XC800

微控制器系列
架构和指令集

8bit

Microcontrollers

Infineon

英飞凌

Edition 2006-02

Published by Infineon Technologies Asia Pacific Pte Ltd
8 Kallang Sector
Singapore 349282

Attention please!

The information herein is given to describe certain components and shall not be considered as a guarantee of characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XC800

微控制器系列
架构和指令集

Microcontrollers

版本信息： **2006-02** **V0.2**

先前的版本： V0.1

页	内容（对上一版本的主要修正）
	仅进行微小改动；更正句法

期待您的指正

本手册中如有不当、错误及遗漏之处，敬请批评指正，以便我们不断改进用户手册的质量。请将您的建议（以及该手册的相关参考资料）发送至：

mcdocu-chinesecomments.XIY@infineon.com

声明：本中文用户手册是基于英文版本的翻译，如出现与英文用户手册不符之处，请以英文用户手册为主。

目录 页码

1 基本结构

该手册概述了 XC800 微控制器系列的架构和功能特性，并对 XC800 CPU 指令集进行了完整说明。XC800 8 位微控制器不同产品的详细信息参见相关用户手册。

1.1 介绍

英飞凌 XC800 微控制器系列的 CPU 向上兼容标准 8051 内核。标准 8051 CPU 的机器周期由 12 个时钟组成；而 XC800 CPU 的机器周期仅由 2 个时钟组成。

指令集由 45%的单字节指令、41%的双字节指令和 14%的三字节指令组成。每条指令占用 1、2 或 4 个机器周期。访问低速存储器的情况下，因插入等待状态而使得访问时间延长。

XC800 微控制器通过专用 JTAG 接口或标准 UART 接口支持一系列调试特性，包括基本的停止/开始，单步执行，断点支持以及对数据存储器、程序存储器和特殊功能寄存器（SFR）的读/写访问。

XC800 微控制器的主要特性如下：

- 2 个时钟组成的机器周期
- 高达 1 MB 的外部数据存储器；最大 256 B 内部数据存储器
- 高达 1 MB 的程序存储器
- 支持同步或异步程序和数据存储器访问
- 访问低速存储器支持等待状态插入
- 程序存储器下载选择
- 15 个中断源，4 级优先级的中断控制器
- 多达 8 个数据指针
- 省电模式
- 由标准 JTAG 接口或 UART 支持的专用调试模式
- 2 个 16 位定时器（定时器 T0 和定时器 T1）
- 全双工串行端口（UART）

1.2 存储器结构

XC800 微控制器的存储器分区是典型的哈佛结构，即程序存储器和数据存储器空间物理上分开。由占用内部存储器 128 字节的 SFR 对片上外设单元进行访问，通过映射或分页能够增加可寻址 SFR 的个数。

程序存储器空间映射由内部 ROM/Flash，片上 Boot ROM， 片上 XRAM 和/或外部存储器组成。数据存储器空间映射基于典型的标准 8051 架构：内部数据存储器由 128 字节直接寻址 IRAM、128 字节间接寻址 IRAM 和片上“外部”RAM（XRAM）组成。可支持内部数据存储器之外的外部数据存储器。图 1-1 为 XC800 存储器空间的概述及用户模式下典型的存储器映射情况。

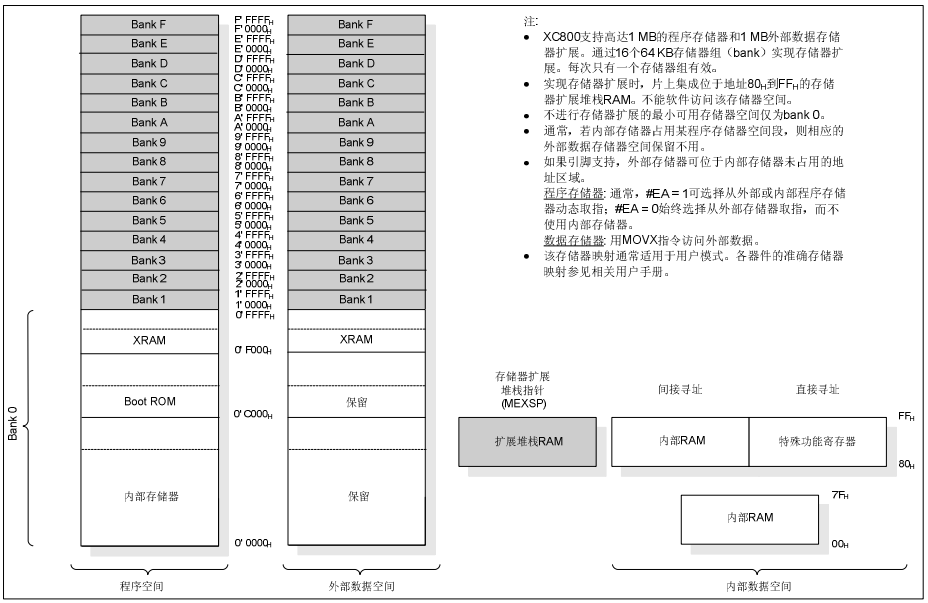


图 1-1 XC800 存储器空间及用户模式下典型的存储器映射

对于存储器扩展的产品，增加地址位于 80_H 至 FF_H 的 128 字节存储器扩展堆栈 RAM。该存储器空间只可由硬件访问，故其对用户而言是透明的。复位后存储器扩展堆栈指针（MEXSP）缺省指向 $7F_H$。在执行调用指令之前指针加 1；执行返回指令之后指针减 1。

1.2.1 存储器扩展

8051 系统可寻址的程序或外部数据存储器标准大小为 64 KB。XC800 内核支持高达 1 MB 的存储器扩展，由存储器管理单元（MMU）和存储器扩展堆栈共同实现该特性。MMU 增加了一组存储器扩展寄存器（MEX1，MEX2 和 MEX3），通过不同的寻址方式控制访问扩展存储器。使用存储器扩展堆栈由硬件“压入”和“弹出”MEX1 的值。

始终从 4 位寄存器位域 CB（当前组）指向的 64 KB 程序存储器组中取指；执行长跳转（LJMP）和调用指令时，CB 由 4 位位域 NB（下一组）更新。CB 和 NB 共同组成 MEX1 寄存器。在执行跳转或调用指令前，编程人员只需将新组编号写入 NB 中。

始终从中断组（IB）寄存器位域指向的 64 KB 程序存储器组中执行中断服务程序。另外，存储器常量数据读取（位于程序存储器）和外部数据访问可能并非针对当前存储器组，由存储器常量组指针（MCB）和 XRAM 组指针（MX）指向目标存储器组。MCB 和 MX 位域在寄存器 MEX2 和 MEX3 中。

1.2.1.1 存储器扩展堆栈

存储器扩展模式下的中断和调用需使用存储器扩展堆栈，该堆栈和标准堆栈同时更新。

使用 SFR 存储器扩展堆栈指针 MEXSP 来寻址存储器扩展堆栈。该读/写寄存器提供的堆栈深度最大 128 字节（位 7 始终为 0）。每次执行调用指令前 SFR 加 1，执行返回指令后 SFR 减 1。复位后 MEXSP 缺省为 $7F_H$，从而首次加 1 后指向堆栈底部。不提供堆栈溢出指示。

1.2.1.2 存储器扩展结果

以下指令可改变所指向的 64KB 存储器组：MOVC、MOVX、LJMP、LCALL、ACALL、RET 和 RETI。

但相对跳转（如 SJMP），间接跳转（JMP @A+DPTR）和 2 KB 范围内的绝对跳转（AJMPs）不会改变当前的存储器组。换言之，这些指令不会取消当前 64 KB 存储器组。

常量传送指令（MOVC）

MOVC 指令访问当前组（CB19 - CB16）或由寄存器 MEX3 和 MEX2 中位域 MCB19 - MCB16 定义的“存储器常量”组中的数据字节。由 MEX2 中的 MCM 位（MEX2.7）选择当前组或存储器常量组。

外部数据传送指令（MOVX）

MOVX 指令访问当前组或由寄存器 MEX3 中位域 MX19 - MX16 定义的“数据存储器”组中的数据。由 MEX3 中的 MXM 位（MEX3.3）选择当前组或数据存储器组。

长跳转指令（LJMP）

当需要从扩展存储器的一个组跳转到另一组时，执行 LJMP 指令前必须正确设置 MEX1 中（MEX1.3 - MEX1.0）下一组 NB19 - NB16 的组地址。执行到 LJMP 指令时，将下一组的值（NB19 - 16）复制到 MEX1 中（MEX1.7 - MEX1.4）的当前组（CB19 - 16），并在下一取指周期开始时将该地址送到地址总线上。

注：跳转指令不改变下一组（NB19 -16）的值。

调用指令（LCALL 和 ACALL）

一旦出现 LCALL 指令，MMU 执行以下操作：

1. 存储器扩展堆栈指针加 1。
2. MEX1 寄存器的值出现在数据总线上。
3. MEXSP 寄存器位域[6:0]的值出现在地址总线上。
4. 存储器扩展堆栈读写信号置位用于写操作。
5. 对存储器扩展堆栈进行写操作。
6. 下一组 NB19 - NB16（MEX1.3 - MEX1.0）的内容被复制到当前组 CB19 - CB16（MEX1.7 - MEX1.4）中。

返回指令（RET 和 RETI）

从子程序返回时，MMU 执行以下操作：

1. MEXSP 寄存器位域[6:0]的值出现在地址总线上。
2. 存储器扩展堆栈读写信号置位用于读操作。
3. 对存储器扩展堆栈进行读操作。
4. 存储器扩展堆栈数据写入 MEX1 寄存器中。
5. 存储器扩展堆栈指针减 1。

1.2.2 程序存储器

XC800 支持高达 1 MB 的同步或异步内部和/或外部程序存储器。如果 XC800 的系列产品支持程序存储器扩展，则用 4 位当前组指针（CB）实现：从 CB 所指向的 64 KB 存储器组中取指，故 XC800 支持的最小代码空间为 64 KB。

若寻址内部程序存储器，$\overline{EA}$（外部访问）引脚必须保持高电平。$\overline{EA}$ 为高电平时，微控制器执行内部程序存储器中的指令，除非地址（程序计数器）已超出内部存储器的寻址范围。在这种情况下，若 XC800 的系列产品支持外部存储器总线，则可动态的从内部和外部程序存储器中取指。若 $\overline{EA}$ 引脚保持低电平，微控制器不执行内部程序存储器中的指令，而执行外部程序存储器中的指令。寻址 Boot ROM，内部 XRAM 和代码空间（如数据 Flash）皆为特例，无论 $\overline{EA}$ 引脚状态如何，始终从内部存储器中取指。

大多数 XC800 系列产品内含 Boot ROM 代码（大小由产品型号决定）。通常，复位后首先执行程序存储器中以 0000_H 为起始地址的 Boot ROM 代码。Boot ROM 会首先交换 Boot ROM 和用户程序的地址空间，从而在进入用户模式之前选择存储器映射方式，可从地址 0000_H 开始执行用户代码。

若程序存储器用 RAM 实现，XC800 内核支持对其进行写操作：MOVC @ (DPTR++), A。通常 XC800 系列产品只支持该指令对内部存储器写入。

1.2.3 数据存储器

数据存储器由内部和外部存储器组成。用 8 位地址寻址内部数据存储器。外部数据存储器和内部 XRAM 数据存储器可使用 8 位或 16 位间接寻址（用‘MOVX’指令），附加最多 4 位地址用于扩展存储器组选择（最大 1 MB）。

1.2.3.1 内部数据存储器 IRAM

内部数据存储器被划分为物理上分开的两个不同区域：256 字节 IRAM 和 128 字节 SFR 区。IRAM 的高 128 字节和 SFR 区共用相同的地址段，通过不同的寻址模式访问这两个不同区域。IRAM 的低 128 字节可通过直接寻址或寄存器间接寻址方式访问；RAM 的高 128 字节只能通过寄存器间接寻址方式访问；SFR 通过直接寻址方式访问。

IRAM 地址段 20_H 到 $2F_H$ 的 16 个字节可位寻址。地址 20_H 上数据字节的位 0 地址为 00_H，地址 $2F_H$ 上数据字节的位 7 地址为 $7F_H$。

复位后堆栈指针缺省指向 07_H，堆栈可位于内部 IRAM 中任意位置。

IRAM 的地址段 30_H 到 $7F_H$ 可用作暂存存储器。

1.2.3.2 片上外部数据存储器 XRAM

片上 XRAM 大小由 XC800 的系列产品的型号决定（容量不固定）。由于可用‘MOVX’和‘MOVC’指令访问片上 XRAM，故其被映射到外部数据存储器和程序存储器空间。用 8 位 MOVX 指令通过寄存器 R0 或 R1 访问片上 XRAM 时，必须初始化 SFR XADDRH 来规定高位地址字节。

如果某型号 XC800 产品仅支持片上 XRAM 或应用仅访问 XRAM，外部接口端口（若可用）可被用作其它功能或通用 I/O。访问片上 XRAM 不会产生外部总线周期。

1.2.3.3 外部数据存储器

XC800 支持最大 1 MB 的同步或异步外部数据存储器。如果 XC800 的系列产品支持程序存储器扩展，则由 MXM 位选择用 4 位当前组指针（CB）或 4 位 XRAM 组指针（MX）加以实现。从 CB 或 MX 指向的 64 KB 存储器组中取数据。部分 XC800 系列产品不支持外部数据存储器。

1.2.4 寄存器

除程序计数器（PC）和四组通用寄存器之外，所有寄存器均位于 SFR 区。

IRAM 的低 32 个地址字节分配给四组通用寄存器（GPR），每组包含 R0 - R7 8 个工作寄存器。任意时刻只能有一组工作，由程序状态字（PSW）中的 RS0（PSW.3）和 RS1（PSW.4）进行组选择。这种快速上下文切换特性在子程序调用或转入执行中断服务程序时非常有用。可通过寄存器寻址方式访问选中寄存器组中的 8 个通用寄存器。寄存器 R0 和 R1 可用作寄存器间接寻址方式中的指针或变址寄存器，用来寻址内部或外部存储器。

特殊功能寄存器（SFR）被映射到内部数据存储器的地址段 80_H 到 FF_H。通过直接寻址可访问 SFR。地址位 0-2 等于 0 所对应的地址上的 SFR 为可位寻址寄存器（如地址 80_H、88_H、90_H、…、$F8_H$）。可位寻址的 SFR 中每一位的位地址对应 SFR 字节地址和该位在 SFR 中的位置。例如，地址 80_H 上 SFR 的位 7 地址为 87_H，SFR 的位地址范围在 80_H 到 FF_H 之间。

内部数据存储区中共有 128 个 SFR，小于所需的寄存器总数，从而采用寄存器扩展机制来增加可寻址 SFR 的个数。寄存器扩展机制包括：

- 映射
- 分页

1.2.4.1 通过映射实现 SFR 扩展

在系统级通过映射进行 SFR 扩展。SFR 区被扩展为两部分：标准（非映射）SFR 区和映射 SFR 区。两个 SFR 区占据相同的地址段 80_H 到 FF_H，从而使可寻址的 SFR 个数增加到 256 个。访问映射 SFR 区时，软件置位 SFR SYSCON0 中的位 RMAP。映射 SFR 区和标准 SFR 区具有相同的寻址能力（直接寻址，位寻址）。访问标准 SFR 区时，软件必须对位 RMAP 清零。硬件不会自动清零/置位 RMAP。

SYSCON0

系统控制寄存器 0 **复位值: XXXX XXX0$_B$**

7	6	5	4	3	2	1	0
-							RMAP
-							rw

符号	位序号	读写类型	功能描述
RMAP	0	rw	**SFR 映射控制** 0 访问标准 SFR 区 1 访问映射 SFR 区

1.2.4.2 通过分页方式扩展 SFR

在模块级通过分页机制进一步扩展（一些片上外设所需）SFR 的个数。这些外设采用内嵌局部 SFR 扩展机制，增加可寻址 SFR 的个数。由模块分页寄存器 MOD_PAGE 中的位域 PAGE 控制分页。在访问目标模块的 SFR 之前，必须先设置位域 PAGE。根据具体要求，每个模块中可能包含的页数不同，每页上 SFR 的个数不同。除了正确设置 RMAP 值来选择标准或映射 SFR 区之外，用户必须确保选择了有效 PAGE 指向所需 SFR。分页机制如**图 1-2** 所示。

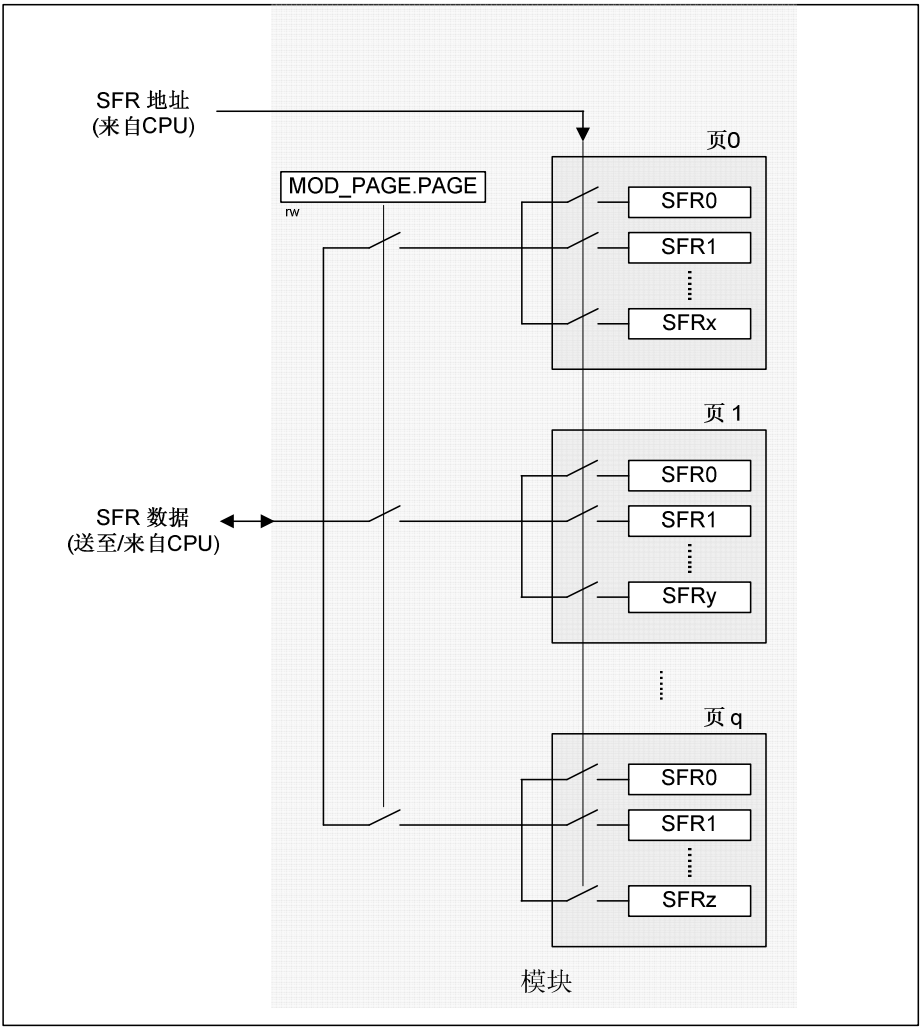


图1-2　　通过分页方式扩展 SFR

如果在访问分页寄存器和模块寄存器之间开始执行某个中断服务程序，且中断需要访问位于另一页上的寄存器，要保存当前页设置，然后设置新页，最后恢复原先页设置。可以用保存域 STx（x = 0 - 3）来保存和恢复当前页的设置，如图 1-3 所示。同时指出应使用哪些保存域和新页值，一个写操作即能够：

- 被新页值覆盖之前将 PAGE 值保存在 STx 中
  （在中断服务程序开始处保存当前页设置，并设置新页编号）；或
- 用 STx 中的值覆盖 PAGE 值，忽略写入 PAGE 的新值
  （在中断服务程序结束时恢复中断发生之前上一次的页设置）

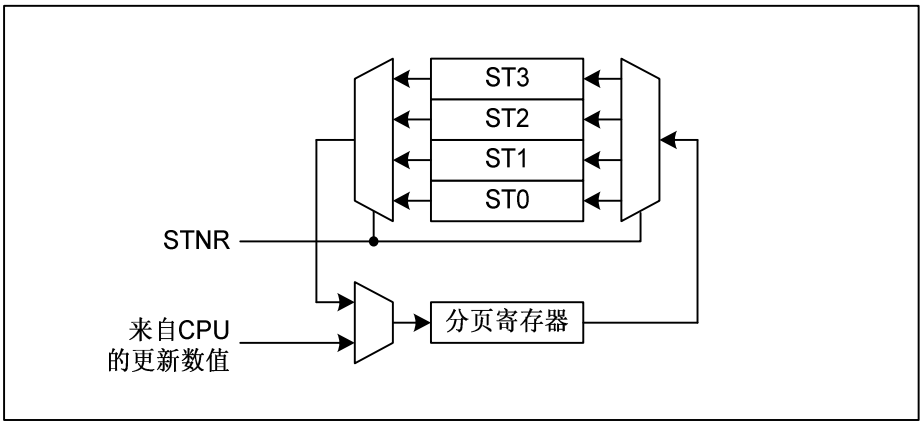


图1-3　　页保存

通过这种分页机制，无需读出并保存上次使用的页信息，若干个中断服务程序（或其它程序）即可修改页设置。仅用写操作使系统更加简单、快速。该机制显著改善了短中断服务程序的性能。

分页寄存器定义如下：

MOD_PAGE

模块 MOD 分页寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
OP		**STNR**		**0**	**PAGE**		
w		w		r	rw		

符号	位序号	读写类型	功能描述
PAGE	[2:0]	rw	**分页位** 写入时，该值表示新页的值 读出时，该值表示当前有效页的值
STNR	[5:4]	w	**保存编号** 该编号指明在哪个保存位域上执行 OP 定义的操作。 若 $OP=10_B$， PAGE 的内容在被新值覆盖之前保存在 STx 中 若 $OP=11_B$， PAGE 的内容被 STx 覆盖。忽略写入 PAGE 的值 00 选择 ST0 01 选择 ST1 10 选择 ST2 11 选择 ST3
OP	[7:6]	w	**操作** 0X 手动保存页模式，STNR 的值被忽略，PAGE 被直接写入 10 有自动页保存的新页设置。当前写入 PAGE 中的内容被保存的同时，上次

符号	位序号	读写类型	功能描述
			写入 PAGE 的内容被保存在 STNR 指定的位域 STx 中 11 自动恢复页。忽略写入 PAGE 的内容，由 STNR 指定的位域 STx 中的内容覆盖 PAGE
0	3	r	**保留** 读操作返回 0；应写入 0

1.3 位保护方案

位保护方案通过 PASSWD 寄存器来阻止软件对选定位（即被保护位）直接写入。当位域 MODE 为 11_B，位域 PASS 中写入 10011_B 会开放所有被保护位的访问权限；位域 PASS 中写入 10101_B 会关闭所有被保护位的访问权限。注意如果未写入“关闭访问权限”口令，权限最多开放 32 个 CCLK 时钟周期。如果在 32 个 CCLK 时钟周期结束前再次写入“开放访问权限”口令，将重新计数 32 个 CCLK 周期。

XC800 系列不同产品被保护的位或位域有所不同。

PASSWD

口令寄存器 **复位值: 07_H**

7	6	5	4	3	2	1	0
PASS					PROTECT_S	MODE	
wh					rh	rw	

符号	位序号	读写类型	功能描述
MODE	[1:0]	rw	**位保护方案控制位** 00 保护方案禁止 11 保护方案使能（缺省值） 其它：保护方案使能 这两位不能直接写入。要在 11_B 和 00_B 之间改变，必须在位域 PASS 中写入 11000_B，只有这样 MODE[1:0]的值才能被写入。
PROTECT_S	2	rh	**位保护信号状态位** 该位指示保护状态。 0 软件能够写入所有被保护位。 1 软件不能写入任何被保护位。
PASS	[7:3]	wh	**口令位** 位保护方案只能识别三个序列。 11000_B MODE 写入使能 10011_B 开放所有被保护位的写权限 10101_B 关闭所有被保护位的写权限

2 CPU 架构

图 2-1 为 XC800 系列微控制器的典型架构，包括主要功能模块和标准单元。不同的 XC800 系列产品对应不同的、用虚线方框表示的功能模块（如外设单元和外部存储器总线）及不同容量的存储器。

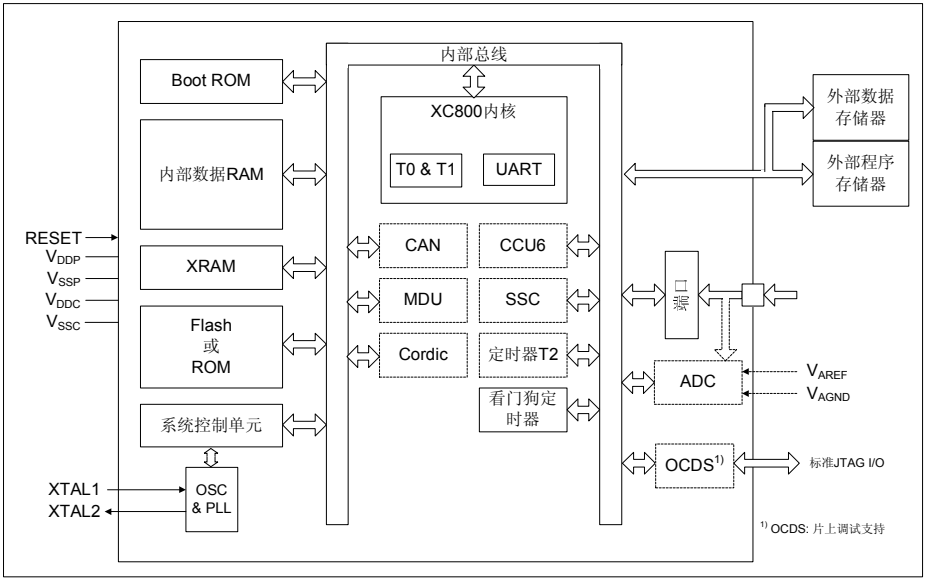


图2-1 XC800 系列微控制器典型架构

CPU 功能模块如图 2-2 所示。CPU 主要由指令译码器、算术单元、程序控制单元、访问控制单元和中断控制器组成。CPU 还提供了省电模式。

由指令译码器对每条指令进行译码，并相应地产生内部信号用来控制 CPU 内部各个模块的功能。这些内部信号影响数据传送并控制 ALU 的操作。

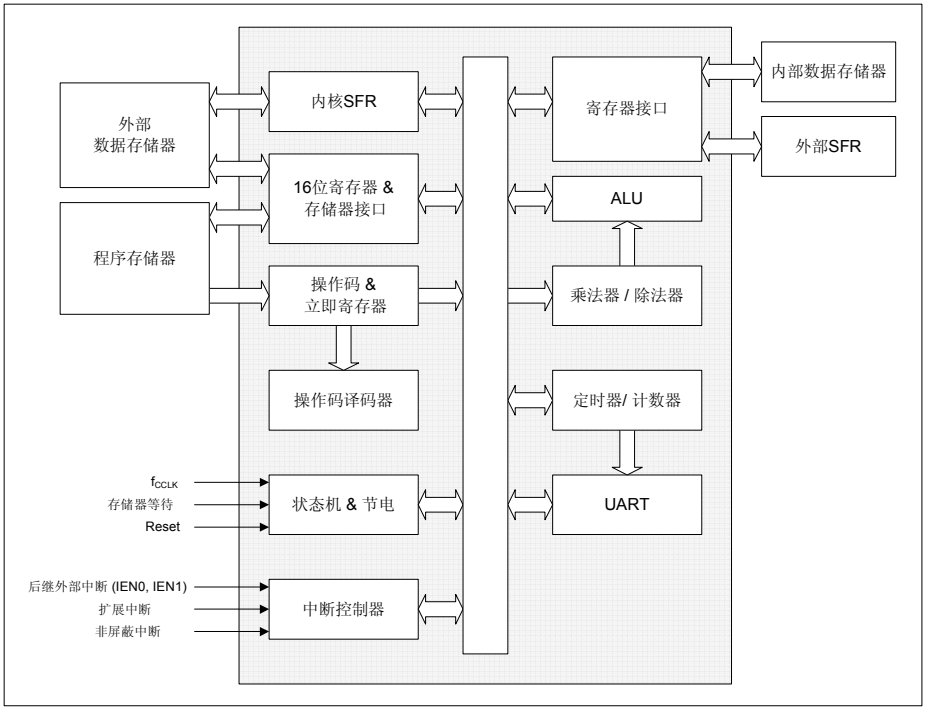


图2-2　　XC800 内核功能方框图

处理器的算术单元具有强大的数据处理能力，由算术/逻辑运算单元（ALU），A 寄存器，B 寄存器和 PSW 寄存器组成。从一个或两个源地址取出的 8 位数据送至 ALU，在译码器控制之下产生 8 位数据结果。ALU 用来完成算术运算和逻辑运算。算术运算包括加、减、乘、除、加 1、减 1、BCD 加法的十进制调整和比较。逻辑运算包括与（AND）、或（OR）、异或、补码、循环移位（右环移、左环移、或 4 位环移）。ALU 还包括一个布尔处理机，可执行置位、清零、补码、等于 1 跳转、等于 0 跳转、等于 1 跳转并清 0、送入/取自进位位的位操作。对任何可寻址的位（或其补码）和进位标志进行逻辑与或者逻辑或操作，结果存入进位标志位。

程序控制单元控制程序存储器中指令的执行顺序。16 位程序计数器（PC）中保存下一条将要执行指令的地址。条件跳转逻辑允许处理器响应内部和外部事件，改变程序执行顺序。

访问控制单元用于选择片上存储器资源。中断控制器处理由外设产生的中断请求。

2.1 CPU 寄存器描述

CPU 寄存器占用内部数据存储器的地址段 80_H-FF_H。

2.1.1 堆栈指针（SP）

堆栈指针寄存器保存堆栈指针 SP。在执行 LCALL 和 ACALL 指令时，用 SP 保存 PC 值；在执行 RET 和 RETI 指令时，从 SP 中恢复 PC 值。也可以通过 PUSH 和 POP 指令在堆栈中保存或从堆栈中取回数据内容。压栈前 SP 自动加 1，出栈后 SP 自动减 1，从而保证了堆栈指针始终指向写入堆栈的最后一个字节，即栈顶。复位后，SP 总是初始化指向 07_H，所以第一个压入堆栈的数据放在寄存器组 0 之上的地址单元 08_H 中。SP 的读写操作可由软件控制。编程人员必须确保内部数据存储器中的堆栈位置和大小不会影响到其它应用数据。

2.1.2 数据指针（DPTR）

数据指针（DPTR）保存在寄存器 DPL（数据指针低位字节）和 DPH（数据指针高位字节）中构成 16 位地址，用于外部数据存储器访问（MOVX A,@DPTR 和 MOVX @DPTR,A），程序字节传送（MOVC A,@A+DPTR），及间接程序跳转（JMP @A+DPTR）。

数据指针支持两个真正的 16 位操作：加载立即数(MOV DPTR,# data）和加 1（INC DPTR）操作。

CPU 最多支持 8 个数据指针，这有助于用高级语言编程将数据保存到大容量外部数据存储器中。通过 SFR EO 选择数据指针（见**章节 2.1.6**）。不同的 XC800 系列产品所包含的数据指针个数不同。

2.1.3 累加器（ACC）

大多数 ALU 操作中，该寄存器存放其中一个操作数。累加寄存器用 ACC 表示。和累加器相关的指令，其助记符简化为符号“A”。

2.1.4 B 寄存器

乘除法操作时 B 寄存器用来存放第二个操作数，在其它指令中用作暂存寄存器。

2.1.5 程序状态字

程序状态字（PSW）寄存器中存放反映 CPU 当前状态的状态位。

2.1.5.1 程序状态字寄存器

PSW

程序状态字寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
CY	**AC**	**F0**	**RS1**	**RS0**	**OV**	**F1**	**P**
rwh	rwh	rw	rw	rw	rwh	rw	rh

符号	位序号	读写类型	功能描述
P	0	rh	**奇偶校验标志** 每条指令执行后由硬件置位/清零，指示累加器中"1"的个数为奇数/偶数，如偶校验
F1	1	rw	**通用标志**
OV	2	rwh	**溢出标志** 用于算术运算指令
RS1 **RS0**	4:3	rw	**寄存器组（Register Bank）选择位** 由这两位进行组（Bank）选择。 00 选择 Bank 0，数据地址 00_H - 07_H 01 选择 Bank 1，数据地址 08_H - $0F_H$ 10 选择 Bank 2，数据地址 10_H - 17_H 11 选择 Bank 3，数据地址 18_H - $1F_H$
F0	5	rw	**通用标志**
AC	6	rwh	**辅助进位标志** 用于执行 BCD 操作的指令
CY	7	rwh	**进位标志** 用于算术运算指令

2.1.6 扩展操作（EO）

EO 寄存器具有双重功能。一个是为具有多数据指针的产品选择数据指针；另一个功能是选择操作码 $A5_H$ 执行的指令，即选择执行‘TRAP’或‘MOVC @(DPTR++), A’指令。后一个指令支持写入程序存储器。

2.1.6.1 扩展操作寄存器

EO

扩展操作寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
0			**TRAP_EN**	**0**	**DPSEL**		
r			rw	r	rw		

符号	位序号	读写类型	功能描述
DPSEL	[2:0]	rw	**数据指针选择** 000 选择 DPTR0 001 选择 DPTR1（若可用） 010 选择 DPTR2（若可用） 011 选择 DPTR3（若可用） 100 选择 DPTR4（若可用） 101 选择 DPTR5（若可用） 110 选择 DPTR6（若可用） 111 选择 DPTR7（若可用）
TRAP_EN	4	rw	**TRAP 使能** 0 选择 MOVC @(DPTR++),A 指令 1 选择软件 TRAP 指令
0	3, [7:5]	r	**保留** 读操作返回 0；应写入 0。

2.1.7 存储器扩展

这些寄存器支持存储器扩展特性，某些 XC800 系列产品不提供这些寄存器。

2.1.7.1 存储器扩展寄存器

MEX1

存储器扩展寄存器 1 **复位值: 00_H**

7	6	5	4	3	2	1	0
CB[19:16]				NB[19:16]			
rh				rw			

符号	位序号	读写类型	功能描述
NB[19:16]	[3:0]	rw	**下一组编号**
CB[19:16]	[7:4]	rh	**当前组编号**

MEX2

存储器扩展寄存器 2 **复位值: 00_H**

7	6	5	4	3	2	1	0
MCM	MCB[18:16]			IB[19:16]			
rw	rw			rw			

符号	位序号	读写类型	功能描述
IB[19:16]	[3:0]	rw	**中断组编号**
MCB[18:16]	[6:4]	rw	**存储器常量组编号（连同 MEX3.7）**
MCM	7	rw	**存储器常量组选择** 0 MOVC 访问当前组中的数据 1 MOVC 访问存储器常量组中的数据

MEX3

存储器扩展寄存器 3 **复位值: 00_H**

7	6	5	4	3	2	1	0
MCB19	**0**		**MX19**	**MXM**	**MX[18:16]**		
rw	r		rw	rw	rw		

符号	位序号	读写类型	功能描述
MX[19:16]	4, [2:0]	rw	**XRAM 组编号**
MXM	3	rw	**XRAM 组选择** 0 MOVX 访问当前组中的数据 1 MOVX 访问存储器 XRAM 组中的数据
MCB19	7	rw	**存储器常量组编号的 MSB**
0	[6:5]	r	**保留** 读操作返回 0；应写入 0。

MEXSP

存储器扩展堆栈指针寄存器 **复位值: $7F_H$**

7	6	5	4	3	2	1	0
0	**MXSP**						
r	rwh						

符号	位序号	读写类型	功能描述
MXSP	[6:0]	rwh	**存储器扩展堆栈指针** 堆栈深度最大 128 字节。执行调用指令前 SP 加 1；执行返回指令后 SP 减 1。
0	7	r	**保留** 读操作返回 0；应写入 0。

2.1.8 功率控制（PCON）

XC800 CPU 有两种省电模式：空闲模式和掉电模式。空闲模式下 CPU 的时钟被禁止，但其它外设可能继续工作（可能低速）。掉电模式下，整个 CPU 的时钟全部停止。

2.1.8.1 功率控制寄存器

PCON

功率控制寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
SMOD	**0**			**GF1**	**GF0**	**0**	**IDLE**
rw	r			rw	rw	r	rw

符号	位序号	读写类型	功能描述
IDLE	0	rw	**空闲模式使能** 0 不进入空闲模式 1 进入空闲模式
GF0	2	rw	**通用标志位 0**
GF1	3	rw	**通用标志位 1**
SMOD	7	rw	**波特率双倍频使能** 0 模式 2 中串口波特率不双倍频 1 模式 2 中串口波特率双倍频
0	1, [6:4]	r	**保留** 读操作返回 0；应写入 0。

2.1.9 UART

UART 有两个 SFR，控制寄存器 SCON 和数据寄存器 SBUF。SCON 是串口控制和状态寄存器，该寄存器中存放模式选择位、发送和接收的第 9 个数据位（TB8 和 RB8），以及串口中断位（TI 和 RI）。

SBUF 是发送和接收缓冲寄存器。写入 SBUF 将加载发送寄存器并启动数据发送；从 SBUF 读数据实际是从物理上分开的接收寄存器中读取。两条数据通路互相独立并支持全双工操作。

2.1.9.1 UART 寄存器

SBUF

串行数据寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
VAL							
rwh							

符号	位序号	读写类型	功能描述
VAL	[7:0]	rwh	**串行接口缓存寄存器**

SCON

串行通道控制寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**
rw	rw	rw	rw	rw	rwh	rwh	rwh

符号	位序号	读写类型	功能描述
RI	0	rwh	**接收中断标志** 模式 0 中，第 8 位接收结束时由硬件置位；模式 1、2 和 3 中，接收到停止位的中间时刻由硬件置位。必须由软件清零。
TI	1	rwh	**发送中断标志** 模式 0 中，第 8 位发送结束时由硬件置位；模式 1、2 和 3 中，开始发送停止位时由硬件置位。必须由软件清零。
RB8	2	rwh	**串口接收的第 9 位** 模式 2 和 3 中接收到的第 9 位数据；模式 1 中若 SM2 = 0 该位是接收到的停止位；模式 0 不使用 RB8。
TB8	3	rw	**串口发送的第 9 位** 模式 2 和 3 中发送的第 9 位数据
REN	4	rw	**串口接收使能** 0 禁止串口接收 1 使能串口接收
SM2	5	rw	**模式 2 和 3 中串口多处理器通信使能位** 模式 2 或 3 中，若 SM2 置 1，如果接收到的第 9 位数据（RB8）为 0，RI 不会被激活； 模式 1 中，若 SM2 被置 1，如果没有接收到有效的停止位（RB8），RI 不会被激活； 模式 0 中，SM2 应被设置为 0。

符号	位序号	读写类型	功能描述
SM0 **SM1**	7:6	rw	**串口工作模式选择** 00 模式 0：8 位移位寄存器，固定波特率（f_{PCLK}/2） 01 模式 1：8 位 UART，可变波特率 10 模式 2：9 位 UART，固定波特率（f_{PCLK}/32 或 f_{PCLK}/64） 11 模式 3：9 位 UART，可变波特率

2.1.10 定时器/计数器

XC800 内核中有两个 16 位定时器：定时器 T0 和定时器 T1。

SFR TCON 控制定时器运行并产生中断；SFR TMOD 设置定时器的工作模式。定时器/计数器的值保存在两对 8 位寄存器 TL0、TH0 和 TL1、TH1 中（复位值 = 0000_H）。

2.1.10.1 定时器/计数器寄存器

TCON

定时器控制寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
TF1	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**
rwh	rw	rwh	rw	rw	rw	rw	rw

符号	位序号	读写类型	功能描述
TR0	4	rw	**定时器 T0 运行控制** 0 定时器暂停 1 定时器运行
TF0	5	rwh	**定时器 T0 溢出标志** 定时器 T0 溢出时由硬件置位。当处理器执行中断服务程序时由硬件清零。
TR1	6	rw	**定时器 T1 运行控制** 0 定时器暂停 1 定时器运行 如果定时器 T0 工作在模式 3，该位还影响 TH0。
TF1	7	rwh	**定时器 T1 溢出标志** 定时器 T1 溢出时由硬件置位。当处理器执行中断服务程序时由硬件清零。 *注：如果定时器 T0 工作在模式 3，TF1 由 TH0 置位。*

TMOD

定时器模式寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
GATE1	CT1	T1M		GATE0	CT0	T0M	
rw	rw	rw		rw	rw	rw	

符号	位序号	读写类型	功能描述
T0M, T1M	[1:0] [5:4]	rw	**模式选择位** 00 13 位定时器：THx 作为 8 位定时器/计数器，TLx 作为 5 位预分频器 01 16 位定时器：THx 和 TLx 级联 10 8 位自动重载定时器：THx 保存重载值，每次定时器溢出时，将重载值载入到 TLx 中。 11 定时器 T0 被分成两部分。8 位定时器 TL0 由标准定时器 T0 的控制位控制；8 位定时器 TH0 由标准定时器 T1 的控制位控制。 定时器 T1：TL1 和 TH1 的内容保持不变（定时器 T1 停止工作）。
CT0, CT1	2, 6	rw	**定时器 Tx 的计数器功能选择** 0 定时器模式（输入来自内部系统时钟） 1 计数器模式（输入来自 Tx 引脚）
GATE0, GATE1	3, 7	rw	**定时器 Tx 门控标志** 0 仅当 TCON.TRx = 1 时定时器 Tx 才运行 （软件控制）。 1 仅当引脚 NINTx = 0（硬件控制）且 TCON.TRx = 1 时定时器 Tx 才运行。

2.1.11 中断寄存器

每个中断节点可被分别使能或禁止，通过置位或清零可位寻址中断使能寄存器 IEN0 和 IEN1 中的相应位来实现。寄存器 IEN0 中还包含全局使能/禁止位（EA），清零操作可一次有效地禁止所有中断。

非屏蔽中断（NMI）节点始终使能。

复位后，IEN0 和 IEN1 中的使能位全都被清零，暗指所有中断被禁止。

IEN0

中断使能寄存器 0 **复位值: 00_H**

7	6	5	4	3	2	1	0
EA	**0**	**ET2**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**
rw	r	rw	rw	rw	rw	rw	rw

符号	位序号	读写类型	功能描述
EX0	0	rw	**外部中断 0 使能** 0 禁止外部中断 0 1 使能外部中断 0
ET0	1	rw	**定时器 T0 溢出中断使能** 0 禁止定时器 T0 溢出中断 1 使能定时器 T0 溢出中断
EX1	2	rw	**外部中断 1 使能** 0 禁止外部中断 1 1 使能外部中断 1
ET1	3	rw	**定时器 T1 溢出中断使能** 0 禁止定时器 T1 溢出中断 1 使能定时器 T1 溢出中断
ES	4	rw	**串行端口中断使能** 0 禁止串行端口中断 1 使能串行端口中断
ET2	5	rw	**定时器 T2 中断使能** 0 禁止定时器 T2 中断

符号	位序号	读写类型	功能描述
			1 使能定时器 T2 中断
EA	7	rw	**使能/禁止所有中断** 0 不响应中断请求（NMI 除外） 1 通过置位或清零使能位，分别使能或禁止每个中断源
0	6	r	**保留** 读操作返回 0；应写入 0

IEN1 寄存器存放的中断使能位用来使能或禁止相应中断。根据 XC800 系列产品中可用的外设单元来分配这些使能位。

IEN1

中断使能寄存器 1 **复位值: 00_H**

7	6	5	4	3	2	1	0
EI13	**EI12**	**EI11**	**EI10**	**EI9**	**EI8**	**EI7**	**EI6**
rw	rw	rw	rw	rw	rw	rw	rw

符号	位序号	读写类型	功能描述
EIx **(x = 13:6)**	[7:0]	rw	**扩展中断使能** 0 禁止 XINTRx 1 使能 XINTRx

每个中断源的优先级（共有四级优先级）可通过相应的 IP、IPH 或 IP1、IPH1 寄存器分别设定。IP 和 IP1 可位寻址；IPH 和 IPH1 不可位寻址。

IP(H)

中断优先级(高位字节)寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
0		**PT2(H)**	**PS(H)**	**PT1(H)**	**PX1(H)**	**PT0(H)**	**PX0(H)**
r		rw	rw	rw	rw	rw	rw

符号	位序号	读写类型	功能描述
PX0, PX0H	0	rw	**外部中断 0 优先级**
PT0, PT0H	1	rw	**定时器 T0 溢出中断优先级**
PX1, PX1H	2	rw	**外部中断 1 优先级**
PT1, PT1H	3	rw	**定时器 T1 溢出中断优先级**
PS, PSH	4	rw	**串口中断优先级**
PT2, PT2H	5	rw	**中断节点 XINTR5 中断优先级（定时器 T2）**
0	[7:6]	r	**保留** 读操作返回 0；应写入 0

IP(H)1

中断优先级 1(高位字节)寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
PI13(H)	**PI12(H)**	**PI11(H)**	**PI10(H)**	**PI9(H)**	**PI8(H)**	**PI7(H)**	**PI6(H))**
rw	rw	rw	rw	rw	rw	rw	rw

符号	位序号	读写类型	功能描述
PI6, PI6H	0	rw	**中断节点 XINTR6 中断优先级**
PI7, PI7H	1	rw	**中断节点 XINTR7 中断优先级**
PI8, PI8H	2	rw	**中断节点 XINTR8 中断优先级**
PI9, PI9H	3	rw	**中断节点 XINTR9 中断优先级**
PI10, PI10H	4	rw	**中断节点 XINTR10 中断优先级**
PI11, PI11H	5	rw	**中断节点 XINTR11 中断优先级**
PI12, PI12H	6	rw	**中断节点 XINTR12 中断优先级**
PI13, PI13H	7	rw	**中断节点 XINTR13 中断优先级**

TCON 中的四位用于控制和外部中断标志。

TCON

定时器控制寄存器 **复位值: 00_H**

7	6	5	4	3	2	1	0
TF1	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**
rwh	rw	rwh	rw	rwh	rw	rwh	rw

符号	位序号	读写类型	功能描述
IT0	0	rw	**外部中断 0 电平/边沿触发控制** 0 选择低电平触发外部中断 0 1 选择下降沿触发外部中断 0
IE0	1	rwh	**外部中断 0 请求标志** 当检测到外部中断 0 事件时由硬件置位。 处理器响应中断转入中断服务程序时，该标志由硬件清零。该标志还能由软件清零。
IT1	2	rw	**外部中断 1 电平/边沿触发控制** 0 选择低电平触发外部中断 1 1 选择下降沿触发外部中断 1
IE1	3	rwh	**外部中断 1 请求标志** 当检测到外部中断 1 事件时由硬件置位。 处理器响应中断转入中断服务程序时，该标志由硬件清零。还可由软件清零该标志。

2.2 片上调试支持系统

XC800 微控制器具有片上调试支持（OCDS）单元，提供了基于 XC800 系统进行软件开发与调试所需的基本功能。器件进入 OCDS 模式之后通常使能调试功能。

调试概念基于 OCDS 硬件和保存在 Boot ROM 中的专用软件（即监控器程序）之间的紧密作用。使用标准接口（如 JTAG 或 UART）和外部主机（如调试器）通信。

调试系统控制、访问和接口如图 2-3 所示。

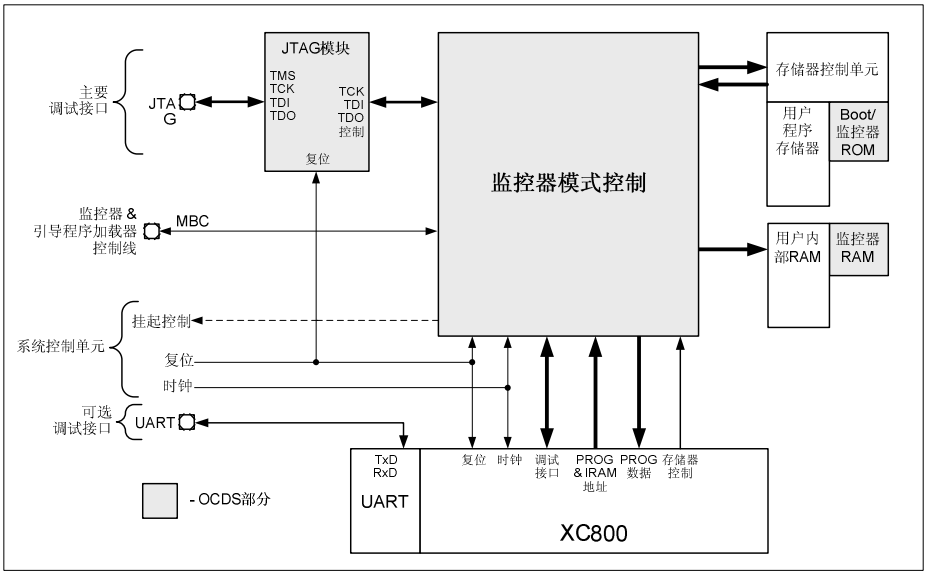


图2-3 XC800 OCDS 功能框图

- 监控器模式控制（MMC）模块是 OCDS 系统的核心，连接 OCDS 的控制信号，支持总体调试功能。
- MMC 主要通过调试接口和 XC800 内核通信，并接收复位和时钟信号。
- MMC 根据来自 CPU 的内存地址和控制信号，可对专用存储器，即监控器 ROM（保存程序）和监控器 RAM（保存工作数据和监控器堆栈）进行恰当访问。
- 可用两种接口访问 OCDS 系统：
  - 为主要调试接口：专用于测试和调试，在应用中通常不使用
  - UART 为可选调试接口：优点在于所需引脚较少
- 调试和引导程序加载使用专用引脚进行外部配置和控制

片上调试概念基于调试事件和相应调试动作的产生和检测。

- 调试事件
  - 硬件断点
  - 软件断点
  - 外部断点
- 调试事件动作（任一种皆可）
  - 调用监控器程序
  - 激活 MBC 引脚
- 其它调试特性：
  - 单步执行
  - 返回到用户程序

2.3 CPU 中断系统

本节概述 XC800 中断系统。

2.3.1 中断源和中断向量地址

每个中断源对应一个中断向量地址。从该中断向量转入相应中断源请求的中断服务程序。通过使能位，每个中断节点的中断服务程序可单独使能或禁止。XC800 的中断源分配见表 2-1。不同的 XC800 系列产品为片上外设分配的扩展中断有所不同。

表 2-1 中断向量地址

中断输入	中断向量地址	中断请求源
NMI	0073_H	非屏蔽中断
XINTR0	0003_H	外部中断 0
XINTR1	$000B_H$	定时器 T0
XINTR2	0013_H	外部中断 1
XINTR3	$001B_H$	定时器 T1
XINTR4	0023_H	UART
XINTR5	$002B_H$	扩展中断 5（定时器 T2）
XINTR6	0033_H	扩展中断 6
XINTR7	$003B_H$	扩展中断 7
XINTR8	0043_H	扩展中断 8
XINTR9	$004B_H$	扩展中断 9
XINTR10	0053_H	扩展中断 10
XINTR11	$005B_H$	扩展中断 11
XINTR12	0063_H	扩展中断 12
XINTR13	$006B_H$	扩展中断 13

2.3.2 中断处理

在每个机器周期的 P2 时采样中断请求信号，在下一个机器周期内对采样到的中断请求进行查询。如果某中断节点请求在前一周期的 P2 已被置位，查询周期将发现该请求，中断系统将产生一个 LCALL 指令调用中断节点的服务程序。由硬件生成的 LCALL 指令在下列任意一种情况下都会推迟执行：

1. 同级或更高级的中断已在处理中。
2. 当前周期（查询周期）不是正在执行指令的最后一个周期。
3. 正在执行的指令是 RETI 或是对寄存器 IEN0/IEN1 或 IP/IPH 或 IP1/IP1H 的写操作。

上述任意一种情况都会推迟执行 LCALL 进入中断服务程序。条件 2 保证了正在执行的指令在进入任何中断服务程序之前可执行完毕。条件 3 保证了如果正在执行的指令是 RETI 或是对寄存器 IEN0/IEN1 或 IP/IPH 或/IP1/IP1H 的写操作时，进入中断服务程序之前必须至少再执行一条指令，该延迟保证了中断状态的改变可被 CPU 监测到。

中断查询在每个机器周期重复执行，查询到的值为前一机器周期 P2 的采样值。注意，如中断标志有效，但由于上述的条件之一而未被响应；或稍后在上述条件撤销后中断标志已不再有效，相应中断将不再被响应。换言之，曾经有效的中断请求标志若未能被 CPU 及时响应，将不被记忆。每个查询周期仅查询挂起的中断请求。

处理器通过执行由硬件生成的 LCALL 指令调用相应的中断服务程序来响应中断请求。在有些情况下，硬件清除中断标志；另一些情况下，必须由用户软件清除中断标志。硬件生成的 LCALL 会将程序计数器的内容压栈（但不保存 PSW 的内容），并将中断向量地址装入 PC。

中断服务程序执行到 RETI 指令时，程序返回继续执行调用中断之后的下一条指令。RETI 指令通知处理器中断服务程序已执行完毕，然后从堆栈弹出两个字节重新装入 PC，继续执行被中断的程序。需要注意的是，RETI 指令非常重要，它通知处理器中断服务程序已执行完毕。简单的 RET 指令也可以返回到被中断的程序，但这样会使中断控制系统认为中断服务程序仍在执行。这种情况下，同级或低级中断就无法响应。

2.3.3 中断响应时间

对于某中断节点上的（不同中断源产生的）中断事件，其相应的中断请求信号将在每个机器周期的 P2 被采样，在下一个机器周期之前该值不会被电路查询。如果中断请求有效并且响应中断的条件成立，下一条将执行硬件生成的调用指令，进入中断服务程序。调用指令本身占用两个机器周期，因此，从中断请求有效到开始执行中断服务程序中的第一条指令至少需要三个完整的机器周期，如**图 2-4** 所示。

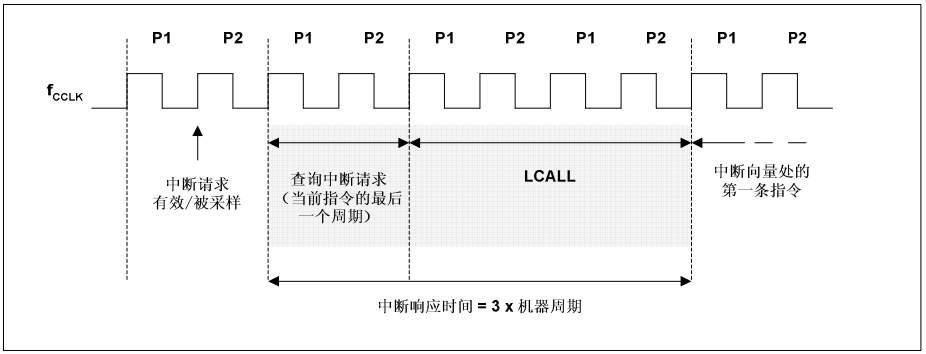


图 2-4　　最短中断响应时间

如果下列任意一种情况出现，中断请求则需要更长的响应时间。

1. 如果一个同级或更高优先级的中断正在处理中，则附加的等待时间取决于其它中断服务程序所需时间；
2. 如果正在执行的指令还没有执行到它的最后一个周期，则附加的等待时间不会超过三个机器周期，因为最长的指令（MUL 和 DIV 指令）仅四个机器周期，见**图 2-5**。
3. 如果正在执行的指令是 RETI 指令或访问寄存器 IEN0、IEN1、或 IP(H)、IP1(H)的写指令，则附加的等待时间不会超过五个机器周期（最多再用一个机器周期完成当前指令，如果指令是 MUL 或 DIV，再加四个机器周期完成下条指令），见**图 2-6**。

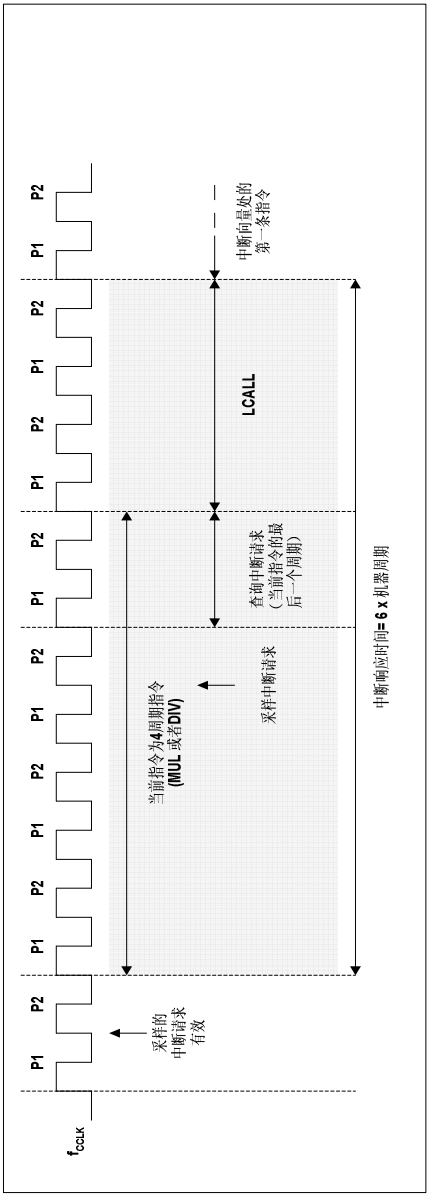


图 2-5　条件 2 下的中断响应时间

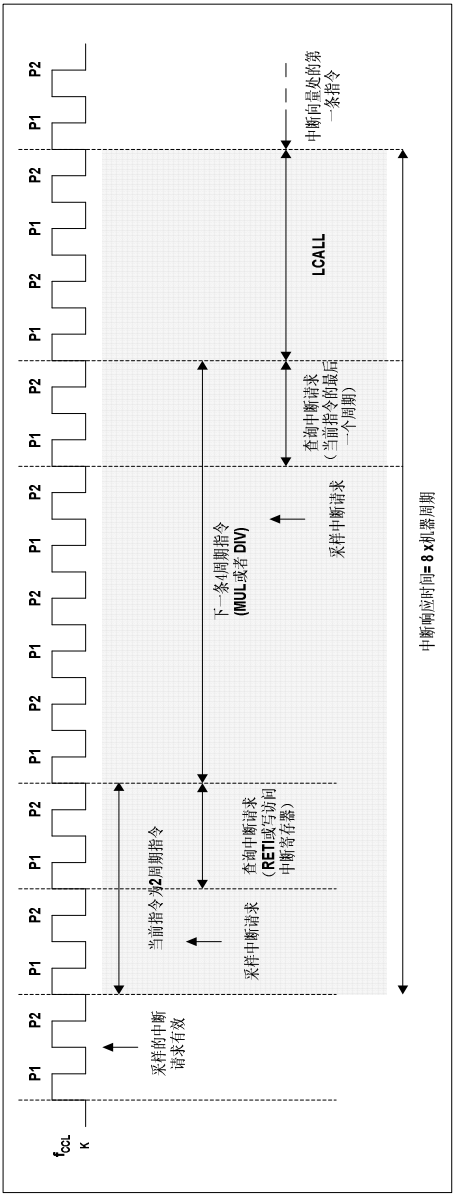


图 2-6　条件 3 下的中断响应时间

因此，在单中断系统中，如果不考虑等待状态，中断响应时间大于等于三个机器周期、小于九个机器周期。当考虑等待状态时，中断响应时间将会延长，这取决于中断响应期间（图 2-5 和图 2-6 的阴影部分）执行的用户指令占用的时间（还包括硬件生成的 LCALL）。

2.3.4 中断节点优先级

低优先级中断服务程序可被更高优先级中断，但不能被同级或低级中断。最高级中断不被其它任何中断源中断。

如果两个或更多不同优先级的中断源同时请求中断时，首先响应最高优先级的中断请求。

每对中断优先级寄存器中的对应位相组合，共同确定该中断源的优先级（4 级优先级供选择），见表 2-2。

表 2-2 中断优先级选择

IPH.x / IPH1.x	IP.x / IP1.x	优先级
0	0	优先级 0（最低）
0	1	优先级 1
1	0	优先级 2
1	1	优先级 3（最高）

注：由于 NMI 的优先级始终高于其它任何中断的优先级。

如果几个同优先级的中断源同时请求中断时，由内部查询顺序将决定首先响应哪个中断请求。因此，每级优先级内部又有由查询顺序决定的次级优先级结构，见表 2-3。不同的 XC800 系列产品对应片上外设产生的扩展中断不同。

表 2-3 同级内的优先级结构

中断源	优先级
非屏蔽中断（NMI）	（最高）
外部中断 0	1
定时器 T0 中断	2
外部中断 1	3

中断源	优先级
定时器 T1 中断	4
UART 中断	5
扩展中断 5（定时器 T2）	6
扩展中断 6	7
扩展中断 7	8
扩展中断 8	9
扩展中断 9	10
扩展中断 10	11
扩展中断 11	12
扩展中断 12	13
扩展中断 13	14

3 CPU 时序

下面各节描述 CPU 指令时序以及外部存储器访问时序。

3.1 指令时序

CPU 机器周期由两个输入时钟周期组成，分别用拍 1（P1）和拍 2（P2）表示，对应 CPU 的两个不同状态。执行指令时，CPU 的状态由机器周期和状态编号（P1/P2）共同表示，例如：C2P1 表示第二个机器周期中的第一个时钟周期。可在机器周期的任一拍、或两拍访问存储器；只在 P2 结束时对 SFR 进行写操作。执行一条指令需要一个、两个或者四个机器周期。通常在一条指令的最后一个机器周期的 P2 结束时进行寄存器内容的更新和下一个操作码的读取。

XC800 内核通过插入等待状态的方式支持对低速（内部）存储器的访问。每个等待状态持续一个机器周期，例如：所访问存储器需要插入一个等待状态时，在读取操作码/操作数的每个字节之后，访问时间增加一个机器周期。

图 3-1 给出取指/执行时序的内部状态和节拍。每条指令从 C1P1 开始执行，双字节指令的第二个字节的读取从 C1P1 开始。

图 3-1（a）给出两个单字节单周期（1× 机器周期）指令的时序图。第一个时序图表示下一个操作码（C1P2）从无等待状态的存储器中读取，指令在一个周期内完成；第二个时序图表示操作码从低速存储器中读取，插入一个等待状态，相同的指令需要两个机器周期完成（指令时间延长）。

图 3-1（b）给出两个双字节单周期（1× 机器周期）指令的时序图。第一个时序图表示第二个字节（C1P1）和下一个操作码（C1P2）从无等待状态的存储器中读取，指令在一个周期内完成；第二个时序图表示每次访问低速存储器插入一个等待状态（共插入两个等待状态），相同的指令需要三个机器周期完成（指令时间延长）。

图 3-1（c）给出两个单字节双周期（2× 机器周期）指令的时序图。第一个时序图表示下一个操作码（C2P2）从无等待状态的存储器中读取，指令在两个周期内完成；第二个时序图表示操作码从低速存储器中读取，插入一个等待状态，相同的指令需要三个机器周期完成（指令时间延长）。

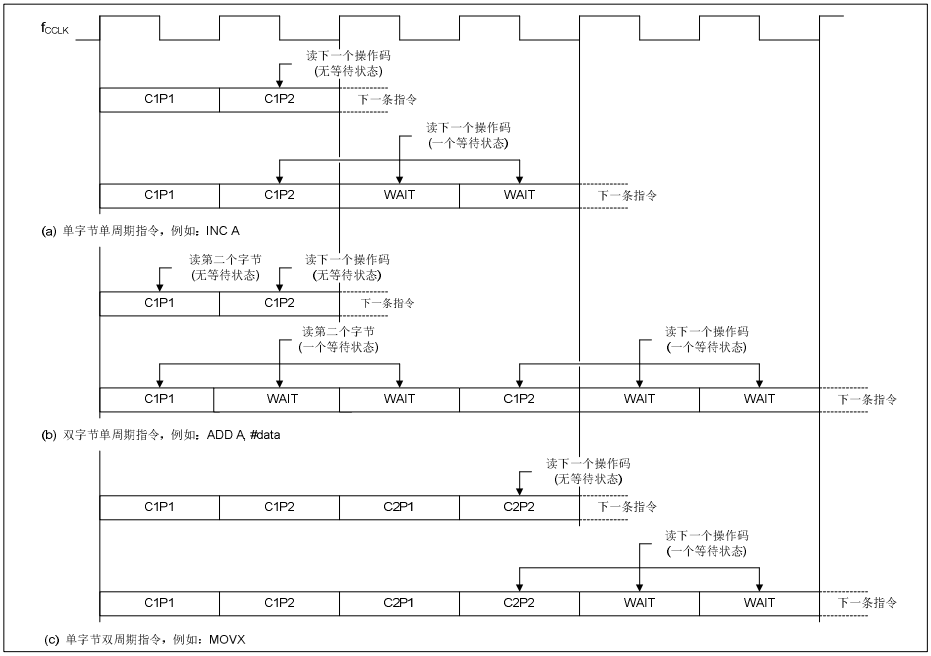


图3-1　　CPU 指令时序

执行每条指令所需的时间包括：

- 译码/执行操作码
- 取操作数（指令长度超过一个字节时）
- 取下条指令的首字节（操作码）（CPU 流水结构）

注：XC800 CPU 在执行当前指令的同时预取下条指令的操作码。

即使每次读取操作数和操作码都要插入一个等待状态，XC800 CPU 执行指令的速度仍然是标准 8051 处理器的 2 倍（如双字节单周期指令）到 6 倍（如单字节四周期指令）。

3.2 访问外部存储器

XC800 支持两种外部存储器访问，即外部程序存储器和外部数据存储器访问。访问外部程序存储器时 $\overline{\text{PSEN}}$ 用作读选通信号；访问外部数据存储器时 $\overline{\text{RD}}$ 或 $\overline{\text{WR}}$ 用作读选通或写选通信号；支持外部存储器访问的 XC800 系列产品，地址线（Ax）和数据线（D[7:0]）还可用作端口的其它功能选择。

3.2.1 访问外部程序存储器

通常以下两个条件成立时，才访问外部程序存储器：

- 只要 $\overline{\text{EA}}$ 有效（低电平），或
- 只要 $\overline{\text{EA}}$ 无效（高电平）的同时，程序计数器（PC）中的地址超出内部程序存储器地址范围

用 16 位地址总线寻址外部程序存储器，扩展存储器的地址总线最多 20 位（高四位用作存储器组的选择）。这些地址引脚使用端口的其他功能选择，当 CPU 访问外部程序存储器时，不能将这些引脚用作其它端口功能。

图 3-2 所示为访问外部程序存储器的时序图。

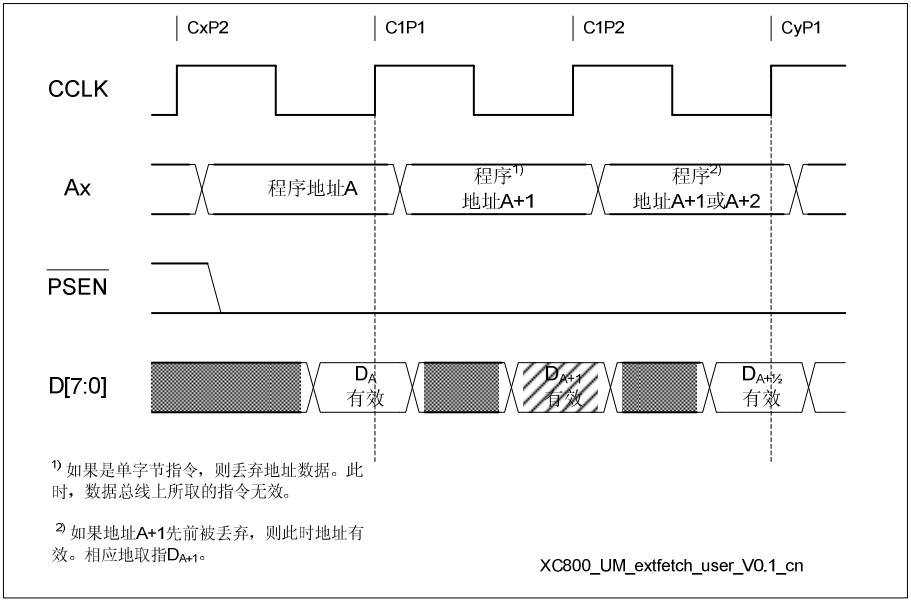


图3-2 访问外部程序存储器

3.2.2 访问外部数据存储器

通常只有当代码空间某地址段未被内部程序存储器占用时，外部数据存储器相应地址才可被访问。

访问外部数据存储器时，地址总线的位 17 到位 20（若可用）用来选择存储器组。在每个存储器组内，可通过 16 位寻址模式（MOVX @DPTR）或 8 位寻址模式（MOVX @Ri）进行访问。如果采用 8 位寻址模式，可用任意输出端口引脚输出高位地址。或者访问外部存储器期间，高字节地址引脚相应端口 SFR 的内容可能用来保存引脚上的高字节地址，因此通过定义高地址字节，这些引脚可用作外部存储器的当前有效的存储器组（由 MEX1.CBx 或 MEX3.MXx 选择当前有效的存储器组）的分页操作。

读周期过程中，在读选通信号 $\overline{RD}$ 刚好被拉高之前才读入数据。

图 3-3 所示为外部数据存储器读周期的时序图，该图假设只访问外部数据存储器。

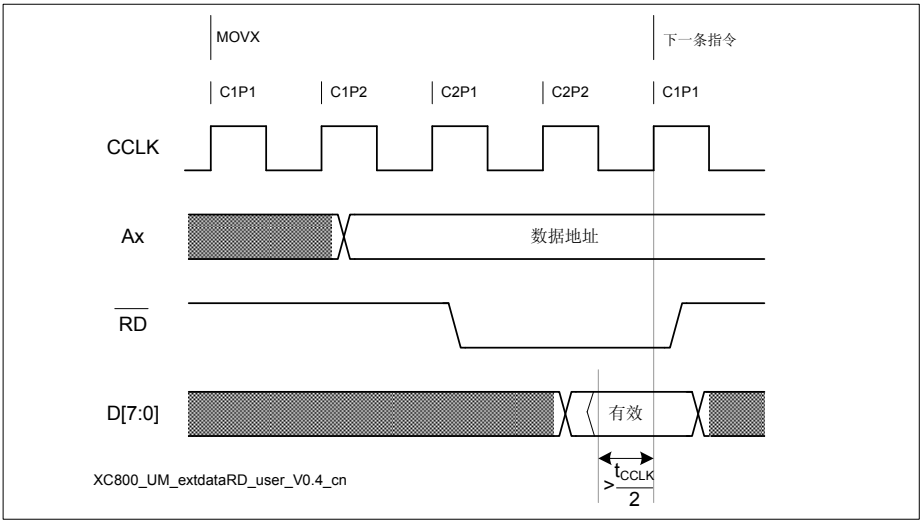


图3-3 外部数据存储器读周期

写周期过程中，在写选通信号 $\overline{WR}$ 有效之前，准备写入存储器的数据字节已送到引脚上，在 $\overline{WR}$ 被拉高之后始终保持。

图 3-4 为外部数据存储器写周期的时序图，该时序假设访问外部程序存储器和外部数据存储器交叉进行。

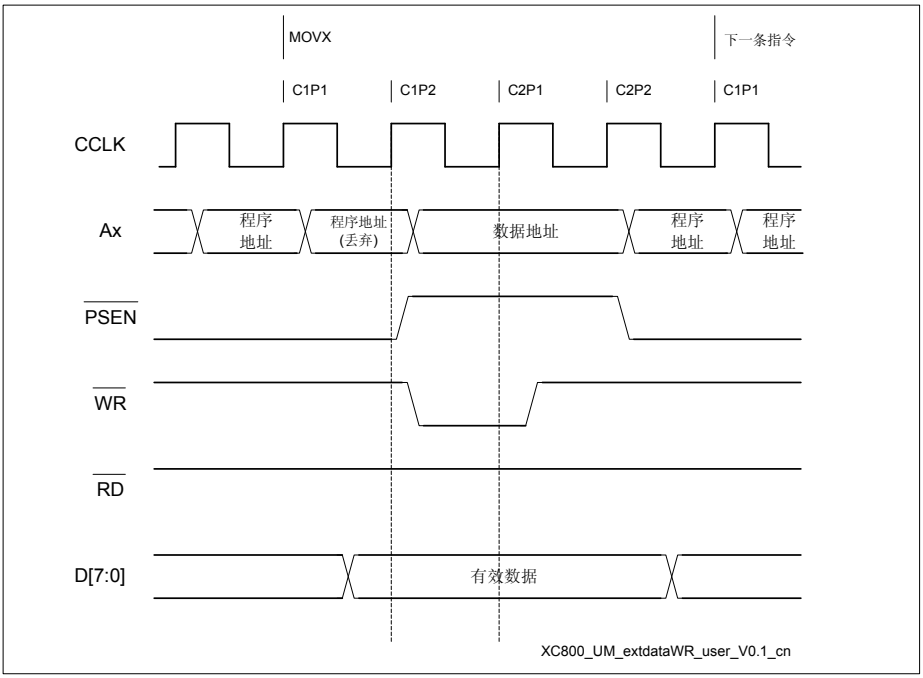


图3-4　　外部数据存储器写周期

4 指令集

XC800 8 位微控制器系列的指令集包含 111 条标准 8051 指令，以及两条附加指令：‘MOVC @ (DPTR ++) , A’ 和‘TRAP’ 。这两条指令共用一个操作码，由特殊功能寄存器（SFR）EO 进行选择。113 条指令中有 51 条单字节指令、46 条双字节指令和 16 条三字节指令。

指令助记符由操作码和操作数组成。操作码代表指令的功能；紧跟其后是“目的，源”操作数域，操作数指定数据类型以及寻址模式。

4.1 寻址模式

XC800 有五种常用寻址模式：

- 寄存器
- 直接寻址
- 立即寻址
- 寄存器间接寻址
- 基址变址间接寻址

还包括对可位寻址空间的位寻址

表 4-1 对每种寻址模式对应的存储器空间加以总结。

表 4-1 寻址模式和相应存储器空间

寻址模式	相应存储器空间
寄存器寻址	选定寄存器组的 R0-R7, ACC, B, CY(位), DPTR
直接寻址	内部 RAM 的低 128 字节，特殊功能寄存器
立即寻址	程序存储器
寄存器间接寻址	内部 RAM（@R1, @R0, SP） 外部数据 RAM（@R1, @R0, @DPTR）
基址变址寻址	程序存储器（@A+DPTR, @A+PC）
位寻址	可位寻址 SFR，内部数据 RAM 低位地址段 128 个可位寻址区域

4.1.1 寄存器寻址

寄存器寻址访问选定寄存器组的 8 个工作寄存器（R0 - R7）。指令操作码中的低有效位指示选择哪个寄存器。某些指令只对如 ACC（A）、B、DPTR 或位 CY（布尔累加器）等特定寄存器进行操作。

4.1.2 直接寻址

只能通过直接寻址访问 SFR。内部 RAM 的低 128 字节也可直接寻址。直接寻址时操作数由一个 8 位地址域指定。

4.1.3 立即寻址

立即寻址模式允许指令中包含立即数。这些指令是双字节或多字节指令。

4.1.4 寄存器间接寻址

寄存器间接寻址使用（选定寄存器组的）R0 或 R1 的内容作为 256 字节存储器区（内部 RAM （256 字节）、或外部数据存储器低 256 字节）的地址指针。注意该寻址模式不能访问 SFR；内部 RAM 的高 128 字节只能采用间接寻址。用 16 位数据指针可遍访外部数据存储器当前有效组的全部 64KB 地址单元。

4.1.5 基址变址寻址

基址变址寻址方式的操作数地址是基址寄存器（DPTR 或 PC）和变址寄存器 ACC 相加的结果。该寻址方式便于访问查找表。

4.1.6 位寻址

可位寻址区域支持直接位寻址：可位寻址 SFR，以及内部数据 RAM 低位地址段 128 个可位寻址区域。

4.2 指令集简介

指令集可按功能划分为六类：

- 算术运算
- 逻辑操作
- 数据传送
- 控制跳转（分支）
- 布尔操作
- 其它操作

4.2.1 算术运算指令

XC800 微控制器有四种基本算术运算。

- 加法：ADD、ADDC、INC、DA
- 减法：SUBB、DEC
- 乘法：MUL
- 除法：DIV

ALU 仅直接支持 8 位无符号整数的算术运算，但溢出标志给无符号和有符号二进制整数的加减操作提供了方便。还可对直接对 BCD 码进行算术运算。

4.2.2 逻辑操作指令

XC800 微控制器对位和字节操作数的基本逻辑操作包括：ANL、ORL、SRL、CLR、SETB、CPL、RL、RLC、RR、RRC、SWAP。

4.2.3 数据传送指令

数据传送操作分为三类：

- 通用
- 累加器专用
- 目标地址

除 POP 或 MOV 指令直接影响 PSW 标志之外，其它数据传送指令不影响 PSW 标志。

4.2.4 控制跳转指令

所有控制跳转操作（一些操作基于特定条件），使程序跳转到程序存储器的非连续地址处继续执行。有三类控制跳转操作：

- 无条件跳转
- 条件跳转
- 子程序/中断调用和返回

无条件跳转指令控制将当前 PC 指向目标地址。这类指令有：AJMP、LJMP、SJMP 和 JMP @A+DPTR。

条件跳转指令在满足某个条件时执行跳转操作。跳转的目的地址范围在下一指令的起始地址为中心的 256 字节内（-128 到+127 之间）。这类指令有：JZ、JNZ、JC、JNC、JB、JNB、JBC、CJNE、DJNZ。

只有两类子程序调用：ACALL 和 LCALL。由硬件控制中断服务程序调用。返回指令为 RET 和 RETI。RETI 指令从中断返回，将中断优先级恢复为当前优先级。

4.2.5 布尔操作指令

内部 RAM 和 SFR 中的位可寻址寄存器可使用布尔操作指令。位操作指令包括：

- 置位
- 清零
- 取反
- 若指定位置位则跳转
- 若指定位清零则跳转
- 若指定位置位则跳转、并对该位清零
- 取自/送入进位位

可寻址位或其补码可和进位标志逻辑“与”或逻辑“或”，结果保存在进位位中。

4.2.6 其它指令

这类指令包括：

- NOP：无操作
- TRAP：软件断点命令

4.3 指令

XC800 的指令可精简为 55 种基本操作。这些操作将在以下章节作详细说明。

4.3.1 受影响的标志位

某些指令会影响一个或多个 PSW 标志位，通常如表 4-2 所示。

表 4-2 PSW 标志位修改（CY, OV, AC）

指令	标志			指令	标志		
	CY	OV	AC		CY	OV	AC
ADD	X	X	X	SETB C	1		
ADDC	X	X	X	CLR C	0		
SUBB	X	X	X	CPL C	X		
MUL	0	X		ANL C, bit	X		
DIV	0	X		ANL C, /bit	X		
DA	X			ORL C, bit	X		
RRC	X			ORL C, /bit	X		
RLC	X			MOV C, bit	X		
CJNE	X						

上表中，“0”表示标志位始终清零，“1”表示标志位始终置位；“X”表示标志位的状态由操作结果决定。空白表示该标志不受指令影响。

上表中只讨论了进位标志、辅助进位标志和溢出标志。奇偶校验位始终由累加器的实际值计算得到。

- 如果操作导致最高位有进位或有借位时 CY 置 1；否则 CY 清零。
- 如果操作结果的低四位（相加）有进位或（相减）低位向高位借位时 AC 置 1；否则 AC 清零。
- 如果操作结果最高位有进位但并非来自相邻低位的进位；或相邻低位有进位但最高位不进位，则 OV 置 1；否则 OV 清零。OV 用作二进制补码运算，因为运算结果无法用 8 位表示时 OV 被置位。
- 如果累加器 8 位的模 2 和为 1（奇校验）则 P 置 1；否则 P 清零（偶校验）。对 PSW 寄存器写入时，P 位保持不变，因为它始终反映 A 的奇偶性。

若被寻址寄存器为 PSW，则直接修改该寄存器内容的指令可改变其它的状态标志位；位操作也可改变状态标志位。

4.3.2 指令表

表 4-3 列出 XC800 支持的所有指令。指令长度为 1、2 或 3 字节，在“字节数”列中给出。执行每条指令需 1、2 或 4 个机器周期（无等待状态）。每个机器周期由两个 CCLK 时钟周期组成。

表 4-3 指令表

助记符	说明	十六进制代码	字节数	周期数
		算术操作		
ADD A, Rn	寄存器加到 A	28-2F	1	1
ADD A, direct	直接字节加到 A	25	2	1
ADD A, @Ri	间接存储器加到 A	26-27	1	1
ADD A, #data	立即数加到 A	24	2	1
ADDC A, Rn	寄存器和进位位加到 A	38-3F	1	1
ADDC A, direct	直接字节和进位位加到 A	35	2	1
ADDC A, @Ri	间接存储器和进位位加到 A	36-37	1	1
ADDC A, #data	立即数和进位位加到 A	34	2	1
SUBB A, Rn	A 减去寄存器和借位位	98-9F	1	1
SUBB A, direct	A 减去直接字节和借位位	95	2	1
SUBB A, @Ri	A 减去间接存储器和借位位	96-97	1	1
SUBB A, #data	A 减去立即数和借位位	94	2	1
INC A	A 加 1	04	1	1
INC Rn	寄存器加 1	08-0F	1	1
INC direct	直接字节加 1	05	2	1
INC @Ri	间接存储器加 1	06-07	1	1
DEC A	A 减 1	14	1	1
DEC Rn	寄存器减 1	18-1F	1	1
DEC direct	直接字节减 1	15	2	1

助记符	说明	十六进制代码	字节数	周期数
DEC @Ri	间接存储器减 1	16-17	1	1
INC DPTR	数据指针加 1	A3	1	2
MUL AB	A 乘以 B	A4	1	4
DIV AB	A 除以 B	84	1	4
DA A	A 的十进制加法调整	D4	1	1
逻辑操作				
ANL A, Rn	寄存器“与”到 A	58-5F	1	1
ANL A, direct	直接字节“与”到 A	55	2	1
ANL A, @Ri	间接存储器“与”到 A	56-57	1	1
ANL A, #data	立即数“与”到 A	54	2	1
ANL direct, A	A “与”到直接字节	52	2	1
ANL direct, #data	立即数 “与”到直接字节	53	3	2
ORL A, Rn	寄存器“或”到 A	48-4F	1	1
ORL A, direct	直接字节“或”到 A	45	2	1
ORL A, @Ri	间接存储器“或”到 A	46-47	1	1
ORL A, #data	立即数“或”到 A	44	2	1
ORL direct, A	A “或”到直接字节	42	2	1
ORL direct, #data	立即数 “或”到直接字节	43	3	2
XRL A, Rn	寄存器“异或”到 A	68-6F	1	1
XRL A, direct	直接字节“异或”到 A	65	2	1
XRL A, @Ri	间接存储器“异或”到 A	66-67	1	1
XRL A, #data	立即数“异或”到 A	64	2	1
XRL direct, A	A “异或”到直接字节	62	2	1
XRL direct,	立即数 “异或”到直接字节	63	3	1

助记符	说明	十六进制代码	字节数	周期数
#data				
CLR A	A 清零	E4	1	1
CPL A	A 取反	F4	1	1
SWAP A	A 半字节交换	C4	1	1
RL A	A 左环移	23	1	1
RLC A	A 连同进位左环移	33	1	1
RR A	A 右环移	03	1	1
RRC A	A 连同进位右环移	13	1	1
数据传送				
MOV A, Rn	寄存器送到 A	E8-EF	1	1
MOV A, direct	直接字节送到 A	E5	2	1
MOV A, @Ri	间接存储器送到 A	E6-E7	1	1
MOV A, #data	立即数送到 A	74	2	1
MOV Rn, A	A 送到寄存器	F8-FF	1	1
MOV Rn, direct	直接字节送到寄存器	A8-AF	2	2
MOV Rn, #data	立即数送到寄存器	78-7F	2	1
MOV direct, A	A 送到直接字节	F5	2	1
MOV direct, Rn	寄存器送到直接字节	88-8F	2	2
MOV direct, direct	直接字节送到直接字节	85	3	2
MOV direct, @Ri	间接存储器送到直接字节	86-87	2	2
MOV direct, #data	立即数送到直接字节	75	3	2
MOV @Ri, A	A 送到间接存储器	F6-F7	1	1
MOV @Ri, direct	直接字节送到间接存储器	A6-A7	2	2

助记符	说明	十六进制代码	字节数	周期数
MOV @Ri, #data	立即数送到间接存储器	76-77	2	1
MOV DPTR, #data 16	立即数送到数据指针	90	3	2
MOVC A, @A+DPTR	由（（A）+（DPTR））寻址的程序存储器字节送到 A	93	1	2
MOVC A, @A+PC	由（（A）+（PC））寻址的程序存储器字节送到 A	83	1	2
MOVX A, @Ri	外部数据（8 位地址）送到 A	E2-E3	1	2
MOVX A, @DPTR	外部数据（16 位地址）送到 A	E0	1	2
MOVX @Ri, A	A 送到外部数据（8 位地址）	F2-F3	1	2
MOVX @DPTR, A	A 送到外部数据（16 位地址）	F0	1	2
PUSH direct	直接字节压栈	C0	2	2
POP direct	直接字节出栈	D0	2	2
XCH A, Rn	交换 A 和寄存器	C8-CF	1	1
XCH A, direct	交换 A 和直接字节	C5	2	1
XCH A, @Ri	交换 A 和间接存储器	C6-C7	1	1
XCHD A, @Ri	交换 A 和间接存储器低 4 位	D6-D7	1	1
布尔变量操作				
CLR C	清零进位	C3	1	1
CLR bit	清零直接位	C2	2	1
SETB C	置位进位	D3	1	1
SETB bit	置位直接位	D2	2	1
CPL C	进位取反	B3	1	1

助记符	说明	十六进制代码	字节数	周期数
CPL bit	直接位取反	B2	2	1
ANL C, bit	直接位“与”到进位	82	2	2
ANL C, /bit	直接位的反“与”到进位	B0	2	2
ORL C, bit	直接位“或”到进位	72	2	2
ORL C, /bit	直接位的反“或”到进位	A0	2	2
MOV C, bit	直接位送进位	A2	2	1
MOV bit, C	进位送直接位	92	2	2
控制程序跳转				
ACALL addr11	绝对子程序调用（在当前 2KB 范围内）	11->F1	2	2
LCALL addr16	长子程序调用	12	3	2
RET	子程序调用返回	22	1	2
RETI	中断调用返回	32	1	2
AJMP addr 11	绝对跳转（在当前 2KB 范围内）	01->E1	2	2
LJMP addr 16	无条件长跳转	02	3	2
SJMP rel	相对短跳转	80	2	2
JC rel	进位位 = 1 则相对跳转	40	2	2
JNC rel	进位位 = 0 则相对跳转	50	2	2
JB bit, rel	直接位 = 1 则相对跳转	20	3	2
JNB bit, rel	直接位 = 0 则相对跳转	30	3	2
JBC bit, rel	直接位 = 1 则相对跳转并清零直接位	10	3	2
JMP @A+DPTR	相对 DPTR 间接跳转	73	1	2
JZ rel	A = 0 则相对跳转	60	2	2

助记符	说明	十六进制代码	字节数	周期数
JNZ rel	A = 1 则相对跳转	70	2	2
CJNE A, direct, rel	直接字节和 A 比较，不等则相对跳转	B5	3	2
CJNE A, #data, rel	立即数和 A 比较，不等则相对跳转	B4	3	2
CJNE Rn, #data, rel	立即数和寄存器比较，不等则相对跳转	B8-BF	3	2
CJNE @Ri, #data, rel	立即数和间接存储器比较，不等则相对跳转	B6-B7	3	2
DJNZ Rn, rel	寄存器减 1，不为零则相对跳转	D8-DF	2	2
DJNZ direct, rel	直接字节减 1，不为零则相对跳转	D5	3	2
其它指令				
NOP	空操作	00	1	1
附加指令（由 EO[7:4]控制选择）				
MOVC @(DPTR++), A	XC800 专用软件下载（到程序存储器）指令：复制 A 后 DPTR 加 1	A5	1	2
TRAP	XC800 专用软件断点指令	A5	1	1

数据存储器寻址模式注释：

- Rn：工作寄存器 R0-R7
- direct：128 个内部 RAM 地址，特殊功能寄存器（SFR）
- @Ri：由寄存器 R0 或 R1 间接寻址内部或外部 RAM
- #data：8 位立即数
- #data16：16 位立即数
- bit：内部数据 RAM 低位地址段的 128 个可寻址位，SFR 中任意可寻址位
- A：累加器

程序存储器寻址模式注释：

- addr16：LCALL 和 LJMP 的目的地址，位于程序存储器 64K 字节有效存储器组内的任意地址。
- addr11：ACALL 和 AJMP 的目的地址，和下一指令的首字节位于同一 2KB 的地址范围内。
- rel：SJMP 及所有条件跳转指令均包含一个 8 位偏移量。地址跳转范围在相对下一指令首字节的+127/-128 字节之间。

4.3.3 指令定义

按基本操作对指令分组，根据操作助记符的字母顺序对指令进行说明。

绝对子程序调用

ACALL addr11

表 4-4 ACALL

描述：	ACALL 无条件的调用指定地址处的子程序。指令先将 PC 值加 2，指向下条指令的地址，然后将 16 位当前地址压栈（低位字节在先），堆栈指针加 2。将当前 PC 值（已加 2）的高 5 位、操作码位 7-5 和指令的第二字节级联构成目的地址。因此被调用的子程序必定和 ACALL 下一指令的首字节地址在同一个 2KB 范围内。本指令不影响标志位。
例子：	SP 初始化为 07_H。标签“SUBRTN”对应程序存储器的地址单元 0345_H。执行 0123_H 地址上的指令 ACALL SUBTRN 之后，SP 的值变为 09_H。内部 RAM 地址单元 08_H 和 09_H 分别保存 25_H 和 01_H。相应地 PC 当前值为 0345_H。
指令：	
ACALL addr11	操作： (PC) ← (PC) + 2 (SP) ← (SP) +1 ((SP)) ← (PC7-0) (SP) ← (SP) +1 ((SP)) ← (PC15-8) (PC10-0) ← 页地址 字节数：2 周期数：2 编码：

a10	a9	a8	1	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0

加法指令

ADD A, <src-byte>

表 4-5 ADD

描述：	ADD 将指定的字节变量加到累加器 A，结果保存在 A 中。若位 7 或位 3 有进位，分别置位进位标志和辅助进位标志；无进位则清零。无符号整数相加时，进位标志指示溢出情况。 若位 6 和位 7 有一位产生进位而另一位不产生进位，OV 被置位；否则 OV 清零。OV = 1 表示两正数相加，和变成负数，或两负数相加和变成正数。 支持四种源操作数寻址方式：寄存器、直接、寄存器间接寻址，或立即寻址。
例子：	累加器中存放 $0C3_H$（11000011_B），寄存器 R0 中存放 0AA（10101010_B）。执行指令 ADD A, R0 A 中内容变为 $6D_H$（01101101_B）、辅助进位标志 AC 清零、进位标志和 OV 均置 1。
指令：	
ADD A, Rn	操作：(A) ← (A) + (Rn) 字节数：1 周期数：1 编码： 0 0 1 0 | 1 r r r
ADD A, direct	操作：(A) ← (A) + (direct) 字节数：2 周期数：1 编码： 0 0 1 0 | 0 1 0 1 | 直接地址

ADD A, @Ri	操作：(A) ← (A) + ((Ri)) 字节数：1 周期数：1 编码： 0 0 1 0 | 0 1 1 i
ADD A, #data	操作：(A) ← (A) + #data 字节数：2 周期数：1 编码： 0 0 1 0 | 0 1 0 0 | 立即数

带进位的加法

ADDC A, <src-byte>

表 4-6 ADDC

描述	ADDC 将指定的字节变量、进位标志和累加器 A 的内容相加，结果保存在 A 中。若位 7 和位 3 有进位，分别置位进位标志和辅助进位标志；否则清零。无符号整数相加时，进位标志指示溢出情况。 若位 6 和位 7 有一位产生进位而另一位不产生进位，OV 被置位；否则 OV 清零。OV = 1 表示两正数相加，和变成负数，或两负数相加和变成正数。 支持四种源操作数寻址方式：寄存器、直接、寄存器间接寻址，或立即寻址。
例子	累加器中存放 $0C3_H$（11000011_B），寄存器 R0 中存放 $0AA_H$（10101010_B），进位标志为 1。执行指令 ADDC A, R0 A 中内容变为 $6E_H$（01101110_B）、辅助进位标志 AC 清零、进位标志和 OV 均置 1。
指令：	
ADDC A, Rn	操作：(A) ← (A) + (C) + (Rn) 字节数：1 周期数：1 编码： 0 0 1 1 | 1 r r r
ADDC A, direct	操作：(A) ← (A) + (C) + (direct) 字节数：2 周期数：1 编码： 0 0 1 1 | 0 1 0 1 | 直接地址

ADDC A, @Ri	操作：(A) ← (A) + (C) + ((Ri)) 字节数：1 周期数：1 编码： 0 0 1 1 | 0 1 1 i
ADDC A, #data	操作：(A) ← (A) + (C) + #data 字节数：2 周期数：1 编码： 0 0 1 1 | 0 1 0 0 | 立即数

绝对跳转

AJMP addr11

表 4-7 AJUMP

描述：	AJMP 将跳转到指定地址上继续执行程序，该地址由当前 PC 值（已连续两次加 1）的高 5 位、操作码位 7-5 和指令的第二字节级联构成。跳转地址必须和 AJMP 下一指令的首字节地址在同一 2KB 范围内。
例子	标签“JMPADR”位于程序存储器的地址单元 0123_H。执行 0345_H 地址处的指令 AJMP JMPADR 则将 0123_H 加载到 PC。
指令：	
AJMP addr11	操作： (PC) ← (PC) + 2 (PC10-0) ← 页地址 字节数：2 周期数：2 编码： a10 a9 a8 0 | 0 0 0 1 | a7 a6 a5 a4 | a3 a2 a1 a0

字节变量的逻辑与操作

ANL <dest-byte>, <src-byte>

表 4-8 ANL(Byte)

描述：	ANL 执行指定字节变量的按位“与”操作，结果保存在目的变量中。该指令不影响标志位（<dest-byte> = A 时，该指令影响奇偶检验标志 P）。 两个操作数支持六种组合寻址方式。目的操作数为累加器时，源操作数可寄存器、直接、寄存器间接寻址，或立即寻址；目的操作数为直接地址时，源操作数可为累加器或立即数。 *注：用该指令修改输出端口时，端口数据的原始值从输出数据锁存器中读取，而不从输出口的引脚读取。*
例子：	累加器中存放 $0C3_H$（11000011_B），寄存器 R0 中存放 $0AA_H$（10101010_B），执行指令 ANL A, R0 A 中内容为 82_H（10000010_B）。目的操作数为直接寻址字节时，该指令将对硬件寄存器或 RAM 任意地址单元的位组合清零。决定清零位的屏蔽字节可以是指令中指定的常量，或是运行过程中累加器计算得到的值。 执行指令 ANL P1, # 01110011_B 输出端口 P1 的位 7、3 和 2 被清零。
指令：	
ANL A, Rn	操作：(A) ← (A) ^ (Rn) 字节数：1 周期数：1 编码： 0 1 0 1 1 r r r

ANL A, direct	操作：(A)← (A) ^ (direct) 字节数：2 周期数：1 编码： 0 1 0 1 | 0 1 0 1 | 直接地址
ANL A, @Ri	操作：(A) ← (A) ^ ((Ri)) 字节数：1 周期数：1 编码： 0 1 0 1 | 0 1 1 i
ANL A, #data	操作：(A)← (A) ^ #data 字节数：2 周期数：1 编码： 0 1 0 1 | 0 1 0 0 | 立即数
ANL direct, A	操作：(direct)← (direct) ^ (A) 字节数：2 周期数：1 编码： 0 1 0 1 | 0 0 1 0 | 直接地址
ANL direct, #data	操作：(direct)← (direct) ^ #data 字节数：3 周期数：2 编码： 0 1 0 1 | 0 0 1 1 | 直接地址 | 立即数

位变量的逻辑与

ANL C, <src-bit>

表 4-9 ANL(Bit)

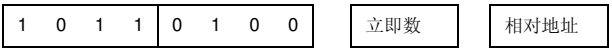
描述：	如果源操作位的逻辑值为‘0’将清零进位标志；否则进位标志值不变。源操作数前加斜杠（“/”）表示被寻址位取反之后作为源操作位，源操作位本身不受影响。该指令不影响其他标志位。 源操作数之能够直接位寻址。
例子：	当且仅当 P1.0 = 1，ACC.7 = 1，OV = 0，进位标志置 1： MOV C, P1.0 ；输入引脚状态加载至进位标志 ANL C, ACC.7；进位标志和累加器位 7 逻辑“与” ANL C, /OV；进位标志和溢出标志的取反值逻辑“与”
指令：	
ANL C, bit	操作： (C)← (C) ^ (bit) 字节数：2 周期数：2 编码： 1 0 0 0 | 0 0 1 0 | 位地址
ANL C, /bit	(C)← (C) ^ / (bit) 字节数：2 周期数：2 编码： 1 0 1 1 | 0 0 0 0 | 位地址

比较、不相等则跳转

CJNE <dest-byte>, <src-byte>, rel

表 4-10 CJNE

描述：	CJNE 比较前两个操作数的值，不相等则跳转。PC 值先加至指向下一指令，然后将 CJNE 指令的最后一个字节中的相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。如果<dest-byte>的无符号整数值小于<src-byte>的无符号整数值，进位标志被置位；否则进位标志清零。源和目的操作数都不受该指令影响。 前两个操作数支持四种寻址模式组合：累加器可和任何直接寻址字节或立即数进行比较；任何间接寻址 RAM 字节或工作寄存器寻址字节可和立即数进行比较。
例子：	累加器中存放 34_H，寄存器 R7 中存放 56_H，下面的第一条指令置位进位标志，并将程序跳转到 “NOT_EQ”标记的指令处，该指令通过检验进位标志可比较 R7 和 60_H 的大小。 CJNE R7, # 60_H, NOT_EQ ; ... .; R7 = 60_H NOT_EQ JC REQ_LOW; If R7< 60_H ; .... ;R7 > 60_H 若 P1 口的值也为 34_H，由于累加器和 P1 口读取的值相等，则指令： WAIT: CJNE A, P1, WAIT 清除进位标志，继续执行下条指令。（若 P1 口为其它值，程序将循环执行该指令，直到 P1 口变为 34_H 才跳出循环）。
指令：	
CJNE A, direct, rel	操作： (PC) ← (PC) + 3 若 (A) <> (direct) 则 (PC) ← (PC) + 相对偏移地址 若 (A) < (direct) 则 (C) ← 1 否则 (C) ← 0 字节数：3

	周期数：2 编码： 
CJNE A, #data, rel	操作： (PC) ← (PC) + 3 若 (A) <> data 则 (PC) ← (PC) + 相对偏移地址 若 (A) < data 则 (C) ← 1 否则 (C) ← 0 字节数：3 周期数：2 编码： 
CJNE Rn, #data, rel	操作： (PC) ← (PC) + 3 若 (Rn) <> data 则 (PC) ← (PC) + 相对偏移地址 若 (Rn) < data 则 (C) ← 1 否则 (C) ← 0 字节数：3 周期数：2 编码： 

CJNE @Ri, #data, rel	操作： (PC) ← (PC) + 3 若 ((Ri)) <> data 则 (PC) ← (PC) + 相对偏移地址 若 ((Ri)) < data 则 (C) ← 1 否则 (C) ← 0 字节数：3 周期数：2 编码： 

累加器清零

CLR A

表 4-11 CLR (A)

描述：	累加器清零（所有位都清零）。该指令不影响标志位。
例子：	累加器中存放 $5C_H$（01011100_B）。执行指令 CLR A 累加器的内容将为 00_H（00000000_B）。
指令：	
CLR A	操作：(A) ← 0 字节数：1 周期数：1 编码： 1 1 1 0 0 1 0 0

位清零

CLR <bit>

表 4-12 CLR (Bit)

描述：	指定位清零（复位至 0）。该指令不影响其它标志位。CLR 可对进位标志或任何可直接寻址位进行操作。
例子：	P1 口原先写入 $5D_H$（01011101_B）。执行指令 CLR P1.2 P1 口被设置为 59_H（01011001_B）。
指令：	
CLR C	操作：(C) ← 0 字节数：1 周期数：1 编码： 1 1 0 0 0 0 1 1
CLR bit	操作：(bit) ← 0 字节数：2 周期数：1 编码： 1 1 0 0 0 0 1 0 | 位地址

累加器取反
CPL A

表 4-13 CPL (A)

描述：	累加器中的每位都取反。原先为 1 取反变为 0，反之亦然。该指令不影响标志位。
例子：	累加器中存放 $5C_H$（01011100_B）。执行指令 CPL A 累加器的内容将为 $0A3_H$（10100011_B）。
指令：	
CPL A	操作：(A) ← / (A) 字节数：1 周期数：1 编码： 1 1 1 1 0 1 0 0

位取反

CPL <bit>

表 4-14 CPL (Bit)

描述：	指定位变量取反。原先为 1 取反变为 0，反之亦然。该指令不影响标志位。CPL 可对进位标志或任何可直接寻址位进行操作。 *注：用该指令修改输出端口值时，端口数据的原始值从输出锁存器中读取，而不从输出口的引脚读取。*
例子：	P1 口原先写入 $5D_H$（01011101_B）。执行指令序列 CPL P1.1 CPL P1.2 P1 口被设置为 $5B_H$（01011011_B）。
指令：	
CPL C	操作：(C) ← / (C) 字节数：1 周期数：1 编码： 1 0 1 1 | 0 0 1 1
CPL bit	操作：(bit) ← / (bit) 字节数：2 周期数：1 编码： 1 0 1 1 | 0 0 1 0 | 位地址

累加器的十进制加法调整

DA A

表 4-15 DA

描述：	两个压缩型 BCD 码相加结果存放在累加器中，DA A 将 A 中的 8 位结果调整为压缩型 BCD 码，生成两个 4 位值。该指令之前 ADD 或 ADDC 指令已执行加法操作。 若累加器的位 3-0 大于 9（xxxx1010 - xxxx1111），或者 AC 标志为 1，则累加器加 6 产生低四位的正确 BCD 编码；若低四位的进位一直传递到所有高四位，则置位进位标志，但它不会对进位标志清零。 若进位标志已置位，或累加器的位 7-4 大于 9（1010xxxx – 1111xxxx），则高四位加 6 产生正确的 BCD 编码；同样，若高四位有进位则置位进位标志，但它不会对进位标志清零。进位标志指示两个 BCD 加数之和是否大于 100，从而允许进行多精度十进制加法操作。OV 不受影响。 所有调整操作均在单指令周期内进行。实际上，该指令根据累加器的初值和 PSW 的状态，将 00_H、06_H、60_H 或 66_H 加到累加器以实现十进制转换。
例子：	累加器中的 56_H（01010110_B）代表十进制数 56 的压缩型 BCD 码；寄存器 R3 中的 67_H（01100111_B）代表十进制数 67 的压缩型 BCD 码；进位位置位。执行指令序列 ADDC A, R3 DA A 首先执行标准二进制补码加法操作，结果 $0BE_H$(10111110_B)存放在 A 中。进位标志和辅助进位标志清零。 接着 DA A 指令将累加器的值调整为 24_H（00100100_B），表明十进制数 56，67 和进位标值相加的和是以压缩型 BCD 码表示的十进制数 24，该指令还置位进位标志，表示产生十进制溢出，56，67 和 1 相加的真正结果为 124。 BCD 码加 01_H 或加 99_H 即实现 BCD 加 1 或减 1。若累加器中存放 30_H（代表十进制数 30），执行指令 ADD A, # 99H DA A 则进位标志置位，累加器中的值为 29_H（因为 30 + 99 = 129）。总和的低位字节产生可理解为：30 – 1 = 29。

指令：	
DA A	操作：累加器中的值为 BCD 码 若[[(A3-0) > 9] v [(AC) = 1]] 则 (A3-0) ← (A3-0) + 6 且 若[[(A7-4) > 9] v [(C) = 1]] 则 (A7-4) ← (A7-4) + 6 字节数：1 周期数：1 编码： 1 1 0 1 0 1 0 0

减 1

DEC <byte>

表 4-16 DEC

描述：	指定变量减 1。00_H 减 1 下溢为 $0FF_H$。该指令不影响标志位。支持四种操作数寻址方式：累加器、寄存器、直接或寄存器间接寻址。 *注：用该指令修改输出端口值时，端口数据的原始值从输出锁存器中读取，而不从输出口的引脚读取。*
例子：	寄存器 R0 中存放 $7F_H$（01111111_B）。内部 RAM 的地址单元 $7E_H$ 和 $7F_H$ 中分别存放 00_H 和 40_H。执行指令序列 DEC @R0 DEC R0 DEC @R0 R0 的值变为 $7E_H$，内部 RAM 地址单元 $7E_H$ 和 $7F_H$ 中的值分别为 $0FF_H$ 和 $3F_H$。
指令：	
DEC A	操作：(A) ← (A) -1 字节数：1 周期数：1 编码： 0 0 0 1 | 0 1 0 0
DEC Rn	操作：(Rn) ← (Rn) -1 字节数：1 周期数：1 编码： 0 0 0 1 | 1 r r r

DEC direct	操作：(direct) ← (direct) -1 字节数：2 周期数：1 编码： 0 0 0 1 | 0 1 0 1 | 直接地址
DEC @Ri	操作：((Ri)) ← ((Ri)) -1 字节数：1 周期数：1 编码： 0 0 0 1 | 0 1 1 i

除法

DIV AB

表 4-17 DIV

描述：	DIV AB 指令表示用累加器 A 中的无符号 8 位整数除以寄存器 B 中的无符号 8 位整数。将商存放在 A 中，余数存放在 B 中。进位和 OV 标志被清零。 例外情况：若 B 原先为 00_H，则返回累加器和 B 的结果不确定，溢出标志置位。任何情况下进位标志清零。
例子：	累加器中存放 251（$0FB_H$ 或 11111011_B），B 中存放 18（12_H 或 00010010_B）。执行指令 DIV AB A 中的值为 13（$0D_H$ 或 00001101_B），B 中的值为 17（11_H 或 00010001_B）（因为 251 = （13 × 18）+ 17）。进位和 OV 标志被清零。
指令：	
DIV AB	操作： (A) ← 商 [(A) / (B)] (B) ← 余数 [(A) / (B)] 字节数：1 周期数：4 编码： 1 0 0 0 | 0 1 0 0

减 1、不相等则跳转

DJNZ <byte>, <rel-addr>

表 4-18 DJNZ

描述：	DJNZ 对指定地址单元的值减 1，结果不为零则跳转到第二个操作数指定的地址上。00_H 减 1 下溢为 $0FF_H$。该指令不影响标志位。PC 值先加至指向下一指令的首字节地址，然后将 DJNZ 指令的最后一个字节，相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。 支持寄存器或直接寻址方式。 *注：用该指令修改输出端口值时，端口数据的原始值从输出锁存器中读取，而不从输出口的引脚读取。*
例子：	内部 RAM 的地址单元 40_H、50_H 和 60_H 中分别存放 01_H、70_H 和 15_H 。执行指令序列 DJNZ 40H, LABEL_1 DJNZ 50H, LABEL_2 DJNZ 60H, LABEL_3 程序跳转到标签 LABEL_2 指令处，内部 RAM 的地址单元 40_H、50_H 和 60_H 中分别存放 00_H、$6F_H$ 和 15_H 。因为第一条指令的减 1 结果为 0，故未执行跳转操作。 该指令提供了一个简单的方法来实现给定次数的循环；或产生适当的时延（2 到 512 个机器周期）。执行指令序列 MOV R2, #8 TOGGLE: CPL P1.7 DJNZ R2, TOGGLE P1.7 将翻转 8 次，P1 口的位 7 产生四个输出脉冲。每个脉冲持续三个机器周期，DJNZ 指令消耗两个机器周期，引脚翻转消耗一个机器周期。
指令：	

DJNZ Rn, rel	操作： (PC) ← (PC) + 2 (Rn) ← (Rn) - 1 若 (Rn) > 0 或 (Rn) < 0 则 (PC) ← (PC) + rel 字节数：2 周期数：2 编码： 1 1 0 1 1 r r r 相对地址
DJNZ direct, rel	操作： (PC) ← (PC) + 2 (direct) ← (direct) – 1 若 (direct) > 0 或 (direct) < 0 则 (PC) ← (PC) + rel 字节数：3 周期数：2 编码： 1 1 0 1 0 1 0 1 直接地址 相对地址

加 1

INC <byte>

表 4-19 INC (Byte)

描述：	指定变量加 1。$0FF_H$ 加 1 上溢为 00_H。该指令不影响标志位。支持三种操作数寻址方式：寄存器、直接或寄存器间接寻址。 *注：用该指令修改输出端口值时，端口数据的原始值从输出锁存器中读取，而不从输出口的引脚读取。*
例子：	寄存器 R0 中存放 $7E_H$（01111110_B）。内部 RAM 的地址单元 $7E_H$ 和 $7F_H$ 中分别存放 $0FF_H$ 和 40_H。执行指令 INC @R0 INC R0 INC @R0 R0 的值变为 $7F_H$，内部 RAM 地址单元 $7E_H$ 和 $7F_H$ 中的值分别为 00_H 和 41_H。
指令：	
INC A	操作：(A) ← (A) + 1 字节数：1 周期数：1 编码： 0 0 0 0 | 0 1 0 0
INC Rn	操作：(Rn) ← (Rn) + 1 字节数：1 周期数：1 编码： 0 0 0 0 | 1 r r r

INC direct	操作：(direct) ← (direct) + 1 字节数：2 周期数：1 编码： 0 0 0 0 | 0 1 0 1 | 直接地址
INC @Ri	操作：((Ri)) ← ((Ri)) + 1 字节数：1 周期数：1 编码： 0 0 0 0 | 0 1 1 i

数据指针加 1
INC DPTR

表 4-20 INC (DPTR)

描述：	16 位数据指针加 1。执行 16 位加 1 操作：若数据指针的低位字节（DPL）从 $0FF_H$ 到 00_H 溢出，则高位字节（DPH）加 1。该指令不影响标志位。 DPTR 是唯一可执行加 1 操作的 16 位寄存器。
例子：	寄存器 DPH 和 DPL 中存放 12_H 和 $0FE_H$。执行指令序列 INC DPTR INC DPTR INC DPTR DPH 和 DPL 的值变为 13_H 和 01_H。
指令：	
INC DPTR	操作：(DPTR) ← (DPTR) + 1 字节数：1 周期数：2 编码： 1 0 1 0 | 0 0 1 1

指定位被置 1 则跳转

JB <bit>, rel

表 4-21 JB

描述：	若指定位的值为 1，则跳转到指定地址上；否则继续执行下条指令。PC 值先加至指向下一指令的首字节地址，然后将 JB 指令的第三个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。指定位不会被修改，该指令不影响标志位。
例子：	P1 口上的值为 11001010_B。累加器中存放 56_H（01010110_B）。执行指令序列 JB P1.2, LABEL 1 JB ACC.2, LABEL 2 程序将跳转到标签 LABEL2 指令处继续执行。
指令：	
JB bit, rel	操作： (PC) ← (PC) + 3 若(bit) = 1 则(PC) ← (PC) +rel 字节数：3 周期数：2 编码： 0 0 1 0 | 0 0 0 0 | 位地址 | 相对地址

指定位为 1 则跳转，并将指定位清零

JBC <bit>, rel

表 4-22 JBC

描述：	若指定位的值为 1，则跳转到指定地址上；否则继续执行下条指令。无论哪种情况均对指定位清零。PC 值先加至指向下一指令的首字节地址，然后将 JB 指令的第三个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。该指令不影响标志位。 *注：用该指令测试输出引脚时，端口数据的原始值从输出锁存器中读取，而不从输出口的引脚读取。*
例子：	累加器中存放 56_H（01010110_B）。执行指令序列 JBC ACC.3, LABEL 1 JBC ACC.2, LABEL 2 程序将跳转到标签 LABEL2 指令处继续执行，累加器的值变为 52_H（01010010_B）。
指令：	
JBC bit, rel	操作： (PC) ← (PC) + 3 若(bit) = 1 则(bit) ← 0 (PC) ← (PC) + rel 字节数：3 周期数：2 编码： 0 0 0 1 | 0 0 0 0 | 位地址 | 相对地址

进位标志被置 1 则跳转

JC rel

表 4-23 JC

描述：	若进位标志位为 1，则跳转到指定地址上；否则继续执行下条指令。PC 值先两次加 1，然后将 JC 指令的第二个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。该指令不影响标志位。
例子：	进位标志被清零，执行指令序列 JC LABEL 1 CPL C JC LABEL 2 进位标志被置 1，程序将跳转到标签 LABEL2 指令处执行。
指令：	
JC rel	操作： (PC) ← (PC) + 2 若(C) = 1 则(PC) ← (PC) + rel 字节数：2 周期数：2 编码： 0 1 0 0 0 0 0 0 | 相对地址

间接跳转

JMP @A + DPTR

表 4-24 JMP

描述：	累加器 8 位无符号整数和 16 位数据指针相加，结果存放在程序计数器 PC 中，作为下条指令的取指地址。执行 16 位加法操作（模 2^{16}）：低位字节的进位会传递给高位字节。累加器和数据指针的值都不改变。该指令不影响标志位。
例子：	累加器中存放 0-6 之间的一个偶数。执行下列指令，程序将跳转到从标签 JMP_TBL 开始的四条 AJMP 指令中的一条指令上。 MOV DPTR, #JMP_TBL JMP @A + DPTR JMP_TBL: AJMP LABEL 0 AJMP LABEL 1 AJMP LABEL 2 AJMP LABEL 3 若累加器的当前值为 04_H，程序将跳转到标签 LABEL2 指令处。AJMP 为双字节指令，故跳转地址之间隔一个地址字节。
指令：	
JMP @A + DPTR	操作：(PC) ← (A) + (DPTR) 字节数：1 周期数：2 编码： 0 1 1 1 0 0 1 1

指定位为 0 则跳转
JNB <bit>, rel

表 4-25 JNB

描述：	若指定位的值为 0，则跳转到指定地址上；否则继续执行下条指令。PC 值先加至指向下一指令的首字节地址，然后将 JNB 指令的第三个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。指定位不会被该指令修改，标志位不受影响。
例子：	P1 口上的值为 11001010_B。累加器中存放 $56_H(01010110_B)$。执行指令序列 JNB P1.3, LABEL 1 JNB ACC.3, LABEL 2 程序将跳转到标签 LABEL2 指令处继续执行。
指令：	
JNB bit, rel	操作： (PC) ← (PC) + 3 若(bit) = 0 则(PC) ← (PC) + rel 字节数：3 周期数：2 编码： 0 0 1 1 | 0 0 0 0 | 位地址 | 相对地址

进位标志为 0 则跳转

JNC rel

表 4-26 JNC

描述：	若进位标志位为 0，则跳转到指定地址上；否则继续执行下条指令。PC 值先两次加 1 指向下条指令，然后将 JNC 指令的第二个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。该指令不修改进位标志。
例子：	进位标志被置位，执行指令序列 JNC LABEL 1 CPL C JNC LABEL 2 进位标志被清零，程序将跳转到标签 LABEL2 指令处执行。
指令：	
JNC rel	操作： (PC) ← (PC) + 2 若(C) = 0 则(PC) ← (PC) + rel 字节数：2 周期数：2 编码： 0 1 0 1 | 0 0 0 0 | 相对地址

累加器非 0 则跳转
JNZ rel

表 4-27 JNZ

描述：	若累加器中任意位为 1，则跳转到指定地址上；否则继续执行下条指令。PC 值先加 2 指向下条指令，然后将 JNZ 指令的第二个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。该指令不修改累加器内容，标志位不受影响。
例子：	累加器原先值为 00_H，执行指令序列 JNZ LABEL 1 INC A JNZ LABEL 2 累加器的值变为 01_H，程序将跳转到标签 LABEL2 指令处继续执行。
指令：	
JNZ rel	操作： (PC) ← (PC) + 2 若(A) ≠ 0 则(PC) ← (PC) + rel 字节数：2 周期数：2 编码： 0 1 1 1 | 0 0 0 0 | 相对地址

累加器为 0 则跳转

JZ rel

表 4-28 JZ

描述：	若累加器中所有位均为 0，则跳转到指定地址上；否则继续执行下条指令。PC 值先加 2 指向下条指令，然后将 JZ 指令的第二个字节，即相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。该指令不修改累加器内容，标志位不受影响。
例子：	累加器原先值为 01_H，执行指令 JZ LABEL 1 DEC A JZ LABEL 2 累加器的值变为 00_H，程序将跳转到标签 LABEL2 指令处继续执行。
指令：	
JZ rel	操作： (PC) ← (PC) + 2 若(A) = 0 则(PC) ← (PC) + rel 字节数：2 周期数：2 编码： 0 1 1 0 | 0 0 0 0 | 相对地址

长子程序调用
LCALL addr16

表 4-29 LCALL

描述：	LCALL 调用指定地址处的子程序。指令先将 PC 值加 3 指向下条指令的地址，然后将 16 位当前地址压栈（低位字节在先），堆栈指针加 2。接着将 LCALL 指令的第二、三字节分别装入 PC 的高位和低位字节，程序从该地址继续执行。子程序可从 64KB 程序存储器地址空间的任何地方开始。本指令不影响标志位。
例子：	SP 初始化值为 07_H。标签“SUBRTN”对应程序存储器的地址单元 1234_H。执行 0123_H 地址上的指令 LCALL SUBTRN SP 将指向 09_H。内部 RAM 的地址单元 08_H 和 09_H 分别保存 26_H 和 01_H。PC 当前值为 1234_H。
指令：	
LCALL addr16	操作： (PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC7-0) (SP) ← (SP) + 1 ((SP)) ← (PC15-8) (PC) ← addr15-0 字节数：3 周期数：2 编码： 0 0 0 1 | 0 0 1 0 | 地址 15…8 | 地址 7…0

长跳转指令
LJMP addr16

表 4-30 LJMP

描述：	LJMP 为无条件跳转指令，指令的第二、三字节分别装入 PC 的高位和低位字节，决定目的地址。目的地址可指向 64KB 程序存储器地址空间的任何单元。本指令不影响标志位。
例子：	标签“JMPADR”对应程序存储器的地址单元 1234_H。执行 0123_H 地址上的指令 LJMP JMPADR PC 的值变为 1234_H。
指令：	
LJMP addr16	操作：(PC) ← addr15-0 字节数：3 周期数：2 编码： 0 0 0 0 0 0 1 0 | 地址 15…8 | 地址 7…0

传送字节变量

MOV <dest-byte>,<src-byte>

表 4-31 MOV (Byte)

描述：	将第二个操作数指定的字节变量复制到第一个操作数指定的地址单元中。该指令不影响源字节，其它寄存器或标志位不受影响。 该指令最灵活，支持 15 种源操作数和目的操作数寻址模式的组合。
例子：	内部 RAM 的地址单元 30_H 中存放 40_H；地址 40_H 中存放 10_H。输入 P1 口为 11001010_B（$0CA_H$）。执行指令序列 MOV R0, #30H ; R0 <= 30H MOVA, @R0 ; A <= 40H MOVR1, A ; R1 <= 40H MOVB, @R0 ; B <= 10H MOV@R1, P1 ; RAM (40H) <= 0CAH MOVP2, P1 ; P2 <= 0CAH R0 中的值为 30_H，累加器和 R1 中的值为 40_H，寄存器 B 的值为 10_H，RAM 地址单元 40_H 和 P2 口输出值均为 $0CA_H$（11001010_B）。
指令：	
MOV A, Rn	操作： (A) ← (Rn) 字节数：1 周期数：1 编码： 1 1 1 0 1 r r r

MOV A, direct

操作：

(A)← (direct)

字节数：2

周期数：1

编码：

1	1	1	0	0	1	0	1	直接地址

注：MOV A, ACC 是无效指令。执行该指令后累加器内的值不确定。

MOV A, @Ri

操作：

(A) ← ((Ri))

字节数：1

周期数：1

编码：

1	1	1	0	0	1	1	i

MOV A, #data

操作：

(A)← #data

字节数：2

周期数：1

编码：

0	1	1	1	0	1	0	0	立即数

MOV Rn, A

操作：

(Rn)← (A)

字节数：1

周期数：1

编码：

1	1	1	1	1	r	r	r

MOV Rn, direct	操作： (Rn)← (direct) 字节数：2 周期数：2 编码： 1 0 1 0 1 r r r 直接地址
MOV Rn, #data	操作： (Rn)← #data 字节数：2 周期数：1 编码： 0 1 1 1 1 r r r 立即数
MOV direct, A	操作： (direct)← (A) 字节数：2 周期数：1 编码： 1 1 1 1 0 1 0 1 直接地址
MOV direct, Rn	操作： (direct)← (Rn) 字节数：2 周期数：2 编码： 1 0 0 0 1 r r r 直接地址

MOV direct, direct

操作：(direct)← (direct)
字节数：3
周期数：2
编码：

1	0	0	0	0	1	0	1	直接地址（源操作数）	直接地址(目的操作数)

MOV direct @Ri

操作：
(direct)← ((Ri))
字节数：2
周期数：2
编码：

1	0	0	0	0	1	1	i	直接地址	

MOV direct, #data

操作：(direct)← #data
字节数：3
周期数：2
编码：

0	1	1	1	0	1	0	1	直接地址	立即数

MOV @Ri, A

操作：
((Ri))← (A)
字节数：1
周期数：1
编码：

1	1	1	1	0	1	1	i		

MOV @Ri, direct

操作：((Ri))← (direct)
字节数：2
周期数：2
编码：

1	0	1	0	0	1	1	i	直接地址	

MOV @Ri, #data	操作： ((Ri))← #dara 字节数：2 周期数：1 编码： 0 1 1 1 | 0 1 1 i | 立即数

传送位数据

MOV <dest-bit>, <src-bit>

表 4-32 MOV (Bit)

描述：	将第二个操作数指定的布尔变量复制到第一个操作数指定的地址单元中。两个操作数其中一个必须为进位标志，另一个可为任何可寻址位。该指令不影响其它寄存器或标志位。

例子：	进位标志原先为 1。输入 P3 口的值为 11000101_B，输出 P1 口原先写入 35_H（00110101_B）。执行指令序列
	MOV P1.3, C
	MOV C, P3.3
	MOV P1.2, C
	进位标志被清零，P1 口变为 39_H（00111001_B）。

指令：	
MOV C, bit	操作：(C) ← (bit) 字节数：2 周期数：1 编码： 1 0 1 0 | 0 0 1 0 | 位地址
MOV bit, C	操作：(bit)← (C) 字节数：2 周期数：2 编码： 1 0 0 1 | 0 0 1 0 | 位地址

将 16 位立即数装入数据指针
MOV DPTR, #data16

表 4-33 MOV (DPTR)

描述：	将 16 位指定常数装入数据指针。16 位常数分别存放在指令的第二、三字节中。第二字节（DPH）为高位字节；第三字节（DPL）为低位字节。该指令不影响标志位。 这是唯一一条一次转移 16 位数据的指令。
例子：	执行指令 MOV DPTR, #1234H 值 1234_H 装入数据指针：DPH 保存 12_H，DPL 保存 34_H。
指令：	
MOV DPTR, #data16	操作： (DPTR) ← #data15-0 DPH, DPL← #data15-8, #data7-0 字节数：3 周期数：2 编码： 1 0 0 1 | 0 0 0 0 | 立即数 15…8 | 立即数 7…0

读取代码字节

MOVC A, @A + <base-reg>

表 4-34 MOVC (Read)

描述：	将程序存储器中的代码字节或常量装入累加器。地址由无符号 8 位累加器的值和 16 位基址寄存器（数据指针或 PC）的值之和组成。若基址寄存器为 PC，则 PC 应首先加至指向下条指令、然后再和累加器相加；否则基址寄存器内容不变。执行 16 位加法操作，从而低位字节进位可传递到高位字节。该指令不影响标志位。
例子：	累加器的值在 0-3 之间。以下指令会将累加器的值转换为由 DB（定义字节）伪指令定义的四个值中的一个。 REL_PC: INC A MOVC A, @A + PC RET DB 66H DB 77H DB 88H DB 99H 若调用子程序时 A 为 01_H，则执行以上指令后 A 变为 77_H。MOVC 前的 INC A 指令先跳过 RET 指令获取查找表中的内容，然后执行 RET 从子程序返回。若 MOVC 和查找表之间有多个指令字节，要累加相应的字节数到 A 中。
指令：	
MOVC A, @A + DPTR	操作： (A) ← ((A) + (DPTR)) 字节数：1 周期数：2 编码： 1 0 0 1 0 0 1 1

MOVC A, @A + PC	操作： (PC) ← (PC) + 1 (A) ← ((A) + (PC)) 字节数：1 周期数：2 编码： 1 0 0 0 | 0 0 1 1

写入代码字节

MOVC @ (DPTR++), A

表 4-35 MOVC (Write)

描述：	将累加器的内容存入数据指针指向的程序存储器单元。执行写操作后数据指针硬件加 1。该指令不影响标志位。

例子：	将 $E4_H$ 存入存储器地址单元 1000_H 中。操作码 $E4_H$ 代表 CLR A 指令。执行指令
	MOV A, # E4H
	MOV DPTR, #1000H
	MOVC @(DPTR++), A
	指令 CLR A 写入存储器地址单元 1000_H

指令：	
MOVC @(DPTR++), A	操作： ((DPTR)) ← (A) (DPTR) = (DPTR) +1 字节数：1 周期数：2 编码： 1 0 1 0 | 0 1 0 1

注：该指令为 XC800 专用指令，标准 8051 汇编器不支持该指令。此类情况下，该功能可通过直接字节声明和定义来实现，如 ".byte #A5H" (语法由汇编器决定)。

注：该指令和另一条 XC800 专用指令 TRAP 共用操作码，只有 EO.TRAP_EN = 0 时选择该 MOVC 指令。

外部传送

MOVX <dest-byte>, <src-byte>

表 4-36 MOVX

描述：	MOVX 指令在累加器和外部数据存储器之间传送字节数据，故在 MOV 后添加“X”。MOVX 有两种类型，分别对应 8 位或 16 位间接寻址外部数据 RAM。 MOVX 指令的第一种类型，当前寄存器组的 R0 或 R1 提供了低位字节地址端口的 8 位地址。对于外部扩展解码或较小的 RAM 区域，8 位地址足够使用。对于较大容量的 RAM，可用任意输出端口引脚输出高位地址字节。由 MOVX 指令之前的输出指令对这些引脚加以控制。 MOVX 指令的第二种类型，数据指针产生 16 位地址。高位字节地址端口输出高 8 位地址（存放在 DPH 中）；低位字节地址端口输出低 8 位地址（存放在 DPL 中）。地址端口 SFR 不受影响保持原值。由于无需额外指令来设置输出端口，因此 16 位寻址方式更为快速和高效。 某些情况下这两种 MOVX 指令类型可混合使用。大容量 RAM 地址端口的高位地址由数据指针寻址，或者编程将高位地址输出到高位字节地址端口上；接着执行由 R0 或 R1 寻址的 MOVX 指令。

例子：	外部 256 字节 RAM 的地址/数据线复用，和低位字节地址端口相连。P3 口用来控制外部 RAM，其它端口（如高位字节地址端口）用作正常 I/O 口。R0 和 R1 中分别为 12_H 和 34_H，外部 RAM 地址单元 34_H 中存放 56_H。执行指令
	MOVX A, @R1
	MOVX @R0, A
	值 56_H 被复制到累加器和外部 RAM 的地址单元 12_H 中。

指令：	
MOVX A, @Ri	操作：(A) ← ((Ri)) 字节数：1 周期数：2 编码： 1 1 1 0 0 0 1 i

MOVX A, @DPTR	操作：(A) ← ((DPTR)) 字节数：1 周期数：2 编码： 1 1 1 0 0 0 0 0
MOVX @Ri, A	操作：((Ri)) ← (A) 字节数：1 周期数：2 编码： 1 1 1 1 0 0 1 i
MOVX @DPTR, A	操作：((DPTR)) ← (A) 字节数：1 周期数：2 编码： 1 1 1 1 0 0 0 0

乘法指令

MUL AB

表 4-37 MUL

描述：	MUL AB 将累加器 A 和寄存器 B 中的无符号 8 位整数相乘。16 位乘积的低位字节保存在累加器中；高位字节保存在 B 中。若乘积大于 256（$0FF_H$）则置位溢出标志，否则清零。进位标志始终清零。
例子：	累加器的原值为 80（50_H）。寄存器 B 的值为 160（$0A0_H$）。执行指令 MUL AB 乘积为 12,800（3200_H），故 B 的值为 32_H（00110010_B），累加器清零。溢出标志置位，进位标志清零。
指令：	
MUL AB	操作： (B) ←高位字节[(A) × (B)] (A) ←低位字节[(A) × (B)] 字节数：1 周期数：4 编码： 1 0 1 0 | 0 1 0 0

空操作
NOP

表 4-38 NOP

描述：	继续执行下条指令。该指令只影响 PC 值，寄存器和标志位不受影响。
例子：	要在 P2 口的位 7 产生一个持续 5 个时钟周期的低电平脉冲。SETB/CLR 指令会产生一个时钟周期的脉冲，故必须额外插入四个周期，可由以下指令操作来实现（假定禁止产生中断）。 CLR P2.7 NOP NOP NOP NOP SETB P2.7
指令：	
NOP	操作：NULL 字节数：1 周期数：1 编码： 0 0 0 0 | 0 0 0 0

字节变量的逻辑“或”

ORL <dest-byte>, <src-byte>

表 4-39 ORL (Byte)

描述：	ORL 执行指定字节变量的按位“或”操作，结果保存在目的变量中。该指令不影响标志位（<dest-byte> = A 时，该指令影响奇偶检验标志 P）。 两个操作数支持六种组合寻址方式。目的操作数为累加器时，源操作数可为寄存器、直接、寄存器间接寻址，或立即寻址；目的操作数为直接地址时，源操作数可为累加器或立即数。 *注：用该指令修改输出端口值时，端口数据的原始值从输出数据锁存器中读取，而不从输出口的引脚读取。*
例子：	累加器中存放 $0C3_H$（11000011_B），寄存器 R0 中存放 55_H（01010101_B）。执行指令 ORL A, R0 A 的值为 $0D7_H$（11010111_B）。 目的操作数为直接寻址字节时，该指令将对 SFR 或 RAM 任意地址单元的位组合置位。决定被置位位的屏蔽字节可以是指令中指定的常量，或是运行过程中累加器计算得到的值。执行指令 ORL P1, # 00110010_B 输出端口 P1 的位 5、4 和 1 被置位。
ORL A, Rn	操作：(A) ← (A) v (Rn) 字节数：1 周期数：1 编码： 0 1 0 0 1 r r r
ORL A, direct	操作：(A)← (A) v (direct) 字节数：2 周期数：1 编码： 0 1 0 0 0 1 0 1 直接地址

ORL A, @Ri	操作：(A) ← (A) v ((Ri)) 字节数：1 周期数：1 编码： 0 1 0 0 | 0 1 1 i
ORL A, #data	操作：(A)← (A) v #data 字节数：2 周期数：1 编码： 0 1 0 0 | 0 1 0 0 | 立即数
ORL direct, A	操作：(direct)← (direct) v (A) 字节数：2 周期数：1 编码： 0 1 0 0 | 0 0 1 0 | 直接地址
ORL direct, #data	操作：(direct)← (direct) v #data 字节数：3 周期数：2 编码： 0 1 0 0 | 0 0 1 1 | 直接地址 | 立即数

位变量的逻辑“或”

ORL C, <src-bit>

表 4-40 ORL (Bit)

描述：	布尔变量为 ‘1’ 则置位进位标志；否则进位标志值不变。源操作数前加斜杠（“/”）表示对被寻址位取反作为源操作位，但被寻址位本身不受影响，其它标志位不受影响。
例子：	当且仅当 P1.0 = 1，ACC.7 = 1，或 OV = 0，置位进位标志： MOV C, P1.0 ;输入引脚 P1.0 状态送至进位标志 ORL C, ACC.7 ;进位标志和累加器第 7 位逻辑“或” ORL C, /OV ;进位标志和溢出标志的取反值逻辑“或”
指令：	
ORL C, bit	操作：(C)← (C) v (bit) 字节数：2 周期数：2 编码： 0 1 1 1 | 0 0 1 0 | 位地址
ORL C, /bit	操作：(C)← (C) v / (bit) 字节数：2 周期数：2 编码： 1 0 1 0 | 0 0 0 0 | 位地址

出栈

POP direct

表 4-41 POP

描述：	从堆栈指针指向的内部 RAM 地址单元中读取内容，堆栈指针减 1。读取的数值送给指定的直接寻址单元。该指令不影响标志位。
例子：	堆栈指针 SP 原先指向 32_H，内部 RAM 地址单元 30_H 到 32_H 分别存放值 20_H，23_H 和 01_H。执行指令序列 POP DPH POP DPL SP 将指向 30_H，数据指针将指向 0123_H。然后执行 POP SP SP 将指向 20_H。注意在这种特殊情况下，SP 在装入弹出值（20_H）之前先减 1 变为 $2F_H$。
指令：	
POP direct	操作： (direct)← ((SP)) (SP) ← (SP) - 1 字节数：2 周期数：2 编码： 1 1 0 1 | 0 0 0 0 | 直接地址

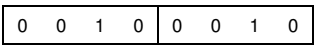
压栈
PUSH direct

表 4-42 PUSH

描述：	堆栈指针加 1，然后将指定变量的内容复制到堆栈指针指向的内部 RAM 地址单元中。该指令不影响标志位。
例子：	进入某中断服务程序前堆栈指针指向 09_H，数据指针指向 0123_H。执行指令 PUSH DPL PUSH DPH 堆栈指针将指向 $0B_H$，值 23_H 和 01_H 分别存入内部 RAM 的地址单元 $0A_H$ 和 $0B_H$ 中。
指令：	
PUSH direct	操作： (SP) ← (SP) + 1 ((SP)) ← (direct) 字节数：2 周期数：2 编码： 1 1 0 0 | 0 0 0 0 | 直接地址

从子程序返回
RET

表 4-43 RET

描述：	RET 从堆栈中连续弹出 PC 的高位和低位字节，堆栈指针减 2。程序从当前 PC 指令地址继续执行，通常该指令紧跟在 ACALL 或 LCALL 指令之后。该指令不影响标志位。
例子：	堆栈指针 SP 原先指向 $0B_H$，内部 RAM 地址单元 $0A_H$ 和 $0B_H$ 分别存放值 23_H 和 01_H。执行指令 RET 堆栈指针将指向 09_H。程序从地址单元 0123_H 继续执行。
指令：	
RET	操作： (PC15-8) ← ((SP)) (SP) ← (SP) - 1 (PC7-0) ← ((SP)) (SP) ← (SP) – 1 字节数：1 周期数：2 编码： 

从中断返回
RETI

表 4-44 RETI

描述：	RETI 从堆栈中连续弹出 PC 的高位和低位字节，恢复中断逻辑控制响应其它同级中断。堆栈指针减 2。其它寄存器不受影响；PSW 不会自动恢复到中断前的状态。程序从弹出的 PC 指令地址处继续执行，通常即为检测到中断请求之后紧跟的指令。执行 RETI 指令时若有已挂起的低级或同级中断，在响应这些挂起中断之前要再执行一条指令。
例子：	堆栈指针 SP 原先指向 $0B_H$，在指令结束前的地址单元 0122_H 检测到某个中断。内部 RAM 地址单元 $0A_H$ 和 $0B_H$ 分别存放值 23_H 和 01_H。执行指令 RETI 堆栈指针将指向 09_H。程序从地址单元 0123_H 继续执行。
指令：	
RETI	操作： (PC15-8) ← ((SP)) (SP) ← (SP) - 1 (PC7-0) ← ((SP)) (SP) ← (SP) – 1 字节数：1 周期数：2 编码： 0 0 1 1 | 0 0 1 0

累加器左环移
RL A

表 4-45 RL

描述：	累加器中的 8 位位元左移一位。位 7 内容移入位 0。该指令不影响标志位。
例子：	累加器中存放的值为 $0C5_H$（11000101_B）。执行指令 RL A 累加器中的值变为 $8B_H$（10001011_B），进位标志不受影响。
指令：	
RL A	操作： (An + 1) ← (An) n = 0-6 (A0) ← (A7) 字节数：1 周期数：1 编码： 0 0 1 0 0 0 1 1

带进位标志的累加器左环移

RLC A

表 4-46 RLC

描述：	累加器中的 8 位位元和进位标志一起左移一位。位 7 移入进位标志位；原进位标志位移入位 0。该指令不影响其他标志位。
例子：	累加器中存放的值为 $0C5_H$（11000101_B），进位标志为 0。执行指令 RLC A 累加器中的值变为 $8A_H$（10001010_B），进位标志为 1。
指令：	
RLC A	操作： (An + 1) ← (An) n = 0-6 (A0) ← (C) (C) ← (A7) 字节数：1 周期数：1 编码： 0 0 1 1 0 0 1 1

累加器右环移

RR A

表 4-47 RR

描述：	累加器中的 8 位位元右移一位。位 0 内容移入位 7。该指令不影响标志位。
例子：	举例：累加器中存放的值为 $0C5_H$（11000101_B）。执行指令 RR A 累加器中的值变为 $0E2_H$（11100010_B），进位标志不受影响。
指令：	
RR A	操作： (An) ← (An + 1) n =0-6 (A7) ← (A0) 字节数：1 周期数：1 编码： 0 0 0 0 0 0 1 1

带进位标志的累加器右环移

RRC A

表 4-48 RRC

描述：	累加器中的 8 位位元和进位标志一起右移一位。位 0 移入进位标志位；原进位标志位移入位 7。该指令不影响其他标志位。
例子：	累加器中存放的值为 $0C5_H$（11000101_B），进位标志为 0。执行指令 RRC A 累加器中的值变为 62_H（01100010_B），进位标志为 1。
指令：	
RRC A	操作： (An) ← (An + 1) n =0-6 (A7) ← (C) (C) ← (A0) 字节数：1 周期数：1 编码： 0 0 0 1 | 0 0 1 1

位变量置位操作

SETB <bit>

表 4-49 RRC

描述：	SETB 对指定位置 1。SETB 可对进位标志或任何可直接寻址位进行操作。该指令不影响其它标志位。
例子：	进位标志被清零。输出 P1 口原先写入 34_H（00110100_B）。 执行指令 SETB C SETB P1.0 进位标志变为 1，P1 口输出值变为 35_H（00110101_B）。
指令：	
SETB C	操作：(C) ← 1 字节数：1 周期数：1 编码： 1 1 0 1 | 0 0 1 1
SETB bit	操作：(bit) ← 1 字节数：2 周期数：1 编码： 1 1 0 1 | 0 0 1 0 | 位地址

短跳转

SJMP rel

表 4-50 SJMP

描述：	SJMP 无条件跳转到指定地址，PC 值先加 2，然后将 SJMP 指令的第二字节，相对偏移地址（有符号数）和 PC 值相加，计算得到跳转的目的地址。因此跳转地址可在该指令的前 128 字节到后 127 字节之间。
例子：	标签“RELADR”对应程序存储器的地址单元 0123_H。执行 0100_H 地址上的指令 SJMP RELADR PC 的值变为 0123_H。 *注：SJMP 的下一指令地址为 102_H。因此指令的偏移字节为相对偏移量（0123_H -0102_H）= 21_H。换言之，若 SJMP 的偏移量为 $0FE_H$ 该指令会死循环。*
指令：	
SJMP rel	操作： (PC) ← (PC) + 2 (PC) ← (PC) + rel 字节数：2 周期数：2 编码： 1 0 0 0 0 0 0 0 | 相对地址

带借位的减法

SUBB A, <src-byte>

表 4-51 SUBB

描述：	SUBB 从累加器 A 中减去指定的字节变量和进位标志，结果保存在 A 中。若位 7 要借位，SUBB 置位进位（借位）标志；否则 C 被清零。（若执行 SUBB 指令前 C 已置位，表明在多精度减法计算中提前需要借位，故借位标志位和源操作数一同从累加器中减掉）。若位 3 要借位，SUBB 置位辅助进位（借位）标志 AC；否则 AC 被清零。若位 6 和位 7 有一位要借位而另一位不借位，OV 被置位。 有符号数相减时，OV = 1 表示从一个正数中减去一个负数结果为负数，或从一个负数中减去一个正数结果为正数。 支持四种源操作数寻址方式：寄存器、直接、寄存器间接寻址，或立即寻址。
例子：	累加器中存放 $0C9_H$（11001001_B），寄存器 R2 中存放 54_H（01010100_B），进位标志置 1。执行指令 SUBB A, R2 A 中内容为 74_H（01110100_B）、进位标志和 AC 均清零，OV 置 1。 注意 $0C9_H$ 减 54_H 的结果为 75_H。由于在执行 SUBB 前借位标志已置 1，故运算结果为 74_H。若在执行单精度或多精度减法运算前进位标志状态未知，则应使用指令 CLR C 对其清零。
指令：	
SUBB A, Rn	操作：(A) ← (A) - (C) - (Rn) 字节数：1 周期数：1 编码： 1 0 0 1 1 r r r

SUBB A, direct

操作：(A) ← (A) - (C) - (direct)
字节数：2
周期数：1
编码：

1	0	0	1	0	1	0	1	直接地址

SUBB A, @Ri

操作：(A) ← (A) - (C) - ((Ri))
字节数：1
周期数：1
编码：

1	0	0	1	0	1	1	i

SUBB A, #data

操作：(A) ← (A) - (C) - #data
字节数：2
周期数：1
编码：

1	0	0	1	0	1	0	0	立即数

累加器中半字节交换
SWAP A

表 4-52 SWAP

描述：	SWAP A 指令将累加器中的低四位（位 3-0）和高四位（位 7-4）互换。该指令可看作是四位环移指令。标志位不受影响。
例子：	累加器中存放 $0C5_H$（11000101_B）。执行指令 SWAP A 累加器中的值变为 $5C_H$（01011100_B）。
指令：	
SWAP A	操作：(A3-0) ↔ (A7-4) 字节数：1 周期数：1 编码： 1 1 0 0 | 0 1 0 0

软件断点
TRAP

表 4-53　　TRAP

描述：	产生软件断点。在机器周期的 P1 结束时进入调试模式。该指令不影响标志位。
例子：	若 EO.TRAP_EN = 1，操作码 $A5_H$ 表示 TRAP 指令。 MOV A, #55H TRAP　　;断点 INC A
指令：	
TRAP	操作：Break 字节数：1 周期数：1 编码： 1 0 1 0 | 0 1 0 1

注：该指令为 XC800 专用指令，标准 8051 汇编器不支持该指令。此情况下，该功能可通过直接字节声明和定义来实现，如 "byte #$A5_H$" (语法由汇编器决定)。

注：该指令和另一条 XC800 专用指令 MOVC @(DPTR++),A 共用操作码，只有 EO.TRAP_EN = 1 时选择 TRAP 指令。

累加器和字节变量内容互换

XCH A, <byte>

表 4-54 XCH

描述：	XCH 将指定变量的内容装入累加器中，同时将累加器的内容装入指定变量中。源操作数/目的操作数可以是寄存器、直接或寄存器间接寻址。
例子：	R0 中存放地址 20_H，累加器中存放 $3F_H$（00111111_B），内部 RAM 地址单元 20_H 中存放 75_H（01110101_B）。执行指令 XCH A,@Ri RAM 的地址单元 20_H 中将存放 $3F_H$（00111111_B），累加器中存放 75_H（01110101_B）。
指令：	
XCH A, Rn	操作：(A) ↔ (Rn) 字节数：1 周期数：1 编码： 1 1 0 0 | 1 r r r
XCH A, direct	操作：(A) ↔ (direct) 字节数：2 周期数：1 编码： 1 1 0 0 | 0 1 0 1 | 直接地址
XCH A, @Ri	操作：(A) ↔ ((Ri)) 字节数：1 周期数：1 编码： 1 1 0 0 | 0 1 1 i

位互换

XCHD A, @Ri

表 4-55 XCHD

描述：	XCHD 将累加器中的低四位（位 3-0，通常代表十六进制或 BCD）和由特定寄存器间接寻址的内部 RAM 的低四位互换。该指令不影响高四位（位 7-4）以及标志位。
例子：	R0 中存放地址 20_H，累加器中存放 36_H（00110110_B），内部 RAM 地址单元 20_H 中存放 75_H（01110101_B）。执行指令 XCHD A,@Ri RAM 的地址单元 20_H 中将存放 76_H（01110110_B），累加器中存放 35_H。
指令：	
XCHD A, @Ri	操作：(A3-0) ↔ ((Ri)3-0) 字节数：1 周期数：1 编码： 1 1 0 1 | 0 1 1 i

字节变量的逻辑“异或”

XRL <dest-byte>, <src-byte>

表 4-56 XRL

描述：	XRL 执行指定字节变量的按位“异或”操作，结果保存在目的地址中。该指令不影响标志位（<dest-byte> = A 时，该指令将影响奇偶检验标志 P）。 两个操作数支持六种寻址模式组合。目的操作数为累加器时，源操作数可寄存器、直接、寄存器间接寻址，或立即寻址；目的操作数为直接寻址时，源操作数可为累加器或立即数。 *注：用该指令修改输出端口值时，端口数据的原始值从输出锁存器中读取，而不从输出口的引脚读取。*
例子：	累加器中存放 $0C3_H$（11000011_B），寄存器 R0 中存放 $0AA_H$（10101010_B），执行指令 XRL A, R0 A 中内容为 69_H（01101001_B）。 目的操作数为直接寻址字节时，该指令将对 SFR 或 RAM 任意地址单元的位组合取反。决定取反位的屏蔽字节可以是指令中指定的常量，或是运行过程中累加器计算得到的变量。执行指令 XRL P1, # 00110001B 输出端口 P1 的位 5、4 和 0 取反。
指令：	
XRL A, Rn	操作：(A) ← (A) ∀ (Rn) 字节数：1 周期数：1 编码： 0 1 1 0 1 r r r

XRL A, direct	操作：(A)← (A) ∀ (direct) 字节数：2 周期数：1 编码： 0 1 1 0 0 1 0 1 | 直接地址
XRL A, @Ri	操作：(A) ← (A) ∀ ((Ri)) 字节数：1 周期数：1 编码： 0 1 1 0 0 1 1 i
XRL A, #data	操作：(A)← (A) ∀ #data 字节数：2 周期数：1 编码： 0 1 1 0 0 1 0 0 | 立即数
XRL direct, A	操作：(direct)← (direct) ∀ (A) 字节数：2 周期数：1 编码： 0 1 1 0 0 0 1 0 | 直接地址
XRL direct, #data	操作：(direct)← (direct) ∀#data 字节数：3 周期数：2 编码： 0 1 1 0 0 0 1 1 | 直接地址 | 立即数

5 索引

5.1 关键词索引

本节列出用户手册中出现的关键词，以便用户查阅与 XC800 的架构、功能单元或相关功能。

K

M

P

S

T

英飞凌科技中国总部地址及联系方式

英飞凌科技（中国）有限公司
地址：张江高科技园区，松涛路647弄，7-8号
邮编：201203
电话：+86-21-61019000
传真：+86-21-50806204
主页：www.infineon.com/cn

www.infineon.com

英飞凌科技出版