# 第 4 章: PIC 指令系统

井艳军

沈阳工业大学电气工程学院

# 主要内容

指令流水线

指令集说明

数据传送类指令

算术运算类指令

逻辑运算类指令

控制转移类指令

# PIC 指令系统

指令就是用来指挥 CPU 完成某一项基本操纵的命令，单片机能识别的全部指令的集合，称为单片机的指令系统或指令集。不同厂家的单片机，或者不同 CPU 内核的单片机，一般具有不同的指令系统。

| 产品等级 | 指令集数量 | 每条指令字节长度 | 主要代表产品 |
|---|---|---|---|
| 初级 | 33 | 12 | PIC12C5XX |
| 中级 | 35 | 14 | PIC16F87X |
| 高级 | 58 | 16 | PIC18CXXX |

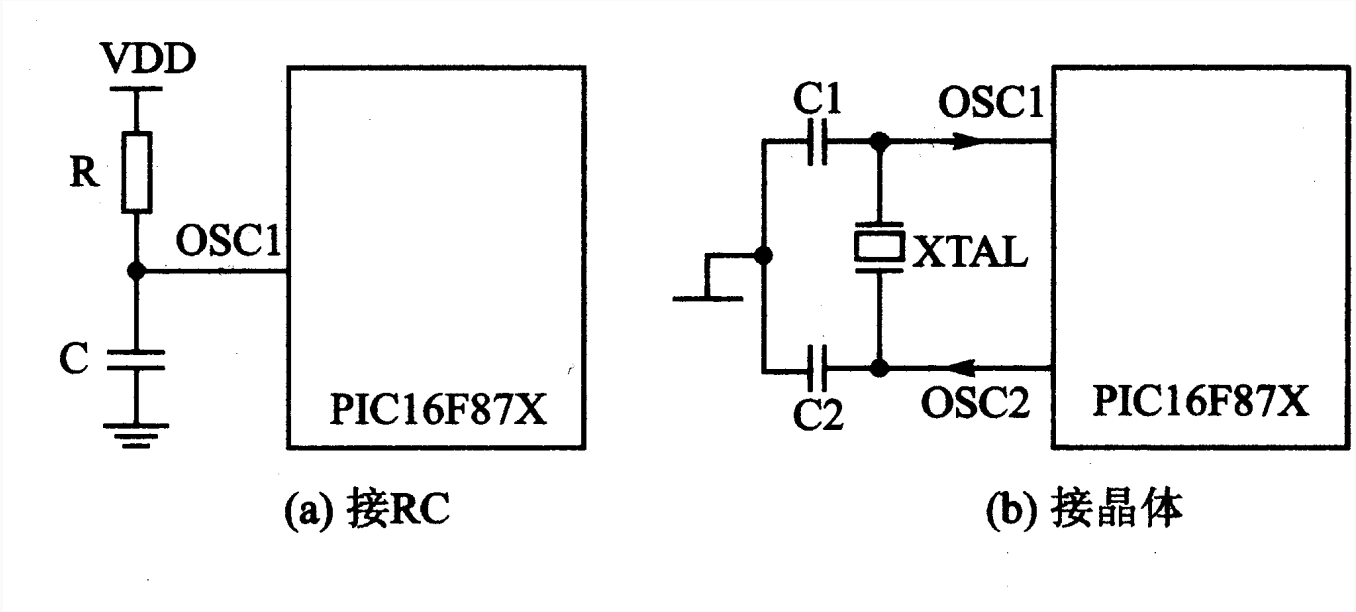
# 指令流水线

# 指令时序

时序是单片机中比较重要的技术之一，所谓“时序”就是“时间的次序”的意思，即单片机内部各部件操作的时间规范，这样才能步调一致，统一协调地工作。

任何计算机的时序都是由振荡电路控制的，当 PIC 单片机的 OSC1 和 OSC2 加上晶体振荡器以后，片内的振荡电路就会产生周期性的脉冲，这个脉冲称为“时钟脉冲”，时钟脉冲的频率和周期分别称为“时钟频率”和“时钟周期”，分别记为 $f_{OSC}$ 和 $T_{OSC}$ 。

# 指令时序



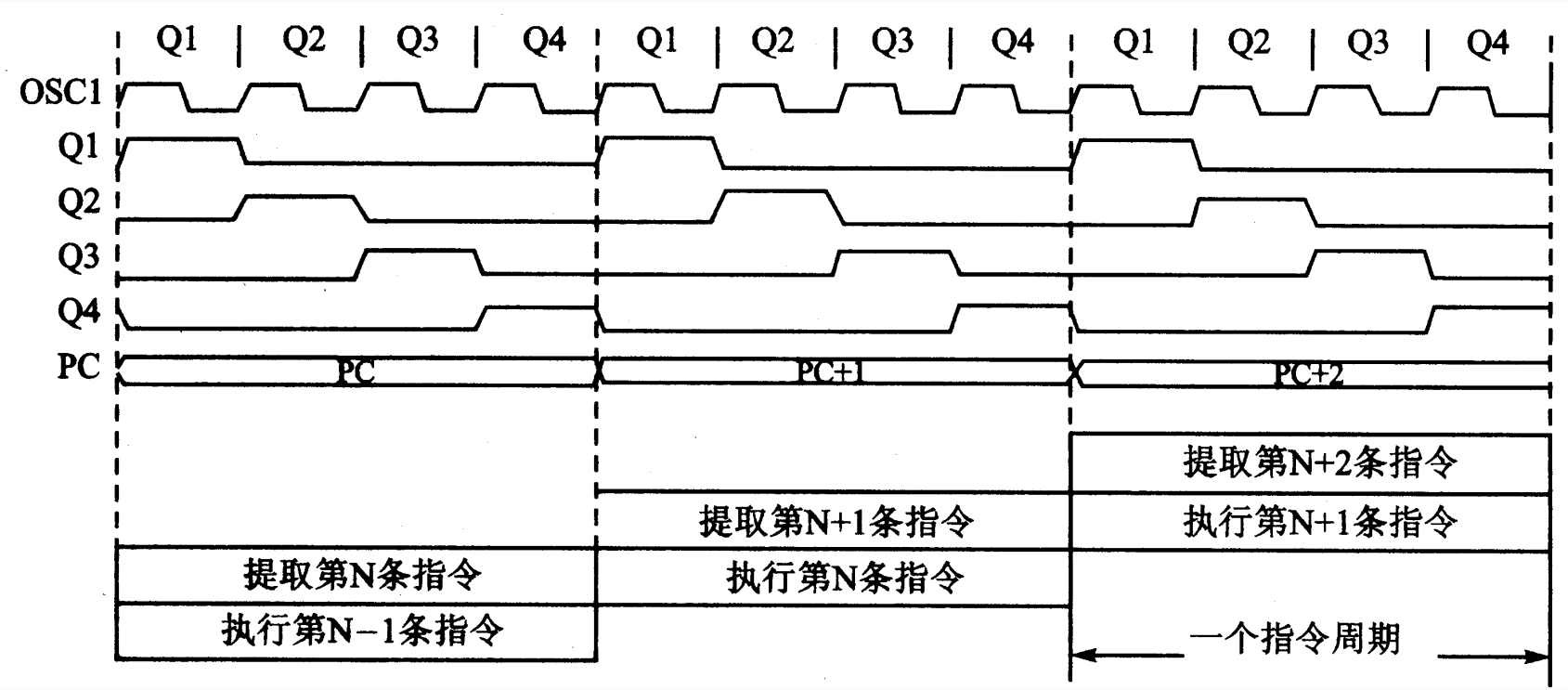

单片机工作时，总是逐条地从 ROM 中取指令，然后逐条执行。计算机访问一次存储器的时间，称为一个“机器周期”，这是一个时间基准，好象我们用“秒”作为时间基准一样，而“指令周期”就是指执行一条指令所花费的时间，记为 $T_{CYC}$ 。

# 指令流水线

下面我们讨论一下，PIC 单片机系统中，指令周期 $T_{CYC}$ 和时钟周期 $T_{OSC}$ 之间的关系。

由时钟振荡器电路产生的时钟信号并不是直接作为 PIC 单片机的工作脉冲的，而是在片内经过一个 1/4 倍数的分频器，四分频后形成四个不重叠的方波，记为 Q1～Q4。见下图:

# 指令流水线




$Q1 \sim Q4$ 也称为四个节拍，四个节拍组合一起构成了一个指令周期 $T_{CYC}$，即一个指令周期包含了四个时钟周期 $T_{OSC}$。

# 指令流水线

从图中可以看出，在一个指令周期内，PIC 单片机要完成两部分的工作:

一是执行指令；二是从程序存储器取出下一条指令。也即在同一个指令周期中，完成了当前指令的执行，同时取出了下一条待执行的指令。

以上的特点是很多其他类型的单片机所没有的特点。

这是因为 PIC 单片机采用了哈佛结构，使得程序存储器 (存放指令) 的访问和数据存储器 (存放运算数据) 的访问可以并行处理，构成了所谓的“指令流水线”结构，简称指令流。

# 指令流水线

| | 第 $N$–3 周期 | 第 $N$–2 周期 | 第 $N$–1 周期 | 第 $N$ 周期 | 第 $N$+1 周期 | 第 $N$+2 周期 |
|---|---|---|---|---|---|---|
| 执行： | （PC–3） | （PC–2） | （PC–1） | （PC） | （PC+1） | （PC+2） |
| 取指： | （PC–2） | （PC–1） | （PC） | （PC+1） | （PC+2） | （PC+3） |

从指令流中可以看出:

在 PIC 单片机中，一条指令的取指操作和译码执行操作，实际上是在两个指令周期内完成的。但是由于“指令流水线”的结构，使得每条指令的取指和执行过程平均只花费一个指令周期，除了个别跳转指令。因此人们习惯于说成，PIC 单片机采用的是单周期指令。

# 指令集说明

# 指令集说明

每条指令的字节长度为 14 位，主要由说明指令功能的操作码和参与指令处理的操作数组成。

操作码部分，简称助记符，如表 4-2 核心助记符，是借用英语单词来间接表达和定义其操作功能。

操作数部分，是按照操作码的操作功能，对操作数进行处理。

根据操作数的源地址和目标地址的访问性质，可以有多种表现形式：主要有直接寻址、间接寻址、立即寻址和位寻址四类。

# 核心助记符

| 助记符 | 功能说明 | 助记符 | 功能说明 |
| --- | --- | --- | --- |
| ADD | 相加 | MOV | 传送 |
| SUB | 相减 | RL | 左移 |
| AND | 与 | RR | 右移 |
| IOR | 或 | CLR | 清零 |
| XOR | 异或 | COM | 取反 |
| INC | 加 1 | RET | 返回 |
| DEC | 减 1 | BTF | 测试 |

# 指令系统补充字符说明

| 字符 | 功能说明 |
| --- | --- |
| W | 工作寄存器（即累加器） |
| f | 寄存器地址（取 7 位寄存器地址，00H 到 7FH） |
| b | 8 位寄存器内位地址（0 到 7） |
| K | 立即数（8 位常数或 11 位地址）、常量或标号 |
| L | 指令操作数中含有 8 位立即数 K |
| d | 目标地址选择：d = 0, 结果至 W；d = 1, 结果至 f |

# 指令系统补充字符说明

| 字符 | 功能说明 |
| --- | --- |
| FSZ | 寄存器 f 为 0，则间跳 |
| FSC | 寄存器 f 的 b 位为 0，则间跳 |
| FSS | 寄存器 f 的 b 位为 1，则间跳 |
| ( ) | 表示寄存器的内容 |
| (( )) | 表示寄存器简介寻址的内容 |
| $\rightarrow$ | 表示运算结果送入目标寄存器 |

# 数据传送类指令

# 数据传送类指令

数据传送类指令共有 4 条指令，主要功能是将数据从源地址（或立即数）传送至目标地址中。

| 助记符 | 操作说明 | 影响的标志位 |
| --- | --- | --- |
| MOVF f,d | f 传送至 d | Z |
| MOVWF f | W 传送至 f | - |
| MOVLW K | K 传送至 W | - |
| SWAPF f,d | f 半字节交换至 d | - |

# 数据传送类指令 1

## MOVF f,d

**说明:** 将 f 寄存器内容传送至 W(d=0) 或 f(d=1)

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** $(f) \rightarrow (d)$

**标志:** Z

# 数据传送类指令 2

## MOVWF f

**说明:** 将 W 寄存器内容传送至 f 寄存器

**数域:** $0 \leqslant f \leqslant 127$

**操作:** $(W) \rightarrow (f)$

**标志:** 无

# 数据传送类指令 3

## MOVLW K

**说明:** 将立即数 K 传送至 W 寄存器

**数域:** $0 \leqslant K \leqslant 255$

**操作:** $K \rightarrow (W)$

**标志:** 无

# 数据传送类指令 4

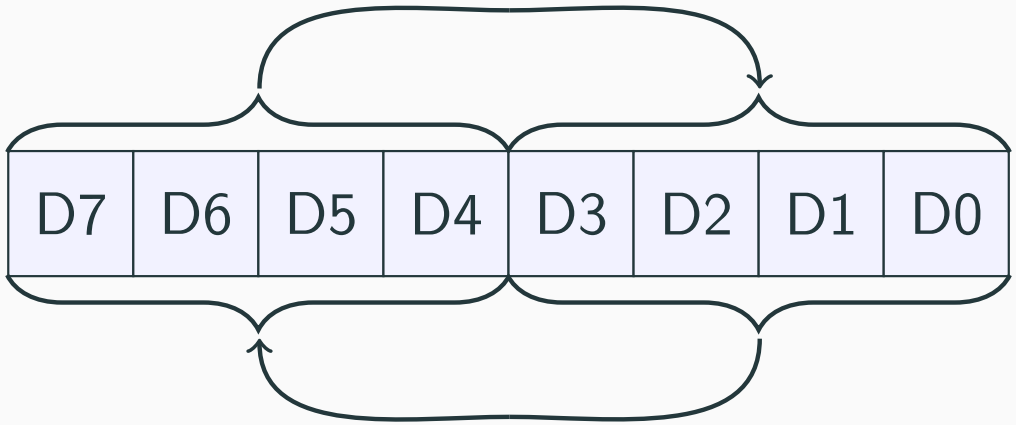
## SWAPF f,d

**说明：** 将 f 寄存器的内容高 4 位和低 4 位交换后，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作：** $(f_{0\sim3} \longleftrightarrow f_{4\sim7}) \rightarrow (d)$

**标志：** 无



# 数据传送类指令

**例题 4-1**

请利用数据传送类指令编写一段子程序，将立即数 20H 传送到通用寄存器 20H 中。

# 数据传送类指令

**例题 4-2**

请利用数据传送类指令编写一段子程序，将通用寄存器 20H 和 30H 中的内容进行交换。

# 算术运算类指令

# 算术运算类指令

算术运算类指令是 PIC 单片机指令系统中，承担运算功能的重要部分，共有 6 条指令。主要有加减指令、增量和减量指令

| 助记符 | 操作说明 | 影响的标志位 |
| --- | --- | --- |
| ADDWF f,d | f 加 W 至 d | C、DC、Z |
| SUBWF f,d | f 减 W 至 d | C、DC、Z |
| ADDLW K | K 加 W 至 W | C、DC、Z |
| SUBLW K | K 减 W 至 W | C、DC、Z |
| INCF f,d | f 加 1 至 d | Z |
| DECF f,d | f 减 1 至 d | Z |

# 算数运算类指令 1

## ADDWF f,d

**说明：** 将 f 寄存器的内容加 W 寄存器的内容，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0,1]$

**操作：** $(f) + (W) \rightarrow (d)$

**标志：** C、DC、Z

# 算数运算类指令 2

## SUBWF f,d

**说明：** 将 f 寄存器的内容减 W 寄存器的内容，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作：** $(f) - (W) \rightarrow (d)$

**标志：** C、DC、Z

# 算数运算类指令 3

## ADDLW K

**说明：** 将立即数 K 加 W 寄存器的内容，送至 W

**数域：** $0 \leqslant K \leqslant 255$

**操作：** $K + (W) \rightarrow (W)$

**标志：** C、DC、Z

# 算数运算类指令 4

## SUBLW K

**说明:** 将立即数 K 减 W 寄存器的内容，送至 W

**数域:** $0 \leqslant K \leqslant 255$

**操作:** $K - (W) \rightarrow (W)$

**标志:** C、DC、Z

# 算数运算类指令 5

**INCF f,d**

**说明:** 将 f 寄存器内容加 1，送至 W(d=0) 或 f(d=1)

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** $(f) + 1 \rightarrow (d)$

**标志:** Z

# 算数运算类指令 6

## DECF f,d

**说明:** 将 f 寄存器内容减 1，送至 W(d=0) 或 f(d=1)

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** $(f) - 1 \rightarrow (d)$

**标志:** Z

# 算数运算类指令

加减运算是组成程序的核心部分，将对标志位 Z、DC 和 C 产生影响。有三点需要重视:

- 目标操作地址位 d，这是一种通用的表达方式，将引出两种可能的目标地址，即 W（d = 0）或数据存储器 f（d = 1）。

# 算数运算类指令

加减运算是组成程序的核心部分，将对标志位 Z、DC 和 C 产生影响。有三点需要重视:

- 目标操作地址位 d，这是一种通用的表达方式，将引出两种可能的目标地址，即 W（d = 0）或数据存储器 f（d = 1）。
- 以上两条相减指令中，注意减数都是 W 工作寄存器。如果被减数大于或等于减数 W，则运算结果状态标志位 C 为 1；如果被减数小于减数 W，则运算结果标志位 C 位 0。半进位标志 DC 也一样，如果有借位 DC 位 0；如果无借位 DC 位 1。

# 算数运算类指令

加减运算是组成程序的核心部分，将对标志位 Z、DC 和 C 产生影响。有三点需要重视:

- 目标操作地址位 d，这是一种通用的表达方式，将引出两种可能的目标地址，即 W（d = 0）或数据存储器 f（d = 1）。
- 以上两条相减指令中，注意减数都是 W 工作寄存器。如果被减数大于或等于减数 W，则运算结果状态标志位 C 为 1; 如果被减数小于减数 W，则运算结果标志位 C 位 0。半进位标志 DC 也一样，如果有借位 DC 位 0；如果无借位 DC 位 1。
- 在通用指令语句中，f 表示数据存储器的直接地址，取值为 $00H \sim 7FH$。

# 算数运算类指令

**例题 4-3**

请将通用寄存器 20H、30H 构成的 16 位数据与通用寄存器 40H、50H 构成的 16 位数据相加后放入 40H、50H 中，已知其和不会超出 65535。

# 算数运算类指令

**练习**

请将通用寄存器 20H、30H 构成的 16 位数据与通用寄存器 40H、50H 构成的 16 位数据相减后放入 40H、50H 中。

# 逻辑运算类指令

逻辑运算类指令是一组比较复杂的指令，形式较多，可以对位和字节进行逻辑操作。主要有与、或、异或、清零、置位、取反和左右移位等 14 条指令。

# 逻辑运算类指令

# 逻辑运算类指令

| 助记符 | | 操作说明 | 影响的标志位 |
|---|---|---|---|
| CLRF | f | f 清零 | Z |
| CLRW | | W 清零 | Z |
| CLRWDT | | WDT 清零 | $\overline{T0}$ 、$\overline{PD}$ |
| BCF | f,b | f 的 b 位清零 | - |
| BSF | f,b | f 的 b 位置位 | - |
| RLF | f,d | f 带 C 左循环 | C |
| RRF | f,d | f 带 C 右循环 | C |

# 逻辑运算类指令

| 助记符 | 操作说明 | 影响的标志位 |
| --- | --- | --- |
| ANDWF f,d | W 与 f 至 d | Z |
| IORWF f,d | W 或 f 至 d | Z |
| XORWF f,d | W 异或 f 至 d | Z |
| ANDLW K | K 与 W 至 d | Z |
| IORLW K | K 或 W 至 d | Z |
| XORLW K | K 异或 W 至 d | Z |
| COMF f,d | f 取反至 d | Z |

# 逻辑运算类指令 1

## CLRF f

**说明:** f 寄存器内容清零

**数域:** $0 \leqslant f \leqslant 127$

**操作:** $0 \rightarrow (f)$

**标志:** Z

# 逻辑运算类指令 2

## CLRW

**说明:** W 寄存器内容清零

**操作:** $0 \rightarrow (W)$

**标志:** Z

# 逻辑运算类指令 3

## CLRWDT

**说明:** WDT 寄存器内容清零

**操作:** $0 \rightarrow (WDT), 0 \rightarrow WDT$ 预分频器 $, 1 \rightarrow \overline{T0}, 1 \rightarrow \overline{PD}$

**标志:** $\overline{T0}$、$\overline{PD}$

# 逻辑运算类指令 4

## BCF f,b

**说明:** f 寄存器内容的 b 位清零

**数域:** $0 \leqslant f \leqslant 127, 0 \leqslant b \leqslant 7$

**操作:** $0 \rightarrow (f_b)$

**标志:** 无

# 逻辑运算类指令 5

## BSF f,b

**说明:** f 寄存器内容的 b 位置位

**数域:** $0 \leqslant f \leqslant 127, 0 \leqslant b \leqslant 7$

**操作:** $1 \rightarrow (f_b)$

**标志:** 无

# 逻辑运算类指令 6

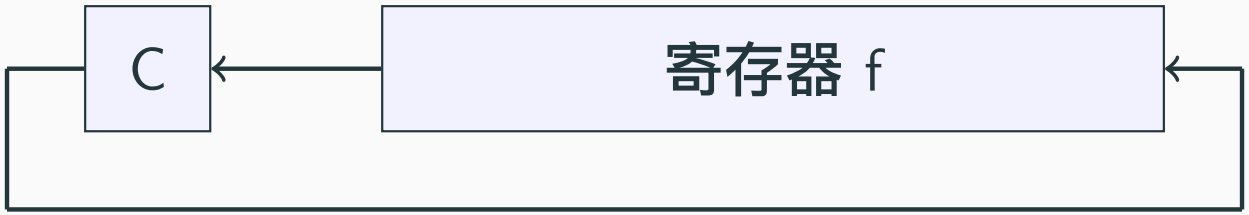
**RLF f,d**

**说明：** f 寄存器的内容带 C 左循环后，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作：** 下图

**标志：** C



# 逻辑运算类指令 7

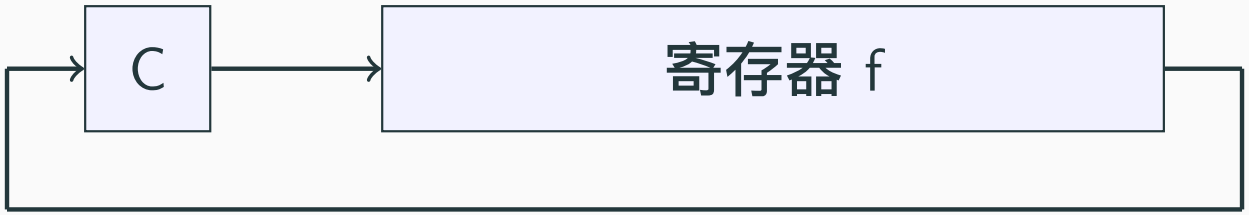
## RRF f,d

**说明:** f 寄存器的内容带 C 右循环后，送至 W(d=0) 或 f(d=1)

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** 下图

**标志:** C



# 逻辑运算类指令 8

## ANDWF f,d

**说明：** 将 f 寄存器的内容和 W 寄存器的内容相与后，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作：** $(f) \wedge (W) \rightarrow (d)$

**标志：** Z

# 逻辑运算类指令 9

## IORWF f,d

**说明：** 将 f 寄存器的内容和 W 寄存器的内容相或后，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0,1]$

**操作：** $(f) \vee (W) \rightarrow (d)$

**标志：** Z

# 逻辑运算类指令 10

## XORWF f,d

**说明：** 将 f 寄存器的内容和 W 寄存器的内容相异或后，送至 W(d=0) 或 f(d=1)

**数域：** $0 \leqslant f \leqslant 127, d \in [0,1]$

**操作：** $(f) \oplus (W) \rightarrow (d)$

**标志：** Z

# 逻辑运算类指令 11

## ANDLW K

**说明：** 将立即数 K 和 W 寄存器的内容相与后，送至 W

**数域：** $0 \leqslant K \leqslant 255$

**操作：** $K \wedge (W) \rightarrow (W)$

**标志：** Z

# 逻辑运算类指令 12

## IORLW K

**说明:** 将立即数 K 和 W 寄存器的内容相或后，送至 W

**数域:** $0 \leqslant K \leqslant 255$

**操作:** $K \vee (W) \rightarrow (W)$

**标志:** Z

# 逻辑运算类指令 13

## XORLW K

**说明：** 将立即数 K 和 W 寄存器的内容相异或后，送至 W

**数域：** $0 \leqslant K \leqslant 255$

**操作：** $K \oplus (W) \rightarrow (W)$

**标志：** Z

# 逻辑运算类指令 14

## COMF f,d

**说明:** 将 f 寄存器的内容取反后，送至 W(d=0) 或 f(d=1)

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** $(\bar{f}) \rightarrow (d)$

**标志:** Z

# 逻辑运算类指令

**例题 4-4**

请将数据存储器 20H 和 30H 中的数据分别与立即数 20H、30H 相与和相或后相加，结果放入 40H 存储器中，请编写相应的程序。

# 逻辑运算类指令

**例题 4-5**

请编写一个完整的程序，将数据存储器 20H 低 4 位和 30H 高 4 位组合成一个八位二进制数据，并从 RC 端口输出。

# 控制转移类指令

控制转移类指令，是在指令系统中形式灵活、功能较强的一组指令，共 11 条。它们是构成程序循环和跳转的关键要素，一般可以分为有条件跳转和无条件跳转两大类。

# 控制转移类指令

# 控制转移类指令

| 助记符 | | 操作说明 | 影响的标志位 |
|---|---|---|---|
| CALL | K | 调用 K 处子程序 | - |
| GOTO | K | 跳转至 K 处 | - |
| INCFSZ | f,d | f 加 1 至 d，为 0 间跳 | - |
| DECFSZ | f,d | f 减 1 至 d，为 0 间跳 | - |
| BTFSC | f,b | f 的 b 位，为 0 间跳 | - |
| BTFSS | f,b | f 的 b 位，为 1 间跳 | - |
| RETFIE | | 中断返回 | - |
| RETLW | K | 子程序返回（K 传递给 W） | - |
| RETURN | | 子程序返回 | - |
| NOP | | 空操作 | - |
| SLEEP | | 进入休眠状态 | $\overline{T0}$ 、$\overline{PD}$ |

# 控制转移类指令 1

## CALL K

**说明:** 调用 K 处子程序

**数域:** $0 \leqslant K \leqslant 2047$

**操作:** $(PC) + 1 \rightarrow TOS$（堆栈）

$K \rightarrow PC_{0\sim10}$

$(PCLATH_{3\sim4}) \rightarrow PC_{11\sim12}$

**标志:** 无

# 控制转移类指令 2

## GOTO K

**说明:** 无条件跳转至 K 处

**数域:** $0 \leqslant K \leqslant 2047$

**操作:** $K \rightarrow PC_{0\sim10}$

$(PCLATH_{3\sim4}) \rightarrow PC_{11\sim12}$

**标志:** 无

# 控制转移类指令 3

## INCFSZ f,d

**说明:** f 加 1 传送至 W(d=0) 或 f(d=1)，为 0 则间跳

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** $(f) + 1 \rightarrow (d)$,

若 $(f) + 1 == 0$ 成立,

则 $(PC) + 2 \rightarrow PC$;

否则 PC 自动加 1

**标志:** Z

# 控制转移类指令 4

## DECFSZ f,d

**说明:** f 减 1 传送至 W(d=0) 或 f(d=1)，为 0 则间跳

**数域:** $0 \leqslant f \leqslant 127, d \in [0, 1]$

**操作:** $(f) - 1 \rightarrow (d)$,

若 $(f) - 1 == 0$ 成立,

则 $(PC) + 2 \rightarrow PC$;

否则 PC 自动加 1

**标志:** Z

# 控制转移类指令 5

## BTFSC f,d

**说明:** 测试 f 寄存器内容的 b 位，为 0 则间跳

**数域:** $0 \leqslant f \leqslant 127, 0 \leqslant b \leqslant 7$

**操作:** 若 $(f_b) == 0$ 成立，

则 $(PC) + 2 \rightarrow PC$;

否则 PC 自动加 1

**标志:** 无

# 控制转移类指令 6

**BTFSS f,d**

**说明：** 测试 f 寄存器内容的 b 位，为 1 则间跳

**数域：** $0 \leqslant f \leqslant 127, 0 \leqslant b \leqslant 7$

**操作：** 若 $(f_b) == 1$ 成立，

则 $(PC) + 2 \rightarrow PC$;

否则 PC 自动加 1

**标志：** 无

# 控制转移类指令 7

## RETFIE

**说明:** 从中断服务程序返回

**操作:** TOS（栈顶数据）$\rightarrow$ *PC*

$1 \rightarrow$ *GIE*（总中断使能位）

**标志:** 无

# 控制转移类指令 8

**RETLW K**

**说明:** 将立即数传送至 W，返回至原断点（对应 CALL 子程序）

**数域:** $0 \leqslant K \leqslant 255$

**操作:** $K \rightarrow (W)$

TOS（栈顶数据）$\rightarrow PC$

**标志:** 无

# 控制转移类指令 9

## RETURN

**说明:** 从 CALL 子程序返回

**操作:** TOS（栈顶数据）$\rightarrow PC$

**标志:** 无

# 控制转移类指令 10

## NOP

**说明:** 空操作，单挑指令周期的延时

**操作:** 无

**标志:** 无

# 控制转移类指令 11

## SLEEP

**说明:** 睡眠状态

**操作:** $0 \rightarrow (WDT)$

$0 \rightarrow WDT$ 预分频器

$1 \rightarrow \overline{T0}$

$1 \rightarrow \overline{PD}$

**标志:** $\overline{T0}$ 、$\overline{PD}$

# 控制转移类指令两点注意

## 1、相对转移间跳

这是一种比较特殊的转移形式，根据位测试或加减 1 后的内容判断条件的成立与否，而决定程序继续执行还是间跳执行指令。

当前判断语句 A

下一条语句 B

再下一条语句 C

# 控制转移类指令两点注意

## 2、绝对转移和调用

PIC 指令系统的绝对转移，主要由 CALL 和 GOTO 语句引出。在指令机器码内部本身并没有携带完整的转移目标地址，只包含低 11 位地址，而高 2 位将由 PCLATH 寄存器给出。

**例题 4-6**

请将通用寄存器单元 20H-2FH，分别对应赋值 20H-2FH，请编写相应的软件程序。

**练习**

请将通用寄存器单元 30H-3FH，分别与 40-4FH 单元相加，结果放入 40H-4FH，请编写相应的软件程序。

## 例题 4-7

请分析以下程序片段，并指出当程序执行完后，涉及到的所有存储器单元的结果。

```
1     MOVLW   22H
2     MOVWF   22H
3     MOVWF   FSR
4     ADDWF   INDF,F
5     INCF    INDF
6     SWAPF   22H,W
7     RLF     22H,W
8     DECF    FSR,F
9     MOVWF   INDF
10    BSF     INDF,7
```

# 指令码的分配格式

| 类　型 | 14位指令码D13~D0的分配格式 | | |
|---|---|---|---|
| 面向字节操作类<br>（对寄存器 操作） | 13～8（6bit） | 7 （1bit） | 6～0 （7bit） |
| | 操作码 | d | F（寄存器地址） |
| 面向位操作类<br>对寄存器的某一位 | 13～10（4bit） | 9～7（3bit） | 6～0 （7bit） |
| | 操作码 | B | F（寄存器地址） |
| 常数操作类<br>（立即数操作）<br><br>控制操作类<br>（转移、调用） | 带8位常数 | 13～8（6bit） | 7～0 （8bit） |
| | | 操作码 | K（数据） |
| | 带11位常数的CALL 和 GOTO | 13～11（3bit） | 10～0 （11bit） |
| | | 操作码 | K（程序地址） |
| | 不带常数 | 13～0 | |
| | | 操作码 | |

# 指令系统小结

- 在大多数指令的格式中，指令中表示数据目标寄存器的方式是通过“d”来指示的:

  d=0(W) 表明目标地址是 W;

  d=1(F) 表明目标地址是文件寄存器。

  在编写指令的过程中，如果忽略了 d 的描述，则编译器按照 d=1(F) 处理;

# 指令系统小结

- 在大多数指令的格式中，指令中表示数据目标寄存器的方式是通过“d”来指示的:

  d=0(W) 表明目标地址是 W;

  d=1(F) 表明目标地址是文件寄存器。

  在编写指令的过程中，如果忽略了 d 的描述，则编译器按照 d=1(F) 处理;
- 在访问 RAM 所采用的“直接寻址”时，指令本身的 7 位地址码无法对 9 位地址码的空间“直接”覆盖、访问，所以:

  先由 STATUS 中的 RP0、RP1 事先选体;

  再由指令中的 7 位地址在“体内”寻址。

# 指令系统小结

- 对文件寄存器 F 赋初值只能通过 W 实现，指令系统不支持对 F 的直接赋值;

# 指令系统小结

- 对文件寄存器 F 赋初值只能通过 W 实现，指令系统不支持对 F 的直接赋值;
- RAM 中的数据不能直接交换、传送，只能靠 W 中转;

# 指令系统小结

- 对文件寄存器 F 赋初值只能通过 W 实现，指令系统不支持对 F 的直接赋值;
- RAM 中的数据不能直接交换、传送，只能靠 W 中转;
- 所有的“条件转移”都是“跳一步——skip ”;

# 指令系统小结

- 对文件寄存器 F 赋初值只能通过 W 实现，指令系统不支持对 F 的直接赋值;
- RAM 中的数据不能直接交换、传送，只能靠 W 中转;
- 所有的“条件转移”都是“跳一步——skip ”;
- 没有专用的堆栈操作，因为 13 位硬件是专用于“程序断点”的自动保护。“数据保护”使用字节传送操作在 RAM 实现;

# 指令系统小结

- 对文件寄存器 F 赋初值只能通过 W 实现，指令系统不支持对 F 的直接赋值;
- RAM 中的数据不能直接交换、传送，只能靠 W 中转;
- 所有的“条件转移”都是“跳一步——skip ”;
- 没有专用的堆栈操作，因为 13 位硬件是专用于“程序断点”的自动保护。“数据保护”使用字节传送操作在 RAM 实现;
- 指令系统没有乘法、除法指令。需要是靠自程序实现;

# 指令系统小结

- 对文件寄存器 F 赋初值只能通过 W 实现，指令系统不支持对 F 的直接赋值;
- RAM 中的数据不能直接交换、传送，只能靠 W 中转;
- 所有的“条件转移”都是“跳一步——skip ”;
- 没有专用的堆栈操作，因为 13 位硬件是专用于“程序断点”的自动保护。“数据保护”使用字节传送操作在 RAM 实现;
- 指令系统没有乘法、除法指令。需要是靠自程序实现;
- 所有的指令“单字节”，绝大多是指令在运行中都是“单周期”。