INTRODUCTION TO THE PIC18 MICROCONTROLLER

PIC Microcontroller: An Introduction to Software & Hardware Interfacing

Han-Way Huang, Thomson Delmar Learning, 2005

Chung-Ping Young

Networked Embedded Applications and Technologies Lab

Department of Computer Science and Information Engineering
National Cheng Kung University, TAIWAN

What is a computer?

Software

Hardware

Computer Hardware Organization

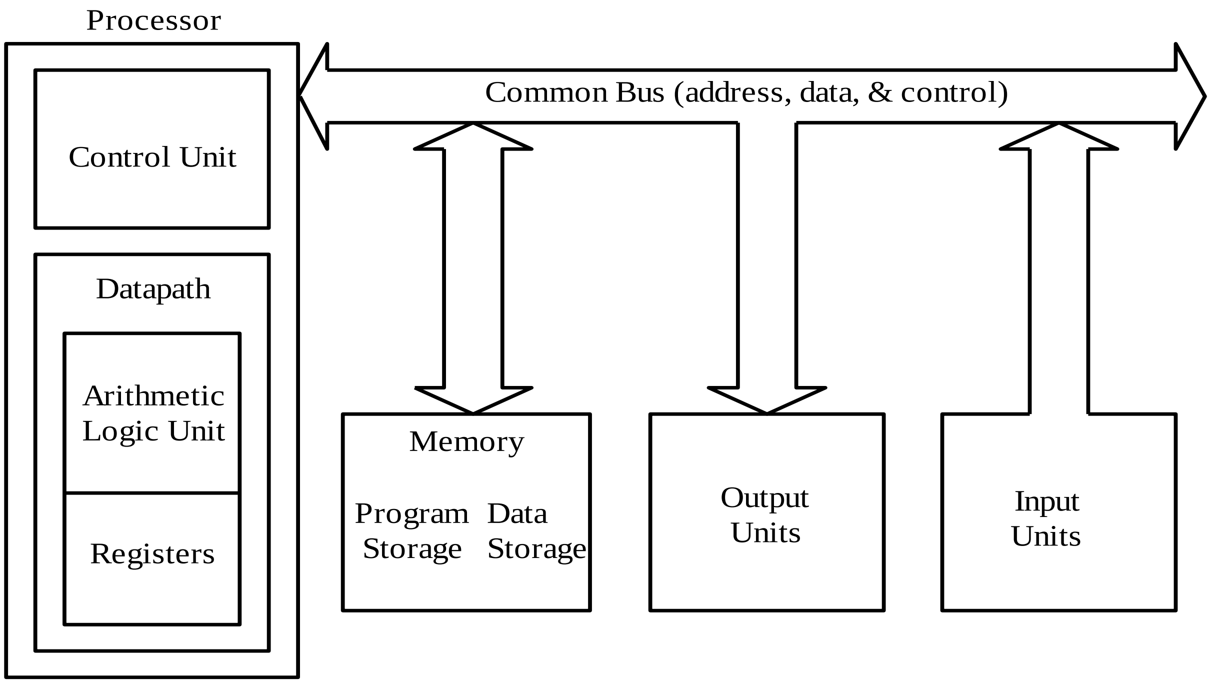
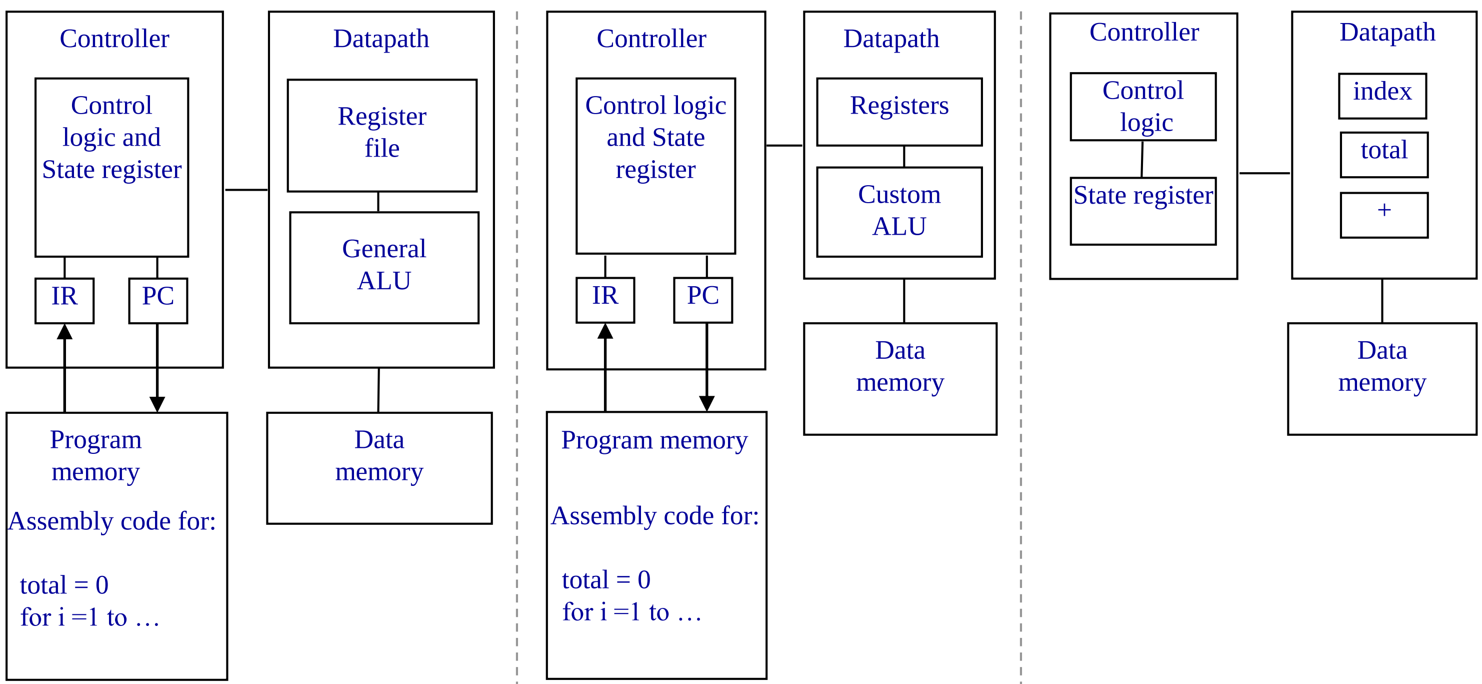


Figure 1.1 Computer Organization

Processor Technology

- The architecture of the computation engine used to implement a system's desired functionality
- Processor does not have to be programmable

"Processor" *not* equal to general-purpose processor



General-purpose ("software") **Application-specific** **Single-purpose ("hardware")**

The processor

Registers -- storage locations in the processor

Arithmetic logic unit

Control unit

program counter contains the address of the next instruction to be executed

status register flags the instruction execution result

The microprocessor

A processor implemented on a very large scale integration (VLSI) chip

Peripheral chips are needed to construct a product

The Microcontroller

The processor and peripheral functions implemented on one VLSI chip

Features of the PIC18 microcontroller

- 8-bit CPU
- 2 MB program memory space
- 256 bytes to 1KB of data EEPROM
- Up to 3968 bytes of on-chip SRAM
- 4 KB to 128KB flash program memory
- Sophisticated timer functions that include: input capture, output compare, PWM, real-time interrupt, and watchdog timer
- Serial communication interfaces: SCI, SPI, I2C, and CAN
- Background debug mode (BDM)
- 10-bit A/D converter
- Memory protection capability
- Instruction pipelining
- Operates at up to 40 MHz crystal oscillator

Embedded Systems

- A product that uses one or more microcontrollers as controller (s).
- End users are only interested in the functionality of the product but not on the microcontroller itself.
- Cell phones, home security system, automobiles, and many other products are examples of embedded products.

Semiconductor memory

- **Random-access memory** (RAM): same amount of time is required to access any location on the same chip
- **Read-only memory** (ROM): can only be read but not written to by the processor

Random-access memory

- **Dynamic random-access memory** (DRAM): need periodic refresh
- **Static random-access memory** (SRAM): no periodic refresh is required

Read-only memory

- **Mask-programmed read-only memory** (MROM): programmed when being manufactured
- **Programmable read-only memory** (PROM): can be programmed by the end user

Erasable programmable ROM (EPROM)

1. electrically programmable many times
2. erased by ultraviolet light (through a window)
3. erasable in bulk (whole chip in one erasure operation)

Electrically erasable programmable ROM (EEPROM)

1. electrically programmable many times
2. electrically erasable many times
3. can be erased one location, one row, or whole chip in one operation

Flash memory

1. electrically programmable many times
2. electrically erasable many times
3. can only be erased in bulk (either a block or the whole chip)

Computer software

- Computer programs are known as software
- A program is a sequence of instructions

Machine instruction

- A sequence of binary digits which can be executed by the processor
- Hard to understand, program, and debug for human being

Assembly language

- Defined by assembly instructions
- An assembly instruction is a mnemonic representation of a machine instruction
- Assembly programs must be translated before it can be executed -- translated by an assembler
- Programmers need to work on the program logic at a very low level and can't achieve high productivity.

High-level language

- Syntax of a high-level language is similar to English
- A translator is required to translate the program written in a high-level language -- done by a compiler
- Allows the user to work on the program logic at higher level.

Source code

- A program written in assembly or high-level language

Object code

- The output of an assembler or compiler

Source code and object code examples

address	object code	line no.	Source code	
00001E	0E06	00010	movlw	0x06
000020	6E11	00011	movwf	0x11,A
000022	0E07	00012	movlw	0x07
000024	6E12	00013	movwf	0x12,A
000026	0E08	00014	movlw	0x08
000028	6E13	00015	movwf	0x13,A
00002A	0E05	00016	movlw	0x05
00002C	5E10	00017	subwf	0x10,F,A
00002E	5E11	00018	subwf	x11,F,A

Radix Specification

- Hexadecimal (or hex) number is specified by adding the prefix **0x** or by enclosing the number with single quotes and preceding it by an H.
- **0x02**, **0x1234**, **H`2040’** are hex numbers
- Decimal numbers are enclosed by single quotes and preceded by letter D.
- **D`10’** and **D`123’** are decimal numbers
- Octal and binary numbers are similarly specified.
- **O`234’** is an octal number; **B’01011100’** is a binary number.

Memory Addressing

- Memory consists of a sequence of directly addressable **locations**.
- A location is referred to as an **information unit**.
- A memory location can be used to store **data**, **instruction**, and the **status** of peripheral devices.
- A memory location has two components: an **address** and its **contents**.

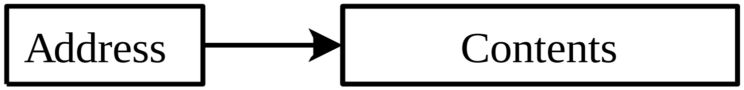


Figure 1.2 The components of a memory location

The PIC18 Memory Organization

- Data Memory and Program Memory are separated
- Separation of data memory and program memory makes possible the simultaneous access of data and instruction.
- Data memory are used as general-purpose registers or special function registers
- On-chip Data EEPROM are provided in some PIC18 MCUs

Separation of Data Memory and Program Memory

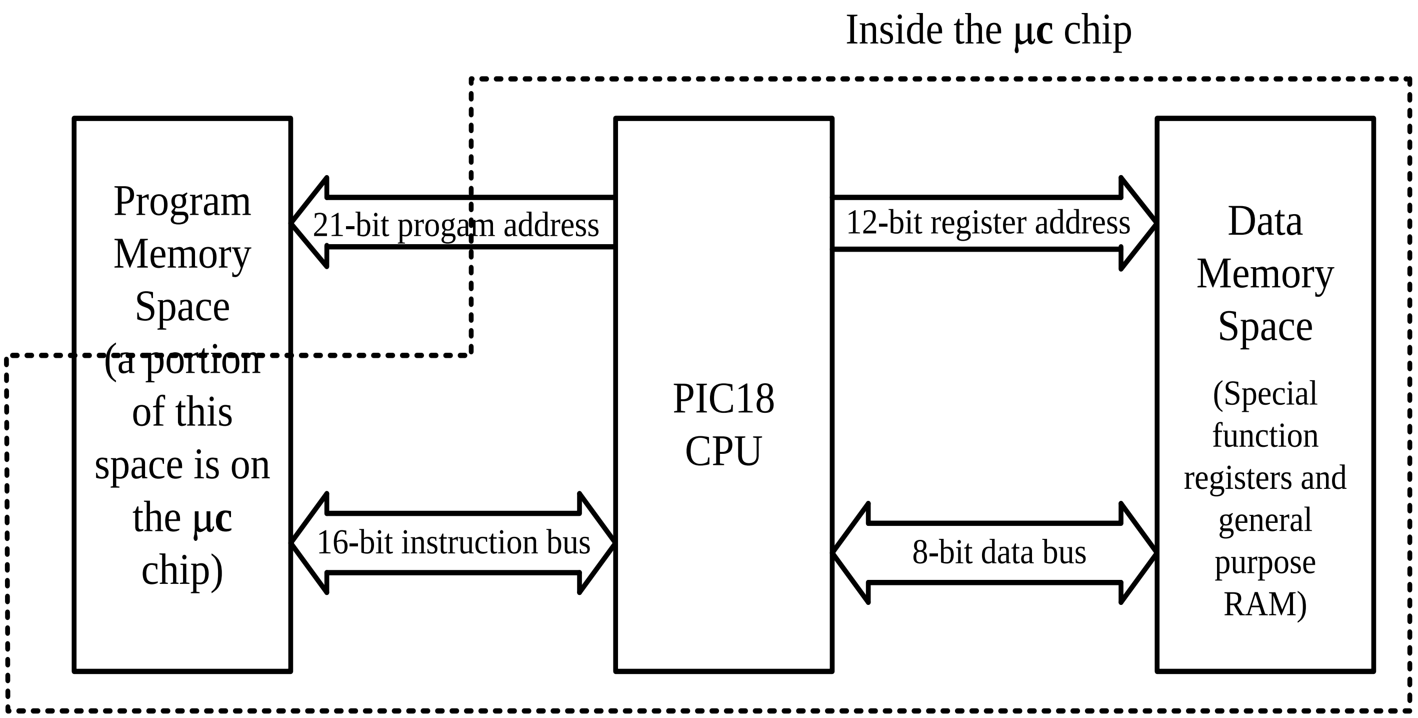
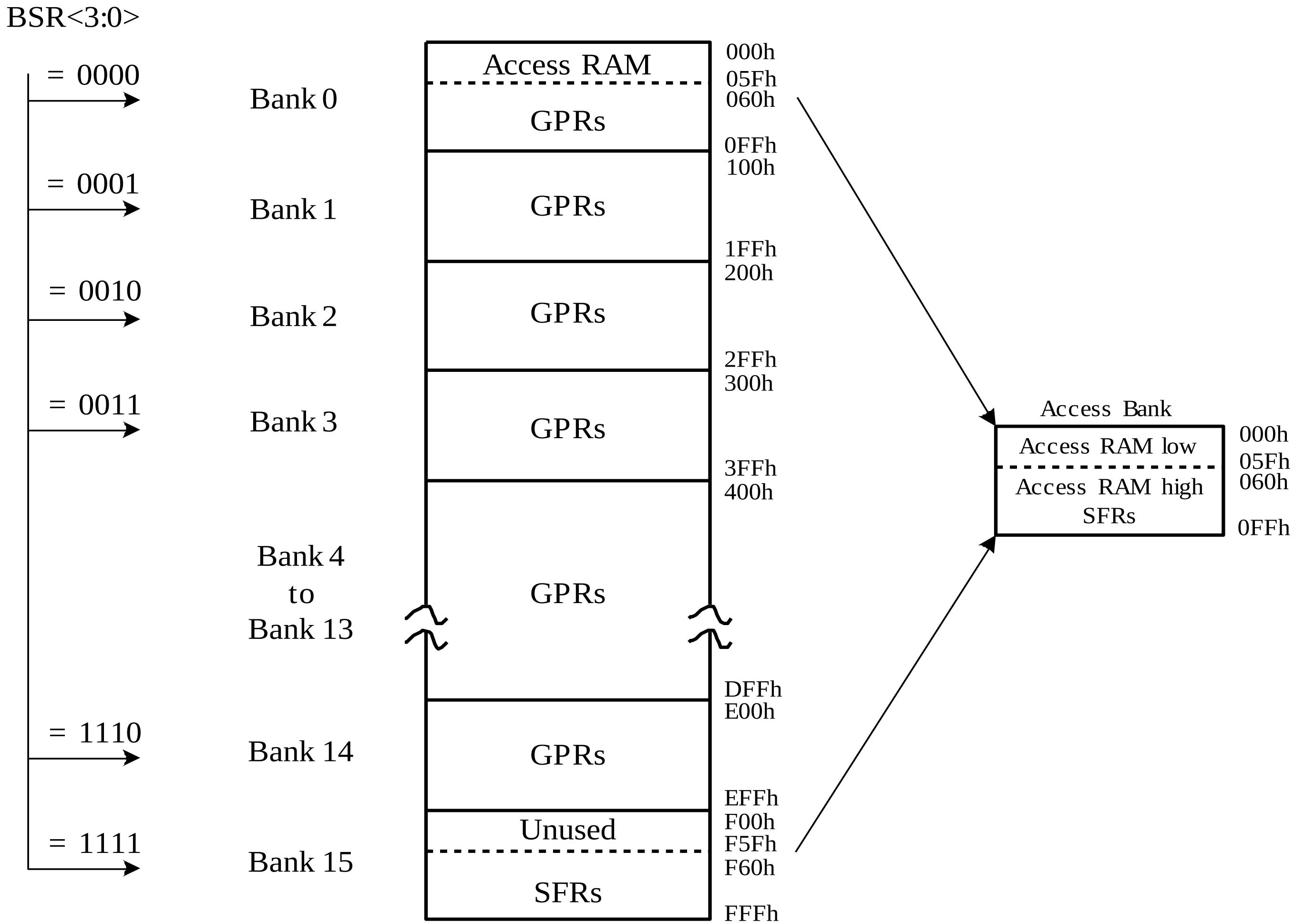


Figure 1.3 The PIC18 memory spaces

PIC18 Data Memory

- Implemented in SRAM and consists of **general-purpose registers** and **special-function registers**. Both are referred to as **data registers**.
- A PIC18 MCU may have up to 4096 bytes of data memory.
- Data memory is divided into banks. Each bank has 256 bytes.
- General-purpose registers are used to hold dynamic data.
- Special-function registers are used to control the operation of peripheral functions.
- Only one bank is active at any time. The active bank is specified by the BSR register.
- Bank switching is an overhead and can be error-prone
- PIC18 implements the **access bank** to reduce the problem caused by bank switching.
- **Access bank** consists of the lowest 96 bytes and the highest 160 bytes of the data memory space.

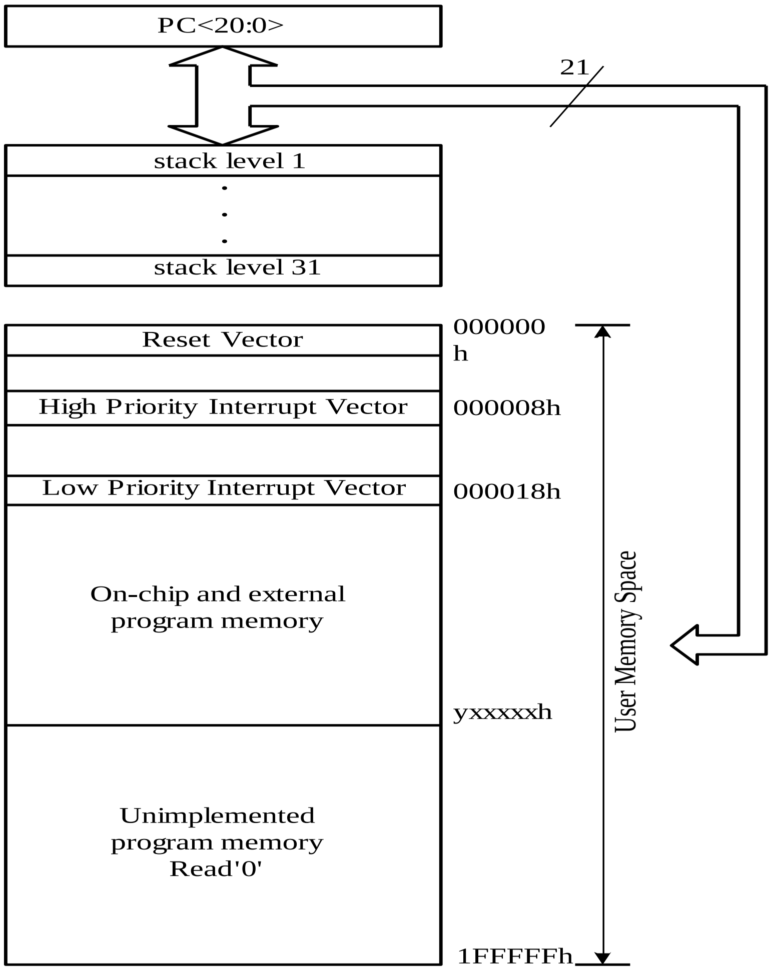


Note. 1. **BSR** is the 4-bit bank select register.

Figure 1.4 Data memory map for PIC18 devices (redraw with permission of Microchip)

Program Memory Organization

- The program counter (PC) is 21-bit long, which enables the user program to access up to 2 MB of program memory.
- The PIC18 has a 31-entry return address stack to hold the return address for subroutine call.
- After power-on, the PIC18 starts to execute instructions from address 0.
- The location at address 0x08 is reserved for high-priority interrupt service routine.
- The location at address 0x18 is reserved for low-priority interrupt service routine.
- Up to 128KB (at present time) of program memory is inside the MCU chip.
- Part of the program memory is located outside of the MCU chip.



Note. **y** can be 0 or 1 whereas **x** can be 0-F

Figure 1.5 PIC18 Program memory Organization (redraw with permission of Microchip)

The PIC18 CPU Register

- The group of registers from 0xFD8 to 0xFFF are dedicated to the general control of MCU operation.
- The CPU registers are listed in Table 1.2.
- The WREG register is involved in the execution of many instructions.
- The STATUS register holds the status flags for the instruction execution and is shown in Figure 1.6.

Table 1.2 PIC18 CPU registers

address	Name	Description
0xFFF	TOSU	Top of stack (upper)
0xFFE	TOSH	Top of stack (high)
0xFFD	TOSL	Top of stack (low)
0xFFC	STKPTR	Stack pointer
0xFFB	PCLATU	Upper program counter latch
0xFFA	PCLATH	High program counter latch
0xFF9	PCL	Program counter low byte
0xFF8	TBLPTRU	Table pointer upper byte
0xFF7	TBLPTRH	Table pointer high byte
0xFF6	TBLPTRL	Table pointer low byte
0xFF5	TABLAT	Table latch
0xFF4	PRODH	High product register
0xFF3	PRODL	Low product register
0xFF2	INTCON	Interrupt control register
0xFF1	INTCON2	Interrupt control register 2
0xFF0	INTCON3	Interrupt control register 3
0xFEF	INDF0 (1)	Indirect file register pointer 0
0xFEE	POSTINC0 (1)	Post increment pointer 0 (to GPRs)
0xFED	POSTDEC0 (1)	Post decrement pointer 0 (to GPRs)
0xFEC	PREINC0 (1)	Preincrement pointer 0 (to GPRs)
0xFEB	PLUSW0 (1)	Add WREG to FSR0
0xFEA	FSR0H	File select register 0 high byte
0xFE9	FSR0L	File select register 0 low byte
0xFE8	WREG	Working register
0xFE7	INDF1 (1)	Indirect file register pointer 1
0xFE6	POSTINC1 (1)	Post increment pointer 1 (to GPRs)
0xFE5	POSTDEC1 (1)	Post decrement pointer 1 (to GPRs)
0xFE4	PREINC1 (1)	Preincrement pointer 1 (to GPRs)
0xFE3	PLUSW1 (1)	Add WREG to FSR1
0xFE2	FSR1H	File select register 1 high byte
0xFE1	FSR1L	File select register 1 low byte
0xFE0	BSR	Bank select register
0xFDF	INDF2 (1)	Indirect file register pointer 2
0xFDE	POSTINC2 (1)	Post increment pointer 2 (to GPRs)
0xFDD	POSRDEC2 (1)	Post decrement pointer 2 (to GPRs)
0xFDC	PREINC2 (1)	Preincrement pointer 2 (to GPRs)
0xFDB	PLUSW2 (1)	Add WREG to FSR2
0xFDA	FSR2H	File select register 2 high byte
0xFD9	FSR2L	File select register 2 low byte
0xFD8	STATUS	Status regoster

Note 1. This is not a physical regiser

7	6	5	4	3	2	1	0
--	--	--	N	OV	Z	DC	C

N: Negative bit
 1 = arithmetic result is negative
 0 = arithmetic result is positive

OV: Overflow bit
 1 = Overflow occurred for signed arithmetic
 0 = No overflow occurred

Z: Zero flag
 1 = The result of an arithmetic or logic operation is zero.
 0 = The result of an arithmetic or logic operation is not zero.

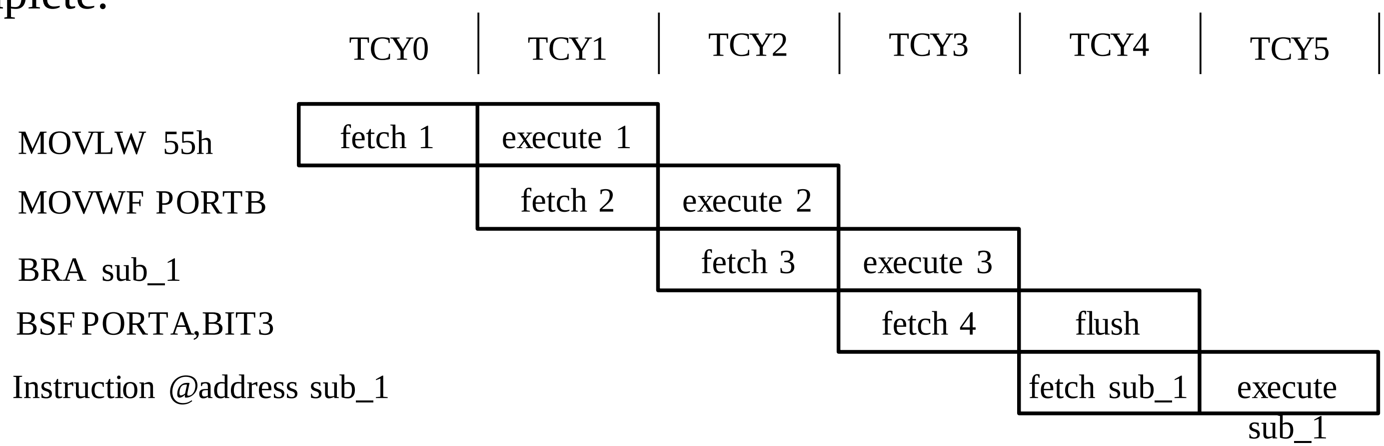
DC: Digit carry/borrow bit
 For ADDWF, ADDLW, SUBLW, SUBWF instructions.
 1 = A carry-out from the 4th low-order bit of the result occurred.
 0 = No carry-out from the 4th low-order bit of the result occurred.
 For borrow, the polarity is reversed. For rotate (RRF, RLF) instructions, this bit is loaded with either the bit 4 or bit 3 of the source register.

C: Carry/borrow bit
 For ADDWF, ADDLW, SUBLW, SUBWF instructions.
 1 = A carry-out from the most significant bit of the result occurred.
 0 = No carry-out from the most significant bit of the result has occurred.
 For borrow, the polarity is reversed. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Figure 1.6 The STATUS register (0xFD8) (redraw with permission of Microchip)

The PIC18 Pipelining

- The PIC18 Divide most of the instruction execution into two stages: instruction fetch and instruction execution.
- Up to two instructions are overlapped in their execution. One instruction is in fetch stage while the second instruction is in execution stage.
- Because of pipelining, each instruction appears to take one instruction cycle to complete.



Note: All instructions are single cycle, except for any program branches.

Figure 1.7 An example of instruction pipeline flow

Instruction Format

- Format for **byte oriented** instructions

15 ... 10	9	8	7 ... 0
opcode	d	a	f

d = 0 for result destination to be WREG register.
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Figure 1.8 Byte-oriented file register operations (redraw with permission of Microchip)

Byte-to-byte Operations

15 … 12	11 … 0
opcode	f (source file register)

15 … 12	11 … 0
1111	f (destination file register)

f = 12-bit file register address

Figure 1.9 Byte to byte move operations (2 words) (redraw with permission of Microchip)

Bit-oriented file register operations

15 … 12	11 … 9	8	7 … 0
opcode	b	a	f

b = 3-bit position of bit in the file register (f).
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Figure 1.10 Bit-oriented file register operations (redraw with permission of Microchip)

Literal operations

- A literal is a number to be operated on directly by the CPU

15 8	7 0
opcode	k

k = 8-bit immediate value

Figure 1.11 Literal operations (redraw with permission of Microchip)

Control operations

- These instructions are used to change the program execution sequence and making subroutine calls.

15	8	7	0	
opcode		n<7:0> (literal)		

15	8	7	0	
1111	n<19:8> (literal)			GOTO label

n = 20-bit immediate value

15	8		7	0	
opcode		S	n<7:0> (literal)		

15	8	7	0	
1111	n<19:8> (literal)			CALL funct_name

S = fast bit

15	11	10	0	
opcode		n<10:0> (literal)		BRA label

15	8	7	0	
opcode		n<7:0> (literal)		BC label

Figure 1.12 Control operations (redraw with permission of Microchip)

Access Bank

- In Figures 1.8 to 1.12, PIC18 uses 8 bits to specify a data register (**f** field).
- Eight bits can specify only 256 registers.
- This limitation forces the PIC18 to divide data registers (up to 4096 bytes) into banks.
- Only one bank is active at a time.
- When operating on a data register in a different bank, bank switching is needed.
- Bank switching incurs overhead and may cause program errors.
- Access bank is created to minimize the problems of bank switching.
- Access bank consists of the lowest 96 bytes in general-purpose registers and the highest 160 bytes of special function registers.
- When operands are in the access bank, no bank switching is needed.

Examples of the Use of Access Bank

```
1. addwf   0x20,F,A       ; add the data register at 0x20 in access bank with WREG
                          ; register and store the sum in 0x20.
2. subwf   0x30,F,BANKED  ; subtract the value of WREG from the data register
                          ; 0x30 in the bank specified by the current contents
                          ; of the BSR register. The difference is stored in
                          ; data register 0x30.
3. addwf   0x40,W,A       ; add the WREG register with data register at 0x40 in
                          ; access bank and leaves the sum in WREG.
```

PIC18 Addressing Modes

- **Register direct**: Use an 8-bit value to specify a data register.

```
movwf  0x20,A        ; the value 0x20 is register direct mode
```

- **Immediate Mode** : A value in the instruction to be used as an operand

```
addlw  0x10          ; add hex value 0x10 to WREG

movlw  0x30          ; load 0x30 into WREG
```

- **Inherent Mode:** an implied operand

```
andlw  0x3C          ; the operand WREG is implied

daw                  ; the operand WREG is implied
```

- **Indirect Mode:** A special function register (FSRx) is used as a pointer to the actual data register.

Format		Example	
INDFx	x = 0, 1, 2	movwf	INDF0
POSTINCx		movff	POSTINC0,PRODL
POSTDECx		movf	POSTDEC0,W
PREINCx		addwf	PREINC1,F
PLUSWx		movff	PLUSW2,PRODL

PIC18 Instruction Examples

Data Movement Instruction

lfsr	FSR1,0xB00	; place the value 0xB00 in FSR1
movf	PRODL,W	; copy PRODL into WREG
movff	0x100,0x300	; copy data register 0x100 to data register 0x300
movwf	PRODL,A	; copy WREG to PRODL
swapf	PRODL,F	; swap the upper and lower 4 bits of PRODL
movb	3	; load 3 into BSR
movlw	0x10	; WREG $\leftarrow$ 0x10

Add Instructions

addwf	0x20,F,A	; add data register and WREG and place sum in 0x20
addwfc	PRODL,W,A	; add WREG, PRODL, and carry and leave sum ; in WREG
addlw	0x5	; increment WREG by 5

Subtract Instructions

subfwb	PRODL,F	; PRODL ← [WREG] – [PRODL] – borrow flag
subwf	PRODH,W	; WREG ← [PRODH] – [WREG]
subwfb	0x10,F,A	; 0x10 ← [0x10] – [WREG] – borrow flag
sublw	0x10	; WREG ← 0x10 – [WREG]

RISC	CISC
Simple instruction set	Complex instruction set
Regular and fixed instruction format	Irregular instruction format
Simple address modes	Complex address modes
Pipelined instruction execution	May also pipeline instruction execution
Separated data and program memory	Combined data and program memory
Most operations are register to register	Most operations can be register to memory
Take shorter time to design and debug	Take longer time to design and debug
Provide large number of CPU registers	Provide smaller number of CPU registers