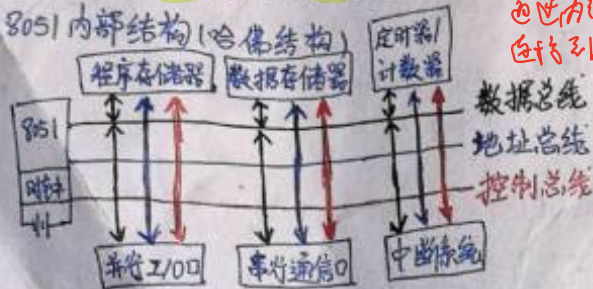
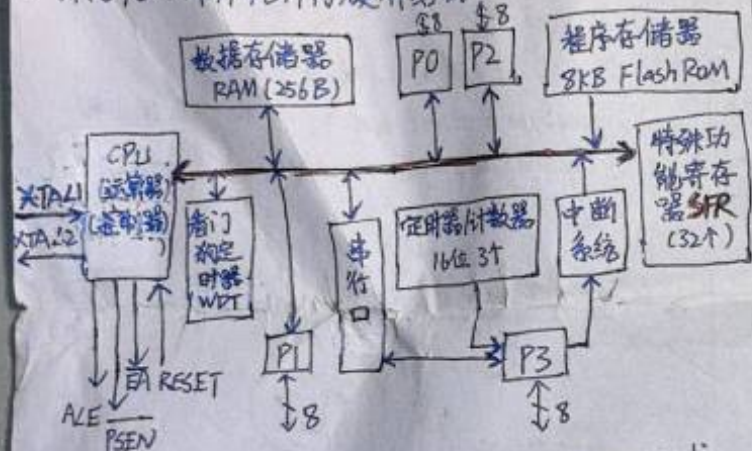


· 单片机 (MCU) 又称微控制器, 嵌入式微控制器 (EMCU)。是在一片半导体硅片上集成了中央处理单元 (CPU)、存储器 (RAM、ROM)、中断系统、定时器/计数器、并行 I/O、串行 I/O、时钟电路及系统总线, 用于测控领域的单片微型计算机。它由 CPU、总线、I/O 总线、总线等组成。

8051 内部结构 (哈佛结构) 与 PC 机不同, 它由 CPU、总线、I/O 总线、总线等组成。



· AT89S52 单片机片内硬件结构



CPU 对各种功能部件的控制是采用 SFR 集中控制方式。

· 存储器结构为哈佛结构, 即片外程序存储器空间和片内数据存储器空间是各自独立的。

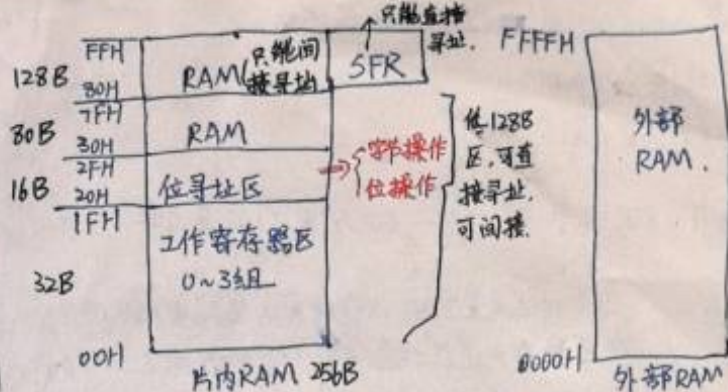
★ 存储器结构



(3) $\overline{EA}=1$ 时, 当 $PC < 1FFFH$ 时, CPU 读片内 ROM; 当 $PC > 1FFFH$ 时, CPU 读片外 ROM, 自动读片外 2000H~FFFFH 的程序代码。

$\overline{EA}=0$ 时, CPU 只读片外 RAM。

(4) PC (程序计数器), 即程序指针。单片机复位后, $PC=0000H$ 。当前程序在运行时, PC 值是下一条指令的地址。



★ 数据存储器区

SFR: (1) 可进行位寻址的 SFR, 其字节地址末位只能是 0H 或 8H。

(2) 堆栈指针 SP。堆栈只能设在片内 RAM 区。

单片机复位后, $SP=07H$, 所以堆栈实际从 08H 单元开始。

堆栈功能: 保护断点, 现场保护

堆栈的操作: \swarrow PUSH: SP 先自动加 1, 再把数据压入。

\searrow POP: SP 先把数据弹出, SP 再减 1。

单片机的位地址空间 (共 219 个可寻址位):
片内 RAM (20H~2FH) (128 个, 00H~7FH)
SFR 区 (字节地址末位为 0H, 8H 的) (9 个)

· 并行 I/O 端口: 4 个双向的 8 位并行 I/O 端口, $P0 \sim P3$ (均为 SFR, 可位寻址)。

功能:

	P0	P1	P2	P3
准双向口	✓	✓	✓	✓
复用端口	✓		✓	
第二功能				✓
SFR 地址	80H	90H	A0H	B0H

① P0 作通用 I/O 输入输出时, 需要外接上拉电阻, (P0 是漏极开路的)。P0 可作地址/数据总线 (双向口)。

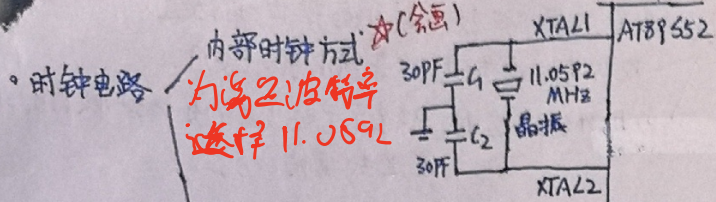
② 双向口, 有三种状态, 低、高、高阻; 准双向口, 没有高阻态, 只有低、高。

③ P0, P1, P2, P3 作通用 I/O 口时, 输入时均要置 1。即 $P0=0xFF$ 。

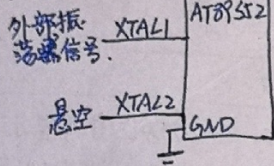
④ P2 口可用作高 8 位地址总线口。

⑤ P3 有第二功能。

P3.0	RXD	串行数据输入
P3.1	TXD	串行数据输出
P3.2	$\overline{INT0}$	外部中断 0 输入
P3.3	$\overline{INT1}$	外部中断 1 输入
P3.4	T0	定时器 0 外部中断输入
P3.5	T1	定时器 1 外部中断输入
P3.6	\overline{WR}	外部数据存储器写选通控制线
P3.7	\overline{RD}	外部数据存储器读选通控制线



外部时钟方式



访问SFR

使用关键字定义 sfr (8位) / sfr16 (16位, 位字节地址作定义地址)

通过头文件访问SFR: #include <reg51.h>

特殊功能寄存器中的位定义: sbit P1_7 = P1^7

绝对宏: #include <absacc.h>

④ C51语言的绝对地址访问

-at- 关键字 (定义的必须为全局变量)

[片内RAM, 片外RAM, I/O空间]

时钟周期: 单片机时钟控制信号的基本时间单位. $T_{osc} = \frac{1}{f_{osc}}$

机器周期: CPU完成一个基本操作所需要的时间. $T_{cy} = \frac{12}{f_{osc}}$

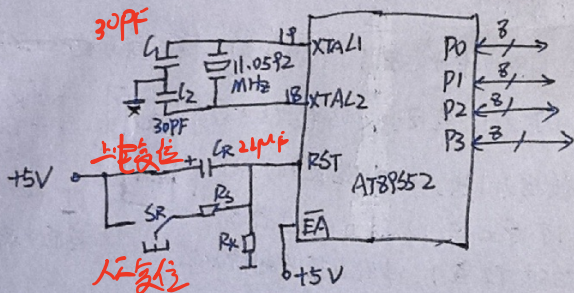
1个机器周期 = 12个时钟周期

可分为6个状态 S1~S6, 每个状态分为2拍: P1和P2, 所以12个时钟周期可表示为 S1P1, S1P2, S2P1, S2P2...

指令周期: 执行一条指令所需的时间

单片机复位后, 只有P0~P3为高电平, SP=07H, 其余寄存器均为0.

单片机的最小系统: (☆ 会画) 51单片机 + 时钟电路 + 复位电路.



& 按位与 (清0) ^ 按位异或 << 按位左移 (高位舍弃, 低位补0)

| 按位或 (置1) ~ 按位取反 >> 按位右移 (低位舍弃, 高位补0)

-crol- (, 1) 按位左移 (高位补低位)

-cror- (, 1) 按位右移 (低位补高位)

* 提取变量的内容 & 提取变量的地址. 文件 #include <intrins.h>

★ 延时函数 ★ (2分)

① void delays (unsigned char j) 延时 jms

```

unsigned char i;
while (j--)
{
    for (i=0; i<125; i++)
    {;}
}

```

晶振为 12MHz 时, 125次/120次代表 1ms.

② void delays (int x) 取值 0~2¹⁶

```

{
    int i, j;
    for (i=0; i<x; i++)    延时 x ms
    for (j=0; j<120; j++);
}

```

中断服务函数的一般形式:

函数类型 函数名 () interrupt n [using n] 组.

中断源的中断号.

03H	INT0	0
0BH	T0	1
13H	INT1	2
1BH	T1	3
23H	Serial	4
2BH	T2	5

C51语言中的数据类型与存储类型.

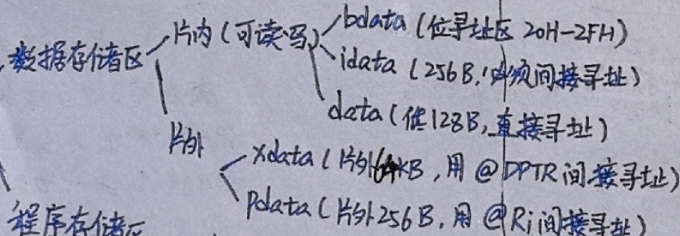
① C51扩展了4种数据类型: bit, sfr, sfr16, sbit.

不能用指针来对它们存取. 不能用来定义指针和数组.

unsigned char (8位, 1字节), 只有 bit (1位) 和 unsigned char 可直接支持机器指令. bit 的存储类型只能是 data/idata.

bit 定义普通的位变量, 只能是0或1. sbit 是定义特殊功能寄存器的可寻址位, 值是可以进行位寻址的SFR的某位的绝对地址.

② 数据存储类型



程序存储区: Code (用DPTR寻址) (只读不写)

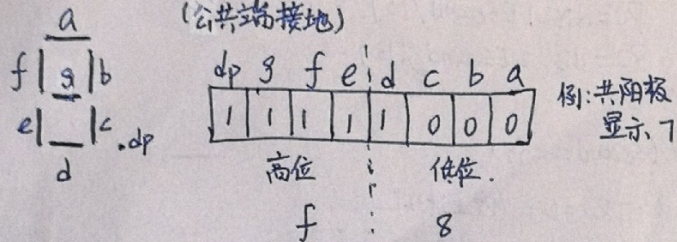
存储模式: SMALL — data (内部的数据存储) COMPACT — pdata LARGE — xdata

I/O口低电平输出驱动, 可以获得较大的驱动能力

共阳极：低电平驱动，想让哪段亮，就给控制这段的赋“0”
(公共端接+5V)

LED数码管

共阴极：高电平驱动。
(公共端接地)

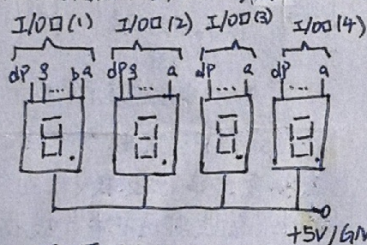


∴输入 f8H / 0x18 就会显示7

LED数码管显示方式

静态显示

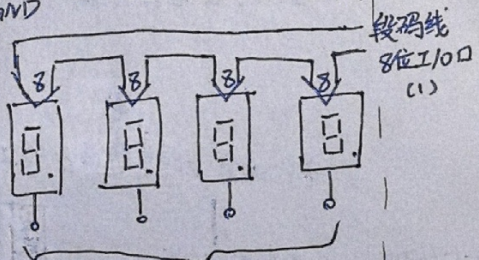
- ① 即无论多少位LED数码管，都同时处于显示状态。
- ② 各数码管的共阳极极连接在一起并且接地/+5V。
- ③ P口单独控制一个数码管。
- ④ 显示无闪烁，亮度较高，软件控制弱。



可见，要进行多位显示时，需要的I/O较多，而单片机只有4个I/O口。

动态显示

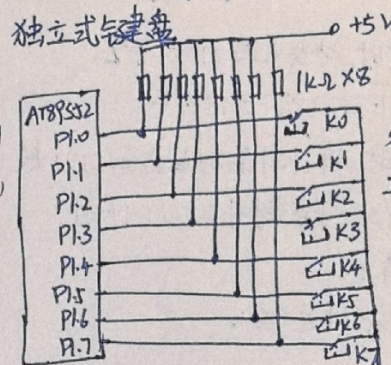
- ① 即每一时刻，只有1位的位选线有效，即选中某一位显示，其他各位的位选线都无效，不显示。
- ② 各数码管的段码线的相应段并接在一起，由一个8位I/O口控制，各显示位的公共端分别由另一单独的I/O端口线控制。
- ③ 每隔一定时间逐位地轮流点亮各LED数码管，控制好每位显示的时间间隔(如5s)就可造成“多位同时点亮”。



位选线4位 I/O口 (2)

可见，同样进行4位显示，动态显示仅用了2个I/O口。动态显示实质：用执行程序的时间来换取I/O端口数目的减少。(时间占用的多了)。降低成本。

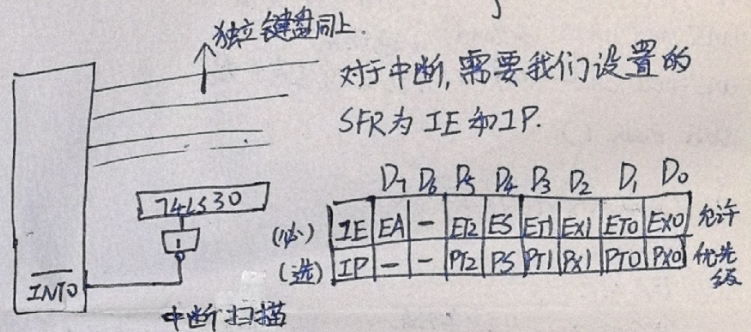
键盘：非编码键盘、独立式键盘、编码键盘、矩阵式键盘



查询扫描

各键相互独立，每个按键各接一条I/O线，通过检测I/O输入线的电平状态，就能判断哪个键被按下。

```
PI = 0xFF; // P1.0 为输入
KeyVal = P1; // 读入状态
KeyVal = ~KeyVal; // 取反
Switch(KeyVal)
{
    case 1: ...
    case 8: ...
}
```



#include <reg51.h>

#define uchar unsigned char

bit keyflag; // keyflag 为按键按下的标志位

uchar KeyVal; // 键号

void delay10ms(void); // 去抖函数

void main()

```
{
    IE = 0x81; // 总中断允许 EA=1, INT0 允许中断
    IP = 0x01; // 设置 INT0 为高优先级
    Keyflag = 0; // 先置 0, 否则中断里怎么判断有无按键按下
    do {
        if (Keyflag) {
            KeyVal = ~KeyVal;
            Switch(KeyVal) {
                case 1: ...; break;
                ...
                case 128: ...; break;
                default: break;
            }
            Keyflag = 0; // 处理完键号后，将按键按下标志位清 0
        }
    } while (1);
}
```

开关/按键去抖函数

```
void delay10ms(void)
{
    unsigned char i, j;
    for (i = 0; i < 100; i++)
        for (j = 0; j < 100; j++)
            ;
}
```


void into() interrupt 0 //INT0的中断函数,有键按下则执行中断函数;

```
{ uchar reread-key;
```

IE=0x80; //中断响应后,关闭本中断。

keyflag=1;

PI=0x0f; //PI口为输入状态。

keyval=PI; //读入键盘状态;

delay10ms;

reread-key=PI;

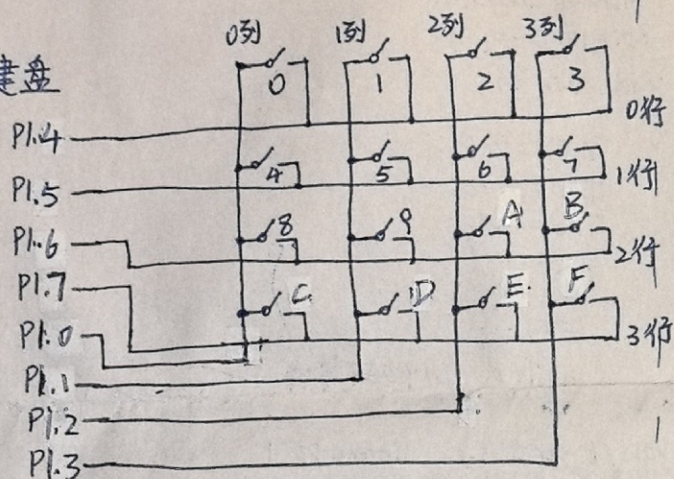
if(keyval==reread-key)

{ keyflag=1; }

IE=0x81; //本中断执行完,打开本中断。

}

矩阵键盘



① 确定扫描方式 (低、高电平/行、列扫描)

② 若行扫描, 则行为输出扫描电平, 列为输入按键电平; 若低电平行扫描, 则逐行置0, 其它均置高电平, 以免影响低电平的检测。

同理, 若低电平列扫描, 则列线为输出扫描电平, 行线为输入按键电平; 逐列置0, 其它均置高电平且保持高电平, 以免影响低电平的检测。

③ 设置键值, 使其与口线对应。

④ 查询扫描: 在主函数里执行键盘扫描程序

or 中断扫描: 在INT0的中断函数中执行键盘扫描程序。

例: P0口接共阳极数码管显示键值, 采用低电平行扫描, 查询方式, 4x4矩阵键盘。

#include<reg51.h>

#define uchar unsigned char

sbit L1=P1^0; //定义4条列线

sbit L2=P1^1;

uchar dis[16]={ ... 共阳极码表了; 0~F

void delay(unsigned int i) //延时函数。

```
{ uchar j;
```

while(i--)

{ for(j=0; j<120; j++); }

void main()

```
{ uchar temp,i;
```

while(1)

{ PI=0xfe; //行扫描, 从第0行开始

for(i=0; i<4; i++) //按行扫描, 为行变量, 共4行

{ if(L1==0) PO=dis[4*i+0];

if(L2==0) PO=dis[4*i+1];

if(L3==0) PO=dis[4*i+2];

if(L4==0) PO=dis[4*i+3];

delay(500);

temp=PI; //读入PI口此时的状态 (列线上有的为0)

temp=temp|0x0f; //使列线为按键输入状态, 防止左移时低位状态, 影响高位

temp=temp<<1; //准备下一行扫描

temp=temp|0x0f; //左移后, 低位有0, 所以要置高位

PI=temp; //送到PI口, 准备下一行扫描

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

输入
按键电平
↑

行	列	1.3	1.2	1.1	1.0
第0行	PI.7	1	1	1	0
	PI.6	1	1	1	1
	PI.5	1	1	1	1
	PI.4	1	1	1	1

输入
按键电平
↑

若变成 3x3 低电平列扫描, 查询方式

行	列	1.3	1.2	1.1	1.0
PI.7	1.6	1	1	1	1
PI.5	1.4	1	1	1	1
PI.3	1.2	1	1	1	1
PI.1	1.0	1	1	1	1

sbit R1=P1^3; //定义3条行线
sbit R2=P1^4;
sbit R3=P1^5;

void main()

```
{ uchar i,temp;
```

while(1)

{ PI=0xfe; //列扫描, 从第0列开始

for(i=0; i<3; i++) //为列变量, 共3列

{ if(R1==0) PO=dis[0*3+i];

if(R2==0) PO=dis[1*3+i];

if(R3==0) PO=dis[2*3+i];

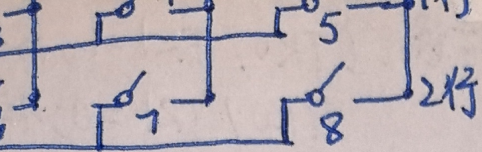
delay(500);

temp=PI; //读PI口此时状态, (行线上有的为0, 但左移时低位补0, 可设

法: temp=~temp;

temp=temp<<1;

使一条列线为0, 其他均为1, 所以有2种方法)



Sbit R2 = P1^4; 条行线
Sbit R3 = P1^5;

main ()

char i, temp;

while (1)

P1 = 0xfe; // 列扫描, 从第 0 列开始

for (i = 0; i < 3; i++) // 为列变量, 共 3 列

{ if (R1 == 0) P0 = dis[0*3 + i];

if (R2 == 0) P0 = dis[1*3 + i];

if (R3 == 0) P0 = dis[2*3 + i];

delay(500);

temp = P1; // 读 P1 口此时状态, (行线上有的为 0, 但左

法①: temp = ~temp;

temp = temp << 1;

temp = ~temp;

★ temp = temp | 0xf8; P1 = temp; } 法②: #include <intrins.h>

用 _crol_ (, 1)

物时低位补 0, 只能
使一条列线为 0, 其他均为 1
才行, 所以有 2 种方法)

中断扫描: #include <reg51.h>

(4x4 接上一页例子) Void main()

```
{ IE=0x81; //EA=1, 允许INT0中断
  IP=0x01; //设置INT0高优先级
  IT0=1; //设置INT0跳沿触发
  P1=0x0f; //P1口线均置0, 数据线均为按键输入状态
```

while(1); //等待中断

void into() interrupt 0 //有键按下, 则执行

```
{
  //键盘扫描程序
}
```

INT0的中断函数

④中断优先级: 有两个中断优先级, 高优先级(置1)和低优先级(置0), 可实现两级中断嵌套。优先级关系需满足两条基本规则: (1) 低优先级可被高优先级中断, 高优先级不能被低优先级中断; (2) 任何一种中断, 一旦得到响应, 不会再被它的同级中断源所中断。

若为同级中断, 则中断请求按时间的先后顺序响应。(即同级无嵌套!) 若同一时间同一级别的中断请求, 则按自然优先级执行。

(INT0 TO INT1 T1 Serial T2) 高 低

⑤外部中断的触发方式

跳沿触发方式(免跳变, 开关按下松即可) (IT0/IT1=1)

电平触发方式: (低电平, 开关要一直按下) (IT0/IT1=0)

⑥外部中断的响应时间

在一个单一中断的系统里, AT89S52单片机对外部中断请求的响应时间总是在3~8个机器周期之间。

最短3个机器周期 / 中断请求标志位查询占1个机器周期 (且这个机器周期恰好处于指令的最后1个机器周期)

执行调用指令LCALL, 以转到相应的中断服务程序入口, 占2个机器周期。

最长8个机器周期

发生在CPU进行中断标志, DIV → 4个 (最长的乘法/除法指令) 查询时, 刚好才开始执行中 LCALL → 2个 断返回或访问IE或IP的指令, 这时需要把当前指令执行完, 再继续执行一条指令后才能响应中断。

⑦推迟中断响应的情况:

(1) CPU正在处理同级或更高优先级的中断
(2) 所查询的机器周期不是当前正在执行指令的最后1个机器周期

RETI

(3) 正在执行的指令是从中断返回或是访问IE或IP的指令

如果存在上述三种情况之一, CPU将丢弃中断查询结果, 能对中断进行响应。

⑧中断请求的撤销

自动撤销 { T0, T1 跳沿触发的外部中断
外加电路撤销: 电平触发的外部中断
软件清除: 串行中断(TI/RXIF) 手动清除 定时/计数器T2

键盘扫描工作方式

查询扫描: 利用单片机空闲时, 调用键盘扫描子程序, 反复扫描键盘来响应键盘的输入请求, 但是如果单片机的查询频率过高, 虽能及时响应键盘输入, 但也会影响其他任务的进行。如果查询频率过低, 可能会有漏判现象。所以要根据单片机系统的繁忙程度和键盘的操作频率, 来调整扫描频率。

定时扫描

中断扫描

每隔一定时间对键盘扫描一次。通常利用定时器产生的定时中断, 在中断子程序中对键盘进行扫描。由于每次按键的时间一般都不会小于100ms, 所以为了不漏判, 定时中断周期一般应小于100ms。

★AT89S52单片机的中断系统

①. 中断响应: 即CPU对中断源提出的中断请求的接受。

② 中断响应需满足的条件 (★简答) (条件缺一不可!)

- (1) 总中断允许开关接通, 即IE寄存器的中断总允许位EA=1;
- (2) 该中断源发出中断请求, 即该中断源对应的中断请求标志为1;
- (3) 该中断源的中断允许位=1, 即该中断被允许;
- (4) 无同级或更高级中断正在被服务。

③ 中断响应的主要过程: 首先由硬件自动生成一条长调用指令(LCALL)跳向相应的中断请求源的中断入口地址(中断向量), 紧接着由CPU执行该指令。首先把PC的内容压入堆栈, 以进行断点保护, 然后把长调用指令的16位地址(addr16)送入PC, 使程序执行转向程序存储器中的中断地址区。

在CPU进行中断标志, DIV → 4个 (最长的乘法/除法指令)

刚好才开始执行中 CALL → 2个

或访问IE或IP的指令,这时需要把当前指令执行完
续执行一条指令后才能响应中断。

中断响应的情况:

正在处理同级或更高优先级的中断

所查询的机器周期不是当前正在执行指令的最后一
机器周期

RETI

正在执行的指令是从中断返回或是访问IE或IP的
指令

存在上述三种情况之一, CPU将丢弃中断查询结果,不
对中断进行响应。

中断请求的撤销 {
 自动撤销 { T0, T1
 跳沿触发的外部中断
 外加电路撤销: 电平触发的外部中断
 软件清除 { 串行中断 (TI/RI 手动清0)
 定时/计数器T2

① 中断系统对应的硬件 SFR

中断请求标志寄存器

TCON (定时器/计数器的控制寄存器) 88H

P7	P6	P5	P4	P3	P2	P1	P0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
8FH	-	8DH	-	8BH	8AH	89H	88H

SCON (串行口控制寄存器) 98H

P7	P6	P5	P4	P3	P2	P1	P0
-	-	-	-	-	-	TI	RI
-	-	-	-	-	-	99H	9BH

T2CON

中断允许寄存器 IE (A8H)

P7	P6	P5	P4	P3	P2	P1	P0
EA	-	ET2	ES	ET1	EX1	ET0	EX0
AFH	-	ADH	ACH	ABH	AAH	A9H	A8H

总中断允许位. T2 Serial T1 INT1 T0 INTO

中断优先级寄存器 IP (8BH)

P7	P6	P5	P4	P3	P2	P1	P0
-	-	PT2	PS	PT1	PX1	PT0	PX0
-	-	BPH	BCH	BBH	BAH	B9H	B8H

T2 Serial T1 INT1 T0 INTO

② 对 SFR 的软件编程

(1) TCON, SCON, 外部中断请求来了会自动置1, 不用自己操作.

(2) IE, IP 需要自己编写. 以 INTO 高级为例:

法一: IE = 0x81; // 总中断打开, EX0 置1
IP = 0x01; // 设置 INTO 为高优先级
ITO = 1;

法二: EA = 1; // 总中断允许
EX0 = 1; // INTO 中断允许
IP = 0x01 / PX0 = 1; // 优先级
ITO = 1; // 外部

中断函数: void 函数名() interrupt n using n
→ 在 中断源 函数中, 要先关闭该中断源 (中断源允许位 置0), 最后再打开该中断源.

注意: 任何情况下都不能直接调用中断函数

注意点: 灯闪烁 (亮, 灭, 亮, 灭...) 所以先点亮, 延时; 再全灭, 延时. (易失)

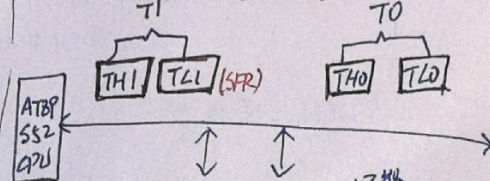
中断服务子程序与普通子程序有哪些相同和不同之处?

答: RETI 指令在返回时自动清除相应不可寻址的优先级触发器, 以允许下次中断, 而 RET 指令则没有这个操作, 除了这一点两条指令不同外, 其它操作都相同.

(子程序没有返回值, 函数有返回值).

定时器/计数器 T0 与 T1 (增1计数器)

T1 引脚 (P3.5) 外部脉冲 (计数) 或系统时钟 12 分频内部脉冲 (定时)
T0 引脚 (P3.4) 外部脉冲 (计数) 或系统时钟 12 分频内部脉冲 (定时)



TCON (控制寄存器) TMOD (工作方式寄存器)
设置 T0/T1 是 否运行 (TR0, TR1) 设置工作方式 (按字节操作)

T0/T1 可看成一个水桶
来1个脉冲, 计一次数, 当计数器计数溢出后, 计数溢出标志位 TFO/TF1 会自动置1.

(注意定时器/计数器 若用查询方式时, 当查询到 TFO/TF1 为1 要软件清0; 中断方式会自动清0).

定时器: 对系统时钟 12 分频内部脉冲中进行计数.

$$\text{定时时间} = (2^{\text{位数}} - \text{初值}) \times \frac{1}{12 \text{ fosc}}$$

计数器: 对外部脉冲中进行计数.

$$\text{计数次数} = 2^{\text{位数}} - \text{初值}$$

P7	P6	P5	P4	P3	P2	P1	P0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
T1 方式字节				T0 方式字节			

GATE 为0, 则 仅由 TRx 控制 T1/T0 运行; 为1, 则由 TRx 与 系统时钟 共同控制.
C/T 1 计数 0 定时器
M1 M0 0 0 → 方式0
0 1 → 方式1
1 0 → 方式2
1 1 → 方式3

方式0: 13位定时/计数器

0~4	0~7
TLx (5位)	THx (8位)

方式1: 16位定时/计数器

方式2: 8位的常数自动重新装入的定时/计数器

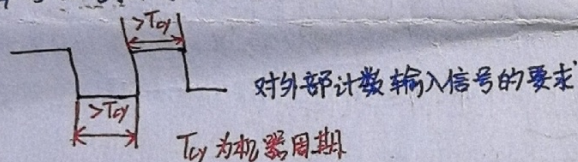
方式3: 仅适用于 T0, 此时 T0 分成两个 8 位计数器, T1 停止计数. (此时 T1 可用来作串行口波特率发生器)

方式0和方式1计数溢出后,计数器为全0,所以在其中,新函数中要重新装入初值。(循环计数/定时的情况下)
方式2是初值自动装载的8位定时器/计数器。当TLx计数溢出时,可自动将THx中的初值送至TLx。这种工作方式可以省去用户软件中重新装载初值的指令执行时间,简化定时初值的计算方法,可以相当精确地确定定时时间。

T0工作在方式3时,TL0作为8位定时器/计数器(TR0, TFO),TH0作为8位定时器(TR1, TF1)。T1用作串行口波特率发生器,有方式0、1、2三种工作方式,T1的运行由C1H控制。

★简答题 计数器可对外部输入计数脉冲的最高计数频率为系统振荡器频率的 $\frac{1}{24}$ 。

原因:当定时器/计数器T1、T0工作在计数器模式时,在第一个机器周期的S5P2对外部输入信号进行采样,在下一个机器周期的S5P2再对外部输入信号进行采样。如果两次采样结果形成一个负脉冲,则计数器加1。所以计数器可对外部输入计数脉冲的最高计数频率为系统振荡器频率的 $\frac{1}{24}$ 。**★**



• 对T1/T0的软件编程. $2^{16} = 65536$

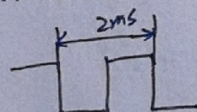
① 设置TMOD

② 计算计数初值X; $TH0/TH1 = X/256$; $TL0/TL1 = X \% 256$;

③ 设置IE (采用中断方式时) (EA=1; ET0/ET1=1)

④ 启动和停止计数器/定时器. (TR0/TR1=1; 启动
TR0/TR1=0; 停止)

例: 系统时钟为12MHz, 实现从P1.0输出一个占空比50%周期为2ms的方波。



选定时器T0, 方式1定时1ms.

① TMOD=0x01;

$$\textcircled{2} (2^{16}-X) \times \frac{12}{12\text{MHz}} = 1 \times 10^{-3}$$

$$X = 64536$$

$$TH0 = 64536 / 256 = 252 = 0xf4$$

$$TL0 = 64536 \% 256 = 24 = 0x18$$

1) 定时用查询方式

#include <reg51.h>

sbit P1_0 = P1^0;

Void main ()

TMOD=0x01; //设置T0为方式1

TR0=1; //启动T0

while (1)

{ TH0=0xf4; //装初值.

TL0=0x18;

do { } while (!TFO); //判断TFO是否为1, TFO=0, 原地循环, TFO=1, 往下执行

P1_0 = !P1_0; // P1.0状态求反

TFO=0; //TFO标志清0. (因为用的是查询方式, 查询TFO是否为1 (即是否有溢出), 所以要手动清0).

(2) 定时中断方式

#include <reg51.h>

sbit P1_0 = P1^0;

Void main ()

{ TMOD=0x01;

TH0=0xf4;

TL0=0x18;

TR0=1;

EA=1;

ET0=1;

while (1); //循环等待

Void timer0 () interrupt 1

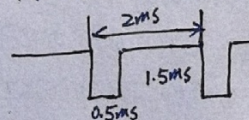
{ TH0=0xf4; //重新装入初值

TL0=0x18;

P1_0 = !P1_0;

}

例: 同上例, 只是将占空比改成3:1, 周期为2ms的方波。



选定时器T0, 方式1定时0.5ms.

① TMOD=0x01;

$$\textcircled{2} (2^{16}-X) \times \frac{12}{12\text{MHz}} = 0.5 \times 10^{-3}$$

$$X = 65036$$

$$TH0 = 65036 / 256 = 254 = 0xfe;$$

$$TL0 = 65036 \% 256 = 12 = 0x0c;$$

查询方式:

#include <reg51.h>

sbit P1_0 = P1^0;

Void main ()

{ unsigned char i=0;

TMOD=0x01;

TR0=1;

while (1)

{ TH0=0xfe; TL0=0x0c;

do { } while (!TFO);

if (i==3) { P1_0 = !P1_0; //保持1.5ms高电平

if (i==4) { P1_0 = !P1_0; i=0; //保持0.5ms低电平

TFO=0; //手动清0.

}

中断方式:

#include <reg51.h>

unsigned char i=0;

sbit P1_0 = P1^0;

Void main ()

{ TMOD=0x01;

TH0=0xfe;

TL0=0x0c;

TR0=1;

EA=1;

ET0=1;

while (1);

}

Void timer0 () interrupt 1

{ TH0=0xfe;

TL0=0x0c;

i++;

if (i==3) P1_0 = !P1_0;

if (i==4) { P1_0 = !P1_0;

i=0; }

例：制作一个LED数码管显示的秒表。

最小计时单位0.1s。

计时范围0.1~9.9s。

当第一次按下计时功能键时，秒表开始计

时并显示：

P3.7 (按下为低电平)

第2次按下键时，停止计时，将计时的时间送到LED数

码管显示；如果计时到9.9s，将重新开始从0计时；

第3次按下键时，秒表清0。

再次按下键，重复上述计时过程。

#include <reg51.h>

unsigned char dis1[] = { ... } // 数码管显示0~9带小数点的段码表

unsigned char dis2[] = { ... } // 数码管显示0~9不带小数点的段码表

unsigned char timer=0; // 记录中断次数

unsigned char second; // 存储秒

unsigned char key=0; // 记录按键次数

void main()

{ TMOD=0x01; // T0方式1

ET0=1;

EA=1;

second=0; // 设初值

P0=dis1[second/10];

P2=dis2[second%10];

while(1)

{ if((P3 & 0x80) == 0x00) // 键被按下

{ key++;

switch(key)

{ case 1: TH0=0x00; TL0=0x00; TR0=1; // 定时5ms break;

case 2: TR0=0; break;

case 3: key=0; // 按键次数清0

second=0; // 秒表清0

P0=dis1[second/10];

P2=dis2[second%10];

break;

}

while((P3 & 0x80) == 0x00); // 如果按键时间过长在地循环

}

} void timer0() interrupt 1

{ TR0=0; // 停止计时，执行以下操作。(会带计时误差)

TH0=0x00; TL0=0x00;

timer++;

if(timer==20) // 计时0.1s

{ timer=0;

second++;

P0=dis1[second/10];

P2=dis2[second%10];

}

if(second==99)

{ TR0=0; // 停止计时

second=0; // 清0

key=2; // 按键数量2，当再次按下键时，key++，

key=3，秒表清0复原，

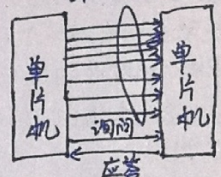
}

else TR0=1; // 计时不到9.9s时，启动继续计时

• 单片机的串行口

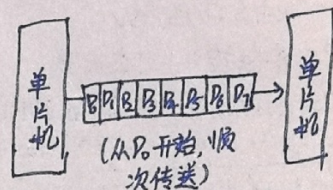
(1) 单片机的数据通信有并行通信与串行通信两种方式。

8位数据同时传送



单片机并行通信示意图

并行通信相对传输速度快。但长距离传送时成本高，因此适合于短距离传输。



单片机串行通信示意图

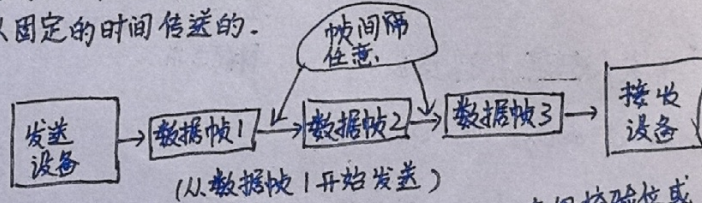
串行通信在发送时，要把并行数据变成串行数据发送到线路上，接收时要把串行数据变成并行数据。传输线少，长距离传送时成本低。

(2) 串行通信 同步串行通信 收发双方采用一个同步时钟。传送数据的位之间的时间间隔均为“位间隔”的整数倍，传送的字符间不留间隔，即保持位同步关系。

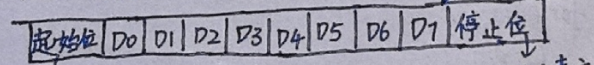
异步串行通信

收发双方使用各自的时钟控制数据的发送和接收。

以数据帧为单位进行数据传输，各数据帧之间的间隔是任意的，但每个数据帧中的各位是以固定的时间传送的。

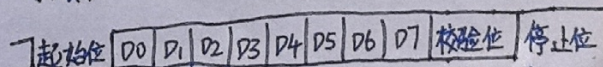


★ 典型的数据帧格式 ★ (含画)



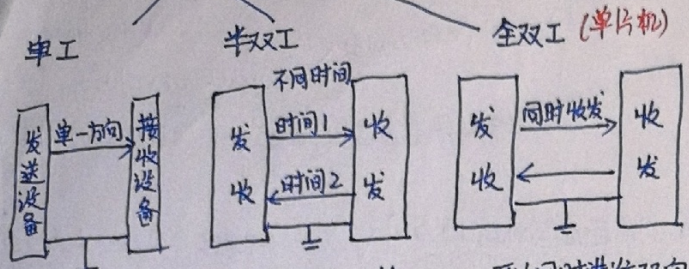
用0表示

用1表示



可见，1帧有10位或11位。

(3) 串行通信的传输模式 (按数据传输的方向及时间关系分)



数据传输只能按一个固定方向传输。

可以双向传输，但不能同时进行。

可以同时双向传输。

(4) 串行通信的错误校验

奇偶校验 (验的是数据中1的个数)
 代码和校验
 循环冗余码校验

TB8: 发送的第9位数据。1 (可编程位, 自动置1)

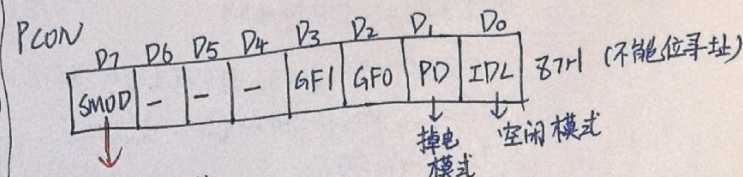
双机串行通信时, 作奇偶校验位使用。

多机串行通信时, 用来表示主机发送的是地址帧 (TB8=1), 还是数据帧 (TB8=0)

RB8: 接收的第9位数据。

串行口的4种工作方式

SM0	SM1	方式
0	0	0 同步移位寄存器方式 (用于扩展I/O口)
0	1	1 8位异步收发, 波特率可变 (由定时器控制)
1	0	2 9位异步收发, 波特率为 $f_{osc}/64$ 或 $f_{osc}/32$
1	1	3 9位异步收发, 波特率可变 (由定时器控制)



波特率选择位 (SM0=0/1)

串行口的工作方式0: 适用于单片机外接移位寄存器, 用来扩展并行I/O口。

① 以8位数据为1帧, 没有起始位和停止位。发送/接收完第8位数据时, TI/RI置1。

② 波特率固定, 为 $\frac{f_{osc}}{12}$

③ RXD(P3.0) 收、发数据, TXD(P3.1) 输出同步移位脉冲。

串行口的工作方式1: 适用于双机串行通信

① 以10位数据为1帧, 一个起始位(0), 8个数据位, 1个停止位(1)。串行口发送停止位的开始时, TI置1, 串行口接收到停止位时, RI置1。

② 波特率可变。波特率 = $\frac{2^{SMOD}}{32} \times \text{定时器T1的溢出率}$

若定时器T1为方式2, 初值为X, 则定时时间为 $(256-X) \cdot \frac{12}{f_{osc}}$

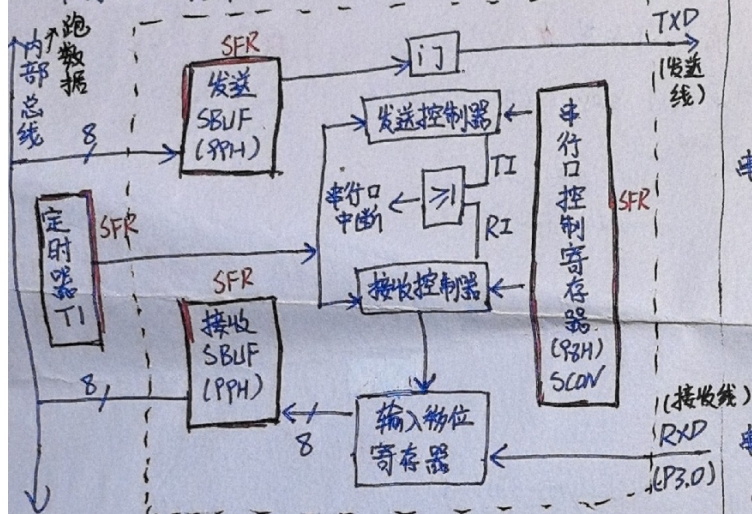
波特率 = $\frac{2^{SMOD}}{32} \times \frac{f_{osc}}{(256-X) \cdot 12}$

③ 当CPU执行写数据到发送缓冲器SBUF的命令后, 就启动发送。当检测到起始位的负跳变时, 则开始接收。

位检测采样, 接收每一位数据时, 都进行3次连续采样 (第7, 8, 9个脉冲中采样), 接收的值是3次采样中至少两次相同的值, 以保证收到的数据位的准确性。

检测负跳变时, 也是3次采样, 取两次相同的值, 以确认是否是真正起始位的开始。

(4) 串行口的硬件



定时器T1用作波特率发生器, 常选工作方式2 (因为可以自动重装初值, 避免了执行重装参数指令所带来的时间误差)。

发送控制器将并行数据转为串行数据;

接收控制器将串行数据转为并行数据;

数据发送/接收完后, TI/RI会自动置1。 (TI/RI要手动清0)

状态可供软件查询, 也可申请中断。CPU响应中断后, 在中断服务程序中向SBUF写入要发送的下一帧数据/要求CPU从接收SBUF中取走数据。

串行口控制寄存器SCON

	D7	D6	D5	D4	D3	D2	D1	D0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H

4种工作方式选择位

REN=1, 允许串行口接收数据
 REN=0, 禁止接收数据

多机通信会用到
 接收端 SM2=1 时, RB8=1, 收
 RB8=0, 丢
 由发送的TB8决定

SM2=0 时, RB8=1 收
 RB8=0 收

串行口工作方式2、3：适用于多机通信

① 每帧数据均为11位

② 方式2波特率 = $\frac{2^{SMOD}}{64} \times f_{osc}$

方式3波特率 = $\frac{2^{SMOD}}{32} \times \text{定时器T1的溢出率}$

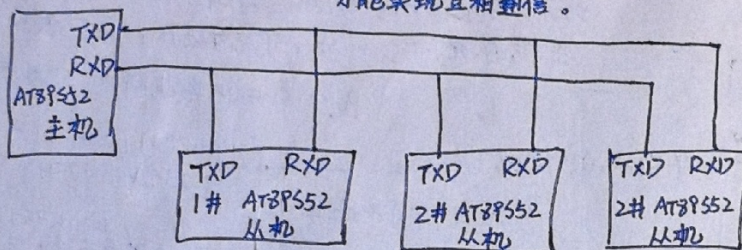
③ 发送：先由软件设置TB8，再将要发的数据写入SBUF，即可启动发送过程。串行口能自动把SCON中的TB8取出，并装入到第9位数据位的位置，再逐一发送出去。（所以TB8从始至终没到过发送SBUF）。

④ 接收：REN=1时，允许接收数据。

★ 多机通信（常用主从结构：只有一个主机，其余全是从机）。

原理电路

某一时刻，主从机间只能单独交流，从机间不能直接通信，只能通过主机才能实现互相通信。



工作原理：Step1. 主机发地址包 { 主机 TB8=1
从机 SM2=1

Step2. 从机进行地址识别，符合的从机 SM2=0

Step3. 主机发数据包，TB8=0。

• 波特率：串行口每秒钟发送（或接收）的位数。单位：bit/s

注意：为减小波特率误差，应该使用的时钟频率必须为 11.0592MHz。

接收 SBUF：只读不写
发送 SBUF：只写不读。

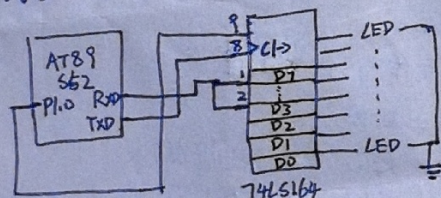
(5) 对 SFR 的编程（P0、SBUF、SCON、PCON、IE、TI 等）

注意 SFR 间不能互相传送数据，要用中间变量！

Step1. 选串行工作方式

Step2. 查询还是中断

例：编写程序控制8只发光二极管流水灯点亮，使用串行口工作方式0。



#include <reg51.h>

Sbit P1_0 = P1^0;

unsigned char nSendByte;

void delay(unsigned int j) // 延时函数

{ unsigned char i;

while(j--)

{ for(i=0; i<120; i++); }

}

void main()

{ SCON=0x00; // 设置串行口为方式0，且单片机不接收。

EA=1; // 开中断

ES=1;

nSendByte=1; // 送初值。

★ SBUF=nSendByte; // CPU向SBUF写入数据，启动串行发送

PI_0=0; // 允许向74LS164发送

while(1); // 等待

}

void Serial() interrupt 4 // 串口中断函数。

{

PI_0=1; // 允许74LS164并行输出，点亮流水灯

SBUF=nSendByte;

delay(500);

PI_0=0; // 关闭74LS164输出，启动输入

nSendByte=nSendByte<<1; // 左移一位，实现流水灯

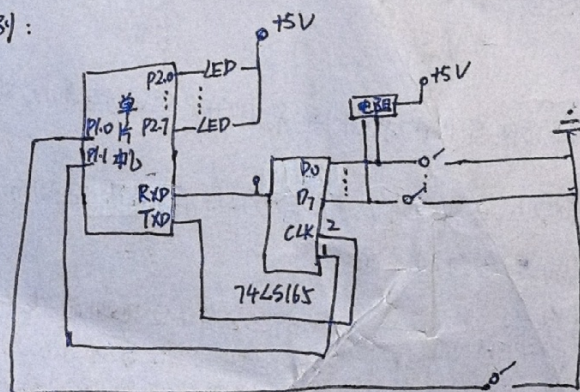
if(nSendByte==0) nSendByte=1;

SBUF=nSendByte;

★ TI=0; // 手动清0
RI=0;

}

例：



串口方式0输入，当P1.0处按键按下时，74LS165读

取开关状态，将数据送入单片机，单片机P2.0点亮灯

P1.1=0时，74LS165并行输入数据，串行输出关闭；

P1.1=1时，74LS165串行输出数据，并行输入关闭。

#include <reg51.h>

Sbit P1_0 = P1^0;

Sbit P1_1 = P1^1;

unsigned char nRxByte;

void delay(unsigned int j) // 延时函数。

{ unsigned char i;

while(j--)

{ for(i=0; i<120; i++); }


```

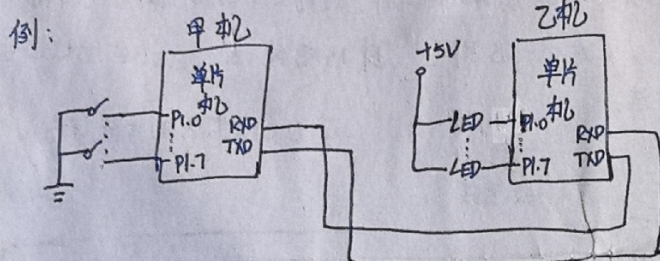
void main ( )
{
    SCON = 0x10; // 串行口方式0, REN=1, 允许接收
    EA=1; // 开中断
    ES=1;
    while(1); // 等待
}

```

```

void Serial ( ) interrupt 4
{
    if (PI-0 == 0)
    {
        PI-1 = 0; // 74LS165 并行读入开关状态
        delay(1);
        PI-1 = 1; // 74LS165 向单片机发送数据
        *RI = 0;
        nRxByte = SBUF;
        P2 = nRxByte; // 点亮灯
    }
}

```



用串行口工作方式1实现双机通信, 晶振频率均为11.0592 Hz.

// 甲机串行发送 (查询方式)

```

#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
void main ( )
{
    uchar temp = 0;
    TMod = 0x20; // 设置T1为方式2
    TH1 = 0xf6; // 波特率9600
    TL1 = 0xf6;
    SCON = 0x40; // 串行口方式1, 发送, 不接收
    PCON = 0x00; // SMOD=0
    TR1 = 1; // 定时器开始工作
    P1 = 0xff; // P1口为输入状态
    while(1)
    {
        temp = P1;
        SBUF = temp;
        while (TI == 0); // 判断是否发送完
        TI = 0; // 发送完, TI清0
    }
}

```

```

// 乙机接收程序
#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
void delay (uint) // 延时函数
{
    uchar i;
    while(i--) { for(i=0; i<120; i++); }
}

```

```

void main ( )
{
    uchar temp = 0;
    TMod = 0x20;
    TH1 = 0xf6;
    TL1 = 0xf6;
    TR1 = 1;
    SCON = 0x50; // 串行口方式1, 允许接收
    PCON = 0x00;
    while(1)
    {
        while (RI == 0); // 等待接收完数据
        RI = 0; // 接收完清0
        temp = SBUF;
        P1 = temp;
        delay(300);
    }
}

```

(6) 串行通信标准接口

- TTL电平通信接口: 传输距离短(1.5m之内), 抗干扰能力差
- RS-232C双机通信接口: 1.5m~30m, 采用负逻辑
- RS-422A双机通信接口: 采用差分信号传输, 抗干扰能力强, 距离可达1000多米, 全双工
- RS-485双机通信接口: 同RS-422A, 不过RS-485为半双工, 常用于工业现场采集和控制信号传输。

• 单片机系统的并行扩展

- 数据存储器扩展
- 程序存储器扩展
- I/O接口的扩展

并行扩展即通过系统总线把AT89S52单片机与各扩展器件连接起来。

系统总线

- 地址总线(AB): 用于传送单片机单向发出的地址信号, 以便进行存储器单元和I/O接口芯片中的寄存器的选择。
- 数据总线(DB): 数据总线是双向的, 用于单片机与外部存储器之间或I/O接口之间传送数据。
- 控制总线(CB): 单片机单向发出的各种控制信号线。

用时,均
需进行电平
转换。因为单片机
输出的为TTL电平。

达1000多米。全双工

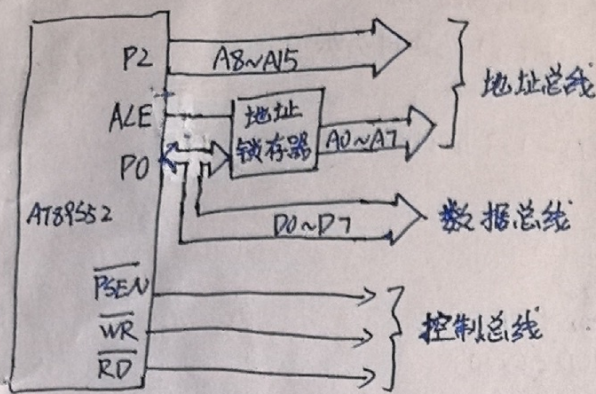
RS-485双机通信接口: 同RS-422A,
只不过RS-485为半双工, 常用于工业
现场采集和控制信号传输。

- 单片机系统的并行扩展
 - 数据存储器扩展
 - 程序存储器扩展
 - I/O接口的扩展

并行扩展即通过系统总线把AT89S52单片机与各扩展器件连接起来。

系统总线 {
地址总线(AB): 用于传送单片机单向发
出的地址信号, 以便进行
存储器单元和I/O接口芯
片中的寄存器的选择。
数据总线(DB): 数据总线是双向的, 用于
单片机与外部存储器之间
或I/O接口之间传送数据。
控制总线(CB): 单片机单向发出的各种控制
信号线。

★ 扩展的核心, 是如何设计三种总线 ★ (15分)



① P0口作为低8位地址/数据总线。通过外接1T8位地址锁存器实现P0口数据总线与地址总线的分时复用。AT89S52单片机对外部扩展的存储器单元进行访问时, 先发出低8位地址送入地址锁存器锁存, 锁存器输出作为系统的低8位地址(A7~A0), 随后, P0口又作为数据总线口(D7~D0)。

② P2口的口线作为高8位地址线(A8~A15)。再加上地址锁存器输出提供的低8位地址, 便形成了系统的16位地址总线, 从而使单片机系统的寻址范围达到64KB (2^{16})。

③ 控制信号线
 RD/WR 作为外部扩展的数据存储器的读/写选通控制信号。
 PSEN 作为外部扩展的程序存储器的读选通控制信号。
 ALE 信号作为P0口发出的低8位地址的锁存控制信号。(与地址锁存器的G端相连, G=1, P0口发的是地址, G=0, P0口发的是数据)。

★ 外设地址空间分配

【单片机发出的地址信号(低电平有效)】

用于选择某个存储器单元, 必须进行“片选”和“单元选择”两种选择)

★ 外部RAM的扩展

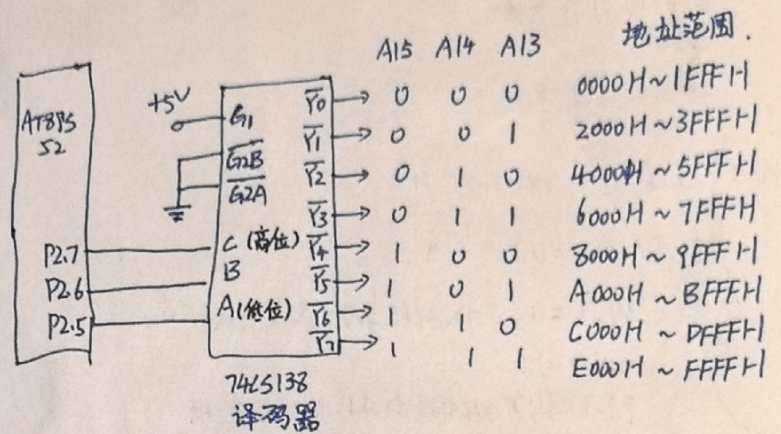
常用的静态RAM芯片

地址 = 片选 + 片内地址
 由高位地址决定 由芯片容量决定

扩RAM时, 单片机

RAM芯片
 RD 连接 OE (读选通信号输入线)
 WR 连接 WE (写选通信号输入线)

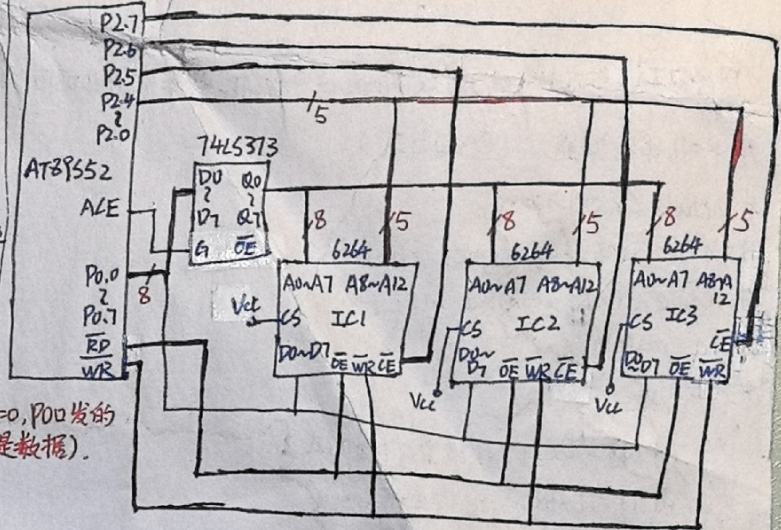
例: 要扩8片8KB的RAM 6264, 如何通过74LS138芯片把64KB空间分配给各个芯片?



Y0~Y7 分别接到8片RAM 6264的片选端, 实现8选1的片选。而低位地址 P2.4~P2.0 与 P0.7~P0.0 完成对选中RAM 6264芯片中的各个存储单元的“单元选择”。

例: 线选法扩展外部RAM。若用RAM 6264芯片, 则可扩展3片。(因为8KB即 2^{13} , 即13根地址线, 高3位地址线可用于片选)。

电路图: — 控制线
 — 地址线
 — 数据线
 CE 为RAM芯片的“片选”端。



P2.7	P2.6	P2.5	选中芯片	地址范围	存储容量
1	1	0	IC1	C000H ~ DFFFH	8KB
1	0	1	IC2	A000H ~ BFFFH	8KB
0	1	1	IC3	6000H ~ 7FFFH	8KB

比如 IC1: 地址范围

	A15	A14	A13	A12	A11~A8	A7~A4	A3~A0
最小	1	1	0	0	0	0	0
最大	1	1	0	1	1	1	1
不变							