME
    MOVLW       MYDEN
    CLRF QU
D_1     INCF    QU
    SUBWF       NUME
    BC    D_1
    ADDWF       NUME
    DECF        QU
    MOVFF       NUME,RMND_L
    MOVFF       QU,NUME
    CLRF QU
D_2     INCF    QU
    SUBWF       NUM
    BC    D_2
    ADDWF       NUM
    DECF        QU
    MOVFF       NUME,RMND_M
    MOVFF       QU,RMND_H
    RETURN
```

Example 9-26

- Assuming that clock pulses are fed into pin T0CKI and a buzzer is connected to pin PORTB.1 write a program for counter0 in 8-bit mode to sound the buzzer every 100 pulses

```
BCF     TRISB,1
        BSF     TRISA,4
        MOVLW   0x68
        MOVWF   T0CON
        MOVLW   -D'100'
        MOVWF   TMR0L
        BCF     INTCON,TMR0IF
        BSF     T0CON,TMR0ON
AGAIN   BTFSS   INTCON,TMR0IF
        BRA     AGAIN
        BCF     T0CON,TMR0ON
OVER    BTG     PORTB,1
        CALL    DELAY
        GOTO    OVER
```

Example 9-27

- Assume that a 1 Hz frequency pulse is connected to input for Timer1(RC0)
- Write a program to display the counter values on PORTB and D in decimal.
- Initial value=0
- 16-bit and no Prescaler

```
        BSF     TRISC,RC0
        CLRF    TRISB
        CLRF    TRISD
        MOVLW   0x02
        MOVWF   T1CON
HERE    MOVLW   0x0
        MOVWF   TMR1H
        MOVLW   0x0
        MOVWF   TMR1L
        BCF     PIR1,TMR1IF
        BSF     T1CON,TMR1ON
```

Example 9-27

```
AGAIN    MOVFF
         TMR1H,PORTD
         MOVFF  TMR1L,PORTB
         BTFSS  PIR1,TMR1IF
         BRA    AGAIN
         BCF    PIR1,TMR1ON
         GOTO   HERE
```

Next: Chapter 10
PIC18 Serial Port Programming in Assembly and C

Chapter 9: Summary

- The PIC18 can have up to four or more timers/counters. Depending on the family member
- **Timers**: Generate Time Delays (using Crystal)
- **Counters**: Event counter (using Pulse outside)
- Timers are accessed as two 8-bit registers, TMRLx and TMRHx
- Can be used either 8-bit or 16-bit
- Each timer has its own Timer Control register

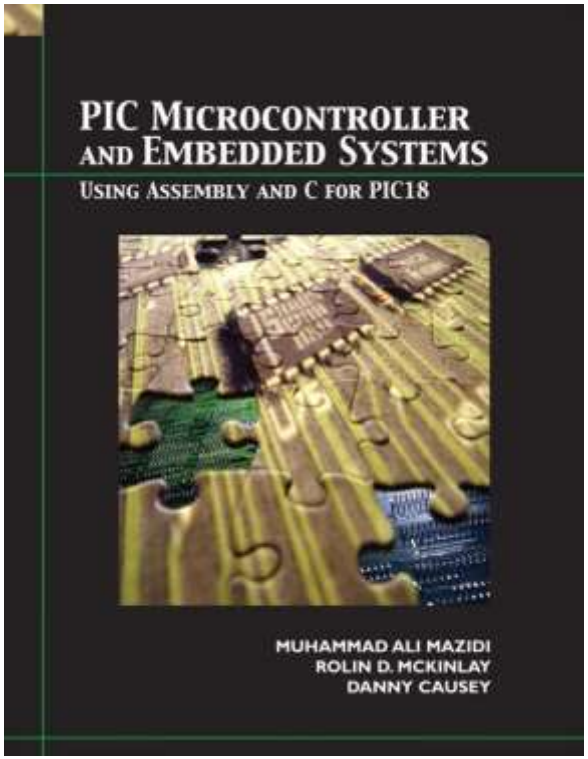
PIC Microcontroller and Embedded Systems

Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey

Eng. Husam Alzaq

The Islamic Uni. Of Gaza

Chapter 10: PIC18 Serial Port Programming in Assembly.



PIC Microcontroller and Embedded Systems Muhammad Ali Mazidi, Rolin McKinlay and Danny Causey, February 2007.

Objective

- Explain serial communication protocol
- Describe data transfer rate and bps rate
- Interface the PIC18 with an RS232 connector
- Describe the main registers used by serial communication of the PIC18
- Program the PIC18 serial port in Assembly

Outlines

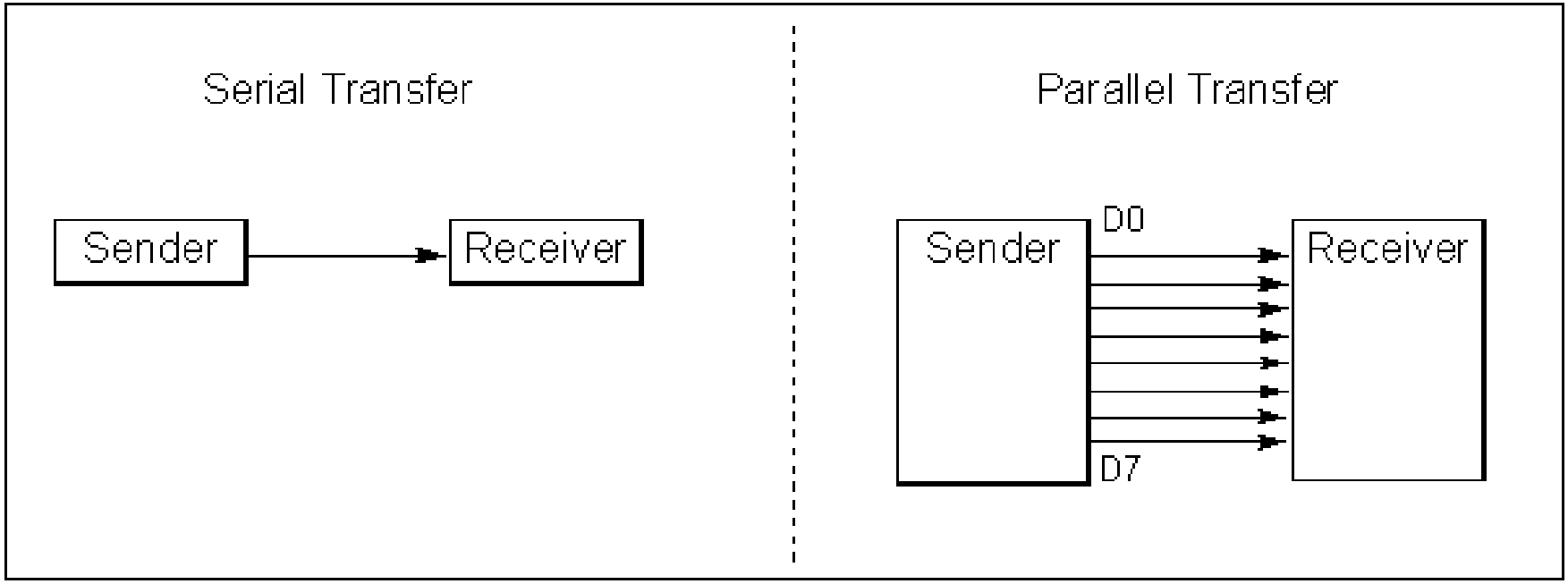
- Programming timers 0 and 1
- Counter Programming

Introduction

- Computers transfer data in two ways: Parallel and Serial.
- Parallel: Eight or more data lines, few feet only, short time
- Serial: Single data line, long distance
- The PIC18 has serial communication capability built into it.

Basics of Serial Communication

- The byte of data must be converted to serial bits using a parallel-in-serial-out shift register



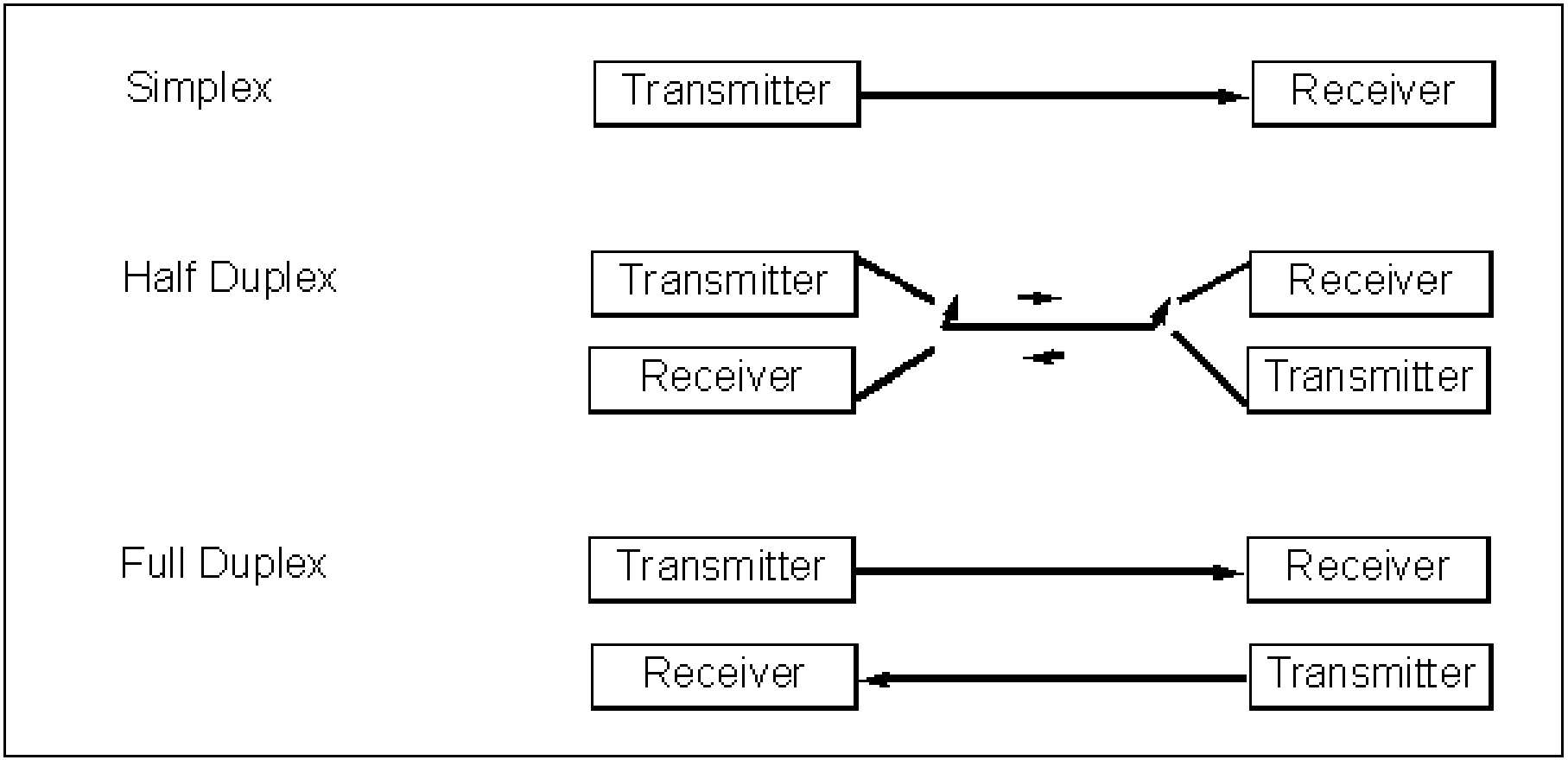
Basics of Serial Communication (cont'd)

- The receiving end must be a serial-in-parallel-out shift register and pack them into a byte.
- Two methods of serial data communication: Asynchronous and Synchronous

Transfers a single byte at a time

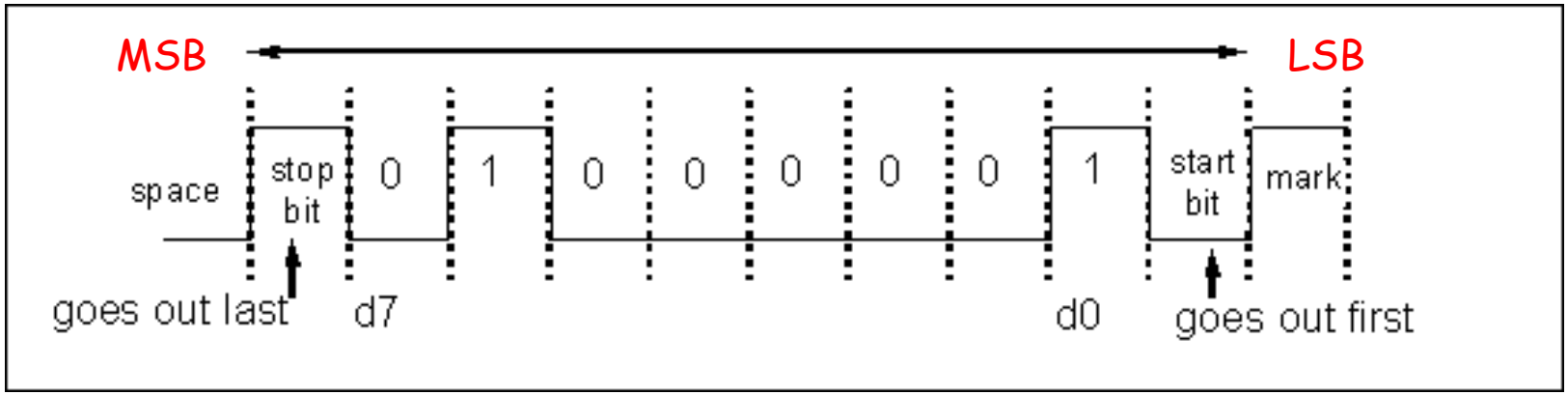
Transfers a block of data at a time

Half-and Full-Duplex Transmission



Start and Stop Bits

- In the asynchronous method, each character is placed between start and stop bits (framing)



Framing ASCII 'A' (41H)

Data Transfer Rate

- Rate of data transfer: *bps* (bits per second)
- Another widely used terminology for bps is baud rate
- For Asynchronous serial data communication, the baud rate is generally limited to 100,000bps

RS232 Standard

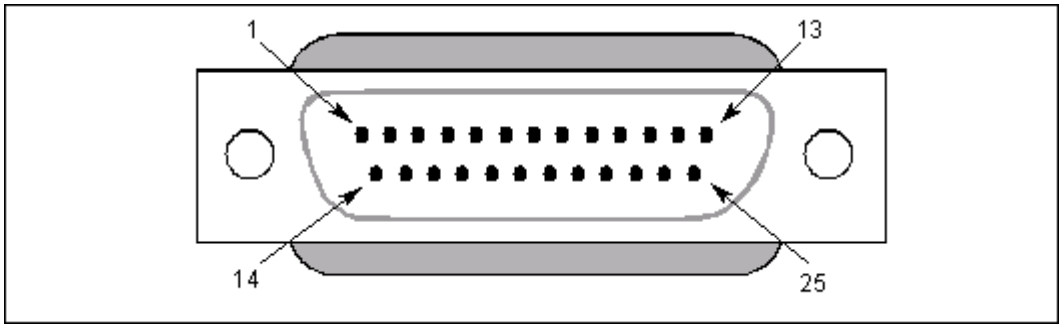
- Standard for serial comm (COM port)
  - 1: -3V to -25V;
  - 0: +3V to +25V
  - Reason: for long distance wired line
- Input-output voltage are not TTL compatible
- So, we need MAX232/233 for voltage converter. Commonly known as line drivers

RS232 Pins

Connectors:

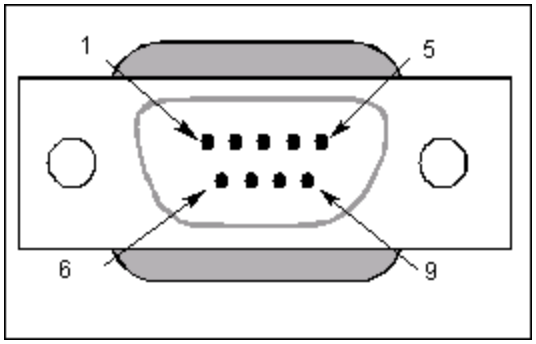
Minimally, 3 wires: RxD, TxD, GND

Could have 9-pin or 25-pin



DB-25

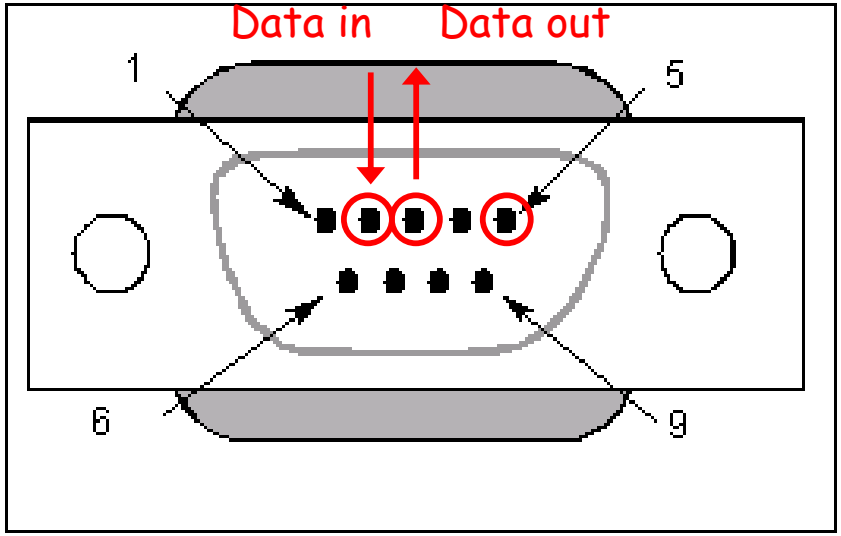
25-Pin Connector



DB-9

9-Pin Connector

RS232 Pins (cont'd)



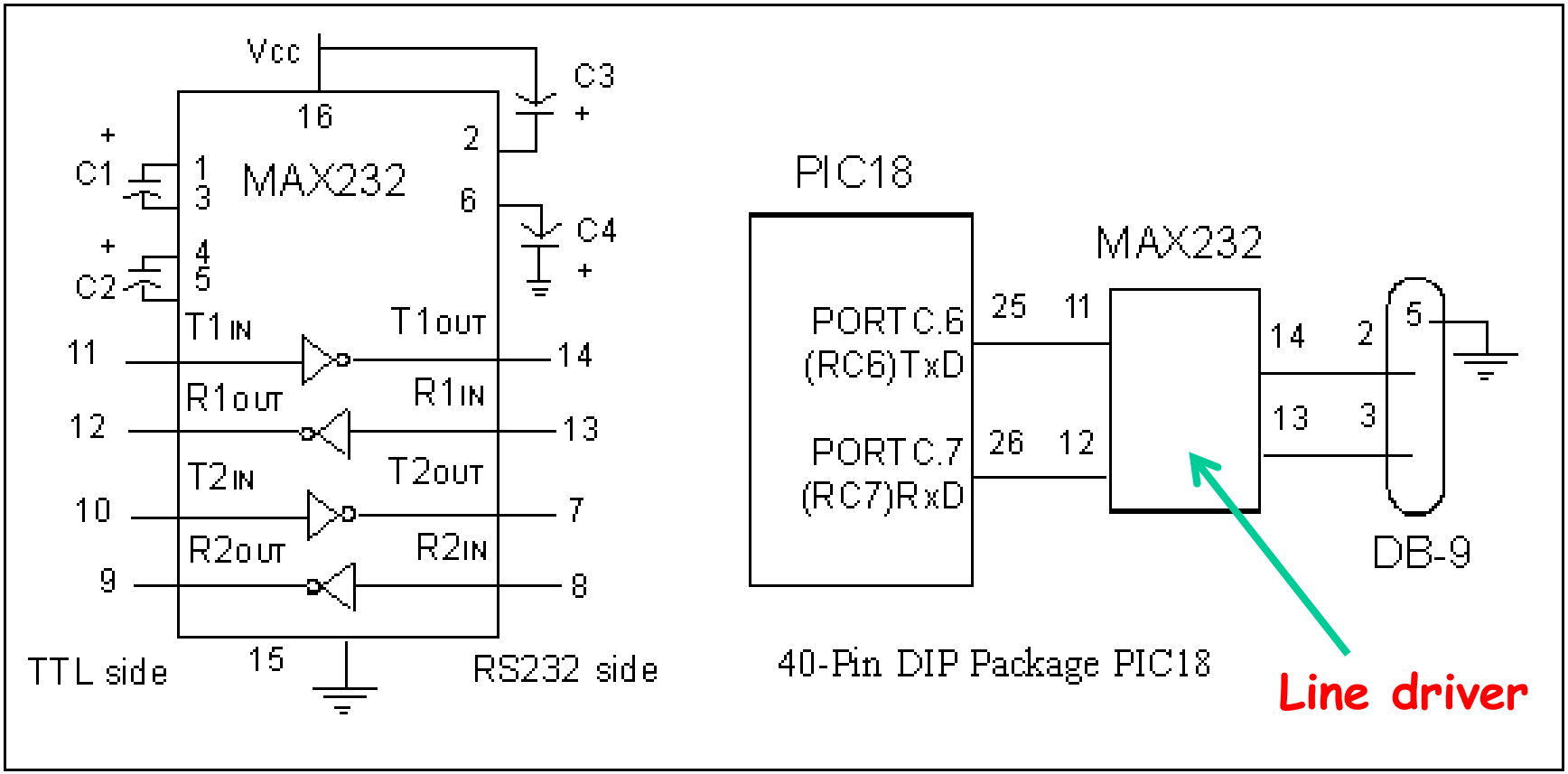
DB-9

9-Pin Connector

IBM PC DB-9 Signals

Pin 1 – Data Carrier Detect (DCD)
Pin 2 – Received Data (RxD)
Pin 3 – Transmitted Data (TxD)
Pin 4 – Data Terminal Ready (DTR)
Pin 5 – Signal Ground (GND)
Pin 6 – Data Set Ready (/DSR)
Pin 7 – Request to Send (/RTS)
Pin 8 – Clear to Send (/CTS)
Pin 9 – Ring Indicator (RI)

PIC18 Connection to RS232

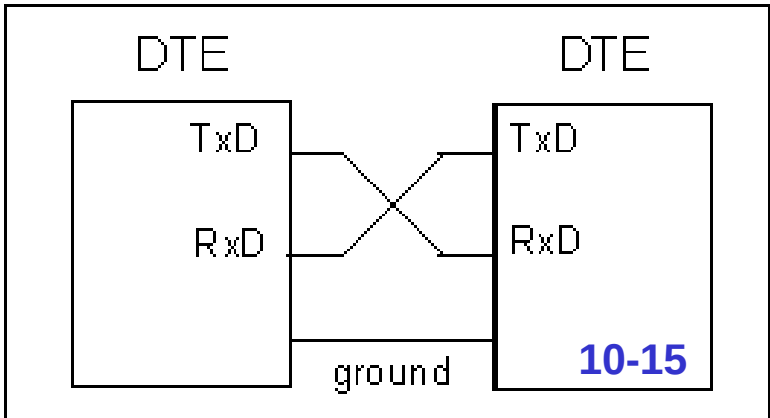


(a) Inside MAX232

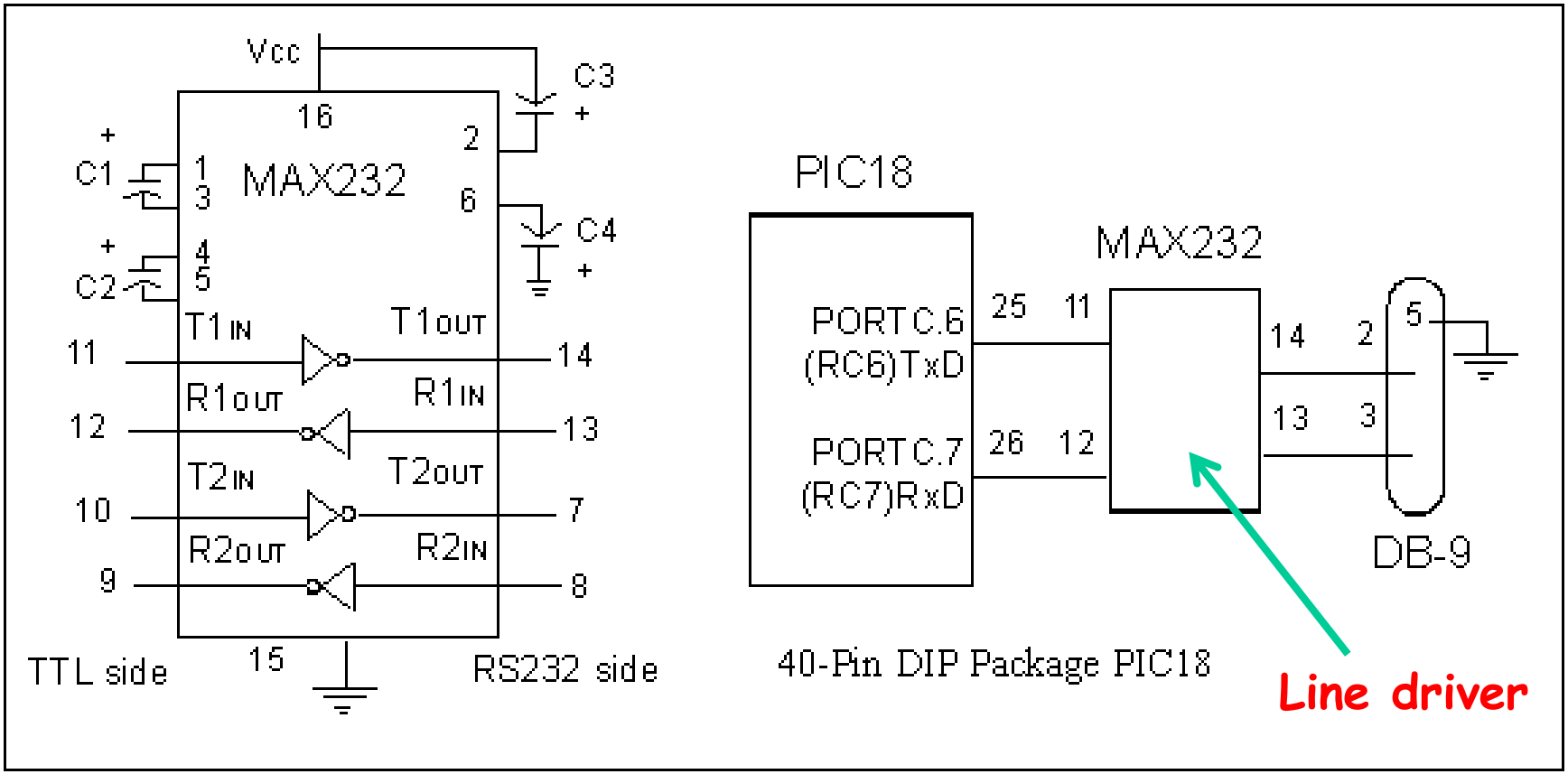
(b) its Connection to the PIC18

Figure 10-6. Null Modem Connection

- **Null modem** is a communication method to connect two DTEs (computer, terminal, printer etc.) directly using a RS-232 serial cable.
- With a null modem connection the transmit and receive lines are **crosslinked**.
- Depending on the purpose, sometimes also one or more handshake lines are crosslinked.



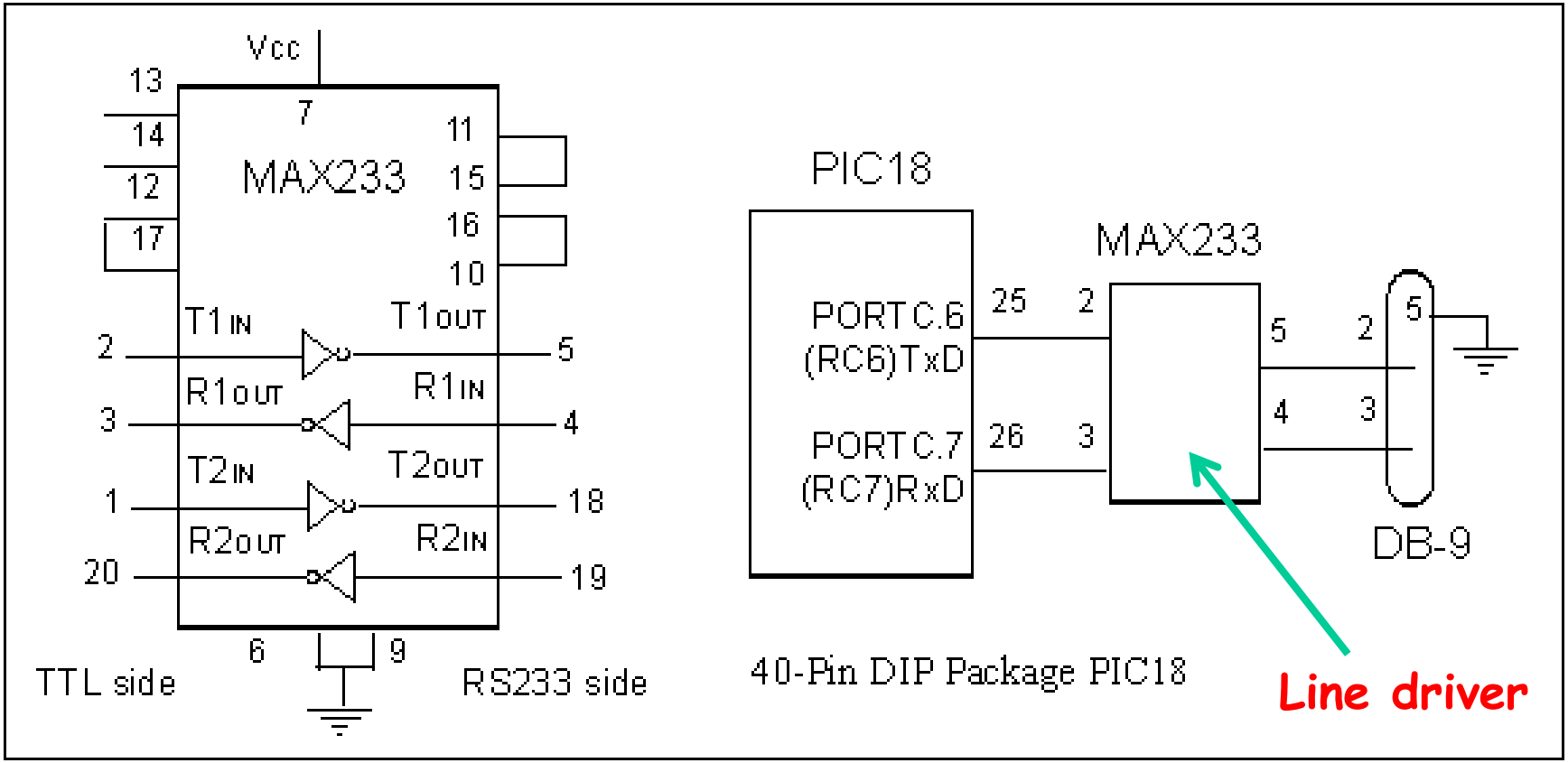
PIC18 Connection to RS232



(a) Inside MAX232

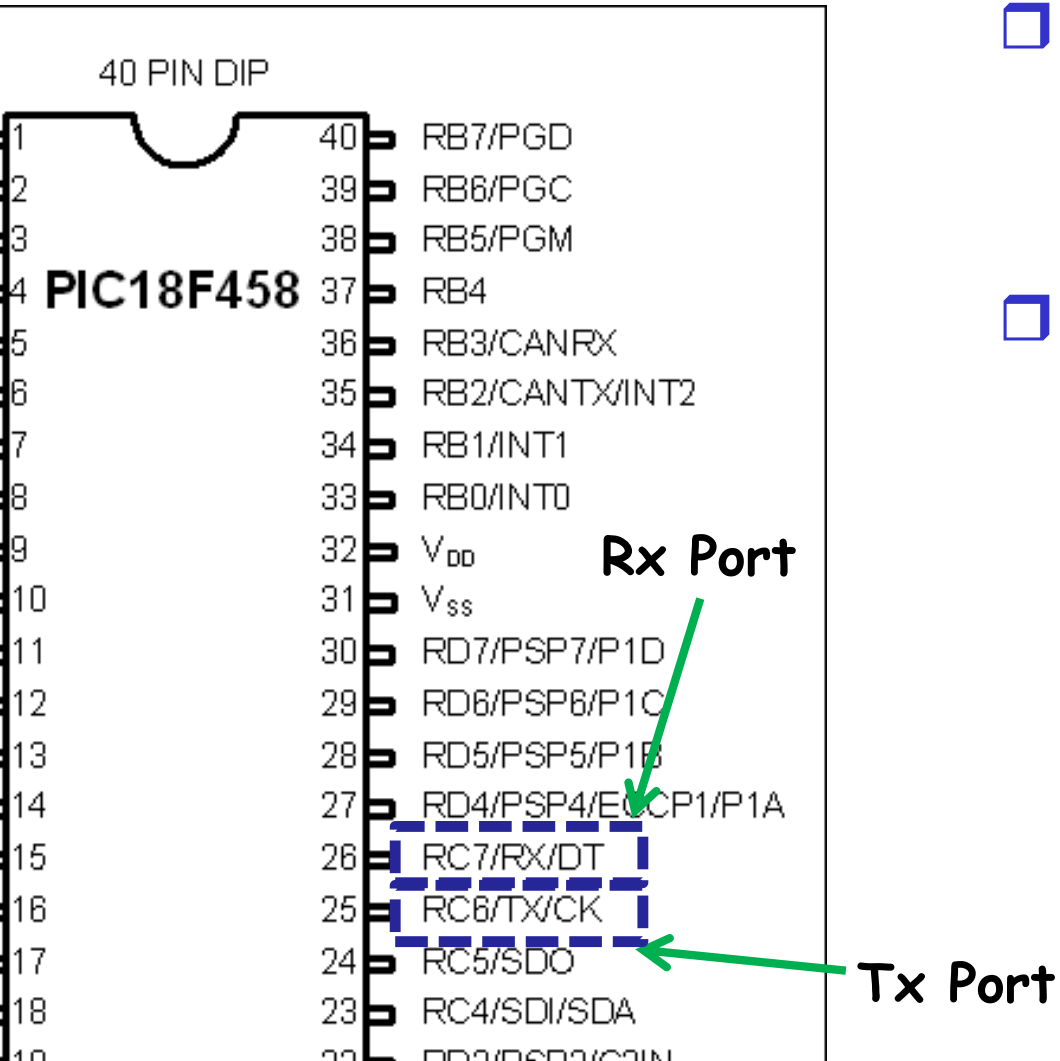
(b) its Connection to the PIC18

PIC18 Connection to RS232 (Cont'd)



(a) Inside MAX233 (b) Its Connection to the PIC18

Section10.3: PIC18 Serial Port Programming in Assembly



- USART has both
  - Synchronous
  - Asynchronous
- 6 registers
  - SPBRG
  - TXREG
  - RCREG
  - TXSTA
  - RCSTA
  - PIR1

SPBRG Register and Baud Rate in the PIC18

- The baud rate in is programmable
- loaded into the SPBRG decides the baud rate
- Depend on crystal frequency
  - $BR = \frac{F_{osc}}{4*16*(X+1))}$

Baud Rate	SPBRG (Hex Value)
38400	3
19200	7
9600	F
4800	20
2400	40
1200	81

*For XTAL = 10MHz only!

Baud rate Formula

If Fosc = 10MHz
X = (156250/Desired Baud Rate) - 1

Example:

Desired baud rate = 1200, Clock Frequency = 10MHz

X = (156250/1200) – 1
X = 129.21 = 129 = 81H

TXREG Register

- 8-bit register used for serial communication in the PIC18
- For a byte of data to be transferred via the Tx pin, it must be placed in the TXREG register first.
- The moment a byte is written into TXREG, it is fetched into a non-accessible register TSR

```
MOVFF PORTB, TXREG
```

- The frame contains 10 bits

RCREG Register

- 8-bit register used for serial communication in the PIC18
- When the bits are received serially via the Rx pin, the PIC18 deframes them by eliminating the START and STOP bit, making a byte out of data received and then placing it in the RCREG register

```
MOVFF RCREG, PORTB
```

TXSTA (Transmit Status and Control Register)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN[1]	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

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

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care.

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9:** 9-Bit Transmit Enable bit

1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit[1]

1 = Transmit enabled
0 = Transmit disabled

TXSTA (Transmit Status and Control Register) (Cont'd)

bit 4 **SYNC:** EUSART Mode Select bit

1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB:** Send Break Character bit

Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed

Synchronous mode:
Don't care.

bit 2 **BRGH:** High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode.

bit 1 **TRMT:** Transmit Shift Register Status bit

1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data

Can be address/data bit or a parity bit.

Thre! **Note 1:** SREN/CREN overrides TXEN in Sync mode.

RCSTA (Receive Status and Control Register)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

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

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled (held in Reset)

bit 6 **RX9:** 9-Bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

RCSTA (Receive Status and Control Register) (Cont'd)

bit 4 **CREN:** Continuous Receive Enable bit

Asynchronous mode:
1 = Enables receiver
0 = Disables receiver

Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit

Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and loads the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

Asynchronous mode 9-bit (RX9 = 0):
Don't care.

bit 2 **FERR:** Framing Error bit

1 = Framing error (can be updated by reading RCREG register and receiving next valid byte)
0 = No framing error

bit 1 **OERR:** Overrun Error bit

1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data

This can be an address/data bit or a parity bit and must be calculated by user firmware.

PIR1 (Peripheral Interrupt Request Register 1)

---	--	RCIF	TXIF	--	---	--	

RCIF Receive interrupt flag bit
1 = The UART has received a byte of data and it is sitting in the RCREG register (receive buffer), waiting to be picked up.
Upon reading the RCREG register, the RCIF is cleared to allow the next byte to be received.
0 = The RCREG is empty.

TXIF Transmit interrupt flag bit
0 = The TXREG register is full.
1 = The TXREG (transmit buffer) register is empty.

The importance of TXIF: To transmit a byte of data, we write it into TXREG. Upon writing a byte into TXREG, the TXIF flag is cleared. When the entire byte is transmitted via the TX pin, the TXIF flag bit is raised to indicate that it is ready for the next byte. So, we must monitor this flag before we write a new byte into TXREG, otherwise, we wipe out the last byte before it is transmitted.

Programming the PIC18 to Transfer Data Serially

1. TXSTA register = 20H: Indicating asynchronous mode with 8-bit data frame, low baud rate and transmit enabled
2. Set Tx pin an output (RC6)
3. Loaded SPBRG for baud rate
4. Enabled the serial port (SPEN = 1 in RCSTA)
5. The character byte to transmit must be written into TXREG
6. Keep Monitor TXIF bit
7. To transmit next character, go to step 5

Example 10.2

```
;Write a program for the PIC18 to transfer the letter 'G' serially
;at 9600 baud continuously. Assume XTAL = 10 MHz

        MOVLW       B'00100000'
        MOVWF       TXSTA
        MOVLW       D'15'; 9600 bps
        MOVWF       SPBRG
        BCF         TRISC,   TX
        BSF         RCSTA,   SPEN
OVER    MOVLW       A'G'
S1      BTFSS   PIR1,   TXIF
        BRA         S1
        MOVWF       TXREG
        BRA         OVER
```

TXSTA: Transmit Status and Control Register

The importance of the TSR register. To transfer a byte of data serially, we write it into TXREG. The TSR (transmit shift register) is an internal register whose job is to get the data from the TXREG, frame it with the start and stop bits, and send it out one bit at a time via the TX pin. When the last bit, which is the stop bit, is transmitted, the TRMT flag is raised to indicate that it is empty and ready for the next byte. When TSR fetches the data from TXREG, it clears the TRMT flag to indicate it is full. Notice that TSR is a parallel-in-serial-out shift register and is not accessible to the programmer. We can only write to TXREG. Whenever the TSR is empty, it gets its data from TXREG and clears the TXREG register immediately, so it does not send out the same data twice.

Programming the PIC18 to Receive Data Serially

1. RCSTA register = 90H: To enable the continuous receive in addition to the 8-bit data size option
2. The TXSTA register = 00H: To choose the low baud rate option
3. Loaded SPBRG for baud rate
4. Set Rx pin an input
5. Keep Monitor RCIF bit
6. Move RCREG into a safe place
7. To receive next character, go to step 5

Example 10.4

```
;Write a program for the PIC18 to receive data serially and
;put them on PORTB. Set the baud rate at 9600, 8-bit data
;and 1 stop bit

        MOVLW       B'10010000'
        MOVWF       RCSTA
        MOVLW       D'15'
        MOVWF       SPBRG
        BSF         TRISC,   RX
        CLRF        TRISB
R1      BTFSS   PIR1,   RCIF
        BRA         R1
        MOVFF       RCREG, PORTB
        BRA         R1
```

Increasing the Baud Rate

- Faster Crystal
  - May not be able to change crystal
- TXSTA.BRGH bit
  - Normally used low
  - Can be set high
  - Quadruples rate when set high

Baud Rate Error Calculation

- ??? Errors in the baud rate? Yep!
  - Caused by using integer division in rate generator

$$Error = \frac{Calculated\ baud\ rate - Desired\ \ Baud\ \ Rate)}{Desired\ \ Baud\ \ Rate}$$

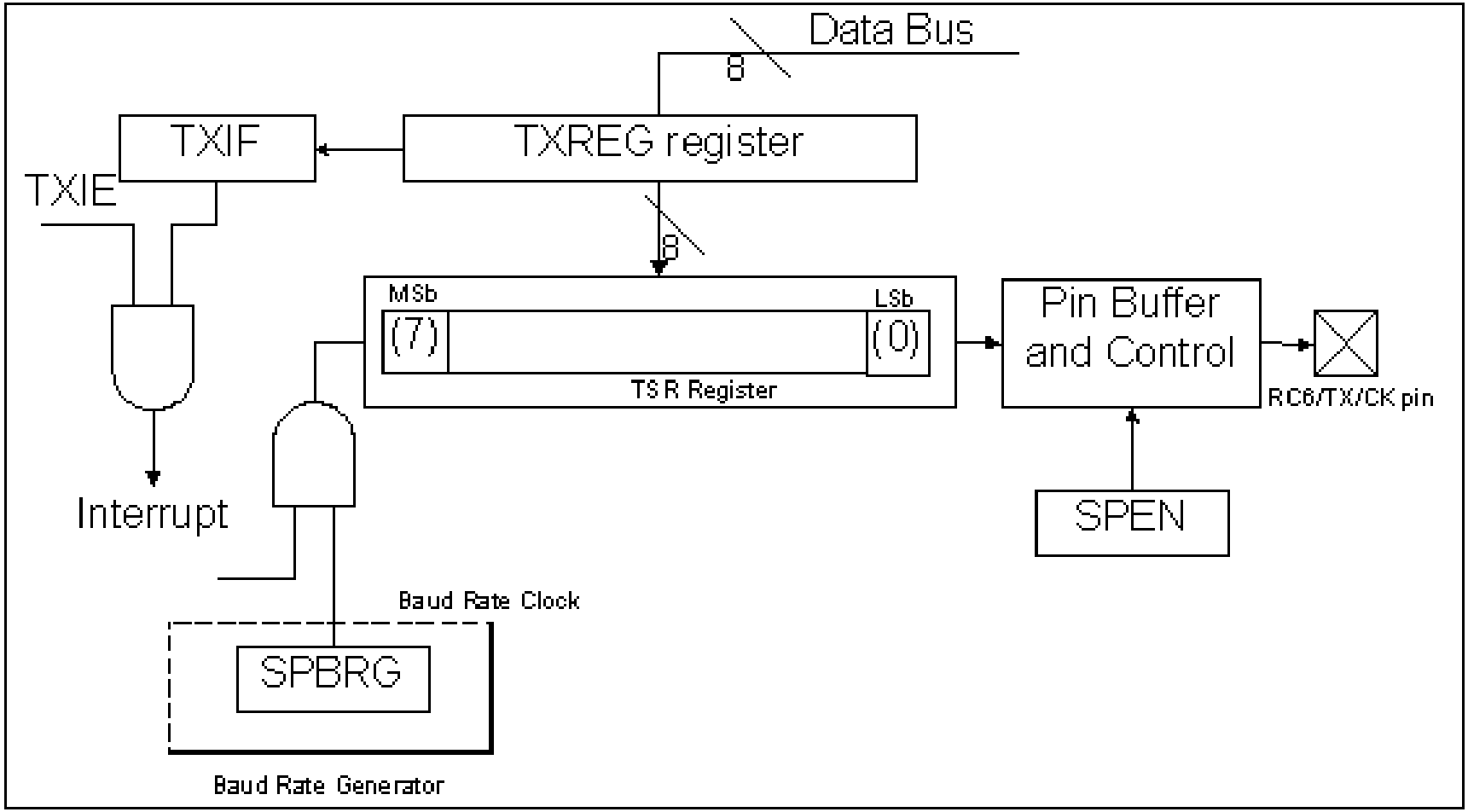
$$where$$

$$Calculated\ Baud\ Rate = \frac{F_{OSC}}{64 \times (SPBRG + 1)}$$

Transmit and Receive

- Please see program 10-1: Page 412

Figure 10-12. Simplified USART Transmit Block Diagram



Chapter 10: Summary

Next: the final exam