Chapter 1: Introduction to PIC18

The PIC18 Microcontroller

Han-Way Huang

Minnesota State University, Mankato
What is a computer?

Software

Hardware

Computer Hardware Organization

Figure 1.1 Computer Organization
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

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

Inside the µc chip

Program Memory Space
(a portion of this space is on the µc chip)

21-bit program address

12-bit register address

PIC18 CPU

16-bit instruction bus

8-bit data bus

Data Memory Space
(Special function registers and general purpose RAM)

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.
The PIC18 Microcontroller

Access RAM 000h
05Fh
BSR<3:0> = 0000

GPRs
060h
0FFh
100h
1FFh

Bank 0

GPRs

Bank 1

GPRs

Bank 2

GPRs

Bank 3

GPRs

Bank 4 to Bank 13

GPRs

Bank 14

GPRs

Bank 15

Unused

SFRs

Access RAM low

Access RAM high

SFRs

000h
05Fh
060h

0FFh
100h
1FFh
200h

2FFh
300h
3FFh
400h

DFFh
E00h

EFFh
F00h
F5Fh
F60h

FFFh

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.
The PIC18 Microcontroller

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

Note: y can be 0 or 1 whereas x can be 0-F.
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>
<thead>
<tr>
<th>address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>TOSU</td>
<td>Top of stack (upper)</td>
</tr>
<tr>
<td>0xFFE</td>
<td>TOSH</td>
<td>Top of stack (high)</td>
</tr>
<tr>
<td>0xFFD</td>
<td>TOSL</td>
<td>Top of stack (low)</td>
</tr>
<tr>
<td>0xFFC</td>
<td>STKPTR</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>0xFB</td>
<td>PCLATU</td>
<td>Upper program counter latch</td>
</tr>
<tr>
<td>0xFA</td>
<td>PCLATH</td>
<td>High program counter latch</td>
</tr>
<tr>
<td>0xFFF9</td>
<td>PLC</td>
<td>Program counter low byte</td>
</tr>
<tr>
<td>0xFFF8</td>
<td>TBLPTRU</td>
<td>Table pointer upper byte</td>
</tr>
<tr>
<td>0xFFF7</td>
<td>TBLPTRH</td>
<td>Table pointer high byte</td>
</tr>
<tr>
<td>0xFFF6</td>
<td>TBLPTRL</td>
<td>Table pointer low byte</td>
</tr>
<tr>
<td>0xFFF5</td>
<td>TABLAT</td>
<td>Table latch</td>
</tr>
<tr>
<td>0xFFF4</td>
<td>PRODH</td>
<td>High product register</td>
</tr>
<tr>
<td>0xFFF3</td>
<td>PRODL</td>
<td>Low product register</td>
</tr>
<tr>
<td>0xFFF2</td>
<td>INTCON</td>
<td>Interrupt control register</td>
</tr>
<tr>
<td>0xFFF1</td>
<td>INTCON2</td>
<td>Interrupt control register 2</td>
</tr>
<tr>
<td>0xFFF0</td>
<td>INTCON3</td>
<td>Interrupt control register 3</td>
</tr>
<tr>
<td>0xFFF</td>
<td>INDFO</td>
<td>Indirect file register pointer 0</td>
</tr>
<tr>
<td>0xFE</td>
<td>POSTINC0</td>
<td>Post increment pointer 0 (to GPRs)</td>
</tr>
<tr>
<td>0xFED</td>
<td>POSTDEC0</td>
<td>Post decrement pointer 0 (to GPRs)</td>
</tr>
<tr>
<td>0xFE</td>
<td>PREINCM0</td>
<td>Preincrement pointer 0 (to GPRs)</td>
</tr>
<tr>
<td>0xFE</td>
<td>PLUSW0</td>
<td>Add WREG to FSR0</td>
</tr>
<tr>
<td>0xFE</td>
<td>FSR0H</td>
<td>File select register 0 high byte</td>
</tr>
<tr>
<td>0xFE</td>
<td>FSR0L</td>
<td>File select register 0 low byte</td>
</tr>
<tr>
<td>0xFE</td>
<td>WREG</td>
<td>Working register</td>
</tr>
<tr>
<td>0xFE</td>
<td>INDFO1</td>
<td>Indirect file register pointer 1</td>
</tr>
<tr>
<td>0xFE</td>
<td>POSTINC1</td>
<td>Post increment pointer 1 (to GPRs)</td>
</tr>
<tr>
<td>0xFE</td>
<td>POSTDEC1</td>
<td>Post decrement pointer 1 (to GPRs)</td>
</tr>
<tr>
<td>0xFE</td>
<td>PREINCM1</td>
<td>Preincrement pointer 1 (to GPRs)</td>
</tr>
<tr>
<td>0xFE</td>
<td>PLUSW1</td>
<td>Add WREG to FSR1</td>
</tr>
<tr>
<td>0xFE</td>
<td>FSR1H</td>
<td>File select register 1 high byte</td>
</tr>
<tr>
<td>0xFE</td>
<td>FSR1L</td>
<td>File select register 1 low byte</td>
</tr>
<tr>
<td>0xFE</td>
<td>BSR</td>
<td>Bank select register</td>
</tr>
<tr>
<td>0xFD</td>
<td>INDFO2</td>
<td>Indirect file register pointer 2</td>
</tr>
<tr>
<td>0xFD</td>
<td>POSTINC2</td>
<td>Post increment pointer 2 (to GPRs)</td>
</tr>
<tr>
<td>0xFD</td>
<td>POSRDEC2</td>
<td>Post decrement pointer 2 (to GPRs)</td>
</tr>
<tr>
<td>0xFD</td>
<td>PREINCM2</td>
<td>Preincrement pointer 2 (to GPRs)</td>
</tr>
<tr>
<td>0xFD</td>
<td>PLUSW2</td>
<td>Add WREG to FSR2</td>
</tr>
<tr>
<td>0xFD</td>
<td>FSR2H</td>
<td>File select register 2 high byte</td>
</tr>
<tr>
<td>0xFD</td>
<td>FSR2L</td>
<td>File select register 2 low byte</td>
</tr>
<tr>
<td>0xFD</td>
<td>STATUS</td>
<td>Status register</td>
</tr>
</tbody>
</table>

Note 1. This is not a physical register
The PIC18 Microcontroller

### Figure 1.6

![The STATUS register (0xFD8)](redraw with permission of Microchip)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>N: Negative bit</td>
</tr>
<tr>
<td>6</td>
<td>OV: Overflow bit</td>
</tr>
<tr>
<td>5</td>
<td>Z: Zero flag</td>
</tr>
</tbody>
</table>
| 4   | DC: Digit carry/borrow bit (
| 3   | For ADDWF, ADDLW, SUBLW, SUBWF instructions. |
| 2   | 1 = A carry-out from the 4th low-order bit of the result occurred. |
| 1   | 0 = No carry-out from the 4th low-order bit of the result occurred. |
| 0   | 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. |

- 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.
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.

```
<table>
<thead>
<tr>
<th>TCY0</th>
<th>TCY1</th>
<th>TCY2</th>
<th>TCY3</th>
<th>TCY4</th>
<th>TCY5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW 55h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVWF PORTB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRA sub_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSF PORTA,BIT3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction @address sub_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

Figure 1.7 An example of instruction pipeline flow

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.1-22
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

 opcode f (source file register)

15 12 11 0

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

 opcode b a f

15 12 11 9 8 7 0

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
```

<table>
<thead>
<tr>
<th>opcode</th>
<th>k</th>
</tr>
</thead>
</table>

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.
The PIC18 Microcontroller

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>07815</td>
<td>GOTO label</td>
</tr>
<tr>
<td>1111 07815</td>
<td>S = fast bit</td>
</tr>
<tr>
<td>0101115</td>
<td>CALL funct_name</td>
</tr>
<tr>
<td>07815</td>
<td>BRA label</td>
</tr>
<tr>
<td>07815</td>
<td>BC label</td>
</tr>
</tbody>
</table>

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.
  
  \[
  \text{movwf 0x20,A} ; \text{the value 0x20 is register direct mode}
  \]

- **Immediate Mode**: A value in the instruction to be used as an operand
  
  \[
  \text{addlw 0x10} ; \text{add hex value 0x10 to WREG}
  \]
  
  \[
  \text{movlw 0x30} ; \text{load 0x30 into WREG}
  \]

- **Inherent Mode**: an implied operand
  
  \[
  \text{andlw 0x3C} ; \text{the operand WREG is implied}
  \]
  
  \[
  \text{daw} ; \text{the operand WREG is implied}
  \]
- **Indirect Mode**: A special function register (FSRx) is used as a pointer to the actual data register.

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDFx</td>
<td>movwf INDFO</td>
</tr>
<tr>
<td>POSTINCx</td>
<td>movff POSTINC0,PRODL</td>
</tr>
<tr>
<td>POSTDECx</td>
<td>movf POSTDEC0,W</td>
</tr>
<tr>
<td>PREINCx</td>
<td>addwf PREINC1,F</td>
</tr>
<tr>
<td>PLUSWx</td>
<td>movff PLUSW2,PRODL</td>
</tr>
</tbody>
</table>
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 ← 0x10
Add Instructions

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

Subtract Instructions

subwf PRODH,W ; WREG ← [PRODH] – [WREG]
subwfb 0x10,F,A ; 0x10 ← [0x10] – [WREG] – borrow flag
sublw 0x10 ; WREG ← 0x10 – [WREG]
<table>
<thead>
<tr>
<th><strong>RISC</strong></th>
<th><strong>CISC</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple instruction set</td>
<td>Complex instruction set</td>
</tr>
<tr>
<td>Regular and fixed instruction format</td>
<td>Irregular instruction format</td>
</tr>
<tr>
<td>Simple address modes</td>
<td>Complex address modes</td>
</tr>
<tr>
<td>Pipelined instruction execution</td>
<td>May also pipeline instruction execution</td>
</tr>
<tr>
<td>Separated data and program memory</td>
<td>Combined data and program memory</td>
</tr>
<tr>
<td>Most operations are register to register</td>
<td>Most operations can be register to memory</td>
</tr>
<tr>
<td>Take shorter time to design and debug</td>
<td>Take longer time to design and debug</td>
</tr>
<tr>
<td>Provide large number of CPU registers</td>
<td>Provide smaller number of CPU registers</td>
</tr>
</tbody>
</table>