# Chapter 2
# The PIC18 Assembly Language Programming



1

## Sections

2.1 Inside the PIC18
2.2 Simple PIC18 Program
2.3 Introduction to PIC18 Assembly Programming
2.4 Assembling and Running an PIC18 Program
2.5 The Program Counter and ROM Space in the PIC18
2.6 RISC Architecture in the PIC
2.7 PIC18 Flag Bits and the PSW Register
2.8 PIC18 Register Banks and Stack



2

## Sections

**2.1 Inside the PIC18**
2.2 Simple PIC18 Program
2.3 Introduction to PIC18 Assembly Programming
2.4 Assembling and Running an PIC18 Program
2.5 The Program Counter and ROM Space in the PIC18
2.6 RISC Architecture in the PIC
2.7 PIC18 Flag Bits and the PSW Register
2.8 PIC18 Register Banks and Stack



3

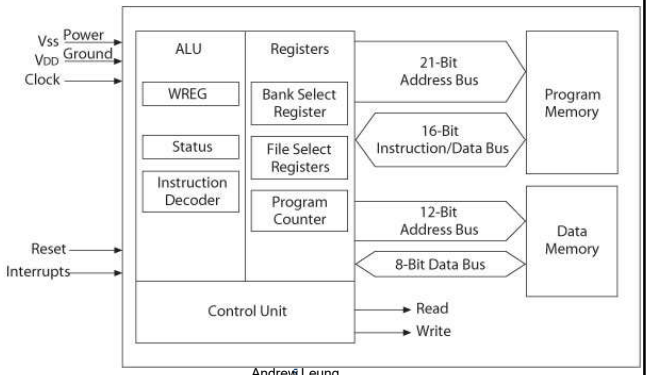
## Review: PIC18 Architecture

- Harvard Architecture which includes:
  - Microprocessor unit (MPU)
  - Program memory for instructions
  - Data memory for data
  - I/O ports
  - Support devices such as timers



4

## Review: MPU and Memory






5

## 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 (4k bytes memory space)
  - 8-bit data bus



6

## Microprocessor Unit

- Includes Arithmetic Logic Unit (ALU), Registers, and Control Unit
  - Arithmetic Logic Unit (ALU)
    - Instruction decoder
      - 16-bit instructions
    - Status register
      - 5-bits (5 flags)
    - WREG – working register
      - accumulator (8-bit)



7

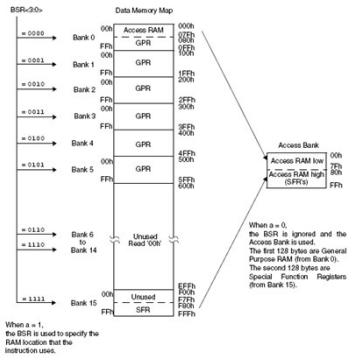
## Microprocessor Unit

- Registers
  - 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



8

## Data Memory in PIC18F452



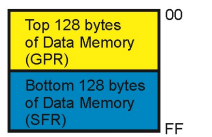

- Implemented as static RAM
- 12-bit (Max: $2^{12}$=4096 bytes)
- PIC18F4520 has
  - 128 bytes of SFRs
  - 1536 bytes of GPRs



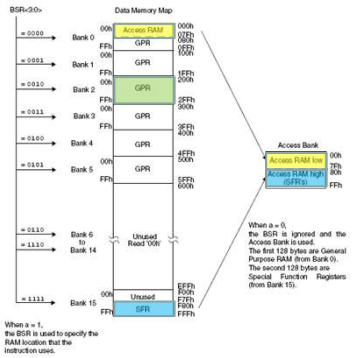
9

## Data Memory in PIC18

Access (Default) Bank Mode:




Bank specified by Bank Select Register (BSR)




10

## 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)



11

## 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.
  - The WREG register is involved in the execution of many instructions.
  - The STATUS register contains the arithmetic status of the ALU.



12

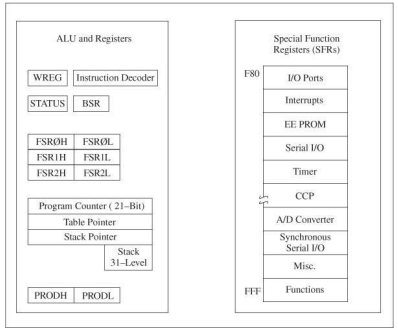
## PIC18F Programming Model

- The representation of the internal architecture of a microprocessor, necessary to write assembly language programs
- Divided into two groups
  - Arithmetic Logic Unit (ALU) and Registers
    - From Microprocessor Unit (MPU)
  - Special Function Registers (SFRs)
    - From Data (File) Memory



13

## PIC18F Programming Model






14

## Registers

- Program Counter (PC)
  - 21-bit register functions as a pointer to program memory during program execution
- STATUS: Flag Register
  - 5 individual bits called flags
- WREG (W): Working Register
  - 8-bit Accumulator
- Product
  - 16-bit Product of 8-bit by 8-bit Multiply



15

## Registers

- Table Pointer (access program ROM)
  - 21-bit register used as a memory pointer to copy bytes between program memory and data registers
- Stack Pointer (SP)
  - 5-bit register used to point to the stack
- Stack
  - 31 registers used for temporary storage of memory addresses during execution of a program



16

## Registers

(Access RAM area)

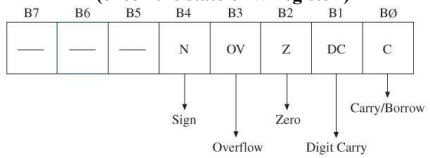
- BSR: Bank Select Register (0 to F)
  - 4-bit Register
    - Provides upper 4-bits of 12-bit address of data memory
- FSR: File Select Registers
  - FSR0, FSR1, and FSR2
  - FSR: composed of two 8-bit registers
    - FSRH and FSRL
  - Used as pointers for data registers
  - Holds 12-bit address of data register



17

## Flags in Status Register

**(check the state of W register )**




- N (Negative Flag)
  - Set when bit B7 is one as the result of an arithmetic/logic operation
- OV (Overflow Flag)
  - Set when result of an operation of signed numbers goes beyond 7-bits
- Z (Zero Flag)
  - Set when result of an operation is zero
- DC (Digit Carry Flag) (Half Carry)
  - Set when carry generated from Bit3 to Bit4 in an arithmetic operation
- C (Carry Flag)
  - Set when an addition generates a carry



18

## Special Function Registers

- SFRs:
  - Data registers associated with I/O ports, support devices, and processes of data transfer
    - I/O Ports (A to E)
    - Interrupts
    - EEPROM
    - Serial I/O
    - Timers
    - Capture/Compare/PWM (CCP)
    - Analog-to-Digital (A/D) Converter



19

## Sections





20

## 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



21

## 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



22

## 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.



23

## PIC18 Instruction Set

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



24

## The WREG register (accumulator)

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




25

## 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
- MOVLW A23H



26

## 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

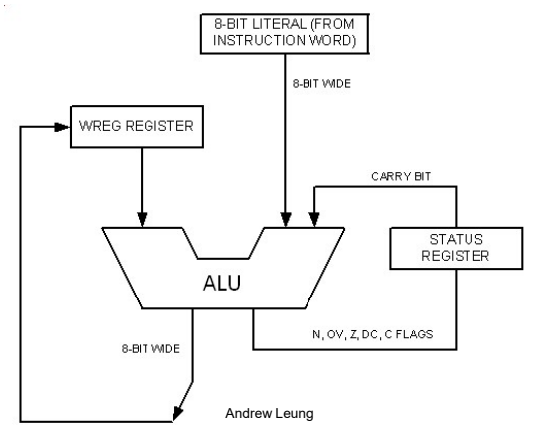
ADDKW 11H ; 0 0 1 1 1 0 0 1

ADDLW 43H ; 0 1 1 1 1 1 0 0



27

## ADDLW

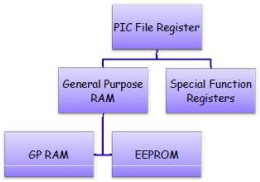


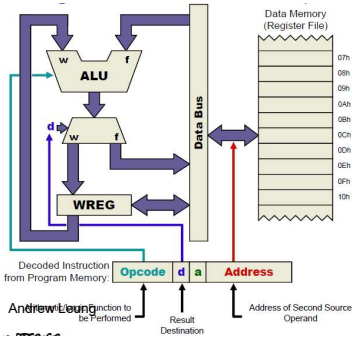



28

## The PIC File register (data memory)

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






29

## 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



30

## General Purpose Registers

- 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.

**GPRAM VS. EEPROM**

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



31

## 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 |



32

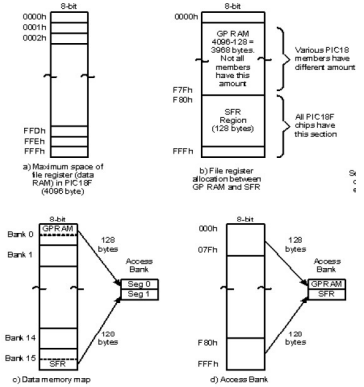
## File Register and access bank in the PIC18

- 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





## File Register and access bank in the PIC18








## 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





## SFRs of the 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
- COMF
- DECF
- MOVF
- MOVFF



37

## MOVWF instruction

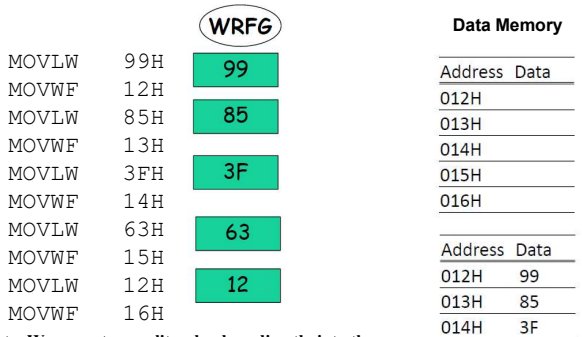
- F indicates for a file register

  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



38

## MOVWF instruction




**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 general purpose RAM location in the PIC18. They must be moved there via WREG.**



39

## ADDWF instruction

- 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



40

## ADDWF

```
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



41

## ADDWF

```
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



42

## 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.

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



43

## COMF

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

Solution:

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



44

## DECF (INCF) 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.

```
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)
```



45

## DECF instruction

`

```
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)
```



46

## MOVF instruction

MOVF File Reg. Address, D

It is intended to content of a File Reg. to WREG.

•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?)



47

## MOVF

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

Solution:

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

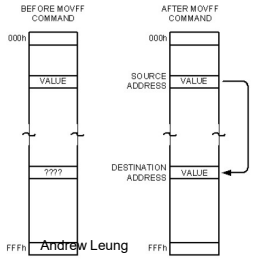


48

## MOVFF instruction

Copy the content of one location in FileReg to another location in FileReg.

MOVFF Source FileReg, destination FileReg






49

## MOVFF

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

```
Solution:

Again:    MOVFF PORTB, PORTC
          GOTO Again
```



50

## Sections

2.1 Inside the PIC18
2.2 Simple PIC18 Program
2.3 **Introduction to PIC18 Assembly Programming**
2.4 Assembling and Running an PIC18 Program
2.5 The Program Counter and ROM Space in the PIC18
2.6 RISC Architecture in the PIC
2.7 PIC18 Flag Bits and the PSW Register
2.8 PIC18 Register Banks and Stack



51

## Program Languages

- Machine language：
  - a program that consists of 0s and 1's.
  - 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++）



52

## Assembly Language

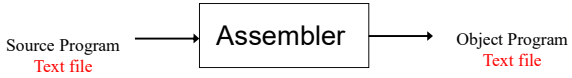
- Assembly languages were developed which provided mnemonics for the machine code instructions, plus other features.
  - Mnemonic：the instruction
    - Example：MOVFF, MOVLW
  - Provide decimal number, named registers, label, command
  - programming faster and less prone to error.
- Assembly language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





## Assembler

- Assembler：
  - a software program can translate an Assembly language program into machine code.
  - Source program
  - Object program, opcode, object code








## Structure of Assembly Language

- An Assembly language program (see Program 2-1) is a series of statements.

  [label:] mnemonic [operands] [;command]

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

    ```
    Here:  GOTO Here
    ```

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





## Mnemonic & Pseudo Instruction

- 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, too.

## 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 Directives

- EQU – associates a constant number with an address label.
- e.g., `COUNT equ 0x25`

```
        ..........
    movlw COUNT; WREG = 0x25
```

- SET – identical to EQU, but value assigned by SET can be reassigned later.
- e.g., `COUNT set 0x00`

```
        ...........
 COUNT set 0x25
        ..........
 movlw COUNT; WREG = 0x25
```





## EQU and SET Directives

**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
```





## CBLOCK Directive

- Defines a list of named constants.
- Format: `cblock` <num>
  <constant label> [:<inc>]
  `endc`
- `If`
- e.g. 1, `cblock 0x50`
  `test1, test2, test3, test4`
  `endc`
- Values Assigned:
  test1 = 0x50, test2 = 0x51,
  test3 = 0x52, test4 = 0x53.

## CBLOCK Directive

- 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



61

## Sample of an Assembly Language Program

```
  LIST P=18F4520        ;directive to define processor
#include <P18F4520.INC> ;CPU specific variable
                        ;definitions
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
```



62

## Sections





63

## Steps to Create an Executable Assembly Language Program

1. Execute the MAPLAB IDE program.
2. Click “File”; “New” and type the code into the file editor window.
3. Click “File”, “Save As”. Create and Select the “C:\Code\Expt1” folder and type “prog0.asm’ as the program file name. (Make sure you save the file into the Expt1 folder).
4. Click “Project”, “Project Wizard…”, “Next >”, select device “PIC18F4520”, click “Next >”, select “Microchip MPASM Toolsuite”, click “Next >”
5. Browse into “C:\Code\Expt1” folder, type “Expt1” as the Project file name and click “Save”.
6. Click “Next >”, expand the folder tree and locate the file prog0.asm. Click “Add >>” and “Next >” to put the prog0.asm file to the Project. Check the project parameters list and click “Finish” to finish the project definition process.



64

## Steps to Create an Executable Assembly Language Program

7. Click “Project”, “Build All” and select “Absolute”.
8. “*BUILD SUCCESSED*” should appear at Output window. Should “BUILD FAILED” appear instead, check for the error messages, fix any errors found and repeat the build process 7 until success.
9. Click “File”, “Save Workspace” to save your work. It will save all your current project related parameters into the file with extension “mcw”. You can double click file Expt1.mcw later to continue your development.
10. Now we can use choose Debugger (such as MPLAB SIM or PICKIT 3) to test our program.



65

## Sample of an Assembly Language Program

```
  LIST P=18F4520         ;directive to define processor
#include <P18F4520.INC> ;CPU specific variable
                         ;definitions
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
```



66

**Notes:** listing file using MPLAB to check

- Memory Content
- The Change in Program Count



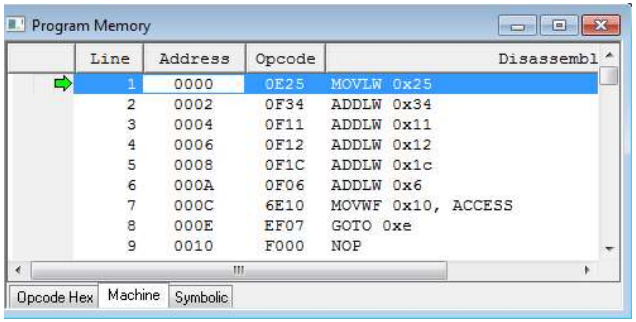
67

```
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
```



68

## Program Memory








## Sections







## 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)

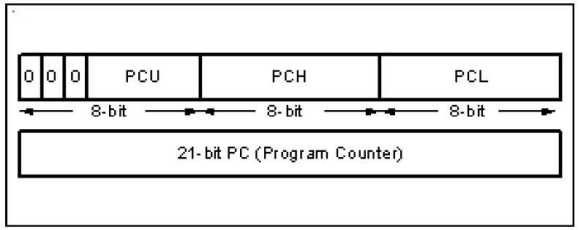
  PIC18 has 21 bits (2M)

  8051 has 16 bits (64K)





## PIC18 PC



## PIC family

**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 PWM | 1 | 1/3 | DIP, SOIC, SSOP, QFN | 18 |
| PIC18F1320 | FLASH | 8K | 256 | 256 | 16 | 7 | — | 1 | 6x PWM | 1 | 1/3 | DIP, SOIC, SSOP, QFN | 18 |
| PIC18F2220 | FLASH | 4K | 512 | 256 | 23 | 10 | I²C/SPI | 1 | 6x PWM | 2 | 1/3 | DIP, SOIC | 28 |
| PIC18F2320 | FLASH | 8K | 512 | 256 | 23 | 10 | I²C/SPI | 1 | 6x PWM | 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 PWM | 2 | 1/3 | DIP, TQFP, QFN | 40/44 |
| PIC18F4320 | FLASH | 8K | 512 | 256 | 34 | 13 | I²C/SPI | 1 | 6x PWM | 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 |

Abbreviations: ADC = Analog-to-Digital Converter; CCP = Capture/Compare/PWM; PWM = Pulse Width Modulation; SPI = Serial Peripheral Interface; USART = Universal Synchronous/Asynchronous Receiver/Transmitter; PWM = Power Managed Mode





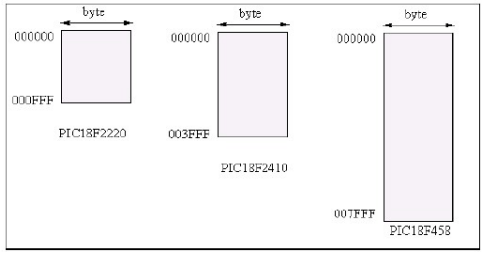
## ROM Size

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

a) PIC18F2220

b) PIC18F2410

c) PIC18F458



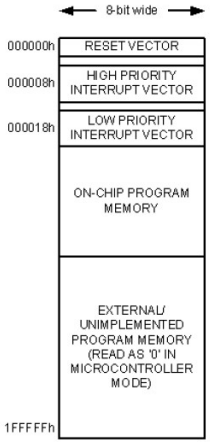





## Place Code in

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

- The uC wakes up at memory address 0
- The PC has the value 0000
- ORG directive put the address of the fi op code at the memory location 0000

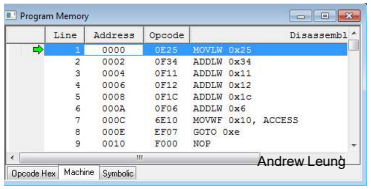







## Place Code in ROM

```
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
```



## Place Code in ROM

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

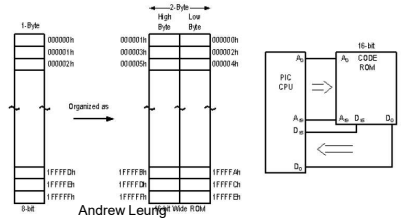
```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00000010              00004 SUM:    EQU 10H ;RAM loc 10H fro 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
```



77

## Program ROM width

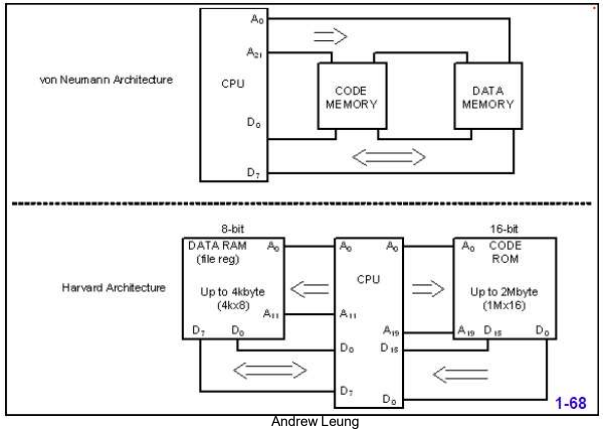
- Byte addressable: each location holds only one byte
  CPU with 8-Bit will fetch one byte a time
  Increasing the data bus will bring more information
- Solution: Data bus between CPU and ROM can be similar to traffic lanes on the highway
- The wide of Data path is 16 bit
  Increase the processing power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






78

## Review: von Neumann vs. Harvard Architecture






79

## 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



80

### Instruction size of the PIC18

- 4-Byte instructions include
  MOVFF (move data within RAM, which is 4k)
  1100 ssss ssss ssss (0≤ fs ≤ FFF)
  1111 dddd dddd dddd (0≤ fd ≤ 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$_8$



81

## Sections





82

## 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



83

## RISC Architecture in the PIC

| RISC | CISC |
|---|---|
| Simple Instruction with regular structure | Complex instructions with irregular strcuture |
| Execute one instruction in one cycle | Different instructions with different execution time |
| pipeline | May also pipeline |
| Many CPU registers | Smaller number of CPU registers |
| Separated data and program memory | One memory space |
| Most operations are registers to registers | Most operations can be register to memory |



84

## Sections

2.1 Inside the PIC18
2.2 Simple PIC18 Program
2.3 Introduction to PIC18 Assembly Programming
2.4 Assembling and Running an PIC18 Program
2.5 The Program Counter and ROM Space in the PIC18
2.6 RISC Architecture in the PIC
2.7 **PIC18 Flag Bits and the PSW Register**
2.8 PIC18 Register Banks and Stack





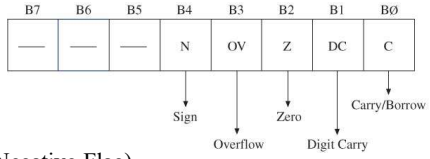
## PIC Flags

- When the CPU performs operations, sometimes an exception may occur.
  - Example：overflow
- How does the CPU tells control units that an exception occurs?
- Answer is the flags.
  - C Carry Flag.
  - DC Digital Carrry Flag
  - Z Zero Flag
  - OV Overflow Flag
  - N Negative Flag





## Flags in Status Register




- N (Negative Flag)
  - Set when bit B7 is one as the result of an arithmetic/logic operation
- OV (Overflow Flag)
  - Set when result of an operation of signed numbers goes beyond 7-bits
- Z (Zero Flag)
  - Set when result of an operation is zero
- DC (Digit Carry Flag) (Half Carry)
  - Set when carry generated from Bit3 to Bit4 in an arithmetic operation
- C (Carry Flag)
  - Set when an addition generates a carry





## Effect of addlw on the status register

- e.g. 1:

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

**38h**
**+ 2Fh**
**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 and second nibble
C=0 ; No carry

## Effect of addlw on the status register

- e.g. 2:

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

**9Ch**
**+ 64h**
**00h**

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 and second nibble

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



89

## Effect of addlw on the status register

- e.g. 3:

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

**80h (in signed sense -128)**
**+ 81h (in signal sense -127)**
**01h**

N = 0 ; bit7=0

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

Z=0 ; All zeros

DC=0 ; A carry from the first and second nibble

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



90

## Effect of addlw on the status register

- e.g. 4:

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

**7Fh**
**+ 7Fh**
**Feh**

N = 1 ; bit7=1

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

Z=0 ; All zeros

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

C=0 ; A carry is not generated (in unsigned sense)



91

## Instruction That Affect Flag Bits

**Table 2-4: 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 |



92

## 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 We will discuss them later.

**Table 2-5: PIC18 Branch (Jump) Instructions Using Flag Bits**

| 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 |
| BNC | Branch if N ≠ 0 |
| BOV | Branch if OV = 1 |
| BNOV | Branch if OV ≠ 0 |





## Sections







## 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

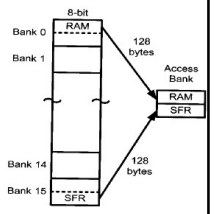
A=0, the access bank

A=1, other bank

To use this feature,

Load BSR with the desired bank number

Make A=1 in the instruction








## Register Banks

**Example:**

```
MYREG EQU 0x40 ;define a location
MOVLB 0x2
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
```

## Register Banks

**Example:**

```
MYREG EQU 0x40 ;define a location
MOVLB 0x2
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
```



97

## Stack

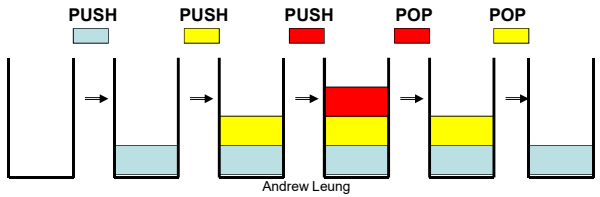
- Temporary memory storage space used during the execution of a program
- Can be part of R/W memory or specially designed group of registers
- Stack Pointer (SP)
  - The MPU uses a register called the stack pointer, similar to the program counter (PC), to keep track of available stack locations.



98

## Stack

- Stack：a section of RAM to store data items
- Two operations on the stack：
  - PUSH：put an item onto the *top* of the stack
  - POP：remove an item from the *top* of the stack






99

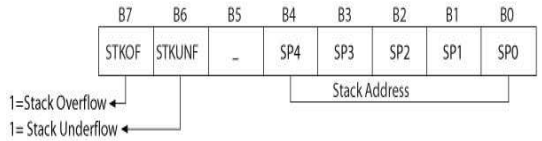
## PIC18 Microcontroller Stack

- Hardware Stack
  - 31 registers
  - 21-bits wide
  - Not part of program memory or data registers
- Stack Pointer (STKPTR)
  - 5-bit address
- Top of the Stack (TOS)
  - Pointed to by the stack pointer
  - Copied into three special function registers
  - TOSU (Upper), TOSH (High), and TOSL (Low)



100

## STKPTR Register




- SP4-SP0: Stack Address
- STKOF: Stack overflow
  - When the user attempts to use more than 31 registers to store information (data bytes) on the stack
- STKUNF: Stack underflow
  - When the user attempts to retrieve more information than what is stored previously on the stack



101

## Stack Instructions

- PUSH
  - Increment the memory address in the stack pointer and store the contents of the program counter (PC+2) on the top of the stack
- POP
  - Discard the address of the top of the stack and decrement the stack pointer by one



102

## Subroutine

- A group of instructions that performs a specified task
- Written independent of a main program
- Can be called multiple times to perform task by main program or by another subroutine
- Call and Return instructions used to call a subroutine and return from the subroutine



103

## Subroutine

```
 LIST      P=18F4520                ;directive to define processor
#include <P18F4520.INC>    ;CPU specific variable definitions
                ORG 0
                GOTO Main
Main:           ORG 20H                 ; start at address 0
                MOVLW 25H               ; WREG = 25
                CALL sub1
                MOVLW 20H
                CALL sub1
HERE:           GOTO HERE               ;stay here forever
sub1:           nop                     ; a subroutine
                nop
                MOVLW 0H
                Return
                END                     ; end of asm source file
```



104

## Call and Return Instructions

- CALL Label, s ;Call subroutine at Label
- CALL Label, FAST ;FAST equivalent to s = 1
  - If s = 0: Increment the stack pointer and store the return address (PC+4) on the top of the stack (TOS) and branch to the subroutine address located at Label
  - If s = 1: Also copy the contents of W, STATUS, and BSR registers in their respective shadow registers
- RCALL, n ;Relative call to subroutine
  - Increment the stack pointer and store the return address (PC+2) on the top of the stack (TOS) and branch to the location Label within = -2048 to + 2046



105

## Call and Return Instructions

- RETURN, s ;Return from subroutine
- RETURN FAST :FAST equivalent to s = 1
  - If s = 0: Get the return address from the stack (TOS) and place it in PC and decrement the stack pointer
  - If s = 1: Also retrieve the contents of W, STATUS, and BSR registers from their shadow registers
- RETLW 8-bit ;Return literal to WREG
  - Get the return address from the stack (TOS) and place it in PC and decrement the stack pointer
  - Return 8-bit literal to WREG



106

## Program Listing with Memory Addresses

```
Main Program                                  Subroutine

0020 0EFE      START: MOVLW   B'11111110'  DELAY50MC:
0022 6E94             MOVWF   TRISC        0040   0EA6   MOVLW   D'166'
0024 6E01             MOVWF   REG1         0042   6E10   MOVWF   REG10
0026 C001 FF82 ONOFF: MOVFF   REG1,PORTC   0044   0610   DECF    REG10,1
002A EC20 F000        CALL    DELAY50MC    0046   E1FE   BNZ     LOOP1
002E 1E01             COMF    REG1,1       0048   0012   RETURN
0030 D7FA             BRA     ONOFF
```

**In the above example, the return address is 002E.**



107

## Demo

| ROM address | Code | Line No. | |
|---|---|---|---|
| 000000 | | 00003 | ORG 0 |
| 000000 | EF10 F000 | 00004 | GOTO Main |
| 000020 | | 00005 Main: | ORG 20H; start at address 0 |
| 000020 | 0E25 | 00006 | MOVLW 25H ; WREG = 25 |
| 000022 | EC18 F000 | 00007 | CALL sub1 |
| 000026 | 0E20 | 00008 | MOVLW 20H |
| 000028 | EC18 F000 | 00009 | CALL sub1 |
| 00002C | EF16 F000 | 00010 HERE: | GOTO HERE ;stay here forever |
| 000030 | 0000 | 00011 sub1: | nop ; a subroutine |
| 000032 | 0000 | 00012 | nop |
| 000034 | 0E00 | 00013 | MOVLW 0H |
| 000036 | 0012 | 00014 | Return |
| | | 00015 | END ; end of asm source file |



108