Chapter 2
Introduction to PIC18 Microcontroller

2.1 PIC Architecture

PIC Architecture

- Harvard Architecture which includes:
  - CPU
  - Program memory for instructions
  - Data memory for data
  - I/O ports
  - Support devices such as timers

PIC18 Memory Organization

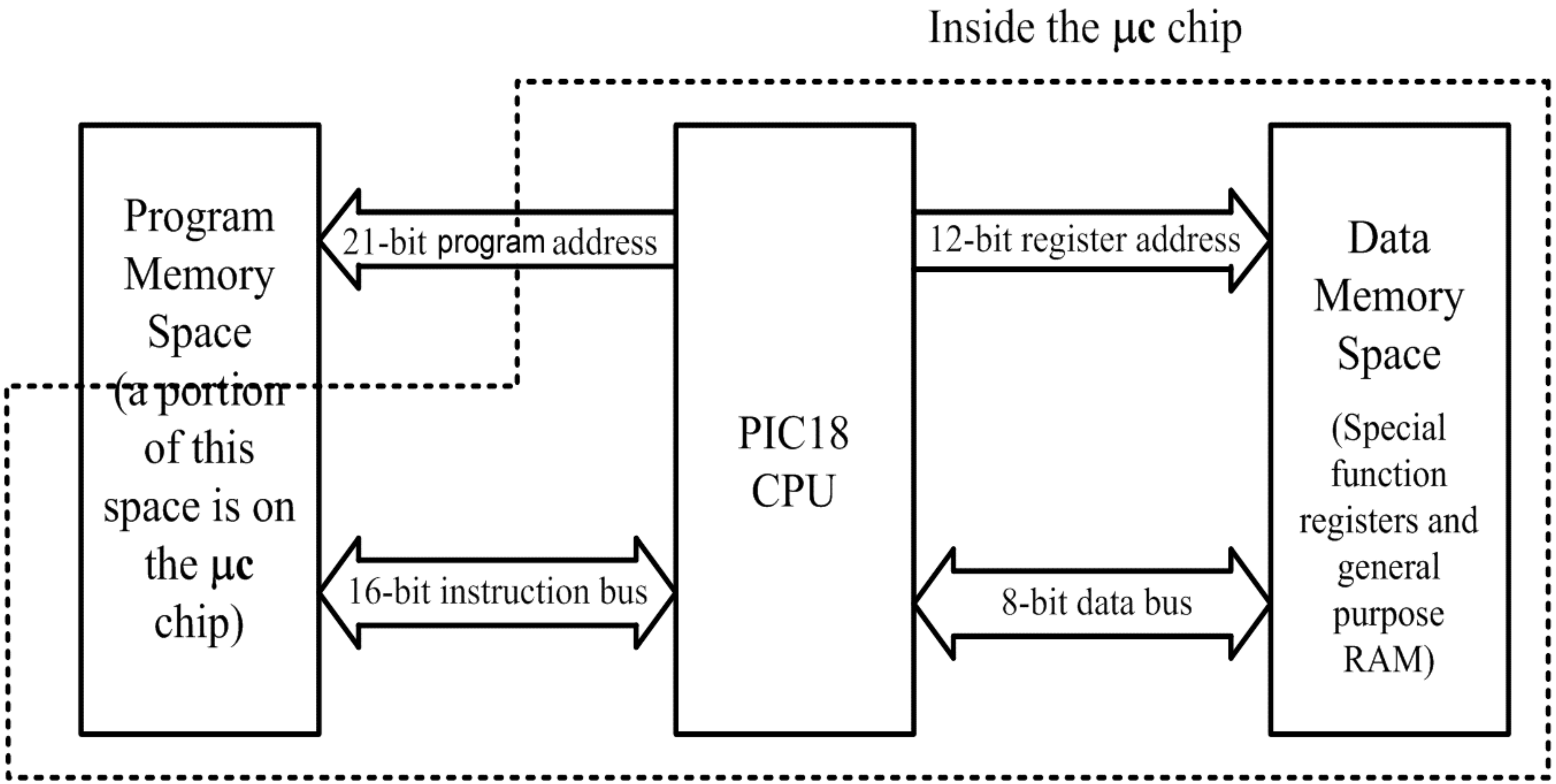


Figure 1.3 The PIC18 memory spaces

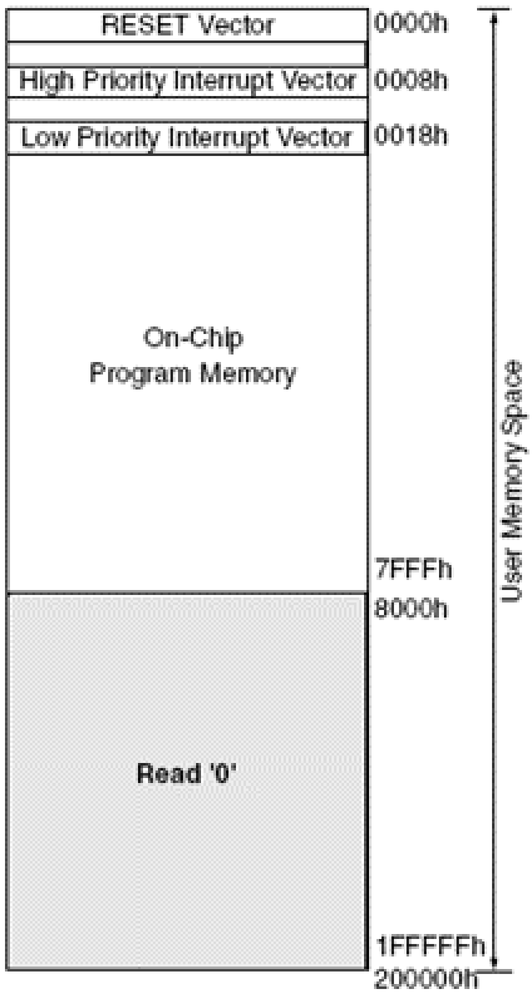
PIC18 Memory Organization

- 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
  - 12-bit address bus
  - Address up to 2^{12} = 4k bytes memory space
  - 8-bit data bus

CPU of a PIC18 microcontroller

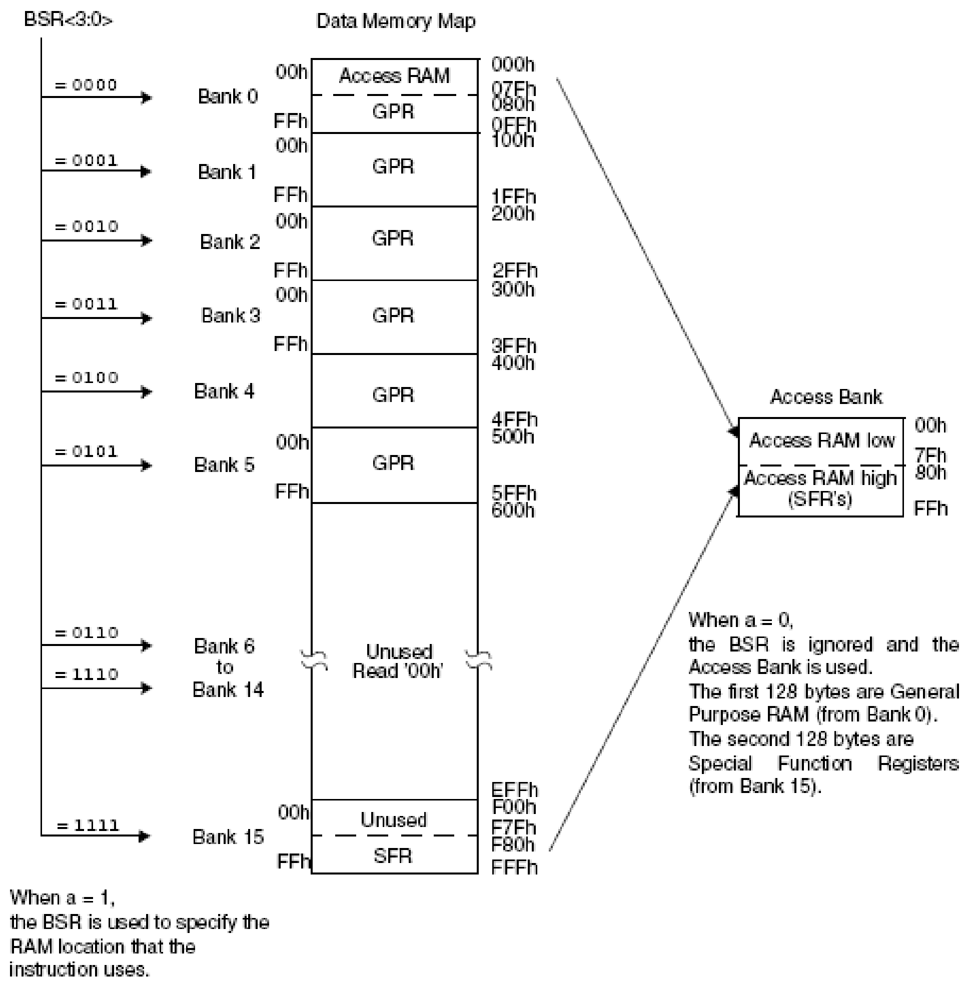
- Includes Arithmetic Logic Unit (ALU), Registers, and Control Unit
  - Arithmetic Logic Unit (ALU)
  - Registers
    - Status register: 5-bits (5 flags)
    - Working register (WREG): accumulator (8-bit)
    - Program Counter (PC): 21-bit
    - Bank Select Register (BSR)
      - 4-bit register used in direct addressing the data memory (16 banks)
    - File Select Registers (FSRs)
      - 12-bit registers used as memory pointers in indirect addressing data memory
  - Control unit
    - Provides timing and control signals to various Read and Write operations
    - Instruction decoder
      - 16-bit instructions

Program Memory



- Each PIC18 member has a 21-bit address bus for program memory.
- Can address up to 2MB program memory space.
- In PIC18F4520, only 32768 bytes were implemented, capable of storing 16384 instructions (most instructions are 2 bytes).
- The first byte in an instruction specifies the operation to be executed (such as addition) and the second byte specifies the *operands* (literal (value) or an address in data memory).

Data Memory in PIC18F4520



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.

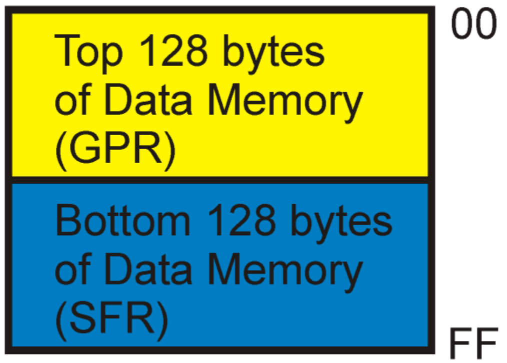
- Implemented as static RAM
- 12-bit (Max: 2^{12}=4096 bytes)
- PIC18F4520 has
  - 128 bytes of SFRs
  - 1536 bytes of GPRs

A problem with 12-bit addressing in data memory

- Recall: Only one byte is used to store the address of the data memory in most instructions. E.g., movwf 0x10
- A byte (8 bits) is not enough to uniquely specify a 12-bit address.
- Two solutions
  – Use of access bank
  – Define the most significant nibble of the address through BSR

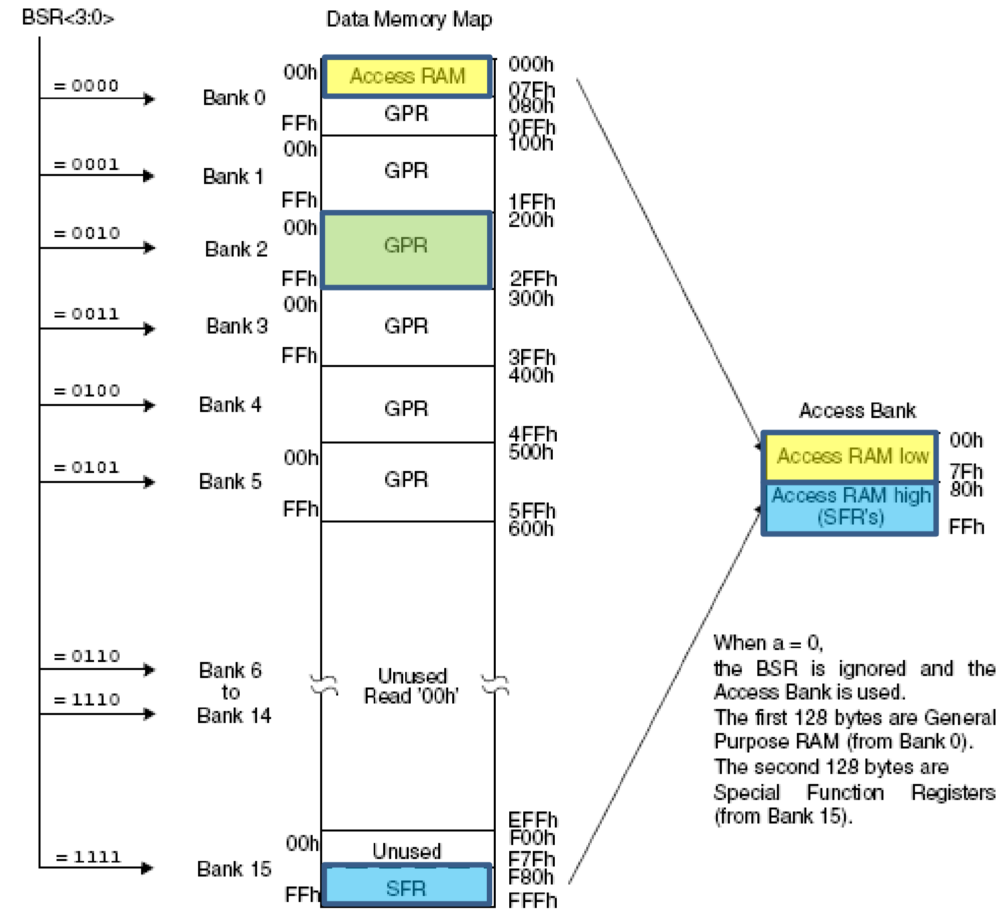
Data Memory in PIC18

Access Bank Mode:



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

Bank specified by
Bank Select Register (BSR)



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 bank is a 256-byte bank consisting of:
  - 128 bytes of GPRs located at 00 to 7F in the access bank, mapped from 000 to 07F of the data memory
  - 128 bytes of SFRs located at 80 to FF in the access bank, mapped from F80 to FFF of the data memory
- A program that requires more than the amount of memory provided in the access bank necessitates *bank switching*.

Exercises on access bank

- Identify the full data memory address (12-bit) corresponding to the following memory locations in the access bank:
  - 0x00
  - 0x5F
  - 0x7F
  - 0x80
  - 0xAA
  - 0xFF

General-purpose and Special-function registers

- Two types of registers: general-purpose (GPR) and special-function registers (SFR)
- GPRs provide storage for variables used in a program.
- SFRs are used to control the operation of the CPU and peripherals, e.g.,
  - The WREG register is involved in the execution of many instructions.
  - The STATUS register contains the arithmetic status of the ALU.

2.2 PIC18 Data Registers and Instructions

Outline

- Data representation in PIC
- WREG register and related Instructions
- PIC file registers
- Instructions involving file registers
- STATUS register
- Effects of instructions on STATUS register

Data Format Representation

- Data can only be represented as a 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: movlw 99. (Hex is the default representation)
  2. Use h (or H) right after the number: movlw 99H
  3. Put 0x (or 0X) before the number: movlw 0x99
  4. Put h in front of the number, with single quotes around the number: movlw h ‘99’
- If 1 or 2 is used and the starting hex digit is A-F, the number must be perceded by a 0.
  – e.g., movlw C6 is invalid. Must be movlw 0C6

Binary and Decimal Numbers

- The only way to represent a binary number is to put a B (or b) in front: movlw B ‘10011001’
- Two ways to present a decimal number:
    1. Put a D (or d) in front: movlw D ‘12’
    2. Use the “.value” format: movlw .12
- The only way to represent an ASCII character is to put a A (or a) in front: movlw A ‘2’.
- The ASCII code 0x32 is used to represent the character ‘2’. 0x32 is stored in WREG.

WREG Register in PIC18

- WREG – working register – is the most widely used register in PIC18
- 8-bit register
- Demonstrate two simple instructions using WREG
  - movlw
  - addlw

movlw

- movlw stands for "MOVe a Literal into WREG".
- Literal means that an actual value must be specified.
- Format: `movlw K` where `K` is a 8-bit value ranging from 0-255.
- e.g., movlw 0x25 moves 25 (in hex) to WREG

addlw

- addlw stands for “ADD Literal and WREG”.
- addlw adds a literal and WREG and stores the result in WREG
- Format: `addlw K` where `K` is a 8-bit value ranging from 0-255.
- e.g., addlw 0x25 adds 25 (in hex) and WREG
- e.g., Add two numbers 25H and 34H:

```
movlw 0x25
addlw 0x34
```

- Result: [WREG] = 0x59

Effect of addlw on the status register

- addlw affects all flag bits: C, Z, DC, OV and N
- N, OV relates only to signed number operations (discussed later).
- e.g. 1:

```
movlw 0x38
addlw 0x2F
```

$$\begin{array}{r} \mathbf{38} \\ \mathbf{+\ 2F} \\ \hline \mathbf{67} \end{array}$$

C = 0: No carry beyond MSB

DC = 1: A carry from the first and second nibble

Z = 0: The result of addition is not zero

Effect of addlw on the status register

- e.g. 2

```
movlw 0x9C
addlw 0x64
```

$$\begin{array}{r} \mathbf{9C} \\ \mathbf{+\ 64} \\ \hline \mathbf{00} \end{array}$$

C = 1: A carry beyond MSB

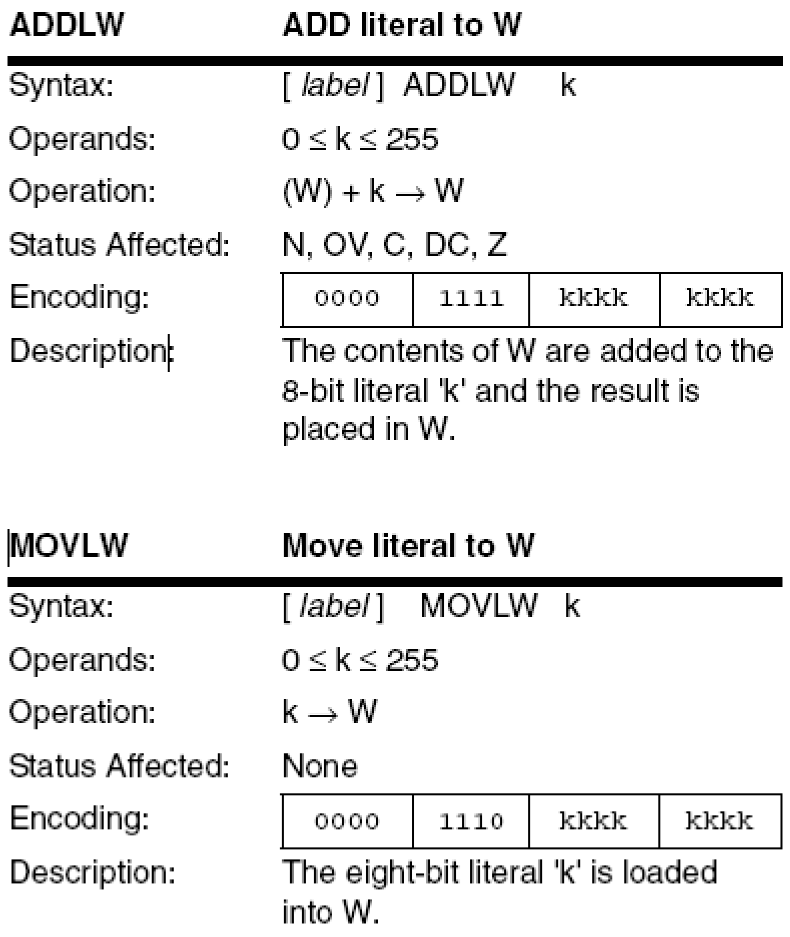
DC = 1: A carry from the first and second nibble

Z = 1: The result of addition is zero

Instruction documentation

The specification of all instruction are documented in Chapter 24 of the PIC18 microcontroller datasheet. Please download the datasheet from here:

http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf



ADDLW	**ADD literal to W**
Syntax:	[*label*] ADDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) + k → W
Status Affected:	N, OV, C, DC, Z
Encoding:	0000 1111 kkkk kkkk
Description:	The contents of W are added to the 8-bit literal 'k' and the result is placed in W.

MOVLW	**Move literal to W**
Syntax:	[*label*] MOVLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → W
Status Affected:	None
Encoding:	0000 1110 kkkk kkkk
Description:	The eight-bit literal 'k' is loaded into W.

Answer the following questions:

Show the status of the C, Z, DC flag bits for the following code:

```
(a) movlw 0x82
    addlw 0x22
(b) movlw 0x67
    addlw 0x99
(c) movlw 0x54
    addlw 0xC4
(d) movlw 0x00
    addlw 0xFF
(e) movlw 0xFF
    addlw 0x05
```

PIC File Registers

- PIC18 has many other 8-bit registers in the data memory that can be divided into two groups:
  - Special Function Register (SFR)
  - General-Purpose Register (GPR)
- SFRs are dedicated to specific functions, e.g., ALU status, program counters, I/O ports.
- The function of each SFR is fixed.
- GPRs are a group of RAM locations in the data memory.
- Data memory that is not allocated to SFRs are typically GPRs.

Some instructions involving file registers

- movwf – move WREG to file register
- addwf – add WREG and f
- Data sheet description

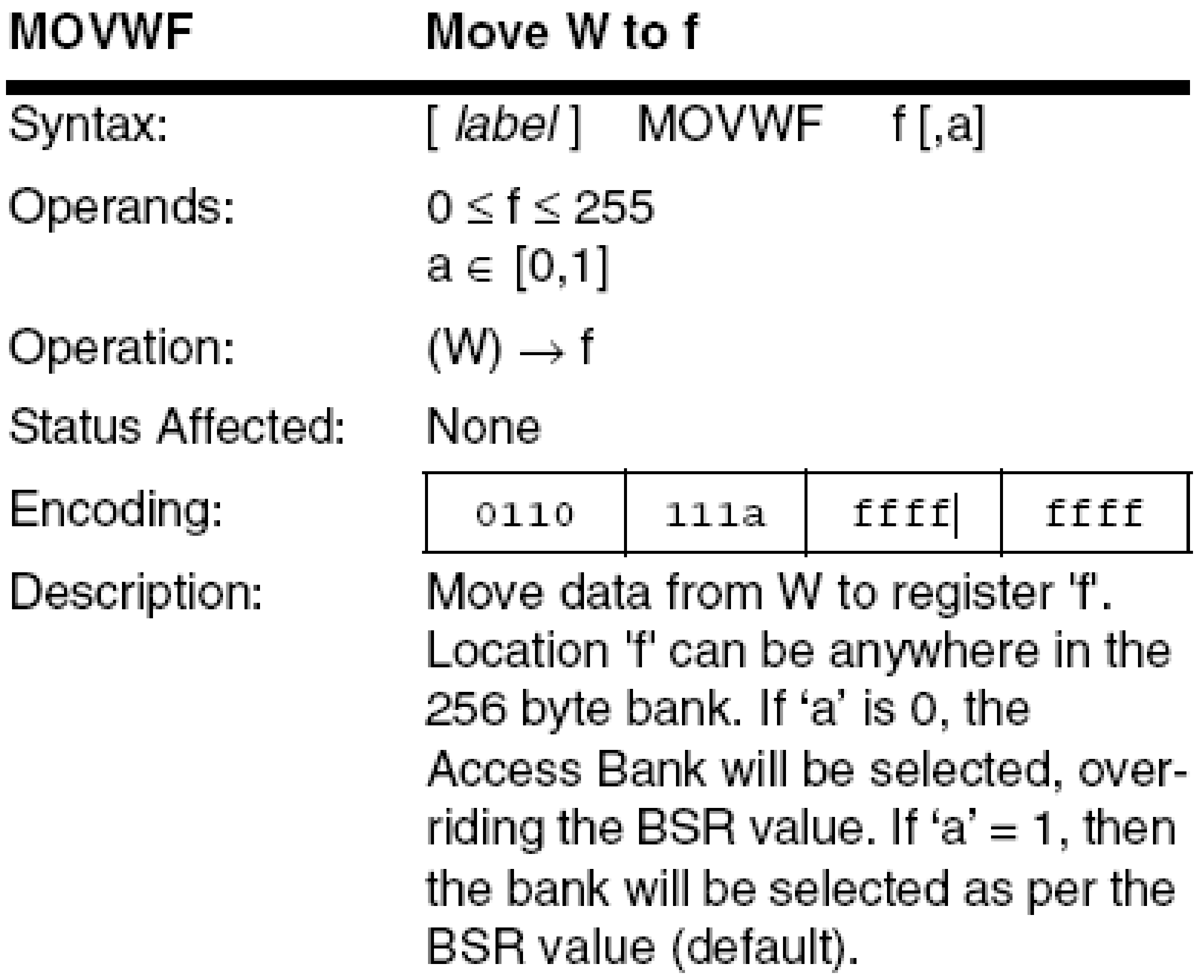
movwf

- movwf stands for "MOVe WREG into File Register"
- File Registers just refer to any register (i.e., SFR or GPR)
- Format: `movwf f[, a]` where
  - `f` is the 8-bit address of the destination file register.
  - `a` (optional) specifies whether you use the access bank (a = A = 0) or BSR (a = BANKED = 1).
- e.g., Move 0x55 into the file register with address 0x30:

```
movlw 0x55
movwf 0x30, A
```

Address	Data
WREG	55
0x030	55

movwf as described in datasheet

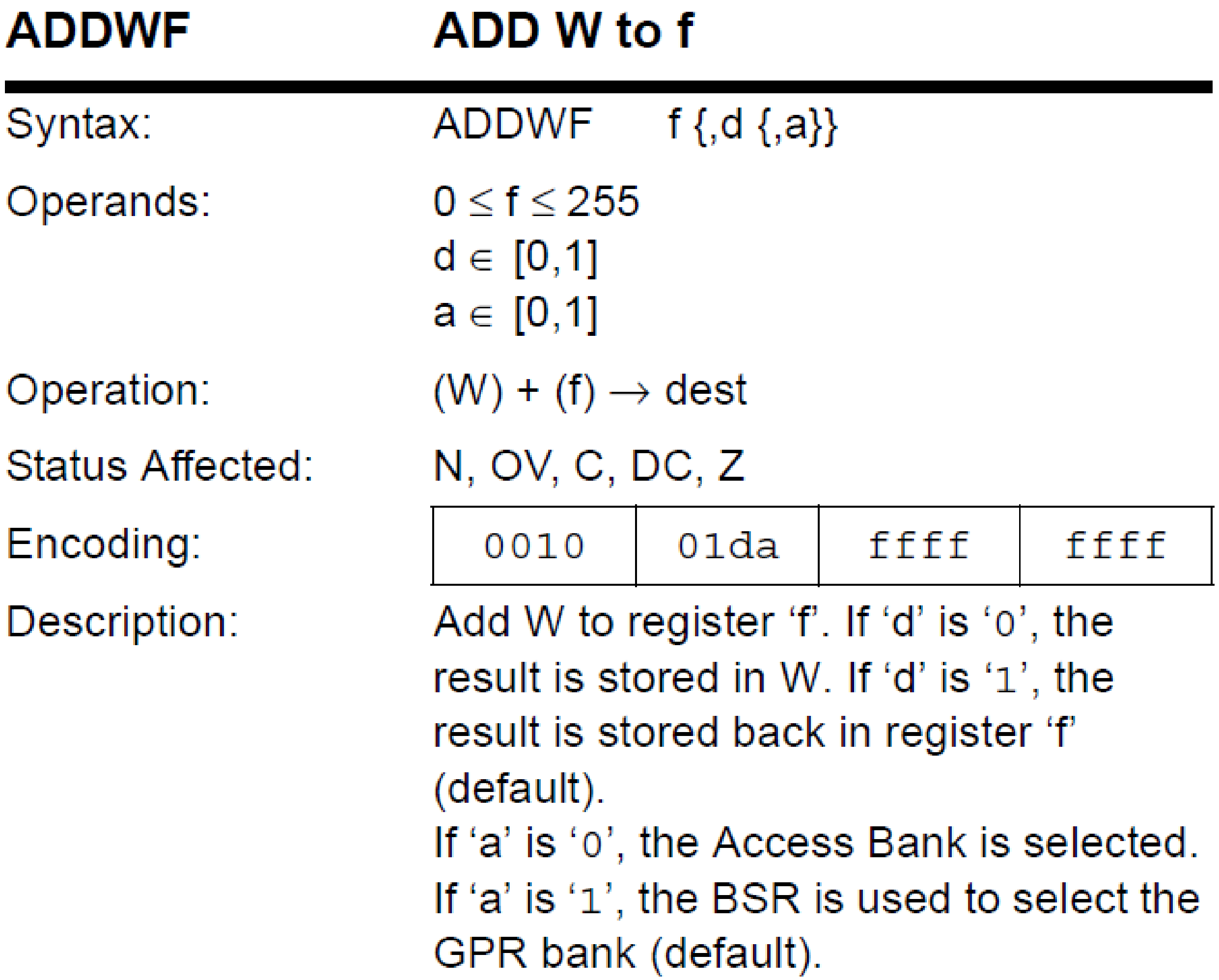


MOVWF	**Move W to f**
Syntax:	[*label*] MOVWF f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	$(W) \rightarrow f$
Status Affected:	None
Encoding:	`0110` | `111a` | `ffff` | `ffff`
Description:	Move data from W to register 'f'. Location 'f' can be anywhere in the 256 byte bank. If ‘a’ is 0, the Access Bank will be selected, over-riding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).

addwf

- addwf stands for "ADD WREG and File Register"
- The destination of the move can be:
  - WREG (Typical)
  - the file register
- Format: `addwf f[, d, a]` where
  - `f` is the 8-bit address of file register
  - `d` (optional) indicates the destination of the move:
    - if d = W = 0 – destination is WREG
    - if d = F = 1 – destination is file register (default)
  - `a` (optional) specifies whether you use the access bank (a = A = 0) or BSR (a = BANKED = 1).

addwf



ADDWF **ADD W to f**

Syntax:	ADDWF f {,d {,a}}
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(W) + (f) \rightarrow$ dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0010` | `01da` | `ffff` | `ffff`
Description:	Add W to register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default). If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).

addwf

- e.g. 1: Move 22H into two file registers with addresses 0x05 and 0x06 in access bank, add the contents of all three registers and put the result in WREG:

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

Address	Data
WREG	66
0x005	22
0x006	22

addwf

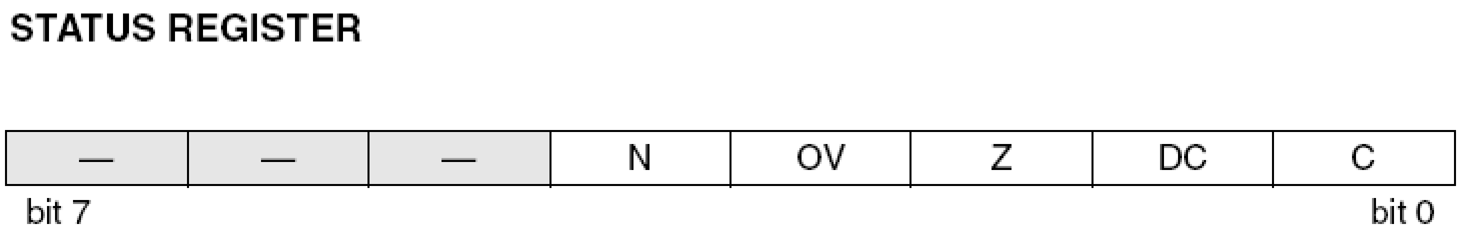
- e.g. 2: Same as e.g. 1, but put the result in file register 0x06 in the access bank

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

Address	Data
WREG	44
0x005	22
0x006	66

Status register

- 8-bit register, only 5 bits are used
- These 5 bits represent five conditional flags, indicating conditions after an instruction is executed.



STATUS REGISTER

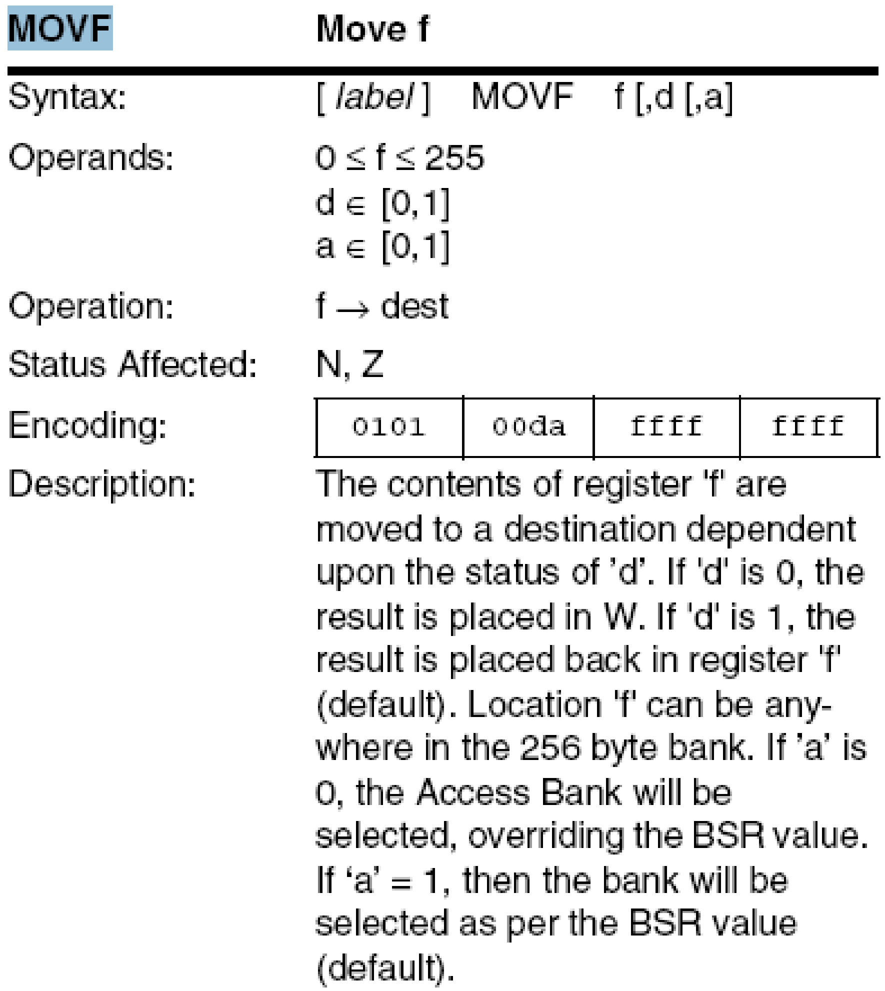
—	—	—	N	OV	Z	DC	C
bit 7							bit 0

- N (Negative Bit): Turns 1 if arithmetic result is negative
- OV (Overflow Bit): Turns 1 if overflow occurred for signed arithmetic
- Z (Zero Flag): Turns 1 if the result of an arithmetic/logic operation is zero
- DC (Digit carry bit): Turns 1 if a carry-out from the 4th low-order bit occurred.
- C (Carry bit): Turns 1 if a carry-out from the MSB occurred

movf

- movf stands for "MOVe File register"
- The destination of the move can be:
  - WREG (Typical)
  - the file register itself
- Format: `movf f[, d, a]` where
  - `f` is the 8-bit address of file register
  - `d` (optional) indicates the destination of the move:
    - if d = W – destination is WREG
    - if d = F – destination is file register (default)
  - `a` (optional) specifies whether you use the access bank (a = A = 0) or BSR (a = BANKED = 1).

movf



MOVF	Move f
Syntax:	[*label*] MOVF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$f \rightarrow dest$
Status Affected:	N, Z
Encoding:	`0101` | `00da` | `ffff` | `ffff`
Description:	The contents of register 'f' are moved to a destination dependent upon the status of 'd'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256 byte bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Compare movwf and movf

movwf f

- Move data from WREG to a file register
- The destination must be the file register

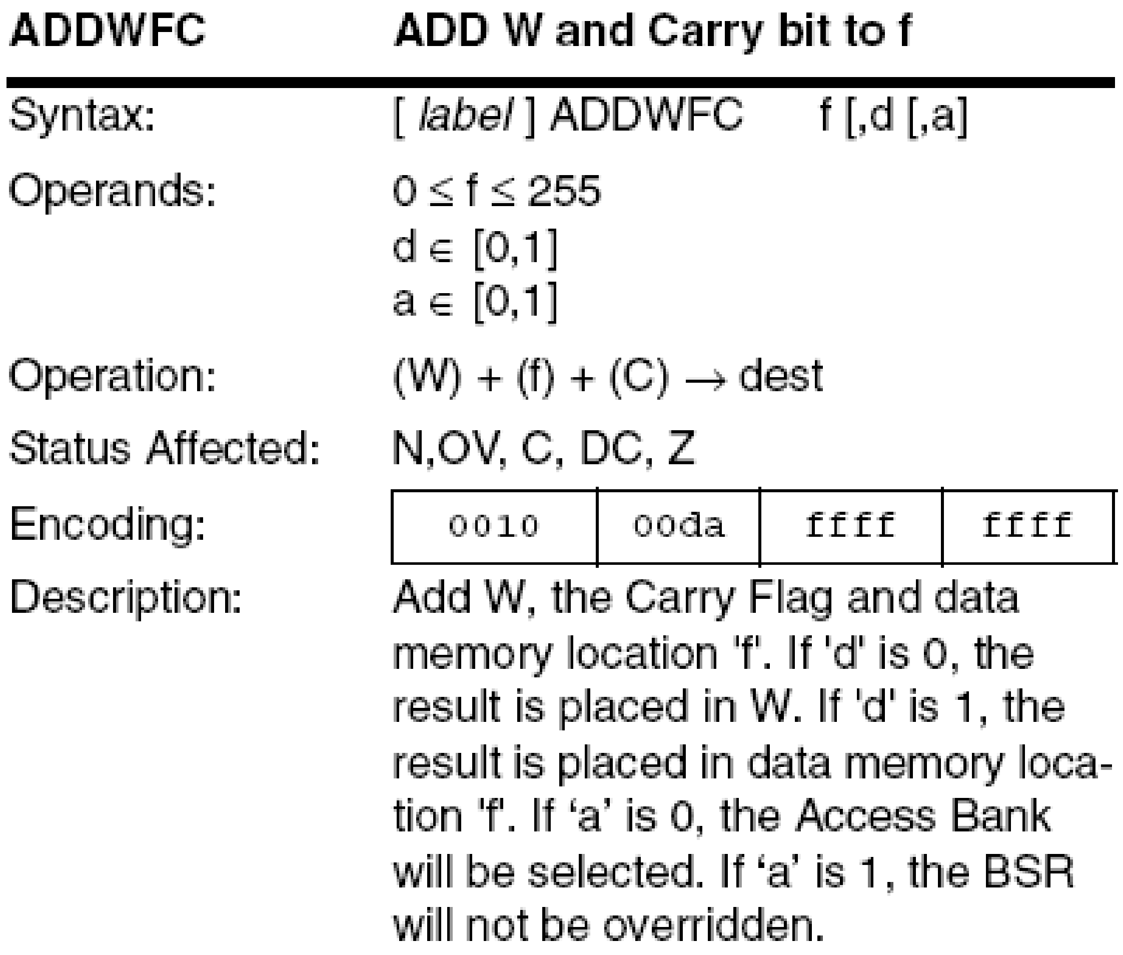
movf f,d

- Move data from a file register to either WREG or to itself
- Move data to itself is a dummy operation. The main purpose is to test the value in f
- e.g., if [f] = 0, Z bit of STATUS register = 1 after execution. [f] = contents in f register.

addwfc

- addwfc stands for “ADD WREG, File Register and Carry Bit”
- Format: `addwfc f[, d, a]`
- Used in addition of values more than 8-bit

addwfc



ADDWFC	ADD W and Carry bit to f
Syntax:	[*label*] ADDWFC f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(W) + (f) + (C) \rightarrow dest$
Status Affected:	N,OV, C, DC, Z
Encoding:	`0010` | `00da` | `ffff` | `ffff`
Description:	Add W, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed in data memory location 'f'. If ‘a’ is 0, the Access Bank will be selected. If ‘a’ is 1, the BSR will not be overridden.

Example: 16-bit addition

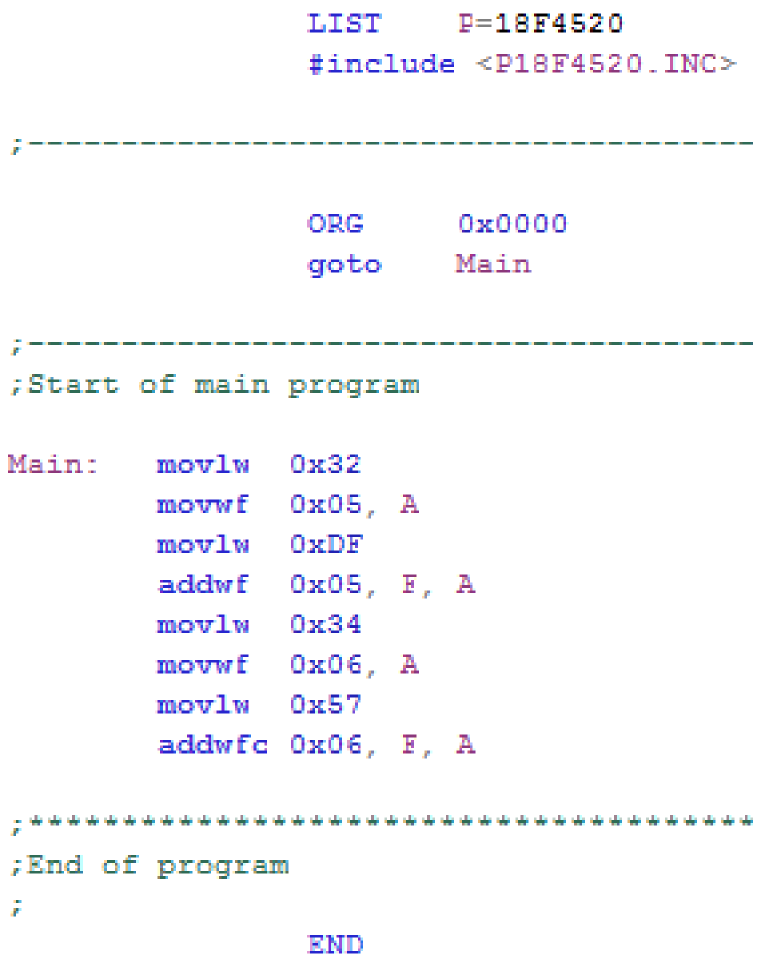
- Write a program to add two 16-bit numbers: 0x3432 and 0x57DF. Put the most significant byte of the result in 0x06 and the least significant byte in 0x05 of the access bank.

```
movlw   0x32
movwf   0x05, A
movlw   0xDF
addwf   0x05, F, A
movlw   0x34
movwf   0x06, A
movlw   0x57
addwfc  0x06, F, A
```

$$\begin{array}{rccc} & 34 & & 32 \\ + & 57 & \xleftarrow{C} & DF \\ \hline & 8C & & 11 \end{array}$$

Test your program using MPLAB IDE

1. Download MPLAB IDE here: http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_IDE_8_92.zip
2. Open MPLAB IDE and create a file. Type the code on the right and save the file as an .asm file



```
                LIST     P=18F4520
                #include <P18F4520.INC>

;-----------------------------------------

                ORG      0x0000
                goto     Main

;-----------------------------------------
;Start of main program

Main:   movlw   0x32
        movwf   0x05, A
        movlw   0xDF
        addwf   0x05, F, A
        movlw   0x34
        movwf   0x06, A
        movlw   0x57
        addwfc  0x06, F, A

;*****************************************
;End of program
;
                END
```

Demo in MPLAB IDE

FAQ regarding my demonstration

- Where the code written by a programmer is placed in program memory? Is there any way to check the program memory?
- I am still not sure I understand why the access bank is there. Can you review it one more time? Is access bank a block of physical memory that is separated from the data memory?

You should be able to ...

- Describe different types of data format representation in PIC18 programming.
- Manipulate data using WREG and MOVE instructions.
- Describe different types of registers in PIC18
- Describe access bank in PIC18
- Perform simple arithmetic operations such as ADD and MOVE using the file registers and access bank in PIC18

You should be able to ....

- Explain the purpose of the status registers
- Understand instruction descriptions in data sheet.
- Code simple PIC assembly language instructions
- Assemble and run a PIC program using MPLAB
- Examine PIC registers using MPLAB simulator