CMS69F012/013

用户手册

I/O 型 MCU

V1.4

（使用前请阅读第二页注意事项）

请注意以下有关CMS知识产权政策

＊ 中微半导体公司已申请了专利，享有绝对的合法权益。与中微半导体公司MCU或其他产品有关的专利权并未被同意授权使用，任何经由不当手段侵害中微半导体公司专利权的公司、组织或个人，中微半导体公司将采取一切可能的法律行动，遏止侵权者不当的侵权行为，并追讨中微半导体公司因侵权行为所受的损失、或侵权者所得的不法利益。

＊ 中微半导体公司的名称和标识都是中微半导体公司的注册商标。

＊ 中微半导体公司保留对规格书中产品在可靠性、功能和设计方面的改进作进一步说明的权利。然而中微半导体公司对于规格内容的使用不负责任。文中提到的应用其目的仅仅是用来做说明，中微半导体公司不保证和不表示这些应用没有更深入的修改就能适用，也不推荐它的产品使用在会由于故障或其它原因可能会对人身造成危害的地方。中微半导体公司的产品不授权适用于救生、维生器件或系统中作为关键器件。中微半导体公司拥有不事先通知而修改产品的权利，对于最新的信息，请参考我们的网站http://www.mcu.com.cn

应用注意事项

寄存器15H、28H、29H、2CH、2DH、2EH、2FH为系统保留寄存器，这些寄存器的值如果发生了变化，可能会导致芯片工作异常。因此建议用户使用时在程序中定时清零这些寄存器，避免在强干扰的时候，这些寄存器发生混乱，造成工作异常。

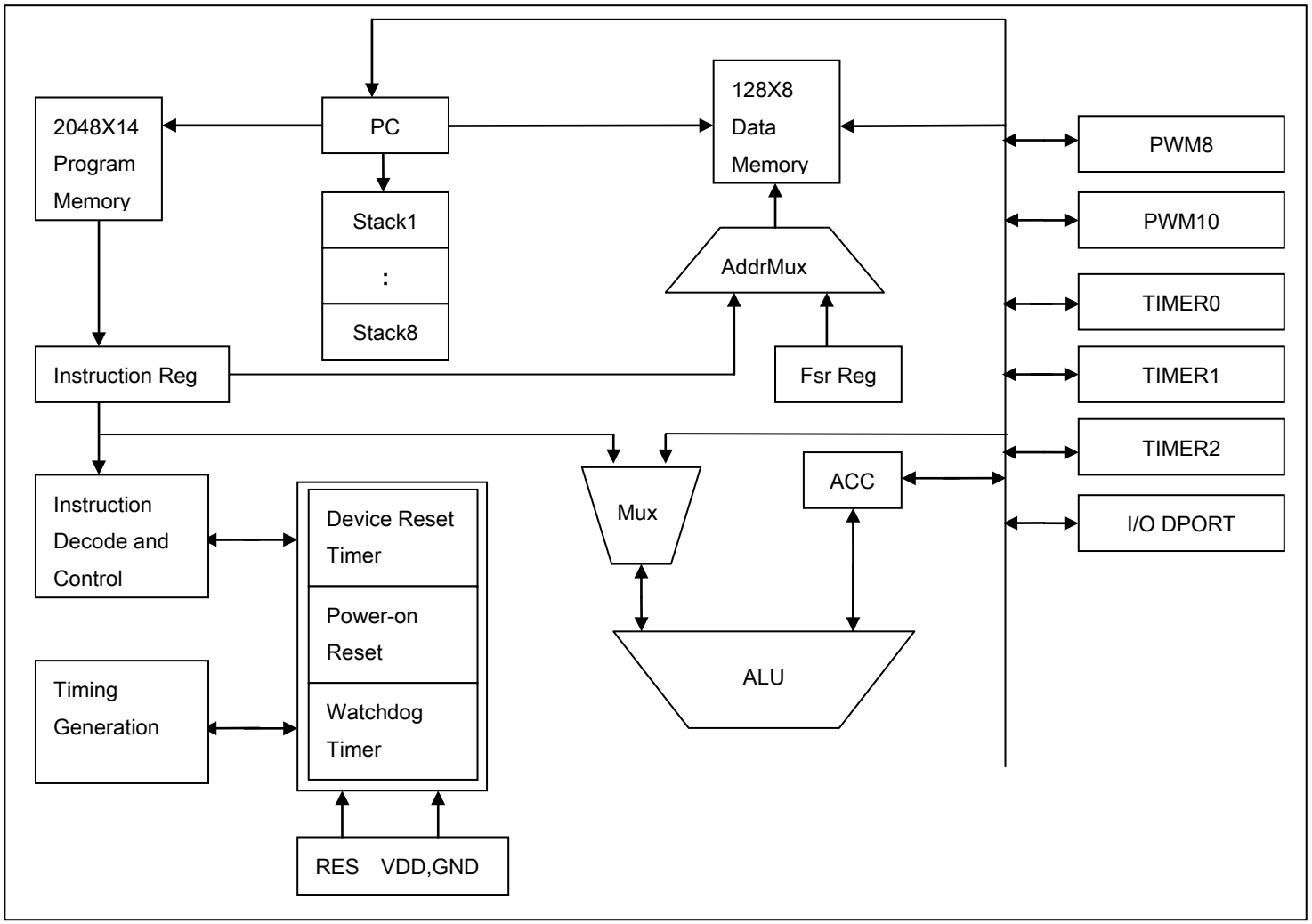
目录

1. 产品概述

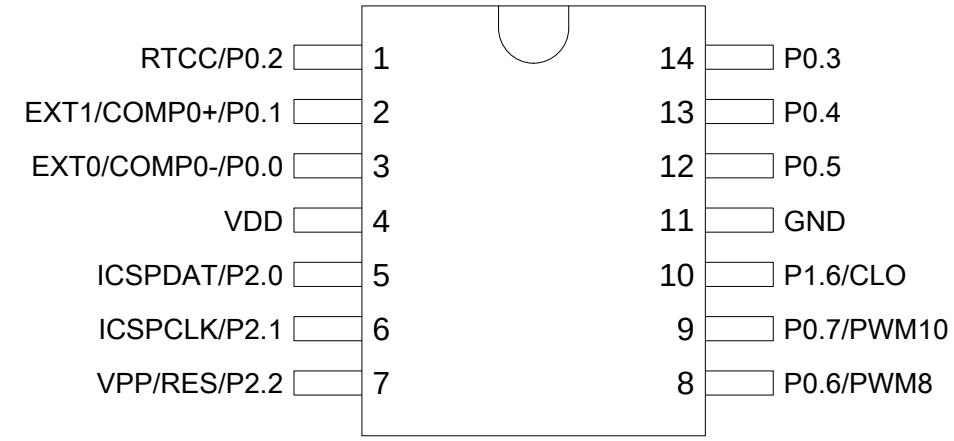
1.1 功能特性

- 内存
  FLASH: 2K*14
  RAM: 128（40 专用 RAM+88 通用 RAM）
- 8 级堆栈缓存器
- 简洁实用的指令系统(68 条指令)
- 内置低压侦测电路,内部复位电压：2.3V/3.6V
- 4 个中断源
  内部中断源 2 个： TMR1、TMR2
  外部中断源 2 个: EXT0、EXT1
- 3 个 8 位定时器
  TMR0、TMR1、TMR2
- 高频信号输出口（CLO）
  占空比可选择：25%、50%、75%。
- 2 个 PWM 输出口
  8 位 PWM
  10 位 PWM
- 工作电压范围：2.5V—5.5V
  工作温度范围：-20℃—85℃
- 指令周期（单指令或双指令周期）
- 查表功能
- 内置 WDT 定时器
- I/O 口配置
  P0：具有唤醒功能、上拉电阻选项。
  P1：具有上拉电阻选项
  P2：具有上、下拉电阻选项
- 两种工作模式
  正常模式、睡眠模式
- 2 路内置比较器
  正端可选择接内部标准电压
- 内置振荡模式
  8M/4M 可选
- 多种封装形式可供选择
  CMS69F012：DIP14、SOP14
  CMS69F013：DIP16、SOP16

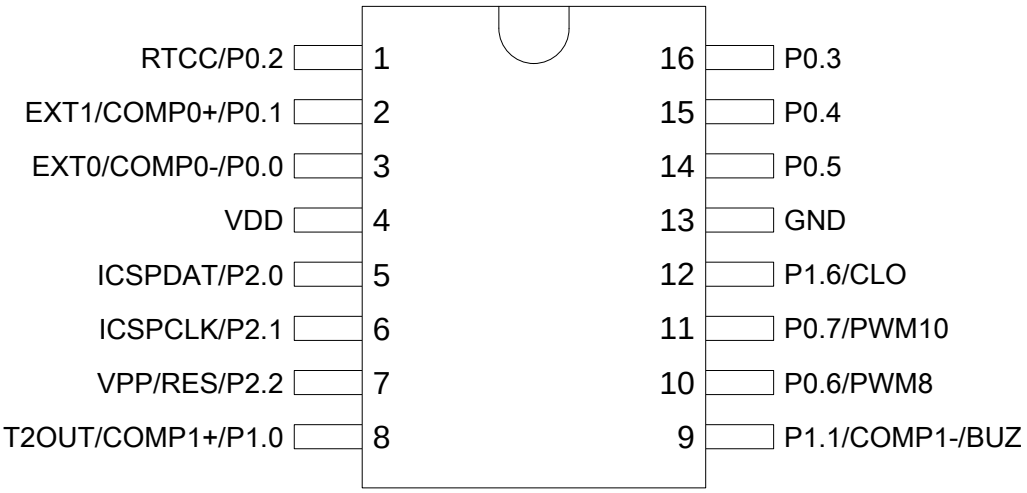
1.2 系统结构框图


2048X14
Program
Memory
PC
128X8
Data
Memory
PWM8
PWM10
TIMER0
TIMER1
TIMER2
I/O DPORT
Stack1
:
Stack8
AddrMux
Fsr Reg
Instruction Reg
Instruction
Decode and
Control
Device Reset
Timer
Power-on
Reset
Watchdog
Timer
Timing
Generation
Mux
ACC
ALU
RES
VDD,GND

1.3 管脚分布



CMS69F012



CMS69F013

1.4 管脚描述

管脚名称	IO 类型	管脚说明	共享引脚
P0.0-P0.7	I/O	可编程为：输入口，带上拉电阻输入口，推挽输出口；也可作为比较器输入口，外部中断输入口，PWM 输出口，带休眠唤醒功能。	RTCC,EXTINT0,EXTINT1 PWM0,PWM1 COMP0-,COMP0+
P1.0 P1.1 P1.6	I/O	可编程为：输入口，带上拉电阻输入口，推挽输出口,开漏输出口，开漏输出口（带上拉），比较器 1 输入口，CLO[PWM2]输出口。	COMP1-,COMP1+ T2OUT,BUZ,CLO
P2.0,P2.1	I/O	输入口，上/下拉电阻输入口，推挽输出口， 开漏输出口	ICSPDAT,ICSPCLK
P2.2	I/O	输入口，开漏输出口（需 CONFIG 使能）	RES
VDD，GND	P	电源电压输入脚，接地脚	--
RES	--	外部复位输入口	P2.2
PWM0	O	8 位 PWM 输出	P0.6
PWM1	O	10 位 PWM 输出	P0.7
T2OUT	O	TMR2 匹配输出	P1.0
BUZ	O	蜂鸣器输出	P1.1
CLO	O	占空比可选的系统时钟输出	P1.6
EXT0，EXT1	I	外部中断输入口	P0.0,P0.1
COMP0-,COMP0+	I	比较器 0 输入	P0.0,P0.1
COMP1-,COMP1+	I	比较器 1 输入	P1.0,P1.1
ICSPDAT	I/O	编程数据输入输出	P2.0
ICSPCLK	I	编程时钟输入	P2.1
VPP	I	编程高压输入	P2.2

1.5 系统配置寄存器

系统配置寄存器（CONFIG）是 MCU 初始条件的 FLASH 选项。它只能被 CMS 烧写器烧写，用户不能访问及操作。它包含了以下内容：

1、WDT(看门狗选择)
- ENABLE 打开看门口定时器
- DISABLE 关闭看门狗定时器

2、PROTECT(加密)
- DISABLE FLASH 代码不加密
- ENABLE FLASH 代码加密，加密后读出来的值将不确定

3、OSC TIME(起振时间)
- 9ms
- 2ms
- 560ms
- 200μs

4、LVR (低压侦测电路)
- ENABLE 打开低压侦测电路，选择内部复位，同时 P2.2 口作为普通 IO
- DISABLE 关闭低压侦测电路，选择外部复位，同时 P2.2 口作为复位口（低电平复位）

5、LVR _SEL(低压侦测电压选择)
- 2.3V
- 3.6V

6、RES_OUT(RES 口开漏输出选择)
- DISABLE 禁止 RES 口作为开漏输出口
- ENABLE 允许 RES 口作为开漏输出口

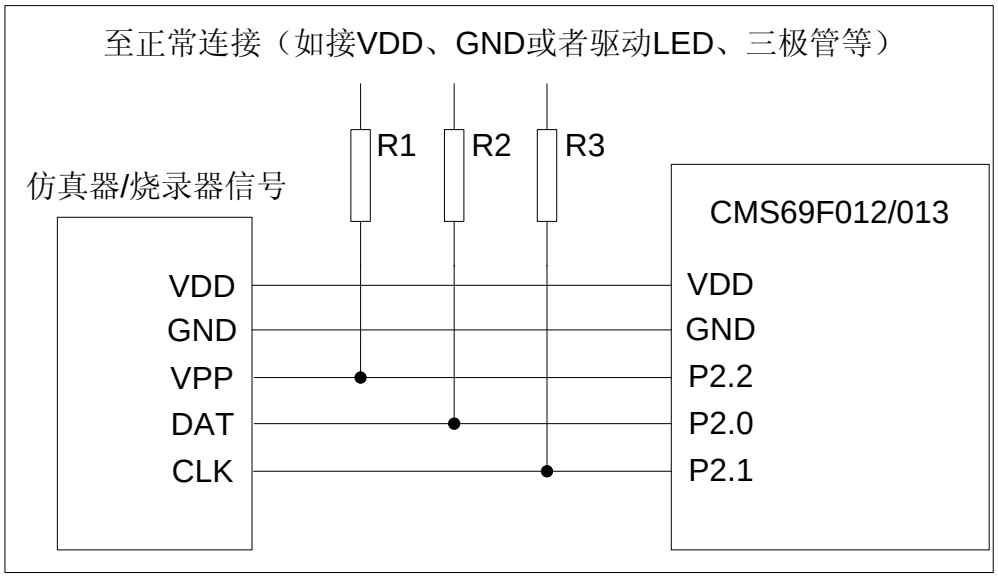
1.6 在线串行编程

可在最终应用电路中对 CMS69F012/013 单片机进行串行编程。编程可以简单地通过以下 5 根线完成：

- 电源线
- 接地线
- 编程电压线
- 数据线
- 时钟线

这使用户可使用未编程的器件制造电路板，而仅在产品交付前才对单片机进行编程。从而可以将最新版本的固件或者定制固件烧写到单片机中。

图 **1.6.1**： 典型的在线串行编程连接方式



上图中，R1、R2、R3 为电气隔离器件，常以电阻代替，其阻值如下：

R1≥47K，R2≥4.7K，R3≥4.7K。

DAT、CLK 管脚如果需要接电容，其容值不能大于 100pF。

2. 中央处理器(CPU)

2.1 内存

2.1.1 程序内存

FLASH:2K

0000H	复位向量	程序开始，跳转至用户程序
0001H	中断向量	中断入口，用户中断程序
0002H		
0003H		
0004H		
…		
…		用户程序区
…		
07FDH		
07FEH		
07FFH		
	跳转至复位向量 0000H	程序结束

2.1.1.1 复位向量(0000H)

CMS69F012/013系列单片机具有一个字长的系统复位向量（0000H）。具有以下四种复位方式：

- 上电复位
- 看门狗复位
- 外部复位
- 低压复位（LVR）

发生上述任一种复位后，程序将从 0000H 处重新开始执行，系统寄存器也都将恢复为默认值。根据 FLAGS 寄存器中的 PF和 TF 标志位的内容可以判断系统复位方式。下面一段程序演示了如何定义 FLASH 中的复位向量。

例：定义复位向量

```
        ORG     0000H       ;系统复位向量
        JP      START
        ORG     0002H       ;用户程序起始
START:
        …                   ;用户程序
        …
        END                 ;程序结束
```

2.1.1.2 中断向量

中断向量地址为 0001H。一旦有中断响应，程序计数器 PC 的当前值就会存入堆栈缓存器并跳转到 0001H 开始执行中断服务程序。所有中断都会进入0001H这个中断向量，具体执行哪个中断将由用户根据中断请求标志位寄存器（INT_FLAG）的位决定。下面的示例程序说明了如何编写中断服务程序。

例：定义中断向量，中断程序放在用户程序之后

```
                ORG     0000H       ;系统复位向量
                JP      START
                ORG     0001H       ;用户程序起始
INT_START:
                CALL    PUSH        ;保存 ACC 跟 FLAGS
                …                   ;用户中断程序
                …
INT_BACK:
                CALL    POP         ;返回 ACC 跟 FLAGS
                RETI                ;中断返回
START:
                …                   ;用户程序
                …
                END                 ;程序结束
```

注：由于 CMS69F012/013 系列芯片并未提供专门的出栈、压栈指令，故用户需自己保护中断现场。

例：中断入口保护现场

```
PUSH:
                LD      ACC_BAK,A   ;保存 ACC 至自定义寄存器 ACC_BAK
                SWAPA   FLAGS       ;状态寄存器 FLAGS 高低半字节互换
                LD      FLAGS_BAK,A ;保存至自定义寄存器 FLAGS_BAK
```

例：中断出口恢复现场

```
POP:
                SWAPA   FLAGS_BAK   ;将保存至 FLAGS_BAK 的数据高低半字节互换给 A
                LD      FLAGS,A     ;将 A 的值给状态寄存器 FLAGS
                SWAPR   ACC_BAK     ;将保存至 ACC_BAK 的数据高低半字节互换
                SWAPA   ACC_BAK     ;将保存至 ACC_BAK 的数据高低半字节互换给 ACC
```

2.1.1.3 查表

CMS69F012/013具有查表功能，FLASH 空间的任何地址都可做为查表使用。

相关指令：

- TABLE [R] 把表格内容的低字节送给寄存器R，高字节送到寄存器TABLE_DATAH(24H)。
- TABLEA 把表格内容的低字节送给累加器A，高字节送到寄存器TABLE_DATAH(24H)。

相关寄存器：

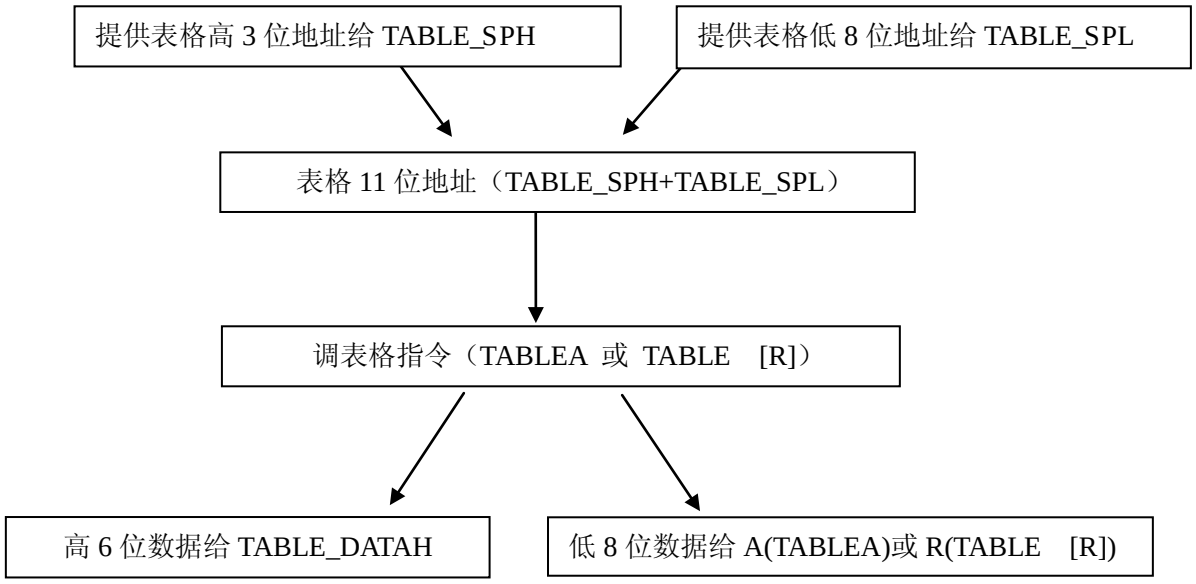
· TABLE_SPH(22H) 可擦写寄存器，用来指明表格高3位地址。

· TABLE_SPL(23H) 可擦写寄存器，用来指明表格低8位地址。

· TABLE_DATAH(24H) 只读寄存器，存放表格高字节内容。

注：在查表之前要先把表格地址写入 TABLE_SPH 和 TABLE_SPL 中。如果主程序和中断服务程序都用到查表指令，主程序中的 TABLE_SPH 的值可能会因为中断中执行的查表指令而发生变化，产生错误。也就是说要避免在主程序和中断服务程序中都使用查表指令。但如果必须这样做的话，我们可以在查表指令前先将中断禁止，在查表结束后再开放中断，以避免发生错误。

下面是表格调用的流程图：



表格调用流程图

下面例子给出了如何在程序中调用表格。

```
    ...                             ;上接用户程序
    LDIA    02H                     ;表格低位地址
    LD      TABLE_SPL,A
    LDIA    06H                     ;表格高位地址
    LD      TABLE_SPH,A
    TABLE   R01                     ;表格指令，将表格低 8 位(56H)给自定义寄存器 R01
    LD      A,TABLE_DATAH           ;将查表结果的高 6 位(34H)给累加器 A
    LD      R02,A                   ;将 ACC 值(34H)给自定义寄存器 R02
    ...                             ;用户程序

    ORG     0600H                   ;表格起始地址
    DW      1234H                   ;0600H 地址表格内容
    DW      2345H                   ;0601H 地址表格内容
    DW      3456H                   ;0602H 地址表格内容
    DW      0000H                   ;0603H 地址表格内容
```

2.1.1.4 跳转表

跳转表能够实现多地址跳转功能。由于 PCL 和 ACC 的值相加即可得到新的 PCL，因此，可以通过对 PCL 加上不同的 ACC 值来实现多地址跳转。ACC 值若为 n，PCL+ACC 即表示当前地址加 n，执行完当前指令后 PCL 值还会自加 1， 可参考以下范例。如果 PCL＋ACC 后发生溢出，PC不会自动进位，故编写程序时应注意。这样，用户就可以通过修改 ACC 的值轻松实现多地址的跳转。

例：正确的多地址跳转程序示例

```
    FLASH 地址
    0010H:      ADDR      PCL       ;ACC+PCL
    0011H:      JP        LOOP1     ;ACC=0，跳转至 LOOP1
    0012H:      JP        LOOP2     ;ACC=1，跳转至 LOOP2
    0013H:      JP        LOOP3     ;ACC=2，跳转至 LOOP3
    0014H:      JP        LOOP4     ;ACC=3，跳转至 LOOP4
    0015H:      JP        LOOP5     ;ACC=4，跳转至 LOOP5
    0016H:      JP        LOOP6     ;ACC=5，跳转至 LOOP6
```

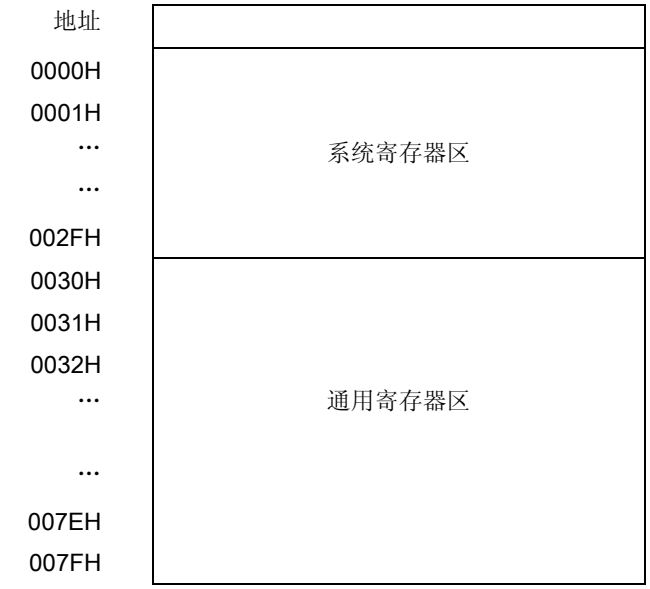
例：错误的多地址跳转程序示例

```
    FLASH 地址
    00FCH:      ADDR      PCL       ;ACC+PCL
    00FDH:      JP        LOOP1     ;ACC=0，跳转至 LOOP1
    00FEH:      JP        LOOP2     ;ACC=1，跳转至 LOOP2
    00FFH:      JP        LOOP3     ;ACC=2，跳转至 LOOP3
    0100H:      JP        LOOP4     ;ACC=3，跳转至 0000H 地址
    0101H:      JP        LOOP5     ;ACC=4，跳转至 0001H 地址
    0102H:      JP        LOOP6     ;ACC=5，跳转至 0002H 地址
```

注：由于 PCL 溢出不会自动向高位进位，故在利用 PCL 作多地址跳转时，一定要注意该段程序一定不能放在 FLASH 空间的分页处。

2.1.2 数据存储器

RAM:128 字节



数据存储器由128×8位组成，分为两个功能区间：特殊功能寄存器（48×8）和通用 数据存储器（80×8）。数据存储器单元大多数是可读/写的，但有些只读的。特殊功能寄存器地址为00H到2FH，通用数据寄存器地址为30H 到7FH。

寄存器08H、0EH、0FH、25H、26H、27H、2AH、2BH这8个地址没有特殊功能，可以当做通用数据寄存器使用。

2.1.2.1 通用数据存储器

RAM 的 0030H～007FH 地址，以及 08H、0EH、0FH、25H、26H、27H、2AH、2BH 这 8 个地址，属于用户可自由定义的通用寄存器区，在此区域的寄存器上电为随机值。当系统上电工作后，若发生意外复位（非上电复位），此区域寄存器保持原来值不变。

2.1.2.2 系统专用数据存储器

系统专用数据存储器表

地 址	名 称	说 明
00H	IAR	间接寻址寄存器
01H	TMR0	内部定时/计数器
02H	PCL	程序指针 PC 低 8 位
03H	FLAGS	系统状态标志寄存器
04H	MP	间接寻址指针
05H	P0	P0 IO 口数据寄存器
06H	P1	P1 IO 口数据寄存器
07H	P2	P2 IO 口数据寄存器
08H	----	通用寄存器（用户可自由操作）
09H	P0CL	P0 IO 口功能控制寄存器
0AH	P0CH	P0 IO 口功能控制寄存器
0BH	P1CL	P1 IO 口功能控制寄存器
0CH	P1CH	P1 IO 口功能控制寄存器
0DH	P2C	P2 IO 口功能控制寄存器
0EH-0FH	----	通用寄存器（用户可自由操作）
10H	SYS_GEN	中断总使能
11H	INT_EN	中断控制寄存器使能位
12H	INT_FLAG	中断控制寄存器标志位
13H	INT_EXT	外部中断边沿触发设置寄存器
14H	----	未用，保留（只读）
15H	----	未用，保留，请在使用时清零
16H	TMR1	定时计数器 1
17H	TMR1C	定时计数器 1 控制寄存器
18H	T2CNT	TMR2 计数器（只读）
19H	T2CON	TMR2 控制寄存器
1AH	T2DATA	TMR2 资料寄存器
1BH	----	未用，保留（只读）
1CH	PWM8DATA	8 位 PWM 数据寄存器
1DH	PWM8CON	8 位 PWM 控制寄存器及 CLO 输出控制
1EH	PWM10CON	10 位 PWM 控制寄存器
1FH	PWM10DATA	10 位 PWM 数据寄存器
20H	COMPCON	比较器控制寄存器
21H	BUZCON	蜂鸣器驱动控制寄存器
22H	TABLE_SPH	查表高 3 位地址
23H	TABLE_SPL	查表低 8 位地址
24H	TABLE_DATAH	查表高 6 位结果

系统专用数据存储器表（续）

25-27H	----	通用寄存器（用户可自由操作）
28H	----	未用，保留，请在使用时清零
29H	----	未用，保留，请在使用时清零
2A-2BH	----	通用寄存器（用户可自由操作）
2C-2FH	----	未用，保留，请在使用时清零

注：寄存器15H、28H、29H、2CH、2DH、2EH、2FH为系统保留寄存器，用户在使用时需要在程序中定时清零这些寄存器。

2.2 寻址方式

2.2.1 直接寻址

通过工作寄存器（ACC）来对 RAM 进行操作。

例：ACC 的值送给 30H 寄存器

```
        LD      30H,A
```

例：30H 寄存器的值送给 ACC

```
        LD      A,30H
```

2.2.2 立即寻址

把立即数传给工作寄存器（ACC）

例：立即数 12H 送给 ACC

```
        LDIA    12H
```

2.2.3 间接寻址

数据存储器能被直接或间接寻址。通过 IAR 寄存器可间接寻址，IAR 不是物理寄存器。当对 IAR 进行存取时，它会根据 MP 寄存器内的值作为地址，并指向该地址的寄存器，因此在设置了 MP 寄存器后，就可把 IAR 寄存器当作目的寄存器来存取。间接读取 IAR（MP=0）将产生 00H。间接写入 IAR 寄存器，将导致一个空运作。以下例子说明了程序中间接寻址的用法。

例：MP 及 IAR 的应用

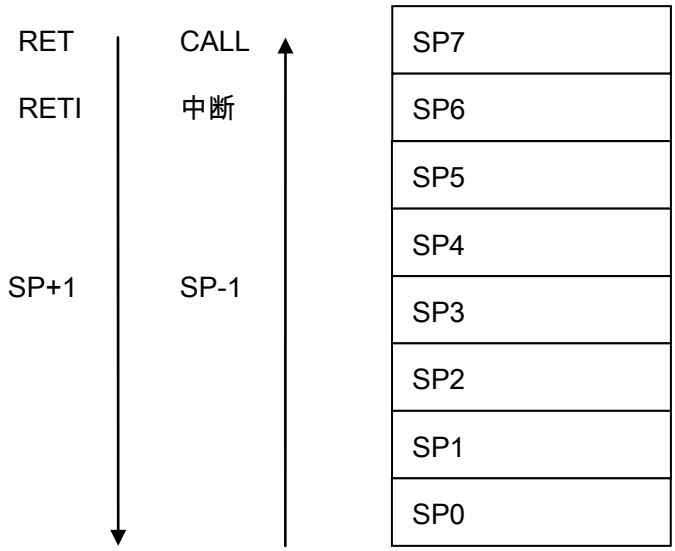
```
        LDIA    30H
        LD      MP,A        ;间接寻址指针指向 30H
        CLR     IAR         ;清零 IAR 实际是清零 MP 指向的 30H 地址 RAM
```

例：间接寻址清 RAM(30H-7FH)举例：

```
        LDIA    2FH
        LD      MP,A        ;间接寻址指针指向 2FH
LOOP:
        INCR    MP          ;地址加 1，初始地址为 30H
        CLR     IAR         ;清零 MP 所指向的地址
        LDIA    7FH
        SUBA    MP
        SNZB    FLAGS,C     ;一直清零至 MP 地址为 7FH
        JP      LOOP
```

2.3 堆栈

CMS69F012/013 的堆栈缓存器共 8 层，堆栈缓存器既不是数据存储器的一部分，也不是程序内存的一部分，且既不能被读出，也不能被写入。对它的操作通过堆栈指针（SP）来实现，堆栈指针（SP）也不能读出或写入，当系统复位后堆栈指针会指向堆栈顶部。当发生子程序调用及中断时的程序计数器（PC）值被压入堆栈缓存器，当从中断或子程序返回时将数值返回给程序计数器（PC）,下图说明其工作原理。



堆栈缓存器的使用将遵循“先进后出”的原则。

注：堆栈缓存器只有 8 层，如果堆栈已满，并且发生不可屏蔽的中断，那么只有中断标志位会被记录下来，而中断响应则会被抑制，直到堆栈指针发生递减，中断才会被响应，这个功能可以防止中断使堆栈溢出，同样如果堆栈已满，并且发生子程序调用，那么堆栈将会发生溢出，首先进入堆栈的内容将会丢失，只有最后 8 个返回地址被保留，故用户在写程序时应注意此点，以免发生程序走飞。

2.4 工作寄存器（ACC）

2.4.1 概述

ALU 是 8BIT 宽的算术逻辑单元，MCU 所有的数学、逻辑运算均通过它来完成。它可以对数据进行加、减、移位元及逻辑运算；ALU 也控制状态位（FLAGS 状态寄存器中），用来表示运算结果的状态。

A 寄存器是一个 8-BIT 的寄存器，ALU 的运算结果可以存放在此，它并不属于数据存储器的一部分而是位于 CPU 中供 ALU 在运算中使用，因此不能被寻址，只能通过所提供的指令来使用。

2.4.2 ACC 应用

例：用 ACC 做数据传送

```
        LD      A,R01       ;将寄存器 R01 的值赋给 ACC
        LD      R02,A       ;将 ACC 的值赋给寄存器 R02
```

例：用 ACC 做立即寻址目标操作数

```
        LDIA    30H         ;给 ACC 赋值 30H
        ANDIA   30H         ;将当前 ACC 的值跟立即数 30H 进行“与”操作，
                            ;结果放入 ACC
        XORIA   30H         ;将当前 ACC 的值跟立即数 30H 进行“异或”操作，
                            ;结果放入 ACC
```

例：用 ACC 做双操作数指令的第一操作数

```
        HSUBA   R01         ;ACC-R01，结果放入 ACC
        HSUBR   R01         ;ACC-R01，结果放入 R01
```

例：用 ACC 做双操作数指令的第二操作数

```
        SUBA    R01         ;R01-ACC，结果放入 ACC
        SUBR    R01         ; R01-ACC，结果放入 R01
```

2.5 程序状态寄存器(FLAGS)

寄存器 FLAGS 中包含 ALU 运算状态信息、系统复位状态信息等。其中，位 TF 和PF 显示系统复位状态信息，包括上电复位、外部复位和看门狗复位等；位 C、HC 和 Z 显示 ALU 的运算信息。

03H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
FLAGS	TF	PF	---	---	OV	Z	HC	C
读写	R	R	---	---	R/W	R/W	R/W	R/W
复位值	X	X	---	---	X	X	X	X

bit 7 TF： WDT溢出标志
1 = 上电或是执行了CLRWDT指令或SLEEP指令
0 = 发生了WDT超时

bit 6 PF：低功耗标志
1 = 上电或执行了CLRWDT指令
0 = 执行了STOP指令

bit 5-4 未用，使用时清零

bit 3 OV： 数学运算溢出标志位
1 =运算结果高两位进位状态异或结果为1
0 =运算结果高两位进位状态异或结果为0

bit 2 Z： 结果为零位
1 = 算术或逻辑运算的结果为零
0 = 算术或逻辑运算的结果不为零

bit 1 HC： 半进位/ 借位位
1 = 发生了结果的第4 低位向高位进位
0 = 结果的第4 低位没有向高位进位

bit 0 C： 进位/ 借位位
1 = 结果的最高位发生了进位
0 = 结果的最高位没有发生进位

FLAGS 寄存器中除了 TF 和 PF 位，其它的都可以用指令设置或者清零。例如指令：“CLR FLAGS”的结果是 FLAGS=“UU000100”，“U”指未改变， 而不是想象中的全零。也就是说执行指令后，PF 和 TF 的值保持不变，而 Z 标志位因清零而置一，所以若需要改变 FLAGS 的值 建议使用“SETB”、“CLRB”、“LD R,A”这几条指令，因为这几条指令不会影响状态标志位。

TF 和 PF 标志位可反映出芯片复位的原因，下面列出影响 TF、PF 的事件及各种复位后 TF、PF 的状态。

事件	TF	PF
电源上电	1	1
WDT 溢出	0	X
STOP 指令	1	0
CLRWDT 指令	1	1
休眠	1	0

影响 PF、TF 的事件表

TF	PF	复位原因
0	0	WDT 溢出唤醒休眠 MCU
0	1	WDT 溢出非休眠态
1	0	按键或外部复位唤醒休眠 MCU
1	1	电源上电
X	X	外部低电平复位

复位后 TF/PF 的状态

2.6 预分频器(OPTION)

预分频器（OPTION）寄存器是 6BIT，只可写的寄存器，它包含各种用于配置 TMR0/WDT 预分频器和 TMR0 的控制位。通过执行 OPTION 指令可将累加寄存器的内容传送到预分频器。

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
OPTION	---	---	T0CS	T0SE	PSA	PS2	PS1	PS0
读写			W	W	W	W	W	W
复位值	X	X	1	1	1	1	1	1

bit 7 未用

bit 6 未用

bit 5 **T0CS**：TMR0 时钟源选择位

0. 内部时钟
1. 外部时钟（RTCC 口输入波形）

bit 4 **T0SE**：RTCC 信号触发源选择位

0. 上升沿触发
1. 下降沿触发

bit 3 **PSA**：预分频器分配位

0. 分给 TMR0 用
1. 分给 WDT 用

bit 2～bit 0 **PS2～PS0**：预分配参数配置位

PS2—PS1—PS0	TMR0 分频比	WDT 分频比
000	1：2	1：1
001	1：4	1：2
010	1：8	1：4
011	1：16	1：8
100	1：32	1：16
101	1：64	1：32
110	1：128	1：64
111	1：256	1：128

预分频寄存器实际上是一个8位的计数器，用于监视寄存器WDT时，是作为一个后分频器；用于定时器/计数器时，作为一个预分频器，通常统称作预分频器。在片内只有一个物理的分频器，只能用于WDT/TMR0两者之一，不能同时使用。也就是说，若用于TMR0，WDT就不能使用预分频器，反之亦然。

当用于WDT时，CLRWDT指令将同时对预分频器和WDT定时器清零。

当用于TMR0时，有关写入TMR0的所有指令（如：CLR TMR0,SETB TMR0,1等）都会对预分频器清零。

由TMR0还是WDT使用预分频器，完全由软件控制。它可以动态改变。为了避免出现不该有的芯片复位，当从TMR0换为WDT使用时，应该执行以下指令。

```
        CLR     TMR0            ;TMR0 清零
        CLRWDT                  ;WDT 清零
        LDIA    B'00xx1xxx'     ;设置新的预分频器
        OPTION
```

将预分频器从分配给 WDT 切换为分配给 TMR0 模块，应该执行以下指令

```
        CLRWDT                  ;WDT 清零
        LDIA    B'00xx0xxx'     ;设置新的预分频器
        OPTION
```

2.7 程序计数器（PC）

程序计数器(PC)控制程序内存 FLASH 中的指令执行顺序，它可以寻址整个 FLASH 的范围，取得指令码后，程序计数器(PC)会自动加一，指向下一个指令码的地址。但如果执行跳转、条件跳转、向 PCL 赋值、子程序调用、初始化复位、中断、中断返回、子程序返回等操作时，PC 会加载与指令相关的地址而不是下一条指令的地址。

当遇到条件跳转指令且符合跳转条件时，当前指令执行过程中读取的下一条指令将会被丢弃，且会插入一个空指令操作周期，随后才能取得正确的指令。反之，就会顺序执行下一条指令。

程序计数器(PC)是 11-BIT 宽度，低 8 位通过 PCL（02H）寄存器用户可以访问，高 3 位用户不能访问。可容纳 2Kx14 位程序地址。对 PCL 赋值将会产生一个短跳转动作，跳转范围为当前页的 256 个地址。

注：由于程序员不能操作 PC 的高 3 位，所以当程序员在利用 PCL 作短跳转时应注意当前 PC 的位置，以免发生错误的程序跳转。

下面给出几种特殊情况的 PC 值

复位时	PC=0000；
中断时	PC=0001(原来的 PC+1 会被自动压入堆栈)；
CALL 时	PC=程序指定地址(原来的 PC+1 会被自动压入堆栈)；
RET、RETI、RETI 时	PC=堆栈出来的值；
操作 PCL 时	PC[10：8]不变，PC[7:0]=用户指定的值；
JP 时	PC=程序指定的值；
其它指令	PC=PC+1；

2.8 看门狗计数器(WDT)

看门狗定时器（Watch Dog Timer）是一个片内自振式的 RC 振荡定时器，无需任何外围组件，即使芯片的主时钟停止工作，WDT 也能保持计时。WDT 计时溢出将产生复位。在 CMS69F012/013 系列芯片中集成了 CONFIG 选项，可将其置“0”来使 WDT 不起作用，详见 1.5 章 CONFIG 烧写的选择。

2.8.1 WDT 周期

WDT 有一个基本的溢出周期 18ms(无预分频器)，假如你需要更长时间的 WDT 周期，可以把预分频器分配给 WDT，最大分频比为 1：128，此时 WDT 的周期约为 2.3s。WDT 的溢出周期将受到环境温度，电源电压等参数影响。

“CLRWDT”和“STOP”指令 将清除 WDT 定时器以及预分频器里的计数值（当预分频器分配给 WDT 时）。WDT 一般用来防止系统失控，或者可以说是用来防止单片机程序失控。在正常情况下，WDT 应该在其溢出前被“CLRWDT”指令清零，以防止产生复位。如果程序由于某种干扰而失控，那么不能在 WDT 溢出前执行“CLRWDT”指令，就会使 WDT 溢出而产生复位。使系统重启而不至于失去控制。若是 WDT 溢出产生的复位，则状态寄存器（FLAGS）的“TF”位会被清零，用户可根据此位来判断复位是否是 WDT 溢出所造成的。

注：1. 若使用 WDT 功能，一定要在程序的某些地方放置“CLRWDT”指令，以保证在 WDT 溢出前能被清零。否者会使芯片不停的复位，造成系统无法正常工作。
2. 不能在中断程序中对 WDT 进行清零，否则无法侦测到主程序“跑飞”的情况。
3. 程序中应在主程序中有一次清 WDT 的操作，尽量不要在多个分支中清零 WDT，这种架构能最大限度发挥看门狗计数器的保护功能。
4. 看门狗计数器不同芯片的溢出时间有一定差异，所以设置清 WDT 时间时，应与 WDT 的溢出时间有较大的冗余，以避免出现不必要的 WDT 复位。

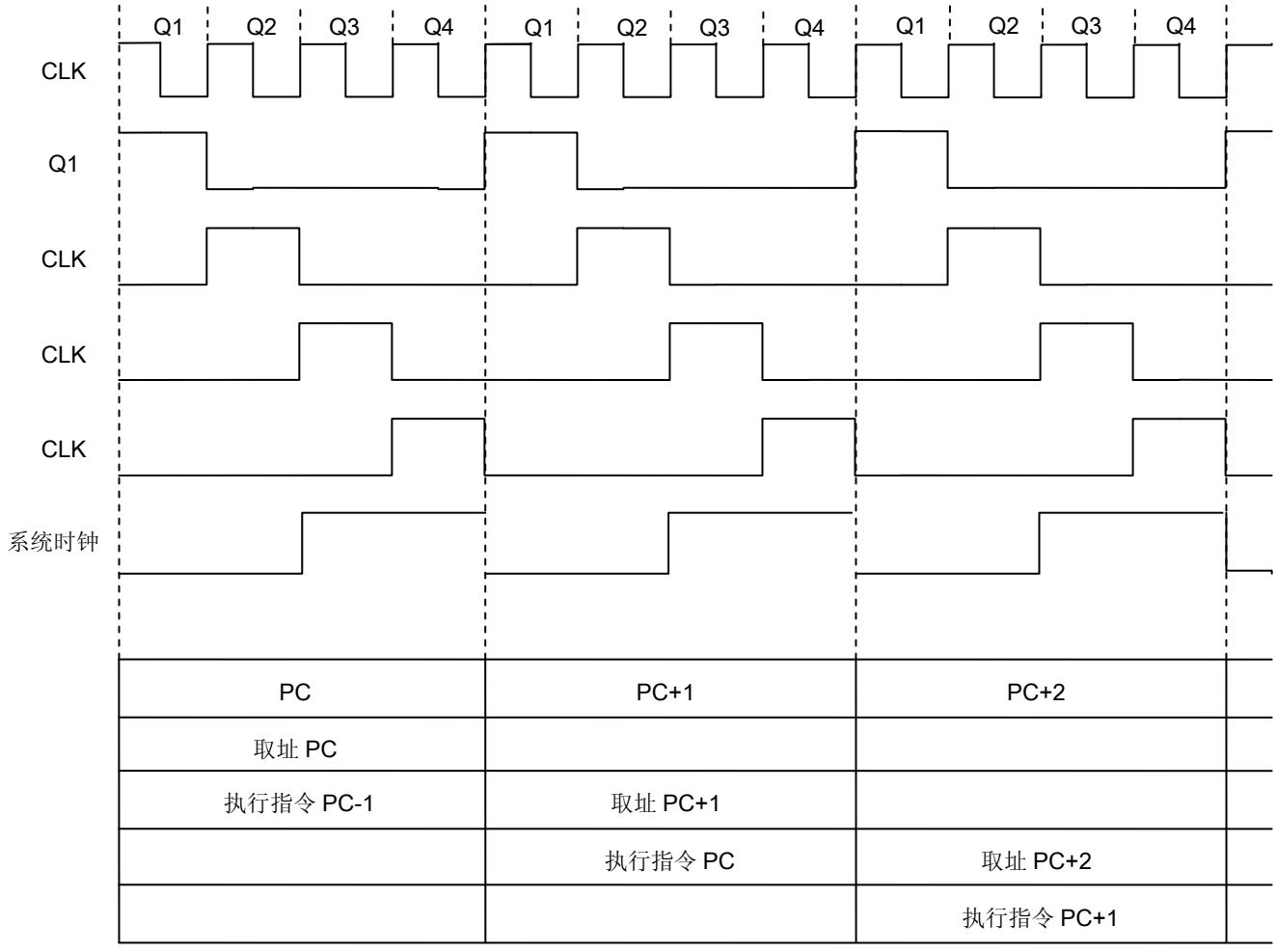
3. 系统时钟

3.1 概述

时钟信号从 OSC1 引脚输入后（或者由内部振荡产生），在片内产生 4 个非重迭正交时钟信号，分别称作 Q1、Q2、Q3、Q4。在 IC 内部每个 Q1 使程序计数器（PC）增量加一，Q4 从程序存储单元中取出该指令，并将其锁存在指令寄存器中。在下一个 Q1 到 Q4 之间对取出的指令进行译码和执行，也就是说 4 个时钟周期才会执行一条指令。下图表示时钟与指令周期执行时序图。

一个指令周期含有 4 个 Q 周期，指令的执行和获取是采用流水线结构，取指占用一个指令周期，而译码和执行占用另一个指令周期，但是由于流水线结构，从宏观上看，每条指令的有效执行时间是一个指令周期。如果一条指令引起程序计数器地址发生改变（例如 JP）那么预取的指令操作码就无效，就需 要两个指令周期来完成该条指令，这就是对 PC 操作指令都占用两个时钟周期的原因。

时钟与指令周期时序图



下面列出振荡频率与指令速度的关系

频率	双指令周期	单指令周期
1MHz	8μs	4μs
2MHz	4μs	2μs
4MHz	2μs	1μs
8MHz	1μs	500ns

3.2 系统振荡器

CMS69F012/013 只有 1 种振荡方式，内部 RC 振荡，其设计频率为 8M/4M 可选。

3.3 起振时间

起振时间（OSC TIME）是指从芯片复位到芯片振荡稳定这段时间，可由内部烧写 CONFIG 选项设置为 9ms、2ms、560μs、200μs；具体设置参数请参照 1.5 烧写选项设定章节。

注：无论芯片是电源上电复位，还是其它原因引起的复位，都会存在这个起振时间。

4. 复位

CMS69F012/013 可用如下 5 种复位方式：

- 上电复位
- 低电压复位（LVR 使能）
- 正常工作下的看门狗溢出复位
- 休眠模式下的看门狗溢出复位
- 复位口低电平（LVR 禁止）

上述任意一种复位发生时，所有的系统寄存器将恢复默认状态，程序停止运行，同时程序计数器 PC 清零，复位结束后程序从复位向量 0000H 开始运行。FLAGS 的 PF 和 TF 标志位能够给出系统复位状态的信息，（详见 FLAGS 的说明），用户可根据 PF 和 TF 的状态，控制程序运行路径。

任何一种复位情况都需要一定的响应时间，系统提供完善的复位流程以保证复位动作的顺利进行。

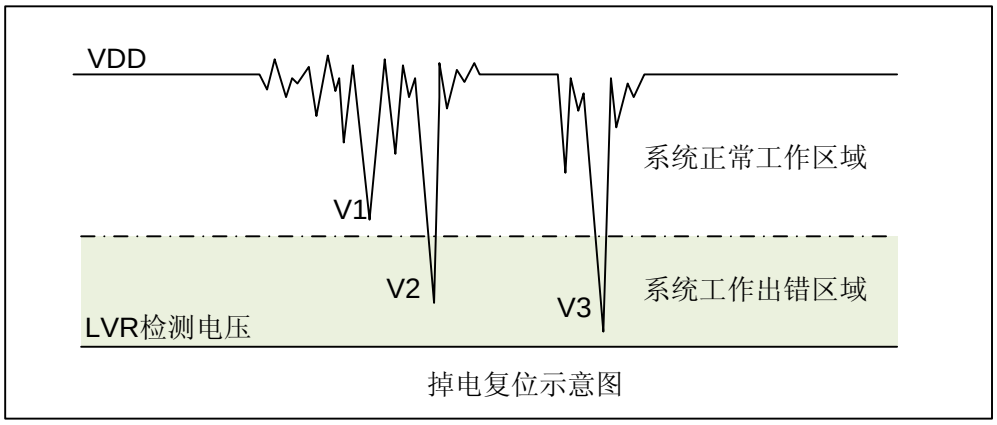
4.1 上电复位

上电复位与 LVR操作密切相关。系统上电的过程呈逐渐上升的曲线形式，需要一定时间才能达到正常电平值。下面 给出上电复位的正常时序：

- 上电：系统检测到电源电压上升并等待其稳定；
- 外部复位（仅限于外部复位引脚使能状态）：系统检测外部复位引脚状态。如果不为高电平，系统保持复位状态直到外部复位引脚释放；
- 系统初始化：所有的系统寄存器被置为初始值；
- 振荡器开始工作：振荡器开始提供系统时钟；
- 执行程序：上电结束，程序开始运行。

4.2 掉电复位

掉电复位针对外部因素引起的系统电压跌落情形（例如，干扰或外部负载的变化）。当使用外部复位时，掉电复位可能会引起系统工作状态不正常或程序执行错误，电压跌落可能会进入系统死区，系统死区意味着电源不能满足系统的最小工作电压要求。



掉电复位示意图

上图是一个典型的掉电复位示意图。图中，VDD 受到严重的干扰，电压值降的非常低。虚线以上区域系统正常工作，在虚线以下的区域内，系统进入未知的工作状态，这个区域称作死区。当 VDD 跌至 V1 时，系统仍处于正常状态；当VDD 跌至 V2和V3时，系统进入死区，则容易导致出错。

以下情况系统可能进入死区：

DC 运用中：

DC 运用中一般都采用电池供电，当电池电压过低或单片机驱动负载时，系统电压可能跌落并进入死区。这时，电源 不会进一步下降到 LVD 检测电压，因此系统维持在死区。

AC 运用中：

系统采用 AC 供电时，DC 电压值受 AC 电源中的噪声影响。当外部负载过高，如驱动马达时，负载动作产生的干扰也影响到 DC 电源。VDD 若由于受到干扰而跌落至最低工作电压以下时，则系统将有可能进入不稳定工作状态。

在 AC 运用中，系统上、下电时间都较长。其中，上电时序保护使得系统正常上电，但下电过程却和DC 运用中情形类似，AC 电源关断后，VDD 电压在缓慢下降的过程中易进入死区。

如上图所示，系统正常工作电压区域一般高于系统复位电压，同时复位电压由低电压检测（LVR）电平决定。当系统执行速度提高时，系统最低工作电压也相应提高，但由于系统复位电压是固定的，因此在系统最低工作电压与系统复位电压之间就会出现一个电压区域，系统不能正常工作，也不会复位，这个区域即为死区。

4.2.1 掉电复位的改进办法

如何改进系统掉电复位性能，以下给出几点建议

- 开启 MCU 的低压侦测功能
- 开启看门狗定时器
- 降低系统的工作频率
- 如果采用外部复位电路，抬高外部复位电压
- 增大电压下降斜率

开启 MCU 的低压侦测功能

CMS69F012/013 系列芯片，内部集成了低压侦测（LVR）功能，可由烧写 CONFIG 控制，详见 1.5 章关于烧写 CONFIG 选择说明。开启 LVR 功能时，当系统电压跌至低于 LVR 电压时，LVR 被触发，系统复位。由于 LVR 电压始终高于芯片的最低工作电压，因此不会存在系统工作死区。

看门狗定时器

看门狗定时器用于保证程序正常运行，当系统进入工作死区或者程序运行出错时，看门狗定时器会溢出，系统复位。

降低系统的工作速度

系统工作频率越快，系统最低工作电压越高。从而增大了工作死区的范围，降低系统工作速度就可以降低最低工作电压，从而有效的减小系统工作在死区电压的机率。

如果采用外部复位电路，抬高外部复位电压

尽量使用芯片内部复位(即将 LVR 打开)，如果特殊情况需要用外部复位时，建议使用三极管复位，并将复位电压设置至芯片的最低工作电压以上。具体请参照 4.5 章外部复位电路的说明。

增大电压下降斜率

此方法可用于系统工作在 AC 供电的环境，一般 AC 供电系统，系统电压在掉电过程中下降很缓慢，这就会造成芯片较长时间工作在死区电压，此时若系统重新上电，芯片工作状态可能出错，建议在芯片电源与地线间加一个放电电阻，以便让 MCU 快速通过死区，进入复位区，避免芯片上电出错可能性。

4.3 看门狗复位

看门狗复位是系统的一种保护设置。在正常状态下，由程序将看门狗定时器清零。若出错，系统处于未知状态，看门狗定时器溢出，此时系统复位。看门狗复位后，系统重启进入正常状态。

看门狗复位的时序如下：

- 看门狗定时器状态：系统检测看门狗定时器是否溢出，若溢出，则系统复位；
- 初始化：所有的系统寄存器被置为默认状态；
- 振荡器开始工作：振荡器开始提供系统时钟；
- 程序：复位结束，程序开始运行。

关于看门狗定时器的应用问题请参看2.8 WDT应用章节。

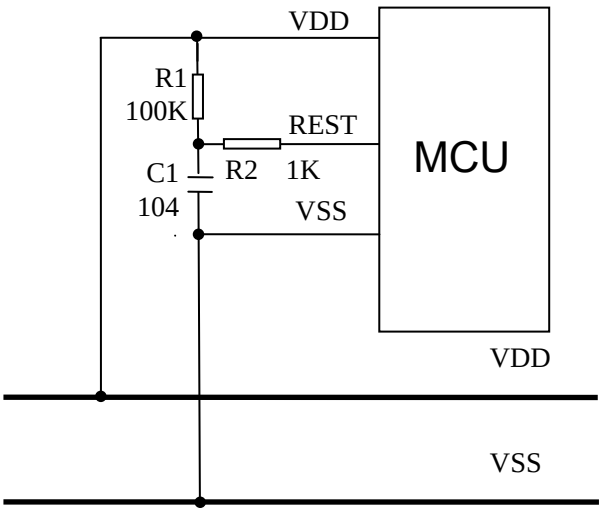
4.4 复位口低电平复位

当 LVR 功能被屏蔽时，REST 口低电平会使 MCU 进入复位态，具体工作时序如下：

- REST 口为低电平
- REST 口电平从低变为高，若一直为低则芯片一直处于复位态。
- 初始化：所有的系统专用寄存器被置为默认状态；
- 振荡器开始工作：振荡器开始提供系统时钟；
- 程序：复位结束，程序开始运行。

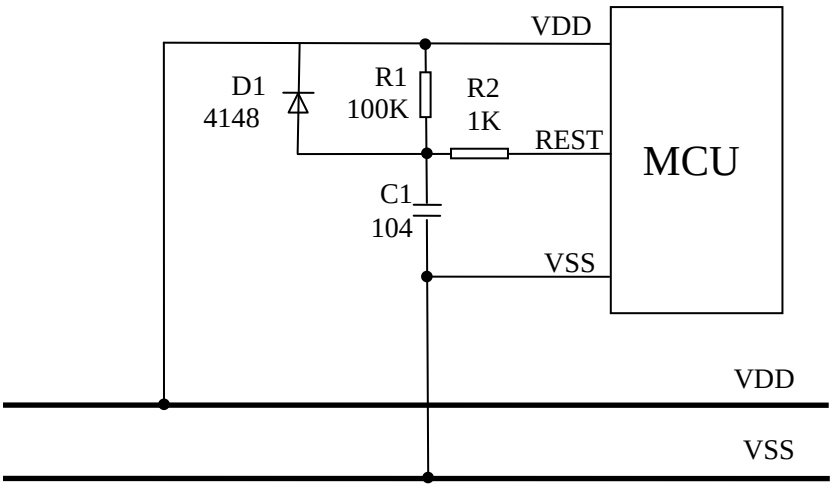
4.5 基本外部复位电路

4.5.1 RC 复位电路



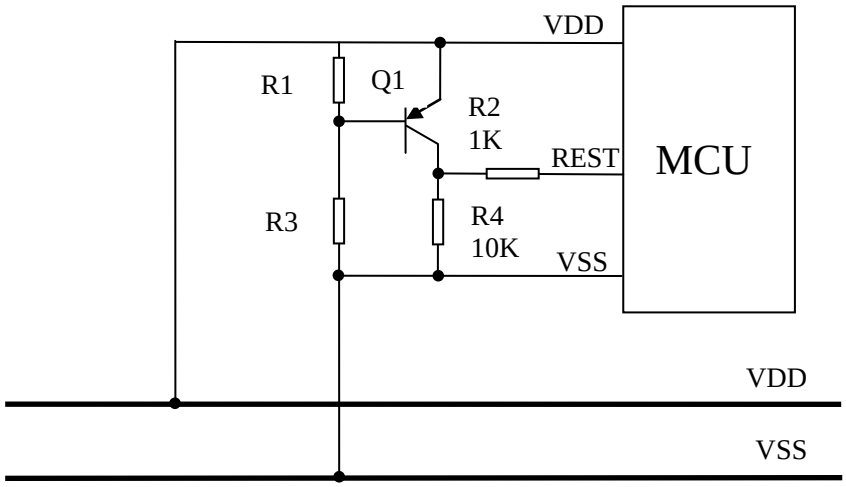
上图为一个由电阻R1和电容C1组成的基本RC复位电路，它在系统上电的过程中能够为复位引脚提供一个缓慢上升的复位信号。这个复位信号的上升速度低于VDD的上电速度，为系统提供合理的复位时序，当复位引脚检测到高电平时，系统复位结束，进入正常工作状态，当系统选择LVR禁止模式时，用户可选用此电路以提高复位的可靠性。

4.5.2 二极管及 RC 复位电路



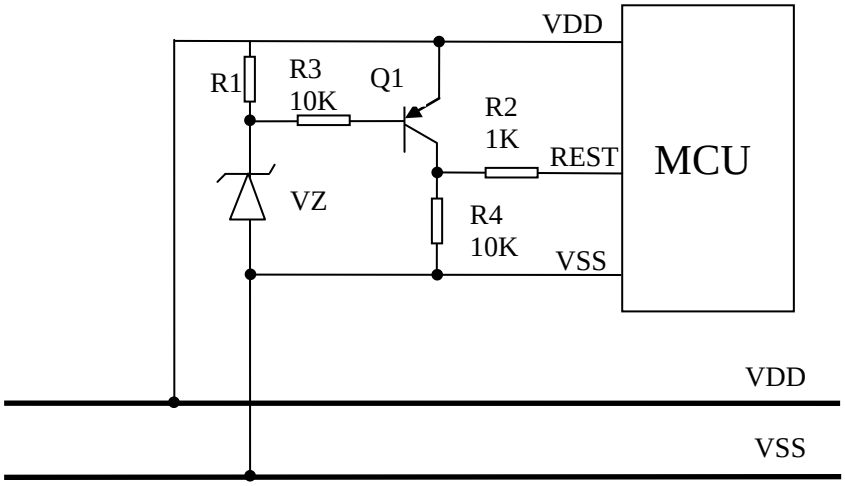
上图中，R1和C1同样是为复位引脚提供输入信号。对于电源异常情况，二极管正向导通使C1快速放电并与 VDD保持一致，避免复位引脚持续高电平、系统无法正常复位。

4.5.3 三极管复位电路



电压偏置复位电路是一种简单的LVR电路，基本上可以解决掉电复位问题。这种复位电路中，R1和 R2 构成分压电路，当VDD高于和等于分压值“0.7V x (R1 + R3)/ R1”时，三极管集电极C输出高电平，单片机正常工作；VDD低于“0.7V x (R1 + R3) / R1”时，集电极C输出低电平，单片机复位。对于不同应用需求，选择适当的分压电阻。单片机复位引脚上电压的变化与VDD电压变化之间的差值为 0.7V。如果VDD跌落并低于复位引脚复位检测值，那么系统将被复位。如果希望提升电路复位电平，可将分压电阻设置为 R3>R1， 并选择VDD与集电极之间的结电压高于0.7V。分压电阻R1和R3在电路中要耗电，此处的功耗必须计入整个系统的功耗中。

4.5.4 稳压二极管复位电路



稳压二极管复位电路是一种简单的 LVR 电路，基本上可以解决掉电复位问题。如上图电路中，利用稳压管的击穿电压作为电路复位检测值，当 VDD 高于“Vz + 0.7V”时，三极管集电极输出高电平，单片机正常工作；当 VDD 低于“Vz + 0.7V”时，三极管集电极输出低电平，单片机复位。稳压管规格不同则电路复位检测值不同，根据电路的要求选 择合适的二极管。

注：上述外部复位电路中，R2 电阻不能取消，以提高 REST 口的抗 EMC 及 ESD 能力。

5. 系统工作模式

CMS69F012/013 系列 MCU 存在两种工作模式，一种是正常工作模式，一种是睡眠工作模式。在正常工作模式下，各个功能模块均处于工作状态，在休眠状态下，系统时钟停止，芯片保持原来的状态不变，此时 WDT 的功能若没有被烧写 CONFIG 选项禁止，则 WDT 定时器一直工作。

5.1 休眠模式

省电模式是被 STOP 指令启动的，在省 电模式状态下，系统振荡停止，以减小功耗，且所有外围停止工作。省电模式可由复位、WDT 溢出或者 P0 口的下降沿而唤醒。当省电模式被唤醒时，时钟电路仍需要振荡稳定时间。当省电模式由复位或者 WDT 溢出被唤醒时，系统从 0000H 地址开始执行程序；当省电模式由 P0 口的下降沿唤醒时，PC 从 STOP 指令的下一个地址开始执行程序。

5.1.1 休眠模式应用举例

系统在进入 SLEEP 模式之前，若用户需要获得较小的休眠电流，请先确认所有 I/O 的状态，若用户方案中存在悬空的 I/O 口，把所有悬空口都设置为输出口，确保每一个输入口都有一个固定的状态，以避免 I/O 为输入状态时，口线电平处于不定态而增大休眠电流；关断比较器模块；根据实际方案的功能需求可禁止 WDT 功能来减小休眠电流。

例：进入 SLEEP 的处理程序

```
SLEEP_MODE：
        CLR     SYS_GEN         ;关闭中断；
        LDIA    B’10101010’
        LD      P0CL,A
        LD      P0CH,A          ;P0 口设置为输出口；
        LD      P1CL,A
        LDIA    B’10010010’
        LD      P1CH,A          ;P1 口设置为输出口；
        LDIA    B’00010010’
        LD      P2C,A           ;P2 口设置为输出口(若采用 INTRC 模式)；
        CLR     BUZCON          ;禁止 BUZ 功能；
        CLR     PWM8CON         ;禁止 8 位 PWM 功能；
        CLR     PWM10CON        ;禁止 10 位 PWM 功能；
        …                       ;各个输出口设置为不带负载状态；
        LDIA    0A5H
        LD      SP_FLAG,A       ;置休眠状态记忆寄存器(用户自定义)；
        CLRWDT                  ;清零 WDT；
        STOP                    ;执行 STOP 指令。
```

5.1.2 休眠模式的唤醒

当系统处于休眠状态时，有以下四种条件可以让 CPU 退出休眠状态：

- 看门狗溢出
- P0 口下降沿
- 复位口低电平(LVR 关闭)
- 系统掉电后，重新上电

处于休眠态的 MCU，发生 P0 口下降沿唤醒时，芯片从 STOP 指令的下一个地址开始运行程序；发生其它三种情况时，芯片 都会从复位地址(0000H)开始运行程序，用户可根据 FLAGS 的 TF 与 PF 标志位及 SP_FLAG（用户要自己定义），判断何种复位。

例：休眠唤醒用户处理程序

```
                ORG     0000H
                JP      START           ;转到复位处理程序
                ORG     0001H
                JP      INT_START       ;转到中断处理程序
                ORG     0002H
START:                                  ;复位处理程序
                SZB     FLAGS,PF
                JP      START_2         ;不是从休眠态复位的处理程序
                SZB     FLAGS,TF
                JP      START_3         ;非 WDT 唤醒 MCU 处理程序
                JP      START_4         ;WDT 唤醒 MCU
                …
                …
SLEEP_MODE:                             ;休眠子程序
                …                       ;设置休眠前的状态
                STOP                    ;芯片进入 SLEEP 态处理程序。
                NOP                     ;按键唤醒，加一个空指令等时钟稳定
                JP      XXXX            ;按键唤醒应该处理的程序
```

5.1.3 休眠模式唤醒时间

当 MCU 从休眠态被唤醒时，需要等待一个振荡稳定时间（OSC TIME）,这个时间可由内部烧写选项设置为 9ms、2ms、560μs、200us。详见 1.5 关于内部 CONFIG 烧写选项。

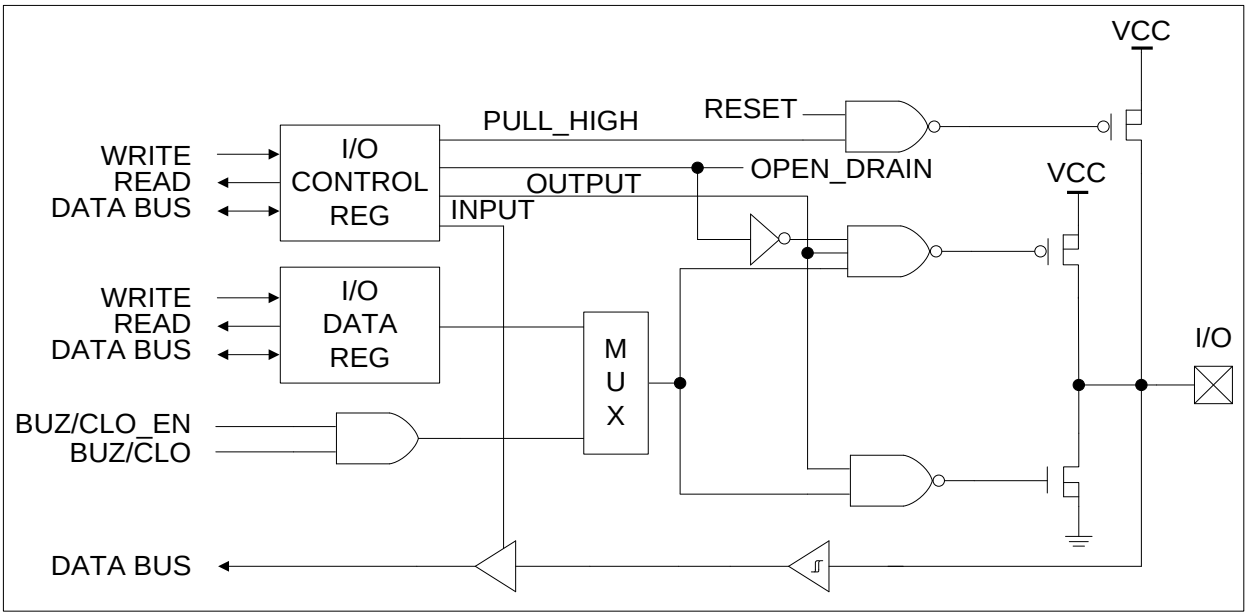
6. I/O 端口

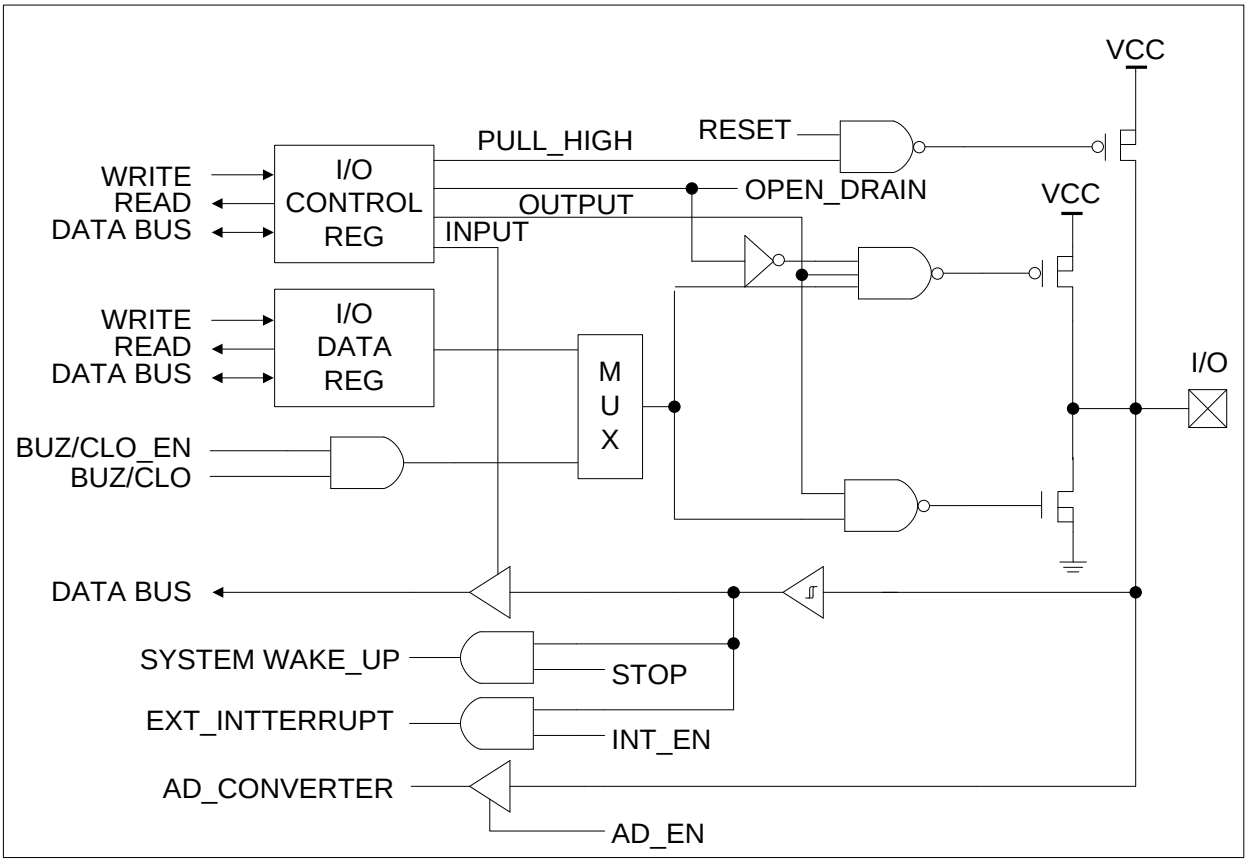
CMS69F012/013 有三个 I/O 端口：Port0,Port1 和 Port2（最多 14 个 I/O）。可读 写端口数据寄存器可直接存取这些端口。

端口	位	管脚描述	输入/输出
PORT 0	0	施密特触发输入，推挽式输出，外部中断 0，比较器 0 正端输入	I/O
	1	施密特触发输入，推挽式输出，外部中断 1，比较器 0 负端输入	I/O
	2	施密特触发输入，推挽式输出	I/O
	3	施密特触发输入，推挽式输出	I/O
	4	施密特触发输入，推挽式输出	I/O
	5	施密特触发输入，推挽式输出	I/O
	6	施密特触发输入，推挽式输出，PWM0	I/O
	7	施密特触发输入，推挽式输出，PWM1	I/O
PORT 2	0	施密特触发输入，推挽式输出，开漏式输出	I/O
	1	施密特触发输入，推挽式输出，开漏式输出	I/O
	2	施密特触发输入，开漏式输出	I/O
PORT 1	0	施密特触发输入，推挽式输出，开漏式输出，比较器 1 正端输入	I/O
	1	施密特触发输入，推挽式输出，开漏式输出，比较器 1 负端输入	I/O
	6	施密特触发输入，推挽式输出，开漏式输出，CLO	I/O

＜表 6-1　端口配置总概＞

◆ I/O 口结构图


VCC
PULL_HIGH
RESET
WRITE
READ
DATA BUS
I/O
CONTROL
REG
OPEN_DRAIN
OUTPUT
INPUT
VCC
WRITE
READ
DATA BUS
I/O
DATA
REG
M
U
X
I/O
BUZ/CLO_EN
BUZ/CLO
DATA BUS


VCC
PULL_HIGH
RESET
WRITE
READ
DATA BUS
I/O
CONTROL
REG
OPEN_DRAIN
OUTPUT
INPUT
VCC
WRITE
READ
DATA BUS
I/O
DATA
REG
M
U
X
I/O
BUZ/CLO_EN
BUZ/CLO
DATA BUS
SYSTEM WAKE_UP
STOP
EXT_INTTERRUPT
INT_EN
AD_CONVERTER
AD_EN

6.1 I/O 口模式及上、下拉电阻

寄存器 P0CL、P0CH、P1CL、P1CH、P2C 用于控制 I/O 口线的工作模式。

6.1.1 P0 口

CMS69F012/013 芯片的 P0 口是一个 8BIT 的 I/0 口，有三个寄存器与之相关。分别为 IO 口数据寄存器（P0）、IO 口功能控制寄存器（P0CL、P0CH）。

P0 口数据寄存器 P0(05H)

05H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X
定义	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O
	PWM10	PWM8				RTCC	EXTINT1	EXTINT0

P0 口功能寄存器 P0CL(09H)

09H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P0CL	P0.3 功能配置		P0.2 功能配置		P0.1 功能配置		P0.0 功能配置	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	0	1	0	1	0	1

bit 7-bit 6 **P0.3 功能配置**

00：上拉输入；

01：输入；

10：推挽输出；

11：未用；

bit 5-bit 4 **P0.2 功能配置**

00：上拉输入；

01：输入；

10：推挽输出；

11：未用；

bit 3-bit 2 **P0.1 功能配置**

00：上拉输入、中断；

01：输入、中断、比较器 0 的“+”端；

10：推挽输出；

11：未用；

bit 1-bit 0 **P0.0 功能配置**

00：上拉输入、中断；

01：输入、中断、比较器 0 的“—”端；

10：推挽输出；

11：未用；

P0 口功能寄存器 P0CH(0AH)

0AH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P0CH	P0.7 功能配置		P0.6 功能配置		P0.5 功能配置		P0.4 功能配置	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	1	1	1	1	0	1	0	1

bit 7-bit 6 **P0.7 功能配置**

00：上拉输入；

01：PWM10 输出；

10：推挽输出；

11：输入；

bit 5-bit 4 **P0.6 功能配置**

00：上拉输入；

01：PWM8 输出；

10：推挽输出；

11：输入；

bit 3-bit 2 **P0.5 功能配置**

00：上拉输入；

01：输入；

10：推挽输出；

11：未用；

bit 1-bit 0 **P0.4 功能配置**

00：上拉输入；

01：输入；

10：推挽输出；

11：未用；

例：P0 口处理程序

```
    LDIA    B'00010101'     ;P0.7 为上拉输入，P0.6 为 PWM 输出
    LD      P0CH,A          ;P0.4-P0.5 为输入
    LDIA    B'10101010'
    LD      P0CL,A          ;P0.0-P0.3 为推挽输出
    LDIA    03H             ;P0.0-P0.1 输出高，P0.2-P0.3 输出低
    LD      P0,A            ;由于 P0.4,P0.5,P0.7 为输入口，P0.6 为 PWM 口，所以
                            ;赋 0 或 1 都没影响
```

6.1.2 P1 口

P1 口有 3-BIT 的输入输出管脚。它可做正常输入输出端口（施密特触发输入，推挽式输入，开漏式输入）或者是一些选择性功能用（比较器输入，CLO 输出）。有三个寄存器与之相关。P1 口数据寄存器 P1、P1 口低位控制寄存器 P1CL(0BH)、 P1 口高位控制寄存器 P1CH(0CH)。

P1 口数据寄存器 P1(06H)

06H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P1	---	P1.6	---	---	---	---	P1.1	P1.0
R/W	---	R/W	---	---	---	---	R/W	R/W
复位值	X	X	X	X	X	X	X	X
定义		I/O	I/O				I/O	I/O
		CLO					COMP1-	COMP1+
							BUZ	T2OUT

P1 口功能寄存器 P1CL(0BH)

0BH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P1CL	----	----	----	----	P1.1 功能配置		P1.0 功能配置	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	0	1	0	0	0	1

bit 7-bit 4 **未用**

bit 3-bit 2 **P1.1 功能配置**

00：上拉输入，比较器 1 负端输入(比较器输入时上拉电阻自动关闭)；

01：蜂鸣器输出；

10：推挽输出；

11：开漏输出，（只能输出“0”,和截止态）。

bit 1-bit 0 **P1.0 功能配置**

00：上拉输入；

01：输入，比较器 1 正端输入；

10：推挽输出；

11：T2OUT 输出.

P1 口功能寄存器 P1CH(0CH)

0CH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P1CH	P1.6 功能配置			----			----	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	1	0	0	1	0	1

bit 7-bit 5 **P1.6 功能配置**

000：上拉输入；

001：输入；

01X：未用

100：推挽输出；

101：开漏输出（带上拉）；

110：开漏输出；

111：CLO 输出；

bit 4-bit 0 **未用**

注：P1 口的使用方法同 P0 口

6.1.3 P2 口

P2 口有 3-BIT 的输入输出管脚。P2.1、P2.0 可做时钟输入或正常的输入输出。如果芯片选择外部复位(LVR DISABLE)，则 P2.2 作为复位口，低电平复位；如果芯片选择内部复位(LVR ENABLE),则 P2.2 由 CONFIG 的控制位“RES_OUT”控制，当 RES_OUT 为 DISABLE 的时候，P2.2 只能作为普通输入口；当 RES_OUT 为 ENABLE 的时候，P2.2 可通过 P2C 设置成开漏输出。有两个寄存器与之相关。P2 口数据寄存器 P2(07H)、P2 口控制寄存器 P2C(0DH)。

P2 口数据寄存器 P2(07H)

07H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P2						P2.2	P2.1	P2.0
R/W						R	R/W	R/W
复位值	X	X	X	X	X	X	X	X
定义	未用					I/O REST	I/O	I/O

P2 口功能寄存器 P2C(0DH)

0DH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
P2C	---	P2.2 功能配置	P2.1 功能配置			P2.0 功能配置		
R/W	---	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	0	0	0	1	0	0	1

Bit 6 **P2.2 功能配置**

0：输入；

1：开漏输出（必须为内部复位，且 RES_OUT 设置为 ENABLE）

bit 5-bit 3 **P2.1 功能配置**

000：上拉输入；

001：输入；

010：推挽输出；

011：下拉输入；

100：开漏输出；

其它：未用；

bit 2-bit 0 **P2.0 功能配置**

000：上拉输入；

001：输入；

010：推挽输出；

011：下拉输入；

100：开漏输出；

其它：未用；

6.1.4 写 I/O 口

CMS69F012/013 系列芯片的 I/O 口寄存器，和一般通用寄存器一样，可以通过数据传输指令，位操作指令等进行写操作。

例：写 I/O 口程序

```
        LD      P0,A        ;ACC 值赋给 P0 口
        CLRB    P1,2        ;P1.2 口置零
        CLR     P2          ;P2 口清零
        SET     P1          ;P1 所有输出口置 1
        SETB    P1.2        ;P1.2 口置 1
```

6.1.5 读 I/O 口

例：读 I/O 口程序

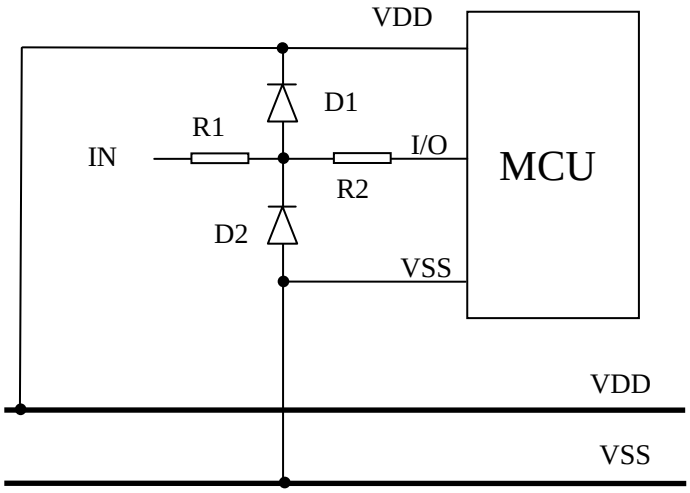
```
        LD      A,P0        ;P0 的值赋给 ACC
        SNZB    P0,1        ;判断 P0,1 口是否为 1，为 1 跳过下一条语句
        SZB     P0,1        ;判断 P0,1 口是否为 0，为 0 跳过下一条语句
```

注：当用户读一个 I/O 口状态时，若此 I/O 口为输入口，则用户读回的数据将是此口线外部电平的状态，若此 I/O 口为输出口那么读出的值将会是此口线内部输出寄存器的数据。

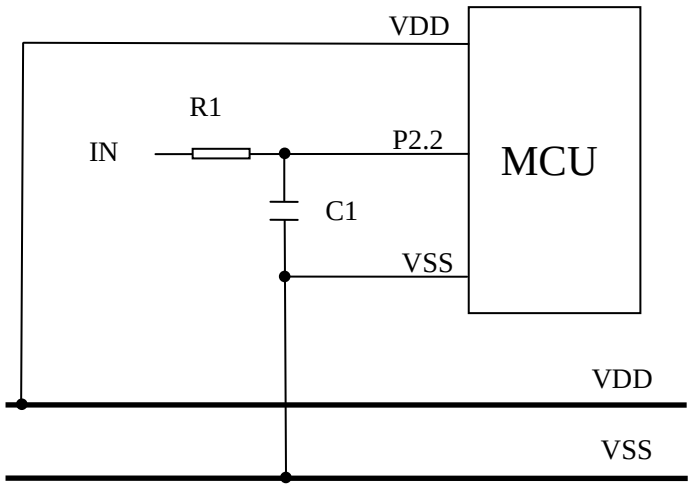
6.2 I/O 口使用注意事项

在操作 I/O 口时，应注意以下几个方面：

1. 当 I/O 从输出转换为输入时，要等待几个指令周期的时间，以便 I/O 口状态稳定。
2. 若使用内部上拉电阻，那么当 I/O 从输出转换为输入时，内部电平的稳定时间，与接在 I/O 口上的电容有关，用户应根据实际情况，设置等待时间，以防止 I/O 口误扫描电平。
3. 当 I/O 口为输入口时，其输入电平应在“VDD+0.7V”与”VSS-0.7V”之间。若输入口电压不在此范围内可采用如下图所示方法。



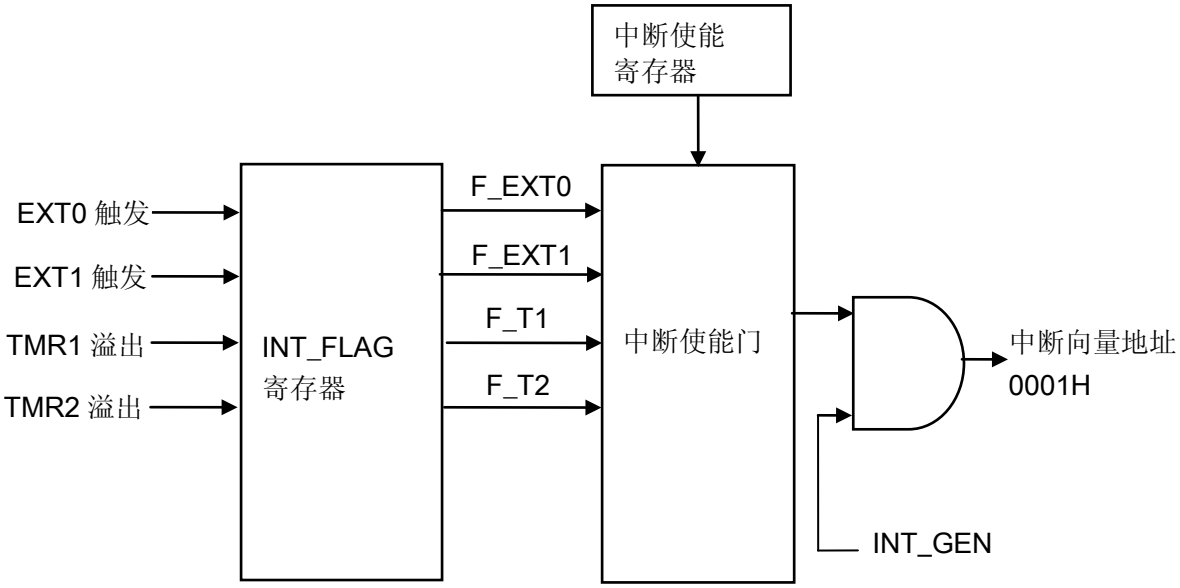
4. 若在 I/O 口在线串入较长的连接线，请在靠近芯片 I/O 的地方加上限流电阻以增强 MCU 抗 EMC 能力
5. 若使用到 P2.2 口作为信号输入口，建议采用如下图做法，以增强 MCU 抗 EMC 及 ESD 能力。



7. 中断

7.1 中断概述

CMS69F012/013共有 4个中断源：2个内部中断(TMR1、TMR2)和 2个外部中断（EXT0、EXT1）。一旦程序进入中断，寄存器 SYS_GEN 的位 INT_GEN位将被硬件自动清零以避免再次响应其它中断。系统退出中断，即执行完 RETI 指令后，硬件自动将 INT_GEN 置“1”，以响应下一个 中断。中断请求存放在寄存器INT_FLAG寄存器中。



7.2 中断控制寄存器

中断请求控制寄存器 INT_EN 包括所有中断的使能控制位。INT_EN 的有效位被置为“1”，则系统进入该中断服务程序，程序计数器入栈，程序转至 0001H 即中断程序。程序运行到指令 RETI 时，中断结束，系统退出中断服务。

11H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
INT_EN	---	---	---	---	EN_T1	EN_T2	EN_EXT1	EN_EXT0
R/W	---	---	---	---	R/W	R/W	R/W	R/W
复位值	X	X	0	0	0	0	0	0

bit 7-bit4 **未用，使用时清零**

bit 3 **EN_T1：TMR1中断使能位**

0：禁止TMR1中断

1：使能TMR1中断

bit 2 **EN_T2：TMR2中断使能位**

0：禁止TMR2中断

1：使能TMR2中断

bit 1 **EN_EXT1：外部中断1中断使能位**

0：禁止EXT1中断

1：使能EXT1中断

bit 0 **EN_EXT0：外部中断0中断使能位**

0：禁止EXT0中断

1：使能EXT0中断

7.3 中断请求寄存器

中断请求寄存器 INT_FLAG 中存放各中断请求标志。一旦有中断请求发生，INT_FLAG 中的相应位将被置“1”，该请求被响应后，程序应将该标志位清零，MCU 不会自动清零该中断请求标志位。根据 INTR_FLAG 的状态，程序判断是否有中断发生，并执行相应的中断服务。

12H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
INT_FLAG	---	---	---	---	F_T1	F_T2	F_EXT1	F_EXT0
R/W	---	---	---	---	R/W	R/W	R/W	R/W
复位值	X	X	0	0	0	0	0	0

bit 7-bit4 **未用，使用时清零**

bit 3 **F_T1：TMR1中断请求标志位**

0：无TMR1中断请求

1：有TMR1中断请求

bit 2 **F_T2：TMR2中断请求标志位**

0：无TMR2中断请求

1：有TMR2中断请求

bit 1 **F_EXT1：外部中断1中断请求标志位**

0：无EXT1中断请求

1：有EXT1中断请求

bit 0 **F_EXT0：外部中断0中断请求标志位**

0：无EXT0中断请求

1：有EXT0中断请求

7.4 总中断使能控制寄存器

只有当全局中断控制位 INT_GEN置“1”的时候程序才能响应中断请求。 一旦有中断发生，程序计数器（PC）指向中断 向量地址（0001H），堆栈层数加 1。

10H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
SYS_GEN	---	---	---	---	---	---	---	INT_GEN
R/W	---	---	---	---	---	---	---	R/W
复位值	X	X	X	X	X	X	0	0

bit 7-1 **未用，使用时清零**

bit 0 **INT_GEN：中断总使能位**

0：禁止所有中断

1：使能中断功能

7.5 中断现场的保护方法

有中断请求发生并被响应后，程序转至 0001H 执行中断子程序。响应中断之前，必须保存 ACC、FLAGS 的内容。 芯片没有提供专用的入栈保存和出栈恢复指令，用户需自己保护ACC和FLAGS的内容，以避免中断结束后可能的程序运行错误。

例：对 ACC 与 FLAGS 进行入栈保护

```
                ORG     0000H
                JP      START           ;用户程序起始地址
                ORG     0001H
                JP      INT_SERVICE     ;中断服务程序
                ORG     0002H
START：
                ...
                ...
INT_SERVICE：
PUSH：                                  ;中断服务程序入口，保存 ACC 及 FLAGS
                LD      ACC_BAK,A       ;保存 ACC 的值，(ACC_BAK 需自定义)
                SWAPA   FLAGS
                LD      FLAGS_BAK,A     ;保存 FLAGS 的值，（FLAGS_BAK 需自定义）
                ...
                ...
POP：                                   ;中断服务程序出口，还原 ACC 及 FLAGS
                SWAPA   FLAGS_BAK
                LD      FLAGS,A         ;还原 FLAGS 的值
                SWAPR   ACC_BAK         ;还原 ACC 的值
                SWAPA   ACC_BAK
                RETI
```

7.6 外部中断

CMS69F012/013 系列芯片有两个外部中断源（EXTINT0、EXTINT1），只有当外部中断被触发，且 EN_EXT0、EN_EXT1 使能时，F_EXT0、F_EXT1 才会被置 1。当任何一个中断使能位与中断请求标志位同时为“1”，且总中断使能位为使能状态，系统就会响应中断。

7.6.1 外部中断控制寄存器

CMS69F012/013 系列芯片的两个外部中断源（EXT0、EXT1）的触发方式可由 INT_EXT 寄存器控制，用户可通过写 INT_EXT 控制外部中断源的触发方式。

13H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
INT_EXT	---	---	---	---	EXT1		EXT0	
R/W	---	---	---	---	R/W	R/W	R/W	R/W
复位值	X	X	X	X	0	0	0	0

bit 3-bit 2 **EXT1：EXT1信号边缘选择**
00： 下降沿
01： 上升沿
1X： 两种边沿

bit 1-bit 0 **EXT0：EXT0信号边缘选择**
00： 下降沿
01： 上升沿
1X： 两种边沿

7.6.2 外部中断 0

外部中断 EXT0 与 P0.0 口共用一个 I/O 口，若要启用外部中断功能，要将 P0.0 口设置为中断输入模式，当 EN_EXT0=1 且 F_EXT0=1 时，系统会响应外部中断 0，当 EN_EXT0=0 时无论 F_EXT0 为任何状态，都不会响应中断。

例：外部中断 0 应用程序

```
                ORG         0000H
                JP          START               ;用户程序起始地址
                ORG         0001H
                JP          INT_SERVICE         ;中断服务程序
                ORG         0002H
START：
                …
                LDIA        B’10101001’
                LD          P0CL,A              ;P0,0 口设置为中断输入口
                CLR         INT_EXT             ;EXT0 为下降沿触发
                CLRB        INT_FLAG,F_EXT0     ;清零 EXT0 中断请求标志位
                SETB        INT_EN,EN_EXT0      ;使能 EXT0 中断
                SETB        SYS_GEN,INT_GEN     ;使能 INT_GEN
                …
                JP          START
INT_SERVICE：
PUSH：                                          ;中断服务程序入口，保存 ACC 及 FLAGS
                …
                SNZB        INT_EN,EN_EXT0      ;判断是否使能 EXT0 中断
                JP          INT_EXIT
                SNZB        INT_FLAG,F_EXT0     ;检查有无 EXT0 中断请求标志位
                JP          INT_EXIT
                CLRB        INT_FLAG,F_EXT0     ;清零 EXT0 中断请求标志位
                …                               ;EXT0 中断处理程序
INT_EXIT：
POP：                                           ;中断服务程序出口，还原 ACC 及 FLAGS
                …
                RETI
```

7.6.3 外部中断 1

外部中断 EXT1 与 P0.1 口共用一个 I/O 口，若要启用外部中断功能，要将 P0.1 口设置为中断输入模式，当 EN_EXT1=1 且 F_EXT1=1 时，系统会响应外部中断 1，当 EN_EXT1=0 时无论 F_EXT0 为任何状态，都不会响应中断。

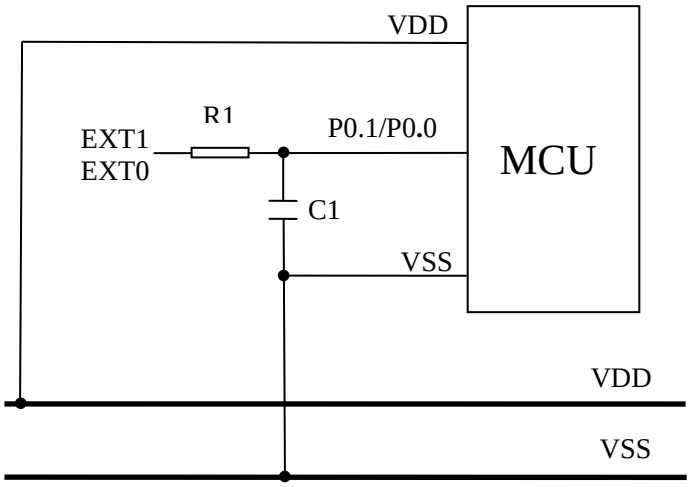
例：外部中断 1 应用程序

```
                ORG     0000H
                JP      START               ;用户程序起始地址
                ORG     0001H
                JP      INT_SERVICE         ;中断服务程序
                ORG     0002H
START：
                …
                LDIA    B’10100110’
                LD      P0CL,A              ;P0.1 口设置为中断输入口
                LDIA    B’00001000
                LD      INT_EXT,A           ;EXT1 为电平触发
                CLRB    INT_FLAG,F_EXT1     ;清零 EXT1 中断请求标志位
                SETB    INT_EN,EN_EXT1      ;使能 EXT1 中断
                SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
                …
                JP      START
INT_SERVICE：
PUSH：                                      ;中断服务程序入口，保存 ACC 及 FLAGS
                …
                …
                SNZB    INT_EN,EN_EXT1      ;判断是否使能 EXT1 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_EXT1     ;检查有无 EXT1 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_EXT1     ;清零 EXT1 中断请求标志位
                …
                …                           ;EXT1 中断处理程序
INT_EXIT：
POP：                                       ;中断服务程序出口，还原 ACC 及 FLAGS
                …
                RETI
```

7.6.4 外部中断的响应时间

外部中断的响应时间为 2 个指令周期。

7.6.5 外部中断的应用注意事项



由于外部中断的反应时间很快，当系统外围电压波动时，或系统受到 EMC 干扰时，MCU 可能误进中断，所以需要加上 RC 滤波电路,如上图所示。用户可根据外部中断所采样的信号频率选择不同的 R1 和 C1，以提高系统抗 EMC 能力。

7.7 内部定时中断

CMS69F012/013 系列芯片内部有三个定时器，TMR0、TMR1、TMR2，只有 TMR1 跟 TMR2 可能产生中断。

7.7.1 TMR1 中断

TMR1 溢出时，如果 EN_T1 置“1”，则F_T1 会被置“1”，系统就会响应 TMR1的中断；若 EN_T1= 0，则 F_T1 不会被置“1”，系统不会响应 TMR1 中断。尤其需要注意多种中断下的情形。

例：TMR1 中断应用程序

```
                ORG     0000H
                JP      START               ;用户程序起始地址
                ORG     0001H
                JP      INT_SERVICE         ;中断服务程序
                ORG     0002H
        START：
                …
                LDIA    06H
                LD      TMR1,A              ;初始化 TMR1
                LDIA    80H
                LD      TMR1C,A             ;设置 TMR1 时钟=Fcpu、TMR1 定时器模式
                CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
                SETB    INT_EN,EN_T1        ;使能 TMR1 中断
                SETB    TMR1C,TON           ;TMR1 计时器开始工作
                SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
                …
         MAIN：
                …
                JP      MAIN
INT_SERVICE：
         PUSH：                             ;中断服务程序入口，保存 ACC 及 FLAGS
                …
                SNZB    INT_EN,EN_T1        ;判断是否使能 TMR1 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_T1       ;检查有无 TMR1 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
                …                           ;TMR1 中断处理程序
    INT_EXIT：
          POP：                             ;中断服务程序出口，还原 ACC 及 FLAGS
                …
                RETI
```

7.7.2 TMR2 中断

TMR2 溢出时，如果 EN_T2 置“1”，则F_T2 会被置“1”，系统就会响应 TMR2的中断；若 EN_T2= 0，则 F_T2 不会被置“1”，系统不会响应 TMR2 中断。尤其需要注意多种中断下的情形。

例：TMR2 中断应用程序

```
                ORG     0000H
                JP      START           ;用户程序起始地址
                ORG     0001H
                JP      INT_SERVICE     ;中断服务程序
                ORG     0002H
START：
                ...
                LDIA    032H
                LD      T2DATA,A        ;设置 T2 溢出时间
                LDIA    B'00110000'
                LD      T2CON,A         ;设置 TMR2 时钟
                CLRB    INT_FLAG,F_T2   ;清零 TMR2 中断请求标志位
                SETB    INT_EN,EN_T2    ;使能 TMR2 中断
                SETB    T2CON,0         ;TMR2 计时器开始工作
                SETB    SYS_GEN,INT_GEN ;使能 INT_GEN
                ...
MAIN：
                ...
                ...
                JP      MAIN
INT_SERVICE：
PUSH：                                  ;中断服务程序入口，保存 ACC 及 FLAGS
                ...
                ...
                SNZB    INT_EN,EN_T2    ;判断是否使能 TMR2 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_T2   ;检查有无 TMR2 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_T2   ;清零 TMR2 中断请求标志位
                ...
                ...                     ;TMR2 中断处理程序
INT_EXIT：
POP：                                   ;中断服务程序出口，还原 ACC 及 FLAGS
                ...
                RETI
```

7.9 中断的优先级，及多中断嵌套

在同一时刻，系统中可能出现多个中断请求。此时，用户必须根据系统的要求对各中断进行优先权的设置。中断请求标志 INT_FLAG由中断事件触发，当 F_XX 处于有效值“1”时， 系统并不一定会响应该中断。各中断触发事件如下表所示：

中断	有效触发
F_EXT0	由INT_EXT决定
F_EXT1	由INT_EXT决定
F_T2	TMR2溢出
F_T1	TMR1溢出

注：**多个中断同时发生时**，MCU 没有预置的中断优先级。**首先，必须预先设定好各中断的优先权**；**其次，利用**中断使能位**和**中断控制位，**控制系统是否响应该中断。在程序中，必须对中断控制位和中断请求标志进行检测。**

例：多个中断处理程序

```
                ORG     0000H
                JP      START           ;用户程序起始地址
                ORG     0001H
                JP      INT_SERVICE     ;中断服务程序
                ORG     0002H
START:                                  ;用户程序起始地址
                ...
MAIN:
                ...                     ;用户主程序
                JP      MAIN
INT_SERVICE:
PUSH:                                   ;中断服务程序入口，保存 ACC 及 FLAGS
                ...
EXT0CH:
                SNZB    INT_EN,EN_EXT0  ;判断是否使能 EXT0 中断
                JP      EXT1CH
                SZB     INT_FLAG,F_EXT0 ;检查有无 EXT0 中断请求标志位
                JP      INT_EXT0        ;EXT0 中断处理程序
EXT1CH:
                SNZB    INT_EN,EN_EXT1  ;判断是否使能 EXT1 中断
                JP      TMR1CH
                SZB     INT_FLAG,F_EXT1 ;检查有无 EXT1 中断请求标志位
                JP      INT_EXT1        ;EXT1 中断处理程序
```

例：多个中断处理程序(续)

```
TMR1CH：
        SNZB    INT_EN,EN_T1        ;判断是否使能 TMR1 中断
        JP      TMR2CH
        SZB     INT_FLAG,F_T1       ;检查有无 TMR1 中断请求标志位
        JP      INT_TMR1            ;TMR1 中断处理程序
TMR2CH：
        SNZB    INT_EN,EN_T2        ;判断是否使能 TMR2 中断
        JP      INT_EXIT
        SZB     INT_FLAG,F_T2       ;检查有无 TMR2 中断请求标志位
        JP      INT_TMR2            ;TMR2 中断处理程序
        JP      INT_EXIT
INT_TMR2：
        CLRB    INT_FLAG,F_TMR2
        …                           ;TMR2 中断处理程序
        JP      INT_EXIT
INT_TMR1：
        CLRB    INT_FLAG,F_TMR1
        …                           ;TMR1 中断处理程序
        JP      INT_EXIT
INT_EXT1：
        CLRB    INT_FLAG,F_EXT1
        …                           ;EXT1 中断处理程序
        JP      INT_EXIT
INT_EXT0：
        CLRB    INT_FLAG,F_EXT0
        …                           ;EXT0 中断处理程序
        JP      INT_EXIT
INT_EXIT：
    POP：                           ;中断服务程序出口，还原 ACC 及 FLAGS
        …
        RETI
```

8. 定时计数器 TMR0

8.1 定时计数器 TMR0 概述

TMR0 由如下功能组成：

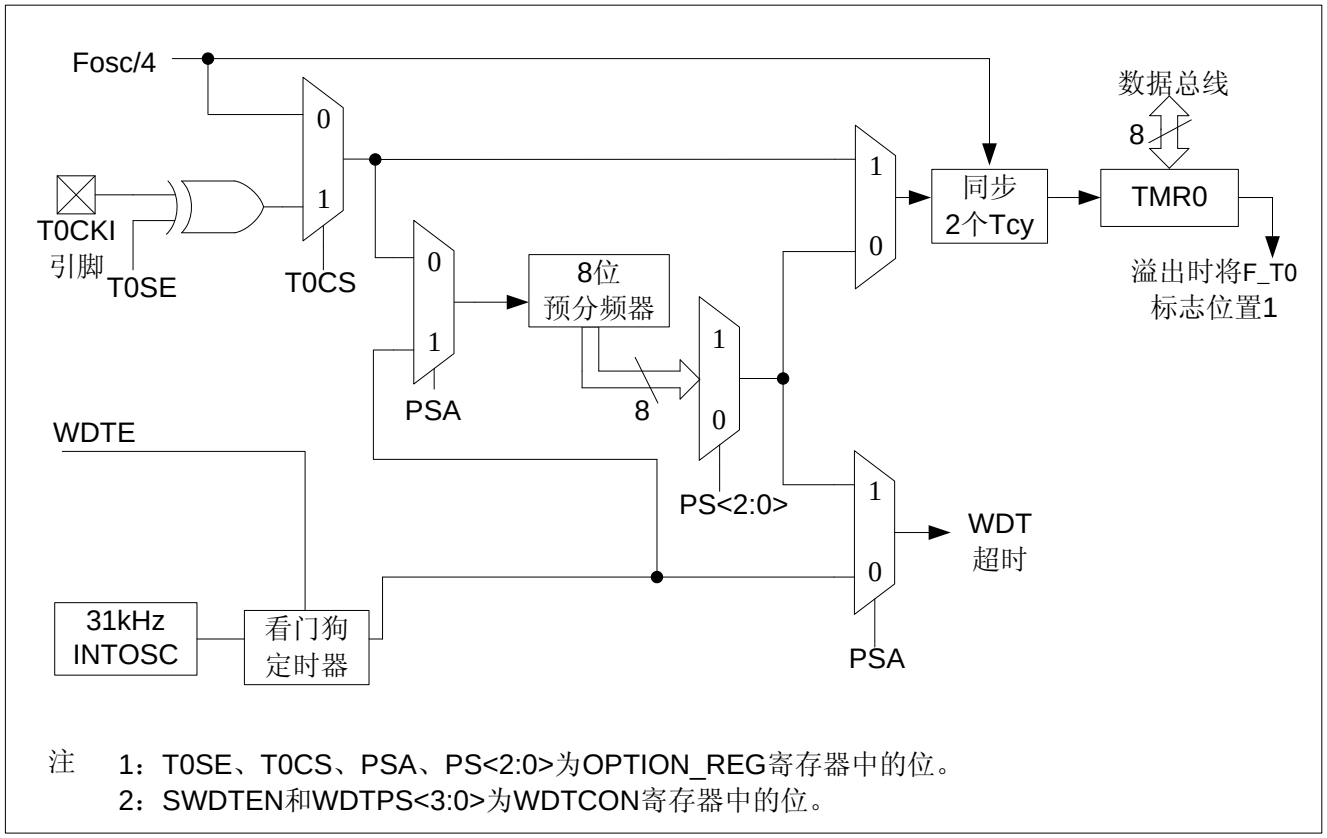
- 8 位定时器 / 计数器
- 可用程序进行读写操作
- 可选择定时器或计数器工作方式
- 8 位可编程控制寄存器（OPTION）
- 外部时钟边沿可选择

TMR0的工作模式由TMR0控制寄存器（OPTION）的T0CS选择：

当T0CS=0时以定时器方式工作，在不用预分频器情况下每个指令周期加1，若对TMR0进行写操作那么增量操作便禁止两个周期，TMR0没有中断。

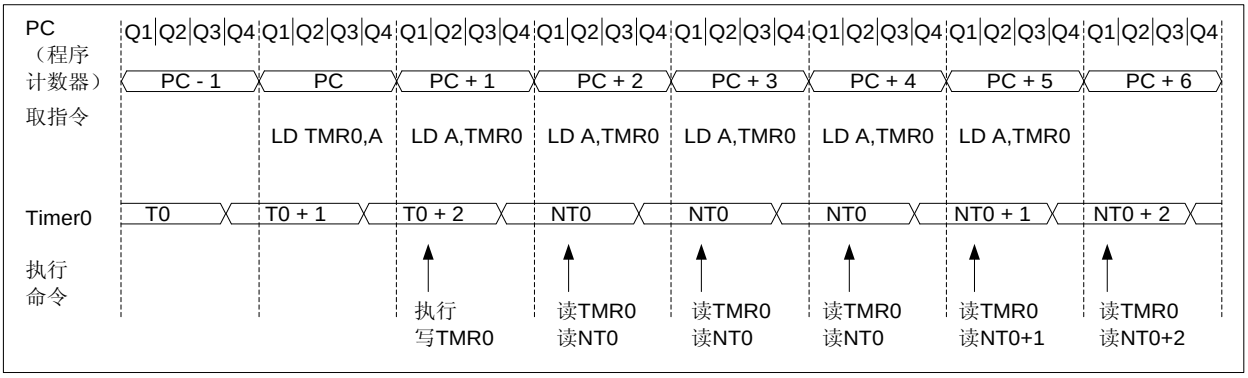
当T0CS=1时以计数器方式工作，TIMER0模块的计数器将对加到RTCC口的脉冲进行计数。是上升沿还是下降沿有效则由位T0SE选择，T0SE=0时选择上升沿有效，T0SE=1时选择下降沿有效。

- TMR0/WDT结构图

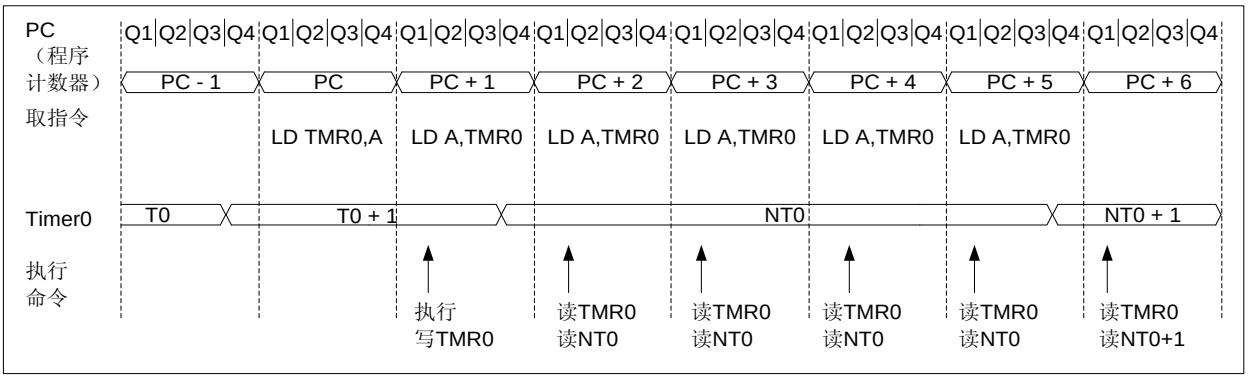


注 1：T0SE、T0CS、PSA、PS<2:0>为OPTION_REG寄存器中的位。
2：SWDTEN和WDTPS<3:0>为WDTCON寄存器中的位。

◆ TMR0 时序图，内部时钟/无预分频器



◆ TMR0 时序图，内部时钟/预分频器 1：2



8.2 与 TMR0 相关寄存器

有两个寄存器与 TMR0 相关，8 位定时器 / 计数器（TMR0），8 位可编程控制寄存器（OPTION）。TMR0 为一个 8 位可读写的定时/计数器，OPTION 为一个 8 位只写寄存器，用户可改变 OPTION 的值，来改变 TMR0 的工作模式等。请参看 2.6 关于预分频寄存器（OPTION）的应用。

01H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
TMR0								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
OPTION	---	---	T0CS	T0SE	PSA	PS2	PS1	PS0
读写			W	W	W	W	W	W
复位值	X	X	1	1	1	1	1	1

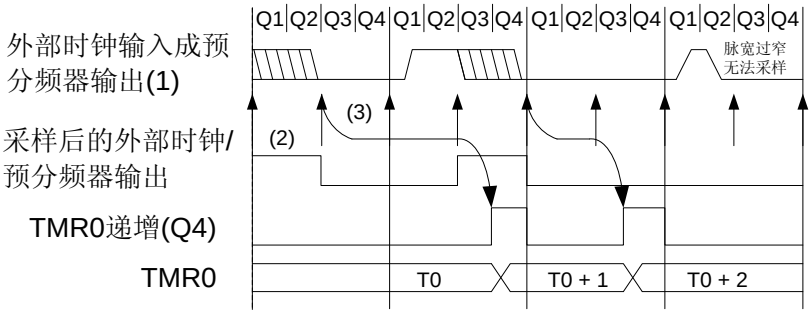
8.3 使用外部时钟作为 TMR0 的时钟源

TMR0 用于外部时钟计数时，外部时钟输入必须满足特定条件。要求外部时钟与内部相位时钟（Tosc）同步，在同步后要经过一定延时，TMR0 才会递增。

如果不使用预分频器，那么外部时钟就是 TMR0 的输入，在内部时钟的 Q2 和 Q4 周期对预分频器输出进行采样可实现 RTCC 与内部相位时钟同步。因此要求 RTCC 引脚信号的高电平时间至少为 2 个 Tosc（加上一小段的 RC 延时），并且低电平时间至少为 2 个 Tosc（加上一小段的 RC 延时 ）。

若使用了预分频器，外部时钟输入要先经过异步脉动计数型预分频器的分频，从而使预分频器的输出对称。为了使外部时钟满足采样要求，必须考虑纹波计数器的影响。因此 RTCC 的时钟周期至少为 4 个 Tosc（加上一小段的 RC 延时）除以预分频值。RTCC 引脚上的高低电平持续时间只须满足 10ns 的最低脉宽要求即可。

◆ TMR0 与外部时钟时序



注： 1、不选择预分频器时为外部时钟；否则为预分频器的输出。
2、箭头所指为采样时刻。
3、时钟输入发生变化到TMR0递增会有3个TOSC到7个TOSC(持续时间Q=TOSC)的延时。
因此测量TMR0输入的相邻脉冲间隔时，其最大误差为±4TOSC。

8.4 TMR0 做定时器的应用

8.4.1 TMR0 的基本时间常数

当 OPTION 的第 3 位被置 1 时，预分频器作为 WDT 计时的分频，此时 TMR0 的输入时钟为系统时钟的 1 分频。当 OPTION 的第 3 位被置 0 时，预分频器作为 TMR0 计数器的分频。其基本时间常数如下表：

OPTION PS3～PS0	TMR0 的输入时钟 T0CLK	Fcpu=4MHz÷4	
		最大溢出间隔时间	TMR0 递增时间
1xxx	Fcpu/1	256 μs	1 μs
0000	Fcpu/2	512 μs	2 μs
0001	Fcpu/4	1024 μs	4 μs
0010	Fcpu/8	2048 μs	8 μs
0011	Fcpu/16	4096 μs	16 μs
0100	Fcpu/32	8192 μs	32 μs
0101	Fcpu/64	16384 μs	64 μs
0110	Fcpu/128	32768 μs	128 μs
0111	Fcpu/256	65536 μs	256 μs

8.4.2 TMR0 操作流程

TMR0 的操作流程为：

- 设置 TMR0 工作模式，及分频比；
- 设置 TMR0 初值

例：TMR0 定时设置程序

```
        LDIA    00H
        OPTION                  ;设置 TMR0 时钟=Fcpu/2
        CLR     TMR0            ;初始化 TMR0
```

注：每次 TMR0 溢出时 TMR0 的初值并不会被自动加载，故用户需在每次 TMR0 溢出时重新加载 TMR0 初值；由于对 TMR0 进行写操作，TMR0 将会有一个 T0CLKS 时钟不递增，用户要自己输入校正值来避开这个问题。

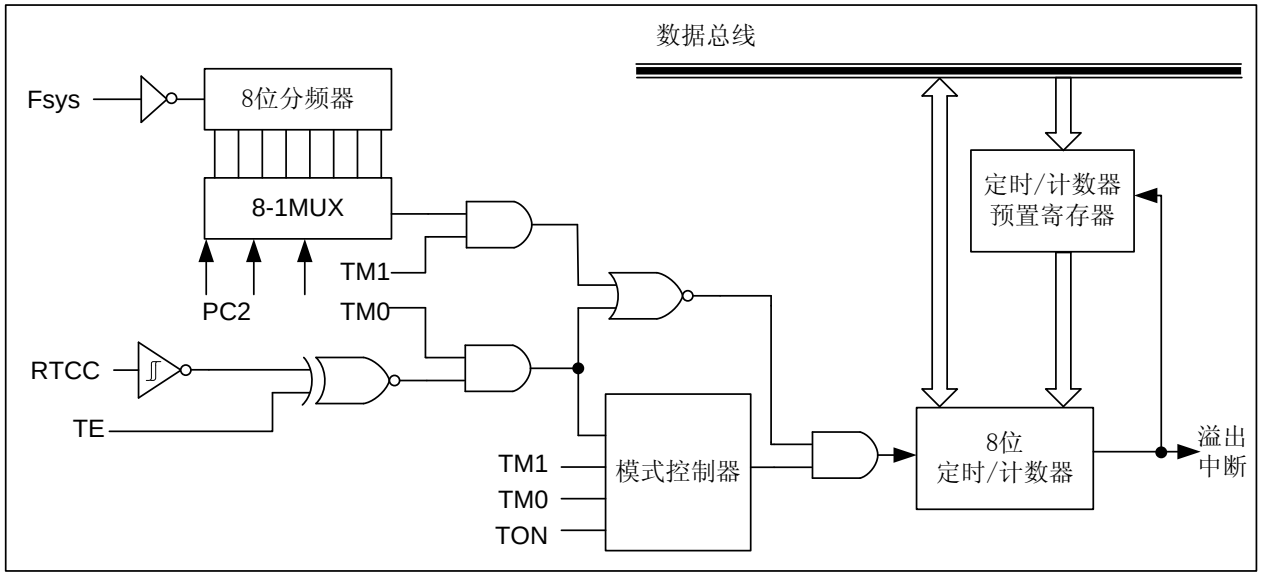
9. 定时计数器 TMR1

9.1 TMR1 概述

TMR1 由如下功能组成：

- 可选择时钟频率
- 8 位定时器 / 计数器
- TMR1 控制寄存器（TMR1C）
- 中断在 FF 到 00 时溢出
- 外部时钟为边沿可选择
- 二种不同工作模式

TMR1结构图：



TMR1有两个与定时/计数器有关的寄存器，TMR1和TMR1C。TMR1 寄存器有两个物理空间；写入TMR1会将初始值装入到定时/计数器的预置寄存器中，而读TMR1则会取得定时/计数器的内容。TMR1C 是定时/计数器控制寄存器，它可以定义定时/计数器的工作模式。

TM0、TM1 用来定义定时/计数器的工作模式。外部事件计数模式是用来记录外部事件的，其时钟来源为外部RTCC引脚输入。

定时器模式是一个常用模式，其时钟来源为内部时钟。无论是定时模式还是外部事件计数模式，一旦开始计数，定时/计数器会从寄存器当前值向上计到0FFH。一旦发生溢出，定时/计数器会从预置寄存器中重新加载初值，并开始计数；同时置位中断请求标志(F_T1；INT_FLAG 的第3位)，位F_T1须用软件清零。

当计数器溢出时，会从定时/计数器的预置寄存器中重新加载初值，而中断的处理方式与其它两种模式一样。要启动计数器，只要置位TON(TMR1C 的第4 位)，TON 只能由指令来清除。定时/计数器溢出可以做为唤醒信号。不管是什么模式，只要写0 到EN_T1 位即可禁止定时/计数器中断服务。

在定时/计数器停止计数时，写资料到定时/计数器的预置寄存器中，同时会将该数据写入到定时/计数器。

但如果在定时/计数器工作时这么做，数据只能写入到预置寄存器中，直到发生溢出时才会将数据从预置寄存器加载到定时/计数器寄存器。读取定时/计数器时，计数会被停止，以避免发生错误；计数停止会导致计数错误，程序员必须注意到这一点。

9.2 TMR1 相关寄存器

TMR1数据寄存器TMR1(16H)

16H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
TMR1								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X

TMR1 控制寄存器 TMR1C(17H)

17H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
TMR1C	TM1	TM0	未用	TON	TE	PSC2	PSC1	PSC0
读写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	1	0	0	0

bit 7-bit 6 **TM1-TM0：TMR1 工作模式选择位**

00：不工作

01：外部事件计数器

10：内部定时器

11：不工作

bit 4 **TON：工作使能位**

0：禁止

1：使能

bit 3 **TE：计数模式边缘选择**

0：上升沿计数

1：下降沿计数

bit 1-bit 0 **PSC2～PSC0：分频比选择**

000： 1：1

001： 1：2

010： 1：4

011： 1：8

100： 1：16

101： 1：32

110： 1：64

111： 1：128

9.3 TMR1 的时间常数

9.3.1 TMR1 基本时间参数

TMR1 在设置不同分频时候的基本事件参数如下表：

TMR1C PSC2～PSC0	TMR1 的输入时钟 T1CLK	Fcpu=4MHz÷4	
		最大溢出间隔时间	TMR1 递增时间
000	Fcpu	256μs	1μs
001	Fcpu/2	512μs	2μs
010	Fcpu/4	1024μs	4μs
011	Fcpu/8	2048μs	8μs
100	Fcpu/16	4096μs	16μs
101	Fcpu/32	8192μs	32μs
110	Fcpu/64	16384μs	64μs
111	Fcpu/128	32768μs	128μs

9.4 TMR1 的应用

9.4.1 TMR1 作定时器使用

TMR1 作内部定时器时，可以产生一个定时中断，设置 T1 定时器的操作流程如下：

- 禁止 TMR1 定时器
- 禁止 TMR1 中断并清除 TMR1 中断标志位；
- 设置 TMR1 为定时器模式，及分频比；
- 设置 TMR1 初值
- 开启 TMR1 中断

例：TMR1 定时设置程序

```
CLRB    TMR1C,TON           ;禁止 TMR1 定时器工作
CLRB    INT_EN,EN_T1        ;禁止 TMR1 中断
CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
LDIA    B'10000000'
LD      TMR1C,A             ;设置 TMR1 为定时器模式，分频比 1：1
LDIA    06H
LD      TMR1,A              ;设置 TMR1 初值
CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
SETB    INT_EN,EN_T1        ;使能 TMR1 中断
SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
SETB    TMR1C,TON           ;使能 TMR1 定时器
```

9.4.2 TMR1 作计数器使用

通过设置 TMR1C 的 TM1、TM0 位可以使 TMR1 工作在外部事件计数模式，当 TMR1 被设置为外部事件计数模式时，在外部计数器模式下 TMR1 会根据 RTCC 口的上升沿或则下降沿递增，在此种模式下分频比选项是不起作用的。

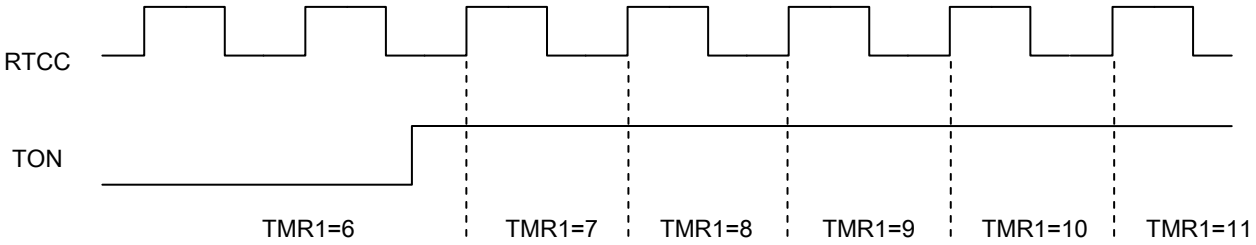
设置 T1 计数器的操作流程如下：

- 禁止 TMR1 计数器
- 禁止 TMR1 中断并清除 TMR1 中断标志位；
- 设置 TMR1 为计数器模式；
- 设置 TMR1 初值
- 开启 TMR1 中断

例：TMR1 计数器设置程序（上升沿递增模式）

```
        CLRB    TMR1C,TON           ;禁止 TMR1 工作；
        CLRB    INT_EN,EN_T1        ;禁止 TMR1 中断；
        CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位；
        LDIA    B'01000000'
        LD      TMR1C,A             ;设置 TMR1 为计数器模式；
        LDIA    06H
        LD      TMR1,A              ;设置 TMR1 初值；
        CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位；
        SETB    INT_EN,EN_T1        ;使能 TMR1 中断；
        SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN；
        SETB    TMR1C,TON           ;使能 TMR1 定时器。
```

TMR1 作外部计数器时工作时序如下：



TMR1 在作外部计数器时工作流程如下：

- RTCC 口有方波信号输入
- TMR1 在方波信号的上升沿或者下降沿递增
- 当 TMR1 从 FF 加到 00 时产生中断请求信号 F_T1
- 若中断使能，则回应 TMR1 中断

关于 TMR1 作计数器的下降沿递增模式，与上升沿一样，只是一个在下降沿递增、一个在上升沿递增。这里不再赘述。

10. 定时计数器 TMR2

10.1 TMR2 概述

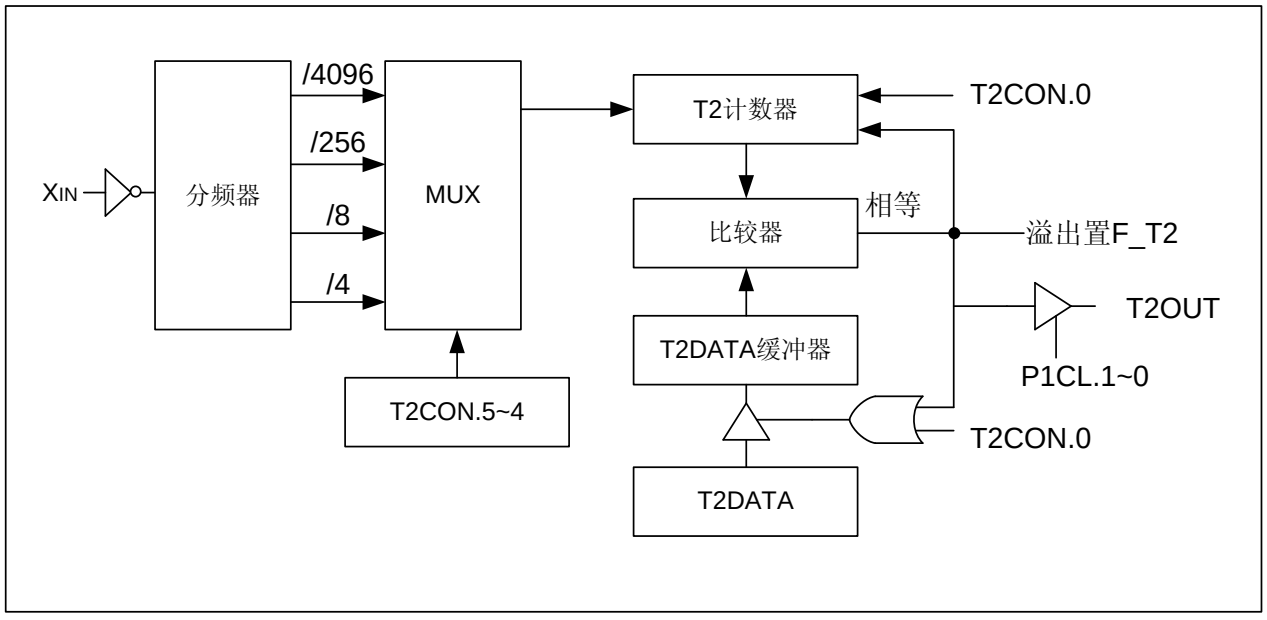
TMR2 由如下功能组成：

- 选择时钟频率
- 8 Bit 计数器（T2CNT）,8Bit 比较器，8Bit 数据寄存器（T2DATA）和 T2DATA 缓冲器
- TMR2 控制寄存器（T2CON）

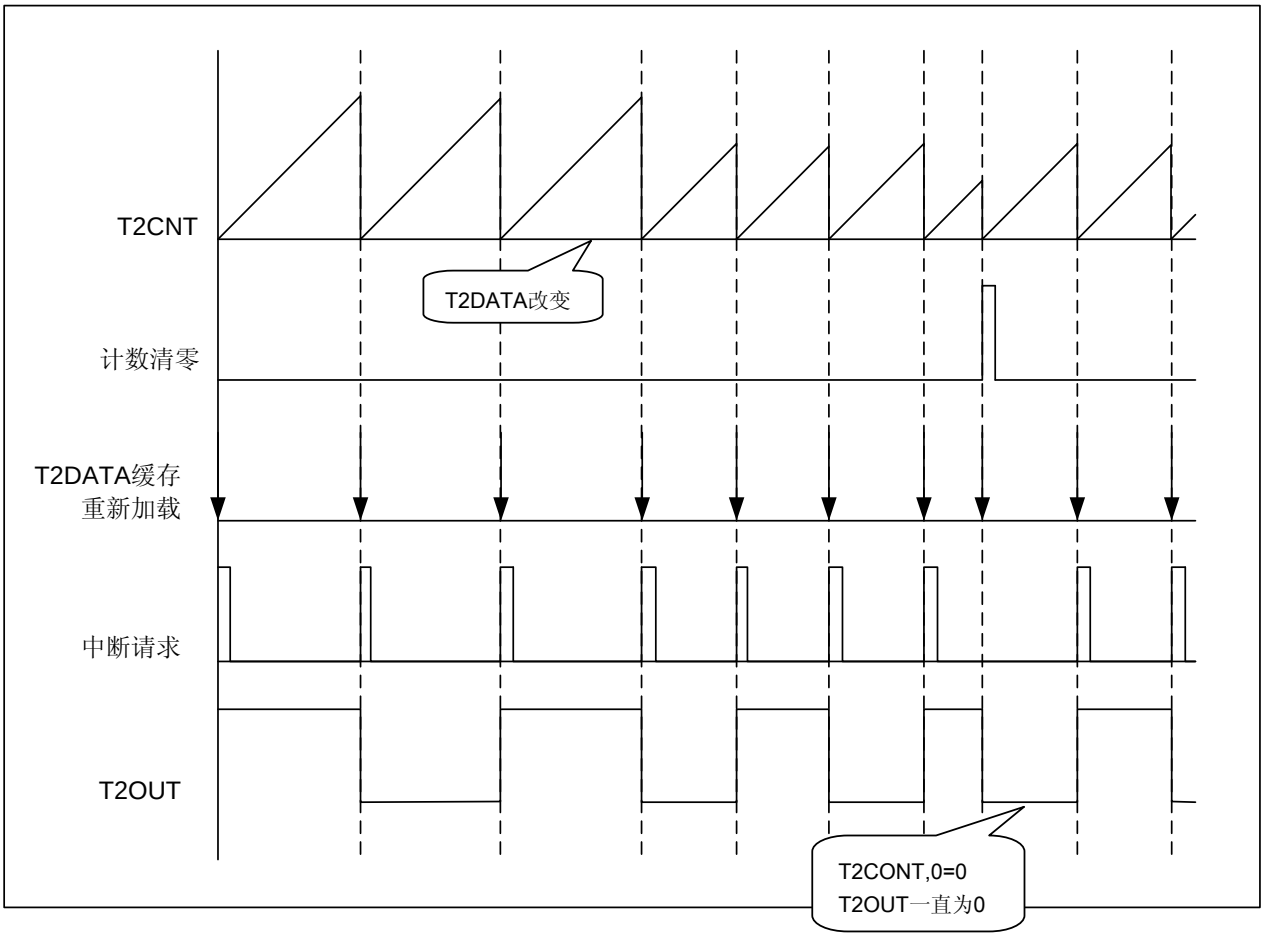
T2CON（bit4 和 bit5）用来选择 TMR2 的输入时钟频率。定时器 2 中断的使能位和标志位由 INT_EN.2 和 INT_FLAG.2 控制。在定时模式下，当 TMR2 计数器的值和 T2DATA 值相等时，将产生一个 TMR2 的匹配信号来清除 T2 计数器的值和重载 T2DATA 的比较值,假如 T2 中断是使能的，那么也会同时产生中断请求信号。如果 TMR2 中断禁止(INT_EN.2=0),匹配的信号不产生匹配的中断请求。时钟分配器不是 TMR2 的组成部分，且分配的时钟和定时器中断使能信号是同步的。所以，在第一个匹配时间间隔里是矛盾的。

注：中断请求标志位 F_T2 必须由软件清除。

TMR2 结构框图如下：



TMR2 时序图


T2CNT
T2DATA改变
计数清零
T2DATA缓存
重新加载
中断请求
T2OUT
T2CONT,0=0
T2OUT一直为0

10.2 TMR2 相关的寄存器

有三个寄存器与 TMR2 相关，分别是数据存储器 T2DATA，计数器 T2CNT，控制寄存器 T2CON

TMR2 计数器 T2CNT(18H)

18H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
T2CNT								
R/W	R	R	R	R	R	R	R	R
复位值	0	0	0	0	0	0	0	0

TMR2 控制寄存器 T2CON(19H)

19H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
T2CON	未用	未用	T2C1	T2C0	未用	未用	未用	T2_CLR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

bit 7-bit 6 未用

bit 5-bit 4 **T2C1、T2C0：时钟选择位**

00：Fosc/4096

01：Fosc/256

10：Fosc/8

11：Fosc/4

bit 3-bit 1 未用

bit 0 **T2_CLR：工作使能位**

0：禁止 TMR2 时钟，T2CNT 清零

1：使能 TMR2 时钟，T2CNT 从“0”开始计数

TMR2 数据存储器 T2DATA(1AH)

1AH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
T2DATA								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	1	1	1	1	1	1	1	1

10.3 TMR2 的时间常数

10.3.1 TMR2 基本时间参数

在 8M 的时钟下，TMR2 基本事件参数如下表：

T2CON T2C1、T2C0	T2CNT 计数时钟	Fosc=8MHz	
		最大溢出间隔时间	最小溢出间隔时间
00	Fosc /4096	131072μs	512μs
01	Fosc /256	8192μs	32μs
10	Fosc /8	256μs	1μs
11	Fosc /4	128μs	0.5μs

10.3.2 T2DATA 初值计算方法：

T2DATA 初值 =T2 溢出时间×时钟频率÷分频比－1

例：Fosc=8MHz、分频比 1：4、T2 溢出时间 100μs 时 T2DATA 的值

T2DATA 初值 =T2 溢出时间×时钟频率÷分频比－1

=100μs×8 MHz÷4－1

=199

10.4 TMR2 应用

TMR2 作计数器时，可以产生一个定时中断，设置 T2 计数器的操作流程如下：

- 禁止 TMR2 定时器
- 禁止 TMR2 中断并清除 TMR2 中断标志位；
- 设置 TMR2 分频比；
- 开启 TMR2 中断
- 开始 TMR2 计数

例：TMR2 定时设置程序

```
CLRB    T2CON,T2_CLR        ;清零 T2CNT
CLRB    INT_EN,EN_T2        ;禁止 TMR2 中断
CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
LDIA    063H
LD      T2DATA,A            ;设置 TMR2 目标值
LDIA    B'00110000'
LD      T2CON,A             ;设置 TMR2 分频比 1：4
CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
SETB    INT_EN,EN_T2        ;使能 TMR2 中断
SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
SETB    T2CON,T2_CLR        ;T2CNT 开始计数
```

10.5 T2OUT 输出

当 T2 溢出时，I/O 口（P1,0）可以与其匹配输出，由 I/O 口控制寄存器 P1CL 控制，当 P1,0 设置为 T2OUT 口时，无论此时往 P1,0 I/O 寄存器写“1”，还是“0”；P1,0 都会匹配 T2 溢出输出。

10.5.1 T2OUT 的周期

T2OUT 的输出周期为 T2 溢出中断的两倍。计算公式如下：

T2OUT 周期 =T2 溢出时间×2

10.5.2 T2OUT 基本时间参数

在 8M 时钟下，T2OUT 基本时间参数如下表：

T2CON T2C1、T2C0	T2CNT 计数时钟	Fosc=8MHz	
		T2OUT 最大输出周期	T2OUT 最小输出周期
00	Fosc /4096	262144μs	1024μs
01	Fosc /256	16384μs	64μs
10	Fosc /8	512μs	2μs
11	Fosc /4	256μs	1μs

10.5.3 T2OUT 应用

在 P1,0 口输出 T2 溢出匹配信号的操作流程如下：

- 禁止 TMR2 定时器
- 禁止 TMR2 中断并清除 TMR2 中断标志位；
- 设置 TMR2 分频比；
- 设置 P1,0 口为 T2OUT 口
- 开始 TMR2 计数

例：T2OUT 设置程序

```
        CLRB    T2CON,T2_CLR        ;清零 T2CNT
        CLRB    INT_EN,EN_T2        ;禁止 TMR2 中断
        CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
        LDIA    063H
        LD      T2DATA,A            ;设置 TMR2 目标值
        LDIA    B'00110000'
        LD      T2CON,A             ;设置 TMR2 分频比 1：4
        CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
        LDIA    B'00000011'
        LD      P1CL,A              ;P1,0 设置为 T2OUT 口
        SETB    T2CON,T2_CLR        ;T2CNT 开始计数
```

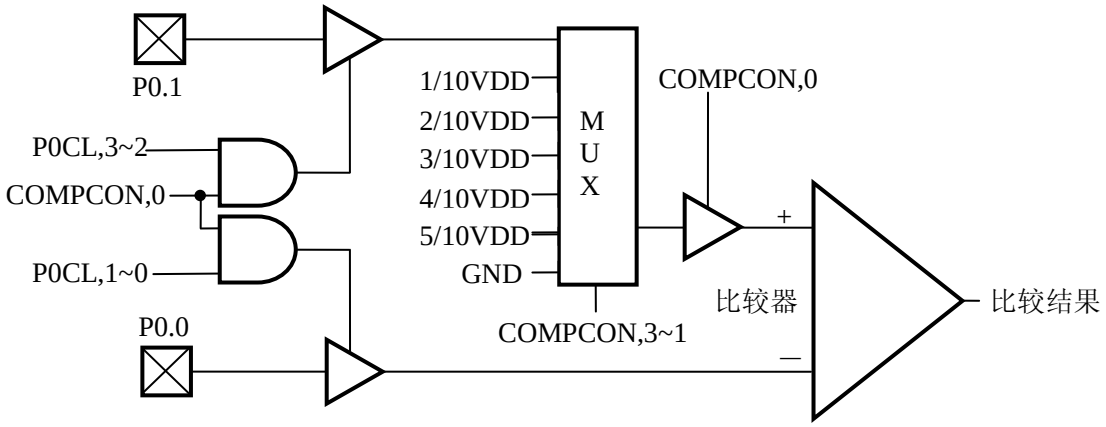
11. 内置比较器

11.1 内置比较器概述

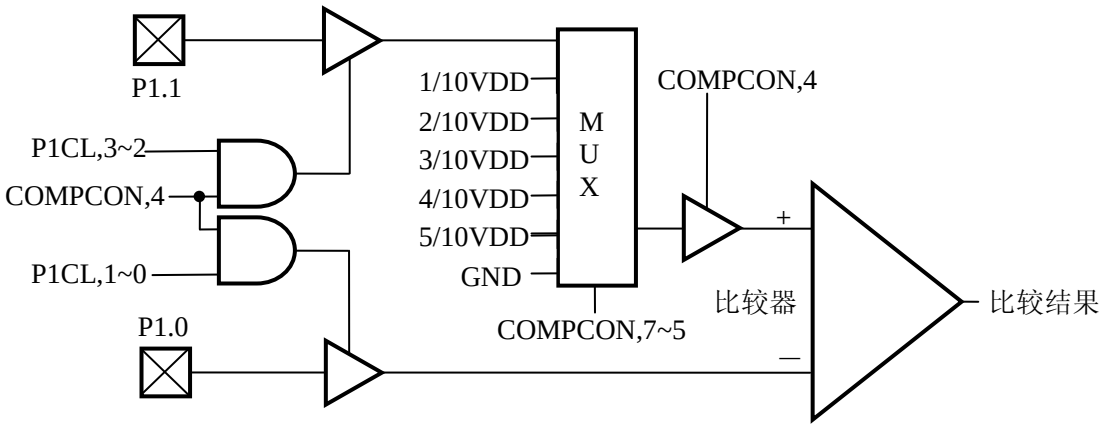
CMS69F012/013系列芯片内置了两路高精度比较器:COMP0及COMP1(69F012只有1路)，芯片比较器的正端输入可以设置为接外部输入，也可设置为GND,1/10 VDD，2/10 VDD， 3/10 VDD， 4/10 VDD， 5/10 VDD等参考电压。用户可通过设置COMPCON寄存器来开启、关断比较器功能及设置比较器正端标准电压。

比较器工作原理：当“＋”＞“－”时输出高，当“＋”＜“－”时输出低。用户可以读取P0.0及P1.0的状态可以确定比较器的输出状态，此时程序读取的不是P0.0或者P1.0的外部电平状态，而是读取其内部寄存器的值。

◆ COMP0结构框图



◆ COMP1结构框图



11.2 与比较器相关的寄存器

比较器控制寄存器COMPCON(20H)

20H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
COMPCON	BPLUS2	BPLUS1	BPLUS0	COMP1_EN	APLUS2	APLUS1	APLUS0	COMP0_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

bit 7- bit 5 **BPLUS2-BPLUS0**：比较器1正端电压选择

000：比较器1正端电压为GND、P1.1为普通I/O口；

001：比较器1正端电压为1/10VDD、P1.1为普通I/O口；

010：比较器1正端电压为2/10VDD、P1.1为普通I/O口；

011：比较器1正端电压为3/10VDD、P1.1为普通I/O口；

100：比较器1正端电压为4/10VDD、P1.1为普通I/O口；

101：比较器1正端电压为5/10VDD、P1.1为普通I/O口；

110：比较器1正端接P1.1口；

111：比较器1正端接P1.1口。

bit 4 **COMP1_EN**：比较器1使能控制

0：比较器1关闭，P1,0为普通I/O口；

1：比较器1开启，P1,0为比较器1”－”端；

bit 3-bit 1 **APLUS2-APLUS0**：比较器0正端电压选择

000：比较器0正端电压为GND、P0,1为普通I/O口；

001：比较器0正端电压为1/10VDD、P0.1为普通I/O口；

010：比较器0正端电压为2/10VDD、P0.1为普通I/O口；

011：比较器0正端电压为3/10VDD、P0.1为普通I/O口；

100：比较器0正端电压为4/10VDD、P0.1为普通I/O口；

101：比较器0正端电压为5/10VDD、P0.1为普通I/O口；

110：比较器0正端接P0.1口；

111：比较器0正端接P0.1口。

bit 0 **COMP0_EN**：比较器0使能控制

0：比较器0关闭，P0.0为普通I/O口；

1：比较器0开启，P0.0为比较器0”－”端；

11.3 比较器 0 应用

输入输出端口说明：

“－”端 ----- 对应于IO口P0.0;

“＋”端 ----- 对应于IO口P0.1或内部选择比较电压;

“输出”端 -- 对应于[05H].0 即 P0.0。

芯片由COMPCON控制打开或关闭比较器0，COMPCON.0=1打开比较器，COMPCON.0=0关闭比较器。当打开比较器时，只要COMPCON.0=1，P0.0自动设为比较器输入口“-”端，否则为普通IO口；当COMPCON.[3:2:1]=110或111，P0.1自动设为比较器“＋”端输入口，COMPCON.[3:2:1]为其余数，比较器“＋”端连接到内部选择比较电压。需要判断比较器输出时，直接读取P0.0的状态即为输出结果。

比较器使用程序流程：

- 设置P0.0、P0.1口状态
- 设置COMPCON.0=1,(如果需要正端输入则设置COMPCON,[3：2：1]=110或111)
- 等待比较稳定
- 读取比较结果P0.0
- 设置COMPCON,0=0 ， 关闭比较器0。

例：COMP0的应用程序

```
            LDIA    B “XXXX0101”
            LD      P0CL,A          ;设置P0,0、P0,1为比较器输入
            LDIA    B “XXXX1111”
            LD      COMPCON,A       ;设置并开启比较器
            CALL    DELY_TIME       ;延时等待比较结果稳定
            SZB     P0,0            ;判断比较结果
            JP      P_LT_N
N_LT_P:
            …
            JP      EXIT
P_LT_N:
            …
            JP      EXIT
  EXIT:
            CLRB    COMPCON,0       ;关闭比较器
```

11.4 比较器 1 应用

输入输出端口说明：

“－”端 ----- 对应于IO口P1,0，

“＋”端 ----- 对应于IO口P1,1或内部选择比较电压

“输出”端 -- 对应于[06H],0 即 P1,0。

芯片由COMPCON控制打开或关闭比较器1，COMPCON.4=1打开比较器，COMPCON.4=0关闭比较器。当打开比较器时，只要COMPCON.4=1，P1,0自动设为比较器输入口“-”端，否则为普通IO口；当COMPCON.[7:6:5]=110或111，P1,1自动设为比较器“＋”端输入口，COMPCON.[7:6:5]为其余数，比较器“＋”端连接到内部选择比较电压。需要判断比较器输出时，直接读取P1,0的状态即为输出结果。

比较器1程序流程：

- 设置COMPCON.4=1,(如果需要正端输入则设置COMPCON,[7：6：5]=110或111)
- 等待比较器稳定
- 读取比较器结果P1,0
- 设置COMPCON,4=0关闭比较器1

例：COMP1的应用程序

```
            LDIA    B “1111XXXX”
            LD      COMPCON,A           ;设置并开启比较器
            CALL    DELY_TIME           ;延时等待比较结果稳定
            SZB     P0,0                ;判断比较结果
            JP      P_LT_N
N_LT_P:
            …
            JP      EXIT
P_LT_N:
            …
            JP      EXIT
  EXIT:
            CLRB    COMPCON,4           ;关闭比较器
```

注：当用户使用CMS69F012/013系列芯片的内部比较器时需要注意以下几点：

1. P0.0、P0.1、P1.0、P1.1口电压不能超过(VDD－1)V；若有可能请把电压限制在1/2VDD以下， 以提高比较器精度。

2. 当比较器从开启到输出稳定的比较结果需要一定的等待时间，具体时间与外围元件有关，用户可根据不同的应用环境设置不同的等待时间。

12. 8 位 PWM(PWM0)

12.1 8 位 PWM 概述

PWM8 由如下功能组成：

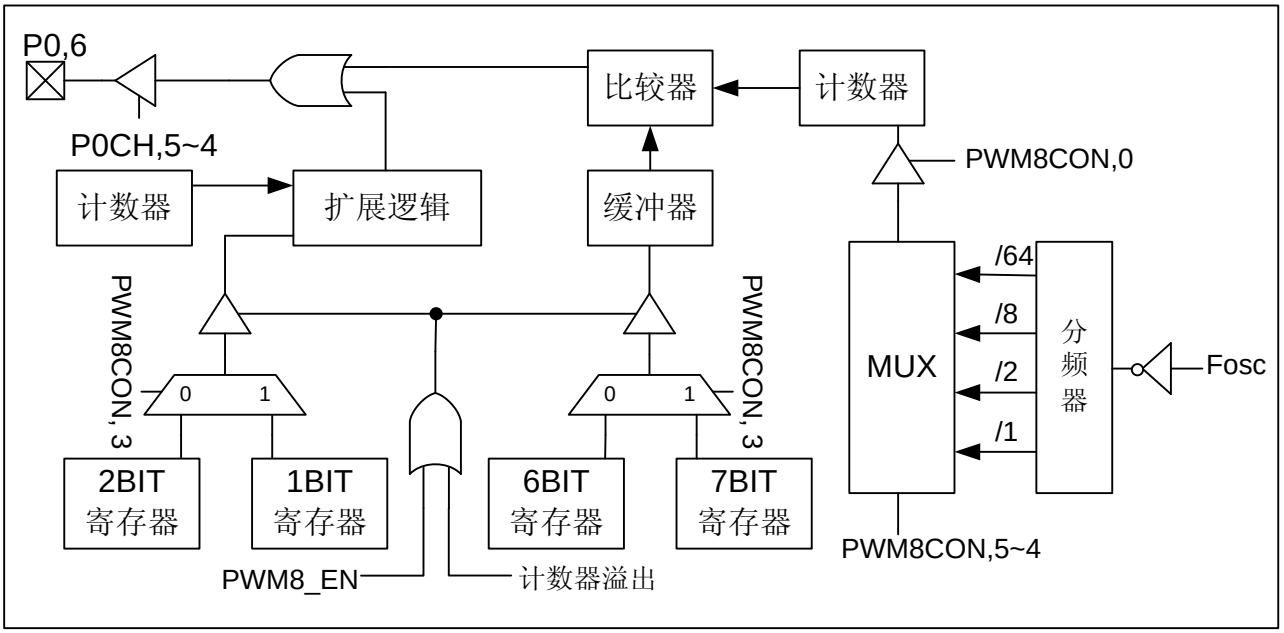
- 选择时钟频率
- 8-bit 计数器（PWM8CON），6-bit 比较器，6-bit 数据存储器（PWM6DATA）,和 6-bit 数据缓冲器。
- 2-bit 扩展逻辑，2-bit 扩展寄存器和数据缓冲器。
- 两种模式选择(6 + 2)/(7 + 1)

CMS69F012/013 的 8 位脉冲宽度调制器有两种工作模式，由 PWM8CON.3 位控制，PWM8CON.3=1 选择“7+1”模式，PWM8CON.3=0 选择“6+2”模式。PWM8CON.2=1 为 6 位溢出时加载，即改变 PWM8 的数据存储器后会在下一个波形输出时改变占空比，PWM8CON.2=0 时为 8 位溢出时加载，即改变 PWM8 的数据存储器后会在下一个周期时改变占空比（也就是说当选择“6+2”模式时，PWM8 将为 4 个波形一个周期，此时不论你在哪一个波形输出时改变 PWM8 的数据存储器都将在下一个周期才生效）。

所谓“6+2”模式就是指 PWM8DATA 的高 6 位（PWMDATA,7～2）用于控制 PWM8 的调制周期及在调制周期内占空比，低 2 位（PWMDATA,1～0）用于控制扩展周期。

所谓“7+1”模式就是指 PWM8DATA 的高 7 位（PWMDATA,7～1）用于控制 PWM8 的调制周期及在调制周期内占空比，低 1（PWMDATA,0）位用于控制扩展周期。

- 8 位 PWM 框图



12.2 与 8 位 PWM 相关寄存器

有两个寄存器与PWM8有关，PWM8DATA(数据存储器)、PWM8CON(控制寄存器)。

1CH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
PWM8DATA								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

1DH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
PWM8CON	CLO占空比选择		PWM8时钟选择		模式选择	加载选择	-	PWM8_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

bit 7-bit 6 **CLO 占空比选择**

00： CLO 占空比 50%

01： CLO 占空比 25%

10： CLO 占空比 50%

11： CLO 占空比 75%

bit 5-bit 4 **PWM8 时钟选择**

00： PWM8 时钟为 Fosc/64

01： PWM8 时钟为 Fosc/8

10： PWM8 时钟为 Fosc/2

11： PWM8 时钟为 Fosc/1

bit 3 **模式选择位**

0：“6+2”模式

1：“7+1”模式

bit 2 **加载选择位**

0：数据缓冲在 8 位溢出时加载

1：数据缓冲在 6 位溢出时加载

bit 0 **PWM8_EN：PWM8 使能控制位**

0：PWM8 停止工作

1：PWM8 允许工作

12.3 8 位 PWM 的周期

12.3.1 8 位 PWM 调制周期

8 位 PWM 调制周期由系统主频（Fosc）、PWM8 分频比、PWM8 模式决定，计算公式如下：

$$\text{PWM8 调制周期} = 2^N \times \text{PWM8 分频比} \div F_{OSC}$$

注：N=6 或者 7 由 PWM8 模式决定

例：Fosc=8MHz、分频比 1：2、“6+2”模式，时 PWM 调制周期

$$\text{PWM8 调制周期} = 2^N \times \text{PWM8 分频比} \div F_{OSC}$$
$$= 2^6 \times 2 \div (8 \times 10^6)\ s$$
$$= 16 \times 10^{-6}\ s$$
$$= 16\ \mu s$$

Fosc=8MHz 时 PWM8 的调制周期表

PWM8CON BIT5、BIT4	PWM8 时钟	Fosc=8MHz	
		“6+2”模式	“7+1”模式
00	Fosc /64	512μs	1024 μs
01	Fosc /8	64 μs	64 μs
10	Fosc /2	16 μs	32 μs
11	Fosc /1	8 μs	16 μs

12.3.2 8 位 PWM 输出周期

8 位 PWM 输出周期由 PWM8 模式确定，当选择“6+2”模式时，4 个调制周期为一个输出周期，当选择“7+1”模式时 2 个调制周期为一个输出周期。

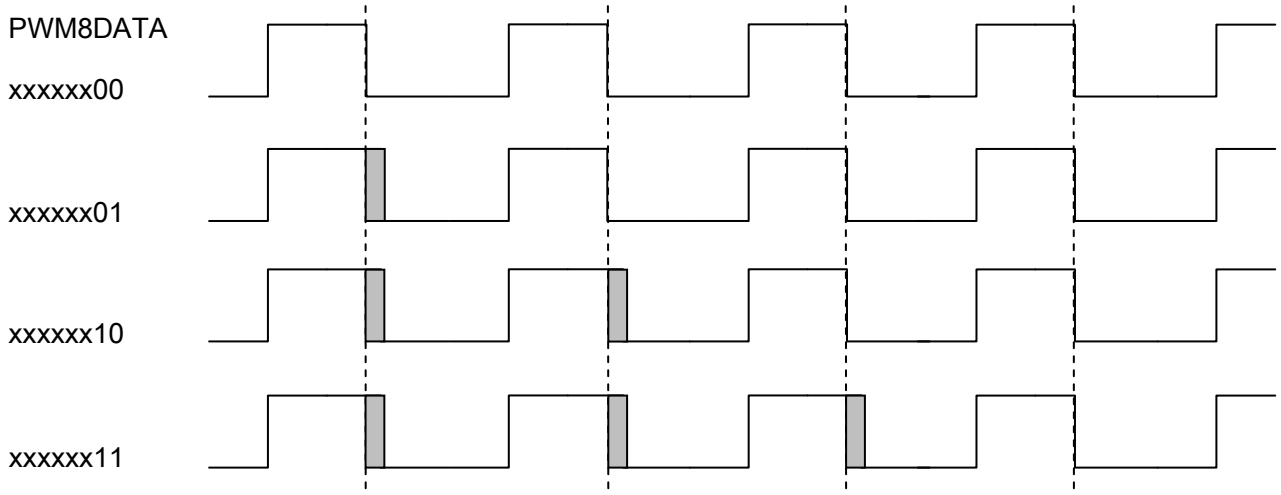
12.4 8 位 PWM 占空比算法

8 位 PWM 输出的占空比与 PWM8DATA 的数值相关，从整体上来说，其占空比近似等于 PWMDATA÷256。不同的模式 PWM 占空比算法不同，我们不妨把 PWM8DATA 的值分为两个部分：基本输出周期控制部分（DC）和额外输出周期控制部分（AC）。

12.4.1 6+2 模式 PWM 占空比

当选择 6+2 模式的时候，PWMDATA 的高 6 位为基本输出周期，低 2 位为额外输出周期。根据额外输出周期的不同，在基本输出周期的基础上分 4 个周期的补偿。

6+2 模式 8 位 PWM 输出示意图：



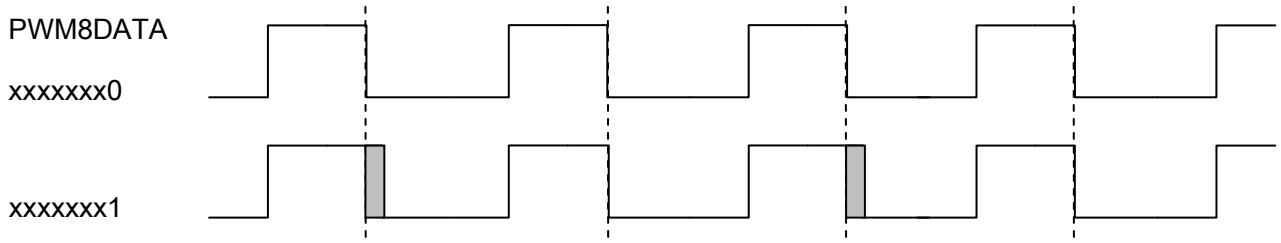
如上图所示，当基本输出周期为 DC，那么不同的额外输出周期时，PWM8 的连续 4 个实际周期分别为：

额外输出周期	周期 1	周期 2	周期 3	周期 4
00	DC/64	DC/64	DC/64	DC/64
01	DC/64+1	DC/64	DC/64	DC/64
10	DC/64+1	DC/64+1	DC/64	DC/64
11	DC/64+1	DC/64+1	DC/64+1	DC/64

12.4.2 7+1 模式 PWM 占空比

当选择 7+1 模式的时候，PWMDATA 的高 7 位为基本输出周期，最低位为额外输出周期。根据额外输出周期的不同，在基本输出周期的基础上分 2 个周期的补偿。

7+1 模式 8 位 PWM 输出示意图：



如上图所示，当基本输出周期为 DC，那么不同的额外输出周期时，PWM8 的连续 2 个实际周期分别为：

额外输出周期	周期 1	周期 2
0	DC/64	DC/64
1	DC/64+1	DC/64

12.5 8 位 PWM 应用

PWM8 的应用设置需的操作流程如下：

- 设置 PWM8 工作模式及时钟
- 设置 PWM8DATA
- P0,6 设置为 PWM8 输出口
- PWM8 开始工作

例：PWM8 的设置程序

```
        LDIA    B'XX110000'
        LD      PWM8CON,A       ;FPWM=FOSC、"6+2"模式
        LDIA    B'10000001'
        LD      PWM8DATA,A      ;设置 PWM8 占空比
        LDIA    B'XX01XXXX'
        LD      P0CH,A          ;P0,6 设置为 PWM 输出口
        SETB    PWM8CON,0       ;开启 PWM8
```

13. 10 位 PWM（PWM1）

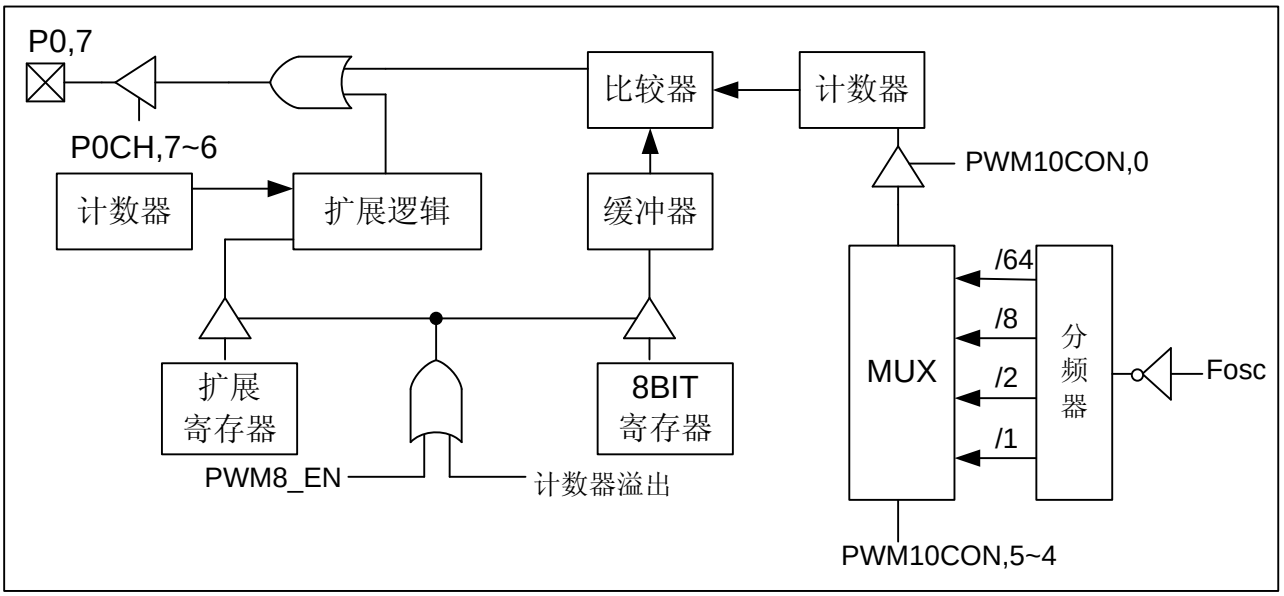
13.1 10 位 PWM 概述

PWM10 由如下功能组成：

- 选择时钟频率
- 10-bit 计数 器（PWM10CON），8-bit 比较器，8-bit 数据存储器（PWM10DATA）,和 8-bit 数据缓冲器。
- 2-bit 扩展逻辑，2-bit 扩展寄存器和数据缓冲器。
- 控制寄存器（PWM10CON）。

计数器的高 8 位和 PWM10 数据存储器相比较（PWM10DATA）来确定 PWM10 的工作频率。为了更高的精度，计数器的低 2 位可被用来作为扩展周期。

- PWM10 结构框图



当计数器的高 8 位和相关的数据寄存器（PWM10DATA）匹配时，PWM 输出低电平。如果 PWM10DATA 寄存器的值不是 0，计数器的高 8 位溢出将使 PWM 输出高电平。这样的话，写进相关寄存器的值就决定了模块的基本工作周期。

计数器的低 2 位值和 2 位扩展数据寄存器（PWM10CON.7-6）的扩展设置比较。计数器值的低 2 位被用来扩展 PWM 的输出工作周期。扩展值是一个在特殊周期的额外时钟周期（如下表）。

PWM10CON.7-6	扩展周期
00	None
01	1
10	1,2
11	1,2,3

13.2 与 10 位 PWM 相关寄存器

有两个寄存器与PWM10有关，PWM10数据寄存器PWM10DATA、PWM10控制寄存器PWM10CON。

1EH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
PWM10DATA								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

1FH	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
PWM10CON	扩展周期选择		PWM10时钟选择		未用	加载选择	-	PWM10_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

bit 7-bit 6 **扩展周期选择**

00： 无扩展周期

01： 扩展周期为 1

10： 扩展周期为 1、2

11： 扩展周期为 1、2、3

bit 5-bit 4 **PWM10 时钟选择**

00： PWM10 时钟为 Fosc/64

01： PWM10 时钟为 Fosc/8

10： PWM10 时钟为 Fosc/2

11： PWM10 时钟为 Fosc/1

bit 3 未用

bit 2 **加载选择位**

0：数据缓冲在 10 位溢出时加载

1：数据缓冲在 8 位溢出时加载

bit 0 **PWM10_EN：PWM10 使能控制位**

0：PWM10 停止工作

1：PWM10 允许工作

13.3 10 位 PWM 调制周期

13.3.1 10 位 PWM 调制周期

10 位 PWM 调制周期由系统主频（Fosc）、PWM10 分频比，计算公式如下：

$$\text{PWM10 调制周期} = 2^8 \times \text{PWM10 分频比} \div F_{OSC}$$

例：Fosc=8MHz，分频比 1：1，时 PWM 调制周期

$$\begin{aligned}\text{PWM10 调制周期} &= 2^8 \times \text{PWM10 分频比} \div F_{OSC} \\ &= 2^8 \times 1 \div (8 \times 10^6)\ s \\ &= 32 \times 10^{-6}\ s \\ &= 32\mu s\end{aligned}$$

例：Fosc=8MHz 时 PWM10 的调制周期表

PWM10CON BIT5、BIT4	PWM10 时钟	Fosc=8MHz
00	Fosc /64	2048 μs
01	Fosc /8	256 μs
10	Fosc /2	64 μs
11	Fosc /1	32 μs

13.3.2 10 位 PWM 输出周期

4 个调制周期为一个输出周期。

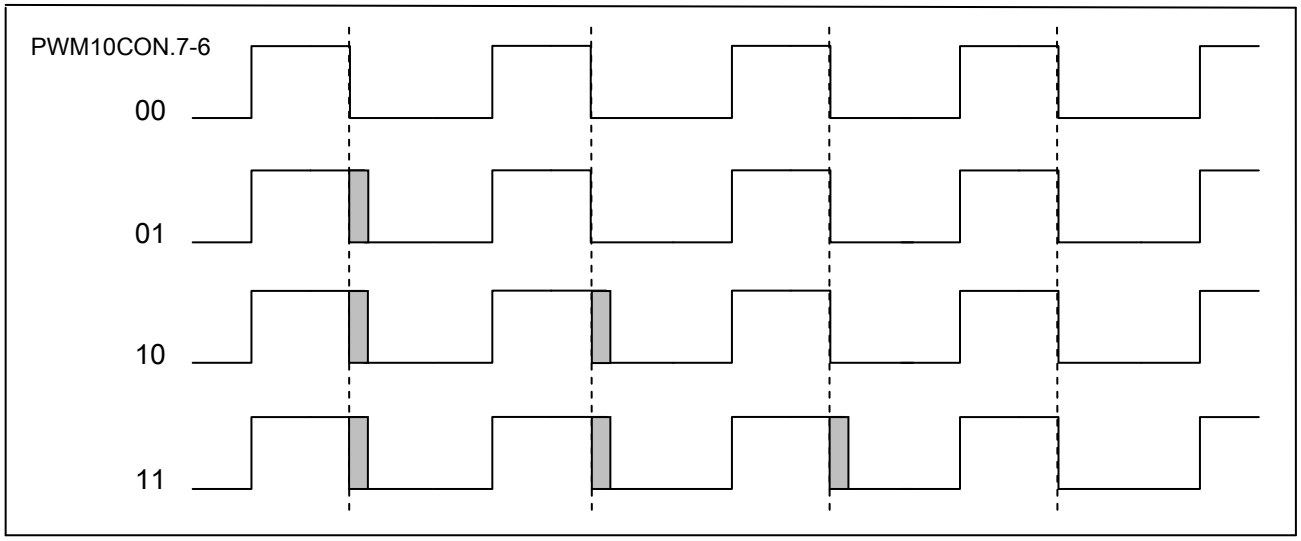
13.4 10 位 PWM 占空比算法

10 位 PWM 输出的占空比与 PWM10DATA（DC）及 PWM10CON,7～6(AC)的数值相关。从整体上来看，PWM10 的占空比可用下面公式计算：

PWM10 的占空比 = （PWM10 输出周期×4 + PWM10 额外周期）÷ 1024

实际输出时，PWM10 的占空比根据扩展位的不同，分 4 个周期不同的输出。

PWM10 占空比输出示意图



如上图所示，当基本输出周期为 PWM10DATA，那么不同的扩展周期时，PWM10 的连续 4 个实际周期分别为：

额外输出周期	周期 1	周期 2	周期 3	周期 4
00	PWM10DATA/256	PWM10DATA/256	PWM10DATA/256	PWM10DATA/256
01	PWM10DATA/256+1	PWM10DATA/256	PWM10DATA/256	PWM10DATA/256
10	PWM10DATA/256+1	PWM10DATA/256+1	PWM10DATA/256	PWM10DATA/256
11	PWM10DATA/256+1	PWM10DATA/256+1	PWM10DATA/256+1	PWM10DATA/256

13.5 10 位 PWM 应用

PWM10 的应用设置需的操作流程如下：

- 设置 PWM10 工作模式及时钟
- 设置 PWM10DATA
- P0,7 设置为 PWM10 输出口
- PWM10 开始工作

例：PWM10 的设置程序

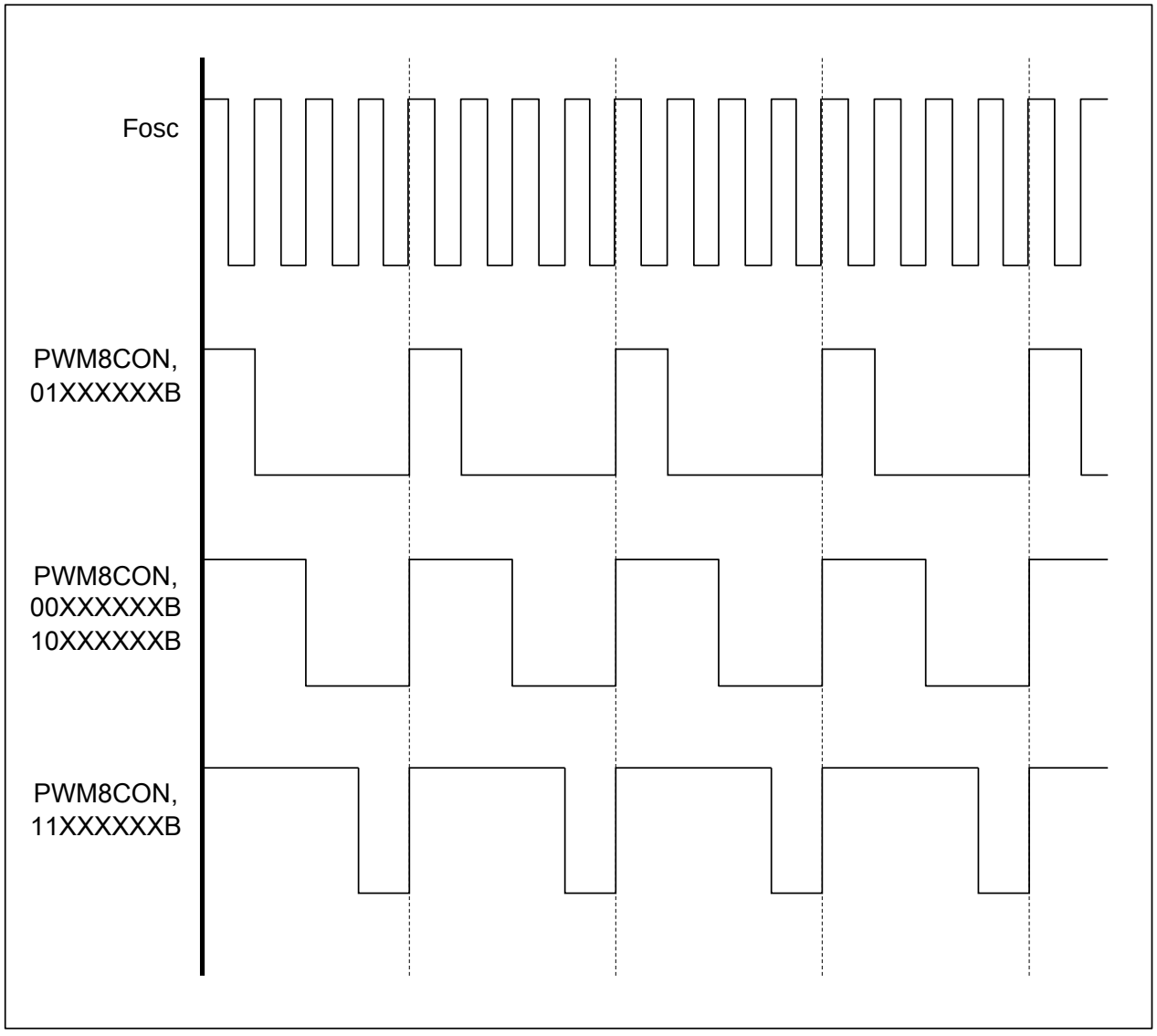
```
LDIA    B'00110010'
LD      PWM10CON,A      ;FPWM=FOSC, 8 位溢出加载，无扩展周期
LDIA    B'10000001'
LD      PWM10DATA,A     ;设置 PWM10 占空比
LDIA    B'01XXXXXX'
LD      P0CH,A          ;P0.7 设置为 PWM 输出口
SETB    PWM10CON,0      ;开启 PWM10
```

14. 高频时钟（CLO）输出

14.1 高频时钟（CLO）输出概述

CMS69F012/013 有一个高频脉冲宽度调制器 PWM2，其输出频率为系统时钟的 4 分频，占空比为 25%、50%、75% 可调，由 PWM8CON.7～6 控制，具体请参照关于 PWM8CON 的说明表格。

14.2 高频时钟（CLO）输出波形



14.3 高频时钟（CLO）应用

CLO 的应用设置需的操作流程如下：

- 设置 CLO 占空比
- P1,6 设置为 CLO 输出口

★ 例：CLO 的设置程序

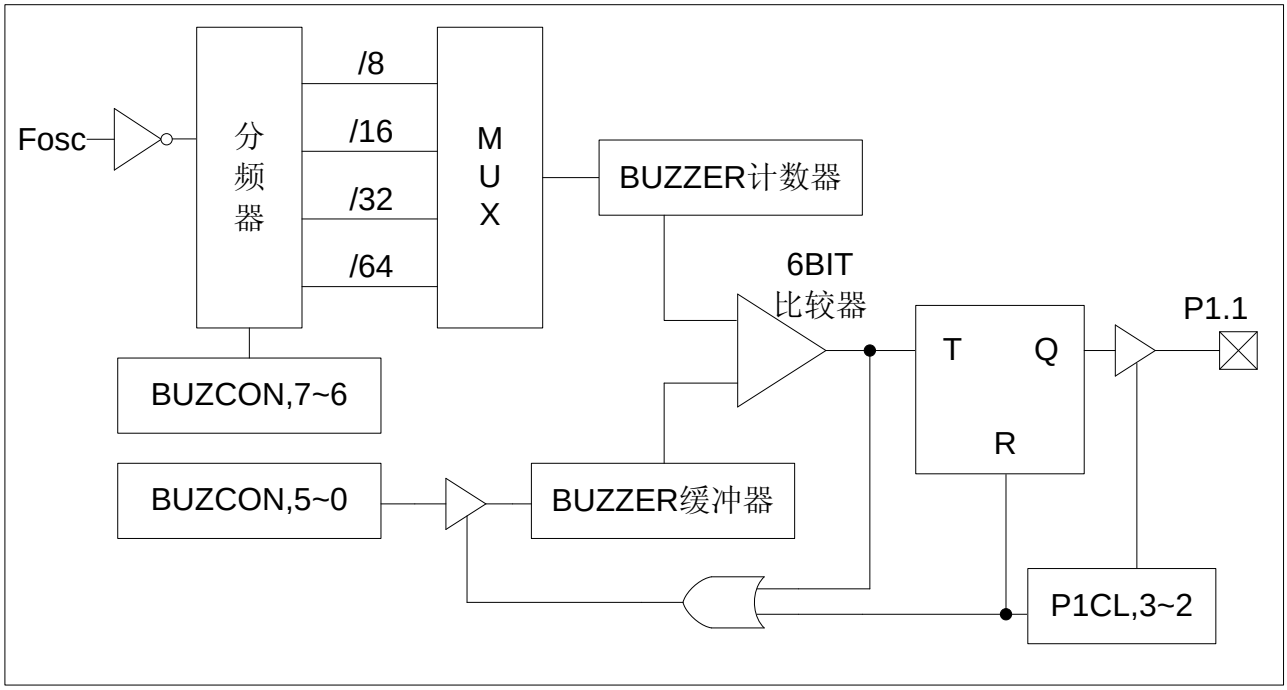
```
        LDIA    B'00XXXXXX'
        LD      PWM8CON,A       ；CLO 占空比 50%
        LDIA    B'111XXXXX'
        LD      P1CH,A          ；P1,6 设置为 CLO 输出口
```

15. 蜂鸣器输出（BUZZER）

15.1 BUZZER 概述

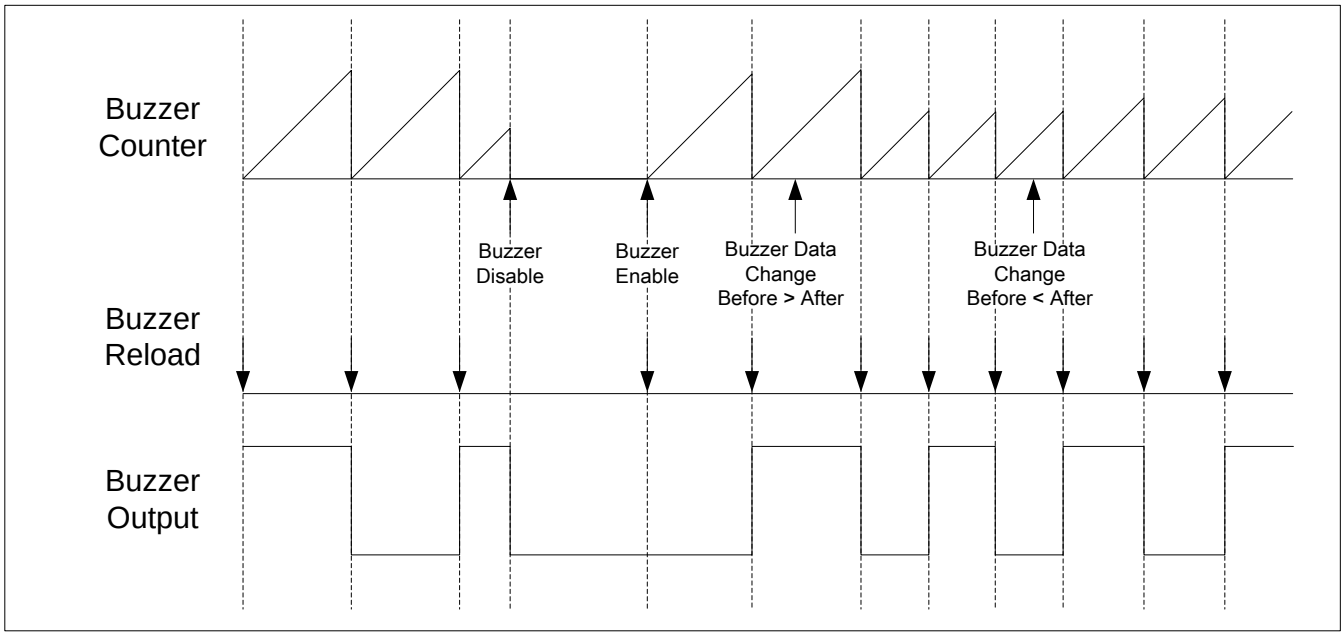
CMS69F012/013 的蜂鸣驱动器由 6-BIT 计数器，时钟驱动器，控制寄存器组成。它产生 50%的占空方波，其频率覆盖一个较宽的范围。BUZZER 的输出频率由 BUZCON 寄存器的值控制。

◆ BUZZER 结构框图



设置 P1.1 的控制寄存器，即将 P1CL 的 B3,B2 设为 01，可使蜂鸣输出功能处于使能状态，当蜂鸣输出使能时，6-BIT 计数器被清零，P1.1 输出状态为 0，开始往上计数。如果计数器值和周期数据（BUZCON.5-0）相符，则 P1.1 输出状态被固定，计数器被清零。另外，6-BIT 计数器溢出也可使计数器清零，BZCON.5-0 决定输出频率。

BUZZER 输出时序图



15.2 与 BUZZER 相关的寄存器

21H	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
BUZCON	时钟选择		BUZDATA					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

BIT7～BIT6 **时钟选择**

00： BUZZER 时钟为 Fosc/8

01： BUZZER 时钟为 Fosc/16

10： BUZZER 时钟为 Fosc/32

11： BUZZER 时钟为 Fosc/64

BIT5～BIT0 **BUZDATA:BUZ 输出周期数据**

15.3 BUZZER 输出频率

15.3.1 BUZZER 输出频率计算方法

BUZZER 输出频率 =Fosc÷[2×分频比×（BUZDATA+1）]

例：Fosc=4MHz BUZDATA=4 ,BUZZER 时钟为 Fosc/8,时 BUZZER 的输出频率

BUZZER 输出频率 =Fosc÷[2×分频比×（BUZDATA+1）]

$=4\times10^6\div[2\times8\times(4+1)]$

$=5\times10^4$

=50kHz

15.3.2 BUZZER 输出频率表

BUZCON BIT7、BIT6	BUZZER 计数时钟	Fosc=8MHz	
		BUZZER 最小输出频率	BUZZER 最大输出频率
00	Fosc /8	1.95kHz	500kHz
01	Fosc /16	0.98kHz	250kHz
10	Fosc /32	0.49kHz	125kHz
11	Fosc /64	0.25kHz	62.5kHz

15.4 BUZZER 应用

BUZZER 应用设置的操作流程如下：

◆ 设置 BUZZER 频率

◆ P1,1 设置为 BUZZER 输出口

★ 例：BUZZER 的设置程序

```
        LDIA    B'00000001'
        LD      BUZCON,A        ;
        LDIA    B'XXXX01XX'
        LD      P1CL,A          ;
        CALL    DELY_TIME
        LDIA    B'XXXX10XX'
        LD      P1CL,A
```

16. 电气参数

16.1 DC 特性

符 号	参 数	测试条件		最 小	典 型	最 大	单 位
		VDD	条件				
VDD	工作电压	-	Fsys=INTRC	2.5	-	5.5	V
Idd	工作电流	5V		-	3	-	mA
		3V		-	2	-	mA
Istb	静态电流	5V	----	0.1	1	10	μA
		3V	----	0.01	0.1	1	μA
Vil	低电平输入电压	-	----	-	-	0.3VDD	V
Vih	高电平输入电压	-	----	0.7VDD	-	-	V
Voh	高电平输出电压	-	不带负载	0.9VDD	-	-	V
VoL	低电平输出电压	-	不带负载	-	-	0.1VDD	V
Rph	上拉电阻阻值	5V	----	-	35	-	K
		3V	----	-	65	-	K
Rpl	下拉电阻阻值	5V	----	-	45	-	K
		3V	----	-	100	-	K
	比较器精度	-	----	-	20	-	mV
IoL	输入口灌电流	5V	Vol=0.3VDD	-	60	-	mV
		3V	Vol=0.3VDD	-	25	-	mA
IoH	输入口拉电流	5V	VOH=0.7VDD	-	15	-	mA
		3V	VOH=0.7VDD	-	10	-	mA

16.2 AC 特性

符 号	参 数	测试条件		最 小	典 型	最 大	单 位
		VDD	条件				
TWDT	WDT 复位时间	5V	---	-	18	-	ms
		3V	---	-	34	-	ms

16.3 内部 RC 振荡特性

16.3.1 内部 RC 振荡电压特性

测试条件	振荡频率（典型值）（Hz）
2.5V	8.0M
2.6V	8.1M
2.8V	8.2M
3.0V	8.3M
3.2V	8.3M
3.4V	8.2M
3.6V	8.2M
3.8V	8.2M
4.0V	8.1M
4.2V	8.1M
4.4V	8.1M
4.6V	8.0M
4.8V	8.0M
5.0V	8.0M
5.2V	8.0M
5.4V	7.9M
5.5V	7.8M

16.3.2 内部 RC 振荡温度特性

测试条件	-20℃	25℃	40℃	60℃	85℃
振荡频率（典型值）（Hz）	7.9M	8.0M	8.0M	8.1M	8.1M

17. 指令

17.1 指令一览表

助记符	操 作	指令周期	标 志
控制类-4			
NOP	空操作	1	None
OPTION	装载 OPTION 寄存器	1	None
STOP	进入休眠模式	1	TO,PD
CLRWDT	清零看门狗计数器	1	TO,PD
数据传送-4			
LD [R],A	将 ACC 内容传送到 R	1	NONE
LD A,[R]	将 R 内容传送到 ACC	1	Z
TESTZ R	将数据存储器内容传给数据存储器	1	Z
LDIA i	立即数 i 送给 ACC	1	NONE
逻辑运算-16			
CLRA	清零 ACC	1	Z
SET [R]	置位数据存储器 R	1	NONE
CLR [R]	清零数据存储器 R	1	Z
ORA [R]	R 与 ACC 内容做“或”运算，结果存入 ACC	1	Z
ORR [R]	R 与 ACC 内容做“或”运算，结果存入 R	1	Z
ANDA [R]	R 与 ACC 内容做“与”运算，结果存入 ACC	1	Z
ANDR [R]	R 与 ACC 内容做“与”运算，结果存入 R	1	Z
XORA [R]	R 与 ACC 内容做“异或”运算，结果存入 ACC	1	Z
XORR [R]	R 与 ACC 内容做“异或”运算，结果存入 R	1	Z
SWAPA [R]	R 寄存器内容的高低半字节转换，结果存入 ACC	1	NONE
SWAPR [R]	R 寄存器内容的高低半字节转换，结果存入 R	1	NONE
COMA [R]	R 寄存器内容取反，结果存入 ACC	1	Z
COMR [R]	R 寄存器内容取反，结果存入 R	1	Z
XORIA i	ACC 与立即数 i 做“异或”运算，结果存入 ACC	1	Z
ANDIA i	ACC 与立即数 i 做“与”运算，结果存入 ACC	1	Z
ORIA i	ACC 与立即数 i 做“或”运算，结果存入 ACC	1	Z
移位操作，8			
RRCA [R]	数据存储器带进位循环右移一位，结果存入 ACC	1	C
RRCR [R]	数据存储器带进位循环右移一位，结果存入 R	1	C
RLCA [R]	数据存储器带进位循环左移一位，结果存入 ACC	1	C
RLCR [R]	数据存储器带进位循环左移一位，结果存入 R	1	C
RLA [R]	数据存储器不带进位循环左移一位，结果存入 ACC	1	NONE
RLR [R]	数据存储器不带进位循环左移一位，结果存入 R	1	NONE
RRA [R]	数据存储器不带进位循环右移一位，结果存入 ACC	1	NONE

RRR [R]	数据存储器不带进位循环右移一位，结果存入 R	1	NONE
递增递减，4			
INCA [R]	递增数据存储器 R，结果放入 ACC	1	Z
INCR [R]	递增数据存储器 R，结果放入 R	1	Z
DECA [R]	递减数据存储器 R，结果放入 ACC	1	Z
DECR [R]	递减数据存储器 R，结果放入 R	1	Z
位操作，2			
CLRB [R],b	将数据存储器 R 中某位清零	1	NONE
SETB [R],b	将数据存储器 R 中某位置一	1	NONE
查表，2			
TABLE [R]	读取 FLASH 内容结果放入 TABLE_DATAH 与 R	2	NONE
TABLEA	读取 FLASH 内容结果放入 TABLE_DATAH 与 ACC	2	NONE
数学运算，16			
ADDA [R]	ACC+[R]→ACC	1	C,DC,Z,OV
ADDR [R]	ACC+[R]→R	1	C,DC,Z,OV
ADDCA [R]	ACC+[R]+C→ACC	1	Z,C,DC,OV
ADDCR [R]	ACC+[R]+C→R	1	Z,C,DC,OV
ADDIA i	ACC+i→ACC	1	Z,C,DC,OV
SUBA [R]	[R]-ACC→ACC	1	C,DC,Z,OV
SUBR [R]	[R]-ACC→R	1	C,DC,Z,OV
SUBCA [R]	[R]-ACC-C→ACC	1	Z,C,DC,OV
SUBCR [R]	[R]-ACC-C→R	1	Z,C,DC,OV
SUBIA i	i-ACC→ACC	1	Z,C,DC,OV
HSUBA [R]	ACC-[R]→ACC	1	Z,C,DC,OV
HSUBR [R]	ACC-[R]→R	1	Z,C,DC,OV
HSUBCA [R]	ACC-[R]- $\overline{C}$ →ACC	1	Z,C,DC,OV
HSUBCR [R]	ACC-[R]- $\overline{C}$ →R	1	Z,C,DC,OV
HSUBIA i	ACC-i→ACC	1	Z,C,DC,OV
无条件转移，5			
RET	从子程序返回	2	NONE
RET i	从子程序返回，并将立即数 I 存入 ACC	2	NONE
RETI	从中断返回	2	NONE
CALL ADD	子程序调用	2	NONE
JP ADD	无条件跳转	2	NONE
条件转移，8			
SZB [R],b	如果数据存储器 R 的 b 位为“0”，则跳过下一条指令	1 or 2	NONE
SNZB [R],b	如果数据存储器 R 的 b 位为“1”，则跳过下一条指令	1 or 2	NONE
SZA [R]	数据存储器 R 送至 ACC，若内容为”0”，则跳过下一条指令	1 or 2	NONE
SZR [R]	数据存储器 R 内容为”0”，则跳过下一条指令	1 or 2	NONE
SZINCA [R]	数据存储器 R 加“1”，结果放入 ACC，若结果为”0”，则跳过下一条指令	1 or 2	NONE
SZINCR [R]	数据存储器 R 加“1”，结果放入 R，若结果为”0”，则跳过下一条指令	1 or 2	NONE

SZDECA [R]	数据存储器 R 减“1”，结果放入 ACC，若结果为”0”，则跳过下一条指令	1 or 2	NONE
SZDECR [R]	数据存储器 R 减“1”，结果放入 R，若结果为”0”，则跳过下一条指令	1 or 2	NONE

17.2 指令说明

ADDA [R]

操作: 将 R 加 ACC，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDA    R01     ;执行结果：ACC= 09H + 77H = 80H
```

ADDR [R]

操作: 将 R 加 ACC，结果放入 R

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDR    R01     ;执行结果：R01= 09H + 77H = 80H
```

ADDCA [R]

操作: 将 R 加 ACC 加 C 位，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDCA   R01     ;执行结果：ACC = 09H + 77H + C = 80H (C=0)
                          ACC = 09H + 77H + C = 81H (C=1)
```

ADDCR [R]

操作: 将 R 加 ACC 加 C 位，结果放入 R

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDCR   R01     ;执行结果：R01 = 09H + 77H + C = 80H (C=0)
```

R01 = 09H + 77H + C = 81H (C=1)

ADDIA i

操作: 将立即数 i 加 ACC，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
ADDIA   077H    ;执行结果：ACC = ACC(09H) + i(77H)=80H
```

ANDA [R]

操作: 寄存器 R 跟 ACC 进行逻辑与运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0FH     ;给 ACC 赋值 0FH
LD      R01,A   ;将 ACC 的值(0FH)赋给寄存器 R01
LDIA    77H     ;给 ACC 赋值 77H
ANDA    R01     ;执行结果：ACC=(0FH  and  77H)=07H
```

ANDR [R]

操作: 寄存器 R 跟 ACC 进行逻辑与运算，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0FH     ;给 ACC 赋值 0FH
LD      R01,A   ;将 ACC 的值(0FH)赋给寄存器 R01
LDIA    77H     ;给 ACC 赋值 77H
ANDR    R01     ;执行结果：R01=(0FH  and  77H)=07H
```

ANDIA i

操作: 将立即数 i 与 ACC 进行逻辑与运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0FH     ;给 ACC 赋值 0FH
ANDIA   77H     ;执行结果：ACC =(0FH  and  77H)=07H
```

CALL add

操作: 调用子程序
周期: 2
影响标志位: 无
举例:

```
CALL    LOOP    ;调用名称定义为"LOOP"的子程序地址
```

CLRA

操作: ACC 清零
周期: 1
影响标志位: Z
举例:

```
CLRA            ;执行结果：ACC=0
```

CLR [R]

操作: 寄存器 R 清零
周期: 1
影响标志位: Z
举例:

```
CLR     R01     ;执行结果：R01=0
```

CLRB [R],b

操作: 寄存器 R 的第 b 位清零
周期: 1
影响标志位: 无
举例:

```
CLRB    R01,3   ;执行结果：R01 的第 3 位为零
```

CLRWDT

操作: 清零看门狗计数器
周期: 1
影响标志位: TO，PD
举例:

```
CLRWDT          ;看门狗计数器清零
```

COMA [R]

操作: 寄存器 R 取反，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
COMA    R01     ;执行结果：ACC=0F5H
```

COMR [R]

操作: 寄存器 R 取反，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
COMR    R01     ;执行结果：R01=0F5H
```

DECA [R]

操作: 寄存器 R 自减 1，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
DECA    R01     ;执行结果：ACC=(0AH-1)=09H
```

DECR [R]

操作: 寄存器 R 自减 1，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
DECR    R01     ;执行结果：R01=(0AH-1)=09H
```

HSUBA [R]

操作: ACC 减 R，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBA   R01     ;执行结果：ACC=(80H-77H)=09H
```

HSUBR [R]

操作: ACC 减 R，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBR   R01     ;执行结果：R01=(80H-77H)=09H
```

HSUBCA [R]

操作: ACC 减 R 减 C，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBCA  R01     ;执行结果：ACC=(80H-77H-C)=09H(C=0)
                          ACC=(80H-77H-C)=08H(C=1)
```

HSUBCR [R]

操作: ACC 减 R 减 C，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBCR  R01     ;执行结果：R01=(80H-77H-C)=09H(C=0)
                          R01=(80H-77H-C)=08H(C=1)
```

INCA [R]

操作: 寄存器 R 自加 1，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
INCA    R01     ;执行结果：ACC=(0AH+1)=0BH
```

INCR [R]

操作: 寄存器 R 自加 1，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
INCR    R01     ;执行结果：R01=(0AH+1)=0BH
```

JP add

操作: 跳转到 add 地址

周期: 2

影响标志位: 无

举例:

```
JP      LOOP    ;跳转至名称定义为"LOOP"的子程序地址
```

LD A，[R]

操作: 将 R 的值赋给 ACC

周期: 1

影响标志位: Z

举例:

```
LD      A,R01   ;将寄存器 R0 的值赋给 ACC
LD      R02,A   ;将 ACC 的值赋给寄存器 R02，实现了数据从 R01→R02 的移动
```

LD [R]，A

操作: 将 ACC 的值赋给 R

周期: 1

影响标志位: 无

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
```

```
LD          R01,A          ;执行结果：R01=09H
```

LDIA i

操作: 立即数 i 赋给 ACC
周期: 1
影响标志位: 无
举例:

```
LDIA        0AH            ;ACC 赋值 0AH
```

NOP

操作: 空指令
周期: 1
影响标志位: 无
举例:

```
NOP
NOP
```

OPTION

操作: 写预分频器
周期: 1
影响标志位: 无
举例:

```
LDIA        00H
OPTION                     ;OPTION 赋值 00H，预分频器给 TMR0 用，分频比 1:2
```

ORIA i

操作: 立即数与 ACC 进行逻辑或操作，结果赋给 ACC
周期: 1
影响标志位: Z
举例:

```
LDIA        0AH            ;ACC 赋值 0AH
ORIA        030H           ;执行结果：ACC =(0AH  or  30H)=3AH
```

ORA [R]

操作: 寄存器 R 跟 ACC 进行逻辑或运算，结果放入 ACC
周期: 1
影响标志位: Z
举例:

```
LDIA        0AH            ;给 ACC 赋值 0AH
LD          R01,A          ;将 ACC(0AH)赋给寄存器 R01
LDIA        30H            ;给 ACC 赋值 30H
ORA         R01            ;执行结果：ACC=(0AH  or  30H)=3AH
```

ORR [R]

操作: 寄存器 R 跟 ACC 进行逻辑或运算，结果放入 R

周期: 1

影响标志位: Z

举例:

```
        LDIA    0AH         ;给 ACC 赋值 0AH
        LD      R01,A       ;将 ACC(0AH)赋给寄存器 R01
        LDIA    30H         ;给 ACC 赋值 30H
        ORR     R01         ;执行结果：R01=(0AH  or  30H)=3AH
```

RET

操作: 从子程序返回

周期: 2

影响标志位: 无

举例:

```
        CALL    LOOP        ;调用子程序 LOOP
        NOP                 ;RET 指令返回后将执行这条语句
        …                   ;其它程序
LOOP:
        …                   ;子程序
        RET                 ;子程序返回
```

RET i

操作: 从子程序带参数返回，参数放入 ACC

周期: 2

影响标志位: 无

举例:

```
        CALL    LOOP        ;调用子程序 LOOP
        NOP                 ;RET 指令返回后将执行这条语句
        …                   ;其它程序
LOOP:
        …                   ;子程序
        RET     35H         ;子程序返回,ACC=35H
```

RETI

操作: 中断返回

周期: 2

影响标志位: 无

举例:

```
INT_START                   ;中断程序入口
```

```
        …                       ;中断处理程序
        RETI                    ;中断返回
```

RLCA [R]

操作: 寄存器 R 带 C 循环左移一位，结果放入 ACC

周期: 1

影响标志位: C

举例:

```
        LDIA    03H             ;ACC 赋值 03H
        LD      R01,A           ;ACC 值赋给 R01,R01=03H
        RLCA    R01             ;操作结果：ACC=06H(C=0);
                                        ACC=07H(C=1)
                                        C=0
```

RLCR [R]

操作: 寄存器 R 带 C 循环左移一位，结果放入 R

周期: 1

影响标志位: C

举例:

```
        LDIA    03H             ;ACC 赋值 03H
        LD      R01,A           ;ACC 值赋给 R01,R01=03H
        RLCR    R01             ;操作结果：R01=06H(C=0);
                                        R01=07H(C=1);
                                        C=0
```

RLA [R]

操作: 寄存器 R 不带 C 循环左移一位，结果放入 ACC

周期: 1

影响标志位: 无

举例:

```
        LDIA    03H             ;ACC 赋值 03H
        LD      R01,A           ;ACC 值赋给 R01,R01=03H
        RLA     R01             ;操作结果：ACC=06H
```

RLR [R]

操作: 寄存器 R 不带 C 循环左移一位，结果放入 R

周期: 1

影响标志位: 无

举例:

```
        LDIA    03H             ;ACC 赋值 03H
        LD      R01,A           ;ACC 值赋给 R01,R01=03H
        RLR     R01             ;操作结果：R01=06H
```

RRCA [R]

操作: 寄存器 R 带 C 循环右移一位，结果放入 ACC

周期: 1

影响标志位: C

举例:

```
LDIA    03H     ;ACC 赋值 03H
LD      R01,A   ;ACC 值赋给 R01,R01=03H
RRCA    R01     ;操作结果：ACC=01H(C=0);
                        ACC=081H(C=1);
                        C=1
```

RRCR [R]

操作: 寄存器 R 带 C 循环右移一位，结果放入 R

周期: 1

影响标志位: C

举例:

```
LDIA    03H     ;ACC 赋值 03H
LD      R01,A   ;ACC 值赋给 R01,R01=03H
RRCR    R01     ;操作结果：R01=01H(C=0);
                        R01=81H(C=1);
                        C=1
```

RRA [R]

操作: 寄存器 R 不带 C 循环右移一位，结果放入 ACC

周期: 1

影响标志位: 无

举例:

```
LDIA    03H     ;ACC 赋值 03H
LD      R01,A   ;ACC 值赋给 R01,R01=03H
RRA     R01     ;操作结果：ACC=81H
```

RRR [R]

操作: 寄存器 R 不带 C 循环右移一位，结果放入 R

周期: 1

影响标志位: 无

举例:

```
LDIA    03H     ;ACC 赋值 03H
LD      R01,A   ;ACC 值赋给 R01,R01=03H
RRR     R01     ;操作结果：R01=81H
```

SET [R]

操作: 寄存器 R 所有位置 1

周期: 1

影响标志位: 无

举例:

```
SET     R01     ;操作结果：R01=0FFH
```

SETB [R],b

操作: 寄存器 R 的第 b 位置 1

周期: 1

影响标志位: 无

举例:

```
CLR     R01     ;R01=0
SETB    R01,3   ;操作结果：R01=08H
```

STOP

操作: 进入休眠状态

周期: 1

影响标志位: TO，PD

举例:

```
STOP            ;芯片进入省电模式，CPU、振荡器停止工作，IO 口保持原来状态。
```

SUBIA i

操作: 立即数 i 减 ACC，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 77H
SUBIA   80H     ;操作结果：ACC=80H-77H=09H
```

SUBA [R]

操作: 寄存器 R 减 ACC，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H    ;ACC 赋值 80H
LD      R01,A   ;ACC 的值赋给 R01，R01=80H
LDIA    77H     ;ACC 赋值 77H
SUBA    R01     ;操作结果：ACC=80H-77H=09H
```

SUBR [R]

操作: 寄存器 R 减 ACC，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H    ;ACC 赋值 80H
LD      R01,A   ;ACC 的值赋给 R01，R01=80H
LDIA    77H     ;ACC 赋值 77H
SUBR    R01     ;操作结果：R01=80H-77H=09H
```

SUBCA [R]

操作: 寄存器 R 减 ACC 减 C，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H    ;ACC 赋值 80H
LD      R01,A   ;ACC 的值赋给 R01，R01=80H
LDIA    77H     ;ACC 赋值 77H
SUBCA   R01     ;操作结果：ACC=80H-77H-C=09H(C=0);
                          ACC=80H-77H-C=08H(C=1);
```

SUBCR [R]

操作: 寄存器 R 减 ACC 减 C，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H    ;ACC 赋值 80H
LD      R01,A   ;ACC 的值赋给 R01，R01=80H
LDIA    77H     ;ACC 赋值 77H
SUBCR   R01     ;操作结果：R01=80H-77H-C=09H(C=0)
                          R01=80H-77H-C=08H(C=1)
```

SWAPA [R]

操作: 寄存器 R 高低半字节交换，结果放入 ACC

周期: 1

影响标志位: 无

举例:

```
LDIA    035H    ;ACC 赋值 35H
LD      R01,A   ;ACC 的值赋给 R01，R01=35H
SWAPA   R01     ;操作结果：ACC=53H
```

SWAPR [R]

操作: 寄存器 R 高低半字节交换，结果放入 R

周期: 1

影响标志位: 无

举例:

```
LDIA    035H    ;ACC 赋值 35H
LD      R01,A   ;ACC 的值赋给 R01，R01=35H
SWAPR   R01     ;操作结果：R01=53H
```

SZB [R],b

操作: 判断寄存器 R 的第 b 位，为 0 间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZB     R01,3   ;判断寄存器 R01 的第 3 位
JP      LOOP    ;R01 的第 3 位为 1 才执行这条语句，跳转至 LOOP
JP      LOOP1   ;R01 的第 3 位为 0 时间跳，执行这条语句，跳转至 LOOP1
```

SNZB [R],b

操作: 判断寄存器 R 的第 b 位，为 1 间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SNZB    R01,3   ;判断寄存器 R01 的第 3 位
JP      LOOP    ;R01 的第 3 位为 0 才执行这条语句，跳转至 LOOP
JP      LOOP1   ;R01 的第 3 位为 1 时间跳，执行这条语句，跳转至 LOOP1
```

SZA [R]

操作: 将寄存器 R 的值赋给 ACC，若 R 为 0 则间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZA     R01     ;R01→ACC
JP      LOOP    ;R01 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1   ;R01 为 0 时间跳，执行这条语句，跳转至 LOOP1
```

SZR [R]

操作: 将寄存器 R 的值赋给 R，若 R 为 0 则间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZR     R01     ;R01→R01
JP      LOOP    ;R01 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1   ;R01 为 0 时间跳执行这条语句，跳转至 LOOP1
```

SZINCA [R]

操作: 将寄存器 R 自加 1，结果放入 ACC，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZINCA  R01     ;R01+1→ACC
JP      LOOP    ;ACC 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1   ;ACC 为 0 时执行这条语句，跳转至 LOOP1
```

SZINCR [R]

操作: 将寄存器 R 自加 1，结果放入 R，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZINCR  R01     ;R01+1→R01
JP      LOOP    ; R01 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1   ; R01 为 0 时执行这条语句，跳转至 LOOP1
```

SZDECA [R]

操作: 将寄存器 R 自减 1，结果放入 ACC，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZDECA  R01     ;R01-1→ACC
JP      LOOP    ;ACC 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1   ;ACC 为 0 时执行这条语句，跳转至 LOOP1
```

SZDECR [R]

操作: 将寄存器 R 自减 1，结果放入 R，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZDECR    R01       ;R01-1→R01
JP        LOOP      ; R01 不为 0 时执行这条语句，跳转至 LOOP
JP        LOOP1     ; R01 为 0 时执行这条语句，跳转至 LOOP1
```

TABLE [R]

操作: 查表，查表结果低 8 位放入 R，高位放入专用寄存器 TABLE_SPH

周期: 2

影响标志位: 无

举例:

```
LDIA      01H           ;ACC 赋值 01H
LD        TABLE_SPH,A   ;ACC 值赋给表格高位地址，TABLE_SPH=1
LDIA      015H          ;ACC 赋值 15H
LD        TABLE_SPL,A   ;ACC 值赋给表格地位地址，TABLE_SPL=15H
TABLE     R01           ;查表 0115H 地址，操作结果：TABLE_DATAH=12H，R01=34H
…
ORG       0115H
DW        1234H
```

TABLEA

操作: 查表，查表结果低 8 位放入 ACC，高位放入专用寄存器 TABLE_SPH

周期: 2

影响标志位: 无

举例:

```
LDIA      01H           ;ACC 赋值 01H
LD        TABLE_SPH,A   ;ACC 值赋给表格高位地址，TABLE_SPH=1
LDIA      015H          ;ACC 赋值 15H
LD        TABLE_SPL,A   ;ACC 值赋给表格地位地址，TABLE_SPL=15H
TABLEA                  ;查表 0115H 地址，操作结果：TABLE_DATAH=12H，ACC=34H
…
ORG       0115H
DW        1234H
```

TESTZ [R]

操作: 将 R 的值赋给 R,用以影响 Z 标志位

周期: 1

影响标志位: Z

举例:

```
TESTZ   R0      ;将寄存器 R0 的值赋给 R0，用于影响 Z 标志位
SZB     FLAGS,Z ;判断 Z 标志位，为 0 间跳
JP      Add1    ;当寄存器 R0 为 0 的时候跳转至地址 Add1
JP      Add2    ;当寄存器 R0 不为 0 的时候跳转至地址 Add1
```

XORIA i

操作: 立即数与 ACC 进行逻辑异或运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
XORIA   0FH     ;执行结果：ACC=05H
```

XORA [R]

操作: 寄存器 R 与 ACC 进行逻辑异或运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;ACC 值赋给 R01,R01=0AH
LDIA    0FH     ;ACC 赋值 0FH
XORA    R01     ;执行结果：ACC=05H
```

XORR [R]

操作: 寄存器 R 与 ACC 进行逻辑异或运算，结果放入 R

周期: 1

影响标志位: Z

举例:

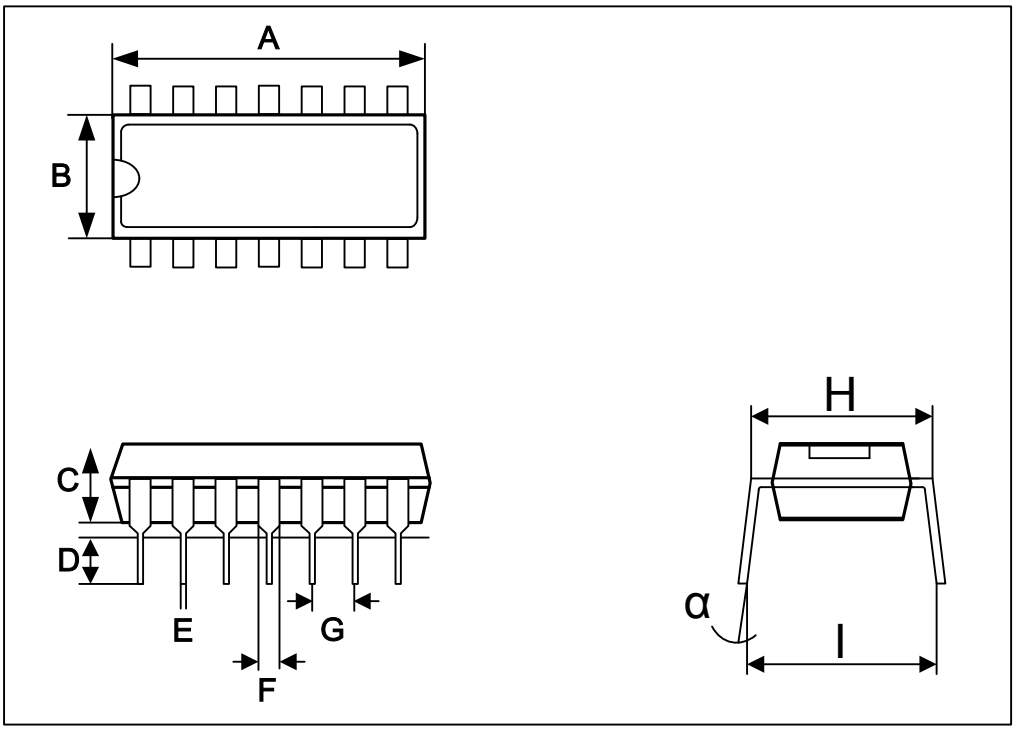
```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;ACC 值赋给 R01,R01=0AH
LDIA    0FH     ;ACC 赋值 0FH
XORR    R01     ;执行结果：R01=05H
```

18. 封装

18.1 CMS69F012 封装参数

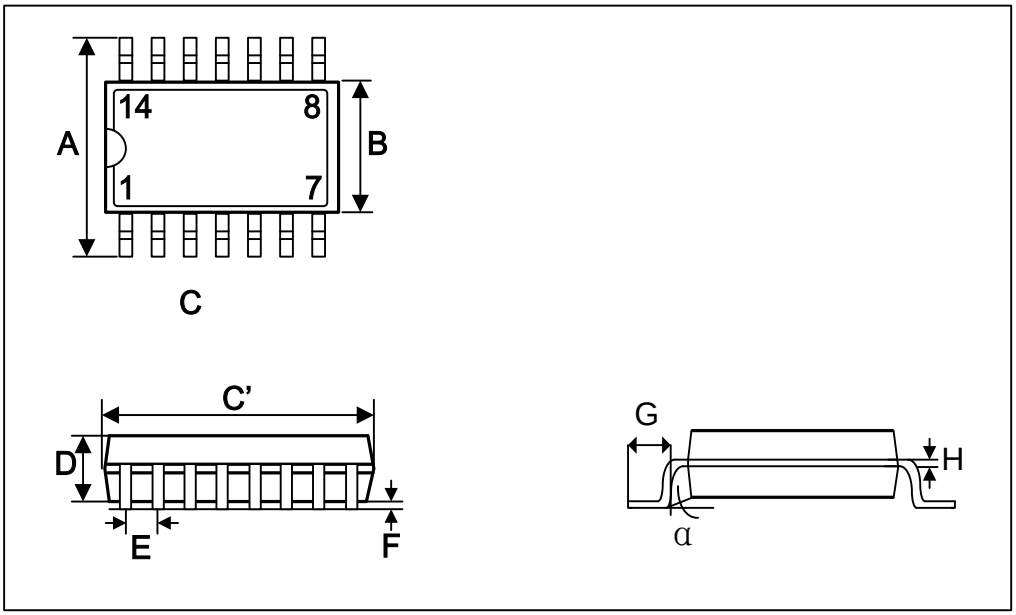
CMS69F012 采用 14-pin DIP 和 14-pin SOP 封装，下面给出其封装参数

18.1.1 Dip 14



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	745	-	775
B	240	-	260
C	125	-	135
D	125	-	145
E	16	-	20
F	50	-	70
G	-	100	-
H	295	-	315
I	335	-	375
α	0°	-	15°

18.1.2 Sop 14

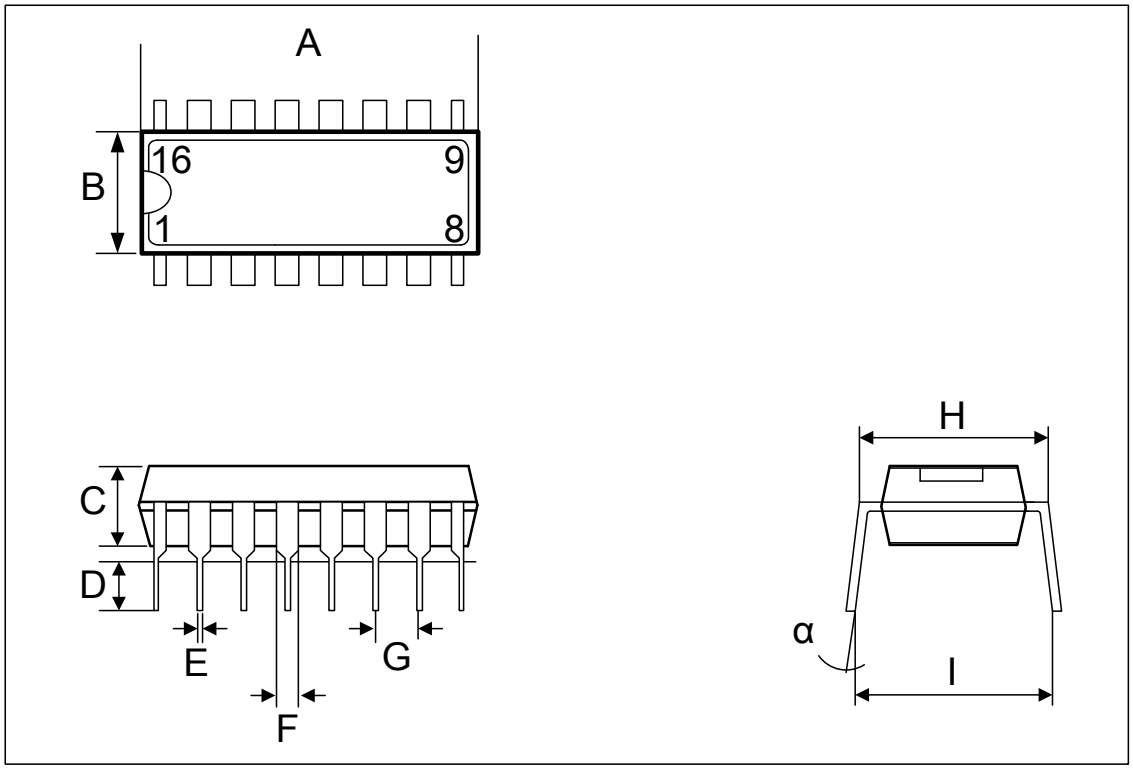


Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	228	-	224
B	149	-	157
C	14	-	20
C'	336		344
D	53	-	69
E	-	50	-
F	4	-	10
G	22	-	28
H	4	-	12
α	0°	-	10°

18.2 CMS69F013 封装参数

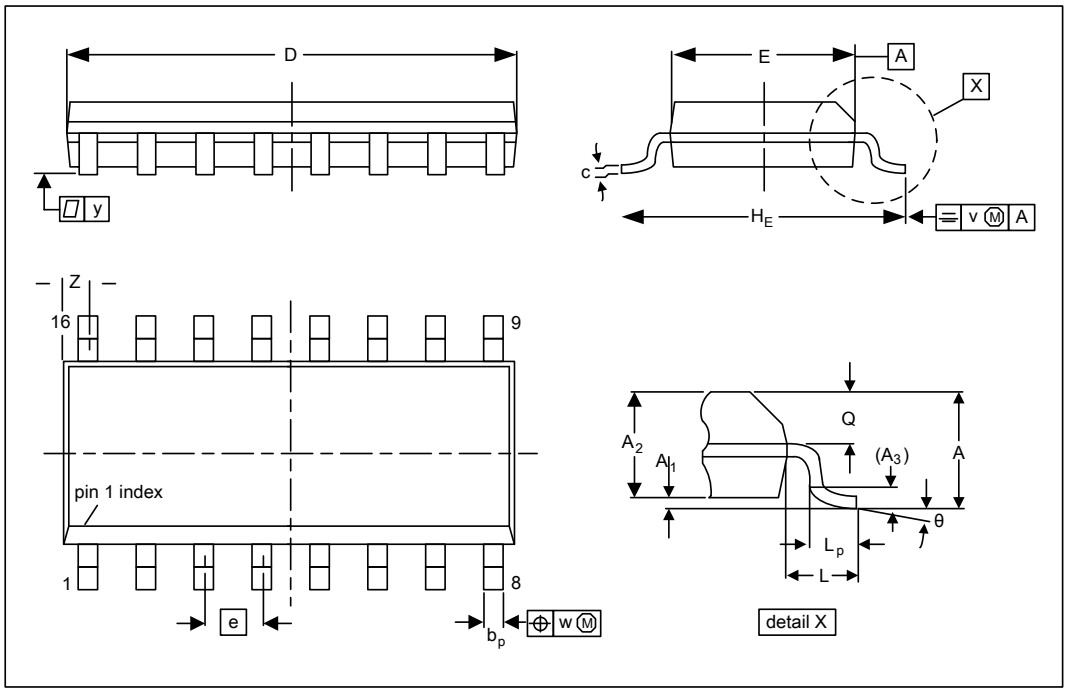
CMS69F013 采用 16-pin DIP 和 16-pin SOP 封装，下面给出其封装参数

18.2.1 Dip 16



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	745	-	775
B	240	-	260
C	125	-	135
D	125	-	145
E	16	-	20
F	50	-	70
G	-	100	-
H	295	-	315
I	335	-	375
α	0°	-	15°

18.2.2 Sop 16



DIMENSIONS (inch dimensions are derived frome the original mm dimensions)

UNIT	A Max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_E	L	Lp	Q	v	w	y	$z^{(1)}$	θ
mm	1.75	0.025 0.10	1.45 1.25	0.25	0.49 0.36	0.25 0.19	10.0 9.8	4.0 3.8	1.27	6.2 5.8	1.05	1.0 0.4	0.7 0.6	0.25	0.25	0.1	0.7 0.3	8°
inches	0.069	0.010 0.004	0.057 0.049	0.01	0.019 0.014	0.0100 0.0075	0.39 0.38	0.16 0.15	0.05	0.244 0.228	0.041	0.039 0.016	0.028 0.020	0.01	0.01	0.004	0.028 0.012	0°

19. 版本修订说明：

版本号	时间	修改内容
V1.0	2014 年 5 月	初步制定
V1.1	2014 年 6 月	修正芯片管脚说明中的错误
V1.2	2014 年 6 月	1，统一多处字体； 2，统一电气参数中的单位符号； 3，修正指令表中 ADDIA 和 HSUBA 中的错误描述； 4，修正掉电复位中关于 LVR 的说明； 5，修正一些错别字； 6，更新一些图片。
V1.3	2014 年 7 月	1，修正电气参数中的数值，使之更加准确； 2，增加 1.6 章节在线编程说明。
V1.4	2014 年 12 月	1. 统一单位代码，统一图片参数