

1

Introduction to the PIC18 Microcontroller

1.1 Objectives

After completing this chapter, you should be able to:

- Define or explain the following terms: computer, processor, microprocessor, microcontroller, hardware, software, cross assembler, cross compiler, RAM, SRAM, ROM, EPROM, EEPROM, flash memory, byte, nibble, bus, KB, MB, GB, mnemonic, opcode, and operand
- Explain the differences among all of the PIC18 addressing modes
- Use appropriate PIC18 addressing mode to access operands
- Explain the banking operation of PIC18 data memory
- Perform simple operations using add, subtract, and data movement instructions

1.2 What Is a Computer?

A computer is made up of hardware and software. The hardware of a computer consists of four types of components:

- *Processor.* The processor is responsible for performing all of the computational operations and the coordination of the usage of resources of a computer. A computer system may consist of one or multiple processors. A processor may perform general-purpose computations or special-purpose computations, such as graphical rendering, printing, or network processing.
- *Input devices.* A computer is designed to execute programs that manipulate certain data. Input devices are needed to enter the program to be executed and data to be processed into the computer. There are a wide variety of input devices: keyboards, keypads, scanners, bar code readers, sensors, and so on.
- *Output devices.* No matter if the user uses the computer to do certain computation or to find information from the Internet or a database, the end results must be displayed or printed on paper so that the user can see them. There are many media and devices that can be used to present the information: CRT displays, flat-panel displays, seven-segment displays, printers, light-emitting diodes (LEDs), and so on.
- *Memory devices.* Programs to be executed and data to be processed must be stored in memory devices so that the processor can readily access them.

1.2.1 The Processor

A processor is also called the *central processing unit* (CPU). The processor consists of at least the following three components:

Registers. A register is a storage location inside the CPU. It is used to hold data and/or a memory address during the execution of an instruction. Because the register is very close to the CPU, it can provide fast access to operands for program execution. The number of registers varies greatly from processor to processor.

Arithmetic logic unit (ALU). The ALU performs all the numerical computations and logical evaluations for the processor. The ALU receives data from the memory, performs the operations, and, if necessary, writes the result back to the memory. Today's supercomputer can perform trillions of operations per second. The ALU and registers together are referred to as the *datapath* of the processor.

Control unit. The control unit contains the hardware instruction logic. The control unit decodes and monitors the execution of instructions. The control unit also acts as an arbiter as various portions of the computer system compete for the resources of the CPU. The activities of the CPU are synchronized by the system clock. The clock rates of modern microprocessors have exceeded 3.0 GHz at the time of this writing, where

1 GHz = 1 billion cycles per second

The period of a 1-GHz clock signal is 1 ns (10^{-9} second). The control unit also maintains a register called the *program counter* (PC) that keeps track of the address of the next instruction to be executed. During the execution of an instruction, the occurrence of an overflow, an addition

carry, a subtraction borrow, and so forth are flagged by the system and stored in another register called a *status register*. The resultant flags are then used by the programmer for program-flow control and decision making.

1.2.2 The Microprocessor

The advancement of semiconductor technology allows the circuitry of a complete processor to be placed in one integrated circuit (also called a *chip*). A *microprocessor* is a processor packaged in a single integrated circuit. A *microcomputer* is a computer that uses a microprocessor as its CPU. A personal computer (PC) is a microcomputer. Early microcomputers were very slow. However, many personal computers manufactured in 2003 run at a clock rate higher than 3.0 GHz and are faster than some supercomputers of a few years ago.

Depending on the number of bits that a microprocessor can manipulate in one operation, a microprocessor is referred to as 4-bit, 8-bit, 16-bit, 32-bit, or 64-bit. This number is the *word length* (or *datapath length*) of the microprocessor. Currently, the most widely used microprocessors are 8-bit.

Although the clock rate of the microprocessor has been increased dramatically, the improvement in the *access time* (or simply called the speed) of the high-capacity memory chips (especially the most widely used DRAM chips to be discussed in Section 1.2.4) has been moderate at best. The microprocessor may complete one arithmetic operation in one clock cycle; however, it may take many clock cycles to access data from the memory chip. This disparity in speed makes the high clock rate of the microprocessor alone useless for achieving high throughput. The solution to this issue is adding a small high-speed memory to the CPU chip. This on-chip memory is called *cache memory*. The CPU can access data from the on-chip cache memory in one or two clock cycles because it is very close to the ALU. The cache memory is effective in improving the average memory access time because the CPU demonstrates *locality* in its access behavior. Within a short period of time, the CPU tends to access a small area in the memory repeatedly. Once the program segment or data has been brought into the cache, it will be referenced many times. This results in an average memory access time very close to that of the access time of the cache memory.

Microprocessors and input/output (I/O) devices have different characteristics and speed. A microprocessor is not designed to deal with I/O devices directly. Instead, peripheral chips (also called *interface chips*) are needed to make up the difference between the microprocessor and the I/O devices. For example, the Intel i8255 was designed to interface the 8-bit 8080 microprocessor from Intel, and the M6821 was designed to interface the 8-bit 6800 from Motorola with I/O devices.

Microprocessors have been widely used in many applications since they were invented. However, there are several limitations in the initial microprocessor designs that led to the development of microcontrollers:

- External memory chips are needed to hold programs and data because the early microprocessors did not have on-chip memory.
- Glue logic (such as address decoder and buffer chips) is required to interface with the memory chips.
- Peripheral chips are needed to interface with I/O devices.

Because of these limitations, a product designed with microprocessors cannot be made as compact as might be desirable. The development of microcontrollers has not only eliminated most of these problems but also enabled the design of many low-cost microprocessor-based products.

1.2.3 Microcontrollers

A microcontroller, or MCU, is a computer implemented on a single very large scale integrated (VLSI) circuit. In addition to those components contained in a microprocessor, an MCU also contains some of the following peripheral components:

- Memory
- Timers, including event counting, input capture, output compare, real-time interrupt, and watchdog timer
- Pulse-width modulation (PWM)
- Analog-to-digital converter (ADC)
- Digital-to-analog converter (DAC)
- Parallel I/O interface
- Asynchronous serial communication interface (UART)
- Synchronous serial communication interfaces (SPI, I²C, and CAN)
- Direct memory access (DMA) controller
- Memory component interface circuitry
- Software debug support hardware

The discussion of the functions and applications of these components is the subject of this text. Most of these functions are discussed in details in later chapters.

Since their introduction, MCUs have been used in almost every application that requires certain amount of intelligence. They are used as controllers for displays, printers, keyboards, modems, charge card phones, palm-top computers, and home appliances, such as refrigerators, washing machines, and microwave ovens. They are also used to control the operation of engines and machines in factories. One of the most important applications of MCUs is probably the automobile control. Today, a luxurious car may use more than 100 MCUs. Today, most homes have one or more MCU-controlled consumer electronics appliances. In these applications, people care about only the functionality of the end product rather than the MCUs being used to perform the control function. Products of this nature are often called *embedded systems*.

1.2.4 Memory

Programs and data are stored in memory in a computer system. A computer may contain semiconductor, magnetic, and/or optical memories. Only semiconductor memory is discussed in this text because magnetic and optical memories are seldom used in 8-bit MCU applications. Semiconductor memory can be further classified into two major types: *random-access memory* (RAM) and *read-only memory* (ROM).

RANDOM-ACCESS MEMORY

Random-access memory is *volatile* in the sense that it cannot retain data in the absence of power. RAM is also called *read/write memory* because it allows the processor to read from and write into it. Both read and write accesses to a RAM chip take roughly the same amount of time. As long as the power is on, the microprocessor can write data into a location in the RAM chip and read back the same contents later. Reading memory is nondestructive. When the microprocessor writes data to memory, the old data is written over and destroyed.

There are two types of RAM technologies: *static RAM* (SRAM) and *dynamic RAM* (DRAM). SRAM uses from four to six transistors to store one bit of information. As long as power is on,

the information stored in the SRAM will not be degraded. Dynamic RAM uses one transistor and one capacitor to store one bit of information. The information is stored in the capacitor in the form of electric charge. The charge stored in the capacitor will leak away over time, so a periodic refresh operation is required to maintain the contents of DRAM.

RAM is mainly used to store *dynamic* programs and data. A computer user often wants to run different programs on the same computer, and these programs usually operate on different sets of data. The programs and data must therefore be loaded into RAM from hard disk or other secondary storage, and for this reason they are called *dynamic*.

READ-ONLY MEMORY

ROM is nonvolatile. If power is removed from ROM and then reapplied, the original data will still be there. As its name implies, ROM data can only be read. This is not exactly true. Most ROM technologies require special algorithm and voltage to write data into the chip. Without using the special algorithm and voltage, any attempt to write to the ROM memory will not be successful. There are many different kinds of ROM technologies in use today:

Masked-programmed read-only memory (MROM) is a type of ROM that is programmed when it is manufactured. The semiconductor manufacturer places binary data in the memory according to customer's specification. To be cost effective, many thousands of MROM memory chips, each containing a copy of the same data (or program), must be sold. Many people simply call MROM as ROM.

Programmable read-only memory (PROM) is a type of read-only memory that can be programmed in the field (often by the end user) using a device called a PROM programmer or PROM *burner*. Once a PROM has been programmed, its contents cannot be changed. PROMs are fuse-based; that is, end users program the fuses to configure the contents of memory.

Erasable programmable read-only memory (EPROM) is a type of read-only memory that can be erased by subjecting it to strong ultraviolet light. The circuit design of EPROM requires the user to erase the contents of a location before a new value can be written into it. A quartz window on top of the EPROM integrated circuit permits ultraviolet light to be shone directly on the silicon chip inside. Once the chip is programmed, the window can be covered with dark tape to prevent gradual erasure of the data. If no window is provided, the EPROM chip becomes *one-time programmable* (OTP) only. EPROM is often used in prototype computers where the software may be revised many times until it is perfected. EPROM does not allow erasure of the contents of an individual location. The only way to make change is to erase the entire EPROM chip and reprogram it. The programming of an EPROM chip is done electrically by using a device called an *EPROM programmer*. Today, most programmers are universal in the sense that they can program many different types of devices including EPROM, EEPROM, flash memory, and *programmable logic devices*.

Electrically erasable programmable read-only memory (EEPROM) is a type of nonvolatile memory that can be erased and reprogrammed by electrical signals. Like EPROM, the circuit design of EEPROM also requires the user to erase the contents of a memory location before writing a new value into it. EEPROM allows each individual location to be erased and reprogrammed. Unlike EPROM, EEPROM can be erased and programmed using the same programmer. However, EEPROM pays the price for being so flexible in its erasability. The cost of an EEPROM chip is much higher than that of an EPROM chip of comparable density.

Flash memory was invented to incorporate the advantages and avoid the disadvantages of both EPROM and EEPROM technologies. Flash memory can be erased and reprogrammed in the system without using a dedicated programmer. It achieves the density of EPROM, but it does not require a window for erasure. Like EEPROM, flash memory can be programmed and erased

electrically. However, it does not allow the erasure of an individual memory location—the user can only erase a section or the entire chip. Today, more and more MCUs are incorporating on-chip flash memory for storing programs and data. The flash-based PIC18 MCUs allow you to erase one block of 64 bytes at a time.

1.3 The Computer Software

Programs are known as *software*. A program is a set of instructions that the computer can execute. The program is stored in the computer's memory in the form of binary numbers called *machine instructions*.

The length of a machine instruction of a computer may be fixed or variable. Fixing the instruction length makes instruction decoding simpler and hence can simplify the design of the processor. However, it has one potential drawback. The program length may be longer because of the inefficiency of instruction encoding. Most of the PIC18 instructions are 16 bits, whereas four of them are 32 bits. For example, the PIC18 machine instruction

0010 0100 0010 0000 (or 2420 in base 16)

adds the contents of the data register at the hexadecimal address 20 to the WREG register and leaves the sum in WREG. The machine instruction

0110 1010 0000 0101 (or 6A05 in base 16)

clears the contents of the data register located at the address 5 to 0.

When a machine instruction is fetched from the memory, it will be decoded in the control unit of the CPU. Appropriate control signals will then be generated to trigger the desired operation.

1.3.1 Assembly Language

It is not difficult to conclude that software development in machine language is extremely hard:

1. *Program entering.* The programmer must use the binary patterns of every machine instruction in order to enter a machine instruction. Before the user can memorize the binary pattern of each instruction, he or she must consult a lookup table constantly. In addition, the programmer must work on the program logic at a very low level, which will hinder the programming productivity.
2. *Program debugging.* Whenever a program does not perform as expected, the programmer will have a hard time to identify the instruction that caused the problem. A programmer will need to identify each machine instruction and then think about what operation is performed by that instruction. This is not an easy task.
3. *Program maintenance.* Most programs will need to be maintained in the long run. A programmer who did not write the program will have a hard time reading the program and figuring out the program logic.

Assembly language was invented to simplify the programming job. An *assembly program* consists of assembly instructions. An assembly instruction is the mnemonic representation of a machine instruction. For example, in the PIC18 MCU,

decf fp_cnt,F,A stands for “decrement the variable *fp_cnt* located at the access bank by 1”

addwf sum,F,A stands for “add the contents of the WREG register and the variable *sum* in the access bank and leaves the result in *sum*.”

where the meaning of *access bank* is explained in Section 1.5.2.

With the invention of the assembly language, a programmer no longer needs to scan through the sequence of 0s and 1s in order to identify what instructions are in the program. This is a significant improvement over machine language programming.

The assembly program that the programmer enters is called *source program* or *source code*. The user needs to invoke an *assembler* program to translate the source program into machine language so that the computer can execute it. The output of an assembler is also called *object code*. There are two types of assemblers: *native assembler* and *cross assembler*. A native assembler runs on a computer and generates the machine code to be executed on the same computer or a different computer having the same instruction set. A cross assembler runs on a computer but generates machine code that will be executed by computers that have a different instruction set. The Microchip MPASM® is a cross assembler designed to run on a PC to translate assembly programs for the PIC MCUs.

1.3.2 High-Level Languages

There are a few drawbacks for assembly language programming:

- The programmer must be familiar with the hardware architecture on which the program is to be executed.
- A program (especially a long one) written in assembly language is difficult to understand for anyone other than the author.
- Programming productivity is not satisfactory for large programming projects because the programmer needs to work on the program logic at a very low level.

For these reasons, high-level languages such as C, C++, and Java were invented to avoid the problems of assembly language programming. High-level languages are close to plain English, and hence a program written in a high-level language becomes easier to understand. A statement in high-level language often needs to be implemented by tens or even hundreds of assembly instructions. The programmer can now work on the program logic at a much higher level, which makes the programming job much easier. A program written in a high-level language is also called a *source code*, and it requires a software program called a *compiler* to translate it into machine instructions. A compiler compiles a program into *object code*. Just as there are cross assemblers, there are *cross compilers* that run on one machine but translate programs into machine instructions to be executed on a computer with a different instruction set.

Some high-level languages are *interpreted*; that is, they use an *interpreter* to scan the user's source code and perform the operations specified. Interpreters do not generate object code. Programming languages that use this approach include Basic, Lisp, and Prolog. The Java language is partially compiled and partially interpreted. A program written in Java language is first compiled into byte code and then interpreted. The design purpose of this language is "compiled once, run everywhere."

High-level languages are not perfect, either. One of the major problems with high-level languages is that the machine code compiled from a program written in a high-level language is much longer and cannot run as fast as its equivalent in the assembly language. For this reason, many time-critical programs are still written in assembly language.

C language has been used extensively in MCU programming in the industry, and most MCU software tool developers provide cross C compilers. Both the C and the PIC18 assembly languages will be used throughout this text. The C programs in this text are compiled by the Microchip C cross compiler and tested on the PIC18 demo board.

1.4 Overview of the PIC18 MCU

Microchip has introduced six different lines of 8-bit MCUs over the years:

1. PIC12XXX: 8-pin, 12- or 14-bit instruction format
2. PIC14000: 28-pin, 14-bit instruction format (same as PIC16XX)
3. PIC16C5X: 12-bit instruction format
4. PIC16CXX: 14-bit instruction format
5. PIC17: 16-bit instruction format
6. PIC18: 16-bit instruction format

Each line of the PIC MCUs support different number of instructions with slightly different instruction formats and different design in their peripheral functions. This makes products designed with a different family of PIC MCUs incompatible. The members of the PIC18 family share the same instruction set and the same peripheral function design and provide from eight to more than 80 signal pins. This makes it possible to upgrade the PIC18-based product without changing the MCU family. One of the design goals of the PIC18 MCU is to eliminate the design flaws of other earlier MCU families and provide a better upgrade path to other families of MCUs. In terms of cost, the PIC18 MCUs are not more expensive than those in other families with similar capability.

The PIC18 MCUs provide the following peripheral functions:

1. Parallel I/O ports
2. Timer functions, including counters, input capture, output compare, real-time interrupt, and watchdog timer
3. Pulse width modulation (PWM)
4. SPI and I²C serial interface
5. Universal Synchronous/Asynchronous Receiver Transmitter (USART)
6. A/D converter with 10-bit resolution
7. Analog comparator
8. Low-power operation mode
9. SRAM and EEPROM
10. EPROM or flash memory
11. Controller Area Network (CAN)

These peripheral functions are studied in detail in the following chapters. By October 2003, 40 devices in the PIC18 family have been in production, and more devices will be introduced in the coming few years. All these MCUs implement 77 instructions. Among them, 73 instructions are 16 bits, and the remaining four are 32 bits.

The features of all PIC18 devices are shown in Table 1.1. This table mentions many acronyms that may not make any sense at this point. However, all of them are explained in detail in later chapters.

Feature	PIC18C242	PIC18C252	PIC18C442	PIC18C452	PIC18C601	PIC18C801	PIC18C658
Operating frequency	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-25 MHz	DC-25 MHz	DC-40 MHz
Program memory	16 KB	32 KB	16 KB	32 KB	0 KB	0 KB	32 KB
Data memory	512 Bytes	1.5 KB	512 Bytes	1.5 KB	1.5 KB	1.5 KB	1.5 KB Data
EEPROM	0	0	0	0	0	0	0
External program memory	No	No	No	No	256 KB	2 MB	No
Interrupt sources	16	16	17	17	15	15	21
I/O ports	A..C	A..C	A..E	A..E	A..G	A..H, J	A..G
Timers	4	4	4	4	4	4	4
Capture/Compare/ PWM modules	2	2	2	2	2	2	2
Serial Communication	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART, CAN
Parallel Communication	No	No	PSP	PSP	No	No	PSP
10-bit A/D	5 channels	5 channels	8 channels	8 channels	8 channels	12 channels	12 channels
Low voltage detect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Brown out reset	Yes	Yes	Yes	Yes	No	No	Yes
Instruction set	77	77	77	77	77	77	77
8-bit external memory	No	No	No	NO	YES	YES	NO
8-bit external demuxed memory	No	No	No	NO	No	YES	NO
8-bit external memory	No	No	No	NO	YES	YES	NO
On-chip chip select signal	No	No	No	NO	CS1	CS1, CS2	NO
Packages	28-pin PDIP 28-pin SOIC 28-pin JW	28-pin DIP 28-pin SOIC 28-pin JW	40-pin DIP 40-pin PLCC 40-pin TQFP 40-pin JW	40-pin DIP 44-pin PLCC 44-pin TQFP 40-pin JW	64-pin TQFP 68-pin PLCC	80-pin TQFP 84-pin PLCC	64-pin TQFP 68-pin CERQUAD 68-pin PLCC
Feature	PIC18C858	PIC18F242	PIC18F252	PIC18F442	PIC18F452	PIC18F258	PIC18F458
Operating frequency	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz
Program memory	32 KB	16 KB	32 KB	16 KB	32 KB	32 KB	32 KB
Data memory	1.5 KB	768 Bytes	1.5 KB	768 Bytes	1.5 KB	1.5 KB	1.5 KB
Data EEPROM	0	256	256	256	256	256	256
External program memory	No	No	No	No	No	No	No
Interrupt sources	21	17	17	18	18	17	21
I/O ports	A..H, J, K	A..C	A..C	A..E	A..E	A..C	A..E
Timers	4	4	4	4	4	4	4
Capture/Compare/PWM modules	2	2	2	2	2	1	1
Enhanced Capture/ Compare/PWM modules	0	0	0	0	0	0	1
Serial Communication	MSSP, USART, CAN	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART, CAN	MSSP, USART, CAN
Parallel Communication	PSP	No	No	PSP	PSP	No	PSP
10-bit A/D	16 channels	5 channels	5 channels	8 channels	8 channels	5 channels	8 channels
Low voltage detect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Brown out reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Instruction set	77	77	77	77	77	77	77
Packages	80-pin TQFP 84-pin CERQUAD 80-pin PLCC	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin PLCC	40-pin DIP 40-pin PLCC 44-pin TQFP	40-pin TQFP 44-pin PLCC 44-pin TQFP	28-pin SPDIP 28-pin SOIC	40-pin PDIP 44-pin PLCC 44-pin TQFP

Table 1.1 ■ Features of the PIC18 Microcontrollers (continued)

Feature	PIC18F248	PIC18F448	PIC18F6620	PIC18F6720	PIC18F8620	PIC18F8720	PIC18F1220
Operating frequency	DC-40 MHz	DC-25 MHz	DC-25 MHz	DC-25 MHz	DC-25 MHz	DC-250 MHz	DC-40 MHz
Program memory	16 KB	16 KB	64 KB	128 KB	64 KB	128 KB	4 KB
Data memory	768 Bytes	768 KB	3840 Bytes	3840 Bytes	3840 Bytes	3840 Bytes	256 Bytes
Data EEPROM	256 Bytes	256 Bytes	1024 Bytes	1024 Bytes	1024 Bytes	1024 Bytes	256 Bytes
External program memory	No	No	No	No	Yes	Yes	No
Interrupt sources	17	21	17	17	18	18	15
I/O ports	A..C	A..E	A..G	A..G	A..H, J	A..H, J	A..B
Timers	4	4	5	5	5	5	4
Capture/Compare/ PWM modules	1	1	5	5	5	5	1
Enhanced Capture/ Compare/PWM modules	0	1	0	0	0	0	0
Serial Communication	MSSP, USART, CAN	MSSP, USART, CAN	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART	USART
Parallel Communication	No	PSP	PSP	PSP	PSP	PSP	No
10-bit A/D module	5 channels	8 channels	12 channels	12 channels	16 channels	16 channels	7 channels
Low voltage detect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Brown out reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Instruction set	77	77	77	77	77	77	77
Packages	28-pin SPDIP 28-pin SOIC	40-pin PDIP 44-pin PLCC 44-pin TQFP	64-pin TQFP	64-pin TQFP	80-pin TQFP	80-pin TQFP	18-pin SDIP 18-pin SOIC 20-pin SSOP 28-pin QFN
Feature	PIC18F1320	PIC18F2220	PIC18F2320	PIC18F2439	PIC18F2539	PIC18F8520	PIC18F8525
Operating frequency	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz
Program memory	8 KB	4 KB	8 KB	12 KB	24 KB	32 KB	48 KB
Data memory	256 Bytes	512 Bytes	512 Bytes	640 Bytes	1408 Bytes	2048 Bytes	3840 Bytes
Data EEPROM	256 Bytes	256 Bytes	256 Bytes	256 Bytes	256 Bytes	1024 Bytes	1024 Bytes
External program memory	No	No	No	No	No	Yes	Yes
Interrupt sources	15	19	19	15	15	18	17
I/O ports	A..B	A..C	A..C	A..C	A..C	A..H, J	A..H, J
Timers	4	4	4	3	3	5	5
Capture/Compare/PWM modules	1	2	2	2	2	5	2
Enhanced Capture/ Compare/PWM modules	0	0	0	0	0	0	3
Serial Communication	EUSART	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART (2)	MSSP, USART (2)
Parallel Communication	No	No	No	No	No	PSP	PSP
10-bit A/D module	7 channels	10 channels	10 channels	5 channels	5 channels	16 channels	12 channels
Low voltage detect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Brown out reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Instruction set	77	77	77	77	77	77	77
Packages	18-pin SDIP 18-pin SOIC 20-pin SSOP 28-pin QFN	28-pin SDIP 28-pin SOIC	28-pin SDIP 28-pin SOIC	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	80-pin TQFP	80-pin TQFP

Table 1.1 ■ Features of the PIC18 Microcontrollers (continued)

Feature	PIC18F4439	PIC18F4539	PIC18F6520	PIC18F6525	PIC18F6585	PIC18F6621	PIC18F6680
Operating frequency	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz
Program memory	12 KB	24 KB	32 KB	48 KB	48 KB	64 KB	64 KB
Data memory	640 Bytes	1408 Bytes	2048 Bytes	3840 Bytes	3328 Bytes	3840 Bytes	3328 Bytes
Data EEPROM	256 Bytes	256 Bytes	1024 Bytes	1024 Bytes	1024 Bytes	1024 Bytes	1024 Bytes
External program memory	No	No	No	No	No	No	No
Interrupt sources	16	16	17	17	29	17	29
I/O ports	A..E	A..E	A..G	A..G	A..G	A..G	A..G
Timers	3	3	5	5	4	5	4
Capture/Compare/PWM modules	2 PWM	2 PWM	5	2	1	2	1
Enhanced Capture/Compare/PWM modules	0	0	0	3	1	3	1
Serial Communication	USART, MSSP	USART, MSSP	MSSP, USART (2)	MSSP, EUSART (2)	MSSP, EAUSART, ECAN	MSSP, EUSART (2)	MSSP, EUSART, ECAN
Parallel Communication	PSP	PSP	PSP	PSP	PSP	PSP	PSP
10-bit A/D module	8 channels	8 channels	12 channels	12 channels	12 channels	12 channels	12 channels
Low voltage detect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Brown out reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Instruction set	77	77	77	77	77	77	77
Packages	40-pin DIP 44-pin TQFP 44-pin QFN	40-pin DIP 44-pin TQFP 44-pin QFN	64-pin TQFP	64-pin TQFP	64-pin TQFP 68-pin PLCC	64-pin TQFP	64-pin TQFP

Feature	PIC18F8585	PIC18F8621	PIC18F8680	PIC18F4220	PIC18F4320
Operating frequency	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz	DC-40 MHz
Program memory	48 KB	64 KB	64 KB	4 KB	8 KB
Data memory	3328 Bytes	3840 Bytes	3328 Bytes	512 Bytes	512 Bytes
Data EEPROM	1024 Bytes	1024 Bytes	1024 Bytes	256 Bytes	256 Bytes
External program memory	Yes	Yes	Yes	No	No
Interrupt sources	16	16	17	17	20
I/O ports	A..H, J	A..H, J	A..H, J	A..E	A..E
Timers	4	5	4	4	4
Capture/Compare/PWM modules	1	2	1	1	1
Enhanced Capture/Compare/PWM modules	1	3	1	1	1
Serial Communication	EAUSART MSSP ECAN	AUSART (2) MSSP	MSSP, USART ECAN	MSSP, USART	MSSP, USART
Parallel Communication	PSP	PSP	PSP	PSP	PSP
10-bit A/D module	16 channels	16 channels	16 channels	13 channels	13 channels
Low voltage detect	Yes	Yes	Yes	Yes	Yes
Brown out reset	Yes	Yes	Yes	Yes	Yes
Instruction set	77	77	77	77	77
Packages	80-pin TQFP	80-pin TQFP	80-pin TQFP	40-pin DIP 44-pin TQFP 44-pin QFN	40-pin DIP 44-pin TQFP 44-pin QFN

Note.
Both PIC18F8585 and PIC18F8680 can only work with 25 MHz crystal when external memory is enabled.

Table 1.1 ■ Features of the PIC18 Microcontrollers (concluded)

The block diagram of the PIC18 member PIC18F8720 is shown in Figure 1.1. This Figure shows the peripheral functions implemented in the PIC18F8720. These are also discussed later in more details.

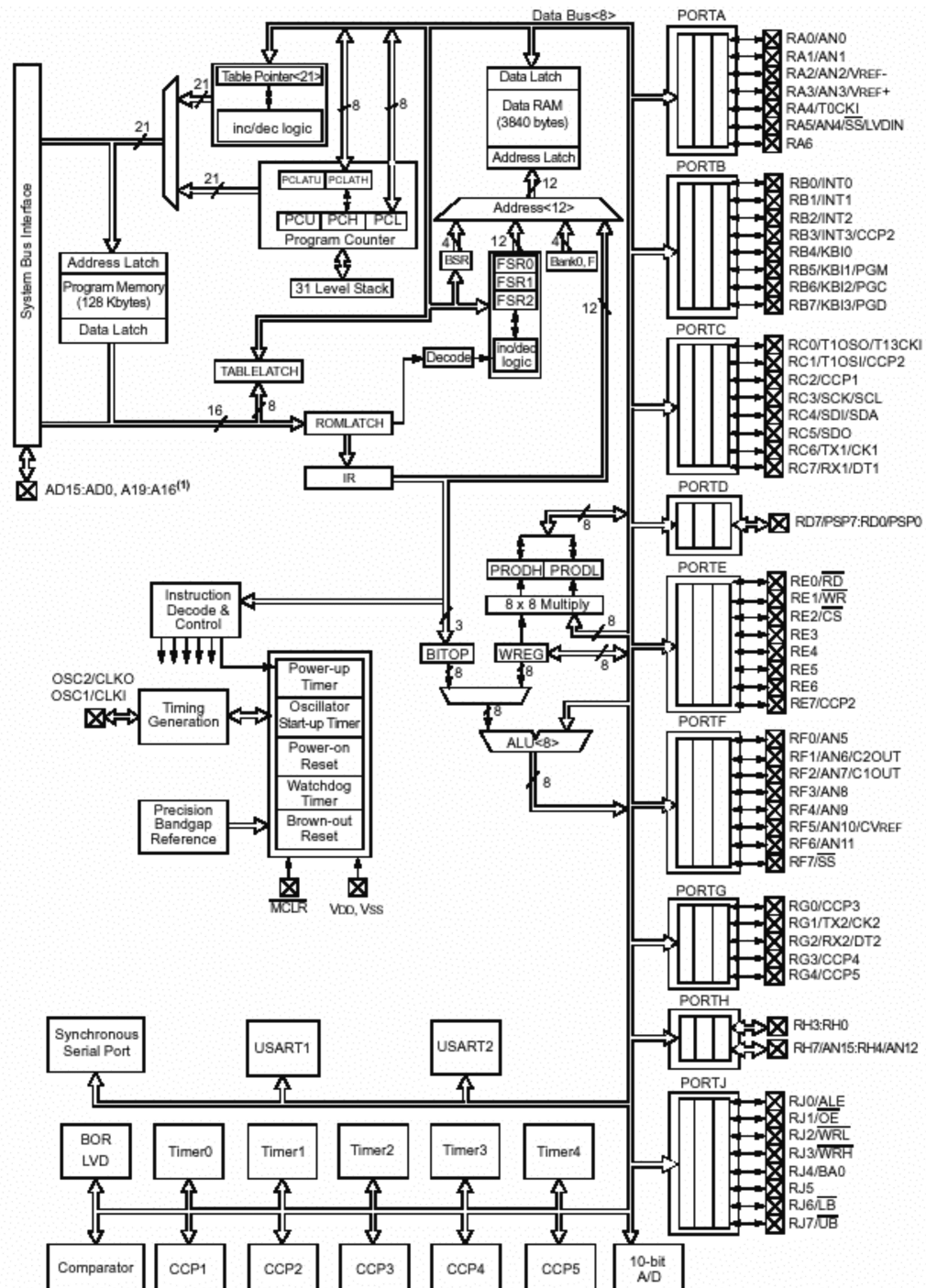


Figure 1.1 ■ Block diagram of the PIC18F8720 (reprint with permission of Microchip)

1.5 The PIC18 Memory Organization

Memory consists of a sequence of directly addressable “locations.” A memory location is referred to as an *information unit*. A memory location in the PIC18 holds eight bits of information. Eight bits of information are called a *byte*. Sometimes one must deal with four bits of information at a time. Four bits of information are called a *nibble*. A memory location can be used to store data, instruction, the status of peripheral devices, and so on. An information unit has two components: its *address* and its *contents*, shown in Figure 1.2.

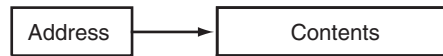


Figure 1.2 ■ The components of a memory location

Each location in memory has an address that must be supplied before its contents can be accessed. The CPU communicates with memory by first identifying the address of the location and then passing this address on the address bus. This is similar to the fact that a mail carrier needs an address in order to deliver a letter. The data is transferred between memory and the CPU along the data bus.

To differentiate the contents of a register or memory location from the address of a register or the memory location, the following notations are used throughout this text:

- *[register address]*: Refers to the contents of the register. For example, [WREG] refers to the contents of the WREG register; [0x20] refers to the contents of the general-purpose register at address 0x20. The prefix 0x indicates that the number is represented in hexadecimal format. A number without a prefix is decimal.
- *address*: Refers to the register or memory location. For example, 0x10 refers to special function register at address 0x10.

1.5.1 Separation of Data Memory and Program Memory

As shown in Figure 1.3, the PIC18 MCU assigns data and program to different memory spaces and provides separate buses to them so that both are available for access at the same time.

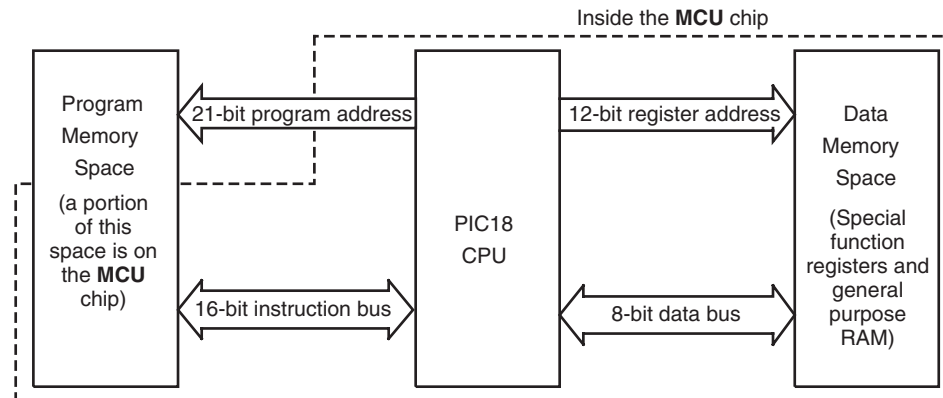


Figure 1.3 ■ The PIC18 memory spaces

The PIC18 MCU has a 21-bit program counter that is divided into three registers: PCU, PCH, and PCL. Among them, only the PCL register is directly accessible to the user. Both the PCH and the PCL are eight bits, whereas the PCU is five bits.

In the following discussion, memory addresses are referred to by using hex or decimal numbers. A hex number is indicated by adding a suffix *H* (or *h*) or prefix *0x* to the number. For example, 10H or 10h indicates hex 10 or decimal 16, and 0x20 represents the hex 20 or decimal 32. The MPASM also allows us to use *H'xx'* to specify a hex number *xx* and to use *D'yy'* to specify a decimal number *yy*. *Hex* refers to *hexadecimal* in the rest of this text.

Many digital systems have a large amount of memory. Therefore, special terms are often used to refer to the size of memory system. Among them, KB, MB, and GB are most often used:

- 1 KB refers to 2^{10} (1,024) bytes of memory.
- 1 MB refers to 2^{20} (1,048,576) bytes of memory.
- 1 GB refers to 2^{30} (1,073,741,824) bytes of memory.

1.5.2 PIC18 Data Memory

The PIC18 data memory is implemented as SRAM. Each location in the data memory is also referred to as a *register* or *file register*. The PIC18 MCU supports 4096 bytes of data memory. It requires 12 bits of address to select one of the data registers. The data memory map of the PIC18 MCU is shown in Figure 1.4.

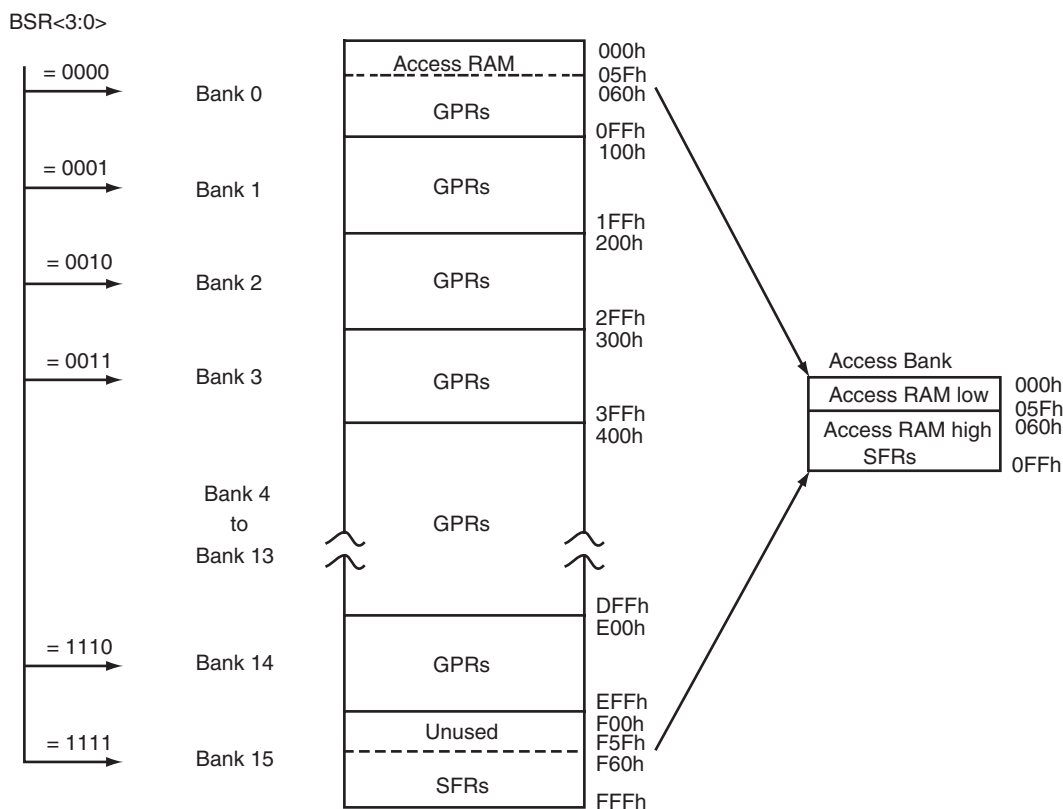


Figure 1.4 ■ Data memory map for PIC18 devices (redraw with permission of Microchip)

Because of the limited length of the PIC18 instruction (most instructions are 16 bits), only eight bits of the PIC18 instruction are used to specify the file register to be operated on. As a result, the PIC18 designers divided the 4096 file registers into 16 banks. Only one bank of 256 file registers is active at any time. An additional four bits are placed in a special register called bank select register (BSR) to select the bank to be active. The user needs to change the contents of the BSR register in order to change the active bank.

There are two types of registers: general-purpose registers (GPRs) and special-function registers (SFRs). GPRs are used to hold dynamic data when the PIC18 CPU is executing a program. SFRs are registers used by the CPU and peripheral modules for controlling the desired operation of the MCU. These registers are implemented as SRAM. A summary of these SFRs is listed in Appendix A.

The SFRs are assigned from the highest addresses and downward, whereas GPRs start from address 0 and upward. Depending on the device, some of the GPRs in the middle are not implemented. For example, the PIC18F452 has 1536 (six banks) bytes of data memory, and then banks 0 to 5 and bank 15 are implemented. The first 96 bytes (in bank 0, 0x000-0x05F) of the GPRs and the last 160 bytes (in bank 15, 0xF60-0xFFFF) of the SFRs are grouped into a special bank called *access bank*. The functioning of the access bank is explained in Section 1.8. For the PIC18F242/252/442/452 MCUs, the access bank comprises of the upper 128 bytes in bank 15 and the lower 128 bytes in bank 0.

1.5.3 EEPROM Data Memory

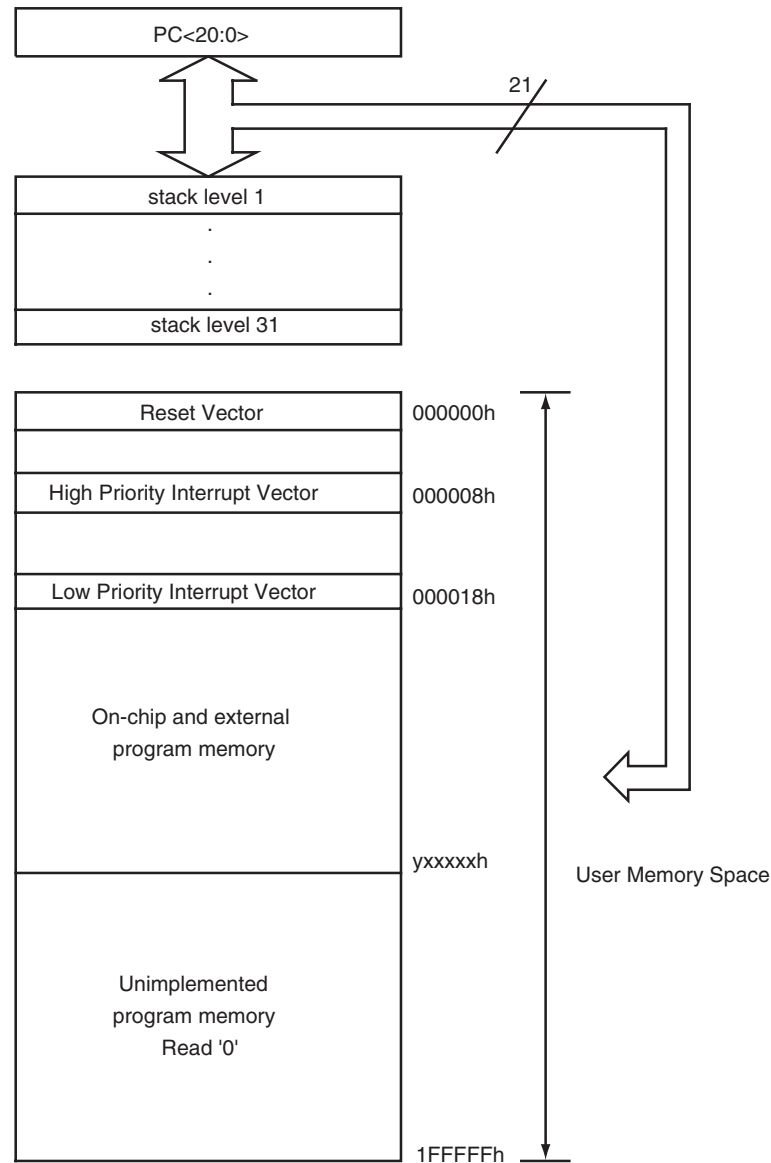
At the time of this writing, all the PIC18 devices that have on-chip flash program memory also have either 256 bytes or 1024 bytes of data EEPROM. The data EEPROM is readable and writable during normal operation over the entire power supply range. The data EEPROM memory is not directly mapped in the register file space. Instead, it is indirectly addressed through the special function register. The operation of the data EEPROM is discussed in Chapter 14.

1.5.4 Program Memory Organization

Each PIC18 member has a 21-bit program counter and hence is capable of addressing the 2-MB program memory space. Accessing a nonexistent memory location will cause a read of all 0s.

Different members of the PIC18 family have different memory configurations. The PIC18CXX2 and PIC18CXX8 devices have on-chip EPROM program memory only and cannot access external memory. The PIC18C601 and PIC18C801 do not have on-chip memory. The PIC18C601 is capable of accessing 256 KB of external program memory, whereas the PIC18C801 can access 2 MB of external program memory. The PIC18FXX2, PIC18FXX8, and the PIC18F6620/6720 devices have on-chip flash program memory only. The PIC18F8585/8680/8621/8620/8720 can also access external program memory in addition to their on-chip flash program memory.

The PIC18 MCU has a 31-entry return address stack that is used to hold return addresses for subroutine call (to be discussed in Chapter 4) and interrupt processing (to be discussed in Chapter 6). This return address stack is not part of the program memory space. The program memory map is illustrated in Figure 1.5.



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

Figure 1.5 ■ PIC18 memory organization (redraw with permission of Microchip)

As shown in Figure 1.5, the address **000000h** is assigned to the *reset vector*, which is the program-starting address after power-on or manual reset. The address **000008h** is the starting address of the high-priority interrupt service routine. Sixteen bytes are allocated to the high-priority interrupt service routine by default. The address **000018h** is the starting address for the low-priority interrupt service routine, and there is no default size for this service routine. The user program should follow the low-priority interrupt service routine. Reset is also discussed in Chapter 6.

1.6 The PIC18 CPU Registers

The PIC18 MCU has a group of registers, from 0xFD8 to 0xFFF (listed in Table 1.2), in the data memory space that are dedicated to the general control of the CPU operation. This group of registers can be referred to as *CPU registers*. Each of these CPU registers is discussed in an appropriate chapter of this book.

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	INDFO ⁽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	FSROH	File select register 0 high byte
0xFE9	FSROL	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	POSTDEC2 ⁽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 register
Note. This is not a physical register		

Table 1.2 ■ PIC18 CPU registers

The STATUS register, shown in Figure 1.6, contains the arithmetic status of the ALU. As with any other register, the STATUS register can be the destination of any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC, C, OV, or N bits, then the write to these five bits is disabled. These bits are set or cleared according to the device logic. Therefore, the result of an instruction with the STATUS register as the destination may be different than intended. It is recommended, therefore, that only BCF, BSF, SWAPF, MOVFF, and MOVWF instructions be used to alter the STATUS register because these instructions do not affect the Z, C, DC, OV, or N bits of the STATUS register.

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.

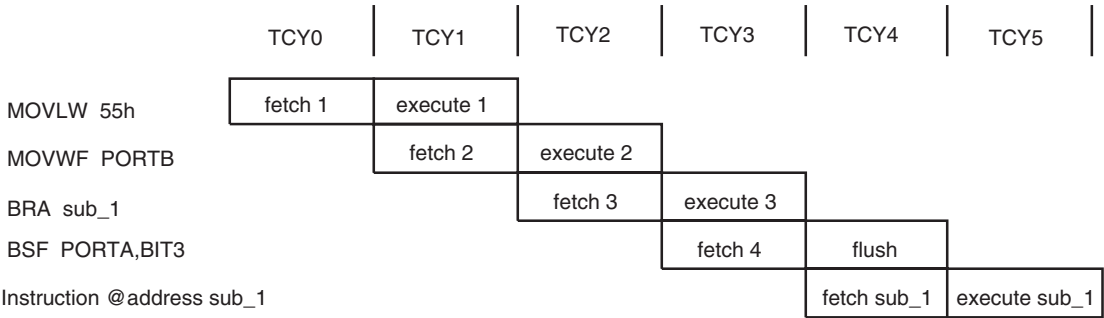
For borrow, the polarity is reversed. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Figure 1.6 ■ The STATUS register (0xF08) (redraw with permission of Microchip)

The WREG register (referred to as *working register*) is a special register that is involved in the execution of many instructions and can be the destination of many instructions.

1.7 The PIC18 Pipelining

The PIC18 designer divided the execution of most of the PIC18 instructions into two stages (*instruction fetch* and *instruction execution*) and then overlapped the execution of two consecutive instructions. Each stage takes one instruction clock cycle to complete. The result of the overlap of instruction execution is that most instructions take one instruction clock cycle to complete. This scheme is called *instruction pipelining*. An example of instruction pipelining is illustrated in Figure 1.7.



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

Figure 1.7 ■ An example of instruction pipeline flow

There are two problems caused by instruction pipelining: *data dependency hazard* and *control hazard*. In a program, it is common for one instruction to perform further operation on the result produced by the previous instruction. If the pipeline is designed in a way that the earlier instruction cannot write the result back to the register or memory location before it is used by the following instruction(s), then the data-dependency hazard has occurred. Most of the data-dependency hazards can be solved by *result forwarding*. However, if an instruction reads from a memory location (e.g., a load instruction) whereas the following instruction will use the returned value to perform certain operation, then result forwarding cannot resolve the hazard. This problem is usually solved by rearranging the instruction sequence to avoid this type of data dependency or inserting a no-op instruction. The dependency hazard problem will occur on pipelined processors with more than two stages. The PIC18 instruction pipeline has only two stages and does not have data-dependency hazard problems.

Control hazard is caused by branch instructions. Whenever a branch instruction reaches the execution stage and the branch is taken, then the following instructions in the pipeline need to be flushed because they are not allowed to take any effect by the program logic. In Figure 1.7, the instruction **BSF PORTA, BIT3** is flushed when it reaches the execution stage for this reason. There are several options to deal with control hazards in a pipelined processor. However, this issue is beyond the scope of this text.

The PIC18 MCU needs to access program memory during the instruction *fetch* stage and needs to access data memory during the instruction *execute* stage. When pipelining the execution of instructions, the PIC18 MCU needs to access the program memory and the data memory in the same clock cycle. This requirement is satisfied by separating the program memory from the data memory and providing separate buses to them.

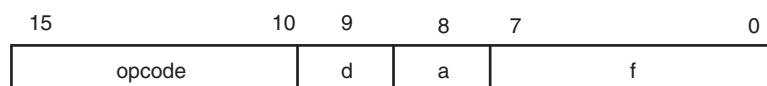
The pipelined processor is explained clearly in Patterson and Hennessy's book *Computer Organization* published by Morgan Kaufman.

1.8 PIC18 Instruction Format

It was mentioned in Section 1.5.2 that data memory is divided into banks. Why would the banking scheme be used to control the access of data memory? The instruction format must be defined in order to understand this issue.

The instruction set is grouped into five basic categories:

1. *Byte-oriented operations*. The format of byte-oriented instructions is shown in Figure 1.8. The 6-bit field *opcode* specifies the operation to be performed by the ALU.



d = 0 for result destination to be WREG register.

d = 1 for result destination to be file register (f)

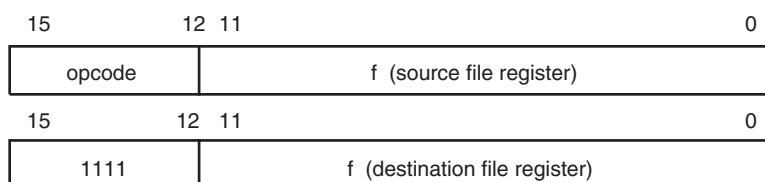
a = 0 to force Access Bank

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)

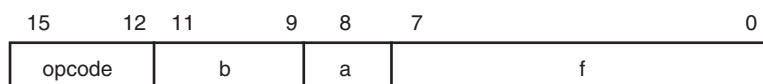
2. *Byte-to-byte operations (two-word)*. The format of the instruction in this category is shown in Figure 1.9. There is only one instruction that uses this format: **movff f1, f2**. This instruction allows one to move data from one file register to another.



f = 12-bit file register address

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

3. *Bit-oriented file register operations*. The format of instructions in this category is shown in Figure 1.10. This format uses an 8-bit field (f) to specify a file register as the operand.



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)

4. *Literal operations.* Instructions in this category specify a *literal* (a number) as an operand. The format of instructions in this category is shown in Figure 1.11.

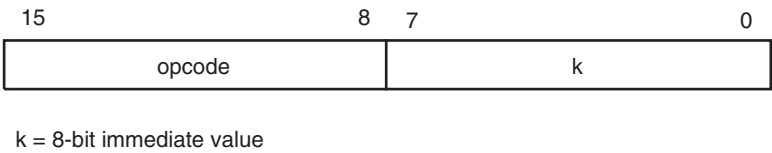


Figure 1.11 ■ Literal operations (redraw with permission of Microchip)

5. *Control operations.* The format of instructions in this category is shown in Figure 1.12. The notation n<7:0> stands for the bit 7 to bit 0 of the number n, whereas the notation n<19:8> stands for the bit 19 to bit 8 of the number n. The notation n<10:0> means that the number n is an 11-bit number. There are four different variations in their formats.

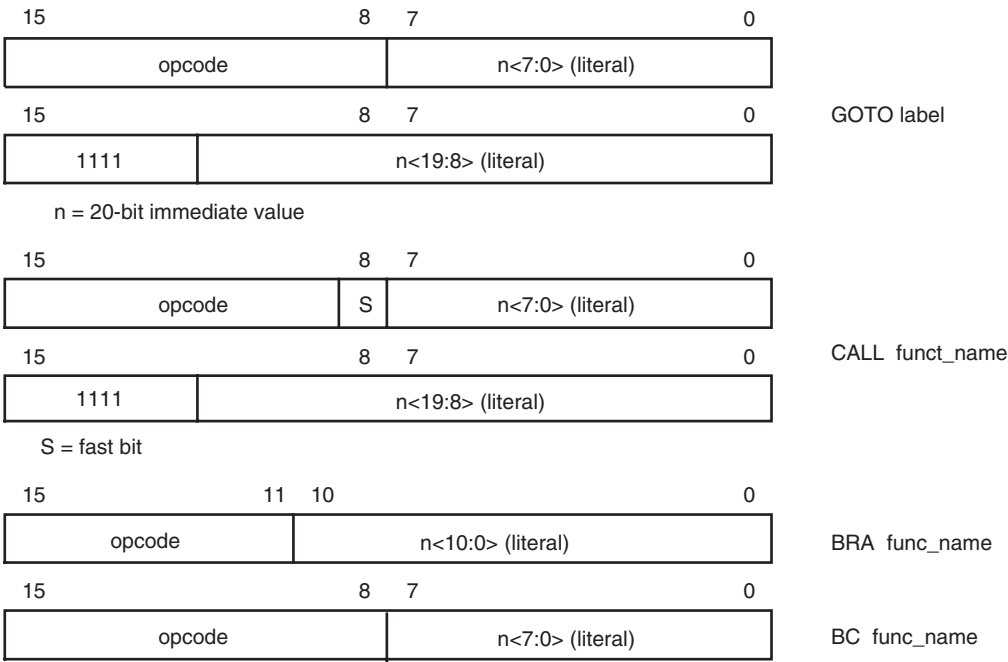


Figure 1.12 ■ Control operations (redraw with permission of Microchip)

As shown in Figures 1.8 to 1.12, all the PIC18 instructions use eight bits to specify the data register operand. A data register (excluding WREG) is also called a *file register*. Only 256 different registers can be specified by eight bits. However, all the PIC18 devices have more than 256 file registers, and hence additional information is needed to pinpoint the exact register to be operated on. This additional information is stored in the BSR register. The designer of the PIC18 MCU divided data memory into 16 (maximum) banks, with each bank having 256 data registers. The BSR register specifies the bank, and the f field in the instruction specifies the register number within the bank.

The banking scheme has been used in the PIC12, the PIC14000, the PIC16, and the PIC17 MCUs. This scheme allows a PIC MCU to incorporate more than 256 data registers on the CPU. However, it also adds a significant amount of overhead to the software because of the need to switch from one bank to another. In addition, it is easy to forget about bank switching, which will cause the software to fail.

In order to solve the problem caused by the banking scheme, the designers of the PIC18 MCU incorporated the *access bank*. The access bank consists of the lowest 96 GPRs and the highest 160 SFRs. As long as an instruction specifies a data register in the access bank, banking is ignored, and bank switching is unnecessary. All SFRs except a subset in the CAN module are in the access bank (CAN stands for *controller area network*). This makes bank switching unnecessary in many cases.

When bank switching is needed, the **movlb k** instruction can be used. This instruction places the value of k in the lower four bits of the BSR register. The result of the execution of this instruction is that it caused the data registers in bank k to become active.

In Figures 1.8 and 1.10, the a field in the PIC18 instruction allows the user to select the access bank. When writing program in assembly language, the assembler (MPASM) allows the user to use the letter A (a = 0) to specify the access bank. When the access bank is not chosen, one should use the word BANKED (a = 1) to allow the BSR register to do the bank selection.

The d field in Figure 1.8 allows the user to choose either the WREG or the file register as the destination of the instruction. The assembler allows the user to use the letter F (d = 1) to specify a file register and use the letter W (d = 0) to specify the WREG register as the destination. For example,

```
addwf    sum, F, A    ; sum is a GPR
```

adds the WREG register and sum in the access bank and places the result in sum.

```
addwf    sum,W, A
```

performs the same operation but leaves the result in the WREG register.

1.9 Addressing Modes

All MCUs use addressing modes to specify the operand to be operated on. The PIC18 MCU provides *register direct*, *immediate*, *inherent*, *indirect*, and *bit-direct* addressing modes for specifying instruction operands. As discussed in Chapter 2, assembler directives allow the user to use symbols to refer to memory locations. Using symbols to refer to memory locations makes the user program more readable. During the following discussion, symbols are used to refer to memory locations when appropriate.

1.9.1 Register Direct

The PIC18 device uses an 8-bit value to specify a data register as an operand. The register may be in the access bank or other banks. In the first case, the 8-bit value is used to select a register in the access bank, and the bank value in the BSR register is ignored. If the access bank is not selected, then the access is completed from the memory of the bank specified in the BSR register. The following instructions illustrate the register direct addressing mode:

```
movwf 0x1A, BANKED
```


copies the contents of the WREG register to the memory location 0x1A in the bank specified by the BSR register. The word **BANKED** (must be in uppercase) informs the assembler that the BSR register must be included in specifying the data register to be operated on.

```
movwf 0x45, A
```

copies the contents of the WREG register to the memory location 0x45 in the access bank.

```
movff reg1, reg2
```

copies the contents of the register reg1 to the register reg2. Both reg1 and reg2 are 12-bit values. The value of BSR is ignored.

1.9.2 Immediate Mode

In the immediate addressing mode, the actual operand is provided in the instruction. There is no need to access any memory location. The following instructions illustrate the immediate addressing mode:

```
addlw 0x20
```

adds the hex value 20 to the WREG register and places the sum in the WREG register.

```
movlw 0x15
```

loads the hex value 15 into the WREG register.

```
movlb 3
```

places the decimal value 3 in the lower four bits of the BSR register. The lower four bits become 0011. This instruction makes bank 3 the active bank. The value to be operated on directly is often called *literal*.

1.9.3 Inherent Mode

In the inherent mode, the operand is implied in the opcode field. The instruction opcode does not provide the address of the implied operand. The following instructions illustrate the inherent mode:

```
movlw 0x20
```

places the hex value 20 (decimal 32) in the WREG register. In this example, the value 0x20 is specified in the instruction machine code. The destination WREG is implied in the opcode field. No other address information for the WREG register is supplied.

```
andlw 0x13
```

performs an AND operation on the corresponding bits of the hex number 13 and the WREG register (i.e., bit i of WREG and with bit i of the value 0x13; i = 0 . . . 7). In this example, only the immediate value 0x13 is specified in the instruction machine code. The address of the WREG register 0xFE8 is not specified.

1.9.4 Indirect Mode

In this mode, a special function register is used as a pointer to the data memory location that is to be read and written. Since this register is in SRAM, the contents can be modified by the program. This can be useful for data tables in data memory and for *software stacks*. The software stack will be explained in Chapter 4.

There are three indirect addressing registers: FSR0, FSR1, and FSR2. To address the entire data memory space (4096 bytes), 12 bits are required. To store the 12-bit address information, two 8-bit registers are used. These indirect addressing registers are the following:

1. FSR0: composed of FSR0H and FSR0L
2. FSR1: composed of FSR1H and FSR1L
3. FSR2: composed of FSR2H and FSR2L

After placing the address of the data in one of the FSR registers, one needs to read from or write into one of the three registers that are not physically implemented in order to activate indirect addressing. These three registers are INDF0, INDF1, and INDF2.

If an instruction writes a value to INDF0, the value will be written to the data register with the address indicated by the register pair FSR0H:FSR0L. A read from INDF1 reads the data from the data register with the address indicated by the register pair FSR1H:FSR1L. INDFn can be used in a program anywhere an operand can be used. The process of indirect addressing is illustrated in Figure 1.13.

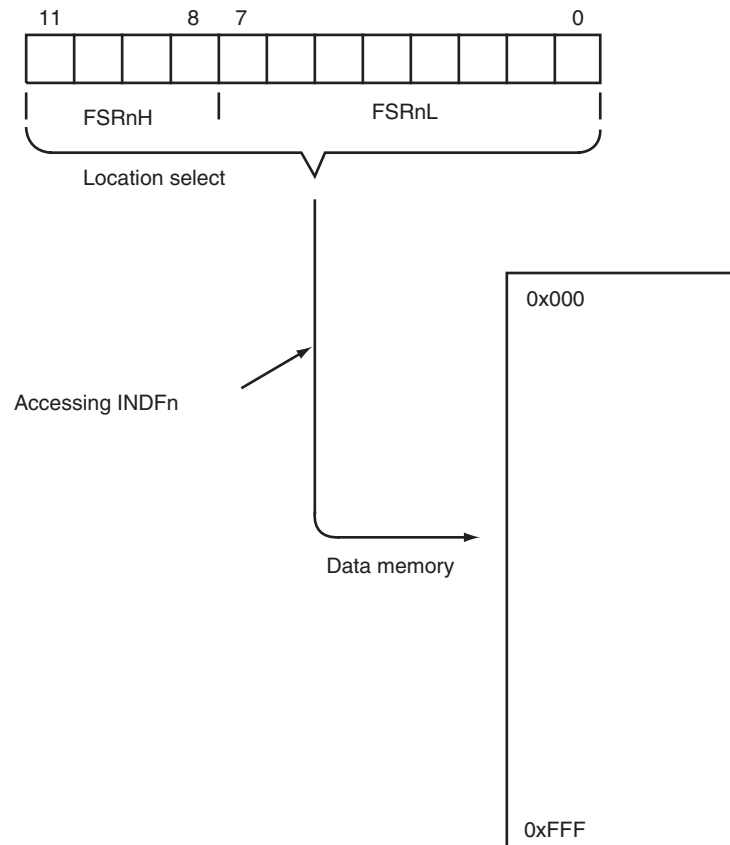


Figure 1.13 ■ Indirect addressing

Each FSR register has an INDF register associated with it plus four additional register addresses. Performing an operation on one of these five registers determines how the FSR will be modified during indirect addressing:

1. Do nothing to FSRn after an indirect access. This access is specified by using the register INDFn (n = 0 . . . 2).
2. Auto-decrement FSRn after an indirect access (postdecrement). This access is specified by using the register POSTDECn (n = 0 . . . 2).
3. Auto-increment FSRn after an indirect access (postincrement). This access is specified by using the register POSTINCn (n = 0 . . . 2).
4. Auto-increment FSRn before an indirect access (preincrement). This access is specified by using the register PREINCn (n = 0 . . . 2).
5. Use the value in the WREG register as an offset to FSRn. The signed value in WREG is added to the value in FSR to form an address before performing an indirect access. Neither the WREG nor the FSRn is modified after the access. This access is specified by using the register PLUSWn.

The following examples illustrate the usage of indirect addressing modes:

```
movwf INDF0
```

copies the contents of the WREG register to the data memory location specified by the FSR0 register. After the execution of this instruction, the contents of the FSR0 register are not changed.

```
movwf POSTDEC0
```

copies the contents of the WREG register to the data memory location specified by the FSR0 register. The contents of FSR0 are decremented by 1 after the operation.

```
movwf PREINC0
```

first increments the FSR0 register by 1 and then copies the contents of the WREG register to the data memory location specified by the FSR0 register.

```
clrf PLUSW0
```

clears the memory location at the address equal to the sum of the value in the WREG register and that in the FSR0 register.

In the previous examples, one does not need to specify whether the register is in the access bank because the complete 12-bit data register address is taken from one of the FSR registers.

1.9.5 Bit-Direct Addressing Mode

The PIC18 MCU has five instructions to deal with an individual bit. These instructions use three bits to specify the bit to be operated on. For example,

```
BCF PORTB,3,A ; integer 3 specifies the bit to be cleared
```

clears bit 3 of the data register PORTB, which will then pull the port B pin RB3 to low.

```
BSF PORTA,4,A ; integer 4 specifies the bit to be set
```

sets bit 4 of the data register PORTA, which will then pull the port A pin RA4 to high.

1.10 A Sample of PIC18 Instructions

The PIC18 has 77 instructions. Four of these are 32-bit instructions, whereas the others are all 16 bits. A subset of the PIC18 instructions is examined in this section.

1.10.1 Data Movement Instructions

Memory data must be placed in appropriate registers before useful operations can be performed. Data movement instructions are provided for this purpose. A subset of the data movement instructions is listed in Table 1.3.

Mnemonic	Description	16-bit instruction word	Status affected
lfsr f, k	load FSR	1110 1110 00ff k ₁₁ kkk	None
		1111 0000 k ₇ kkk kkkk	
movf f, d, a	Move f	0101 00da ffff ffff	Z, N
movff fs, fd	Move fs (source) to f	1100 ffff ffff ffff	None
		1111 ffff ffff ffff	
movwf f,a	Move WREG to f	0110 111a ffff ffff	None
swapf f, d, a	Swapp nibbles in f	0011 10da ffff ffff	None
movlb k	Move literal to BSR<3:0>	0000 0001 kkkk kkkk	None
movlw k	Move literal to WREG	0000 1110 kkkk kkkk	None
Note. Both LFSR f, k and MVFF fs, fd are 32-bit instructions			

Table 1.3 ■ A sample of PIC18 data movement instructions

The instruction **lfsr f, k**, a 32-bit instruction, allows the user to place a 12-bit value in the FSR register specified by the value f. Two bits are provided for selecting the FSR registers (FSR0–FSR2). A 12-bit value (k) is contained in the instruction. The upper four bits (represented as k₁₁kkk in Table 1.3) of k are contained in the first word of the instruction, whereas the lower eight bits (represented as k₇kkk kkkk in Table 1.3) are contained in the second word.

The instruction **movf f, d, a** in Table 1.3 is provided for easy migration from the PIC16 family to the PIC18 family because the PIC16 family also has the same instruction. By setting the d field to 0 (represented by the letter W), this instruction will copy the contents of a file register to the WREG register. For example, the instruction

```
movf 0x20,W,A
```

will copy the contents of the data register at 0x20 to the WREG register.

The **movff** instruction, a 32-bit instruction, can copy a file register in one bank to a file register in another bank without referring to the BSR register. Both the source and the destination registers are specified in 12 bits.

The **movlb k** instruction sets the bank k as the active bank. The **movlw k** instruction places the value k in the WREG register.

Example 1.1

Write a PIC18 instruction (or instruction sequence) to transfer data from (a) WREG to data register at 0x30, (b) the data register at 0x30 to the data register at 0x40, (c) the data register at 0x40 to WREG, and (d) load the value 0x200 into FSR0.

Solution:

- (a) `movwf 0x30,A` ; force access bank
- (b) `movff 0x30, 0x40` ;
- (c) `movf 0x40,W,A` ; force access bank and copy register 0x40 to WREG
- (d) `lfsr FSR0, 0x200` ; load the value 0x200 into FSR0

1.10.2 ADD Instructions

ADD is the generic name of a group of instructions that perform the addition operation. The ADD instruction may have two or three operands. A three-operand ADD instruction includes the carry flag in the STATUS register as one of the operand. The PIC18 MCU has 3 ADD instructions:

- `addwf f, d, a` ; add WREG and f
- `addwfc f, d, a` ; add WREG, carry bit, and f
- `addlw k` ; add literal k to WREG

Example 1.2

Write an instruction to perform the following operations:

- (a) Add the content of WREG and that of the data register at 0x40 (in access bank) and leave the sum in WREG.
- (b) Increment the WREG register by 5.
- (c) Add the WREG register, the register with the name of sum, and carry and leave the result in sum. The variable sum is in access bank.

Solution:

- (a) `addwf 0x40,W,A`
- (b) `addlw 5`
- (c) `addwfc sum, F, A`

Example 1.3

Write an instruction sequence to increment the contents of three registers 0x30–0x32 by 3.

Solution: The procedure for incrementing the value of a register by 3 is as follows:

Step 1

Place the value 3 in the WREG register.

Step 2

Execute the **addwf f, d, a** instruction.

The following instruction will increment the specified three registers by 3:

- `movlw 0x3`
- `addwf 0x30, F, A` ; increment the register at 0x30 by 3
- `addwf 0x31, F, A` ; increment the register at 0x31 by 3
- `addwf 0x32, F, A` ; increment the register at 0x32 by 3

Example 1.4

Write an instruction sequence to add the contents of three data registers located at 0x40–0x42 and store the sum at 0x50.

Solution: The required operation can be achieved by the following procedure:

Step 1

Load the contents of the register at 0x40 into the WREG register.

Step 2

Add the contents of the register at 0x41 into the WREG register.

Step 3

Add the contents of the register at 0x42 into the WREG register.

Step 4

Store the contents of the WREG register in the register at 0x50.

The following instructions will perform the desired operation:

```
movf 0x40, W, A      ; WREG ← [0x40]
addwf 0x41, W, A      ; add the contents of the register at 0x41 to WREG
addwf 0x42, W, A      ; add the contents of the register at 0x42 to WREG
movwf 0x50, A         ; 0x50 ← [WREG]
```

Example 1.5

Write an instruction sequence to add 10 to the data registers at 0x300–0x303 using the indirect postincrement addressing mode.

Solution: The procedure for solving this problem is as follows:

Step 1

Load the value 0x300 into the FSR0 register.

Step 2

Load the value 10 into the WREG register.

Step 3

Add the value in WREG to the data register pointed by FSR0 using the indirect postincrement mode.

Step 4

Repeat Step 3 three more times.

The instruction sequence that carries the operations from Step 1 to Step 3 is as follows:

```
movlw    0x0A
lfsr     FSR0,0x300      ; place 0x300 in FSR0
addwf    POSTINC0,F
addwf    POSTINC0,F
addwf    POSTINC0,F
addwf    POSTINC0,F
```

1.10.3 SUB Instructions

SUB is the generic name of a group of instructions that perform the subtraction operation. The SUB instruction may have two or three operands. A three-operand SUB instruction includes the carry flag in the STATUS register as one of the operands. The PIC18 MCU provides four SUB instructions:

subfwb	f, d, a	; subtract f from WREG with borrow
subwf	f, d, a	; subtract WREG from f
subwfb	f, d, a	; subtract WREG from f with borrow
sublw	k	; subtract WREG from literal

Example 1.6

Write an instruction sequence to subtract 9 from the data registers located at 0x50–0x53.

Solution: This operation can be implemented by placing 9 in the WREG register and then executing the **subwf f,F,A** instruction. The following instruction sequence will implement the required operation:

movlw 0x09	; place 9 in the WREG register
subwf 0x50,F,A	; decrement the contents of the register at 0x50 by 9
subwf 0x51,F,A	; decrement the contents of the register at 0x51 by 9
subwf 0x52,F,A	; decrement the contents of the register at 0x52 by 9
subwf 0x53,F,A	; decrement the contents of the register at 0x53 by 9

Example 1.7

Write an instruction to perform each of the following operations:

Subtract the WREG register from the file register at 0x30 and leave the difference in the file register at 0x30.

Subtract the file register at 0x30 and borrow from the WREG register.

Subtract the WREG register from the file register at 0x50 and leave the difference in WREG.

Solution: The following instructions will perform the specified operations:

- (a) subwf 0x30, F, A
- (b) subfwb 0x30, W, A
- (c) subwf 0x50, W, A

1.11 Overview of the 8-Bit MCU Market

There are many different 8-bit MCUs in the market today. Some of them follow the RISC design approach, whereas others follow the CISC design approach. This section briefly reviews the history of RISC and CISC and then provides a brief overview of the 8-bit MCU market.

1.11.1 CISC versus RISC

In the past, processor design followed either the Complex Instruction Set Computer (CISC) approach or the Reduced Instruction Set Computer (RISC) approach. The instructions of a processor designed with the CISC philosophy tend to be more complicated and perform more functions. The processor designers in the CISC camp believed that the instruction set designed with this philosophy in mind could make the machine code shorter and better match the syntax of high-level languages and hence could support high-level languages better. Experience proved that the resultant machine code was indeed shorter. However, many of the complex instructions are rarely used by the compiler, and the performance of the resultant machine was poor. In addition, the CISC computer took a longer time to design, which would likely cause the market window to be missed.

On the contrary, instructions of a RISC processor perform only simple operations. The addressing modes supported by a RISC computer tend to be simple as well. The followers of the RISC philosophy believed that a simple instruction set and simple addressing modes could simplify the design of the processor and that the resultant processor could run at a faster clock rate. More important, the RISC computer takes a shorter time to design and can be put on the market sooner. The earliest RISC processors have the following common features:

- Pipelined instruction execution.
- Fixed-length instructions (all instructions are either 32 or 16 bits long).
- Simple instruction set, simple instruction formats, and simple addressing modes.
- Large number of registers to support register-to-register operations (such operations are faster than register-to-memory or memory-to-memory operations).
- Delayed branch instruction. The earliest RISC processor allowed the instruction that followed the branch instruction to take effect regardless of whether the branch was taken. The purpose of this technique is to reduce the performance reduction caused by the branch instruction. This technique is no longer supported in the current generation RISC processors.
- Separation of the data memory and the program memory to allow simultaneous access of two memories. The computer architecture with separate data memory and program memory is often called *Harvard architecture*.

The compiled program for the RISC processor tends to be longer than its counterpart for the CISC processor. However, the resultant program for a RISC processor can run faster than its counterpart in a CISC processor. The current practice of processor design is to combine the strengths of both the RISC and the CISC processor.

Computer makers like to benchmark the performance of one another's machines. The adopted metric for comparing the performance of different machines is the running times of the same set of application programs on different machines. Because of the differences in design, machine A may run certain applications faster than machine B but run other applications slower than machine B. The selection of the set of application programs for comparison is arguable. Other metrics were also used to compare the performance of different machines. A common but inaccurate metric is MIPS (million instructions per second). It is obvious that MIPS means different things for processors with different instruction sets. The drawbacks of this metric have been detailed in Patterson and Hennessy's books. For computers made of the same microprocessor, the MIPS is a good metric for performance comparison if the same compiler is used.

1.11.2 Major 8-Bit MCUs

The 8-bit MCU market has experienced significant growth in the past decade. More than a billion 8-bit MCUs have sold annually in the past few years. Microchip started to fabricate 8-bit MCUs in late 1980s and by 1999 had sold its first billion MCUs. Thirty months later (in May 2002), Microchip had sold another billion MCUs. This statistic shows the strength of the 8-bit MCU market. In July 2003, Microchip became the number one vendor of the 8-bit MCU in terms of units of shipment. The success of the Microchip 8-bit MCUs can be attributed to the following factors:

- *Cost effectiveness.* MCUs made by Microchip follow the RISC design philosophy. They provide a simple instruction set and a fixed instruction length and use the pipelining technique to achieve high instruction execution throughput. Many Microchip devices have flash memory and provide in-system programming (ISP) capability, which allows the end user to upgrade product software without removing the MCU from the end product, an attractive feature. In addition, Microchip sells its MCUs at a competitive price.
- *Rich peripheral functions.* MCUs from Microchip have implemented all the peripheral functions that can be found in MCUs from other vendors. These peripheral functions include (1) parallel I/O ports; (2) timer functions, such as input capture, output compare, pulse-width modulation (PWM), counters, real-time interrupt (RTI), and watchdog timer; (3) 8-bit to 12-bit A/D converter; (4) multiple serial interface protocols (USART, SPI, and I²C); and (5) controller area network (CAN) controller.
- *Technical support.* Microchip provides sample designs and design consultation service to help its customers. Customers can ask for technical questions via phone calls or e-mail.
- *Development tools support.* The MPLAB IDE from Microchip is undoubtedly the most important component in this area. MPLAB IDE is essentially an integrated development environment that incorporates a text editor, cross assemblers, simulators, programmer drivers, and a source-level debugger. It allows the user to perform source-level debugging in assembly or C language. Microchip makes MPLAB IDE even more attractive: it is free. Microchip also provides demo boards, programmers, in-circuit emulators, C compilers for the PIC17 MCU and the PIC18 MCU, and low-cost in-circuit debuggers to help customers develop their products. Many of these development tools are reviewed in Chapter 3.
- *University program.* Universities can request (so do other commercial users) free samples of MCUs and other development tools. Universities and students can purchase Microchip products at a discounted price. The web address for Microchip is www.microchip.com.

The PIC18 MCU can run with a 40-MHz crystal oscillator. With pipelining, most of the PIC18 instructions will take one instruction cycle (four crystal oscillator cycles) to execute. Therefore, the PIC18 MCU's performance is close to 10 MIPS.

Other major 8-bit MCUs include the following:

- The 68HC05, 68HC08, and 68HC11 from Motorola
- The 8051/8052 variants from more than 45 companies
- The AVR from Atmel

1.11.3 Motorola 8-Bit MCUs

Motorola is competing with Microchip for being the leader in terms of units of shipment of 8-bit MCUs. The 68HC05 and the 68HC08 MCUs are designed to serve low-end applications. The 68HC11 is the high-end 8-bit MCU from Motorola and was initially designed to serve the automotive market. The 68HC11 follows the CISC design philosophy. Its rich instruction set and addressing modes illustrate that. The rich instruction set and addressing modes make assembly programming easy. However, this is not an advantage for C programmers. Motorola has a good university support program. They provided free samples, low-cost demo boards, and free software (cross assembler) to help promote the 68HC11 and made it the most taught 8-bit MCU in U.S. universities. Most 68HC11 MCUs do not have on-chip flash memory. Serial interface protocols, such as UART and SPI, are standard. The 68HC11 provides a rich set of timer functions, including input capture, output compare, real-time interrupt (RTI), computer operate properly (COP) timer, and pulse accumulator. Most of the 68HC11 members have 8-bit A/D resolution only and do not implement the PWM function, which is an important feature for automotive and motor control applications. Motorola realizes these drawbacks and hence is pushing 68HC08 to take over the low-end applications of the 68HC11 and introduced the 16-bit HCS12 to take over 68HC11's high-end applications. Most of the 68HC08 members and all HCS12 members have on-chip flash memory and provide in-system programming capability. Motorola 8-bit MCUs have a combined 64-KB program and data memory.

1.11.4 Intel 8051/8052 Variants

The 8051 was designed by Intel and is the oldest 8-bit MCU. The 8051 has separate data memory and program memory spaces and follows the CISC design philosophy. Each memory space is 64 KB. The original 8051 has 4 KB of ROM, two 8-bit timers, four 8-bit I/O ports, and an asynchronous serial port. Later, Intel introduced 8052 as an enhancement to the 8051, adding another 8-bit timer to it. The initial 8051 MCU was very slow. It divides the crystal oscillator signal by 12 and uses it as the instruction clock signal to control the instruction execution. One instruction clock cycle takes 12 crystal oscillator cycles. An instruction may take from one to four instruction cycles to execute. The original 8051 MCU can run with a 12-MHz crystal oscillator. Therefore, the throughput of the original 8051 MCU is less than 1 MIPS. The 8051 provides one data pointer DPTR for accessing data memory. In many cases, this is inadequate.

Intel licensed the 8051 design to many semiconductor companies. To be competitive, most of the 8051 vendors add many enhancements to their implementations of the 8051. Many useful peripheral functions have been added by the 8051 variants. Some vendors even modified the 8051 design to shorten the instruction cycle time from 12 to 6 or 4 or even to a single oscillator clock cycle. Many vendors also add one or more data pointers to facilitate the access of data memory. On-chip flash program memory is also added by many vendors.

Philips produces the most 8051 variants, whereas Atmel produces the most flash-memory-based 8051 variants. Dallas/MAXIM Semiconductor shortens the 8051-instruction clock cycle time from 12 to 4 crystal oscillator cycles. Silicon Laboratory takes this approach to the limit and pushes the instruction clock cycle to one crystal oscillator cycle and provides devices running at 100 MHz. Silicon Laboratory 8051 variants are also pipelined. The 8051 variants from Silicon Laboratory provide (1) eight to over 100 I/O pins; (2) A/D converter with 8- to 16-bit resolutions; (3) serial communication interfaces, such as UART, SPI, I²C, and CAN controller; (4) on-chip flash program memory and in-system programming capability; (5) timer functions, such as input capture, output compare, counter, and PWM; (6) on-chip temperature sensor; and (7) free development software tools and inexpensive demo boards. The peak performance of Silicon Laboratory's 8051 variants could range from 20 to 100 MIPS.

Analog Devices Inc. focuses on the data acquisition market. Analog Devices call their 8051-based devices *MicroConverters* (ADuC). The ADuC provides A/D converters with 12- to 24-bit resolution and D/A converters with 12-bit resolution. Other peripheral functions, such as UART and SPI serial interface protocol, timers, power management, and COP, are also standard. The MSC1210 MCU from TI is another 8051 variant that follows an approach similar to that of Analog Devices. The MSC1210 has 32 KB of on-chip flash program memory, a 24-bit A/D converter, a 32-bit accumulator, three 8-bit timers, UART and SPI ports, and a watchdog timer. The MSC1210 uses four crystal oscillator cycles for one instruction cycle, as does the Dallas/MAXIM semiconductor.

Philips and Infineon are two major producers of 8051 variants. Many of their devices have flash program memory and in-system programming capability. Their 8051 variants also provide peripheral functions available in other 8051 vendors.

Because of the large number of 8051 variants and vendors, many hardware and software development tools are available from many vendors. The huge spectrum of the 8051 variants provides the user with the most choices in vendors, features, and development tools.

1.11.5 The Atmel AVR

In addition to producing 8051 variants, Atmel also produces an 8-bit RISC MCU, the AVR. The AVR has devices from 8 pin to over 64 pin. These devices are pipelined to achieve high throughput just like Microchip PIC18 MCU and Silicon Laboratory 8051 variants. The AVR provides up to 128 KB of on-chip flash program memory and optional 64-KB external memory and has in-system programming capability. Most peripheral functions available in the PIC18 are also available in the AVR. The AVR RISC MCU provides more instructions (133) than the PIC18 MCU does. The AVR has 32 general-purpose 8-bit registers and many other registers that are dedicated to the configuration and control of peripheral functions. Atmel also provides free software development tools and low-cost demo boards. Atmel's AVR MCU can run with a 16-MHz crystal oscillator and achieve the peak performance of 16 MIPS.

It is apparent that many 8-bit MCUs can be used in the same application because many devices have similar peripheral functions. The choice is often made on the basis of the following considerations:

- Device cost and system cost
- Availability of development tools and technical support
- Availability of engineering expertise on the specific MCU
- Device performance
- Personal preference

1.12 Summary

A computer system consists of hardware and software. The hardware consists of four major components: a processor (called the CPU), an input unit, an output unit, and memory. The processor can be further divided into three major parts: (1) registers, (2) the arithmetic logic unit, and (3) the control unit. All CPU activities are synchronized by the system clock. The clock frequency of the current generation of microprocessors (mainly Intel Pentium 4 and AMD Athlon-64) has exceeded 3 GHz. The CPU maintains a register called program counter (PC), which controls the memory address of the next instruction to be executed. During the execution of an instruction, the occurrence of an overflow, an addition carry, a subtraction borrow, and so forth are flagged by the system and stored in another register called the status register. The PIC18 MCU uses the STATUS register to record the occurrence of these conditions.

A microprocessor is a processor fabricated on a single integrated circuit. A microcomputer is a computer that uses a microprocessor as its CPU. Although microprocessors have been widely used since their invention, there are several limitations that led to the development of microcontrollers. First, a microprocessor requires external memory to store data and programs. Second, a microprocessor cannot interface directly to I/O devices; peripheral chips are needed. Third, glue logic is needed to interconnect external memory and peripheral interface chips to the microprocessor.

A microcontroller is a computer implemented on a VLSI chip. It contains everything contained in a microprocessor along with one or more of the following components:

- Memory
- Timer
- Pulse-width modulation (PWM) module
- Analog-to-digital converter
- Digital-to-analog converter
- Direct memory access (DMA) controller
- Parallel I/O interface
- Serial I/O interface
- Memory component interface circuitry
- Software debug circuitry

Software and data are stored in memory. Semiconductor memory chips can be classified into two major categories: random-access memory (RAM) and read-only memory (ROM).

There are many different types of ROM. MROM is a type of ROM that is programmed when it is fabricated. PROM is a type of fuse-based ROM that can be programmed in the field by the end user. EPROM is a type of ROM that is programmed electrically and erased by ultraviolet light. EEPROM is a type of ROM that can be erased and programmed electrically. The user can erase a location, a row, or the whole EEPROM chip in one operation. Flash memory can be erased and programmed electrically. However, flash memory can be erased only one row at a time or in bulk.

Programs are known as software. A program is a set of instructions that the computer hardware can execute. Programmers write a program in some kind of programming language. Only machine language was available during the early days of computers. A machine language program consists of a sequence of machine instructions. A machine instruction is a combination of 0s and 1s that inform the CPU to perform certain operation. Using machine language to write programs is difficult, and hence assembly language was developed to improve the productivity of programmers. Programs written in assembly language consist of a sequence of assembly instructions. An assembly instruction is the mnemonic representation of some machine instruction. Programs written in assembly language are still difficult to understand, and programming productivity is not high. High-level languages, such as C, C++, and Java, were invented to avoid the drawback of the assembly language. Programs written in assembly or high-level languages are called source code. Source code must be translated before it can be executed. The translator of a program written in assembly language is called an assembler, whereas the translator of a program written in a high-level language is called a compiler.

A memory location has two components: its contents and its address. When accessing a memory location, the CPU sends out the address on the address bus, and the memory components will place the requested value on the data bus.

The PIC18 MCU separates the program memory and data memory spaces and provides separate buses to access them. This Harvard architecture allows the PIC18 MCU to access the

instruction and data in the same clock cycle. The group of data registers located between 0xFD8 and 0xFFF are used to control the general operation of the CPU and should be referred to as CPU registers. The program counter is 21 bits, which allows the PIC18 MCU to access up to 2 MB of program memory.

The PIC18 MCU provides 77 instructions. Seventy-three instructions are 16 bits. Four are 32 bits. These instructions are pipelined so that most of them will take one clock cycle to complete.

A data memory location is referred to as a data register. A small set of them provides the control and records the status of peripheral functions. This group is often referred to as special-function registers (SFRs). SFRs are located in the highest data memory space. The remaining data registers are used to store data or address information and are referred to as general-purpose registers (GPRs).

Each data register is associated with a 12-bit address. However, all (except one) PIC18 instructions provide 8-bit addresses for selecting the data register operand. An additional four bits of register address are stored in the bank select register (BSR). This addressing scheme divides data registers into 16 banks. The user needs to specifically change the value in the BSR register in order to switch to different bank.

Bank switching increases program complexity and may cause software errors. The designer of the PIC18 MCU minimized the problem of bank switching by creating the access bank. The access bank consists of the first 96 bytes of GPRs and the last 160 bytes of SFRs. Most of the PIC18 instructions have an “a” field for the user to force the access of access bank, which will ignore the BSR register during the addressing process of data registers. This scheme eliminates the need for bank switching in most cases.

The PIC18 instructions provide five different addressing modes for accessing instruction operands:

- Register direct
- Immediate
- Inherent
- Indirect
- Bit direct

The indirect mode is used mainly to access data arrays and matrices. Three indirect pointers (FSR0–FSR2) are provided to support indirect access mode. Indirect pointers can be incremented or decremented before or after the access.

1.13 Exercises

- E1.1 What is a processor? What components does it have?
- E1.2 What makes a microprocessor different from the processor of a large computer?
- E1.3 What makes an MCU different from a general-purpose microprocessor?
- E1.4 What is the length (number of bits) of a PIC18 instruction?
- E1.5 What is the length of the PIC18 program counter? How many different program memory locations can be addressed by the PIC18 MCU?
- E1.6 How many bits are used to select a data register?
- E1.7 What is the access bank? What benefits does the access bank provide?
- E1.8 What is an assembler? What is a compiler?
- E1.9 What is a cross assembler? What is a cross compiler?
- E1.10 Explain the features of EPROM, EEPROM, and flash memory.

- E1.11** What is instruction pipelining? What benefits does it provide?
- E1.12** Write an instruction sequence to swap the contents of data registers at 0x300 and 0x200.
- E1.13** Write an instruction sequence to load the value of 0x39 into data memory locations 0x100–0x103.
- E1.14** Write an instruction sequence to subtract 10 from data memory locations 0x30–0x34.
- E1.15** Write an instruction to store the contents of the WREG register in the data register located at 0x25 of bank 4.
- E1.16** Write an instruction sequence to copy the contents of data memory at 0x100–0x103 to 0x200–0x203 using the postincrement addressing mode.
- E1.17** Write an instruction sequence to copy the contents of data memory at 0x100–0x103 to 0x203–0x200, that is, in the reverse order by combining the use of postincrement and post-decrement modes.
- E1.18** Write an instruction sequence to add 5 to data registers at \$300–\$303.
- E1.19** Write an instruction sequence to add the contents of the data registers at \$10 and \$20 and store the sum at \$30.
- E1.20** Write an instruction sequence to subtract the contents of the data register at 0x20 from that of the data register at 0x30 and store the difference in the data register at 0x10.