Chapter 3 Assembly language programming

* PIC18 assembly language programming
* branching, looping, time delay
* I/O port
* arithmetic and logic instructions
* addressing modes
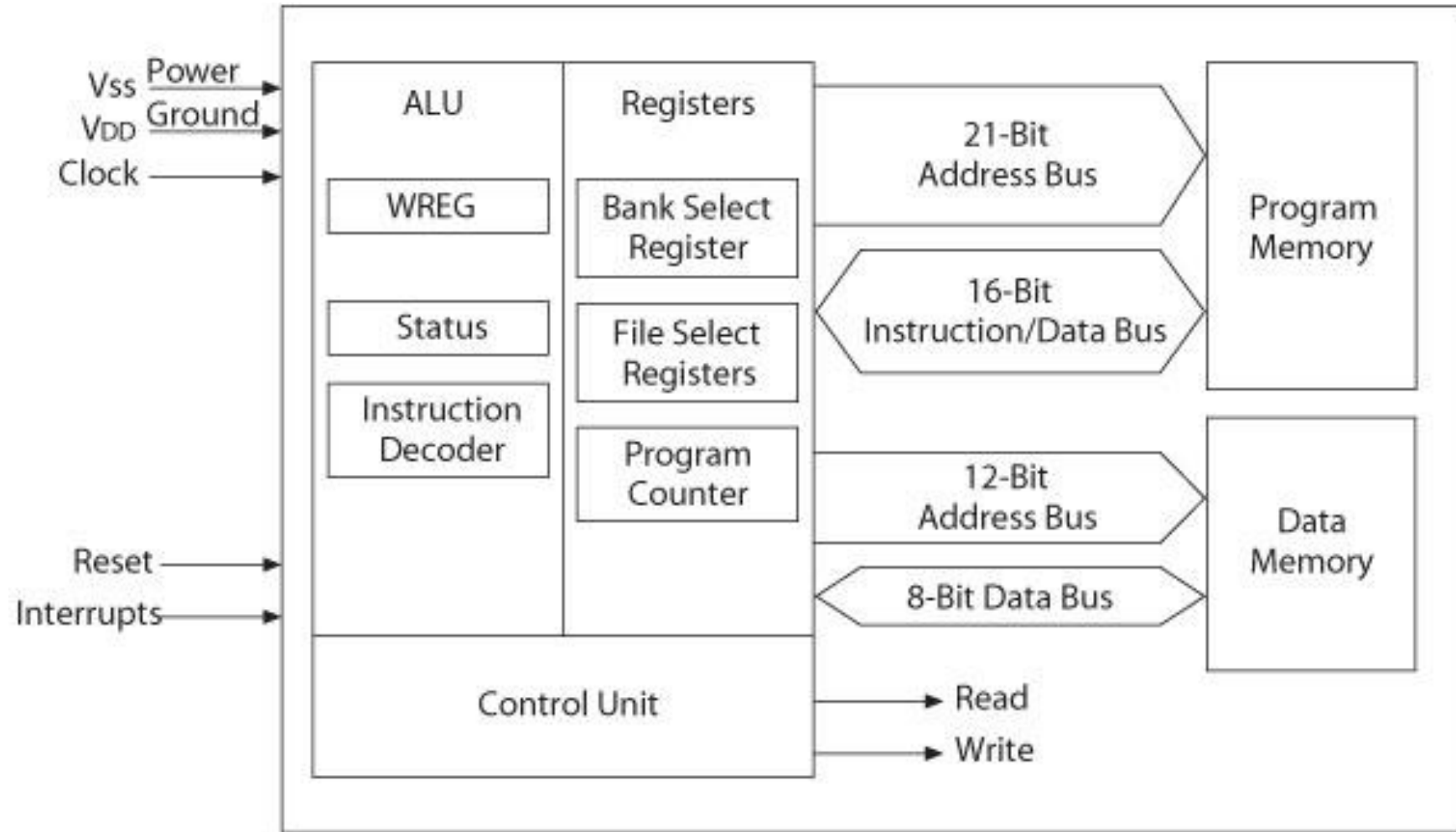* look-up table, stack, subroutine

3.1 PIC18 assembly language programming

- revision of PIC18 architecture

- PIC18 assembly language

- structure of PIC18 assembly language program

- assemble and run PIC18 program

- ROM space

- flags and status register
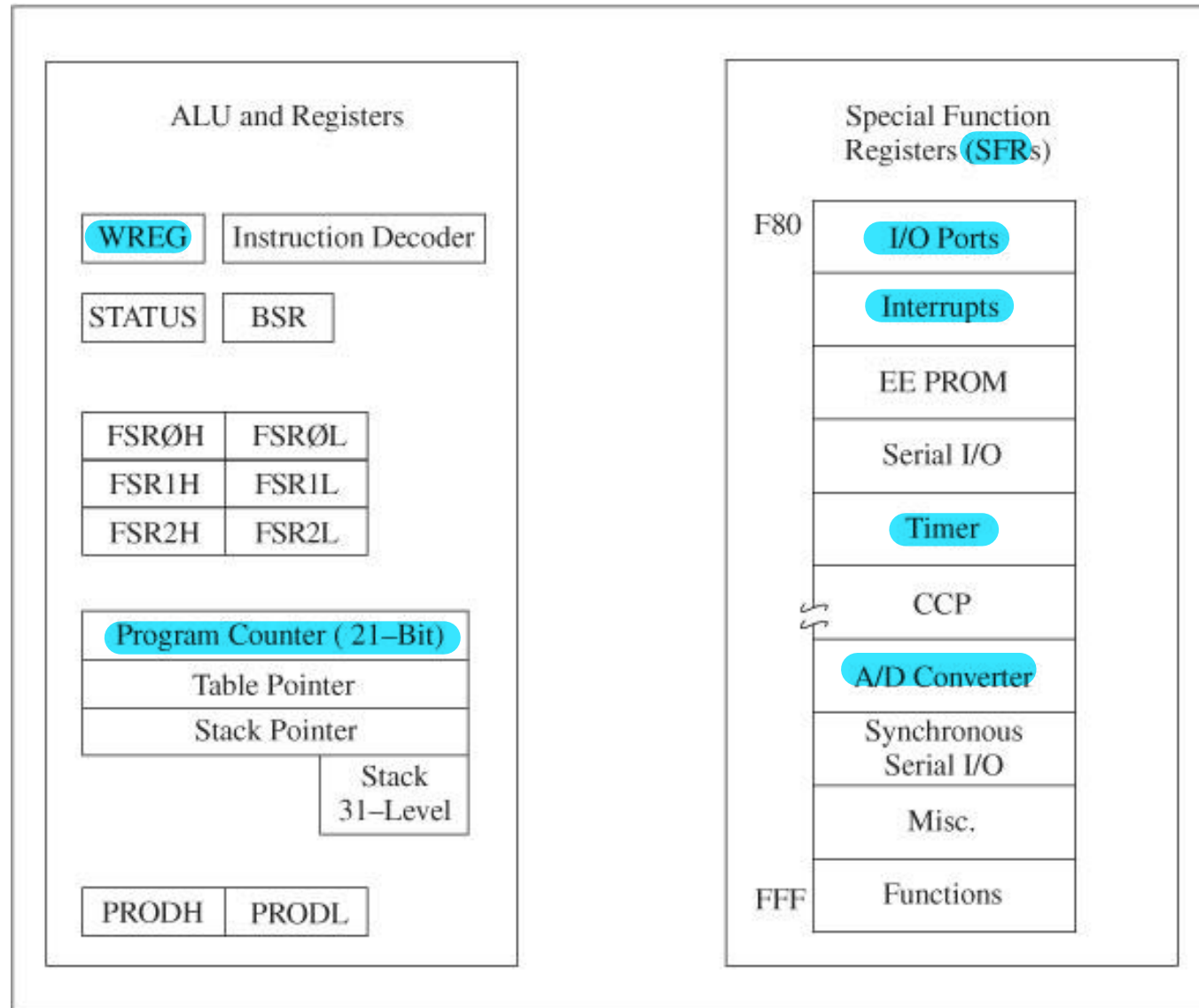
- register banks

3.1.1 revision of PIC18 architecture

- structure of PIC18 – components and their relation

- function of PIC18 – necessary for writing assembly language program
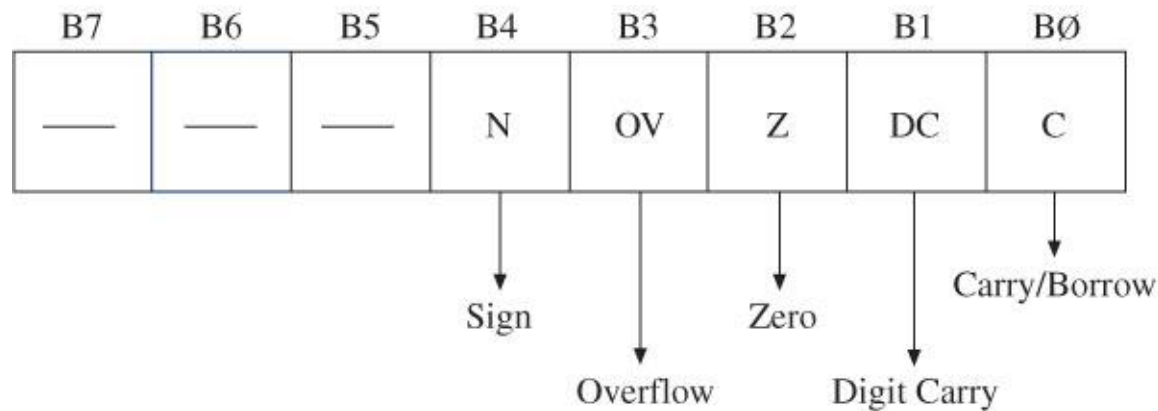
Harvard architecture

Programming model

ALU and Registers

WREG	Instruction Decoder

STATUS	BSR

FSRØH	FSRØL
FSR1H	FSR1L
FSR2H	FSR2L

Program Counter (21–Bit)
Table Pointer
Stack Pointer

Stack 31–Level

PRODH	PRODL

Special Function Registers (SFRs)

F80

I/O Ports
Interrupts
EE PROM
Serial I/O
Timer
CCP
A/D Converter
Synchronous Serial I/O
Misc.
Functions

FFF

- Microprocessor unit contains Arithmetic Logic Unit (ALU), registers, and control unit

- ALU contains:
  - 16-bit instruction decoder
  - 8-bit Working Register (WREG) – involved in the execution of many instructions
  - status register (5 flag bits) – contains arithmetic status of ALU

Flags in Status Register
(check the state of WREG)

B7	B6	B5	B4	B3	B2	B1	BØ
—	—	—	N	OV	Z	DC	C

N → Sign
OV → Overflow
Z → Zero
DC → Digit Carry
C → Carry/Borrow

- **N (Negative Flag)**
  - Set when bit 7 is 1 as the result of an arithmetic/logic operation
- **OV (Overflow Flag)**
  - Set when there is overflow from high-order bit into sign bit
- **Z (Zero Flag)**
  - Set when result of an operation is zero
- **DC (Digit Carry Flag, Half Carry)**
  - Set when there is carry from bit 3 to bit 4
- **C (Carry Flag)**
  - Set when there is a carry out from bit 7

- Registers:

  21-bit Program Counter (PC) – pointer to program memory during program execution

  21-bit Table Pointer – memory pointer to copy bytes between program memory and data registers

  Stack Pointer (SP) – point to stack (31 registers used for temporary storage of memory addresses during program execution

  Product – 16-bit product of 8-bit by 8-bit multiply

  4-bit Bank Select Register (BSR) – upper 4-bit of 12-bit address of data memory

  File Select Registers (FSRs) – FSR0, FSR1, FSR2, each composed of 8-bit H and 8-bit L

- Program Memory
  - 21-bit address bus
  - address up to 2^{21}=2M bytes of memory
  - not all memory locations are implemented
  - 16-bit data bus
- Data Memory
  - also called file register
  - 12-bit address bus
  - address up to 2^{12}=4K bytes of memory
  - 8-bit data bus

- data memory is divided into 2 sections – GPR and SFR

- General-purpose registers (GPRs) provide storage for variables used in a program.

- Special-function registers (SFRs) are used to control the operation of the CPU and peripherals.
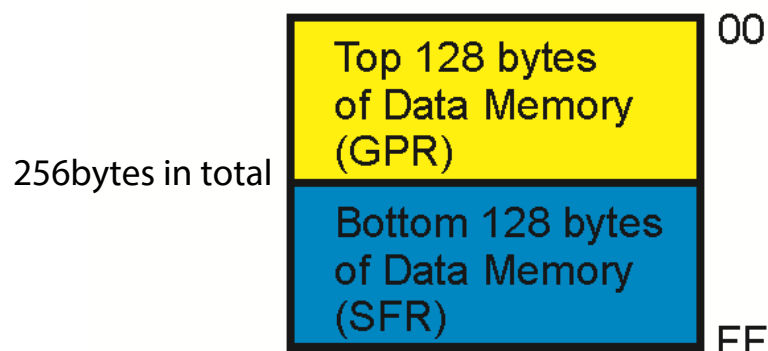 - I/O Ports (A to E)
  - Interrupts
  - EEPROM
  - Serial I/O
  - Timers
  - Capture/Compare/PWM (CCP)
  - Analog-to-Digital (A/D) Converter
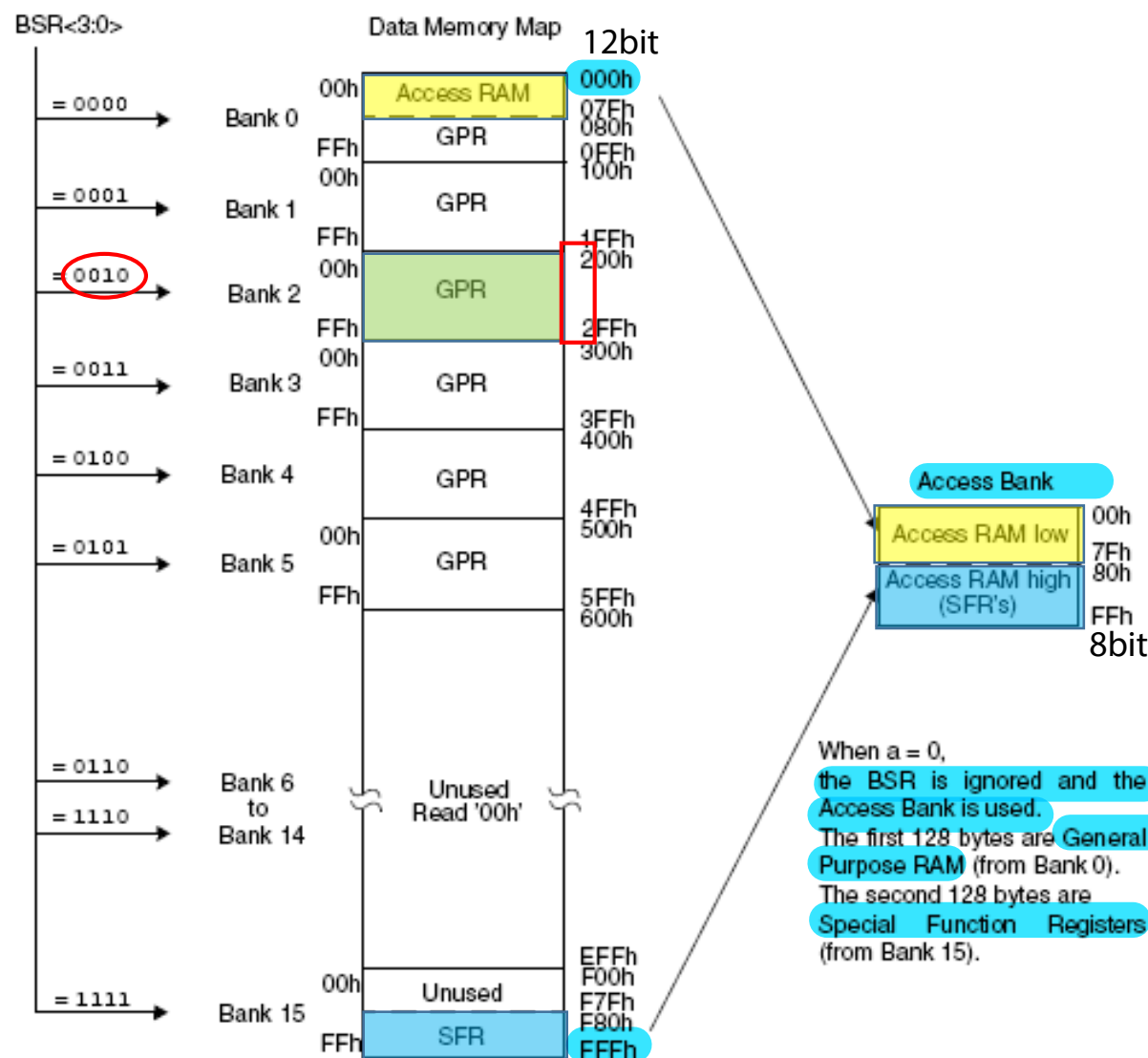
Access (Default) Bank Mode:

256bytes in total

| Top 128 bytes of Data Memory (GPR) | 00 |
| Bottom 128 bytes of Data Memory (SFR) | FF |

Bank specified by Bank Select Register (BSR)

| | 00 |
| | FF |

128 bytes SFRs

How many bytes GPRs?

256 x 6 = 1536

BSR<3:0>   Data Memory Map   12bit

= 0000 → Bank 0   00h Access RAM 000h
                        GPR   07Fh 080h
                   FFh   0FFh 100h
= 0001 → Bank 1   00h GPR
                   FFh   1FFh 200h
= 0010 → Bank 2   00h GPR
                   FFh   2FFh 300h
= 0011 → Bank 3   00h GPR
                   FFh   3FFh 400h
= 0100 → Bank 4   GPR
                                    4FFh 500h
= 0101 → Bank 5   00h GPR
                   FFh   5FFh 600h
= 0110 → Bank 6
= 1110 → to Bank 14   Unused Read '00h'
                                    EFFh F00h
= 1111 → Bank 15   00h Unused   F7Fh F80h
                   FFh SFR   FFFh

Access Bank

Access RAM low	00h
	7Fh
Access RAM high (SFR's)	80h
	FFh

8bit

When a = 0, the BSR is ignored and the Access Bank is used. The first 128 bytes are General Purpose RAM (from Bank 0). The second 128 bytes are Special Function Registers (from Bank 15).

When a = 1, the BSR is used to specify the RAM location that the instruction uses.

Banks in Data Memory

- Divided into 16 banks.

- 256 bytes per bank

- Access (Default) Bank is a 256-byte bank consisting of:

  – 128 bytes of GPRs located at 00H to 7FH in the access bank, mapped from 000H to 07FH of the data memory

  – 128 bytes of SFRs located at 80H to FFH in the access bank, mapped from F80H to FFFH of the data memory

- A program that requires more than the amount of RAM provided in the access bank necessitates *bank switching*.

PIC18 uses the bank concept because in many instructions there are 8 bits to indicate the RAM address (12 bits address)

3.1.2 PIC18 assembly language

- some data types

- some PIC18 instructions

- access data memory (file register)

Data Format Representation

- Data can only be represented as 8-bit number in PIC18
- Four ways to represent a byte:
  - Hexadecimal (Default)
  - Binary
  - Decimal
  - ASCII

Hexadecimal Numbers

- Four ways to show that hex representation is used:
    1. Put nothing in front or back of the number, e.g. movlw 99 (Hex is the default representation)
    2. Use h (or H) right after the number, e.g. movlw 99H
    3. Put 0x (or 0X) before the number, e.g. movlw 0x99
    4. Put h in front of the number, with single quotes around the number, e.g. movlw h'99'
- If the starting hex digit is A-F, the number must be preceded by a 0.
    - e.g. movlw C6 is invalid. Must be movlw 0C6

Binary, Decimal, ASCII

- The only way to represent a binary number is to put a B (or b) in front, with single quotes around the binary digits, e.g. movlw B'10011001'

- Two ways to present a decimal number:

  1. Put a D (or d) in front, with single quotes around the decimal digits, e.g. movlw D'12'

  2. Use the ".value" format, e.g. movlw .12

- The only way to represent an ASCII character is to put a A (or a) in front, with single quotes around the ASCII character, e.g. movlw A'2'.

- The ASCII code 0x32 is used to represent the character '2'. 0x32 is stored in WREG.

PIC18 Instruction Set

- Includes 77 instructions
  - 73 one word (16-bit) long 1 bus cycle
  - 4 two words (32-bit) long 2 bus cycle
- Divided into seven groups
  - Move (Data Copy) and Load
  - Arithmetic
  - Logic
  - Program Redirection (Branch/Jump)
  - Bit Manipulation
  - Table Read/Write
  - Machine Control

MOVLW

Moves 8-bit data into WREG

• MOVLW k; move literal value k into WREG
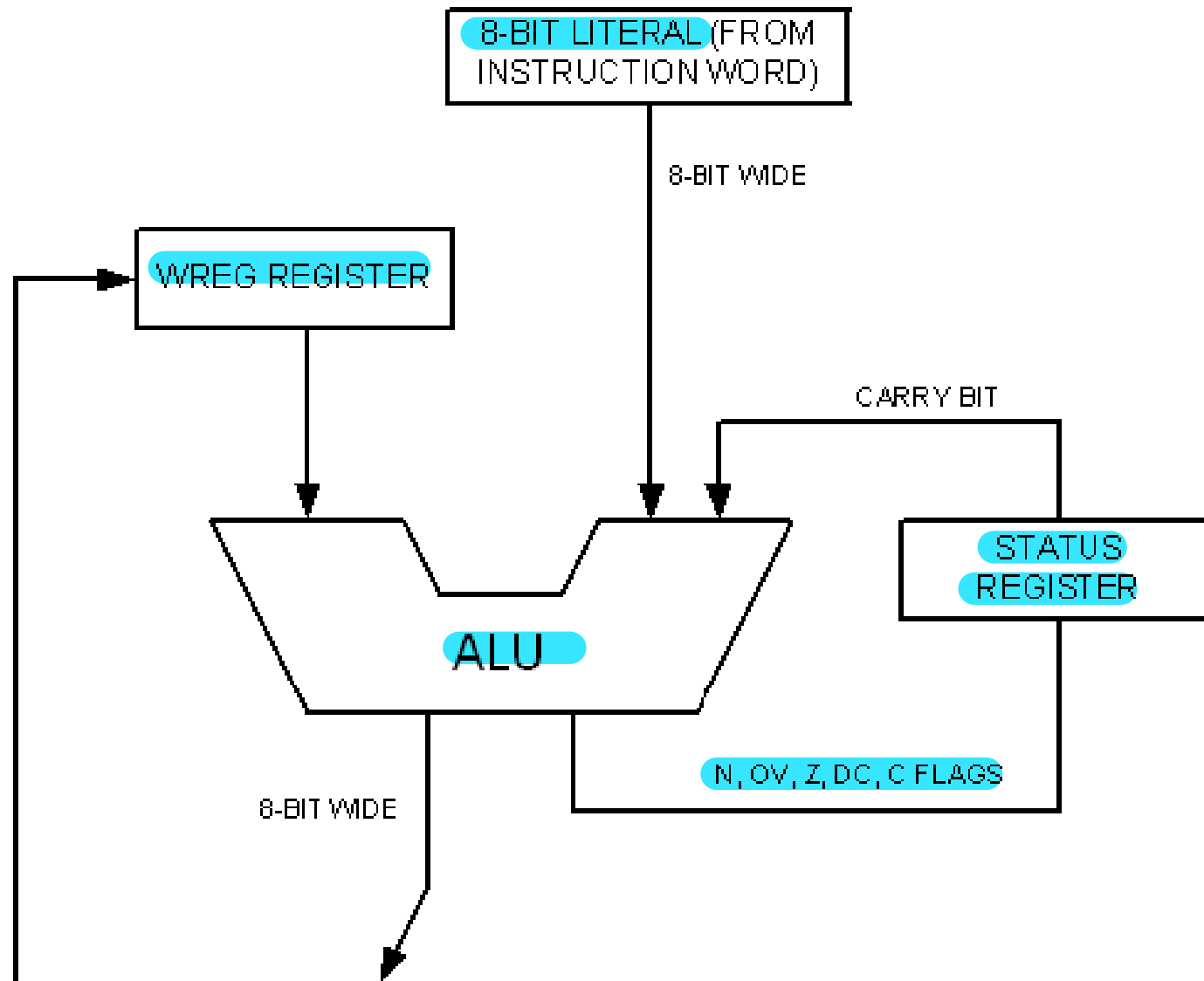
Example

      MOVLW 25H

      MOVLW 0A5H

Is the following code correct?

MOVLW 9H   yes

MOVLW A23H   0A23H

ADDLW

8-BIT LITERAL (FROM INSTRUCTION WORD)

8-BIT WIDE

WREG REGISTER

CARRY BIT

STATUS REGISTER

ALU

N, OV, Z, DC, C FLAGS

8-BIT WIDE

ADDLW

ADDLW k; add literal value k to WREG (k +WREG)

Example: WREG

    MOVLW 12H ;

0	0	0	1	0	0	1	0

    ADDLW 16H ;

0	0	1	0	1	0	0	0

    ADDLW 11H ;

0	0	1	1	1	0	0	1

    ADDLW 43H ;

0	1	1	1	1	1	0	0

The PIC File register (data memory)

- Used for data storage, scratch pad and registers for internal use and function

- 8-bit width
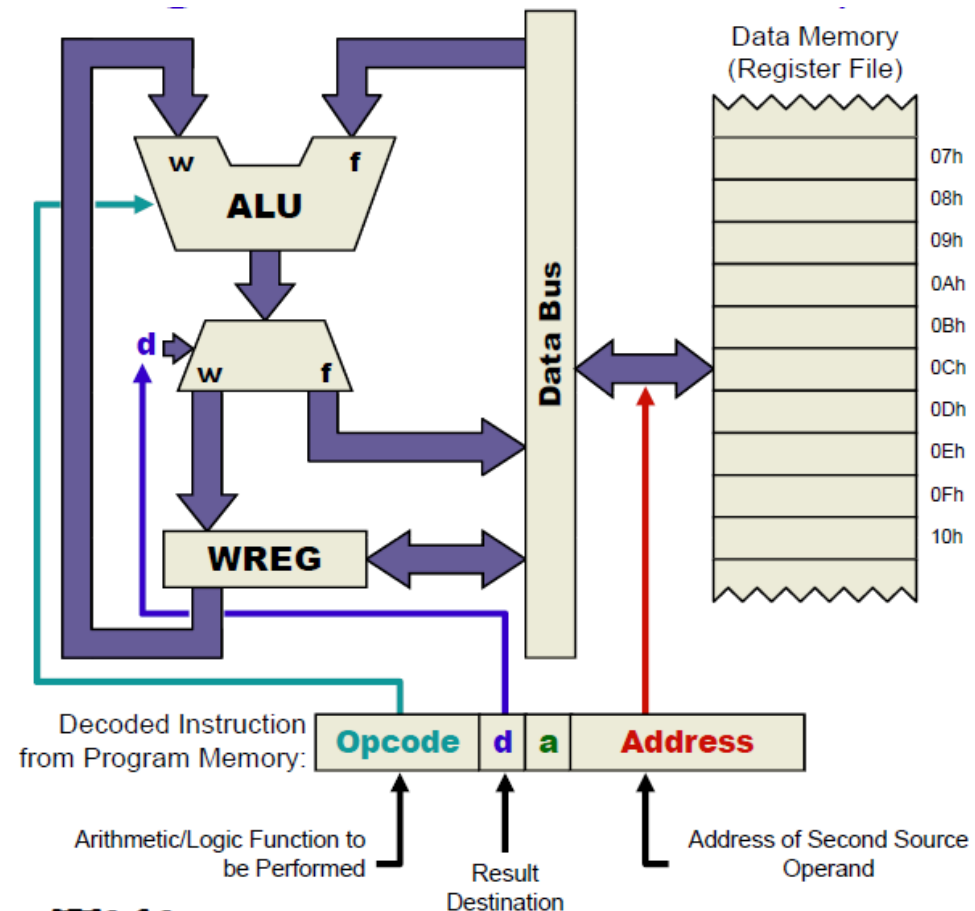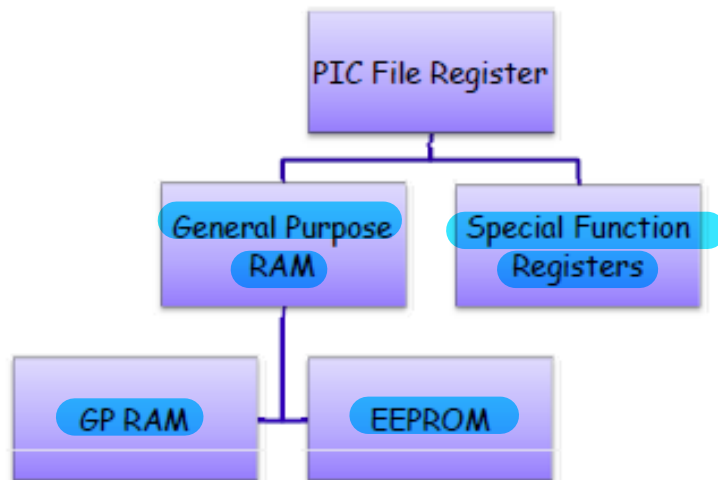
Special-Function Registers

- dedicated to specific functions such as ALU status, timers, serial communication, I/O ports, ADC,…
- They are used for control of the microcontroller or peripheral
- 8-bit registers
- Their numbers varies from one chip to another

General-Purpose Registers

- Group of RAM locations
- 8-bit registers
- Larger than SFR 128

    Difficult to manage them by using Assembly language

    Easier to handle them by C Compiler.

GPRAM vs. EEPROM

    EEPROM: an add-on memory (for holding data after power off), can be zero size
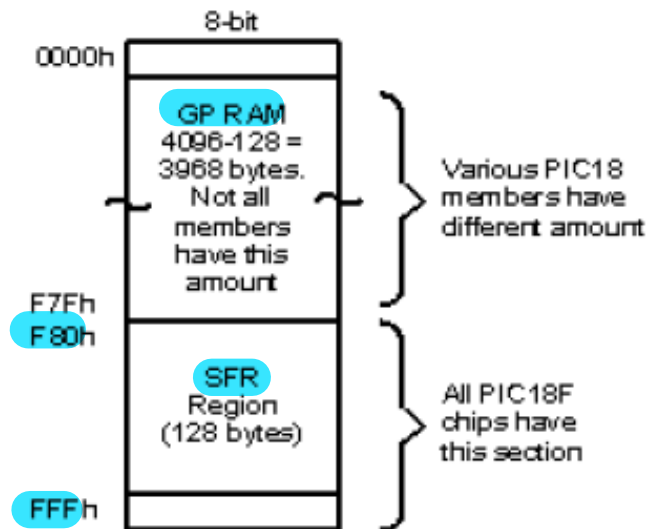
File Register Size

	File Register	=	SFR	+	GPR
	(Bytes)		(Bytes)		(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

File Register and access bank in PIC18

- The PIC18 Family can have a maximum of 4096 bytes.
- The File Register

    has addresses of 000- FFFH

    divided into 256-byte banks

    maximum 16 banks (**Why?**)

- At least there is one bank

☐ Known as default access bank.
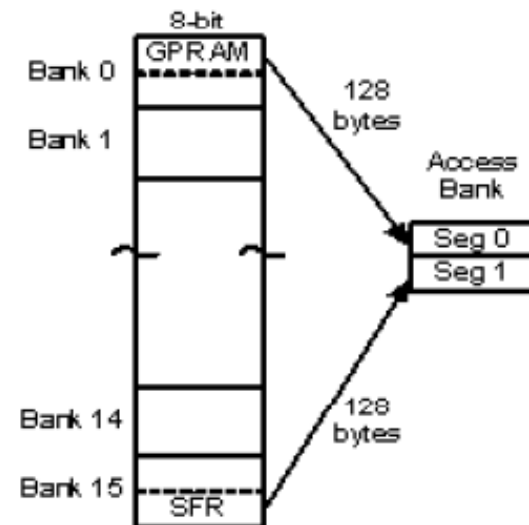
- Bank switching is a method used to access all the banks
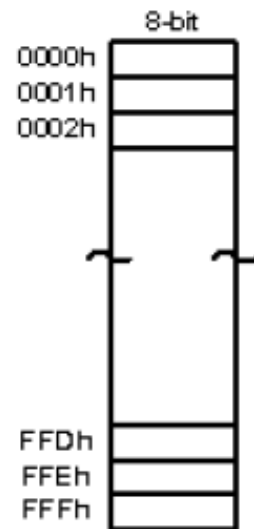
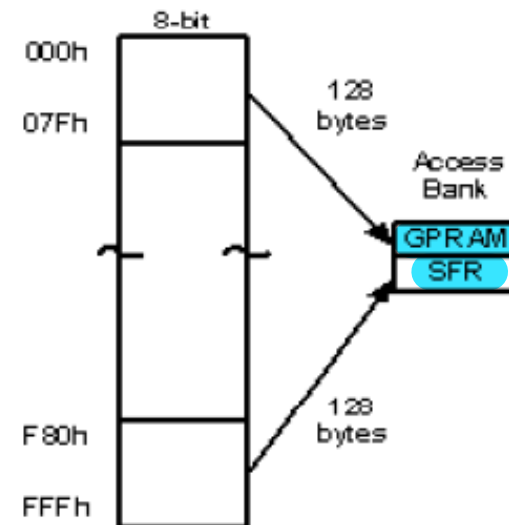a) Maximum space of file register (data RAM) in PIC18F (4096 byte)

b) File register allocation between GP RAM and SFR

c) Data memory map

d) Access Bank

Access bank in PIC18

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

SFRs of PIC18

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	

Using instructions with the default access bank

Instructions to access other locations in the file register for ALU and other operations.

- MOVWF   ADDWF
                       INCF
- COMF
- DECF
- MOVF
- MOVFF

MOVWF instruction

- F indicates a file register

  MOVWF Address

- It tells the CPU to copy the source, WREG, to a destination in the file register.

 a location in the SFR

 a location in GP RAM

MOVWF instruction

WREG

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

Data Memory

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 GP RAM location in the PIC18.

They must be **moved there via WREG.**

ADDWF instruction

- Add together the contents 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 Reg. location

  D indicates the destination bit

- If D=0 or (D=w)

  The result will be placed in WREG

- If D=1 or (D=f)

  The result will be placed in File Reg.

```
MOVLW        22H      ;WREG=22H
MOVWF        5H       ;copy WREG contents to location 5H
MOVWF        6H       ;copy WREG contents to location 6H
MOVWF        7H       ;copy WREG contents to location 7H
ADDWF        5H, 0    ;add W and loc 5, put result in WREG
                      ; WREG=44H
ADDWF        6H, 0    ;add W and loc 6, put result in WREG
                      ; WREG=66H
ADDWF        7H, 0    ;add W and loc 7, put result in WREG
                      ; WREG=88H
```

Address	Data
005	22
006	22
007	22

GPR after the execution up to
"ADDWF 7H, 0"
WREG = 88H

```
MOVLW       22H     ;WREG=22H
MOVWF       5H      ;copy WREG contents to location 5H
MOVWF       6H      ;copy WREG contents to location 6H
MOVWF       7H      ;copy WREG contents to location 7H
ADDWF       5H, 0   ;add W and loc 5, put result in WREG
                    ; WREG=44H
ADDWF       6H, 0   ;add W and loc 6, put result in WREG
                    ; WREG=66H
ADDWF       7H, 1   ;add W and loc 7, put result in
                    ;loc 7, content of loc 07 = 88H,
                    ;WREG=66.
```

Address	Data
005	22
006	22
007	88

GP RAM after the execution up to "ADDWF 7H, 1"

WREG = 66H

COMF instruction

COMF File Reg. Address, D

- It tells the CPU to complement the contents of File Reg. and place the result in WREG or in File Reg.

D indicates the destination bit

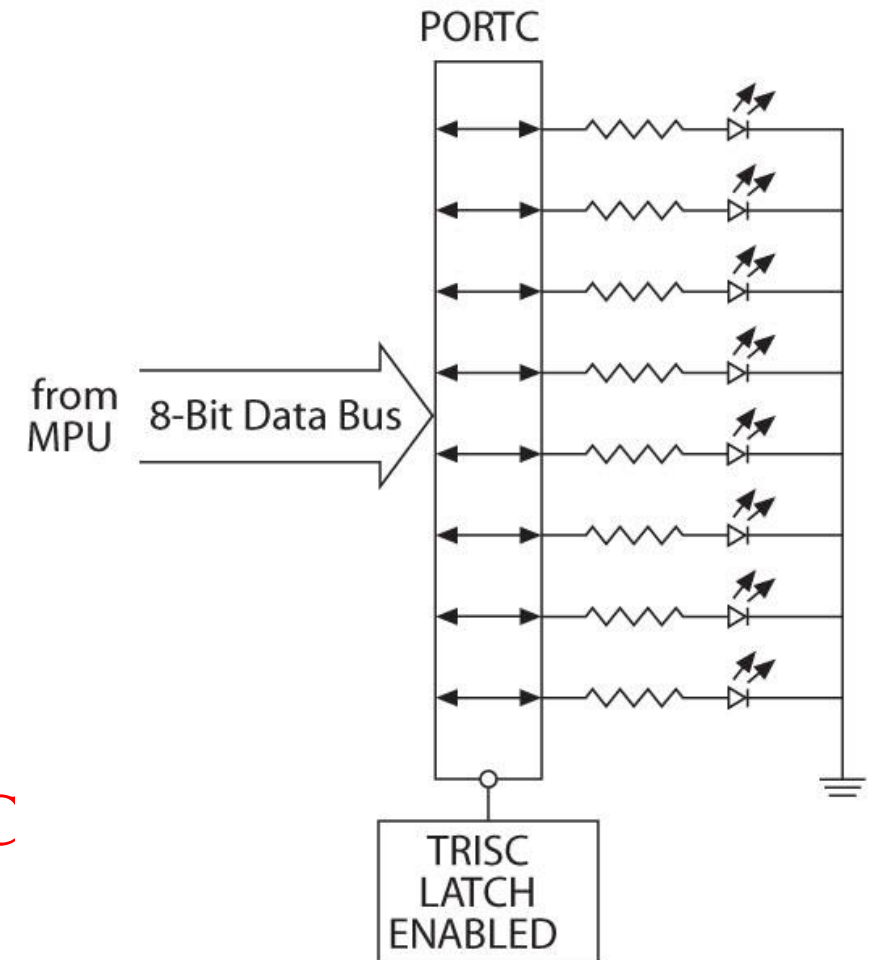- If D=0 or (D=w)

  The result will be placed in WREG

- If D=1 or (D=f)

  The result will be placed in File Reg.

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

```
        Solution:

        MOVLW         00
        MOVWF         TRISC,0    access back
        MOVLW         55H        01010101
        MOVWF         PORTC
B1:     COMF          PORTC,F
        GOTO B1
```

What are the values (in hexadecimal) in Port C

55 -> AA -> ...
H'55' = B'01010101' -> H'AA' = B'10101010'

What do you see if Port C is connected to 8 LEDs?

DECF (INCF) instruction

DECF File Reg. Address, D

- It tells the CPU to decrement the content of File Reg. and place the result in WREG or in File Reg.

```
MOVLW     3      ;WREG=3
MOVWF     20H    ;20H=(3)
DECF  20H,  F    ;WREG=3,  20H=(2)
DECF  20H,  F    ;WREG=3,  20H=(1)
DECF  20H,  F    ;WREG=3,  20H=(0)
```

```
MOVLW     3       ;WREG=3
MOVWF     20H    ;20H=(3)
DECF 20H,  W     ;WREG=2,  20H=(3)
DECF 20H,  W     ;WREG=2,  20H=(3)
DECF 20H,  W     ;WREG=2,  20H=(3)
```

source: 20H = 3, which did not change

INCF is logically the same

MOVF instruction

MOVF File Reg. Address, D

It is intended to copy contents of File Reg. to WREG

- If D=0

    copy contents of File Reg. (from I/O pin) to WREG

- If D=1

    contents of File Reg. are copied to itself

    (Why do this?)

Write a simple program to get data from SFRs of Port B and send it to SFRs of PORT C continuously.
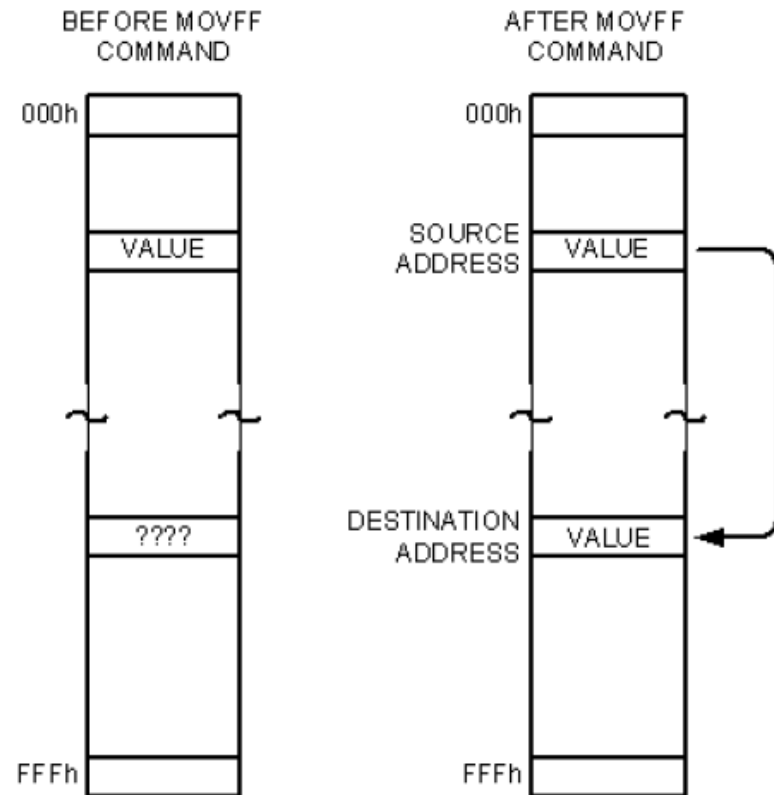
```
Solution:

Again:      MOVF        PORTB,  W
            MOVWF       PORTC
            GOTO        Again
```

MOVFF instruction

Copy contents of one location in File Reg. to another location in File Reg.

MOVFF source File Reg, destination File Reg

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

```
Solution:

                        32 bit long
Again:          MOVFF        PORTB, PORTC
                GOTO         Again
```
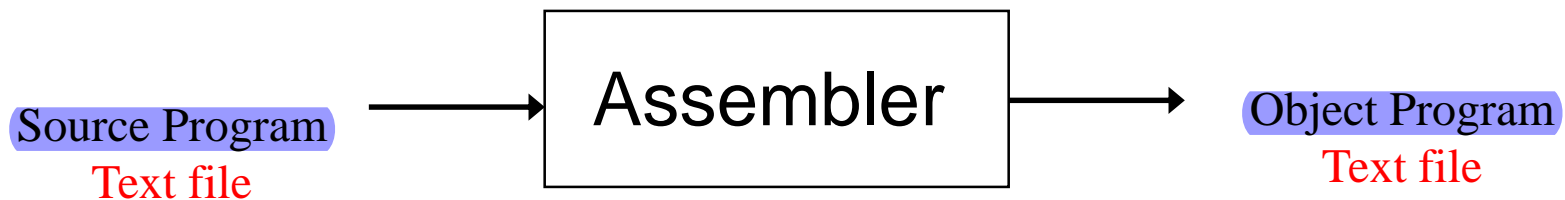
3.1.3 structure of PIC18 assembly language program

- programming language

- directives

- Machine language：
  - a program that consists of 0 and 1.
  - CPU can work on machine language directly.
  - Example： 7D25
- Low-level language：
  - It deals directly with the internal structure of the CPU.
  - Programmers must know all details of the CPU.
  - Example： MOVFF 20H, 21H
- High-level language：
  - Machine independent
  - Example： a=37; （C++）

- Assembly languages were developed which provided mnemonics for the machine code instructions, plus other features.
  - Mnemonic： the instruction
    - Example： MOVFF, MOVLW
  - Provide decimal number, named registers, label, command
  - programming faster and less prone to error
- Assembly language programs must be translated into machine code by a program called an assembler.

- Assembler：
  - a software program can translate an assembly language program into machine code
  - source program
  - object program, object code (opcode)

Source Program
Text file

→ Assembler →

Object Program
Text file

Structure of Assembly Language

- An assembly language program is a series of statements.

  [label:]   mnemonic   [operands]   [;comment]

  – Brackets indicate that a field is optional.
  – Label is the name to refer to a line of program code. A label referring to an instruction must be followed by a ":".

  ```
  Here:   GOTO Here
  ```

  – Mnemonic and operand(s) perform the real work of the program.
  – The comment field begins with ";".

- Two types of assembly instructions：
  - Mnemonic： tell the CPU what to do
    - Example： MOVFF, ADDLW   (opcodes)
  - pseudo-instruction： give directions to the assembler
    - Example： ORG 0H, END
    - Pseudo-instruction is called directives

ORG & END

- ORG tells the assembler to place the opcode at ROM with a chosen start address.

    ORG start-address

    ```
    ORG    0200H    ;put the following codes
                    ;start at location 200H
    ```

- END indicates to the assembler the end of the source code.

    END

    ```
    END             ;end of asm source file
    ```

EQU and SET

like a global constant

- EQU – associates a constant number with an address label.

- e.g., `COUNT` `equ` `0x25`

    . . . .

    `movlw` `COUNT`; WREG = 0x25   usage -> change count -> change all value in the program

- SET – identical to EQU, but value assigned by SET can be reassigned later.

- e.g., `COUNT set 0x00`

    0x00

    0x00

    . . . .

    0x00

    `COUNT set 0x25`

    0x25

    0x25

    . . . .

    0x25

    `movlw COUNT; WREG = 0x25`

e.g., Move 22H into two file registers with addresses 0x05 and 0x06, add the contents of all three registers and put the result in WREG:

Without EQU

```
movlw 0x22
movwf 0x05
movwf 0x06
addwf 0x05, W
addwf 0x06, W
```

With EQU

```
FirstReg EQU 0x05
SecReg   EQU 0x06
movlw 0x22
movwf FirstReg
movwf SecReg
addwf FirstReg, W
addwf SecReg, W
```

Which method is easier to change the file registers?

CBLOCK

- Defines a list of named constants.
- Format: `cblock <num>` starting value

    `<constant label> [:<inc>]`

    `endc`

- e.g. 1:

    ```
    cblock 0x50
        test1, test2, test3, test4
    endc
    ```

- Values Assigned:

    test1 = 0x50, test2 = 0x51,

    test3 = 0x52, test4 = 0x53.

- e.g. 2:

```
cblock 0x30
    twoBytesValue: 0, twoByteHi, twoByteLo
    queue: d'40'
    queuehead, queuetail
    double1: 2, double2: 2
endc
```

- Value Assigned:

twoBytesValue = 0x30, twoByteHi = 0x30

twoByteLo = 0x31, queue = 0x32

queuehead = 0x5A, queuetail = 0x5B

double1 = 0x5C, double2 = 0x5E

Sample of an Assembly Language Program

```
         LIST P=18F4520        ;directive to define processor
#include <P18F4520.INC>        ;CPU specific variable
                               ;definitions
   lable
         SUM EQU 10H           ; RAM loc 10H for 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
```

Summary

♦ review PIC18 architecture – major components, programming model

♦ some data types and instructions

♦ access bank and file registers

♦ directives

♦ assembly language program

3.1.4 assemble and run PIC18 program

- enter assembly language program, create project, build and run the program on IDE (refer to section 2.5 and Tutorial 2 "MPLAB IDE")

- next, you can choose Debugger, e.g. PICKit 3, to run the program on the development kit (refer to Tutorial 3 "upload program")

```
        LIST  P=18F4520            ;directive to define processor
    #include <P18F4520.INC>  ;CPU specific variable
                                  ;definitions
            SUM EQU 10H            ; RAM loc 10H for SUM
            ORG 0H                ; start at address 0
            MOVLW 25H             ; WREG = 25
            ADDLW 0x34            ; WREG = WREG + 34H = 59H
            ADDLW 11H             ; WREG = WREG + 11H = 6AH
            ADDLW d'18'           ; WREG = WREG + 12H = 7CH
            ADDLW 1CH             ; WREG = WREG + 1CH = 98H
            ADDLW b'00000110'     ; WREG = WREG + 6H = 9EH
            MOVWF SUM             ; save the result in SUM location
HERE:  GOTO HERE                 ; stay here forever
            END                   ; end of asm source file
```
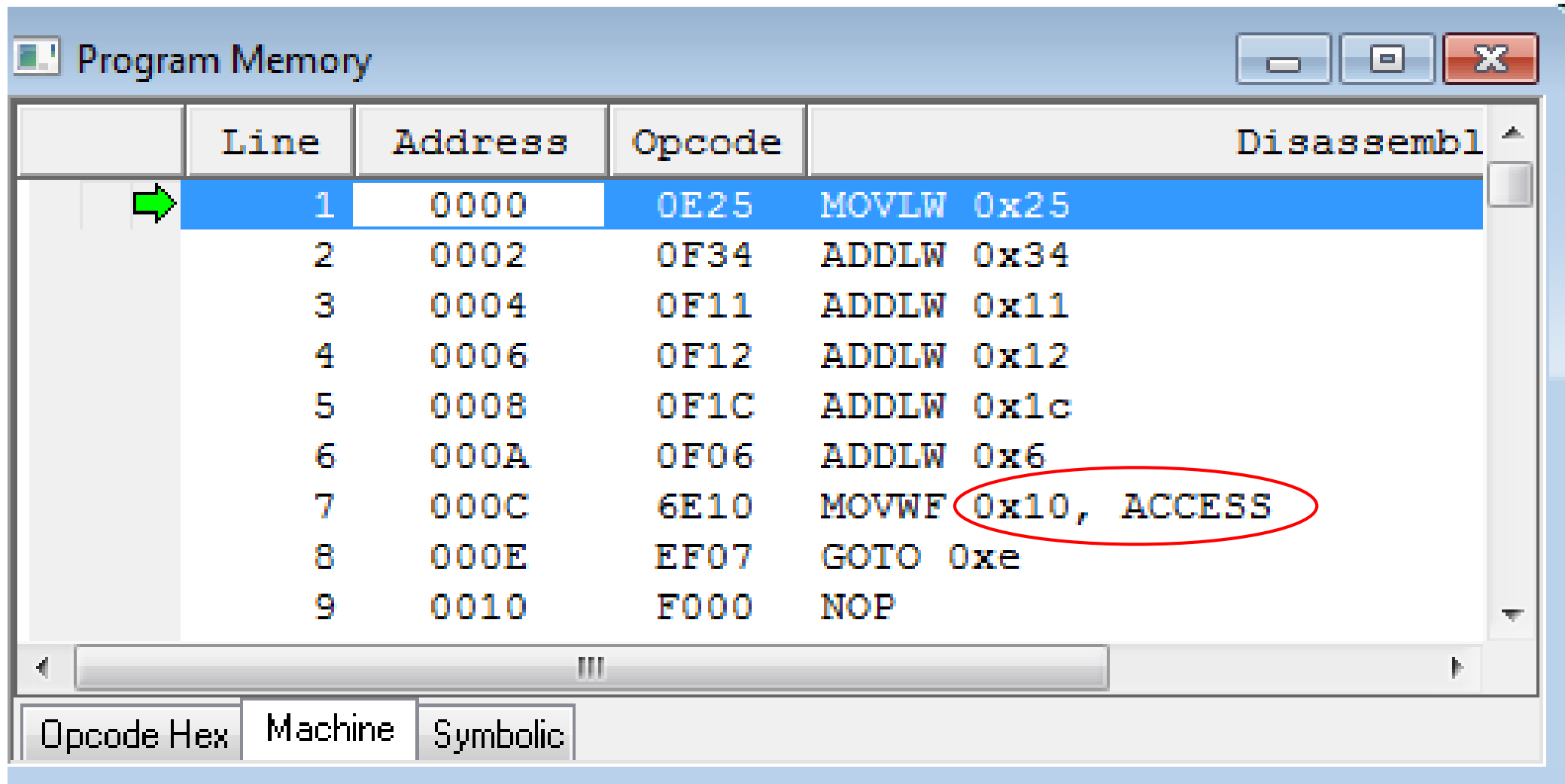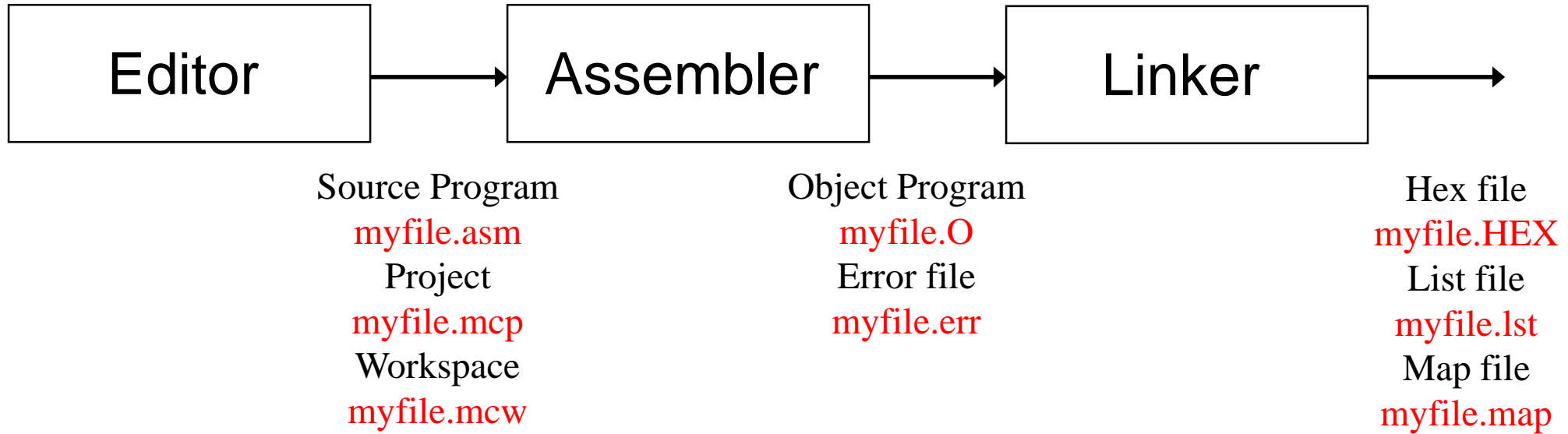
observe Program Counter, machine code

	Line	Address	Opcode	Disassembl
➡	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 0x10, ACCESS
	8	000E	EF07	GOTO 0xe
	9	0010	F000	NOP

Opcode Hex | Machine | Symbolic

Editor	→	Assembler	→	Linker	→

Source Program
myfile.asm
Project
myfile.mcp
Workspace
myfile.mcw

Object Program
myfile.O
Error file
myfile.err

Hex file
myfile.HEX
List file
myfile.lst
Map file
myfile.map

```
LOC     OBJECT CODE   LINE    SOURCE TEXT                      VALUE


                      00001   LIST    P=18F4520 ;directive to define processor
                      00002   #include <P18F4520.INC> ;

0000010               00004   SUM:    EQU 10H                  ; RAM loc 10H fro SUM
000000                00005           ORG 0H                   ; start at address 0
000000 0E25           00006           MOVLW 25H                ; WREG = 25
000002 0F34           00007           ADDLW 0x34               ; add 34H to WREG=59H
000004 0F11           00008           ADDLW 11H                ; add 11H to WREG=6AH
000006 0F12           00009           ADDLW d'18'              ; W = W+12H=7CH
000008 0F1C           00010           ADDLW 1CH                ; W = W+1CH=98H
00000A 0F06           00011           ADDLW b'00000110'        ; W = W+6H=9EH
00000C 6E10           00012           MOVWF SUM                ; save the result in SUM location
00000E EF07 F000      00013   HERE:   GOTO HERE                ; stay here forever
                      00014           END                      ; end of asm source file
```
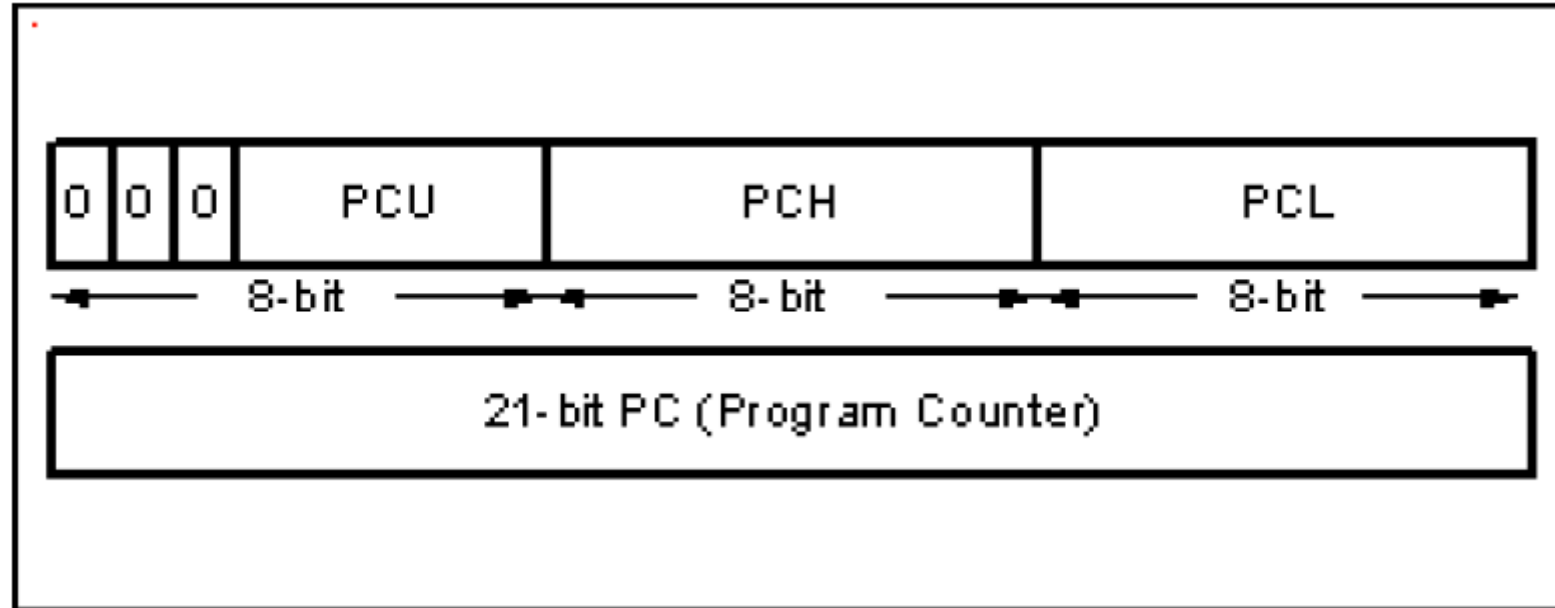
3.1.5 ROM space

- Program Counter (PC) is used by the CPU to point to the address of the next instruction to be executed (*increase automatically*)
- The wider the program counter, more the memory locations can be accessed

  PIC16 has 14 bits (16 KB)

  PIC18 has 21 bits (2 MB)

  8051 has 16 bits (64 KB)

PIC18 PC

PIC18 Microcontroller Family

| Product	Program Memory Type	Program Memory Bytes	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	I^{2}C/SPI	1	6x PMM	2	1/3	DIP, SOIC	28
PIC18F2320	FLASH	8K	512	256	23	10	I^{2}C/SPI	1	6x PMM	2	1/3	DIP, SOIC	28
PIC18C242	OTP	16K	512	—	23	5	I^{2}C/SPI	1	—	2	1/3	DIP, SOIC	28
PIC18C252	OTP	32K	1536	—	23	5	I^{2}C/SPI	1	—	2	1/3	DIP, SOIC	28
PIC18F242	FLASH	16K	512	256	23	5	I^{2}C/SPI	1	—	2	1/3	DIP, SOIC, SSOP	28
PIC18F252	FLASH	32K	1536	256	23	5	I^{2}C/SPI	1	—	2	1/3	DIP, SOIC, SSOP	28
PIC18F258	FLASH	32K	1536	256	22	5	I^{2}C/SPI	1	CAN 2.0B	1	1/3	DIP, SOIC	28
PIC18F4220	FLASH	4K	512	256	34	13	I^{2}C/SPI	1	6x PMM	2	1/3	DIP, TQFP, QFN	40/44
PIC18F4320	FLASH	8K	512	256	34	13	I^{2}C/SPI	1	6x PMM	2	1/3	DIP, TQFP, QFN	40/44
PIC18C442	OTP	16K	512	—	34	8	I^{2}C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18C452	OTP	32K	1536	—	34	8	I^{2}C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F442	FLASH	16K	512	256	34	8	I^{2}C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F452	FLASH	32K	1536	256	34	8	I^{2}C/SPI	1	—	2	1/3	DIP, PLCC, TQFP	40/44
PIC18F458	FLASH	32K	1536	256	33	5	I^{2}C/SPI	1	CAN 2.0B	1	1/3	DIP, PLCC, TQFP	40/44
PIC18C601	—	ROMless	1536	—	31	8	I^{2}C/SPI	1	—	2	1/3	PLCC, TQFP	64/68
PIC18C658	OTP	32K	1536	—	52	12	I^{2}C/SPI	1	CAN 2.0B	2	1/3	PLCC, TQFP	64/68
PIC18F6520	FLASH	32K	2048	1024	52	12	I^{2}C/SPI	2	—	5	2/3	TQFP	64
PIC18F6620	FLASH	64K	3840	1024	52	12	I^{2}C/SPI	2	—	5	2/3	TQFP	64
PIC18F6720	FLASH	128K	3840	1024	52	12	I^{2}C/SPI	2	—	5	2/3	TQFP	64
PIC18C801	—	ROMless	1536	—	42	12	I^{2}C/SPI	1	—	2	1/3	PLCC, TQFP	80/84
PIC18C858	OTP	32K	1536	—	68	16	I^{2}C/SPI	1	CAN 2.0B	2	1/3	PLCC, TQFP	80/84
PIC18F8520	FLASH	32K	2048	1024	68	16	I^{2}C/SPI	2	EMA	5	2/3	TQFP	80
PIC18F8620	FLASH	64K	3840	1024	68	16	I^{2}C/SPI	2	EMA	5	2/3	TQFP	80
PIC18F8720	FLASH	128K	3840	1024	68	16	I^{2}C/SPI	2	EMA	5	2/3	TQFP	80

Abbreviation: ADC = Analog-to-Digital Converter   CCP = Capture/Compare/PWM   I^{2}C = Inter-Integrated Circuit Bus   PMM = Power Managed Mode
PWM = Pulse Width Modulation   SPI = Serial Peripheral Interface   USART = Universal Synchronous/Asynchronous Receiver/Transmitter

Find the ROM size for each of the following PIC chips:
a) PIC18F2220   4KB
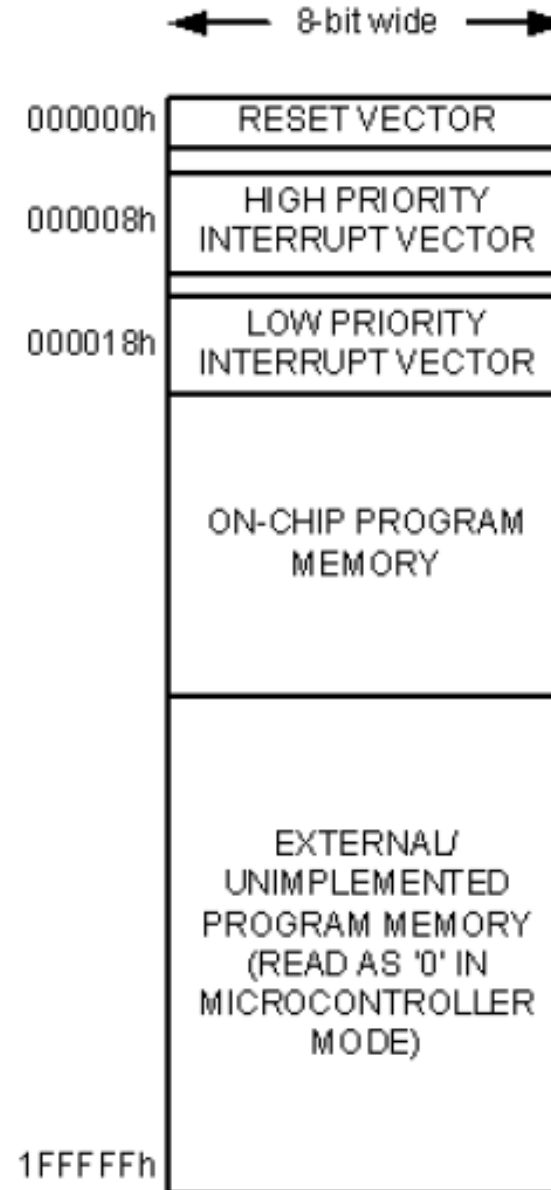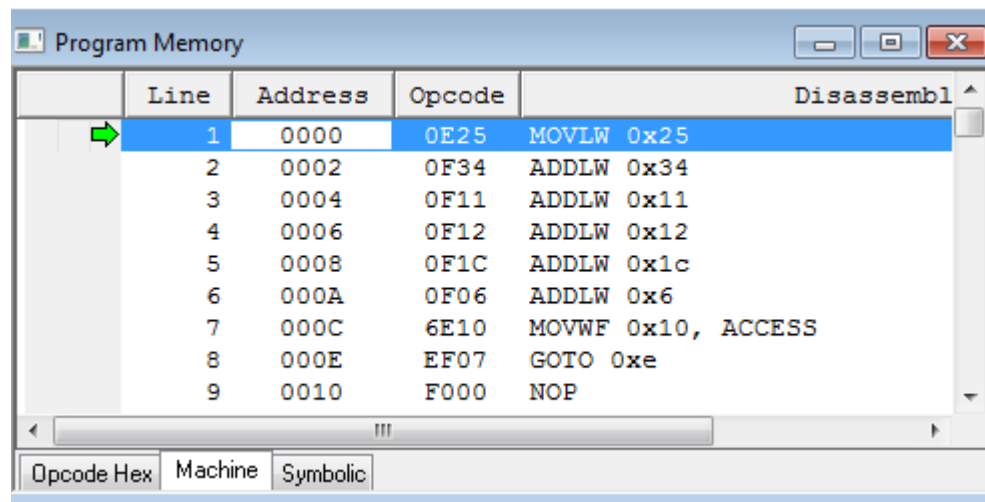b) PIC18F2410   16KB
c) PIC18F458   32KB
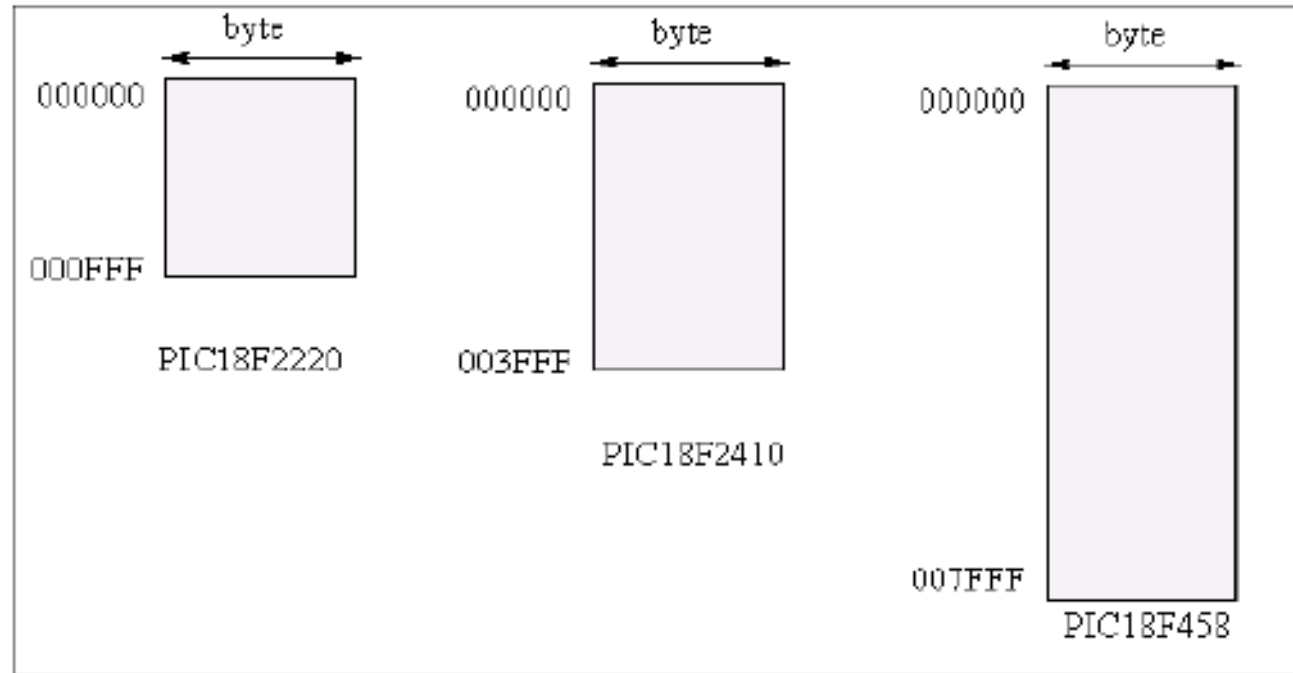
Place program code in ROM

At what address does the CPU wake up when power is applied?
- PIC18 wakes up at memory address 000000H
- PC has the value 000000H
- ORG directive put the address of the first opcode at the memory location 000000H

```
LOC     OBJECT CODE LINE SOURCE TEXT  VALUE
                      00001  LIST   P=18F4520  ;directive to define processor
                      00002   #include <P18F4520.INC> ;

0000010               00004 SUM:    EQU 10H ;RAM loc 10H for SUM
000000                00005         ORG 0H; start at address 0
000000 0E25           00006         MOVLW 25H ; WREG = 25
000002 0F34           00007         ADDLW 0x34 ;add 34H to WREG=59H
000004 0F11           00008         ADDLW 11H ;add 11H to WREG=6AH
000006 0F12           00009         ADDLW d'18' ; W = W+12H=7CH
000008 0F1C           00010         ADDLW 1CH ; W = W+1CH=98H
00000A 0F06           00011         ADDLW b'00000110' ; W = W+6H=9EH
00000C 6E10           00012         MOVWF SUM ;save the result in SUM location
00000E EF07 F000 00013 HERE:    GOTO HERE ;stay here forever
                      00014         END ; end of asm source file
```

If we change ORG 0H to ORG 20H, we have

```
LOC    OBJECT CODE      LINE SOURCE TEXT        VALUE

00000010                00004 SUM:     EQU 10H ;RAM loc 10H for SUM
000000                  00005          ORG 00H; start at address 0
000000 EF10 F000        00006          GOTO Main
000020                  00007 Main:    ORG 20H
000020 0E25             00008          MOVLW 25H ; WREG = 25
000022 0F34             00009          ADDLW 0x34 ;add 34H to WREG=59H
000024 0F11             00010          ADDLW 11H ;add 11H to WREG=6AH
000026 0F12             00011          ADDLW d'18' ; W = W+12H=7CH
000028 0F1C             00012          ADDLW 1CH ; W = W+1CH=98H
00002A 0F06             00013          ADDLW b'00000110' ; W = W+6H=9EH
00002C 6E10             00014          MOVWF SUM ;save the result in SUM location
00002E EF17 F000        00015 HERE:    GOTO HERE ;stay here forever
                        00016          END ; end of asm source file
```

Program ROM width

- Byte addressable: each location holds only one byte
  8-bit CPU will fetch one byte a time
  Increasing the data bus width will bring more information
- Solution: data bus between CPU and ROM is similar to traffic lanes
  on the highway
- 16-bit data bus
  Increase the processing power
  PIC18 can fetch instruction in 1 cycle
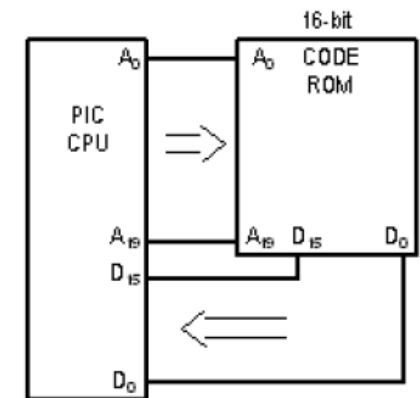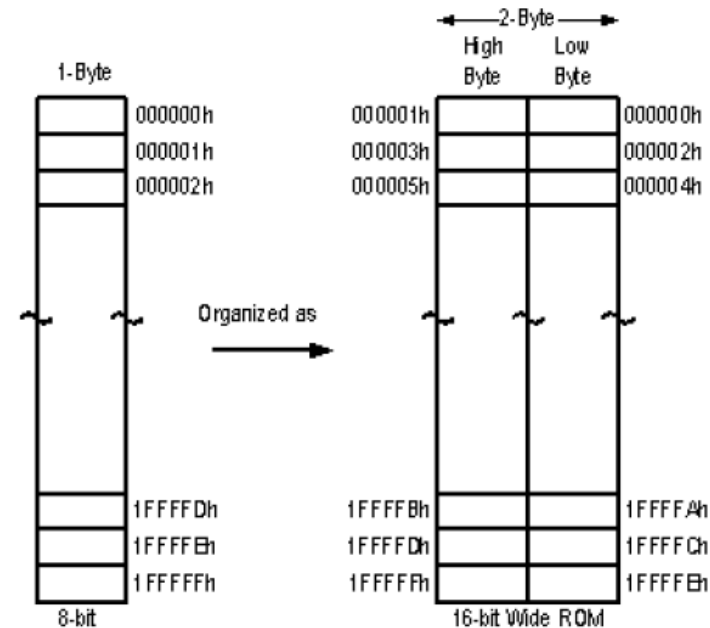
Instruction size of PIC18

- PIC instructions are 2-Byte or 4-Byte
- Most of PIC18 instructions are 2-Byte
- The first seven or eight bits represents the opcode
- The following eight bits is the operand

```
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 = 0 specifies the default access bank
- a = 1 specifies other bank

- 4-Byte instructions include

GOTO ^{EF}

$1110\ 1111$ $k_7kkk\ kkkk_0$

1111 $k_{19}kkk\ kkkk\ kkkk_8$

12-bit opcode, 20-bit target address ($0 \le k \le FFFFF$)
But address is 21-bit!

MOVFF (move data within RAM)

$1100\ ssss\ ssss\ ssss$ ($0 \le s \le FFF$)

$1111\ dddd\ dddd\ dddd$ ($0 \le d \le FFF$)

First 16-bit for opcode and address of source file register
Second 16-bit for opcode and address of destination file register

3.1.6 flags and status register

- when CPU performs operations, sometimes an exception may occur, e.g. overflow

- How does the CPU tells control unit that an exception occurs?

- Answer - flags
  - C   Carry Flag
  - DC   Digital Carry Flag
  - Z   Zero Flag
  - OV   Overflow Flag
  - N   Negative Flag

Effect of addlw on the status register

Example 1:

```
MOVLW 0x38
ADDLW 0x2F
```

$$38h$$
$$+\ 2Fh$$
$$\overline{}$$
$$67h$$

N = 0 ; bit7=0

OV=0 ; +ve + +ve = + ve => no overflow (in signed sense)

Z=0   ; NOT all zeros

DC=1 ; A carry from the first to second nibble

C=0   ; No carry

Example 2:

```
MOVLW 0x9C
ADDLW 0x64
```

$$\begin{array}{r} 9Ch \\ + 64h \\ \hline 00h \end{array}$$

N = 0 ; bit7=0

OV=0 ; -ve + +ve = +ve => no overflow (in signed sense)

Z=1   ; All zeros

DC=1 ; A carry from the first to second nibble

C=1   ; A carry is generated (in unsigned sense)

Example 3:

```
MOVLW 0x80
ADDLW 0x81
```

$$80\text{h (in signed sense -128)}$$
$$+\ 81\text{h (in signed sense -127)}$$
$$\overline{\qquad\qquad\qquad\qquad\qquad}$$
$$01\text{h}$$

N = 0 ; bit7=0

OV=1 ; -ve + -ve = +ve => overflow (in signed sense)

Z=0   ; NOT all zeros

DC=0 ; No carry from the first to second nibble

C=1   ; A carry is generated (in unsigned sense)

Example 4:

```
MOVLW 0x7F
ADDLW 0x7F
```

$$
\begin{array}{r}
\mathbf{7Fh} \\
+ \ \mathbf{7Fh} \\
\hline
\mathbf{FEh}
\end{array}
$$

N = 1 ; bit7=1

OV=1 ; +ve + +ve = -ve => overflow (in signed sense)

Z=0   ; NOT all zeros

DC=1 ; A carry from the first to second nibble

C=0   ; No carry

Instructions That Affect Flag Bits

Instruction	C	DC	Z	OV	N
ADDLW	X	X	X	X	X
ADDWF	X	X	X	X	X
ADDWFC	X	X	X	X	X
ANDLW			X		X
ANDWF			X		X
CLRF			X		
COMF			X		X
DAW	X				
DECF	X	X	X	X	X
INCF	X	X	X	X	X
IORLW			X		X
IORWF			X		X
MOVF			X		
NEGF	X	X	X	X	X
RLCF	X		X		X
RLNCF			X		X
RRCF	X		X		X
RRNCF			X		X
SUBFWB	X	X	X	X	X
SUBLW	X	X	X	X	X
SUBWF	X	X	X	X	X
SUBWFB	X	X	X	X	X
XORLW			X		X
XORWF			X		X

Flag Bits and Decision Making

Status flags are also called conditions, there are instructions that will make a conditional Jump (branch) based on the status of the flag.

Instruction	Action
BC	Branch if C = 1
BNC	Branch if C = 0
BZ	Branch if Z = 1
BNZ	Branch if Z = 0
BN	Branch if N = 1
BNN	Branch if N = 0
BOV	Branch if OV = 1
BNOV	Branch if OV = 0

3.1.7 register banks

So far, we only consider the access bank in data memory. Actually, we can choose other banks
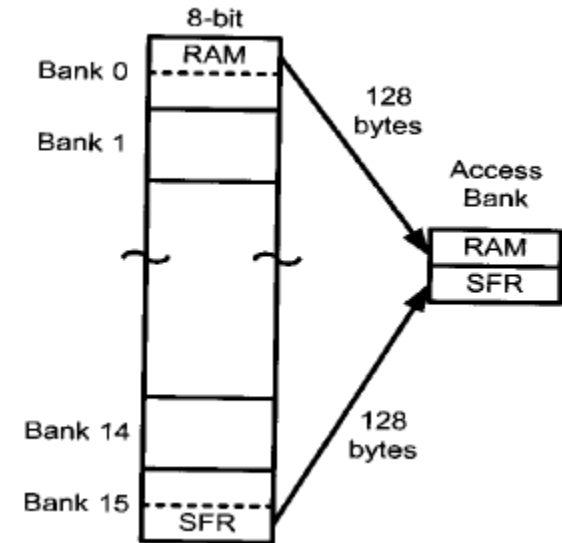
INCF MYREG, D, A

MOVWF MYREG, A

A=0, the access bank

A=1, other bank

To use this feature,

Load BSR with the desired bank number

Make A=1 in the instruction

Example:

```
MYREG EQU 0x40 ;define a location
      MOVLB 0x2        ;load 2 into BSR (use bank 2)
      MOVLW 0          ; WREG=0
      MOVWF MYREG, 1 ; loc (240)=0, WREG=0, A=1
      INCF  MYREG,F,1; loc (240)=1, WREG=0, A=1
      INCF  MYREG,F,1; loc (240)=2, WREG=0, A=1
      INCF  MYREG,F,1; loc (240)=3, WREG=0, A=1
```

Example:

```
MYREG EQU 0x40 ;define a location
MOVLB 0x2       ;load 2 into BSR (use bank 2)
MOVLW 0         ; WREG=0
MOVWF MYREG     ; loc (40)=0, WREG=0, A=0
INCF  MYREG,F   ; loc (40)=1, WREG=0, A=0
INCF  MYREG,F   ; loc (40)=2, WREG=0, A=0
INCF  MYREG,F   ; loc (40)=3, WREG=0, A=0
```

Summary

♦ procedure to assemble and run PIC18 program

♦ program code in ROM

♦ status of flag bits in instruction execution

♦ selection of register bank