LS2051 是我公司设计、生产的高性能 8 位单片机。其指令系统与 MCS-51 兼容，内部功能、引脚功能、引脚排列以及引脚的电气特性与 AT89C2051 兼容，可直接替换 AT89C2051 以及与其兼容的芯片。

LS2051 支持独立或关联的两道程序同时运行。执行第 1 道程序的性能是 AT89C2051 的 1.27 倍，第 1 道和第 2 道程序同时运行时的处理能力最高可达到 AT89C2051 的 2.55 倍。

主要性能指标

- 指令兼容 MCS-51，引脚兼容 AT89C2051。
- 2KB 内部 Flash 程序存储器，20000 次擦写周期。
- 3.0~5.5V 工作电压。
- 0Hz~24MHz 工作频率。
- 两级程序存储器加密机制。
- 128×8B 内部 SRAM。
- 15 个可编程 I/O 端口，20mA 吸入电流，可直接驱动 LED。
- 6 个中断源。
- 2 个 16 位定时器/计数器。
- 1 个可编程 UART。
- SPI 编程接口。
- 1 个内置高精度模拟比较器。
- 低功耗设计，且支持空闲和掉电模式。

功能和特点

LS2051 片内包含 2K 字节程序存储器（Flash），128 字节数据存储器（SRAM），2 个 16 位定时/计数器，1 个 5 向量两级中断机制，1 套两道程序核处理引擎，15 个 I/O 端口驱动器，1 个全双工串行通信端口，1 个精密模拟比较器，振荡器及时钟电路。

LS2051 的工作频率为 0Hz~24MHz，并支持空闲和掉电两种软件可设置的节电工作方式。空闲方式停止 CPU 的工作，但维持 SRAM、定时器/计数器、串行通信口及中断机制继续工作；掉电方式停止振荡器工作，切断片内所有时钟，SRAM 中的内容维持不变。

LS2051 实现了单核双任务并行处理，可以只运行单道程序，也可同时执行关联或非关联的两道程序。执行第 1 道程序的性能是兼容芯片的 1.27 倍，同时执行两道程序的处理能力最高可达到兼容芯片的 2.55 倍。

引脚说明

LS2051 的引脚排列及其复用功能如图 1 所示。

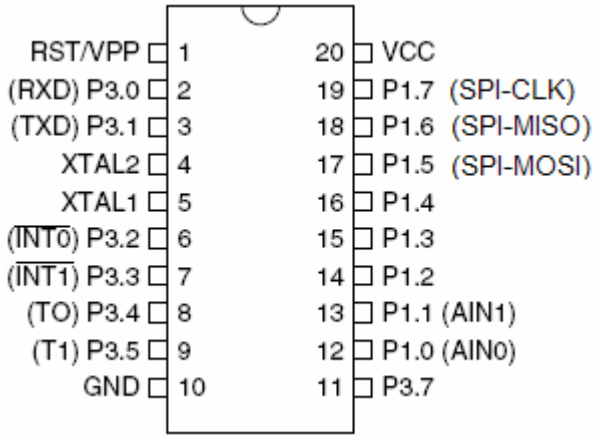


图 1 LS2051 引脚

- VCC：工作电压。
- GND：地。
- P1 口：一组 8 位可编程双向I/O口，P1.2～P1.7 内部设置上拉电阻，P1.0 和P1.1 是内部精密比较器的同相输入（AIN0）和反相输入（AIN1），内部无上拉电阻，如果需要，用户可在外部电路做上拉处理。P1 口输出缓冲器可吸收 20mA电流，可直接驱动LED。当P1 口引脚被写“1”时，可以作为输入；当P1 口引脚被写“0”时，输出为“0”；当芯片被复位后，P1 口为输入。当引脚P1.2～P1.7 用作输入并被外部拉低时，它们将因内部的上拉电阻而产生源电流（I_{IL}）。在编程时，P1.5、P1.6 和 P1.7 分别对应编程接口信号SPI-MOSI、SPI-MISO和SPI-CLK。当用户需要I/O端口速率大于 40KHz时，建议外部加 4.7K欧的上拉电阻。
- P3 口： P3.0～P3.5、P3.7 是带有内部上拉电阻 7 个可编程双向I/O口，P3.6 用于片内模拟比较器的输出，未引出片外。P3 口输出缓冲器可吸收 20mA电流，可直接驱动LED。当P3 口引脚被写“1”时，可作为输入；当P3 口引脚被写“0”时，输出为“0”；当芯片被复位后，P3 口为输入。作为输入时，被外部拉低的P3 口将因内部的上拉电阻而产生源电流（I_{IL}）。P3 口的引脚复用功能如表 1 所示。当用户需要I/O端口速率大于 40KHz时，建议外部加 4.7K欧的上拉电阻。

表 1 P3 口的特殊功能

引脚	功能特性
P3.0	RXD（串口输入）
P3.1	TXD（串口输出）
P3.2	$\overline{INT0}$（外部中断 0）
P3.3	$\overline{INT1}$（外部中断 1）
P3.4	T0 （定时器/计数器 0 输入）
P3.5	T1 （定时器/计数器 1 输入）

- RST：复位输入。高电平有效，芯片加电或掉电模式后，高电平的复位信号必须维持 100us 以上，芯片方可正常工作；其他情况下，持续两个机器周期以上的高电平便可完成复位，每个机器周期等于 12 个振荡器时钟周期。
- XTAL1：振荡器反相放大器及内部时钟发生器的输入端。
- XTAL2：振荡器反相放大器的输出端。

时钟特性

时钟振荡器可采用石英晶体或陶瓷谐振器，如图 2 所示。XTAL1 为用户提供外部时钟的输入，当使用外部时钟时，XTAL2 悬空，如图 3 所示。LS2051 内置一个分频器，完成对内部振荡器输出时钟或来自引脚输入时钟的 2 分频与整形。因此，对外部输入时钟占空比无特殊要求，但必须符合时钟信号的直流和交流规范。

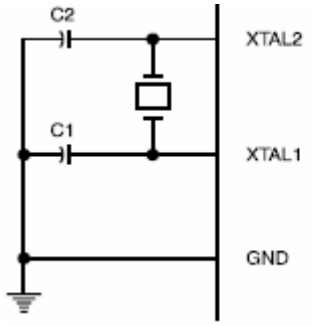


图 2：使用内部振荡器的时钟电路

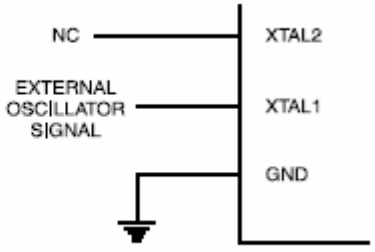


图 3：使用外部时钟的连接

采用石英晶体时： C1,C2=30pF±10pF。

采用陶瓷谐振器时：C1,C2=40pF±10pF。

特殊功能寄存器

片内特殊功能寄存器(SFR)空间存储映像如表 2 所示。其中，空白字节在这个版本中没有定义，读返回为不定值。

表 2 SFR 存储映像和复位值

Addr	B0	B1	B2	B3	B4	B5	B6	B7	Addr
0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H									0CFH
0C0H									0C7H
0B8H	IP XXX00000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 0XX00000								0AFH
0A0H									0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 000000000	TL1 00000000	TH0 00000000	TH1 00000000			8FH
80H		SP 00000111	DPL 00000000	DPH 00000000				PCON 0XXX0000	87H

注：特殊寄存器 IE 的 d7 位为第一道中断总使能，d6 位为第二道的中断总使能；

一些指令的约束

LS2051 的指令系统与 MCS-51 兼容，但和 AT89C2051 一样，在 LS2051 的程序设计时应注意下列两个约束条件。

1、与 MOVX 相关的指令，数据存储器

LS2051 内部数据存储器为 128 字节，其堆栈的深度应在 128 字节范围内，由于 LS2051 不支持外部数据存储器和外部程序存储器的配置。因此程序中不应有 MOVX[……]指令。

典型的 8051 汇编器即使在违反上述约束仍对指令进行汇编，用户应了解 LS2051 微处理器的存储器物理空间和约束条件，正确地调整所使用指令的寻址范围。

2、分支指令

LCALL、LJMP、ACALL、AJMP、SJMP 和 JMP @A+DPTR 无条件分支指令的目的地址必须落在 000H~7FF 之间才能正确执行。否则，会导致程序运行错误。

CJNE [...]、DJNZ [...]、JB, JNB、JC、JNC、JBC、JZ 和 JNZ 条件分支指令的使用约束和无条件分支指令相同。

使用注意事项

- 芯片加电或芯片进入掉电模式后，复位时间必须大于 100us。
- 在某中断服务被挂起期间，不会再次响应该中断。
- 若空闲状态发生在某中断服务期间，则该状态不能被此中断唤醒。
- 两道程序处理引擎分别设置独立的 ACC、B、DPTR、PSW 和 SP 特殊寄存器，但第 1 道所属的通用寄存器为 32 个，而第 2 道所属的通用寄存器为 8 个。因此，在第 2 道程序的设计中要考虑不能过多的压栈。
- LS2051 第 1 道应用程序的设计和 AT89C2051 完全一致，如果用户不使用第 2 道程序，则 LS2051 就是 AT89C2051。
- 由于 LS2051 特有的第 2 道处理引擎，使芯片的中断处理比 AT89C2051 更加灵活和高效。若第 1 道的中断赋能打开，而第 2 道的中断赋能未打开，那么，中断由第 1 道处理；若第 1 道和第 2 道中断赋能都打开，且第 2 道应用程序未启动或没有第 2 道应用程序，那么，两道都可以执行中断处理，这由硬件自动完成，不需用户考虑；若第 1 道中断赋能关闭，而第 2 道中断赋能打开，且第 2 道应用程序未启动或没有第 2 道应用程序，那么，由第 2 道处理引擎执行中断处理，这由硬件自动完成，不需用户考虑。
- 第 2 道程序一旦被启动，则不能被任何事件打断。

- LS2051 的中断处理是在指令执行前切入，而 89C2051 是指令执行后切入，也就是说 LS2051 的中断返回后还是执行切入点的指令，而 89C2051 的中断返回后执行的是切入点下一条指令，这在串口应用程序设计时要特别留意。
- 若使用两道程序，要考虑使用公共资源时可能发生的冲突问题。
- LS2051 串口方式 0 的波特率为 Fosc/24；串口方式 2 的波特率为 Fosc/64 或 Fosc/128。
- P1 和 P3 口在大于 40KHz 的应用时，需在外部接 4.7K 左右的上拉电阻。

程序存储器的加密

LS2051 提供对片上的两个加密位（LB1、LB2）进行编程（P）或不编程（U）来实现用户程序的加密，如表 3 所示，加密位只能用擦除操作置无效。

表 3 LS2051 的程序加密

LB1	LB2	功能
U	U	程序不加密
P	U	禁止 Flash 再编程
P	P	禁止 Flash 再编程，同时禁止校验

工作模式

1、空闲模式

空闲模式由软件设置进入，可由任何被允许的中断请求或硬件复位终止。进入空闲模式后，CPU 停止工作，而 SRAM、定时器/计数器、串行通信口及中断机制继续工作。P1.0 和 P1.1 在不使用外部上拉电阻的情况下应设置为“0”，或者在使用上拉电阻的情况下设置为“1”，可以进一步降低功耗。

应该注意的是，在用硬件复位终止空闲模式时，LS2051 通常从程序停止一直到内部复位获得控制之前的两个机器周期处恢复程序执行。在这种情况下，片内硬件禁止对内部 SRAM 的读写，但允许对端口的访问，要消除硬件复位终止空闲模式对端口意外写入的可能，原则上进入空闲模式指令的下一条指令不应对端口引脚进行访问。

2、掉电模式

掉电模式由软件设置进入，只能由硬件复位终止。进入掉电模式后，振荡器停止工作，掉电模式的设置指令是最后一条被执行的指令，片内 SRAM 和特殊功能寄存器的内容在掉电模式终止前被冻结，复位后将重新定义全部特殊功能寄存器但不改变 SRAM 中的内容，在 VCC 恢复到正常工作电压前，复位无效，且必须保持足够的时间以使振荡器重新启动并稳定工作。P1.0 和 P1.1 在不使用外部上拉电阻的情况下应设置为“0”，或者在使用上拉电阻的情况下设置为“1”，可以进一步降低功耗。

Flash 编程

LS2051 采用 SPI 协议实现对程序存储器（Flash）的擦除和写入，启芯公司为您提供专用程序下载软件和通用下载电缆。

为了增强程序的安全性，SPI 接口不提供对 LS2051 内部 Flash 的读操作。

SPI 编程协议

LS2051 的 SPI 接口包括 RST、SCK（P1.7）、MOSI（P1.5）和 MISO（P1.6）四个信号。如图 5 所示。

复位输入 RST 为“1”时表示芯片处于可编程状态，此时，通过 SCK、MOSI 可向 LS2051 内部发出编程命令和程序执行代码，MISO 为编程错误指示。当复位信号为“0”时，芯片进入正常工作状态。

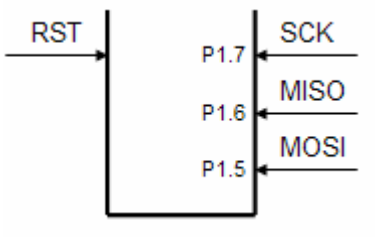


图 5 Flash 编程接口信号

SPI 采用同步串行方式传输字节数据，时序如图 6 所示。

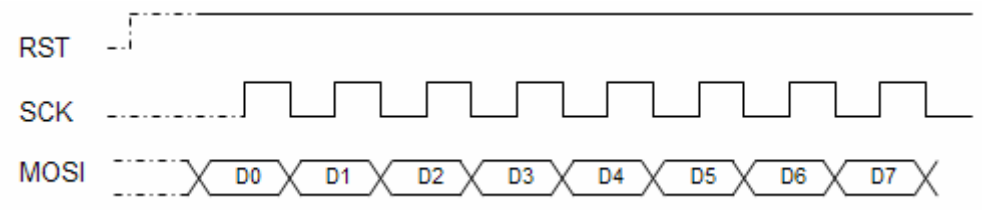


图 6 SPI 接口时序

芯片处于编程状态时，MOSI 承载的串行数据在 SCK 的上升沿被采样，每连续的八位确认为一个字节，数据的第 7 位先行传输。

MISO 用于校验输出，每接收一个字节的数据，芯片都进行校验。当对一个地址的写入失败时，MISO 输出“1”，直到下一个写入校验正确，MISO 输出“0”。

编程命令

- 写 Flash：命令码序列为 AA-50-AX-AY-data。当 SPI 接口连续接收这 5 个字节时，芯片进入 Flash 写操作时序，其中第一个字节 AA 和第二个字节 50 为命令码，第三个字节是 Flash 的高六位地址，第四个字节是 Flash 的低六位地址，第五个字节为要写入的数据。
- 擦除 Flash：命令码序列 AA-8A 或 AA-E4。LS2051 允许一次擦除整个 2KB Flash 空间，当 SPI 接口连续收到这两个字节时，芯片进入 Flash 擦除操作时序，其中第一个字节为 AA，第二个字节为 8A 或 E4。
- 为了增强加密性能，SPI 不支持对 Flash 的操作。

Flash 的擦除时序

当 SPI 收到 AA 和 8A 两个命令字时，芯片进入 Flash 擦除时序，时序如图 7 所示，命令字的两个字节必须连续发送，中间不能插入其它信息。擦除时间不得向 SPI 接口发送其它命令字，否则，将导致擦除失败。

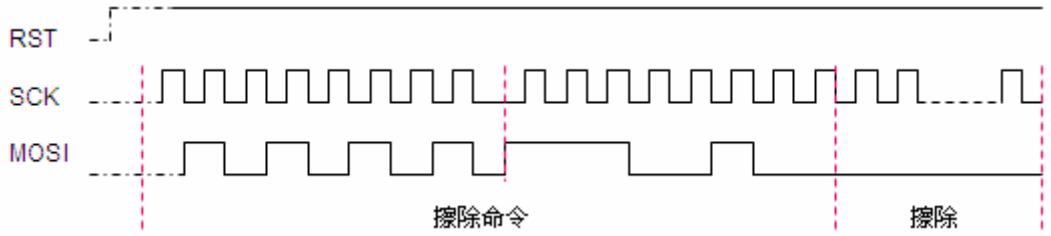


图 7： Flash 擦除时序

Flash 的写时序

当 SPI 收到 AA-50-AX-AY-data 5 个字节时，芯片进入 Flash 写时序，命令字必须连续发送，中间不能插入其它信息，时序如图 8 所示。在写期间不得向 SPI 接口发送其它命令字，否则写操作将失败。在每写完一个字节后，内部将自动校验。若正确，则 MOSI 输出 0，否则，MOSI 输出 1。某单元的写入失败不会影响后续单元的写操作，但对此单元的再次写则不会成功。

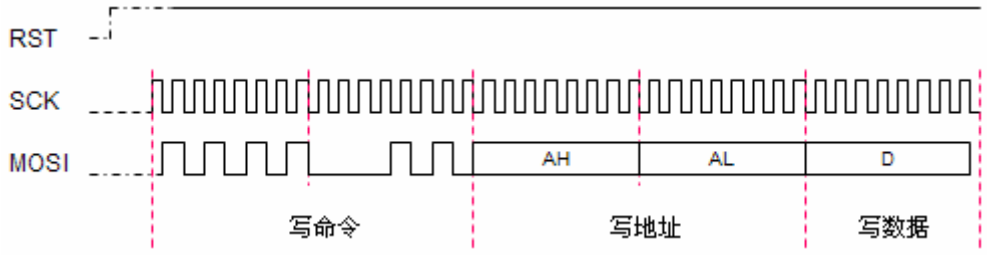


图 8：Flash 写时序

极限参数

极限参数如表 4。

表 4 LS2051 极限

工作温度	-40℃~ +85℃
储藏温度	-60℃~ +125℃
引脚对地电压	-1.0V~ +7.0V
V_{CC}	6.6V
I_{CC}	25.0mA

直流特性

直流特性如表 5。

表 5 LS2051 的直流参数

Sym	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage		-0.5	0.2 V_{CC} - 0.1	V
V_{IH}	Input High-voltage	(Except XTAL1, RST)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Input High-voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Output Low-voltage	I_{OL} = 20 mA, V_{CC} = 5V		0.5	V
V_{OH}	Output High-voltage	I_{OH} = -80 μA, V_{CC} = 5V ± 10%	2.4		V
I_{IL}	Logical 0 Input Current	V_{IN} = 0.45V		-50	μA
I_{TL}	Logical 1 to 0 Transition Current	V_{IN} = 2V, V_{CC} = 5V ± 10%		-750	μA
I_{LI}	Input Leakage Current	0 < V_{IN} < V_{CC}		±10	μA
V_{OS}	Comparator Input Offset Voltage	V_{CC} = 5V		20	mV
V_{CM}	Comparator Input Common Mode Voltage		0	V_{CC}	V
RRST	Reset Pull-down Resistor		30	70	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, TA = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz, V_{CC} =5V		20	mA

说明：

在稳态下（非瞬态），I_{OL}受如下限制：

- 每个端口I_{OL}最大允许值为 20mA。
- 所有输出端口I_{OL}之和的最大值为 80mA。
- 若I_{OL}超过了测试条件所允许的值，则V_{OL}也可能相应超过测试条件所允许值，但不能确保端口值会大于如表所述的测试条件获得值。

指令集

符号定义表

为了方便 LS2051 指令的描述，定义符号如表 6。

表 6 符号定义

符号	含义
Rn	R0～R7 寄存器 n=0～7
Direct	直接地址，内部数据区的地址 RAM(00H～7FH)SFR(80H～FFH) B，ACC，PSW，IP，P3，IE，P2，SCON，P1，TCON，P0
@Ri	间接地址 Ri=R0 或 R1 如 51RAM 地址(00H～7FH)
#data	8 位常数
#data16	16 位常数
Addr16	16 位的目标地址
Addr11	11 位的目标地址
Rel	相对地址
bit	内部数据 RAM(20H～2FH)，特殊功能寄存器的直接地址的位

指令

LS2051 的指令分为算术运算、逻辑运算、布尔代数运算、数据转移、程序转移和两道并行相关 6 类，如表 7。

表 7 LS2051 指令

指令	字节	原周期	周期	操作
算术运算				
1、ADD A,Rn	1	1	1	将累加器与寄存器的内容相加，结果存回累加器
2、ADD A,direct	2	1	1	将累加器与直接地址的内容相加，结果存回累加器
3、ADD A,@Ri	1	1	1	将累加器与间接地址的内容相加，结果存回累加器
4、ADD A,#data	2	1	1	将累加器与常数相加，结果存回累加器
5、ADDC A,Rn	1	1	1	将累加器与寄存器的内容及进位 C 相加，结果存回累加器
6、ADDC A,direct	2	1	1	将累加器与直接地址的内容及进位 C 相加，结果存回累加器
7、ADDC A,@Ri	1	1	1	将累加器与间接地址的内容及进位 C 相加，结果存回累加器
8、ADDC A,#data	2	1	1	将累加器与常数及进位 C 相加，结果存回累加器
9、SUBB A,Rn	1	1	1	将累加器的值减去寄存器的值减借位 C，结果存回累加器
10、SUBB A,direct	2	1	1	将累加器的值减直接地址的值减借位 C，结果存回累加器

11、SUBB A,@Ri	1	1	1	将累加器的值减间接地址的值减借位 C，结果存回累加器
12、SUBB A，#data	2	1	1	将累加器的值减常数值减借位 C，结果存回累加器
13、INC A	1	1	1	将累加器的值加 1
14、INC Rn	1	1	1	将寄存器的值加 1
15、INC direct	2	1	1	将直接地址的内容加 1
16、INC @Ri	1	1	1	将间接地址的内容加 1
17、INC DPTR	1	1	1	数据指针寄存器值加 1
说明：将 16 位的 DPTR 加 1，当 DPTR 的低字节(DPL)从 FFH 溢出至 00H 时，会使高字节(DPH)加 1，不影响任何标志位。				
18、DEC A	1	1	1	将累加器的值减 1
19、DEC Rn	1	1	1	将寄存器的值减 1
20、DEC direct	2	1	1	将直接地址的内容减 1
21、DEC @Ri	1	1	1	将间接地址的内容减 1
22、MUL AB	1	4	1	将累加器的值与 B 寄存器的值相乘，乘积的低位字节存回累加器，高位字节存回 B 寄存器
说明：将累加器 A 和寄存器 B 内的无符号整数相乘，产生 16 位的积，低位字节存入 A，高位字节存入 B 寄存器。如果积大于 FFH，则溢出标志位(OV)被设定为 1，而进位标志位为 0。				
23、DIV AB	1	4	1	将累加器的值除以 B 寄存器的值，结果的商存回累加器，余数存回 B 寄存器
说明：无符号的除法运算，将累加器 A 除以 B 寄存器的值，商存入 A，余数存入 B。执行本指令后，进位位(C)及溢出位(OV)被清除为 0。				
24、DA A	1	1	1	将累加器 A 作十进制调整，若(A) 3-0>9 或(AC)=1，则(A) 3-0←(A)3-0+6　若(A) 7-4>9 或 (C)=1，则(A) 7-4←(A)7-4+6
逻辑运算				
25、ANL A,Rn	1	1	1	将累加器的值与寄存器的值做 AND 逻辑运算，结果存回累加器
26、ANL A,direct	2	1	1	累加器值与直接地址的内容做 AND 逻辑运算，结果存回累加器
27、ANL A,@Ri	1	1	1	累加器值与间接地址的内容做 AND 逻辑运算，结果存回累加器
28、ANL A,#data	2	1	1	将累加器的值与常数做 AND 的逻辑判断，结果存回累加器
29、ANL direct,A	2	1	1	直接地址内容与累加器值做 AND 逻辑判断，结果存回直接地址
30、ANL direct,#data	3	2	1	直接地址的内容与常数值做 AND 逻辑判断，结果存回直接地址
31、ORL A,Rn	1	1	1	将累加器的值与寄存器的值做 OR 的逻辑判断，结果存回累加器
32、ORL A,direct	2	1	1	将累加器值与直接地址的内容做 OR 逻辑判断，结果存回累加器
33、ORL A,@Ri	1	1	1	累加器的值与间接地址的内容做 OR 逻辑判断，结果存回累加器
34、ORL A,#data	2	1	1	将累加器的值与常数做 OR 的逻辑判断，结果存回累加器
35、ORL direct,A	2	1	1	直接地址的内容与累加器值做 OR 逻辑判断，结果存回直接地址

36、ORL direct,#data	3	2	1	将直接地址的内容与常数值做 OR 逻辑判断，结果存回直接地址
37、XRL A,Rn	1	1	1	将累加器的值与寄存器的值做 XOR 逻辑判断，结果存回累加器
38、XRL A,direct	2	1	1	累加器值与直接地址的内容做 XOR 逻辑判断，结果存回累加器
39、XRL A,@Ri	1	1	1	累加器值与间接地扯的内容做 XOR 逻辑判断，结果存回累加器
40、XRL A,#data	2	1	1	将累加器的值与常数作 XOR 的逻辑判断，结果存回累加器
41、XRL direct,A	2	1	1	直接地址内容与累加器值做 XOR 逻辑判断，结果存回直接地址
42、XRL direct,#data	3	2	1	直接地址的内容与常数值做 XOR 逻辑判断，结果存回直接地址
43、CLR A	1	1	1	清除累加器的值为 0
44、CPL A	1	1	1	将累加器的值反相
45、RL A	1	1	1	将累加器的值左移一位
46、RLC A	1	1	1	将累加器含进位 C 左移一位
47、RR A	1	1	1	将累加器的值右移一位
48、RRC A	1	1	1	将累加器含进位 C 右移一位
49、SWAP A	1	1	1	将累加器的高 4 位与低 4 位的内容交换。(A)3-0←(A)7-4
数据转移				
50、MOV A,Rn	1	1	1	将寄存器的内容载入累加器
51、MOV A,direct	2	1	1	将直接地址的内容载入累加器
52、MOV A,@Ri	1	1	1	将间接地址的内容载入累加器
53、MOV A,#data	2	1	1	将常数载入累加器
54、MOV Rn，A	1	1	1	将累加器的内容载入寄存器
55、MOV Rn,direct	2	2	1	将直接地址的内容载入寄存器
56、MOV Rn,gdata	2	1	1	将常数载入寄存器
57、MOV direct,A	2	1	1	将累加器的内容存入直接地址
58、MOV direct,Rn	2	2	1	将寄存器的内容存入直接地址
59、MOV direct1, direct2	3	2	1	将直接地址 2 的内容存入直接地址 1
60、MOV direct,@Ri	2	2	1	将间接地址的内容存入直接地址
61、MOV direct,#data	3	2	1	将常数存入直接地址
62、MOV @Ri,A	1	1	1	将累加器的内容存入某间接地址
63、MOV @Ri,direct	2	2	1	将直接地址的内容存入某间接地址

64、MOV @Ri,#data	2	1	1	将常数存入某间接地址
65、MOV DPTR,#data16	3	2	1	将 16 位的常数存入数据指针寄存器
66、MOVC A,@A+DPTR	1	2	2	(A) ←((A)+(DPTR)) 累加器的值再加数据指针寄存器的值为其所指定地址，将该地址的内容读入累加器
67、MOVC A,@A+PC	1	2	2	(PC)←(PC)+1；(A)←((A)+(PC))累加器的值加程序计数器的值作为其所指定地址，将该地址的内容读入累加器
68、PUSH direct	2	2	1	将直接地址的内容压入堆栈区
69、POP direct	2	2	1	从堆栈弹出该直接地址的内容
70、XCH A,Rn	1	1	1	将累加器的内容与寄存器的内容互换
71、XCH A,direct	2	1	1	将累加器的值与直接地址的内容互换
72、XCH A,@Ri	1	1	1	将累加器的值与间接地址的内容互换
73、XCHD A,@Ri	1	1	1	将累加器的低 4 位与间接地址的低 4 位互换
布尔代数运算				
74、CLR C	1	1	1	清除进位 C 为 0
75、CLR bit	2	1	1	清除直接地址的某位为 0
76、SETB C	1	1	1	设定进位 C 为 1
77、SETB bit	2	1	1	设定直接地址的某位为 1
78、CPL C	1	1	1	将进位 C 的值反相
79、CPL bit	2	1	1	将直接地址的某位值反相
80、ANL C,bit	2	2	1	将进位 C 与直接地址的某位做 AND 逻辑判断，结果存回进位 C
81、ANL C,/bit	2	2	1	进位 C 与直接地址的某位反相值做 AND 逻辑判断，结果存回 C
82、ORL C,bit	2	2	1	将进位 C 与直接地址的某位做 OR 的逻辑判断，结果存回进位 C
83、ORL C,/bit	2	2	1	进位 C 与直接地址的某位反相值做 OR 逻辑判断，结果存回 C
84、MOV C,bit	2	1	1	将直接地址的某位值存入进位 C
85、MOV bit,C	2	2	1	将进位 C 的值存入直接地址的某位
86、JC rel	2	2	1	若进位 C=1 则跳至 rel 的相关地址
87、JNC rel	2	2	1	若进位 C=0 则跳至 rel 的相关地址
88、JB bit,rel	3	2	1	若直接地址的某位为 1，则跳至 rel 的相关地址
89、JNB bit,rel	3	2	1	若直接地址的某位为 0，则跳至 rel 的相关地址
90、JBC bit,rel	3	2	1	直接地址的某位为 1，则跳至 rel 相关地址，并将该位值清为 0
程序转移				
91、ACALL addr11	2	2	1	调用 2K 程序存储器范围内的子程序
92、LCALL addr16	3	2	1	调用 64K 程序存储器范围内的子程序

93、RET	1	2	1	从子程序返回
94、RETI	1	2	1	从中断子程序返回，在第 2 道程序中代表 2 道程序结束指令
95、AJMP addr11	2	2	1	绝对跳跃(2K 内)
96、LJMP addr16	3	2	1	长跳跃(64K 内)
97、SJMP rel	2	2	1	短跳跃(2K 内)-128～+127 字节
98、JMP @A+DPTR	1	2	1	跳至累加器的内容加数据指针所指的相关地址
99、JZ rel	2	2	1	累加器的内容为 0，则跳至 rel 所指相关地址
100、JNZ rel	2	2	1	累加器的内容不为 0，则跳至 rel 所指相关地址
101、CJNE A,direct,rel	3	2	1	累加器内容与直接地址内容比较，不相等跳至 rel 所指相关地址
102、CJNE A,#data,rel	3	2	1	累加器的内容与常数比较，若不相等则跳至 rel 所指的相关地址
103、CJNE Rn,#data,rel	3	2	1	寄存器的内容与常数比较，若不相等则跳至 rel 所指的相关地址
104、CJNE @Ri,#data,rel	3	2	1	间接地址的内容与常数比较，若不相等则跳至 rel 所指相关地址
105、DJNZ Rn,rel	2	2	1	将寄存器的内容减 1，不等于 0 则跳至 rel 所指的相关地址
106、DJNZ direct,rel	3	2	1	将直接地址的内容减 1，不等于 0 则跳至 rel 所指的相关地址
107、NOP	1	1	1	无动作

与并发程序相关指令（并发程序的程序道号为 1）		
MOV 0FEH，#data		并发程序结束。
MOV 0FCH，# addr8		若同步标志为 0，则置同步标志为 1，PC=PC+1；否则 PC= addr8。
MOV 0FBH，# addr8		若同步标志为 1，则清同步标志为 0，PC=PC+1；否则 PC= addr8。
MOV 0FFH，# addr8		启动并发程序，addr8 为并发程序的入口地址。
MOV 0FDH,#1/#0		若为#1 表示同步标志置 1，若为#0 表示同步标志清 0。

应用举例

两道程序并发运行编程

1、第 1 道与第 2 道同步运行测试程序：两道同时做算法运算，再将彼此运算的结果在第 1 道程序中相加，通过 P1 口输出，结果为： 07H,38H,39H,07H,07H,04H,0BH。

程序如下：

```
MAIN:                    ；主程序在第 1 道工作
    MOV 0FFH,#ROAD1      ；打开第 1 道程序
    MOV A,#2             ；A=2
    ADD A,#5             ；A=7
    INC A                ；A=8
    SUBB A,#4            ；A=4
    MOV R0,A
    MOV  0FBH,#$         ；判断同步标志位是否为 1，如为 1 则往下执行，如为 0 则总是执行本条语句
    MOV A,12H
    MOV P1,A             ；A=7
    MOV A,R0
    MOV P1,A             ；从 P1 口输出，A=4
    ADD A,12H            ；A=0BH
    MOV P1,A             ；从 P1 口输出
    AJMP  $              ；第 1 道程序在此循环

ROAD1:                   ；第 2 道程序入口地址
    MOV A,#7             ；A=7
    MOV P1,A             ；从 P1 口输出
    MOV B,#8
    MUL AB               ；B=0 A=56
    MOV P1,A
    INC A                ；A=57
    MOV P1,A
    SUBB A,#50           ；A=7
    MOV P1,A
```

```
MOV 12H,A
MOV 0FDH,#1          ；同步标志位置为 1
RETI                 ;第 2 道结束指令,也可以用 MOV 0FEH,#立即数，代替 RETI
```

指令。

2、两首歌曲由两个道程序同时播放。第 1 道运行 c 语言编写的音乐程序（八月桂花香），通过 P1.0 输出；第 2 道运行汇编语言编写的音乐程序（生日快乐），通过 P1.1 输出。

包含的程序名称：music_c_asm.c, road1.a51, loadp.a51

八月桂花香 music_c_asm.c

```
extern void loadp(void)；
#include <reg2051.h>
#include <intrins.h>
//本例采用的晶振为 11.0592MHZ
unsigned char n=0;                          //n 为节拍常数变量
unsigned char temp_th1=0,temp_tl1=0;
unsigned char code music_tab[] =
{                                           //格式为： 频率常数, 节拍常数, 频率常数, 节
```

拍常数。

```
0x18, 0x30, 0x1C , 0x10,   0x20, 0x40, 0x1C , 0x10,
0x18, 0x10, 0x20 , 0x10,   0x1C, 0x10, 0x18 , 0x40,
0x1C, 0x20, 0x20 , 0x20,   0x1C, 0x20, 0x18 , 0x20,
0x20, 0x80, 0xFF , 0x20,   0x30, 0x1C, 0x10 , 0x18,
0x20, 0x15, 0x20 , 0x1C,   0x20, 0x20, 0x20 , 0x26,
0x40, 0x20, 0x20 , 0x2B,   0x20, 0x26, 0x20 , 0x20,
0x20, 0x30, 0x80 , 0xFF,   0x20, 0x20, 0x1C , 0x10,
0x18, 0x10, 0x20 , 0x20,   0x26, 0x20, 0x2B , 0x20,
0x30, 0x20, 0x2B , 0x40,   0x20, 0x20, 0x1C , 0x10,
0x18, 0x10, 0x20 , 0x20,   0x26, 0x20, 0x2B , 0x20,
0x30, 0x20, 0x2B , 0x40,   0x20, 0x30, 0x1C , 0x10,
0x18, 0x20, 0x15 , 0x20,   0x1C, 0x20, 0x20 , 0x20,
0x26, 0x40, 0x20 , 0x20,   0x2B, 0x20, 0x26 , 0x20,
0x20, 0x20, 0x30 , 0x80,   0x20, 0x30, 0x1C , 0x10,
0x20, 0x10, 0x1C , 0x10,   0x20, 0x20, 0x26 , 0x20,
0x2B, 0x20, 0x30 , 0x20,   0x2B, 0x40, 0x20 , 0x15,
```

```
    0x1F, 0x05, 0x20 , 0x10,    0x1C, 0x10, 0x20 , 0x20,
    0x26, 0x20, 0x2B , 0x20,    0x30, 0x20, 0x2B , 0x40,
    0x20, 0x30, 0x1C , 0x10,    0x18, 0x20, 0x15 , 0x20,
    0x1C, 0x20, 0x20 , 0x20,    0x26, 0x40, 0x20 , 0x20,
    0x2B, 0x20, 0x26 , 0x20,    0x20, 0x20, 0x30 , 0x30,
    0x20, 0x30, 0x1C , 0x10,    0x18, 0x40, 0x1C , 0x20,
    0x20, 0x20, 0x26 , 0x40,    0x13, 0x60, 0x18 , 0x20,
    0x15, 0x40, 0x13 , 0x40,    0x18, 0x80, 0x00
  };
void int0() interrupt 1                    //采用中断 0  控制节拍
  {
    TH0=0xd8;
    TL0=0xef;
    n--;
  }
void int1() interrupt 3                    //采用中断 1  控制第 2 道中的 music
  {
    TL1=temp_tl1;
    TH1=temp_th1;
    P1_1=~P1_1;
  }
void delay (unsigned char m)               //控制频率延时
  {
    unsigned i=3*m;
    while(--i);
  }
void delayms(unsigned char a)              //豪秒延时子程序
  {
    while(--a);                            //采用 while(--a)  不要采用 while(a--);
  }
void main()
  {
    unsigned char p,m;                     //m 为频率常数变量
    unsigned char i=0;
    TMOD&=0xff;
```

```
TMOD|=0x11;
TH0=0xd8;
TL0=0xef;
IE=0x8a;
loadp();                                    //load 第 2 道程序
play:
while(1)
   {
    a:
          p=music_tab[i];
      if(p==0x00)
       {
          i=0;
          delayms(1000);
          goto play;
        }                                   //如果碰到结束符，延时 1 秒，回到开始再来一遍。
      else if(p==0xff)
        {
          i=i+1;
          delayms(100);
          TR0=0;
          goto a;
        }                                   //若碰到休止符，延时 100ms，继续取下一音符。
       else
        {
          m=music_tab[i++];
          n=music_tab[i++];
        }                                   //取频率常数和节拍常数。
       TR0=1; //开定时器 0
       while(n!=0)
         {
          P1_0=~P1_0;
          delay(m);
         }                                  //等待节拍完成,，通过 P1 口输出音频
       TR0=0;                               //关定时器 0
```

```
        }
    }
```

第 2 道程序的加载程序 loadp.a51

```
  extrn code(road1)
  NAME LOADP
  LOADP1   SEGMENT   CODE
           PUBLIC     loadp
           RSEG       LOADP1
  loadp:
           USING  0
           MOV    0FFH,#ROAD1          ;启动第 2 道，ROAD1 在第 2 道中执行。
           RET
           END
```

生日快乐 road1.a51

```
  extrn data(temp_th1)
  extrn data(temp_tl1)

  NAME      ROAD1
  ROAD11   SEGMENT   CODE
           PUBLIC     ROAD1
           RSEG       ROAD11
  ROAD1:
           USING      0
  start0:
       mov 60h,#00h                    ;取简谱码指针
       next: mov a,60h                 ;简谱码指针载入 A
       mov dptr,#table                 ;至 table 取简谱码
       movc a,@a+dptr
       mov r2,a                        ;取到的简谱码暂存于 R2
       jz end1                         ;是否取到 00(结束码)?
       anl a,#0fh                      ;不是,则取低 4 位(节拍码)
       mov r5,a                        ;将节拍码存入 R5
```

```
mov a,r2                    ;将取到的简谱码再载入 A

swap a                      ;高低四位交换
anl a,#0fh                  ;取低四位(音符码)
jnz sing                    ;取到的音符码是否为零?
clr tr1                     ;是,则不发音
jmp d1
sing: dec a                 ;取到的音符码减 1(不含 0).
mov 63h,a                   ;存入(22H).
rl a                        ;乘 2
mov dptr,#table1            ;至 table1 取相对的高位字节计数值
movc a,@a+dptr
mov th1,a                   ;取到的高位字节存入 TH1
mov temp_th1,a              ;取到的高位字节存入(21H)
mov a,63h                   ;再载入取的的音符码
rl a                        ;乘 2
inc a                       ;加 1
movc a,@a+dptr              ;至 table1 取相对的低位字节计数值
mov tl1,a                   ;取到的高位字节存入 TL1
mov temp_tl1,a              ;取到的高位字节存入(20H)
setb tr1                    ;启动 timer0
d1: call delay              ;基本单位时间 1/4 拍 187 毫秒
inc 60h                     ;取简谱码指针加 1
jmp next                    ;取下一个码
end1: clr tr1               ;停止 timer1
jmp start0                  ;重复循环?
delay: mov r7,#02h          ;187 毫秒
d2: mov r4,#187
d3: mov r3,#248
djnz r3,$
djnz r4,d3
djnz r7,d2
```

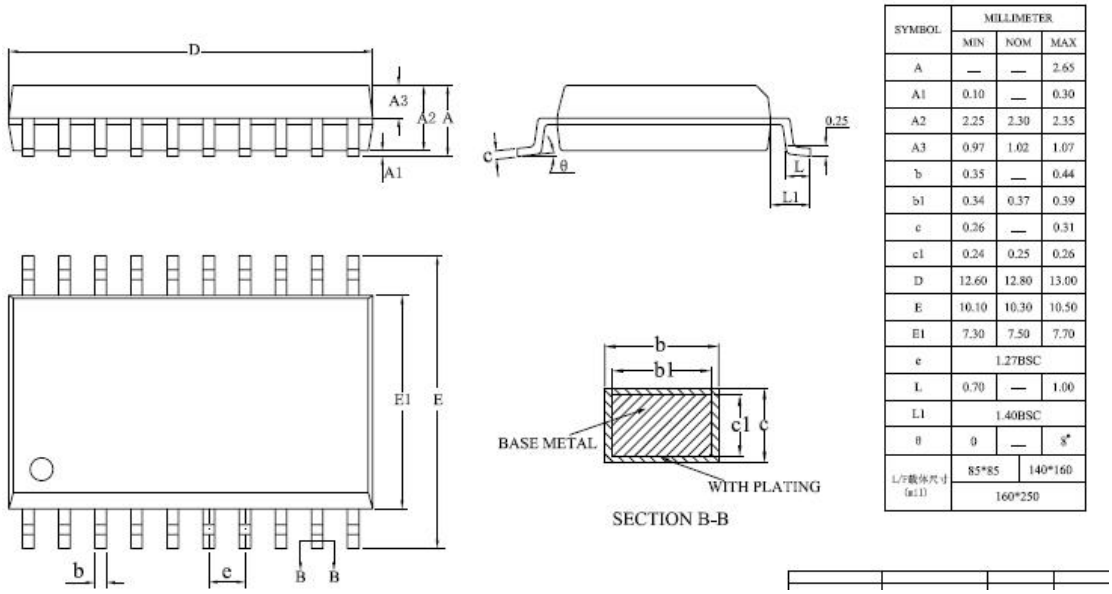
```
        djnz r5,delay                                   ;决定节拍
        ret

table1:
        dw 64260,64400,64524,64580
        dw 64684,64777,64820,64898
        dw 64968,65030,65058,65110
        dw 65157,65178,65217
table:
        ;1
        db 82h,01h,81h,94h,84h,0b4h,0a4h,04h,82h,01h,81h,94h,84h,0c4h,0b4h,04h
        ;2
        db 82h,01h,81h,0f4h,0d4h,0b4h,0a4h,94h,0e2h,01h,0e1h,0d4h,0b4h,0c4h,0b4h,04h
        ;3
        db 82h,01h,81h,94h,84h,0b4h,0a4h,04h,82h,01h,81h,94h,84h,0c4h,0b4h,04h
        ;4
        db      82h,01h,81h,0f4h,0d4h,0b4h,0a4h,94h,0e2h,01h,0e1h,0d4h,0b4h,0c4h,0b4h,04h,00h

RET
END
```

产品序号

工作频率	电源电压	产品序号	封装	工作温度范围
0~24MHz	3.0V~5.5V	LS2051-224SJI	20S	-40℃~85℃
0~24MHz	3.0V~5.5V	LS2051-224PJI	20P3	-40℃~85℃

封装信息

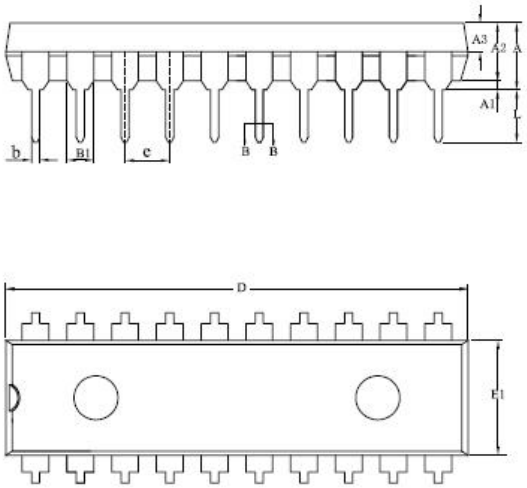
LS2051 采用 20 引脚， SOIC 封装，如图 9 所示。



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	—	—	2.65
A1	0.10	—	0.30
A2	2.25	2.30	2.35
A3	0.97	1.02	1.07
b	0.35	—	0.44
b1	0.34	0.37	0.39
c	0.26	—	0.31
c1	0.24	0.25	0.26
D	12.60	12.80	13.00
E	10.10	10.30	10.50
E1	7.30	7.50	7.70
e	1.27BSC		
L	0.70	—	1.00
L1	1.40BSC		
θ	0	—	8°
上/下散热片尺寸 (单位)	85*85		140*160
	160*250		

图 9 LS2051 封装（SOIC）

LS2051 采用 20 引脚， DIP 封装，如图 10 所示。



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	3.60	3.80	4.00
A1	0.51	—	—
A2	3.20	3.30	3.40
A3	1.47	1.52	1.57
b	0.44	—	0.53
b1	0.43	0.46	0.48
B1	1.52BSC		
c	0.25	—	0.31
c1	0.24	0.25	0.26
D	26.03	26.23	26.43
E1	6.35	6.55	6.75
e	2.54BSC		
eA	7.62BSC		
eB	7.62	—	9.30
eC	0	—	0.84
L	3.00	—	—
L/F载体尺寸 (mil)	110*140		
	150*190		
	140*190（内置散热片）		

图 10 LS2051 封装（DIP）