第 0 章

单片机概论

0

前言

科学日新月异，一日千里，随时随地都有新的产品诞生。在这些新的产品中，很多使用了单片机。正因为单片机得到如此广泛地应用，我们有必要去研究它。到底什么是单片机呢？为什么这么多的产品用到它，为什么这么多人用到它，它可用在何处，我们又如何用它来开发新的产品？这些都是下面要介绍的。

什么是单片机

相信大家都玩过“掌中宝”——俄罗斯方块，只要按键，屏幕上的图案就会旋转，就会排列，还会发出“加油”“加油”的声音，是不是觉得很过瘾。如果你是好奇的人，可能会拆开里面看看是什么东西，你可能会觉得惊讶或者无趣，因为里面只有一颗 IC 或一坨黑胶，其实那颗 IC 正是一颗单片机，你可以想象按键就是电脑的键盘，LCD（液晶显示幕）就像电脑的屏幕，那谁是主机板呢？就是那颗 IC!！电脑主机板有中央处理单元（CPU）、存储器（RAM、ROM）、计时器（TIMER）、中断控制器（Interrupt Controller）、输入/输出（I/O）等功能。这些功能都被做在同一块芯片中，看起来像是一个小型的电脑，因此我们称之为单片机。

当然，单片机并没有像电脑那样强的功能，一般拿来做简单的控制，家电的遥控器，

掌上型电动玩具或者做为一个小系统的简单的控制中心。

单片机开发系统

进入单片机世界最快速的最佳方法当然是使用单片机开发系统，开发系统提供用户一个整合的开发环境，用户可以在这个系统下进行编程、汇编、查错，好一点的系统甚至提供用户一个整合的开发环境，让用户能完全掌握单片机的一举一动。Holtek 的 HT-IDE2000 就是一个功能完善的整合开发环境，Windows 的工作平台让用户不用离开这个环境就能将其产品迅速地开发完成。

第一章

HT48CX0 系列
八位单片机介绍

1

HT48CX0 系列单片机特性

➔ **Holtek 的 HT48CX0 系列单片机主要特性如下：**

- 工作电压：2.4~5.2V
- 最大 8K×16 ROM
- 系统时钟： 400kHZ~4MHZ
- 最大 224×8 RAM
- 可选择 RC 振荡或石英晶体振荡
- 耗电流
  ——静态电流 2μA(5V),1μA(3V)
  ——工作电流 3.4mA(5V、4MHZ Crystal)
- 看门狗计时器（Watchdog Timer）
- 省电模式和唤醒（Wake-up）功能
- 12~56 个双向 I/O 口
- 64 条指令
- 外部中断功能
- 指令周期可达 1μS（系统时钟 4MHZ）
- 最大 2 组 16 位可编程定时/计数器
- 最大 16 位查表指令
- 最大 8 层堆栈
- 提供位操作指令
- OTP（One-Time Programmable）形式
- 强大的 Windows 整合开发环境

➔ **下表为 Holtek 标准 I/O 型单片机区别：**

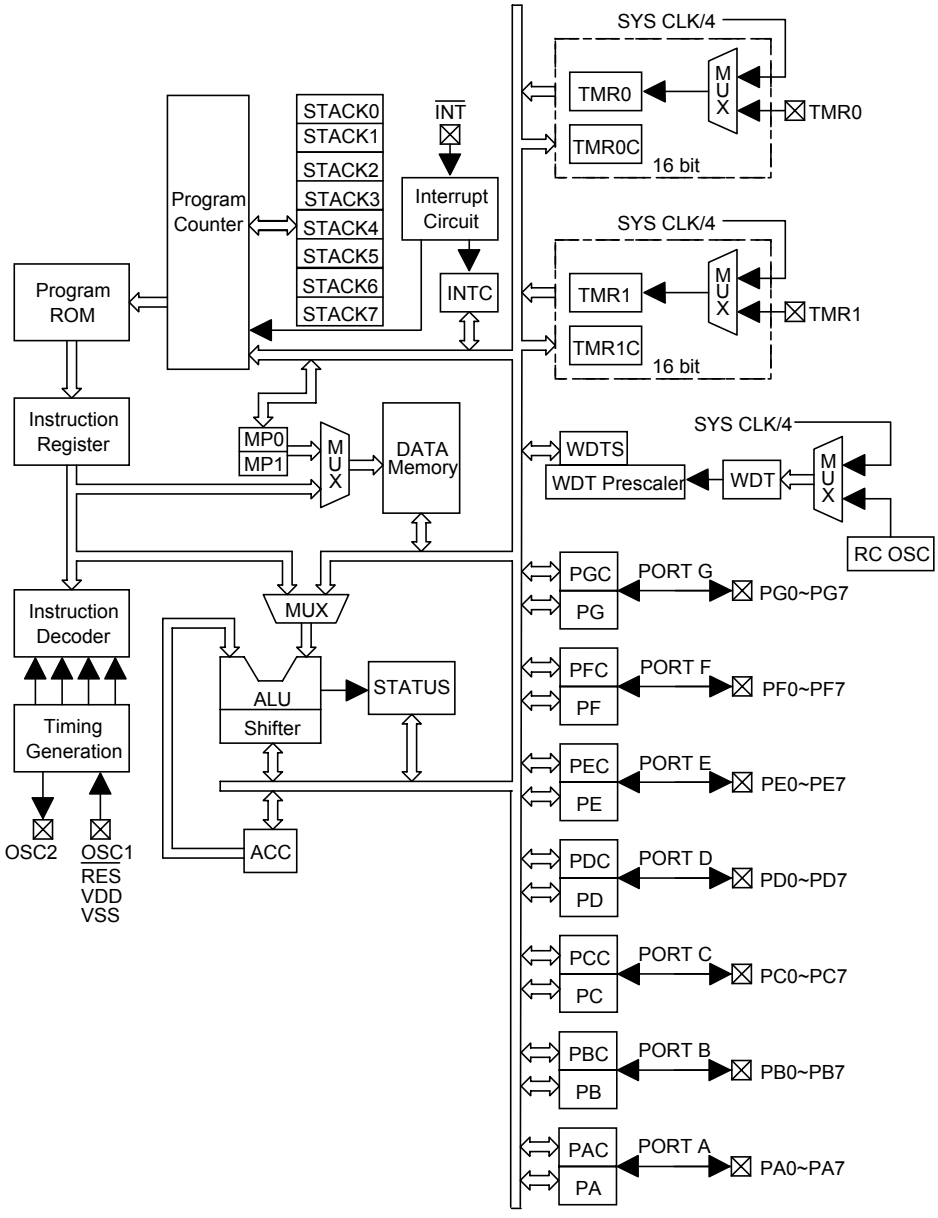
	HT48C10	HT48C30	HT48C50	HT48C70
I/O 口	18	22	32	56
内部中断源	1	1	2	2
外部中断源	1	1	1	1
8 位计时/计数器	1	1	1	—
16 位计时/计数器	—	—	1	2
数据存储器	64 Bytes	96 Bytes	160 Bytes	224 Bytes
程序存储器	1K×14	2K×14	4K×15	8K×16
堆栈	2	2	4	8
存储器指针	1×7 Bits	1×7 Bits	2×8 Bits	2×8 Bits
中断向量	2	2	3	3

注意：如果您想从一种型号的单片机转换到另一种型号单片机，以下有一些事项用户必须加以考虑：

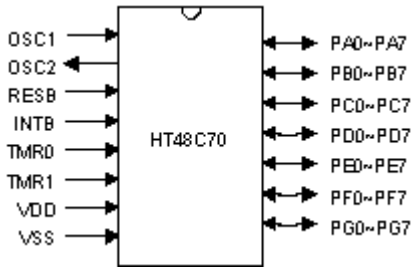
- ROM
  某个特殊的动作也许会在 ROM 的高位存放数值，而利用查表指令去取得时，就必须考虑 PC 高位字长问题。例如 HT48C10、HT48C30 ROM 高位只有 6 位，HT48C50 有 7 位，而 HT48C70 才有 8 位。
- RAM
  I/O 型单片机 RAM 的有效范围是不同的。
- MP
  HT48C10 和 HT48C30 的 MP 为 7 位，HT48C50 和 HT48C70 的 MP 为 8 位，未定义的部分读出值恒为 1
- PC 口
  HT48C10 的 PC 口只有 2 位，HT48C30 的 PC 口只有 6 位，未定义的部分读出来的值恒为 0。
- 其他不同处还有中断源、定时/计数器、堆栈等等。

系统框图和引脚说明

➔ 系统框图如下所示：（**HT48C70**）



我们先由 HT48C70 的外部引脚结构介绍起，让用户很快地了解 Holtek 八位单片机的结构。如下图所示为 HT48C70 外部引脚图和这些引脚的功能。



—OSC1（输入）OSC2（输出）

OSC1 为振荡源的输入脚，OSC2 为输出脚，在这两个引脚上加一个石英晶体或陶瓷谐振器，就可以让单片机得到系统振荡源（Fsys）。RC 振荡模式时，OSC2 输出的频率为 Fsys/4。

—INTB（外部中断信号输入脚）

该引脚为外部中断请求的输入脚，内置上拉电阻，下降沿触发。

—TMR0 和 TMR1

这两个引脚是外部事件计数输入脚，当设置为内部计时功能时，不能接收外部触发。

—PA0~PA7

双向 8 位 I/O 口，每一位可通过掩模选项设置成唤醒功能（Wake-up）的输入脚，或要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—PB0~PB7

双向 8 位 I/O 口，每一位可通过掩模选项选择要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—PC0~PC7

双向 8 位 I/O 口，每一位可通过掩模选项选择要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—PD0~PD7

双向 8 位 I/O 口，每一位可通过掩模选项选择要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—PE0~PE7

双向 8 位 I/O 口，每一位可通过掩模选项选择要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—PF0~PF7

双向 8 位 I/O 口，每一位可通过掩模选项选择要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—PG0~PG7

双向 8 位 I/O 口，每一位可通过掩模选项选择要不要内置的上拉电阻。每一位可用软件指令设置为 CMOS 输出或施密特触发输入。

—RESB（reset）

施密特触发复位输入脚，低电压有效。

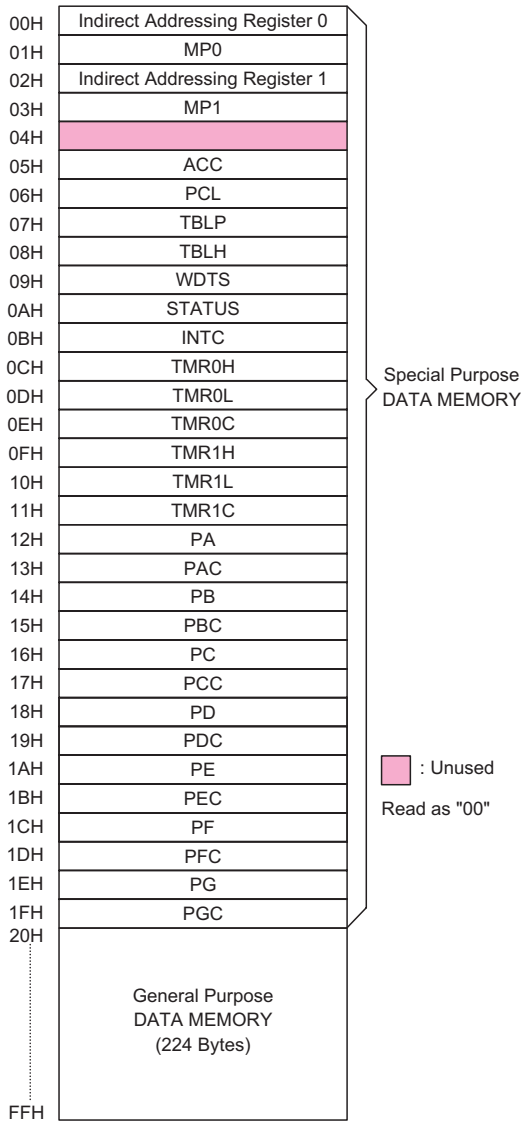
—V_{DD}

正电源

—VSS
负电源

数据存储器（Data Memory）

下表为数据存储器对应情况，从 00H~1FH 为特殊寄存器，20H~FFH 通用数据寄存器。有些位置没有定义，是预留给将来扩充用的，用户不可以去存取，因为没有意义，而且可能造成程序的误操作。



以下为特殊寄存器的说明：

间接寻址寄存器（Indirect Addressing Register）

R0=[00H]和 R1=[02H]为间接寻址寄存器,实际上是用 MP0 和 MP1 作指针来指定存取的地址,下面举例说明:

```
MOV     A,   25H
MOV     MP0,     A          ;MP0 指向数据存储器 25H 的位置
```

```
MOV     A,   10H
MOV     [00H],    A          ;将 ACC=10H 的值放到 MP0 所指的位置
                             ;也就是说[00H]被 MP0 间接定址到 25H 位置
                             ;25H 位置的内容将是 10H
                             ;也可写为 MOV R0，A
```

ACC 的值会被放到 MP0 所指的地址中，反之如果读[00H]的值，将读出 MP0 所指向的地址中的内容。[02H]则是与 MP1 相对应的。所以对[00H]和[02H]进行读写完全是参考 MP0 和 MP1 指针的内容完成的。

RAM 指针 MP0，MP1 (Memory Pointer Register)

RAM 指针是一个 8 位寄存器，依据使用者所选择单片机型号的不同而分为 7 位有效或 8 个位有效。如果只有 7 位元有效的芯片，其第 8 位是没有定义的，而且读取时，第 8 位永远为 1，读者在写程序时，请务必小心。RAM 指针与间接寻址寄存器来使用，会是一个强力的组合，若再配合查表指令，可说是如虎添翼。

累加器 (Accumulator)

累加器简写为 ACC，对应到数据存储器的 05H，一般是用来暂存立即值或 ALU 运算的结果。改变数据存储器的内容也需要通过 ACC，而不能直接由一个地址传送到另一个地址。

程序计数器低字节寄存器 (PCL)

PCL 对应到数据存储器的 06H，存放着程序计数器低字节的值，是个可读写的寄存器，改变 PCL 的值可使程序产生短跳跃，范围在 256 个地址内。

数据表指针寄存器 (TBLP)

数据表指针寄存器 TBLP 对应到数据存储器的 07H，当读取程序区的数据时，可用 TBLP 寄存器的内容作为指针。再与查表指令 TABRDC， TABRDL 一起使用。

数据表高字节寄存器 (TBLH)

注意：此寄存器是只读寄存器，当用查表指令读取程序区数据时，会将所读到数据的高字节放入此寄存器中，该寄存器对应数据存储器的 08H。

看门狗计时预除寄存器 (WDTS)

芯片有一内置的看门狗定时器，大约是每 20ms 周期时间到 (time-out) 一次，若是想加长周期时间可利用 WDTS 寄存器，对应到数据存储器的 09H 。WS0、WS1、WS2 分别为 WDTS 的 bit0、 bit1、bit2，把数据写入这 3 个位能得到更长的周期时间。

状态寄存器 (STATUS)

状态寄存器包括程序执行时的各种状态，用户可利用状态寄存器的各种标志来改变程序执行的流程，状态寄存器对应到数据存储器的 0AH。

下表为状态寄存器各标志的说明：

符号	位元	功　　能
C	0	只有在加法运算产生进位或者是减法运算不产生借位时，会使得进位标志置为"1"。此外在有带进位标志作循环移位的运算时，也会影响进位标志。
AC	1	辅助进位标志只有在低半字节(bit0~bit3)和高半字节(bit4~bit7)之间，作加法运算产生进位或者是减法运算时产生借位时，都会使得辅助进位标志置为"1"，其它情形下皆置为"0"。
Z	2	在作算术或者是逻辑运算後其结果为零时，该位置为"1"，其它情形下皆设定为"0"。
OV	3	溢出标志，在执行带符号运算时，如运算结果超过运算元的容量，则设定为"1"，简单的判断就是 bit6 进位到 bit7 和 bit7 进位到进位标志 C 之两个位元作逻辑"互斥或"运算结果，结果为"0"就无溢出，结果为"1"就有溢出。
PD	4	省电标志，当执行省电模式指令 HALT 时被设定为"1"，当电源启动或者执行"CLR WDT"指令两者其中之一时会被清"0"。
TO	5	时间到标志，当看门狗定时器计数到时(time-out)会置为"1"，当电源启动或者执行"CLR WDT"指令或者执行省电模式指令 HALT 时会被清"0"。
	6，7	6，7未定义，读出为"0"。

状态寄存器各标志表

中断控制寄存器 (INTC)

中断控制寄存器控制着整个芯片的中断是否允许，并且包含中断请求标志。用户可通过软件的设置来控制整个中断的特性。定义在数据存储器的 0BH 位置。下表为中断控制寄存器各位功能表：

符号	位元	功能
EMI	0	主中断控制，"1"为允许，"0"为不允许
EEI	1	外部中断控制，"1"为允许，"0"不允许
ET0I	2	计时/计数　0中断控制，"1"为允许，"0"为不允许
ET1I	3	计时/计数器 1 中断控制，"1"为允许，"0"为不允许
EIF	4	外部中断请　标志
T0F	5	计时/计数　0中断请　标志
T1F	6	计时/计数　1中断请　标志
	7	未定义，读出为"0"

中断控制寄存器各位元功能表

定时/计数寄存器 0 (TMR0H，TMR0L)

计数/定时 0 寄存器分别由两个寄存器所组成，一个是 TMR0H，一个是 TMR0L，分别为八位的寄存器，共同组成 16 位的定时/计数寄存器 0，存放着向上计数的值。用户可通过读出寄存器内的值，得知目前计数的位置。TMR0H 定义在 0CH，TMR0L 定义在 0DH 。

定时/计数寄存器 1 (TMR1H，TMR1L)

定时/计数寄存器 1 分别由两个寄存器组成，一个是 TMR1H，一个是 TMR1L，分别为八位的寄存器，共同组成 16 位元的定时/计数寄存器 1，存放着向上计数的值。可通过读取寄存器内的值，得知目前计数的位置。TMR1H 定义在 0FH，TMR1L 定义在 10H 。

定时/计数控制寄存器 0 (TMR0C)

计数/计时控制寄存器 0 定义定时/计数的操作模式，并且控制是否允许计数和触发的方式。定义在 0EH ，用户可通过软件来设置定时/计数的触发方式与模式。下表为各位的定义说明:

符号	位元	功能
	0~2	未定义，读出为"0"
TE	3	定义定时/计数器0触发边缘 "0"上升沿触发，"1"下降沿触发
TON	4	定时/计数器 0，"0"允许，"1"不允许
	5	未定义，读出为"0"
TM0 TM1	6 7	定义动作模式: 01=计数模式（外部脉冲） 10=计时模式（内部振荡） 11=脉宽测量模式 00=未使用

定时/计数起器 0 控制寄存器各位表

定时/计数器 1 控制寄存器 (TMR1C)

定时/计数器 1 控制寄存器定义着定时/计数的操作模式，并且控制是否允许计数和触发的方式。定义在 11H，用户可通过软件来设置定时/计数的触发方式与模式。下表为各位的定义说明：

符号	位元	功能
	0~2	未定义，读出为"0"
TE	3	定义定时/计数 0触发边缘 "0"上升沿触发，"1"下降沿触发
TON	4	定时/计数器 0，"0"允许，"1"不允许
	5	未定义，读出为"0"
TM0 TM1	6 7	定义动作模式: 01=计数模式（外部脉冲） 10=计时模式（内部振荡） 11=脉宽测量模式 00=未使用

计数/计时 1 控制寄存器各位元表

端口 A~端口 G 寄存器 (PA~PG ports)

PA~PG 分别对应数据存储器的 [12H]，[14H]，[16H]，[18H]，[1AH]，[1CH] 和 [1EH]。 所有 ports 都是双向的，也就是说可用软件设置成输入或输出模式。当设置成输入模式时，电路的结构是不带锁存的，外部数据在 T2 的上升沿被读入单片机。当设置成输出模式时，电路结构是有锁存的，所以输出状态会一直保持到下次改写时才发生改变。

端口 A~端口 G 控制寄存器 (PAC~PGC)

PAC~PGC 分别控制相应的端口。也就是说，用户要用软件设置端口控制寄存器，使相应的端口工作于输入或输出模式。PAC~PGC 分别对应到 [13H]，[15H]，[17H]，[19H]，[1BH]，[1DH]，[1FH]。

数据寄存区 (General Purpose Data Memory)

这个区域就是俗称的 RAM，使用者可利用该区来存放必要的数据。不同型号的 IC 会有不同的 RAM 空间， 如下表所示：

HT48C10	HT48C30	HT48C50	HT48C70
40H	20H	60H	20H
:	:	:	:
:	:	:	:
7FH	7FH	FFH	FFH

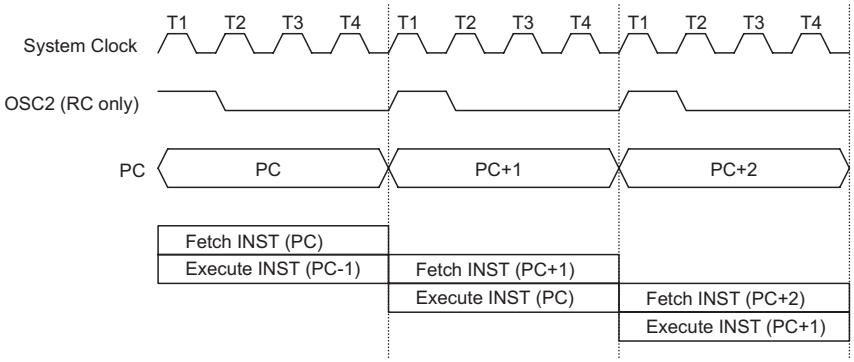
第 二 章
系统结构分析

2

时序分析

系统时钟由石英晶体振荡器或 RC 振荡器产生，系统内部将此频率分为四个不重叠的时钟（一般称为 T 状态，分别为 T1、T2、T3、T4），一个指令周期包含了四个 T 状态，即一个指令周期为四个系统时钟周期。

指令的读取与执行是以流水线方式来进行的。这种方式允许在一个指令周期进行读取指令操作，而在下一个指令周期里进行解码与执指操作。这种流水线方式能在一个指令周期里有效地执行一个指令。但是，如果涉及到的指令要改变程序计数器（如 JMP，CALL 等），就需要花两个指令周期来完成这一条指令。



操作流程

程序计数器 (Program Counter)

程序计数器就是程序码的指标器，控制了整个程序的流程。单片机从程序计数器所指向的程序记忆区里抓取指令来执行，取得指令码后，程序计数器会自动加一指向下一个指令码的地址。但是如果是执行跳跃、条件跳跃、子程序调用、中断等情况，则程序计数器会被载入相对应的地址，而非下一地址。举例而言，如果执行条件跳跃指令，当符合条件时，则在执行此条指令时被抓取的指令码将被抛弃，同时插入一个假的指令周期 (dummy cycle)，程序计数器才会指向正确的程序码地址，因此需要两个指令周期。反之，程序计数器会指向下一个指令码的地址。程序计数器的低位元组 (PCL) 是一个可读写的暂存器 (06H)，如果写入一个值将会产生一个短跳跃动作，这个短跳跃范围是 256 个地址。同样的在此情况下也会插入一个假的指令周期 (dummy cycle)。

下面举个实例说明 PCL 的运作行为：

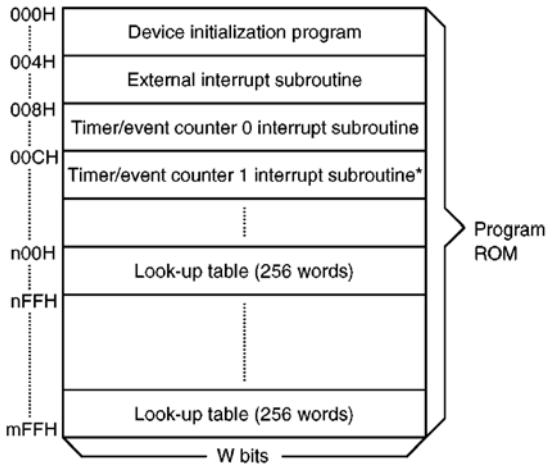
```
ORG 34H
```

```
MOV A,25H      ;
MOV PCL,A      ;执行完此指令后程序将会跳至 25H 的地址
:
```

相对於上述程序只能往后跳 54 个地址 (00H~35H)，而往前跳可跳 203 个地址 (35H~0FFH)。但是请注意 PCL 并不能做跨页的跳跃。

另外须提醒读者一点的是在集成开发环境 (HT-IDE) 中看到的 PCL 值是指向下一个指令的地址，而反白的那行为将要执行的指令，这是 pipeline 的关系。

程序存储器(Program Memory)



程序存储器（ROM）用来存放要执行的指令的代码，还包括数据，表格和中断向量等。程序存储器的大小根据所选定的单片机而定： HT48C10 为 1024×14 位；HT48C30 为 2048×14 位；HT48C50 为 4096×15 位；HT48C70 为 8192×16 位。程序存储器的空间都可以由程序计数器（PC）或表格指针（Table Pointer）来寻址。

以下所列出的程序存储器的地址是专为这个系列单片机的特殊用途而保留的，其中 00CH 地址只供 HT48C50 和 HT48C70 使用，其余所述地址均为这个系列单片机皆可使用的。

- 地址 000H

  此区域保留给程序初始化之用。当系统复位时，程序会从 000H 地址开始执行。

- 地址 004H

  此区域保留给外部中断服务程序使用。若中断开放且堆栈又未满，则一旦 INT 端被正确的电平触发，就能产生中断，程序会从 004H 地址开始执行外部中断服务程序。

- 地址 008H

  此区域保留给定时/计数器 0 中断服务程序使用。当中断开放且堆栈又未满，则一旦定时/计数器 0 发生溢出时，就能产生中断，程序将转至 008H 地址开始执行中断服务程序。

- 地址 00CH

  此区域保留给定时/计数器 1 中断服务程序使用。当中断开放且堆栈又未满，则一旦定时/计数器 1 发生溢出时，就能产生中断，程序将转至 00CH 地址开始执行中断服务程序。

一般程序写法：

```
            ORG 0H
            JMP START
            ORG 4H
            JMP INTERRUPT
            ORG 8H
            JMP TIMER0_INTERRUPT
            ORG 0CH
            JMP TIMER1_INTERRUPT
START:
            :
            :
INTERRUPT:
```

```
            RETI
TIMER0_INTERRUPT:
            RETI
TIMER1_INTERRIPT:
            RETI
```

上面的程序告诉我们当中断子程序超过 4 个地址时，最好将中断子程序定到别的程序空间去，以避免中断发生时，执行到错误的程序。

查表 (Table Look-up)

ROM 内的任何地址都可被用来作为查表地址使用。用户可用两条查表指令 TABRDC [m] 与 TABRDL [m] 来读取程序存储区的内容。TABRDC [m]是查表当前页的数据 [1 页=256 个字 (word)]。TABRDL [m]是查表最后一页的数据。一页相当于 256 个字，[m] 为数据被存入的地址。在执行 TABRDC [m]指令（或 TABRDL [m] 指令）后，将会传送当前页（或最后一页）上的一个字的低位字节到[m]，而这个字的高位字节传送到 TBLH（08H），即只有表格中的低位字节被定义到目标地址中。TBLH 为只读寄存器。而表格指针（TBLP；07H）是可以读写的寄存器，用来指明表格地址。所以在作查表动作时，须先将数据表格的地址放到 TBLP 中。

下面用例子来说明：

<例 1>

```
MOV     A,020H
MOV     TBLP,A      ;table pointer 指向 20H 地址
TABRDC [30H]        ;将 table point 所指地址的值读到
                    ; 地址 30H 中
                    ; 以此例会将 34H 读入地址 30H
                    ; 并将 12H 放入 TBLH
ORG         20H
DC      1234H,5678H,8765H.........
```

<例 2>

假设选用 HT48C70 单片机其程序存储器有 8K，最后一个 PAGE 的地址为 1F00H~1FFFH

```
MOV     A,020H
MOV     TBLP,A      ;table pointer 指向 20H 地址
TABRDL [30H]        ;将最后一页且 table point 所指地址的值
                    ;读到 30H 的地址
                    ;以此例会将 34H 读入 30H 的地址
                    ;并将 12H 放入 TBLH
ORG     1F20H
DC      1234H,5678H,8765H.........
```

堆栈寄存器—STACK

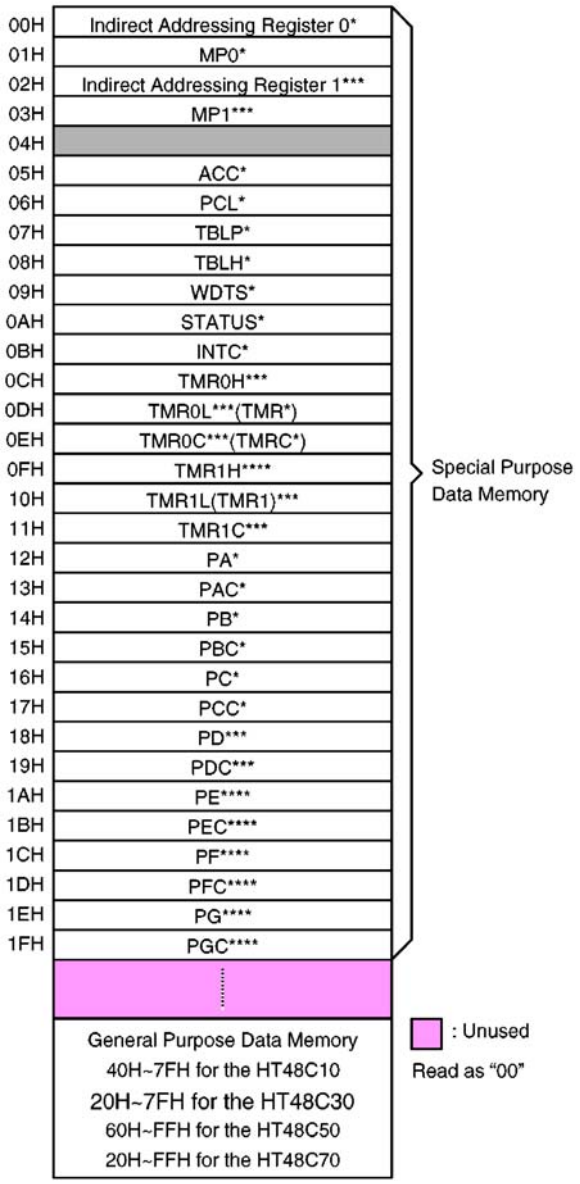
堆栈寄存器（STACK）是一个用来保存 PC 值的特殊存储单元。它既不是数据存储器的一部分，也不是程序存储器的一部分，存放程序计数器的值。任何一级堆栈的使用是由堆栈指针（SP）来确定。堆栈指针也不能读出与写入。一旦调用子程序或响应中断时，程序计数器（PC）的值会被压入堆栈，在子程序调用结束或中断响应结束时，堆栈将原先压入堆栈的内容弹出，重新装入程序计数器（PC）中。这里必须强调的是，堆栈寄存器在处理调用子程序和响应中断这两个情况时有所不同。

如果堆栈已满，并且发生了不可屏蔽的中断，那么中断请求标志将被记录下来，但是并不会执行该中断，

直到堆栈有空时，中断服务才会被响应。这个功能防止堆栈溢出，使得程序员易于使用这种结构。同样地，如果堆栈已满时调用子程序（CALL），那么堆栈会产生溢出，从而丢失首先进入堆栈的内容，程序会因此而无法控制。只有最后的返回地址会被保留着所以用户在使用堆栈时要非常小心。请注意：当发生中断或调用子程序时，只有 PC 的值将被推入堆栈保存，用户必须自己保存在程序中可能被破坏的寄存器。

数据存储器—RAM

数据存储器可分成两个功能组，即，特殊功能寄存器和通用数据存储器。这两个功能组的绝大部分单元可以读写，而某些单元只能读出，不能写入。其中，特殊功能寄存器包括程序计数器（PC）低位字节寄存器（PCL；06H），累加器（ACC；05H），表格指针寄存器（TBLP；07H），表格高位字节寄存器（TBLH；08H），状态寄存器（STATUS；0AH），中断控制寄存器（INTC；0BH），看门狗定时器设置寄存器（WDTS；09H），间接寻址寄存器（00H），存储器指针寄存器（MP；01H），定时/计数器（TMR；0DH），定时/计数器控制寄存器（TMRC；0EH），输入/输出寄存器（PA；12H，PB；14H，PC；16H）和输入/输出控制寄存器（PAC；13H，PBC；15H，PCC；17H）。这四种单片机在 20H 以前的剩余单元都被保留为将来进一步扩展使用。读取这些被保留单元的值，都将返回 00H 的值。所有的 RAM 区单元都能直接执行算术，逻辑，递增，递减和移位等运算。除了某些特殊的位以外，RAM 中的每一位都可以由 SET[m].i 和 CLR[m].i 指令来置位和复位。这些 RAM 都可通过存储器指针寄存器间接寻址来存取。



间接寻址寄存器 (Indirect Addressing Register)

地址 00H 和 02H 是间接寻址寄存器，没有实际的物理结构。直接读取 00H 或 02H 单元，将会返回 00H，而直接地写入 00H 或 02H 单元，则不会产生任何结果。

任何对间接寻址寄存器的读写动作是依据存储器指针寄存器所指的地址操作，也就是对间接寻址寄存器作读写的目的地为指针寄存器所指的地址。请看以下说明：

```
MOV     A,20H
MOV     MP0,A           ;memory pointer 0 先给初值 20H
MOV     A,55H           ;Acc=55H
MOV     [00H],A         ;将 55H 写到 20H 地址内

MOV     A,20H
MOV     MP1,A           ;memory pointer 1 先给初值 20H
```

```
MOV     A,0AAH          ;Acc=0AAH
MOV     [02H],A         ;将 0AAH 写到 20H 地址内
```

由上面的说明我们可知 00H 是参考 MP0，而 02H 则是参考 MP1，读者要知道这点差异。

算术逻辑单元(ALU)

算术逻辑单元（ALU）为执行八位算术及逻辑运算的电路。

➔ **ALU 提供下列功能：**

- 算术运算（ADD，ADC，SUB，SBC，DAA）
- 逻辑运算（AND，OR，XOR，CPL）
- 移位（PL，RR，RLC，RRC）
- 递增及递减（INC，DEC）
- 进位及无进位减法
- 分支判断（SZ，SNZ，SIZ，SDZ 等）

ALU 不仅可以储存数据运算的结果，还可以改变状态寄存器的值。

状态寄存器—STATUS

状态寄存器（0AH）由零标志位（Z），进位标志位（C），辅助进位标志位（AC），溢出标志位（OV），掉电标志位（PD），看门狗定时器溢出标志位（TO）组成。

除了 TO 和 PD 以外，状态寄存器中的位都可用指令改变，这种情况与其它寄存器一样。任何写到状态寄存器的数据不会改变 TO 或 PD 标志位。但是与状态寄存器有关的操作会导致状态寄存器的改变。系统上电，看门狗定时器溢出，执行 HALT 指令，或清除看门狗定时器都能改变 TO 和 PD。而 Z，OV，AC 和 C 标志位都反映了当前的运算状态。进入中断程序或执行子程序调用时，状态寄存器的内容不会自动压入堆栈。如果状态寄存器的内容是重要的，而且子程序会改变状态寄存器的内容，那么程序员必须事先将其保存好，以免被破坏。

状态寄存器

符号	位	功　　能
C	0	如果在加法运算中结果产生了进位，或在减法运算中结果不发生借位，那么 C 被置位；反之，C 被清除。它也可被一个带进位循环移位指令而影响。
AC	1	在加法运算中低四位产生了向高四位进位，或减法运算中低四位不发生从高四位借位，AC 被置位；反之，AC 被清除。
Z	2	算术运算或逻辑运算的结果为零则 Z 被置位；反之，Z 被清除。
OV	3	如果运算结果向最高位进位，但最高位并不产生进位输出，或那么 OV 被置位；反之，OV 被清除。
PD	4	系统上电或执行了 CLR WDT 指令，PD 被清除。执行 HALT 指令 PD 被置位。
TO	5	系统上电或执行了 CLR WDT 指令，或执行 HALT 指令，TO 被清除。WTD 定时溢出，TO 被置位。
—	6	未定义，读出为零
—	7	未定义，读出为零

状态寄存器的标志位

进位标志 (C)

算术运算或带进位旋转指令会影响进位标志。

<例 1> 78H+B6H=?

```
    01111000B
+   10110110B
    ----------
C=1 00101110B      结果:2EH 且 C=1
```

<例 2> 38H-56H=?

```
    00111000B
-   01010110B
    ----------
C=0 11100010B      结果:E2H 且 C=0
```

辅助进位标志 (AC)

算术运算会影响此标志。其原则与进位标志相同，但是根据最低 4 位运算的结果。

溢出标志 (OV)

溢出标志通常用于判断带符号运算是否发生错误。一个字节有 8 个位，通常以第 7 位作为符号位，其余 7 个位作为数值。若两个带符号的数相加，结果超出 +127~-128 范围，则溢出标志 OV=1。

<例> 75H 01110101B

```
+   30H     +   00110000B
    --------    ----------
    A5H         10100101B=A5H
```

两正数相加结果为负值所以 OV=1

零标志(Z)、进位标志(C)、辅助进位标志(AC)、溢出标志(OV)反应最近一次执行结果的状态。要提醒的一点是:当进入中断服务程序或调用子程序，这些状态标志并不会被推入堆栈保存。如果这些标志是有用的，且子程序可能破坏这些标志的状态，那么用户在写程序时必须留意，避免有用的标志被破坏掉。

省电标志 (PD)

当执行 HALT 指令时，PD 标志会被设定。执行 CLR WDT 指令或者芯片重新上电 (Power-On) 时，PD 标志将被清除。用户可利用 PD 标志来判断程序的逻辑流程。

时间到标志 (TO)

当看门狗时间到时 TO 标志会被设定。执行 CLR WDT 或 HALT 指令或者芯片重新上电源 (Power-On) 时，TO 标志会被清除。用户可利用 TO 标志来判断程序的逻辑流程。

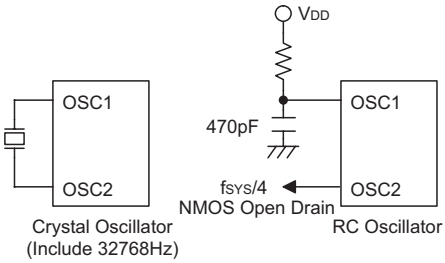
下表为 PD 和 TO 标志发生情况表：

发生情况	TO	PD
芯片重新上电（power-up）	0	0
看门狗时间到	1	×
HALT 后复位唤醒	0	1
CLR WDT 指令	0	0

×：不影响

振荡器配置 (Oscillator Configuration)

八位元单片机都有内置振荡电路，用户只须在外部接石英晶体 (Crystal) 或电阻电容即可产生系统所需的振荡源。在 HT48CX0 系列都有两组振荡电路，都是用来当作系统时钟。一组是 RC 振荡电路，一组是晶体振荡电路。用户在送程序码给 Holtek 开光罩时，必须指定所使用的振荡模式。



使用 RC 模式振荡时，必须在 OSC1 和 VDD 之间加一电阻，电阻值的范围在 47kΩ 到 1MΩ 之间，又因为 OSC2 在 RC 模式下为漏极输出，所以需外加一个约 10kΩ~50kΩ 间的上拉电阻。OSC2 会有一系统时钟 4 分频的信号输出，可用来同步外部逻辑电路。RC 模式提供一项低价有效的振荡方式，然而其时钟却会随著 VDD 、温度和芯片制造过程而有所不同，因此对于时间要求精确的产品上并不适合使用来做为系统时钟源。

使用晶体振荡模式时，必须在 OSC1 和 OSC2 间加一石英晶体 (Crystal) 来提供振荡电路反馈和相移。也可以使用谐振器 (resonator)，但是必须在 OSC1 和 OSC2 各加一颗陶瓷电容。

注意在使用发展系统 HT-ICE 时，我们并不建议用户使用标准振荡方式（内部、外部）以外的振荡源，这是因为 HT-ICE 的系统时钟并非介面卡上的 OSC1、OSC2 接脚，用户用自己的振荡源输入给 HT-ICE，其结果是没有意义的。另外，在介面卡上的 OSC1、OSC2 接脚是空接的，并不会去影响系统.。

假如你真的想使用您自己的振荡源，可以将介面卡上的 JP1 跳线移开，再把您的振荡源输出信号输入到 JP1 的中间接脚，那将会以外部振荡源输入的方式，输入到 HT-ICE 上，即使如此，我们仍强烈建议用户使用内部振荡源。

省电模式 (Power Down Operation-HALT)

➔ **HALT 指令会停止系統时钟，以降低功耗。执行 HALT 指令会产生以下的情况**

- 关闭系统振荡器，但看门狗定时器振荡器仍会继续运行（如果选择的是看门狗定时器 RC 振荡器，）
- 不改变 RAM 和寄存器的内容
- 清除并重新计数看门狗定时器（如果看门狗定时器的时钟来源为看门狗定时器 RC 振荡器）
- 所有输入/输出口都维持其原有状态
- 置位 PD 标志并且清除 TO 标志

➔ **有四种方式可以离开 HALT 模式**

- 执行外部复位
- 外部中断
- PA 口下降沿的信号
- 看门狗定时器溢出

其中，外部复位会造成系统初始化（System Initialization），看门狗定时器溢出则会发生热复位。一旦检测 TO 和 PD 标志之后，即可了解系统复位的原因。欲清除 PD 标志，可通过系统通电或是执行 CLR WDT 指令来达成。而若要置位 PD 标志，即可执行 HALT 指令。如果发生看门狗定时器超时，不仅会置位 TO 标志，还会产生唤醒，用来复位程序计数器和状态指针，而其它的电路则继续维持其原有的状态。

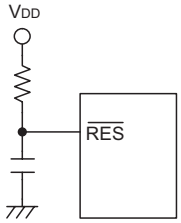
PA 口唤醒和中断这两种方式可以视为正常运行的继续，PA 口上每个位在掩膜制造时都可以单独选择，用来唤醒系统。一旦从输入/输出口启动唤醒之后，程序即从下一条指令重新开始运行。但如果是从中断被唤醒的话，此时可能会发生两种情况。如果堆栈已用满，无论中断是否被禁止，程序都会从下一条指令重新开始运行。但如果该中断被允许，但堆栈尚未用满，则会产生一般中断响应。如果

在进入 HALT 模式之前，中断请求标志位被置 1，与唤醒方式相关的中断被禁止。

若发生唤醒事件，必需额外花费 1024tSYS（系统时钟周期）的时间，才能重新正常运行。换言之，唤醒之后即会插入一个等待周期。如果唤醒是由于中断响应的话，实际中断子程序的执行会延迟大约一个以上的周期。如果唤醒事件导致下一条指令执行，一旦插入的等待周期执行完成之后，会立即执行该指令。另外，为减少电源损耗，在进入省电模式之前，应小心处理所有的输入/输出管脚。

初始复位(Reset)

总共有三种方法会产生初始复位，如下所示：



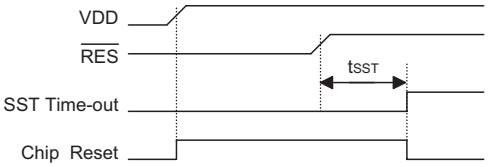
- 正常操作时由$\overline{RES}$引脚发生复位
- 在暂停模式由$\overline{RES}$引脚发生复位
- 正常操作时由看门狗定时器超时发生复位

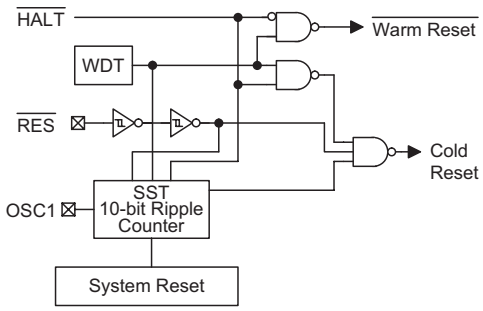
暂停模式中的看门狗定时器超时与其它系统复位状况不同，因为看门狗定时器超时会执行热复位，用来重新设置程序计数器和状态标志，并保持其它电路原有的状态。少数寄存器在其它复位状态皆不会改变，大部分寄存器一旦符合复位条件时，会复位成初始的状态。通过检测 PD 和 TO 这两个标志，程序即可区别出各种不同的系统复位。

TO	PD	复位条件
0	0	电源通电复位
u	u	正常运作时由$\overline{RES}$发生复位
0	1	由$\overline{RES}$唤醒暂停模式
1	u	正常运作时发生看门狗定时器超时
1	1	由看门狗定时器唤醒暂停模式

为了保证系统振荡器一定能启动并运行稳定，系统启动定时器（SST）提供了一个额外的延迟（Delay），这延迟在通电之后或是从暂停模式唤醒时，会延迟 1024 个系统时钟脉冲的时间。

当系统通电时，系统会多延迟系统启动定时器这样一个复位时间，但是如果系统是因为$\overline{RES}$脚而发生复位，则不产生这一延时。另外由暂停模式唤醒时，也会启动系统启动定时器之延迟时间，以稳定系统振荡器。





有关功能的系统复位状态如下所示：

程序计数器（PC）	000H
中断	禁止
预分频器	清除
看门狗定时器	清除、复位后定时器开始计数
定时/计数器	关闭
输入/输出口	输入模式
堆栈指针	指向堆栈的顶端

有关寄存器的状态如下

寄存器	复位(通电)	WDT 超时 (正常运作)	$\overline{\text{RES}}$复位 (正常运作)	$\overline{\text{RES}}$复位 (暂停模式)	WDT 超时 (暂停模式)
TMRAH	×××× ××××	Uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMRAL	×××× ××××	Uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMRC	0000 1---	0000 1---	0000 1---	0000 1---	uuuu u---
TMRBH	×××× ××××	Uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMRBL	×××× ××××	Uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
ADCR	1××× –00	1××× --00	1××× --00	1××× --00	uuuu --uu
Program Counter	000H	000H	000H	000H	000H*
MP0	×××× ××××	Uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	×××× ××××	Uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
ACC	×××× ××××	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLP	×××× ××××	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	×××× ××××	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
STATUS	--00 ××××	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
INTC1	---0 ---0	---0 ---0	---0 ---0	---0 ---0	---u ---u
RTCC	--×0 0111	--×0 0111	--×0 0111	--×0 0111	--uu uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu

附注：　“*”表示“热复位”　　“u”表示“不变”　　“×”表示“未知”

第 三 章

中断、看门狗计时器、定时/计数器

3

中断 (Interrupt)

HT48C70 提供了一组外部中断和两组定时/计数中断源。中断响应时，单片机会将目前程序计数器的值推入堆栈，而跳至对应的中断向量执行程序。其中断向量表如下所示：

NO	中断源	优先级	中断
A	外部中断	1	04H
B	定时/计数器 0 溢出	2	08H
*C	定时/计数器 1 溢出	3	0CH

对每一中断源用户可用中断控制寄存器 (0BH) 来设定使其使能或除能，只要将对应的位元设定或清除即可。在中断控制寄存器里，位元 0 (EMI) 是控制所有中断源是否有效的位元。当 EMI=0 时所有中断被除能，也就是说不会有中断响应。当 EMI=1 时，各个中断源是否使能，还需再看其对应的使能位元是否有被使能。

下表为中断控制寄存器各位元说明：

	位	符号	功　　能
INTC (0BH)	0	EMI	总中断控制位 1=允许，　0=禁止
	1	EEI	外部中断控制位 1=允许，　0=禁止
	2	ET0I	定时/计数器 0 中断控制位，1=允许，　0=禁止
	3	ET1I	定时/计数器 1 中断控制位，（只有 HT48C50/HT48C70）1=允许，　0=禁止
	4	EIF	外部中断请求标志位 1=有效，0=无效
	5	T0F	定时/计数器 0 中断请求位 1=有效，0=无效
	6	T1F	定时/计数器 1 中断请求位，（只有 HT48C50/HT48C70）1=有效，0=无效
	7	—	未使用位，读出为零

中断控制寄存器 (INTC;0BH) 包含了用来使能或除能中断功能的中断控制位元和中断请求标志。当中断服务程序正在执行时，硬件电路会清除 EMI 位元，不让其它中断要求发生，但是中断请求标志会被设定，等到中断服务程序执行完毕，才允许下一个中断请求进入中断服务程序，这是为了避免中断发生嵌套 (INTERRUPT NESTING)。如果中断必须即时得到响应，用户可以在服务程序进入处立刻设定 EMI 及对应的中断使能控制位，即可处理中断嵌套的问题。另外一点要注意的是，如果堆栈满了，即使中断使能，

中断也不会发生，必须等到堆栈有空。

INT 脚接收到一个下降沿，会设定中断请求标志，如果中断使能且堆栈未满，就会发生中断，程序会跳至 004H 的中断入口，并且硬件会清除中断请求标志 EIF 和 EMI，使其它中断在此刻不能发生，需要等到服务程序执行完毕。

定时/计数器 0 发生溢出会产生定时/计数中断请求标志 T0F，如果中断使能且堆栈未满，就会产生中断，程序会跳至 008H 的中断入口，并且硬件会清除中断请求标志 T0F 和 EMI，使其它中断在此刻不能发生，需要等到服务程序执行完毕。一旦中断请求标志被设定，将会保留在中断控制寄存器中，除非执行中断服务程序才会清除标志，或者也可用软件指令的方式将标志清除。

定时/计数器 1 发生溢出会产生定时/计数中断要求标志 T1F ，如果中断是使能且堆栈未满，会触发中断行为的发生，程序会跳至 00CH 的中断服务位址。并且硬件会清除中断要求标志 T1F 和 EMI，使其它中断行为在此刻不能发生，需等到服务程序执行完毕。一旦中断要求标志被设定，将会持续保留在中断控制寄存器中，除非中断服务程序被执行才会清除标志，或者也可用软件指令的方式将标志清除。

下面说明一个中断应该设定的步骤：

1: 先设定中断控制寄存器的 "EMI" 位元为 1

2: 再设定中断控制寄存器里对应中断之控制位元为 1

3: 对应之中断向量位址写入中断后欲处理之指令

4: 中断处理程序最后记得用 "RETI" 指令

<例>

```
        org 00H
        jmp start
        org 04H
        jmp interrupt
        org 20h
start:
        set emi         ← 设定 EMI 为 1
        set eei         ← 设定外部中断使能
interrupt
            :
            :
        reti            ← 记得用 reti 让程序跳回原流程
```

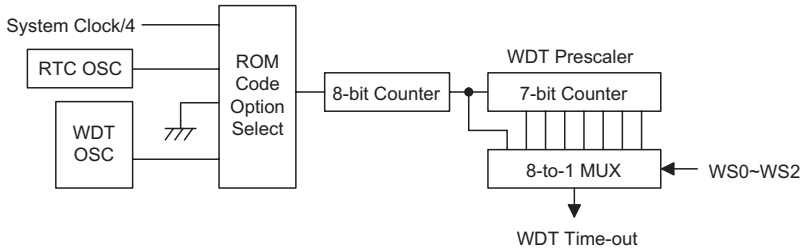
定时/计数中断方式也是类似设定模式，在此必须再补充一点，就是 "RETI" 指令，"RETI" 指令与 "RET" 指令唯一的差别就是 "RETI" 执行后会将 "EMI" 位元设定为 1。当处理中断时，"EMI" 位是硬件自动清除的，以防止其它中断在此刻发生，当中断程序处理完毕时必须靠 "RETI"指令自动将 "EMI" 位元设定使能，这样才能保证下一次中断的发生。

有个建议提供给用户参考：在中断服务程序中，避免使用 CALL 指令。因为中断的发生常常是不可预期的，而且可能要马上处理，如果堆栈只剩一层，则进入中断服务程序后，堆栈就满了，这时若再用 Call 指令，就会发生堆栈溢出，导致程序流程错误，所以用户要小心。

看门狗计时器 (Watchdog Timer)

看门狗计时器的时钟来源有两个方式，一个是从内置的看门狗振荡器引出，一个是从系统时钟除以 4 引出，只能利用掩模选项 (mask option) 选择时钟来源。看门狗主要用来避免程序运作失常及程序跳进一死循环造成不可预期之结果。看门狗可用选项来使能或除能 (disable)，如果在除能状态，所有关于 WDT 的

指令都是没有作用的。



用户先由内置看门狗计时振荡器分析 time-out 周期：(如果 WDT 来源是从内置式看门狗振荡器引出时)，内置看门狗计时振荡器本身为一 RC 结构振荡器，周期大约是 78μs，它会先经过一个除 8 级的电路 (第一级约 156ms) 而得到大约 20ms (第八级) 时间到 (time-out) 的周期，这个周期会随 VDD、温度和晶片的制造改变。如果要得到更长的 time-out 周期，可以利用看门狗预除寄存器来达成。举例来说，如果 WS2、WS1、WS0 的值都为 1，则 time-out 的周期将被拉长 128 倍，那么用户将可得到一个大约 2.6 秒的 WDT time-out 周期。如果 WDT 除能，WDT 丧失保护系统的功能，此时只能靠外部 reset 来重新启动电路。看门狗预置寄存器的位元 3 ~7 保留给用户来定义状态字。

如果来源是选用系统时钟 4 分频，则其周期就不是 78ms，而要取决于当时的系统时钟。譬如说系统时钟为 4MHz，则其原始周期就是 1ms，再经过除 8 级就得到 256ms，再由看门狗预除寄存器的值决定最后 time-out 的时间。

WDT time-out 会使得晶片复位 (chip reset)，并设定 TO 状态标志，如果是在 HALT 模式 WDT time-out 只会引起热复位 (warm reset)，即 PC 和 SP 等于零，其他不变。

有三种方式可以清除 WDT 的计时

第一种是加一低电平到 Reset 脚。

第二种是用软件指令的方式。

第三种是 HALT 指令。

第一种相当于晶片的复位 (reset)，会使 WDT 重新计数。第二种软件指令的方式有两个选项，一个是 CLR WDT，另一个是 CLR WDT1、CLR WDT2，只能选用其中一种方式。选用 CLR WDT 时，只要执行一次此指令就会清除一次 WDT 计时器，选用 CLR WDT1、CLR WDT2，则要执行一对这样的指令才会清一次 WDT 计时器。如果程序跳入一死循环而此循环又有一 CLR WDT 指令，则此程序就会在此跳不出来，如果此时是用两条指令的方式就可避免死循环。第三种是执行 HALT 指令时会清掉 WDT 计时器，清完后会继续计数。如果系统运用在强干扰的环境中，建议用户选择内置式看门狗振荡源，因为如果程序不正常时又刚好执行到 HALT 指令，选用系统时钟 4 分频来源的 WDT 会使系统时钟停止，看门狗也就失去保护的作用。若选用内置的 WDT 振荡源则不会受 HALT 指令影响，就可达到保护作用的功能。

看门狗定时器前置分频器

WS2	WS1	WS0	分频率
0	0	0	1：1
0	0	1	1：2
0	1	0	1：4
0	1	1	1：8
1	0	0	1：16
1	0	1	1：32
1	1	0	1：64
1	1	1	1：128

<例> 选内置 WDT 振荡源并设定 time-out 时间为 160ms

```
MOV A,          03H   ←   WS2=0,WS1=1,WS0=1
MOV WDTS,       A
```

定时/计数器 (Timer/Event Counter)

HT48C70 有两组定时/计数器，每组包含一个 16 位元可编程的向上计数器。它的时钟来源可从外部信号输入或由内部系统时钟 4 分频得来。所以用户可将其规划为定时器，也可将其规划为计数器。这两组定时/计数器功能完全一样，底下用户将只针对其中一组来分析。

如果选择作为计时器时，其时钟来源为系统频率 4 分频，也就是每一个指令周期计时寄存器会加一，举例说，如果系统时钟为 4MHz (外接振荡晶体为 4MHz)，4 分频为 1MHz，指令周期为 1ms，因此计时单位为 1ms，也就是每 1ms 送一个计时钟波到计时寄存器使其值加一。如果是选择当计数器时，外部脉波在 T1、T2、T3、T4 中间有任何变化时就会被记录到计数寄存器中。

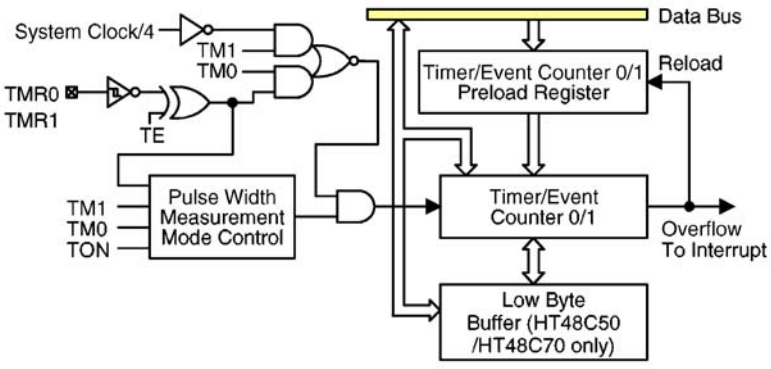
有三个实际位置对应到定时/计数寄存器 0 (TMR0L，TMR0H，TMR0C)，TMR0L 为定时/计数低位元组，TMR0H 为高位元组，TMR0C 为定时/计数控制寄存器。当用户要对 TMR 做写值的动作时必须先写 TMR0L 再写 TMR0H。对 TMR0L 写时是把数据先写入一个缓冲器里，对 TMR0H 写时才会将 16 位元的初值放到定时/计数寄存器 (Timer/event counter pre load register)，请读者注意此顺序。当由初值向上计数到 FFFFH 时，再数一下会发生溢出，此时单片机会自动将初值再载入到定时/计数寄存器然后重新计数。要读 TMR 值的动作刚好要相反，须先对 TMR0H 读取，在此同时会将 TMR0L 的值放入缓冲器里，这样就可读到当时要的值，而不会因有指令差而读错值。如果只想知道 TMR0L 的值亦须先读取 TMR0H。

定时/计数控制寄存器 (TMRC) 里定义了计数的使能与除能，高电平或低电平计数，还有计数的操作模式。如下表所示：

定时/计数器 0/1 控制寄存器 TMR0C/TMR1C

符号	位	功能
—	0-2	未被使用,读出为 0
TE	3	定义定时/计数器边缘触发方式: （TMR0/TMR1）=0：电平由低到高有效 （TMR0/TMR1）=1：电平由高到低有效
TON	4	允许/禁止计数(1=允许 0=禁止)
—	5	未用,读出为 0
TM0 TM1	6 7	定义操作模式 01=外部计数模式(外部时钟) 10=定时模式(内部时钟) 11=脉冲宽度测量模式 00=未用到

定时/计数器功能图



这两组定时/计数寄存器都有 3 种工作模式可以通过软件选择使用。

TM0，TM1 两位元定义操作模式。共有三种模式

外部计数模式

内部计时模式

脉冲宽度量测模式

外部计数模式用来计数外部事件，也就是说脉冲来自外部 TMR 脚。内部计时模式信号则来自指令周期。脉冲宽度测量模式用来测量外部脉冲宽度，计数的方式是以指令周期作为时钟来源的。

在外部计数和内部计时模式中，一旦开始计数，会从目前寄存器值开始往上计数到 FFFFH。发生溢出时会将寄存器的值重新载入到计数寄存器中，同时置位中断请求标志 (T0F/T1F;BIT 5/6 of INTC)。

在脉冲宽度量测模式中，TON=1，TE=1 的情况下，当 TMR 脚测到一个上升沿 (或下降沿：如果 TE=0)，计数器就开始计数，直到外部电平又回到原来的低电平才停止计数，并且会清除 TON，计数到的值将留在计数器中，也就是说一次只能计数一个脉冲的宽度。要再重新计数必需再次设定 TON 。如果计数过程中发生溢出，其动作行为与先前所述的两个模式相同。要使计数器开始计数，须设定 TON 位元，在脉冲宽度量测模式中，当脉冲量测周期结束时，硬件会自动将 TON 清除，但在另外两种模式中，TON 只能用软件指令清除。

无论哪种模式，写 "0" 到 ET0I/ET1I 可以禁止中断服务。在定时/计数器停止计数时，若写值到定时/计数寄存器中会将值放入定时/计数器里，但如果在计数过程中则只会存入定时/计数预置寄存器中，需等到发生溢出时才会从定时/计数预置寄存器中载入到定时/计数器寄存器中。

<例 1> 如何设定内部计时模式

```
MOV     A,80H           ;设定 low 到 high 触发并设定为内部计时模式
MOV     TMR0C,A         ;bit3=0,bit6=0,bit7=1
MOV     A,00H
MOV     TMR0L,A         ;初值先放入 TMR0L
MOV     TMR0H,A         ;再放到 TMR0H
SET     ET0I            ;使能计时中断
SET     TMR0C.4         ;启动计时
        :               ;如果溢出发生会跳至中断向量 08H
        :
```

<例 2> 如何设定外部计时模式

```
MOV     A,40H           ;设定 low 到 high 触发并设定为外部计时模式
MOV     TMR0C,A         ;bit3=0,bit6=1,bit7=0
MOV     A,00H
MOV     TMR0L,A         ;初值先放入 TMR0L
MOV     TMR0H,A         ;再放到 TMR0H
SET     ET0I            ;使能计时中断
SET     TMR0C.4         ;启动计数
        :               ;如果溢出发生会跳至中断向量 08H
        :
```

<例 3> 如何设定脉冲宽度量测模式

```
MOV     A,0C0H          ;设定 low 到 high 触发(侦测 HIGH PULSE 宽度)
                        ;并设定为脉冲宽度量测模式
MOV     TMR0C,A         ;bit3=0,bit6=1,bit7=1
MOV     A,00H
MOV     TMR0L,A         ;初值先放入 TMR0L
MOV     TMR0H,A         ;再放到 TMR0H
```

```
SET     ET0I            ;使能计时中断功能
SET     TMR0C.4         ;启动计数
        :               ;等到 PULSE 结束时自动停止计数
        :               ;要再计数时必须再设定 TMR0C.4
                        ;如果溢出发生会跳至中断向量 08H
```

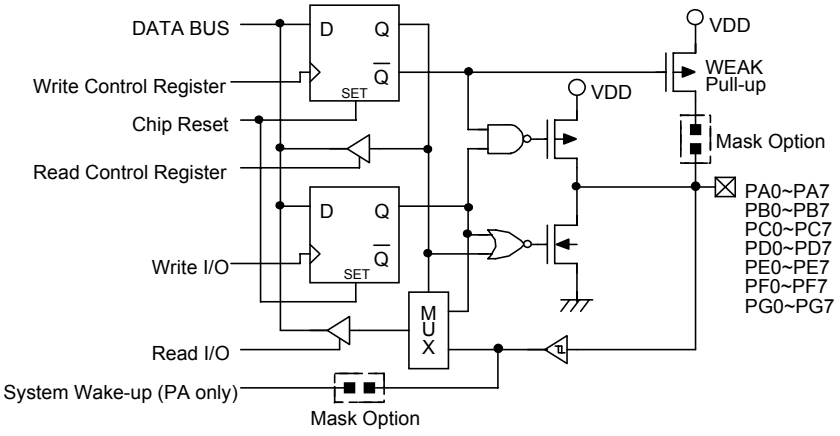
第 四 章

輸入/輸出口

4

每一个输入/输出口对应于数据存储器的某个位置，而且有自己的控制寄存器用来控制输入或输出。用软件指令来更改控制寄存器，可以动态地改变 I/O port 为输入或输出。如果要当输出使用，控制寄存器必须写入 "0" ，输出是 CMOS 结构。如果要当输入使用，控制寄存器必须写入 "1" 。输入来源则要特别注意，是依据控制寄存器而不同的。当控制寄存器位元为 "1" 时，输入来自外部逻辑电平。当控制寄存器位元为 "0" 时，输入则来自内部锁存的值。在晶片复位后，所有输入/输出口是处在输入状态，如果选择上拉电阻 (pull up) 选项则会在高电平，否则是悬空的 (floating)。我们可以用 SET [m].i 或 CLR [m].i 来设定或清除 I/O port 的每一位元。有些指令对输入/输出口的运作是比较特别的，像 SET [m].i，CLR [m].i，CPL [m]，CPLA [m]。这些指令在对输入/输出口动作时，是先将整个口的状态读入，做或 (or) 运算，然后把结果写回对应的锁存器或累加器中。

输入/输出口功能图



所有 I/O 口的每个位元都可被单独设定为输入或输出。下面我们用程序片段来说明如何设定口的状态：

<例>

```
MOV     A,   0F0H
MOV     PAC,A       ←    将 PA0~PA3 规划为输出口，PA4~PA7 规划为输入口
MOV     A,   03H
MOV     PA,  A      ←    PA0、PA1 输出 high，PA2、PA3 输出 low。
                         PA4~PA7 为任意值(因为为输入口)
```

第 五 章

掩模选项

5

掩模选项 (Mask Option) 是提供使用者对晶片多重应用的一个方案，譬如说：振荡模式可以选择 RC 振荡模式或晶体振荡模式，这两种模式各有其应用的范畴，对时钟脉冲精确度的要求如果不很在意的话可以选择 RC 振荡模式，如此可降低零件成本，对时钟脉冲精确度要求较严苛的话则就要选择晶体振荡模式了，相对的零件成本也就较高。当然 Holtek 单片机不仅提供上述选项，它提供了更多选项让使用者有较多的应用，比如要不要上拉电阻，看门狗计时器等，都可让使用者自由决定，让使用者的应用电路更灵活，这也是 Holtek 单片机的特点之一。

下表为掩模选项的说明：

掩模选项	说明
振荡类型	这个选项确定是 RC 或石英振荡器被选择来作为系统时钟。
WDT 振荡源	有三种选择的方式，芯片内的 RC 振荡器，指令时钟，或禁止 WDT。
CLR WDT 次数	这个选项定义用指令清除 WDT 的方法。“once”指的是用 CLR WDT 指令功能清除 WDT。“Twice”指的是必须要用 CLR WDT1 和 CLR WDT2 二条指令来清除 WDT。
Wake-up 选项	这个选项来定义激活唤醒功能。定义外部的输入/输出引脚（仅 PA 具有）使芯片从 HALT 状态中唤醒的功能。
上拉(PULL-HI)电阻选项	这个选项是在设定口的输入模式是否有内置的上拉电阻。所有的 I/O 口的每一位元均可个别设定有无上拉电阻。

选项功能表

最后要说明的一点是：在开发过程中选项是随时可以更改的，可是一旦将程序码还有掩模选项表交给 Holtek 制作 IC 后就决定了这颗 IC 的功能了，除非另外再开一颗 IC。

第 六 章
可烧录一次单片机芯片烧录器

6

什么是 OTP (One-Time Programmable)?

所谓 OTP (One-Time Programmable) 是指可烧录一次单片机，就是单片机再加上一个 PROM (可烧录一次的存储器)，由于半导体生产技术日新月异，可以把开发好的单片机连同其程序，制作在同一块芯片中，这样在大量生产和体积上都可作大幅度的改进。但是生产的必要条件是程序开发完成，接著要开光罩、生产硅片再切割包装成我们要的芯片，因此从程序开发完成到真正批量生产芯片还要一段时间，面对竞争日益加剧的环境下，似乎有必要想办法填补这段等待时间所失去的商机，所以就有可烧录一次单片机芯片的产生。

它的优点不只是可以弥补等待芯片生产的时间，更是在发展程序初期验证程序及将来芯片生产出来后之实际状况所不可或缺的工具。虽然它比大量生产下之芯片在价格上贵了几倍，但是在程序发展初期或者是对于高淘汰率及高时效性的产业中，可烧录一次单片机芯片更有可能由配角跃升为主角，成为主力生产的芯片。

可烧录一次单片机芯片烧录器 (OTP Programmer)

为了适应时代潮流，Holtek 大部份的单片机芯片，都有提供 OTP 的版本，并提供烧录工具帮助用户烧录。

目前 Holtek 所提供的烧录工具有两种，一是 HT-OTP Programmer，一是 HT-HandyWriter。

不管是使用那一种烧录工具，操作都非常简单，只要先在发展系统 HT-IDE 2000 中，将你的源程序产生 .OTP 文件，再选用以上所提的任一种烧录工具将 .OTP 文件载入，即可开始烧录，在这里特别要提到的是，若要使用旧有的 HT-OTP Programmer 烧录新的 0.5 μ s 工艺的 OTP IC，必需先将烧录器作小部份的线路修改，至于修改方式 Holtek 公司的网站 (http://www.holtek.com.tw) 上会提供最新的数据。

至于 HT-HandyWriter，是 Holtek 公司 2000 年最新开发完成的经济型烧录器，在此要对此烧录器作一些介绍。

简介

Holtek HandyWriter 是一种专为烧录 OTP chip(One-Time Programmable)的简易烧录器。凡是 Holtek 半导体公司开发完成的这类可烧录一次单片机芯片(OTP chip)都可以使用。这种简易烧录器将程序烧录到芯片内。这种烧录器的特点为轻巧短小，如手掌大小。同时安装和使用都很容易，功能简单明了。

烧录器的结构如图一，组成元件列举如下：

一个 40 pin 接脚的 DIP 型插槽 (TEXTOOL)

一个 25 pin 接脚的打印端口 D 型接头 (D-type female connector)

一个 96 pin 接脚的 VME 接头 (VME connector)

使用这个烧录器时，需要下列的装备与系统

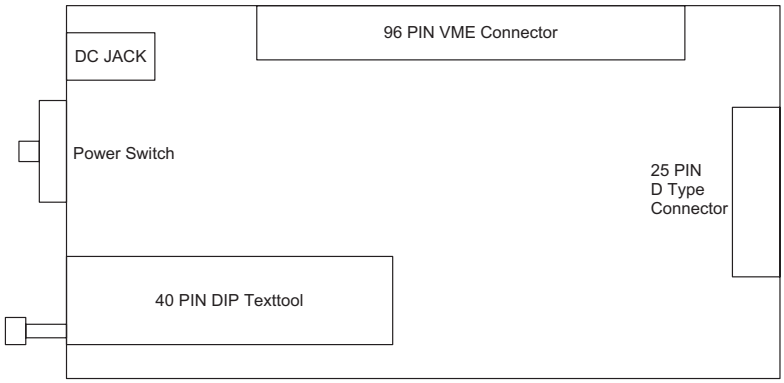
稳压电源(power adapter)，输出电压为 16V，输出电流至少为 500mA。最好是使用烧录器包装盒内所附的稳压电源。

IBM386 或更高级的个人电脑

Win95/98/NT 的操作系统

HT-IDE 2000 发展系统

如果是通过 HT-ICE 与个人电脑连接时，则需要一个 HT-ICE，否则不需要

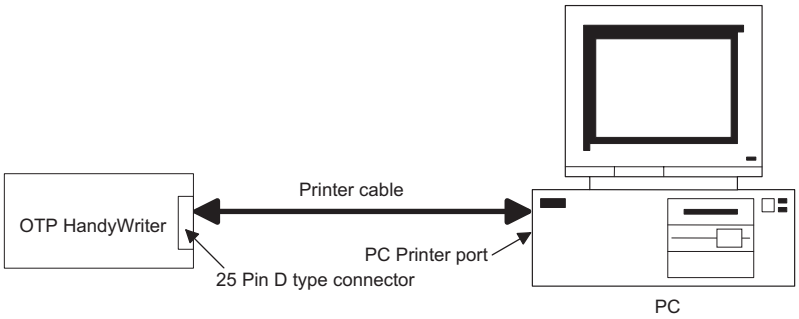


图一

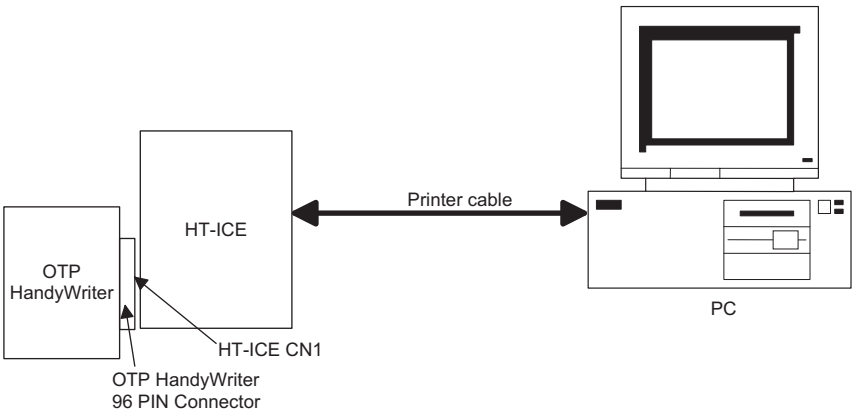
安装

直接与个人电脑连接(图二)，使用并口线连接 HandyWriter 的 25 pin D 型接头与个人电脑的打印端口(Printer Port) 或先将 HandyWriter 插入 HT-ICE 的 CN1 接头(HandyWriter 的 96 pin VME 接头)，再将 HT-ICE 与个人电脑以 Printer cable 相连接 (图三)

安装 HT-IDE 2000 系统，请参考 HT-IDE 2000 User's Guide



图二

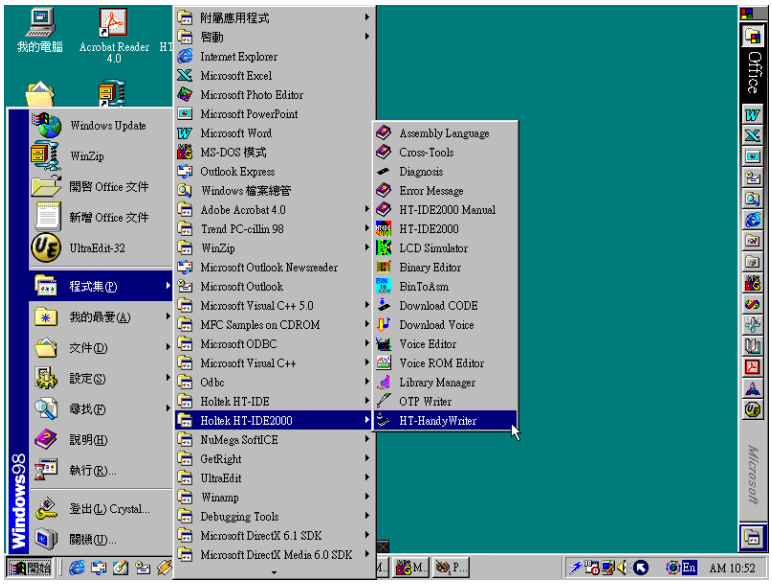


图三

如何使用 HT-HandyWriter 烧写 OTP Chip

➔ 启动 HT-HandyWriter 烧录程序

打开"Holtek HT-IDE 2000"，将鼠标指针移到 HT-HandyWriter 并按下鼠标左键。启动 HT-HandyWriter，如图四。

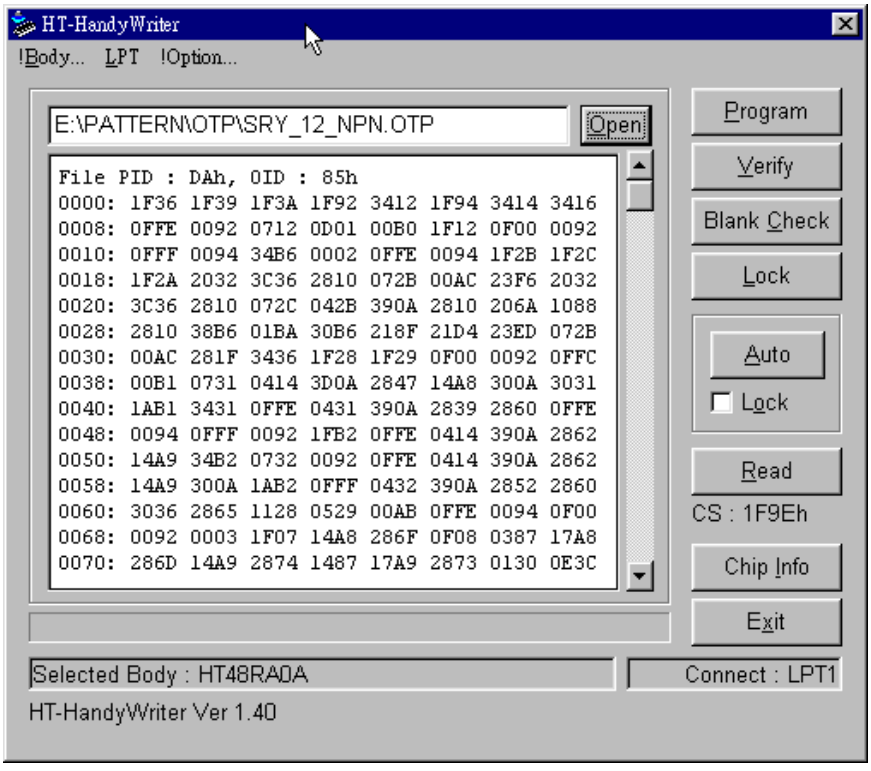


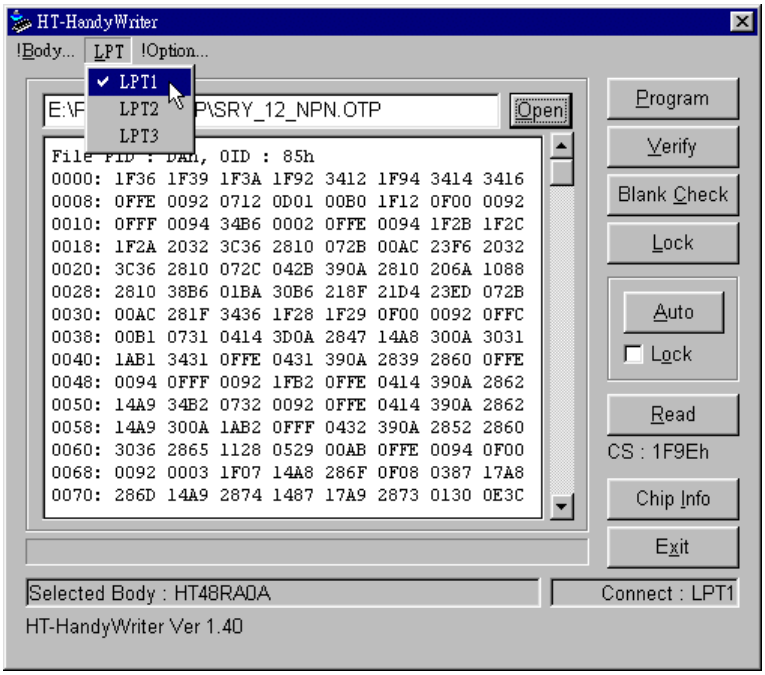
图四

➔ LPT - 设定打印端口(Printer Port)

在启动 HT-HandyWriter 进入图五的窗口后，需要事先设定打印端口。使用鼠标点选 "LPT" 菜单，即出现图六的打印端口功能表。可以选择的有 LPT1、LPT2、LPT3。如果 HandyWriter 是连接到 HT-ICE，则视 HT-ICE 与个人电脑的第几个打印端口相连而定，例如 HT-ICE 与个人电脑的 LPT1 相接，则在图六中选择 LPT1。如果 HandyWriter 是直接与个人电脑的打印端口相连，则设定此相连的打印端口即可。

图五





图六

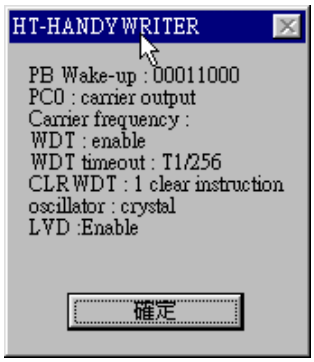
➔ !Body - 选取 OTP Chip 类型

使用鼠标点选 "!Body" 指令，即出现图七所示之 (Set Body) 对话框。若 OTP Chip 内部未记录类型识别码的话，则以用户所选的 Chip 类型为依据，完成所有写入与读取动作。

图七

➔ !Option - 检视 IC Option

使用鼠标点选"!Option" 指令，即出现如图八所示，显示目前所开启文件，或由 OTP Chip 读取出之内容的 Option 设定说明。



图八

➔ HT-HandyWriter 烧录功能 (Programming Functions)

图五显示 HT-HandyWriter 提供的功能，每个按钮代表一个命令，各命令的功能说明如下：

- Open

开启一个后缀名为 .OTP 的文件，并将其内容载入电脑的存储器，当执行烧录时会从存储器取得数据。按下 Open 钮之后，会出现磁盘、目录与文件名可供选定。文件打开后，文件内容会显示在讯息窗口内，并于 Read 按钮下方显示所打开文件的校验码 (Checksum)。

- Program

此命令将会执行两项功能，首先会将电脑存储器中的数据烧录到 HandyWriter 上的 OTP chip 内，之后再比较 OTP chip 的内容是否与电脑存储器的数据一致，并将结果显示在 HT-HandyWriter 窗口内。

- Verify

验证 OTP chip 的内容是否与目前电脑存储器的数据一致。会先从 OTP chip 读取内容再做比较。验证结果会显示在 HT-HandyWriter 窗口内。

- Blank Check

检查 HandyWriter 上的 OTP chip 是否已经烧写过。检查结果会显示在窗口内。如果不是空的 IC，已被烧录过的位置会显示出来。

- Lock

此命令会使 HandyWriter 设定 OTP chip 的保护功能，禁止将此颗 OTP chip 的内容读出。一般是在执行 Program 功能之后，需要将 OTP chip 的内容做保护时，使用此命令。

- Auto

此命令会以 Blank Check、Program 与 Verify 的顺序将此三个命令执行完毕，若任何命令的执行有错，将会停止并且不继续下一个命令的执行。底下有一个 Lock 的选项，如果需要在执行 Auto 命令做烧录 OTP chip 之同时并且将此 OTP chip 设定保护，防止读取时，可以先选取此项，然后再按下 Auto 钮。

- Read

此命令会将 HandyWriter 上 OTP IC 的内容读出，再将它存入电脑的存储器内，并在 Read 按钮下方显示文件内容之校验码 (Checksum)。也可以将之存入后缀名为 .OTP 的文件内。

- Chip Info

此命令会从 IC 内读出 Power-On ID，Software ID，ROM size，Option size 等数据，并显示 2Get Info from chip2。告诉用户以上数据是自 IC 中取得。

若 IC 内部没有记录这些数据的话，则会显示由!Body 命令所设定的 IC 规格，并显示 2Get Info from ini2。告诉用户以上数据由系统设定取得。

讯息

- ➔ HandyWriter connect to LPT1.
  HandyWriter 已与 LPT1 相连接
- ➔ Cannot connect to ICE.
  HandyWriter 没有直接与 Print Port 相连，HT-ICE 也没有接在 Print Port 上
- ➔ Invalid EV Chip!
  HandyWriter 没有支援此 HT-ICE 上的 EV chip，请与相关技术人员联系，升级到 HandyWriter 可支持的版本
- ➔ Connect to HandyWriter through ICE.
  HandyWriter 已透过 ICE 连接成功
- ➔ Cannot find HandyWriter, please connect it to ICE.Or this HandyWriter is an old version.
  HT-ICE 已接到 Print Port 上，但 HandyWriter 却没有接到 HT-ICE 上。
  也可能使用的是旧版的 HandyWriter (THANDYOTP-A)，所以无法侦测连线与否。若为前者，请将 HandyWriter 连接到 ICE 上。
- ➔ File PID : ADh, OID : 50h
  所打开文件记录的 Power-on ID 为 ADh，Software ID 为 50h
- ➔ Invalid OTP file format!
  要打开的文件格式不正确
- ➔ The chip PID:ADh, OID:50h doesn't match with the file PID:ADh, OID:51h
  Are you sure to continue?
  OTP Chip 的类型与打开文件所支援的类型不符合
- ➔ Chip ROM size:0400h, File ROM size:0800h. System will set ROM size as 0400h.
  Are you sure to continue?
  OTP Chip 可写入空间为 400h，文件内容大小为 800h，HandyWriter 将只写入 400h 的值到 Chip 内
- ➔ Addr:xxxxh, Data:yyyyh, Rdata:zzzzh
  Program/Option Verify failed!!
  验证 Program 或 Option 数据时发生错误
  失败原因：从 OTP chip 位置 xxxxh 读出的数据 zzzzh，与电脑存储器内的数据 yyyyh 不一致
- ➔ Addr:xxxxh, Data:zzzzh
  Not Blank!
  OTP chip 位置 xxxxh 的内容是 zzzzh，并非空的。
- ➔ Chip mismatched!
  在 HandyWriter 上的 OTP chip 与已被载入的 .OTP 文件所记录的 OTP chip 不是同一型。无法执行所指定的指令。
- ➔ Chip is locked!
  在 HandyWriter 上的 OTP chip 已经被锁住。无法执行所指定的命令。
- ➔ No data to verify/program!
  在执行 Verify 或 Program 命令之前，必须将 .OTP 的文件载入。请参考 HT-HandyWriter 烧录功能的 Open 命令。

第 七 章

指令集说明

7

HT48CX0 系列之定址方式有下面 5 种定址方式：

➔ **立即定址**

此一定址法是将立即的常数值紧跟在运算码 (opcode) 后，例如：

```
mov  a,33h
add  a,33h
```

➔ **直接定址**

直接定址的情况只允许在存储器之间作数据传送，例如：

```
mov  [33h],a
clr  [33h]
```

➔ **间接定址**

在间接定址方式中，必定要使用到间接定址暂存器(R0，R1)，例如：

```
mov  R0,a
mov  a,R0
```

➔ **特殊暂存器定址**

此一定址方式完全是针对某一暂存器作运算，例如：

```
clr  WDT
clr  WDT1
```

➔ **指针定址**

指针定址只适用在配合做查表指令，例如：

```
mov  a,02h
mov  TBLP,a
```

ADC A, [m]

说明： 本指令把累加器、数据存储器值以及进位标志相加，结果存放到累加器。

运算过程： ACC←ACC+[m]+C

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> ADC A,[60H]

运算前 ACC=11H, [60H]=22H, C=1

运算后 ACC=34H, [60H]=22H, C=0

ADCM A, [m]

说明： 本指令把累加器、数据存储器值以及进位标志相加，结果存放到存储器。

运算过程： [m]←ACC+[m]+C

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> ADCM A,[60H]

运算前 ACC=11H, [60H]=22H, C=1

运算后 ACC=11H, [60H]=34H, C=0

ADD A, [m]

说明： 本指令把累加器、数据存储器值相加，结果存放到累加器。

运算过程： ACC←ACC+[m]

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> ADD A,[60H]

运算前 ACC=11H, [60H]=22H

运算后 ACC=33H, [60H]=22H

ADD A, X

说明： 本指令把累加器值和立即数相加，结果存放到累加器。

运算过程： ACC←ACC+X

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> ADD A,20H

运算前 ACC=11H

运算后 ACC=31H

ADDM A, [m]

说明： 本指令把累加器、数据存储器值相加，结果放到数据存储器。

运算过程： [m]←ACC+[m]

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> ADDM A,[60H]

运算前 ACC=11H, [60H]=22H

运算后 ACC=11H, [60H]=33H

AND A, [m]

说明： 本指令把累加器值、数据存储器值做逻辑与，结果存放到累加器。

运算过程： ACC←ACC “AND” [m]

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> AND A,[60H]

运算前 ACC=11H, [60H]=88H

运算后 ACC=00H, [60H]=88H

AND A, X

说明： 本指令把累加器值、立即数做逻辑与，结果存放到累加器。

运算过程： ACC←ACC “AND” X

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> AND A,33H

运算前 ACC=11H

运算后 ACC=11H

ANDM A, [m]

说明： 本指令把累加器值、数据存储器值做逻辑与，结果放到数据存储器。

运算过程： ACC←ACC “AND” [m]

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> ADD A,[60H]

运算前 ACC=11H, [60H]=88H

运算后 ACC=11H, [60H]=00H

CALL addr

说明： 本指令直接调用地址所在处的子程序，此时程序计数器加一，将此程序计数器值存到堆栈寄存器中，再将子程序所在处的地址存放到程序计数器中。

运算过程： Stack←PC+1

PC ← addr

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> CALL DELAY

MOV A,08H

…

DELAY PROC

SUB A,09H

…

RET

DELAY ENDP

运算前 PC=0100H, DELAY=300H, STACK=0000H

运算后 PC=0300H, DELAY=300H, STACK=0101H

也就是说，当程序执行到 CALL DELAY 这句时，程序计数器指针跳到 DELAY 子程序即执行 SUB A，09H 一句，直到 RET 返回以后才执行 MOV A，08H 等。

CLR [m]

说明： 本指令将数据存储器内值清零。

运算过程： [m] ←00H

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> CLR [60H]

运算前 [60H]=88H

运算后 [60H]=00H

CLR [m]. i

说明： 本指令将数据存储器内第 i 位值清零。

运算过程： [m].i ←0

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> CLR [60H].3

运算前 [60H]=88H

运算后 [60H]=80H

CLR WDT

说明： 本指令清除 WDT 和 WDT 计数器（从 0 开始重新计数），电压下降位（PD）和计数溢出位（TO）也被清零。

运算过程： WDT &WDT 计时器 ← 00H

PD&TO ← **0**

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	0	0	—	—	—	—

<例> CLR WDT

运算前 WDT 计时器=11H, PD=1,TO=1

运算后 WDT 计时器=00H, PD=0,TO=0

CLR WDT1

说明： 本指令是当选择两个指令清除看门狗计时器时，才会使用。必须搭配 CLR WDT2,而且要在 CLR WDT2 之前运行，才可清除 WDT 和 WDT 计时器（从 0 开始重新计数），电压下降位（PD）和计数溢出位（TO）也被清零。

运算过程： WDT &WDT 计时器 ← 00H

PD&TO ← **0**

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	0	0	—	—	—	—

<例> CLR WDT1

…

CLR WDT2

运算前 WDT 计时器=11H, PD=1,TO=1

运算后 WDT 计时器=00H, PD=0,TO=0

注：只有当执行完 CLR WDT2 以后才会 WDT 和 WDT 计时器清零，PD、TO 清零。这也是这条语句与 CLR WDT 的差别。

CLR WDT2

说明： 本指令是当选择两个指令清除看门狗计时器时，才会使用。必须搭配 CLR WDT1,而且要在 CLR WDT1 之前运行，才可清除 WDT 和 WDT 计时器（从 0 开始重新计数），电压下降位（PD）和计数溢出位（TO）也被清零。

运算过程： WDT &WDT 计时器 ← 00H

PD&TO ← **0**

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	0	0	—	—	—	—

<例> CLR WDT1

…

CLR WDT2

运算前 WDT 计时器=11H, PD=1,TO=1

运算后　　WDT 计时器=00H, PD=0,TO=0

CPL [m]

说明：　　本指令是将数据存储器内值取反。

运算过程：　　[m] ← [/m]

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例>　　CPL　[60H]

运算前　　[60H]=11H

运算后　　[60H]=EEH

CPLA　[m]

说明：　　本指令是将存储器内值取反存放在累加器中。

运算过程：　　ACC ← [/m]

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例>　　CPLA [60H]

运算前　　[60H]=11H, ACC=11H

运算后　　[60H]=11H, ACC=EEH

DAA　[m]

说明　　本指令将累加器高低四位分别调整为 BCD 码。如果低四位的值大于“9”或 AC=1，那么 BCD 调整就执行对原值加“6”，并且内部进位标志 AC1=/AC，即 AC 求反；否则原值保持不变。如果高四位的值大于“9”或 C=1，那么 BCD 调整就执行对原值加“6”再加 AC1，并把 C 置位；否则 BCD 调整就执行对原值加 AC1，C 的值保持不变。结果存放到数据存储器中，只有进位标志位（C）受影响.

操作　　如果 ACC.3~ACC.0 > 9 或 AC=1
那么 [m].3~[m].0 ← (ACC.3~ACC.0)+6, AC1=/AC
否则 [m].3~[m].0 ← (ACC.3~ACC.0) , AC1=0
并且
如果 ACC.7~ACC.4+AC1 > 9 或 C=1
那么 [m].7~[m].4 ← (ACC.7~ACC.4)+6+ AC1,C=1
否则 [m].7~[m].4 ← (ACC.7~ACC.4) + AC1,C=C

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	√

<例>　　MOV　A,07BH
DAA　[60H]

运算前　　[60H]=00H, ACC=7BH, AC=0,C=0，AC1=0
运算后　　[60H]=81H, ACC=7BH, AC=0, C=0，AC1=1

DEC　[m]

说明：　　本指令将数据存储器内值减一再放回数据存储器。
运算过程：　　[m] ← [m]-1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例>　　DEC　[60H]
运算前　　[60H]=11H
运算后　　[60H]=10H

DECA [m]

说明：　　本指令将存储器内值减一,再放到累加器。
运算过程：　　ACC ← [m]-1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例>　　DECA [60H]
运算前　　[60H]=11H, ACC=00H
运算后　　[60H]=11H, ACC=10H

HALT

说明：　　本指令终止程序执行并关掉系统时钟，RAM 和寄存器内值被保留，WDT 和 WDT 计数器也被清除，电源下降位(PD)被设为 1， WDT 计数溢出位（TO）被清为 0。
运算过程：　　PC ← PC+1
　　PD ← 1
　　TO ← 0
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	0	1	—	—	—	—

<例>　　HALT
运算前　　PCL=0011H, PD=0, TO=0,WDT 计数器=11H
运算后　　PCL=0012H, PD=1, TO=0,WDT 计数器=00H

INC　[m]

说明：　　本指令将数据存储器内值加一,结果放回数据存储器。
运算过程：　　[m] ← [m]+1

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> INC [60H]
运算前 [60H]=10H
运算后 [60H]=11H

INCA [m]

说明： 本指令是将存储器内值加一,结果放到累加器。
运算过程： ACC ← [m]+1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> DECA [60H]
运算前 [60H]=10H, ACC=00H
运算后 [60H]=10H, ACC=11H

JMP addr

说明： 本指令是将要跳到的目的地直接放到程序计数器内。
运算过程： PC ← addr
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> JMP DELAY
运算前 PC=0010H, DELAY=0100H
运算后 PC=0100H, DELAY=0100H

MOV A, [m]

说明： 本指令是将数据存储器内值送到累加器内。
运算过程： ACC ← [m]
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> MOV A, [60H]
运算前 ACC=00H, [60H]=10H
运算后 ACC=10H, [60H]=10H

MOV A, X

说明： 本指令是将立即数送到累加器内。
运算过程： ACC ← X
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> MOV A, 20H
运算前 ACC=00H
运算后 ACC=20H

MOV [m], A

说明： 本指令是将累加器值送到数据存储器内。
运算过程： [m] ← ACC
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> MOV [60H], A
运算前 ACC=00H, [60H]=10H
运算后 ACC=00H, [60H]=00H

NOP

说明： 本指令不作任何运算，而只将程序计数器加一。
运算过程： PC ← PC+1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> NOP
运算前 PC=0100H
运算后 PC=0101H

OR A, [m]

说明： 本指令是把累加器、数据存储器值做逻辑或，结果放到累加器。
运算过程： ACC←ACC “OR” [m]
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> OR A,[60H]
运算前 ACC=00H, [60H]=10H,
运算后 ACC=10H, [60H]=10H,

OR A, X

说明： 本指令是把累加器值、立即数做逻辑或，结果放到累加器。
运算过程： ACC←ACC “OR” X
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> ADD A,20H
运算前 ACC=00H
运算后 ACC=20H

ORM A, [m]
说明： 本指令是把累加器值、存储器值做逻辑或，结果放到数据存储器。
运算过程： ACC←ACC “OR” [m]
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> ORM A,[60H]
运算前 ACC=01H, [60H]=10H
运算后 ACC=01H, [60H]=11H

RET
说明： 本指令是将堆栈寄存器中的程序计数器值送回程序计数器。
运算过程： PC ← Stack
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>
```
CALL    DELAY
MOV     A,08H
…
DELAY   PROC
…
RET
PROC    ENDP
```
运算前 PC=0111H, Stack=0100H
运算后 PC=0100H
注: 当执行完 RET 指令后,程序跳回调用这个子程序的地方,执行下一句语句,在本例中即 MOV A,09H。在中断返回时，也执行 RET 指令。

RET A, X
说明： 本指令是将堆栈寄存器中的程序计数器值送回程序计数器,并将立即数送回累加器。
运算过程： PC ← Stack
ACC ← X
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> ORG 04H

```
JMP     INC-SEC
ORG    08H
…
INC-SEC:
…
RET    A,20H
```

运算前　　ACC=01H, PC=0100H, Stack=0011H

运算后　　ACC=20H, PC=0011H

注：一般来说，04H 段执行外部中断，跳到 INC-SEC 执行完后，RET　A,20H 返回到原程序流程，并把 20H 送到累加器。

RETI

说明：　　本指令是将堆栈寄存器中的程序计数器值送回程序计数器,与 RET 不同的是它使用在中断程序结束返回时， 它还会将中断控制寄存器 INTC 的 0 位（EMI）中断使能位置 1，允许中断服务。

运算过程：　　PC ← Stack

EMI ← 1

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>　　RETI

运算前　　PC=0100H, Stack=0010H, EMI=0

运算后　　PC=0010H, EMI=1

注：当中断程序被服务时，EMI 被自动清零，以防止其它中断发生。当执行完 RETI 指令后，EMI 被置位，允许中断发生。

RL [m]

说明：　　本指令是将数据存储器内值左移一位，第 7 位移到第 0 位，结果送回数据存储器。

运算过程：　　[m].0 ←[m].7, [m].(i+1) ←[m].i :i=0~6

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>　　RL　[60H]

运算前　　[60H]=10H

运算后　　[60H]=20H

RLA [m]

说明：　　本指令是将存储器内值左移一位，第 7 位移到第 0 位，结果送到累加器，而数据存储器内值不变。

运算过程： ACC.0 ←[m].7, ACC.(i+1) ←[m].i :i=0~6

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> RLA [60H]

运算前 ACC=01H, [60H]=10H

运算后 ACC=20H, [60H]=10H

RLC [m]

说明： 本指令是将存储器内值与进位位左移一位，第 7 位取代进位标志，进位标志移到第 0 位，结果送回数据存储器。

运算过程： [m].(i+1) ←[m].i :i=0~6

[m].0 ←C

C ← [m].7

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	√

<例> RLC [60H]

运算前 C=1, [60H]=10H

运算后 C=0, [60H]=21H

RLCA [m]

说明： 本指令是将存储器内值与进位位左移一位，第七位取代进位标志，进位标志移到第 0 位，结果送回累加器。

运算过程： ACC.(i+1) ←[m].i :i=0~6

ACC.0 ←C

C ← [m].7

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	√

<例> RLCA [60H]

运算前 C=1, ACC=01H, [60H]=10H

运算后 C=0, ACC=21H, [60H]=10H

RR [m]

说明： 本指令是将存储器内值循环右移，第 0 位移到第 7 位，结果送回数据存储器。

运算过程： [m].7 ←[m].0, [m].i ←[m].(i+1) : i=0~6

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C

—	—	—	—	—	—	—	—

<例> RR [60H]
运算前 [60H]=10H
运算后 [60H]=08H

RRA [m]

说明： 本指令是将数据存储器内值循环右移，第 0 位移到第 7 位，结果送回累加器，而数据存储器内值不变。

运算过程： ACC.7 ←[m].0, ACC.i ←[m].(i+1) :i=0~6

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> RRA [60H]
运算前 ACC=01H, [60H]=10H
运算后 ACC=08H, [60H]=10H

RRC [m]

说明： 本指令是将存储器内值加进位位循环右移，第 0 位取代进位标志，进位标志移到第 7 位，结果送回存储器。

运算过程： [m].i ←[m].(i+1) :i=0~6
[m]. 7 ←C
C ← [m].0

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	√

<例> RRC [60H]
运算前 C=1, [20H]=10H
运算后 C=0, [20H]=88H

RRCA [m]

说明： 本指令是将数据存储器内值加进位位循环右移，第 0 位取代进位标志，进位标志移到第 7 位，结果送回累加器，数据存储器内值不变。

运算过程： ACC.i ←[m].(i+1) :i=0~6
ACC.7 ←C
C ← [m].0

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	√

<例> RRCA [60H]
运算前 C=1, ACC=01H, [60H]=10H
运算后 C=0, ACC=88H, [60H]=10H

SBC A, [m]

说明： 本指令是把累加器值减去数据存储器值以及进位标志的取反，结果放到累加器。

运算过程： ACC←ACC+[/m]+ C

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> SBC A,[60H]
运算前 ACC=30H, [60H]=10H, C=1
运算后 ACC=20H, [60H]=10H, C=1

注:在减法中,C=1 时表示没有借位;而 C=0 表示有借位.被减数减去减数,够减时 C=1, 否则 C=0。

SBCM A,[m]

说明： 本指令是把累加器值减去数据存储器值以及进位标志取反，结果放到数据存储器。

运算过程： [m]←ACC+[/m]+C

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> SBCM A,[60H]
运算前 ACC=30H, [60H]=10H, C=1
运算后 ACC=30H, [60H]=20H, C=1

SDZ [M]

说明： 本指令是把数据存储器内值减 1, 判断是否为 0,若为 0 则跳过下一条指令，即如果结果为零，放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以取得正确的指令（二个指令周期）。否则执行下一条指令（一个指令周期）。

运算过程： 如果[m]-1=0，跳过下一条指令执行再下一条。

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C

—	—	—	—	—	—	—	—

<例>

```
MOV     A,03H
MOV     COUNT,A
SDZ     COUNT
JMP     GOON1
JMP     GOON2
GOON1:
        …
GOON2:
        …
```

运算前　　COUNT=03H

运算后　　COUNT=02H,因为 COUNT 不为 0,所以执行 JMP GOON1 指令,
否则执行 JMP　GOON2 指令

SDZA　　[M]

说明：　　本指令是把数据存储器内值减 1, 判断是否为 0, 为 0 则跳过下一行指令并将减完后数据存储器内值送到累加器,而数据存储器内的值不变，即若结果为 0，放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以取得正确的指令（二个指令周期）。否则执行下一条指令（一个指令周期）。

运算过程：　　如果[m]-1=0，跳过下一条指令执行再下一条.
ACC ← （[m]-1）

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>　　SDZA [60H]

运算前　　ACC=30H, [60H]=10H，　PC=0100H

运算后　　ACC=0FH, [60H]=10H，　PC=0101H

注:　　例子可见上例.

SET [m]

说明：　　本指令是把存储器内值每个位置为 1。

运算过程：　　[m] ← FFH

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>　　SET [60H]

运算前　　[60H]=10H
运算后　　[60H]=FFH

SET [m].i

说明：　　本指令是把存储器内值的第 i 位置为 1。
运算过程：　　[m].i ←1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>　　SET [60H].3
运算前　　[60H]=10H
运算后　　[60H]=18H

SIZ [m]

说明：　　本指令是把数据存储器内值加 1, 判断是否为 0。若为 0，跳过下一条指令，即放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以取得正确的指令（二个指令周期）。否则执行下一条指令（一个指令周期）。
运算过程：　　如果 （[m]+1=0），跳过下一行指令；[m] ←[m]+1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>　　SIZ　[60H]
JMP　GOON1
JMP　GOON2
运算前　　[60H]=FFH
运算后　　[60H]=00H，　执行 JMP　GOON2 语句,即跳过 JMP　LOOP1 语句。

SIZA

说明：　　本指令是把数据存储器内值加 1, 判断是否为 0, 若为 0 跳过下一条指令，即放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以取得正确的指令（二个指令周期），并将加完后存储器内值送到累加器,而数据存储器的值保持不变。否则执行下一条指令（一个指令周期）。
运算过程：　　如果[m]+1=0，跳过下一行指令；ACC ←（[m]+1）
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> SIZA [60H]
运算前 ACC=01H, [60H]=FFH， PC=0100H
运算后 ACC=00H, [60H]=FFH， PC=0102H

SNZ [m].i

说明： 本指令是判断数据存储器内值的第 i 位, 若不为 0，则程序计数器再加 1, 跳过下一行指令，放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以取得正确的指令（二个指令周期）。否则执行下一条指令（一个指令周期）。。

运算过程： 如果 [m].i≠0，跳过下一行指令。

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> SNZ [60H].4
JMP GOON1
JMP GOON2
运算前 [60H]=10H
运算后 [60H]=10H，第四位为 1，不为 0，则跳过 JMP GOON1 指令，
执行 JMP GOON2 指令。

SUB A,[m]

说明： 本指令是把累加器值、数据存储器值相减，结果放到累加器。

运算过程： ACC←ACC+[/m]+1

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> SUB A,[60H]
运算前 ACC=30H, [60H]=10H
运算后 ACC=20H, [60H]=10H

SUB A, X

说明： 本指令是把累加器值、立即数相减，结果放到累加器。

运算过程： ACC←ACC+/X+1

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> SUB A,20H
运算前 ACC=30H
运算后 ACC=10H

SUBM A,[m]

说明： 本指令是把累加器值、存储器值相减，结果放到存储器。
运算过程： [m]←ACC+[/m]+1
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	√	√	√	√

<例> SUBM A,[60H]
运算前 ACC=30H, [60H]=10H
运算后 ACC=30H, [60H]=20H

SWAP [m]

说明： 本指令是将数据存储器的低四位和高四位互换,再将结果送回数据存储器。
运算过程： [m].7~[m].4←>[m].3~[m].0
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> SAWP [60H]
运算前 [60H]=10H
运算后 [60H]=01H

SWAPA [m]

说明： 本指令是将数据存储器的低四位和高四位互换,再将结果送回累加器。
运算过程： ACC.3~ACC.0← [m].7~[m].4
ACC.7~ACC.4← [m].3~[m].0

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> SAWPA [60H]
运算前 ACC=30H, [60H]=10H
运算后 ACC=01H, [60H]=10H

SZ [m]

说明： 本指令是判断数据存储器内值是否为 0，为 0 则跳过下一行指令，即放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以得正确的指令（二个指令周期）。否则执行下一条指令（一个指令周期）。

运算过程： 如果 [m] = 0, 跳过下一行指令。

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>
```
SZ   [60H]
JMP  GOON1
JMP  GOON2
```
运算前 [60H]=10H

运算后 [60H]=10H，因为[60H]不为 0，执行 JMP GOON1 语句。

SZA [m]

说明： 本指令是判断存储器内值是否为 0，若为 0 则跳过下一行指令,即放弃在目前指令执行期间所取得的下一条指令，并插入一个虚构周期用以得正确的指令（二个指令周期）。并把存储器内值送到累加器，而存储器的值保持不变。否则执行下一条指令（一个指令周期）。

运算过程： 如果 [m] = 0，跳过下一行指令,并 ACC← [m]。

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>
```
SZA  [60H]
JMP  GOON1
JMP  GOON2
```
运算前 [60H]=00H，ACC=30H

运算后 [60H]=00H，ACC=00H，因为[60H]=0，跳过 JMP GOON1 指令，执行 JMP GOON2 指令。

SZ [m].i

说明： 本指令是判断存储器内第 i 位值是否为 0，若为 0 则跳过下一行指令，即放弃在目前指令执行期间所取得的下一条指令，并插入一个虚

构周期用以得正确的指令（二个指令周期)。否则执行下一条指令
(一个指令周期)。

运算过程：　如果　[m].i = 0，跳过下一行指令。

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>
```
SZ    [60H].3
JMP   GOON1
JMP   GOON2
…
```

运算前　[60H]=10H

运算后　[60H]=10H，因为[60H].3=0，跳过指令 JMP　GOON1，执行指令
JMP　GOON2

TABRDC [m]

说明：　本指令是将 TABLE 指针指向程序寄存器当前页， 将低位送到存储器，高位直接送到 TBLH 寄存器内。

运算过程：　[m]　←程序存储器低位
TBLH←程序存储器高位

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例>
```
MOV     A，020H
MOV     TALP，A
TABRDC  [60H]
…
ORG     20H
DC      1234H，5678H…
```

运算前　TBLP=20H, TBLH=00H, [60H]=10H

运算后　TBLP=20H, TBLH=12H, [60H]=34H

TABRDL [m]

说明：　本指令是将 TABLE 指针指向程序寄存器最后页， 将低位送到存储器，高位直接送到 TBLH 寄存器内。

运算过程：　[m]　←程序存储器低位
TBLH←程序存储器高位

影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	—	—	—

<例> MOV A，020H
MOV TALP，A
TABRDC [60H]
…
ORG 1F20H
DC 1234H，5678H…
运算前 TBLP=20H, TBLH=00H, [60H]=10H
运算后 TBLP=20H, TBLH=12H, [60H]=34H
注：假设 ROM 的空间有 8K，最后一页的地址为 1F00H~1FFFH。

XOR A, [m]
说明： 本指令是把累加器值、数据存储器值做逻辑异或，结果放到累加器。
运算过程： ACC←ACC “XOR” [m]
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> XOR A,[60H]，
运算前 ACC=00H, [60H]=10H
运算后 ACC=10H, [60H]=10H

XORM A, [m]
说明： 本指令是把累加器值、存储器值做逻辑异或，结果放到存储器。
运算过程： [m]←ACC “XOR” [m]
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> XORM A,[60H]
运算前 ACC=01H, [60H]=10H
运算后 ACC=01H, [60H]=11H

XOR A, X
说明： 本指令是把累加器值与立即数做逻辑异或，结果放到累加器。
运算过程： ACC←ACC “XOR” X
影响标志位

TC2	TC1	TO	PD	OV	Z	AC	C
—	—	—	—	—	√	—	—

<例> XOR A, 20H
运算前 ACC=00H
运算后 ACC=20H

备注：

X=8 位立即数
m 为存储器地址
ACC=累加器
i=0~7
addr=程序寄存器地址
√=对标志位有影响
—=对标志位没有影响