单片机最小系统硬件&存储器编程结构

信息工程学院　陈梅

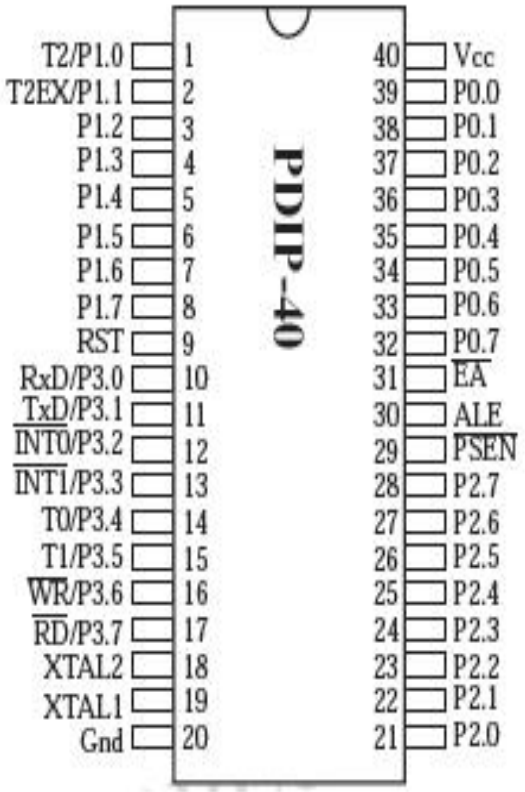
单片机最小系统　重点

重点：

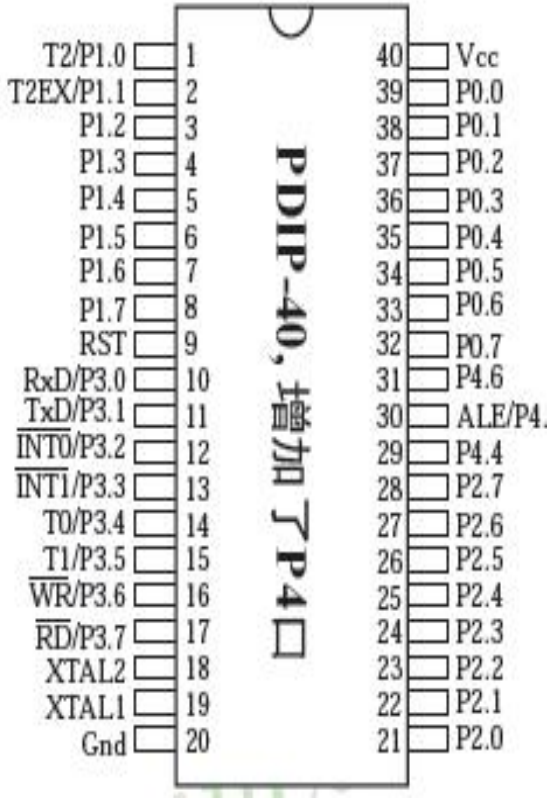
- STC89C52单片机最小系统的构成电路
- STC89C52单片机的三种时序周期的区别和计算
- STC89C52单片机的复位状态值的含义
- STC89C52单片机I/O口

STC89C52单片机的外部引脚及功能

PDIP-40的封装与传统的89C52芯片兼容



PDIP40的HD版本

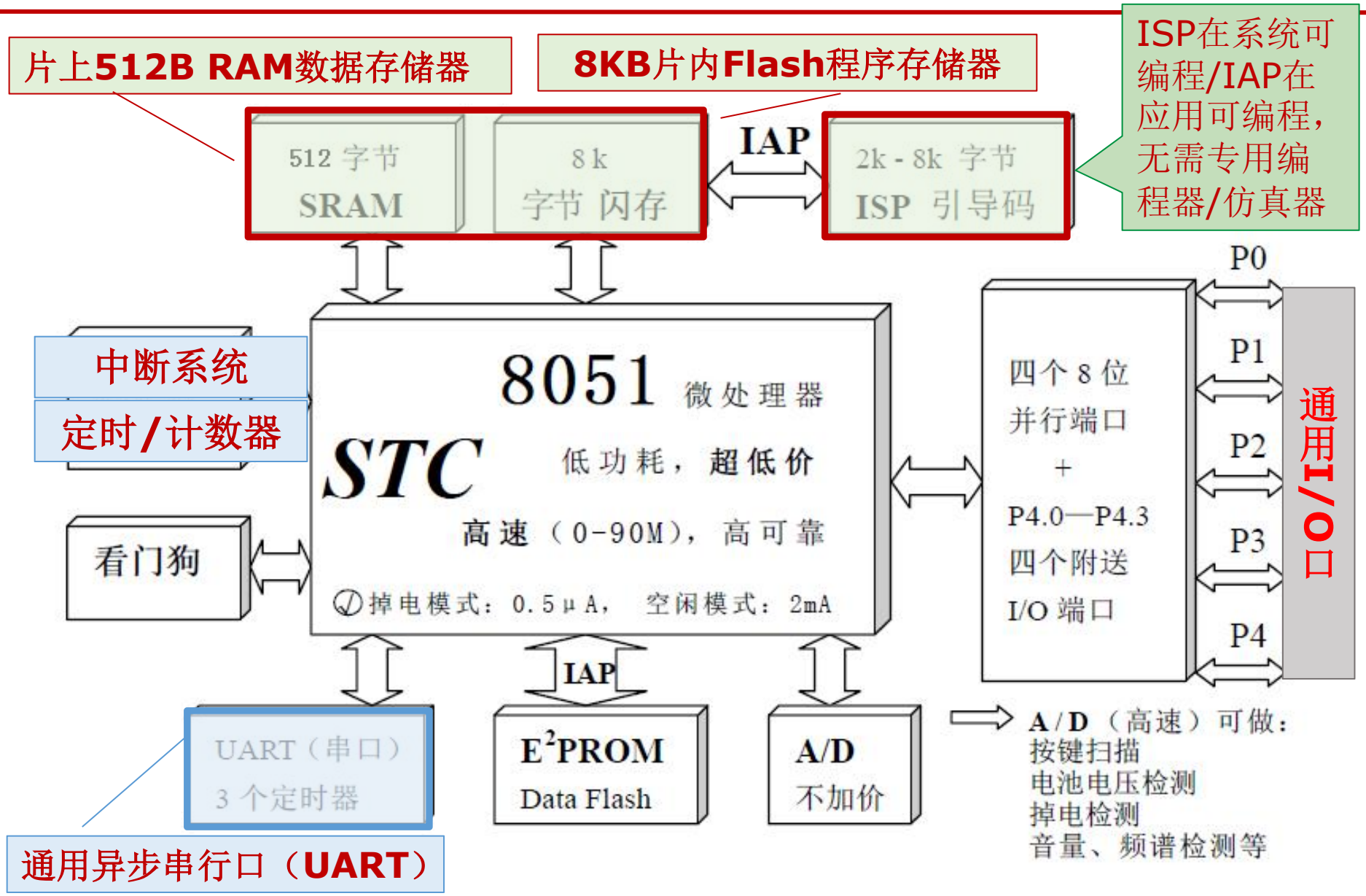


PDIP40的90C版本

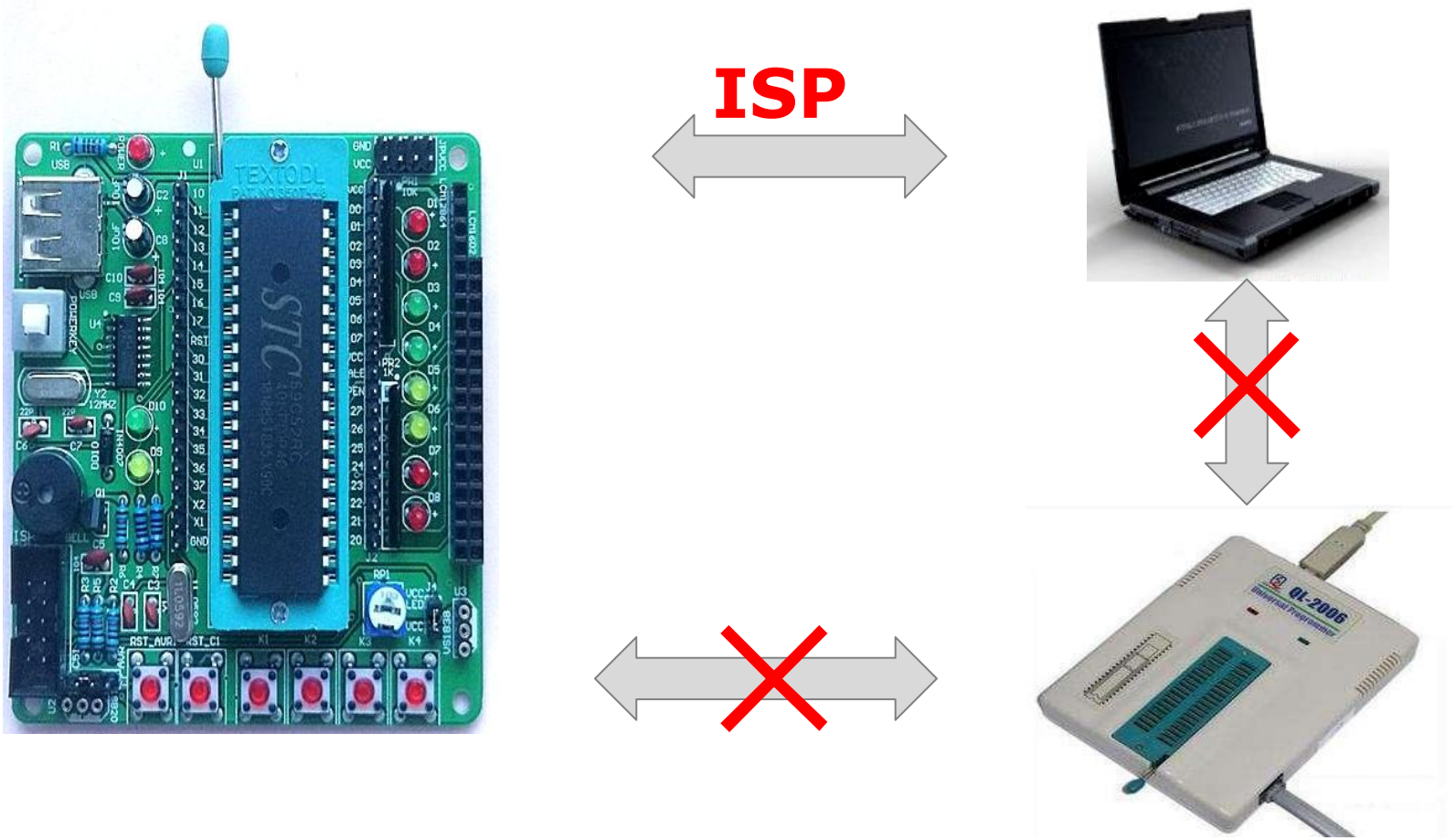


http://www.stcmcu.com/index.htm

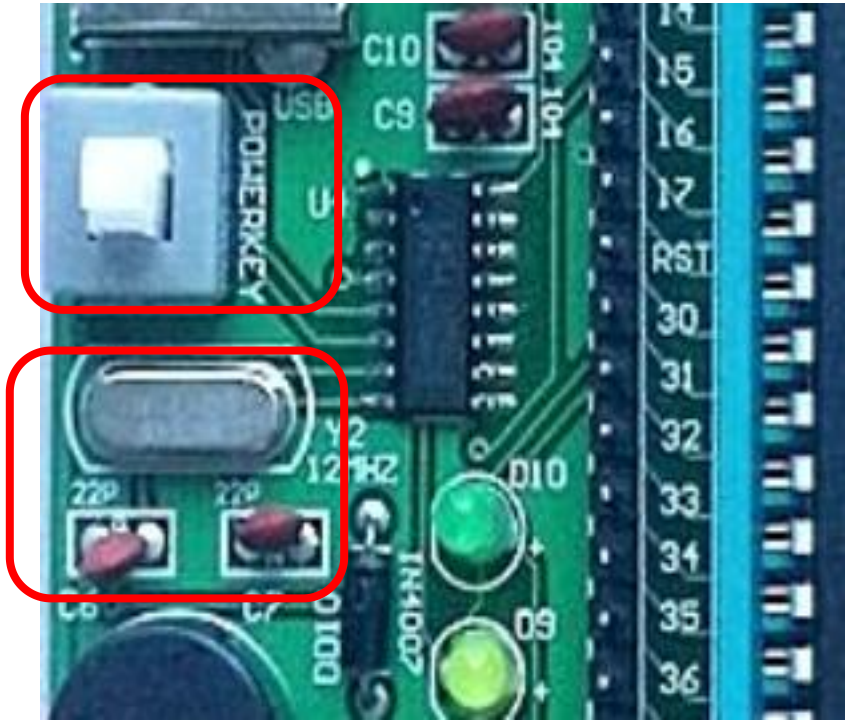
STC89C52单片机的内部组织结构及特点

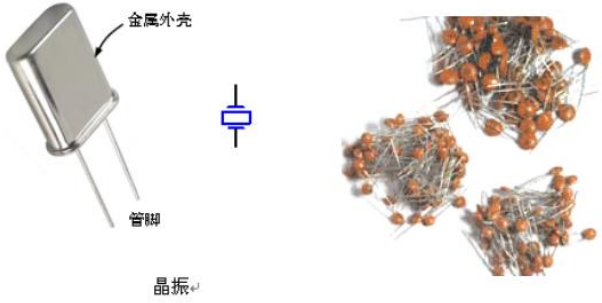


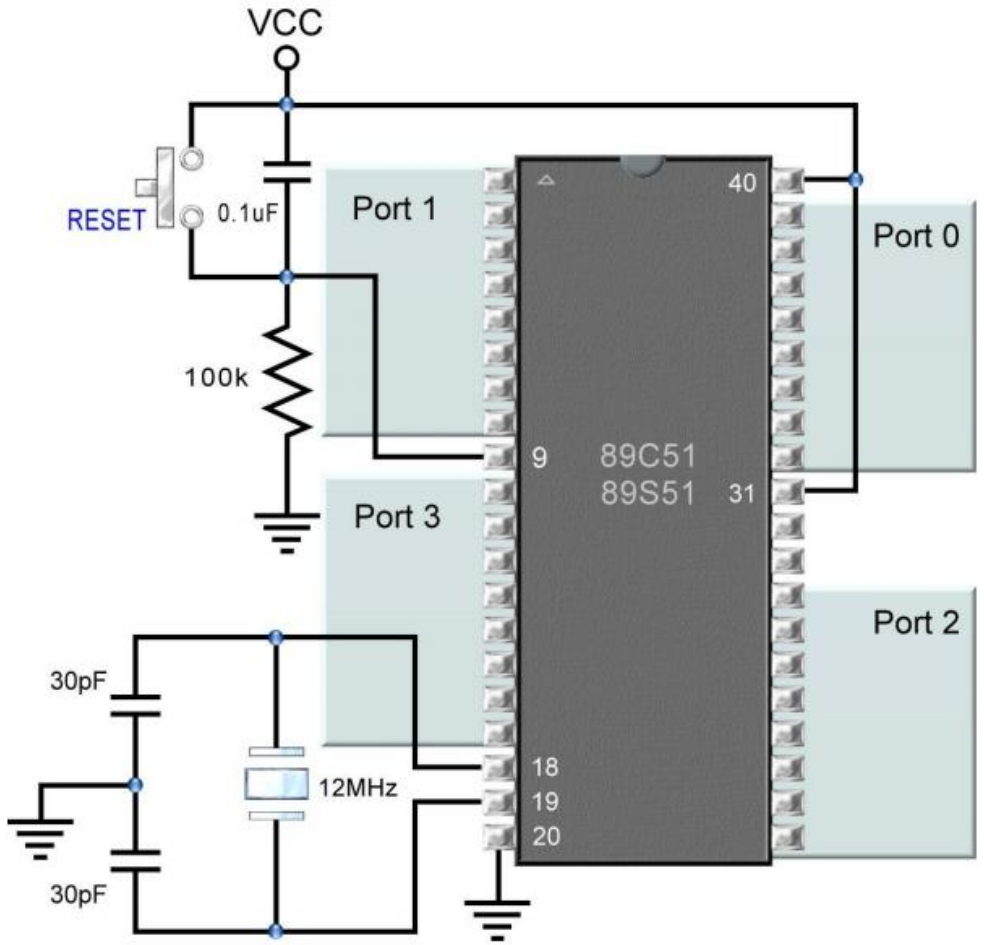
选用STC系列单片机的一个主要原因：在系统可编程（ISP）特性， 无需专用编程器/仿真器，即省却了每次编程必须插拔单片机到专用编程器上的麻烦，可以直接将STC单片机固定焊接在PCB板上，进行程序的下载调试。



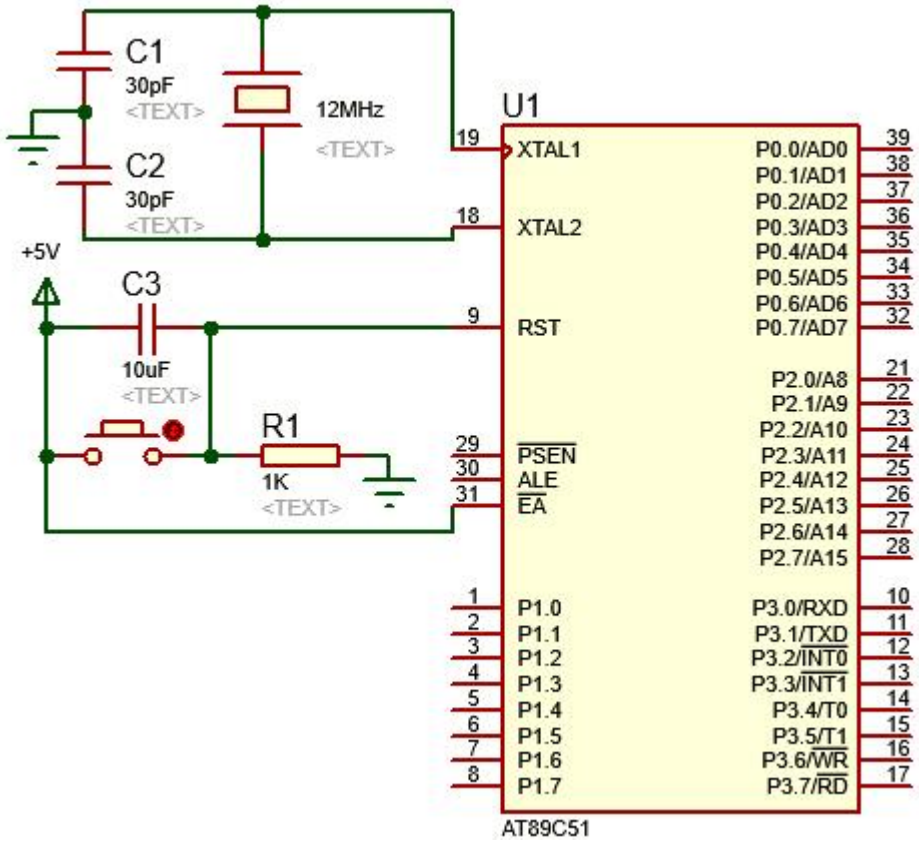
单片机工作最小系统——时钟电路和复位电路

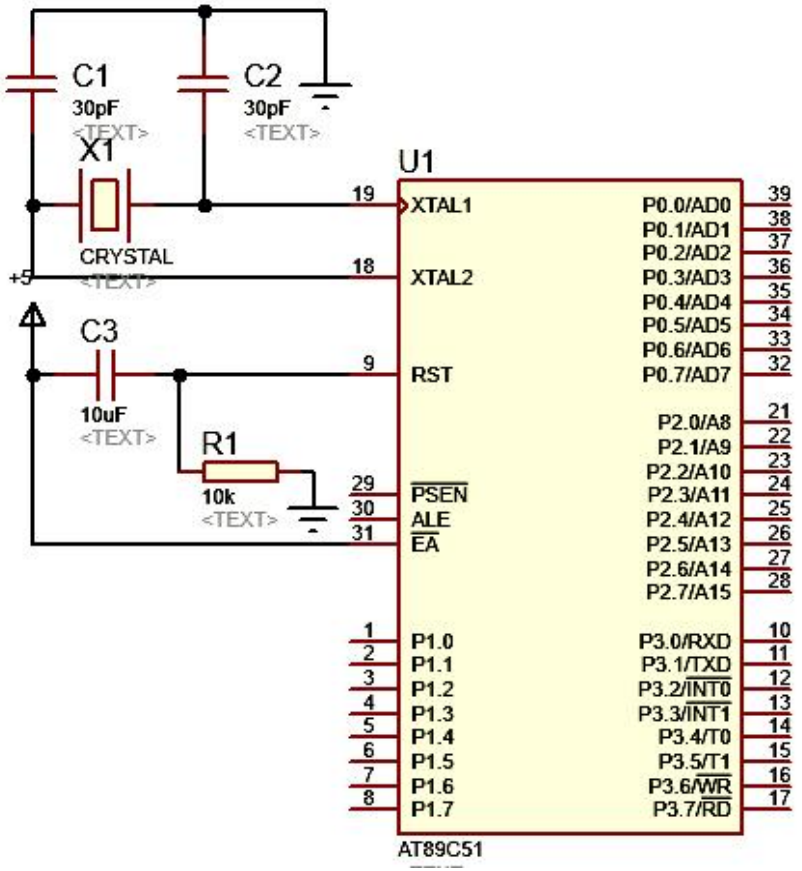






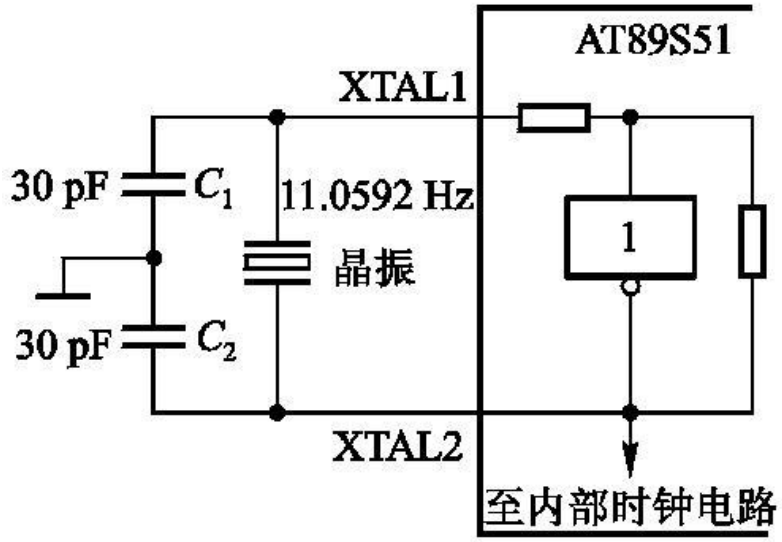
单片机工作最小系统——时钟电路和复位电路





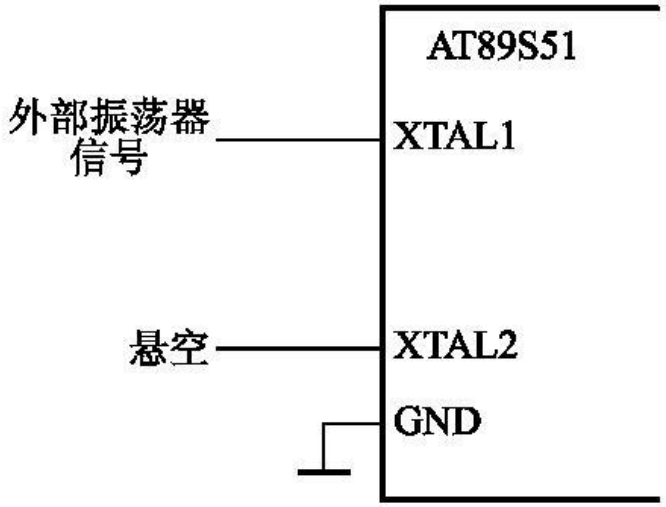
STC89C52单片机常用时钟电路

内部时钟方式电路



STC89C52引脚XTAL1和引脚XTAL2跨接石英晶体振荡器和微调电容，和内部的高增益反相放大器构成一个稳定的自激振荡器。

外部时钟方式电路

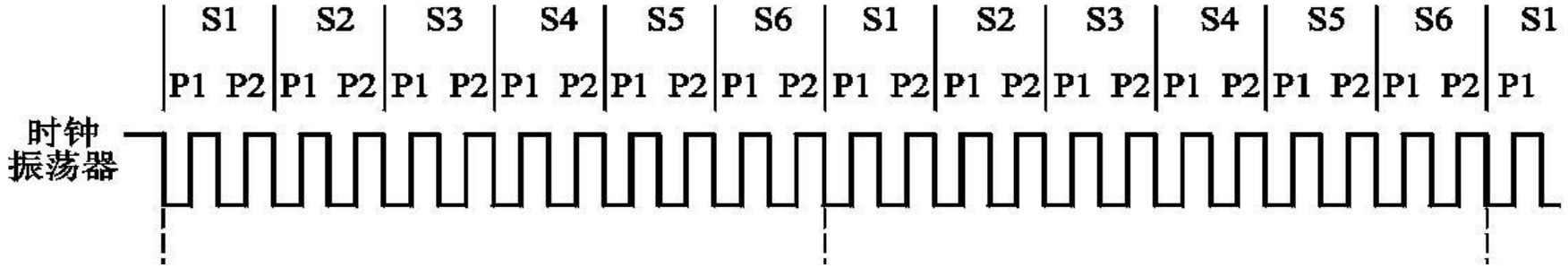


外部时钟源直接接到XTAL1端，XTAL2端悬空。用现成的外部振荡器产生脉冲信号。常用于多片单片机同时工作，以便于多片单片机之间的同步。

单片机时序

- **单片机时序**就是CPU在执行指令时所需控制信号的时间顺序。

一般的时序单位：**时钟周期、机器周期和指令周期**。



指令周期——执行一条指令所需要的时间，一般由若干个机器周期组成。

- **指令不同，所需的机器周期数也不同：**

  单字节和双字节指令一般为单机器周期和双机器周期；

  三字节指令都是双机器周期；

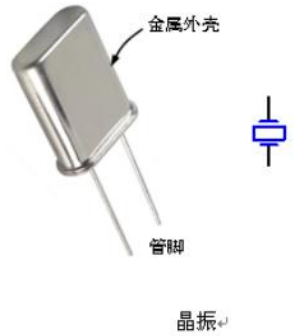
  乘、除指令占用4个机器周期。

时钟周期、机器周期、指令周期

- **时钟周期——最基本的、最小的时间单位。**

  又称振荡周期为一个节拍P，是时钟脉冲的倒数，即外接晶振的倒数。

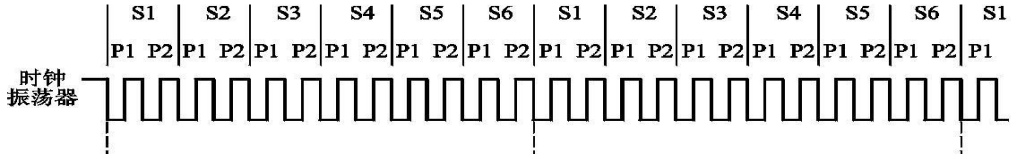
$$T_{\text{时钟周期}} = \frac{1}{F_{\text{osc 晶振 频率}}}$$



- **机器周期——CPU完成一个基本操作所需时间**

  51单片机的一个机器周期一般由6个状态周期构成，二个节拍为一个状态周期S，故一个机器周期是12个节拍即12个时钟周期构成。

$$T_{\text{机器周期}} = 12 * T_{\text{时钟周期}} = 12 * \frac{1}{F_{osc\ \text{晶振频率}}}$$



若晶振频率fosc=6MHz，机器周期=12*时钟周期=$12*\frac{1}{Fosc}$=$12*\frac{1}{6}$ us=2 us

若晶振频率为12MHz，机器周期是多少？

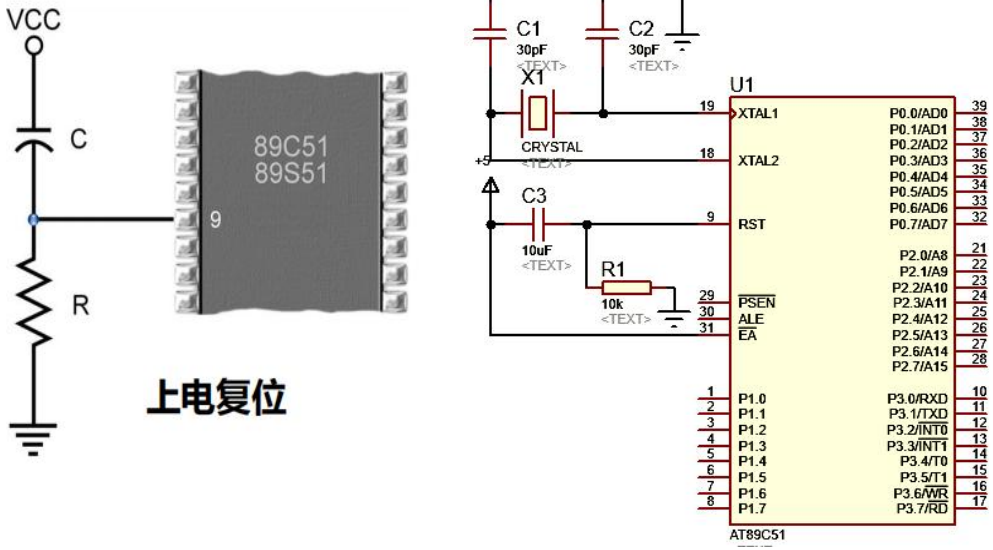
复位引脚及复位电路

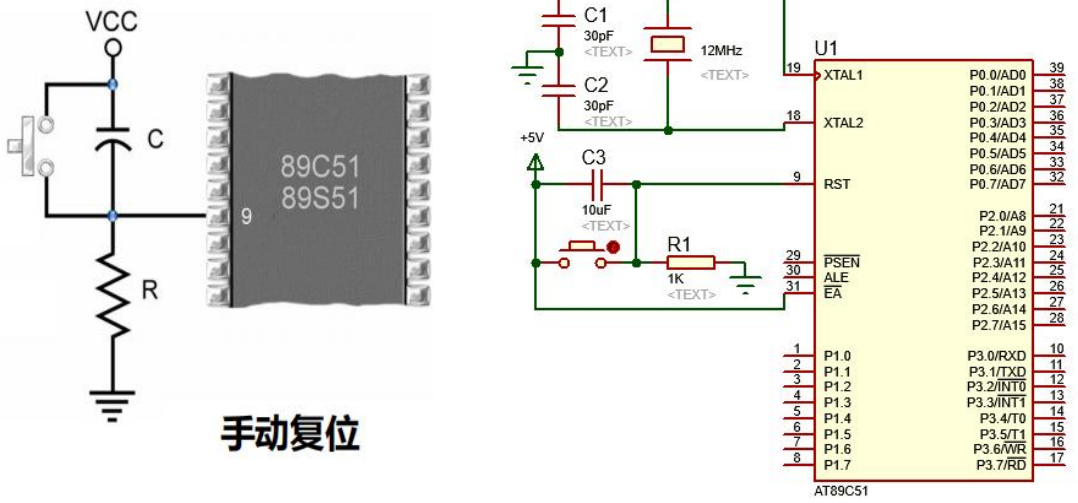
单片机本身是不能自动进行复位的，必须配合相应的外部电路才能实现。

- 最常见的复位电路是**RST**引脚接阻容电路复位。

RST：复位输入。

- 当输入连续两个机器周期以上高电平时为有效，用来完成单片机的复位初始化操作。





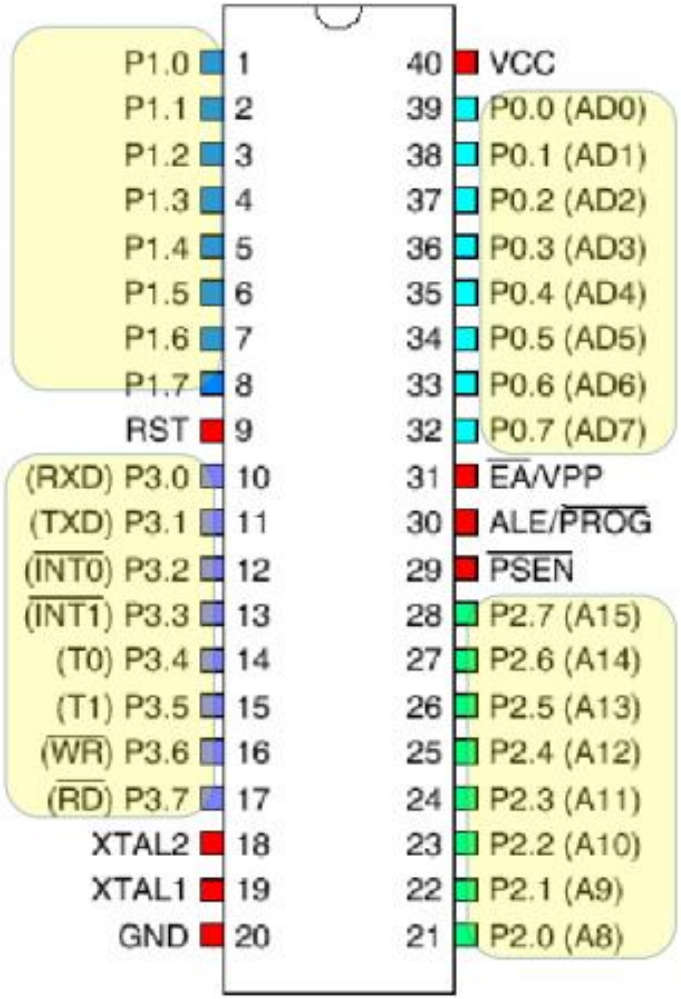
STC89C52单片机的复位状态

- 复位是单片机的初始化操作。

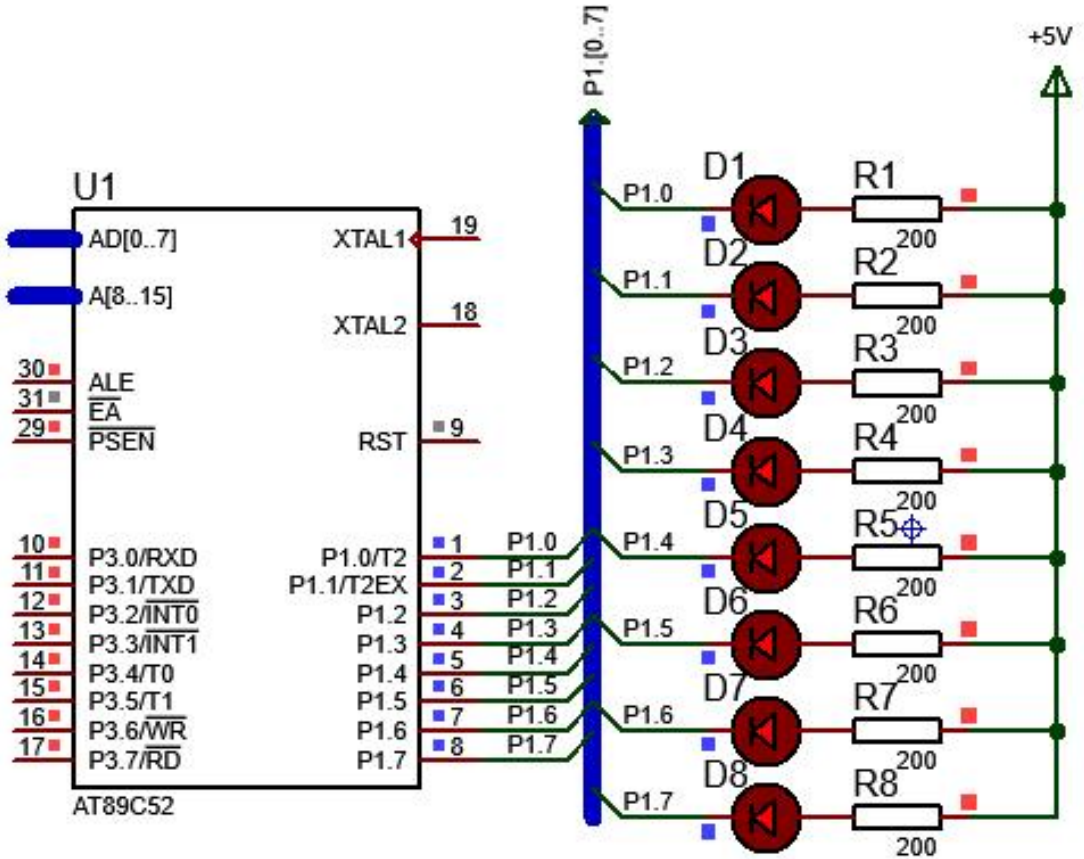
  单片机启动运行时需要使CPU和系统部件处于一个确定的初始状态，并从这个状态开始工作。

寄存器	初始状态	寄存器	初始状态
PC	0000H	TMOD	00H
Acc	00H	TCON	00H
PSW	00H	TH0	00H
B	00H	TL0	00H
SP	07H	TH1	00H
DPTR	0000H	TL1	00H
P0-P3	FFH	SCON	xxxx xxxxB
IP	xxx0 0000B	PCON	0xxx 0000B
IE	0xx0 0000B	AUXR	xxxx 0xx0B
DP0L	00H	AUXR1	xxxx xxx0B
DP0H	00H	WDTRST	xxxx xxxxB
DP1L	00H		
DP1H	00H		

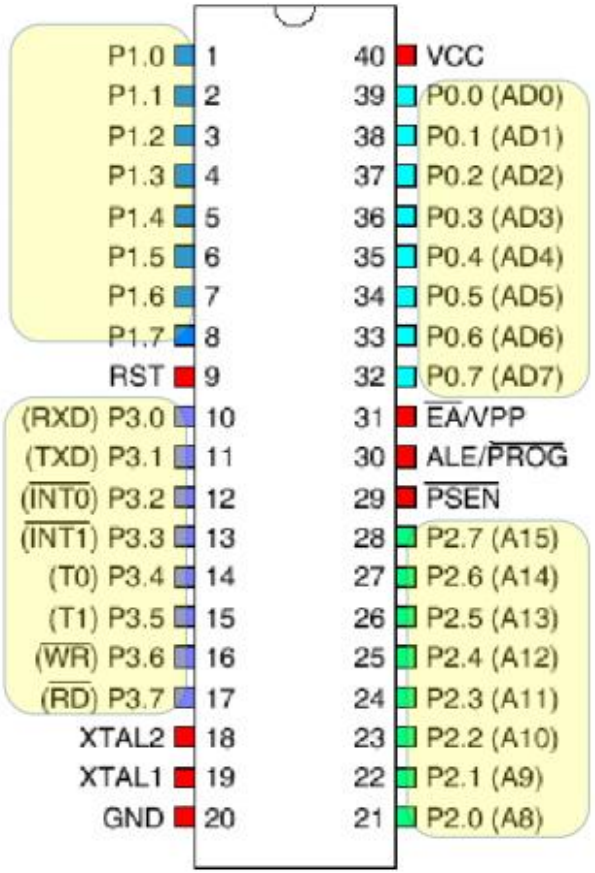
STC89C52单片机I/O口

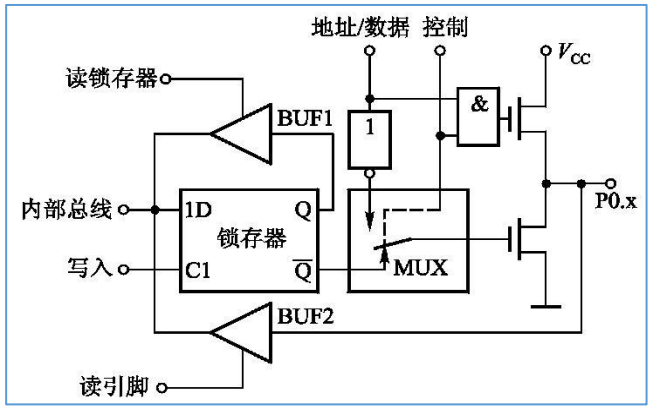


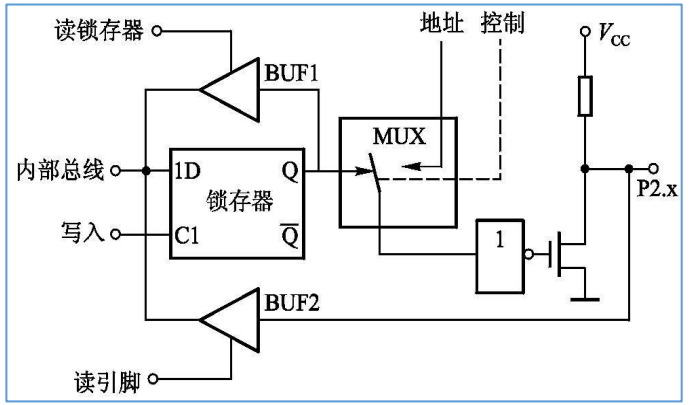
P1口常作通用I/O口使用

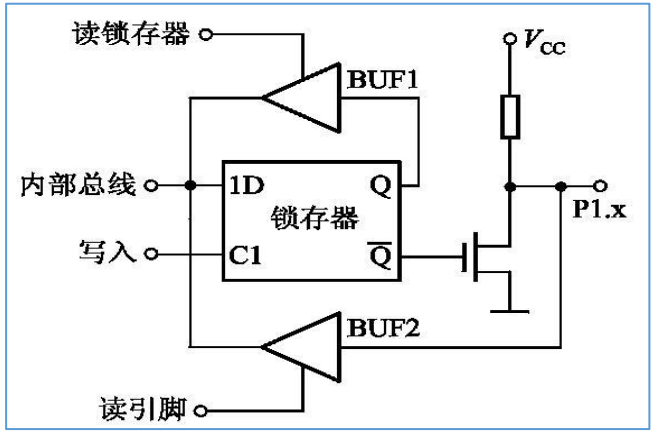


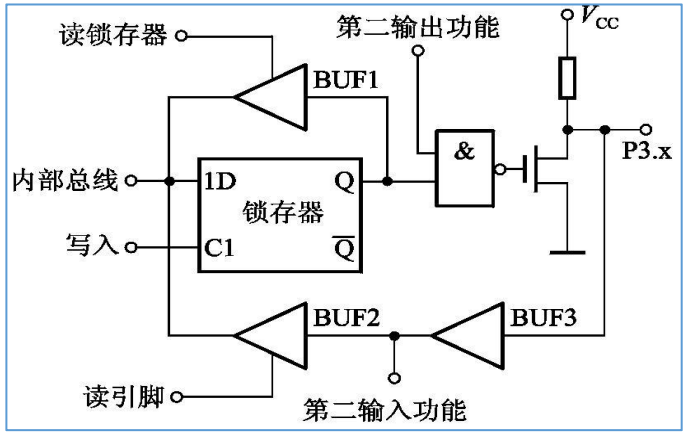
STC89C52单片机I/O口











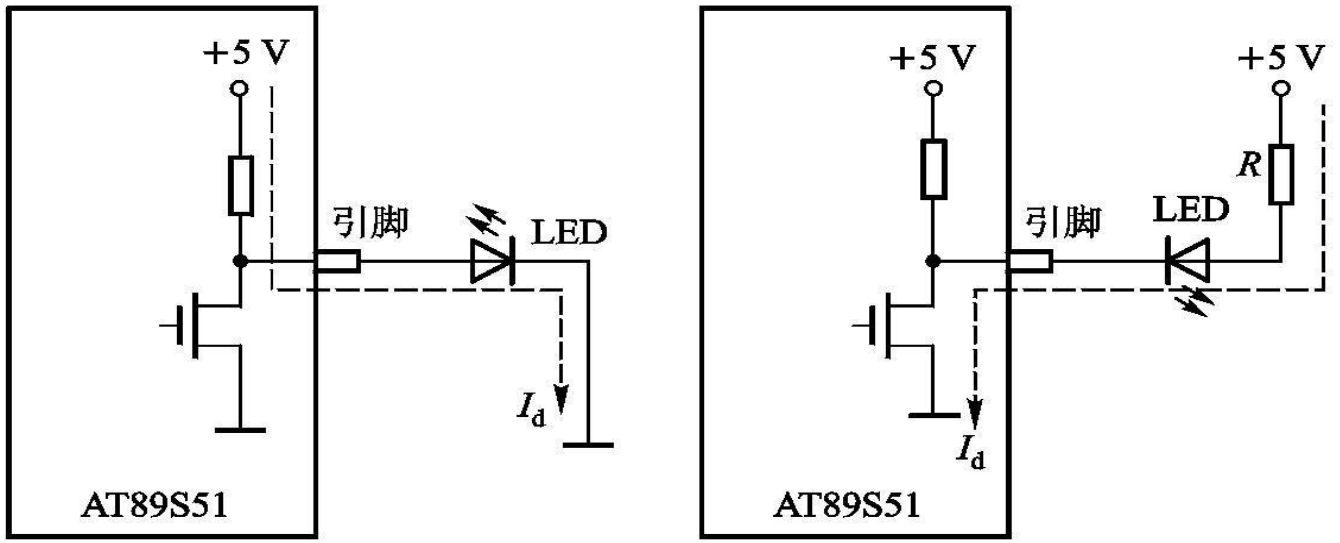
- P1口仅能用作通用I/O口,
- P0、P2、P3也可作通用I/O口

- 作通用I/O口时，P0口必须外接上拉电阻；
- P1、P2、P3口不需要外接上拉电阻；

I/O驱动能力说明

- 当口线输出为1时允许外部装置将其拉低，但驱动能力很弱，只提供μA级电流。
- 当口线输出为0时低电平允许提高，驱动能力很强，可提供mA级的灌电流。

即：　高电平输出，则强行从端口输出的电流 I_d，易造成单片机端口的损坏。

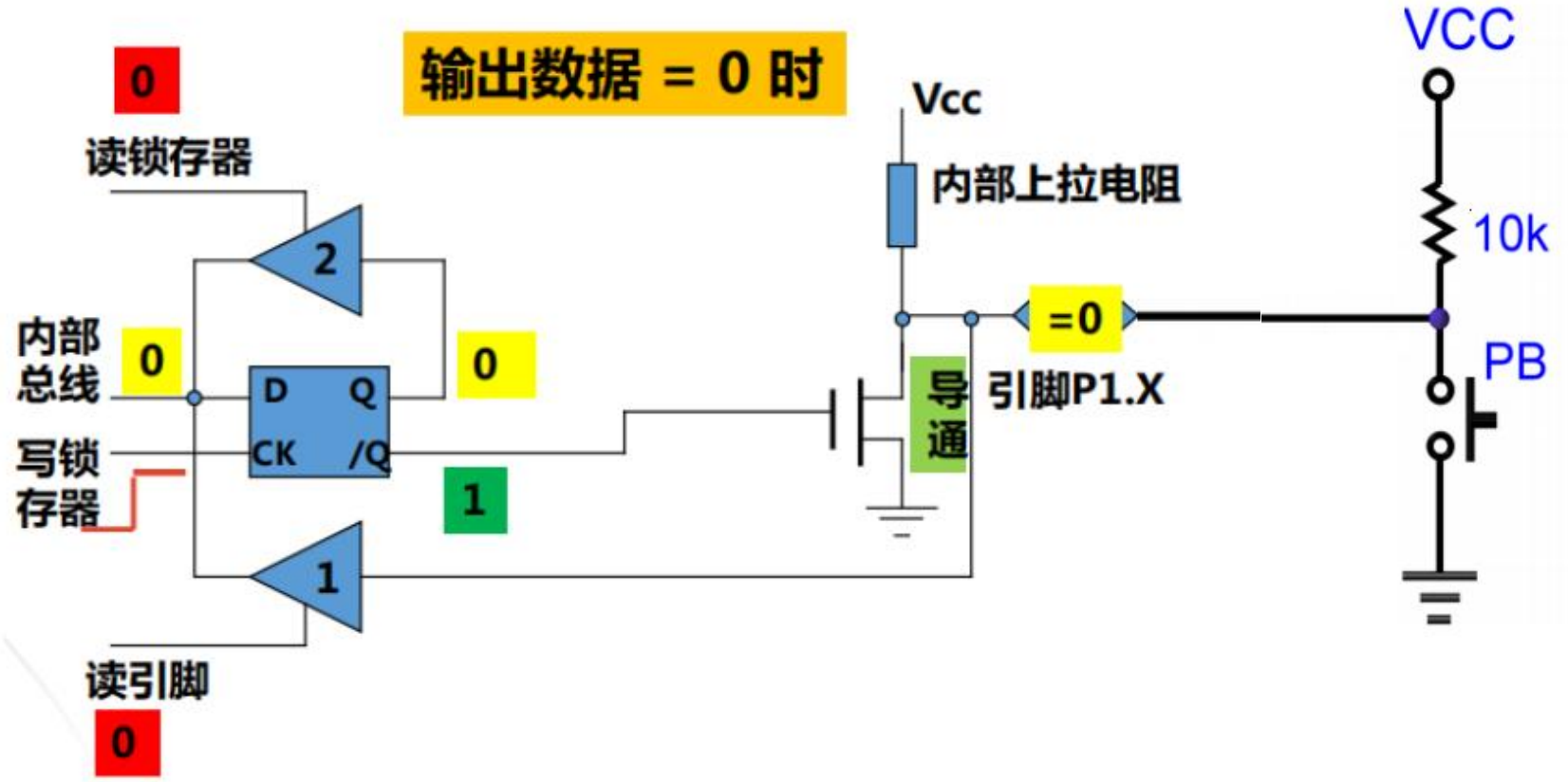


（a）不恰当：高电平驱动　　（b）恰当：低电平驱动

要想获得较大的驱动能力，采用低电平输出。

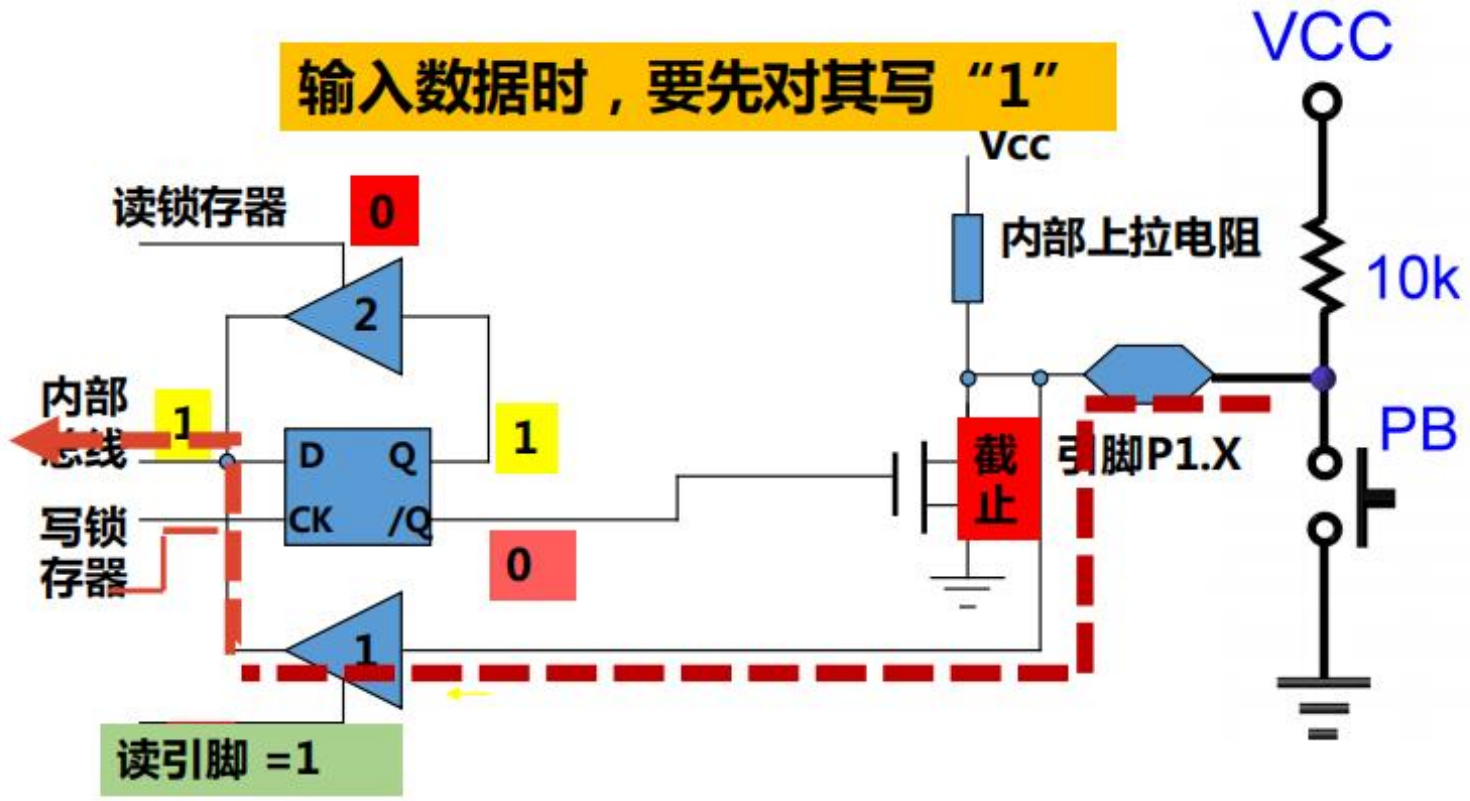
通用I/O口输入使用时的注意点——以P1口为例

例：若先有指令使P1.x或P1口输出0，然后再读取读取P1.x口或P1口状态，此时读到的值是多少？

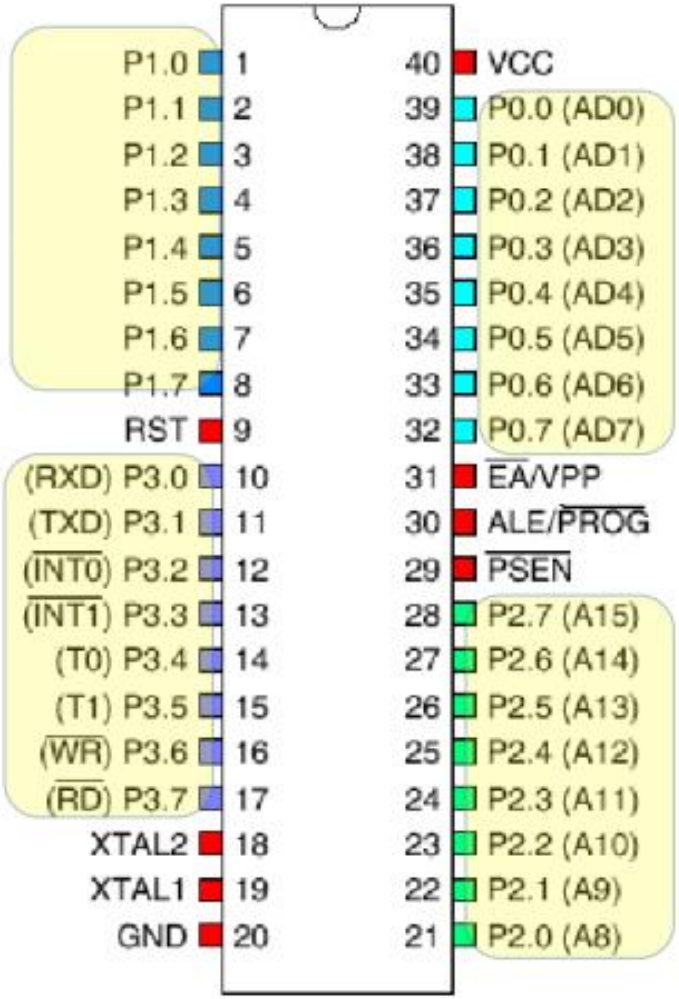


通用I/O口输入使用时的注意点——以P1口为例

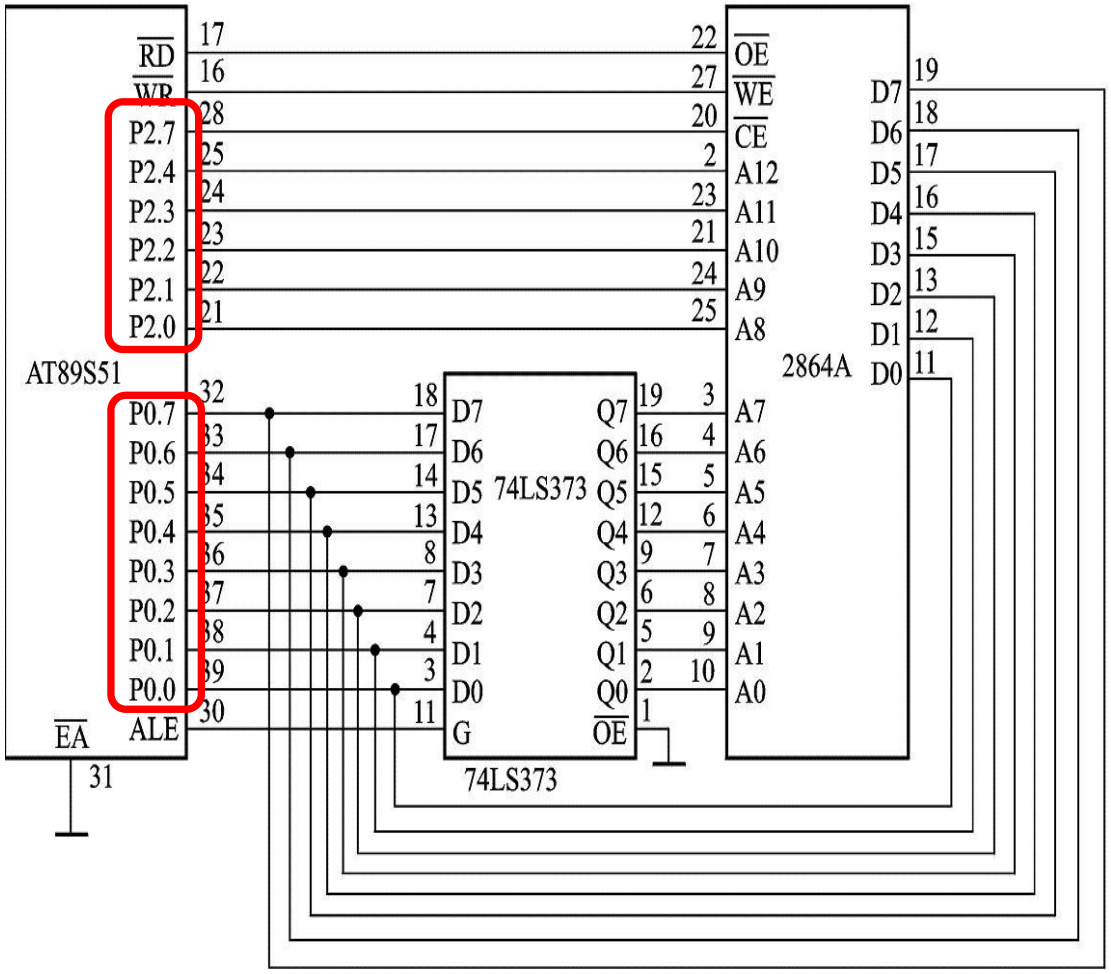
读取Px口状态之前一定要先置1，才能正确读取Px口状态 。



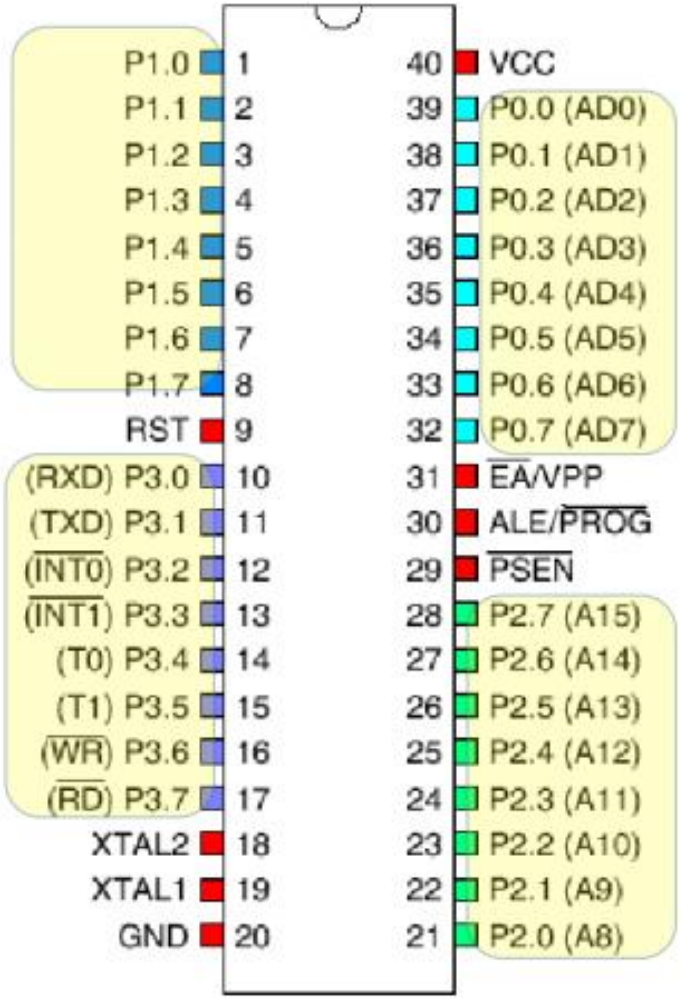
STC89C52单片机I/O口

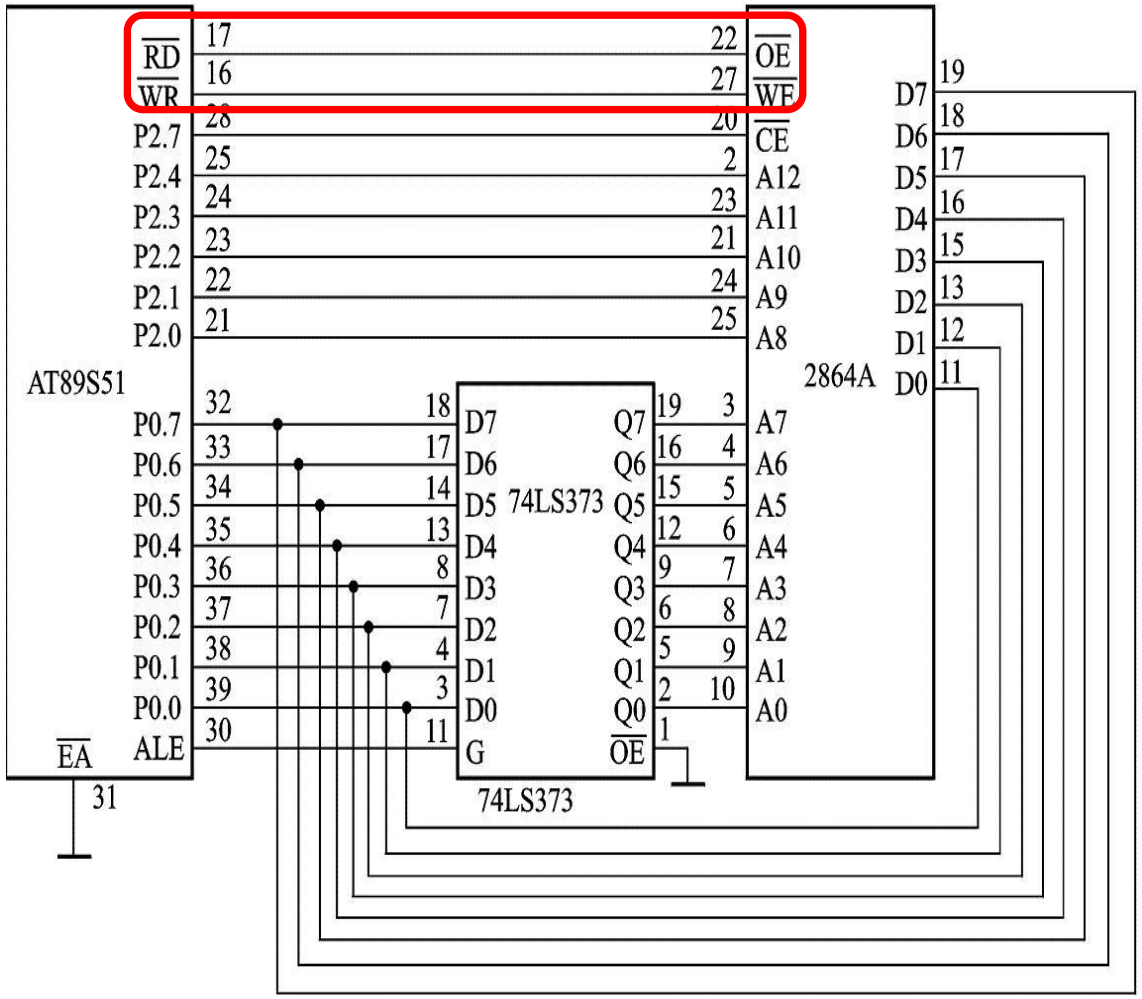


- **P0口和P2口常作总线扩展**

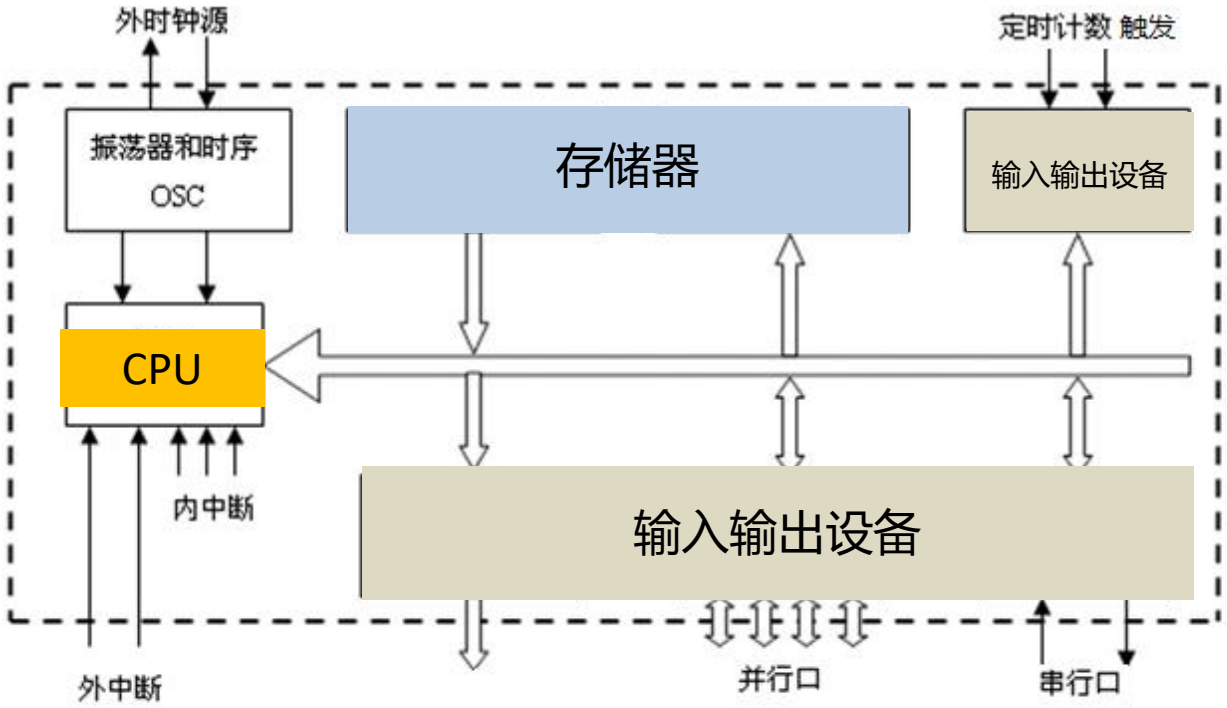


P3口常用第二功能


P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD) P3.0 10
(TXD) P3.1 11
(INT0) P3.2 12
(INT1) P3.3 13
(T0) P3.4 14
(T1) P3.5 15
(WR) P3.6 16
(RD) P3.7 17
XTAL2 18
XTAL1 19
GND 20
40 VCC
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 EA/VPP
30 ALE/PROG
29 PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)


AT89S51
RD 17
WR 16
P2.7 28
P2.4 25
P2.3 24
P2.2 23
P2.1 22
P2.0 21
P0.7 32
P0.6 33
P0.5 34
P0.4 35
P0.3 36
P0.2 37
P0.1 38
P0.0 39
EA
ALE 30
31
74LS373
D7 18
D6 17
D5 14
D4 13
D3 8
D2 7
D1 4
D0 3
G 11
Q7 19
Q6 16
Q5 15
Q4 12
Q3 9
Q2 6
Q1 5
Q0 2
OE 1
74LS373
2864A
OE 22
WE 27
CE 20
A12 2
A11 23
A10 21
A9 24
A8 25
A7 3
A6 4
A5 5
A4 6
A3 7
A2 8
A1 9
A0 10
D7 19
D6 18
D5 17
D4 16
D3 15
D2 13
D1 12
D0 11

单片机存储器结构



一片半导体硅片集成有：

中央处理单元（CPU） ：进行运算、控制；

存储器（RAM：数据存储、ROM ：程序存储）；

输入/输出设备（并行/串行口、定时计数器、中断系统等）



具有微型计算机的属性，因而被称为单片微型计算机，简称单片机。

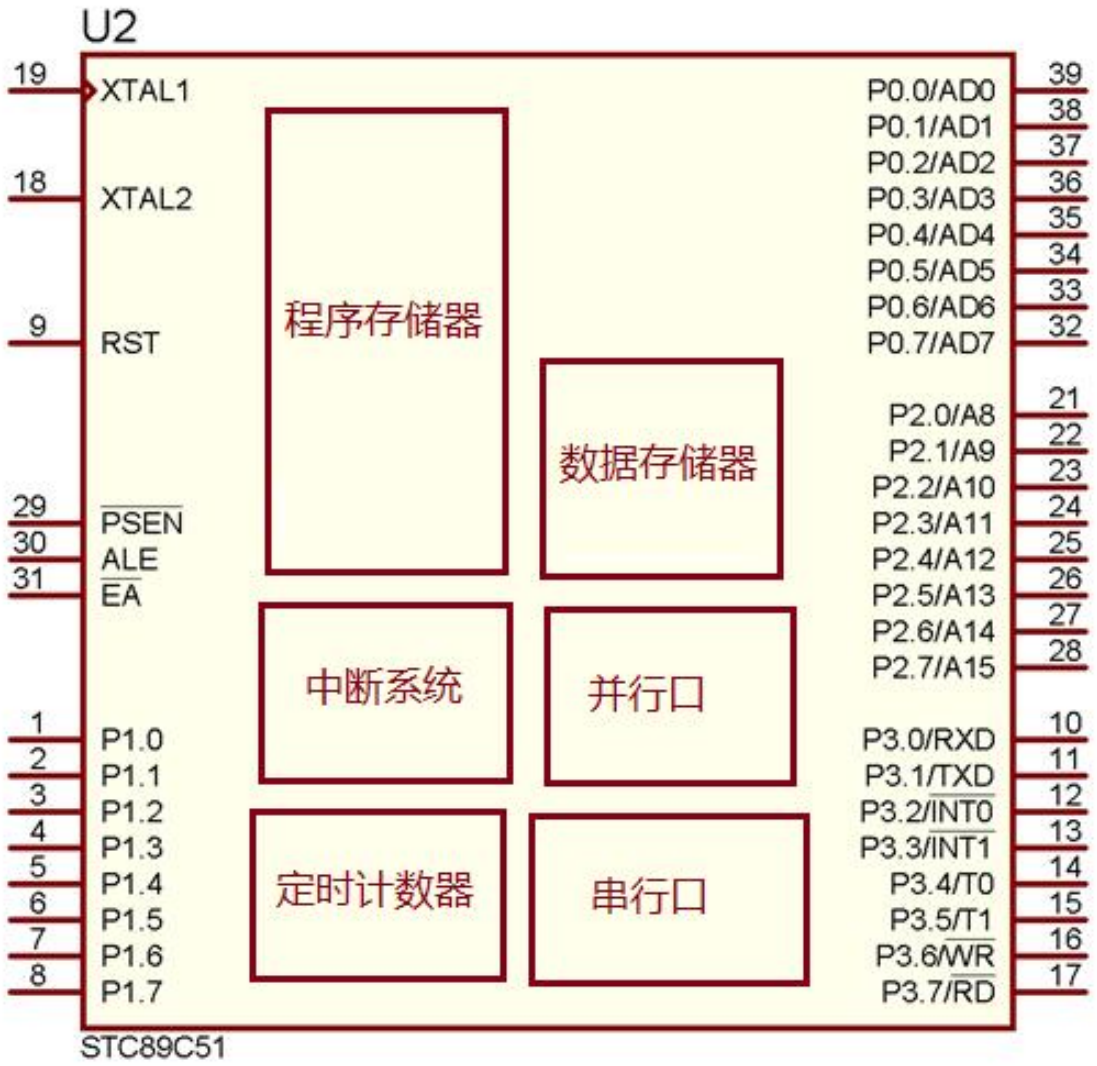
单片机存储器结构

重点

- STC单片机的指令的存放机构——存储器结构
  - 程序存储器
  - 数据存储器

- 单片机C51编程语言的存储类型和数据类型
- 单片机汇编语言的书写格式和编码格式

STC89系列单片机内部可编程资源



prog1.c

```
1 void main()
2 {
3   char i=1, j=2, k;
4   k=i+j;
5 }
```

代码放在哪里？



单片机汇编指令格式

- 51单片机指令是CPU按照人们的意图来完成某种操作的命令。
- 51单片机指令系统是某种微处理器所能执行全部指令的集合。

51汇编指令书写格式——四分段格式

[标号：] 操作码 [操作数1] [，操作数2] [，操作数3] [；注释]

prog1.s*

```
1          ORG 0000H
2 MAIN:    MOV A, 40H   ;(40H)->A
3          ADD A, 41H   ;(41H)+A->A
4          MOV 41H, A   ;A->(41H)
5          END
```

各段之间不同的分隔符“：号 ，号 ；号和空格”，特别注意不能使用中文输入法。

基本语法规则

1. 标号：该语句的符号地址
   - 由1～8个字符组成，第一个字符必须是字母，其余的可以是字母、数字或下划线“_”。
   - 不能使用已经定义的符号作为标号，同一标号在一个程序中只能定义一次，
     标号的有无取决于是否需要被访问。
2. 操作码：是汇编语言指令中唯一不能空缺的部分。
3. 注释：用于解释指令或程序的含义。
   - 每行都必须以分号开头；不会产生机器代码。
4. 操作数：指令的操作数或操作数地址。
   - 十进制数需加后缀D，如 01D。通常都省略。
   - 二进制数需加后缀B，如 01B。
   - 十六进制数需加后缀H，如 01H。若以字符A～F开头，前面要加 0 。
   - 若操作数是寄存器时，常用代号表示。如累加器A(或Acc)表示。

```
1           ORG  0000H
2   MAIN:   MOV  A,  0FAH    ;A <- (0FAH)
3           ADD  A,  0FBH    ;A <- A+(0FBH)
4           MOV  0FAH,A      ;(0FAH) <- A
5           END
```

```
1           ORG  0000H
2   MAIN:   MOV  A,  FAH     ;A <- (FAH)
3           ADD  A,  FBH     ;A <- A+(FBH)
4           MOV  FAH,A       ;(FAH) <- A
5           END
```

教材P48

符号说明：

- MOV A,Rn

例：MOV A,R0

- MOV A,direct

例：MOV A,40H

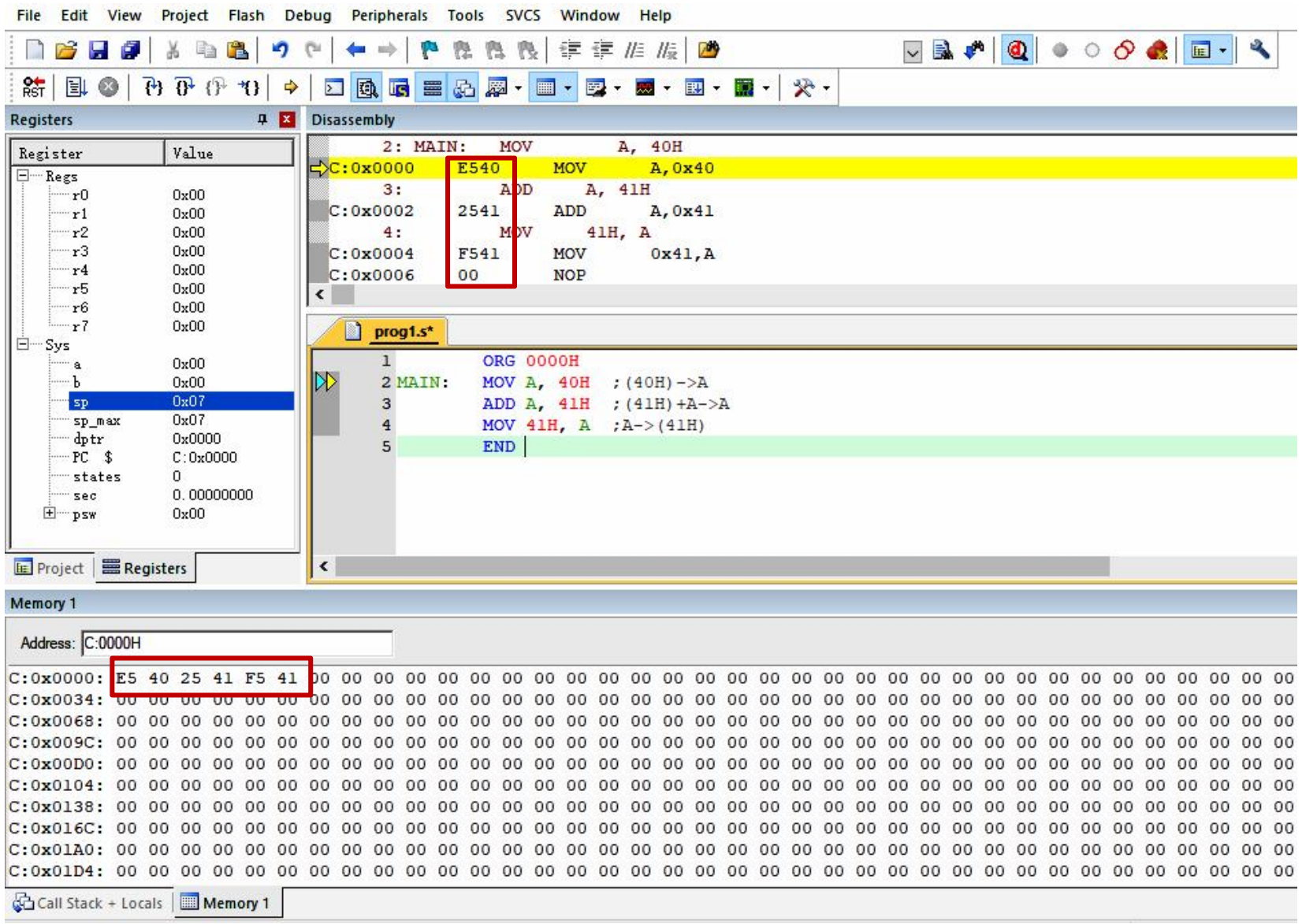
- MOV A,@Ri

例：MOV A,@R0

- MOV A,#data

例：MOV A,#40H

分类	助记符		说明	字节数	机器周期	指令代码(机器代码)
1.数据传送类	MOV	A,Rn	寄存器内容传送到累加器 A	1	1	E8H~EFH
	MOV	A,direct	直接寻址字节传送到累加器	2	1	E5H,direct
	MOV	A,@Ri	间接寻址 RAM 传送到累加器	1	1	E6H~E7H
	MOV	A,#data	立即数传送到累加器	2	1	74H,data
	MOV	Rn,A	累加器内容传送到寄存器	1	1	F8H~FFH
	MOV	Rn,direct	直接寻址字节传送到寄存器	2	2	A8H~AFH,direct
	MOV	Rn,#data	立即数传送到寄存器	2	1	78H~7FH,data
	MOV	direct,A	累加器内容传送到直接寻址字节	2	1	F5H,direct
	MOV	direct,Rn	寄存器内容传送到直接寻址字节	2	2	88H~8FH,direct
	MOV	direct1,direct2	直接寻址字节 2 传送到直接寻址字节 1	3	2	85H,direct2,direct1
	MOV	direct,@Ri	间接寻址 RAM 传送到直接寻址字节	2	2	86H~87H,direct
	MOV	direct,#data	立即数传送到直接寻址字节	3	2	75H,direct,data
	MOV	@Ri,A	累加器传送到间接寻址 RAM	1	1	F6H~F7H
	MOV	@Ri,direct	直接寻址字节传送到间接寻址 RAM	2	2	A6H~A7H,direct
	MOV	@Ri,#data	立即数传送到间接寻址 RAM	2	1	76H~77H,data
	MOV	DPTR,#data16	16 位常数装入到数据指针	3	2	90H,dataH,dataL


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0x00
b 0x00
sp 0x07
sp_max 0x07
dptr 0x0000
PC $ C:0x0000
states 0
sec 0.00000000
psw 0x00
Project Registers
Disassembly
2: MAIN: MOV A, 40H
C:0x0000 E540 MOV A,0x40
3: ADD A, 41H
C:0x0002 2541 ADD A,0x41
4: MOV 41H, A
C:0x0004 F541 MOV 0x41,A
C:0x0006 00 NOP
prog1.s*
1 ORG 0000H
2 MAIN: MOV A, 40H ;(40H)->A
3 ADD A, 41H ;(41H)+A->A
4 MOV 41H, A ;A->(41H)
5 END
Memory 1
Address: C:0000H
C:0x0000: E5 40 25 41 F5 41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0034: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x009C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x00D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0104: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0138: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x016C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x01A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x01D4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Call Stack + Locals Memory 1

指令编码格式

- 单片机指令系统采用的助记符格式，与机器码有一一对应。

```
        ORG   0000H
MAIN:   MOV   A, 40H
        ADD   A, 41H
        MOV   41H, A
        END
```

```
0000H   E540
0002H   2541
0004H   F541
```

- 机器码通常由两部分组成：

  如：**MOV A,40H**机器码为**E540H**，**E5**为操作码，**40**为操作数

  - 操作码—指令进行什么操作。
  - 操作数—指令操作的对象。可以是数据或数据的地址。

3.1.2 指令编码格式

- 按指令所生成机器码的字节长度来分有：

（1）单字节指令：操作码和操作数同在一个字节中。

如：**DEC A**的机器码为**14H**，**INC R1**的机器码为**09H**。

```
     5:                    DEC      A
C:0x0006     14        DEC       A
     6:                    INC R1
C:0x0007     09        INC       R1
```

（2）双字节指令：一个字节为操作码，另一个字节是操作数。

如：**MOV A,#60H**机器码为**7460H**。

```
     7:                    MOV A,#60H
C:0x0008     7460      MOV       A,#0x60
```

（3）三字节指令：操作码占一个字节，操作数占二个字节。

如：**MOV 10H,#60H**机器码为**751060H**。

```
     8:                    MOV 10H,#60H
C:0x000A     751060    MOV       0x10,#0x60
```

1. 单字节指令编码格式有两种：

（1）格式1：

位　7 6 5 4 3 2 1 0

字节 | opcode | 仅为操作码，操作数隐含在其中

如：DEC　A的指令编码为14H

0001 0100　操作码14H，功能是累加器的内容减1.

（2）格式2：

位　7 6 5 4 3 2 1 0

字节 | opcode | rrr | 高5位操作码，低3位寄存器的编号

如：INC　R1指令编码为09H，即0000 1001

0000 1	001

高5位opcode=00001B是操作码，

低3位rrr=001B是寄存器R1对应的编号1

2.双字节指令

格式： 位 7 6 5 4 3 2 1 0

字节		
	opcode	第一字节为操作码
	data或direct	第二字节为操作数或存放操作数的地址

如：MOV A，#60H的机器码为74 60H

0111 0100	操作码74H，功能是将立即数传送到累加器A
0110 0000	操作数60H，即要传送到累加器A的立即数

```
      7:                    MOV A,#60H
C:0x0008      7460      MOV       A,#0x60
```

3.三字节指令

格式： 位 7 6 5 4 3 2 1 0

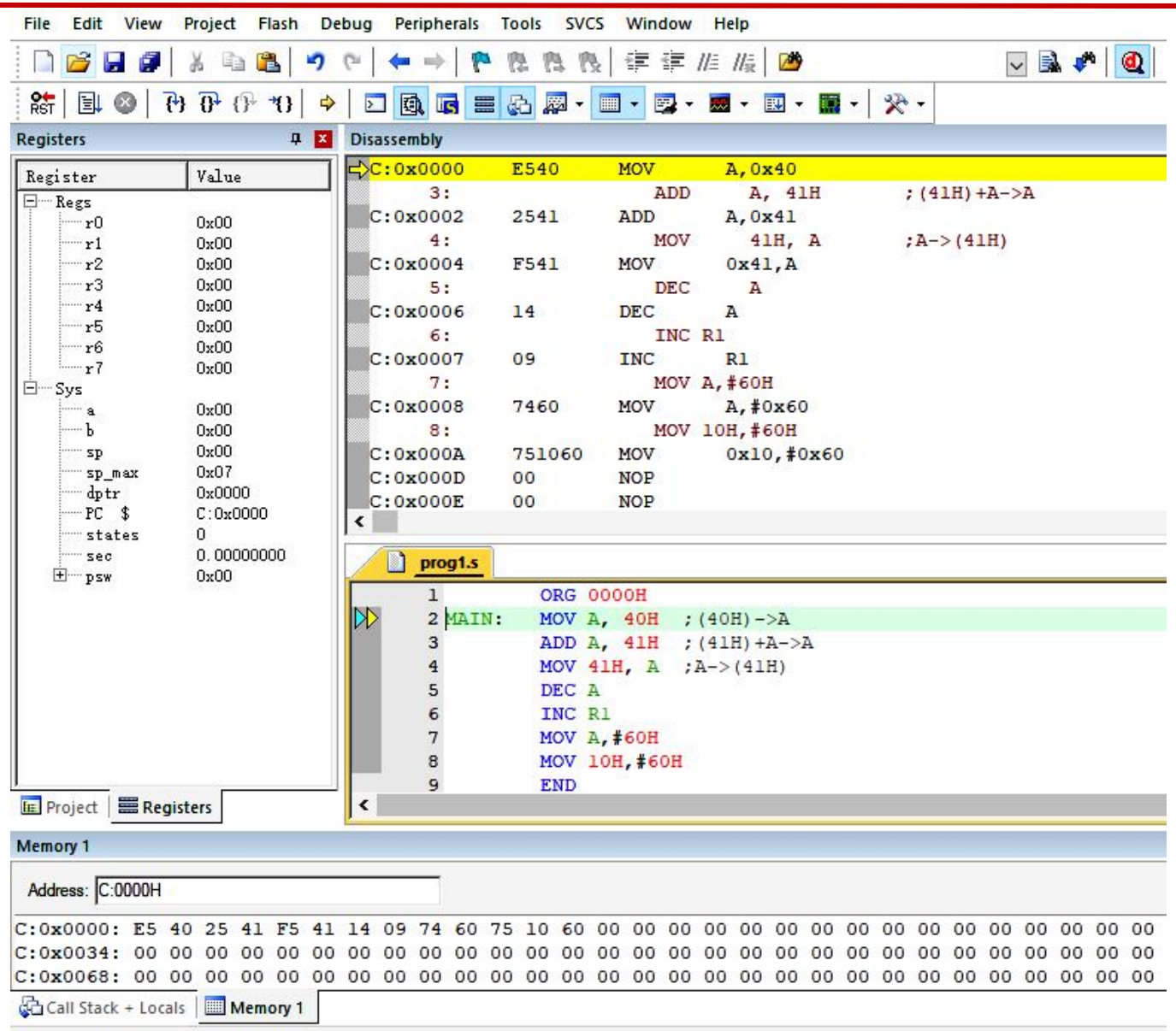
字节

opcode	操作码
direct	操作数或存放操作数的地址
data或 direct	

如：MOV 10H，#60H的指令代码为75 10 60H

0111 0101	操作码75H，功能是将立即数传送到直接地址单元
0001 0000	direct=10H是目标操作数对应的存放地址
0110 0000	data=60H是对应的立即数

```
      8:                        MOV 10H,#60H
C:0x000A     751060    MOV         0x10,#0x60
```

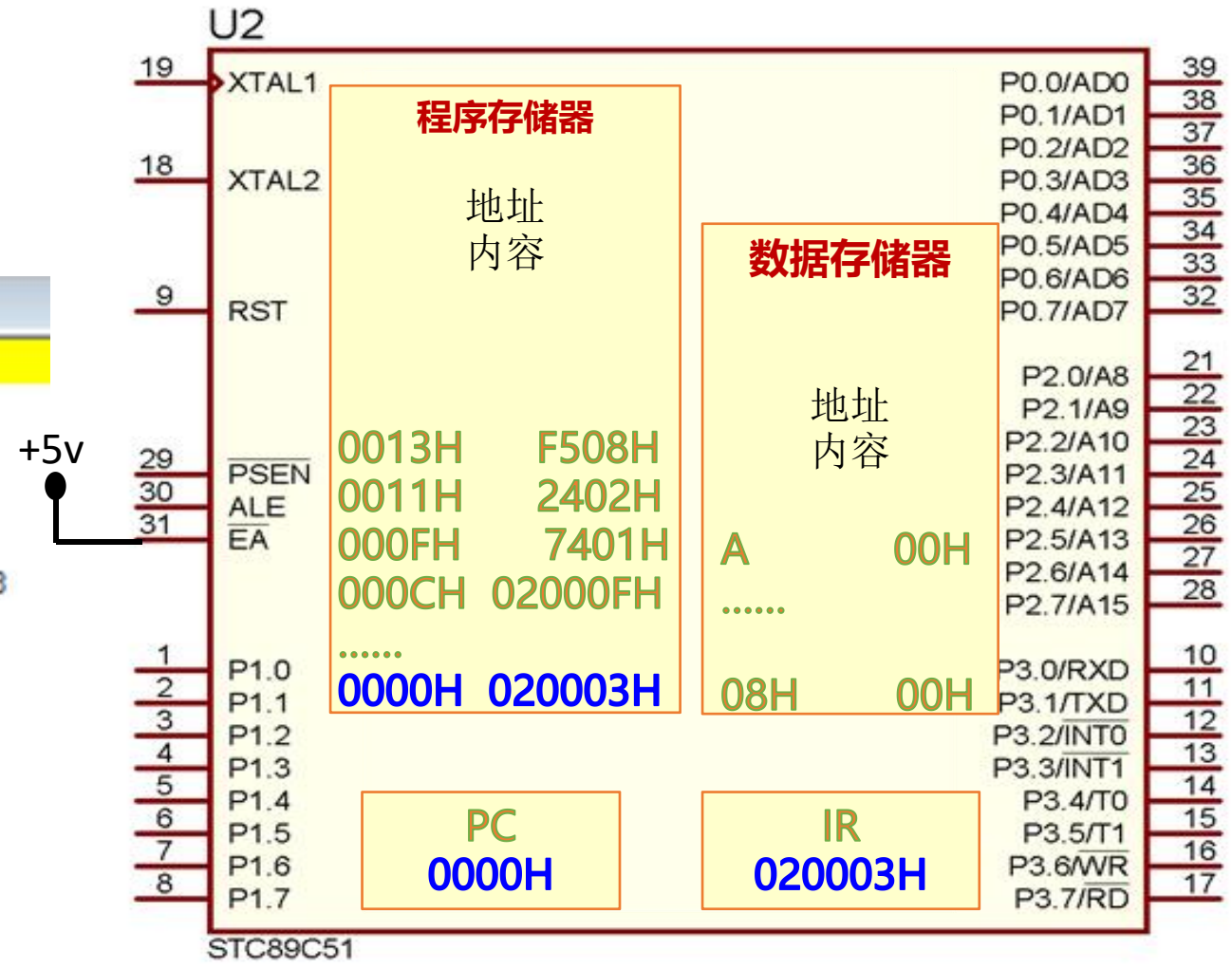

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0x00
b 0x00
sp 0x00
sp_max 0x07
dptr 0x0000
PC $ C:0x0000
states 0
sec 0.00000000
psw 0x00
Project Registers
Disassembly
C:0x0000 E540 MOV A,0x40
3: ADD A, 41H ;(41H)+A->A
C:0x0002 2541 ADD A,0x41
4: MOV 41H, A ;A->(41H)
C:0x0004 F541 MOV 0x41,A
5: DEC A
C:0x0006 14 DEC A
6: INC R1
C:0x0007 09 INC R1
7: MOV A,#60H
C:0x0008 7460 MOV A,#0x60
8: MOV 10H,#60H
C:0x000A 751060 MOV 0x10,#0x60
C:0x000D 00 NOP
C:0x000E 00 NOP
prog1.s
1 ORG 0000H
2 MAIN: MOV A, 40H ;(40H)->A
3 ADD A, 41H ;(41H)+A->A
4 MOV 41H, A ;A->(41H)
5 DEC A
6 INC R1
7 MOV A,#60H
8 MOV 10H,#60H
9 END
Memory 1
Address: C:0000H
C:0x0000: E5 40 25 41 F5 41 14 09 74 60 75 10 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0034: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Call Stack + Locals Memory 1

prog1.c

```
1 void main()
2 {
3   char i=1,j=2,k;
4   k=i+j;
5 }
```

Disassembly

```
C:0x0000    020003    LJMP    C:0003
C:0x0003    787F      MOV     R0,#0x7F
C:0x0005    E4        CLR     A
C:0x0006    F6        MOV     @R0,A
C:0x0007    D8FD      DJNZ    R0,C:0006
C:0x0009    758108    MOV     SP(0x81),#0x08
C:0x000C    02000F    LJMP    main(C:000F)
      2: {
      3:         char i=1,j=2,k;
      4:         k=i+j;
C:0x000F    7401      MOV     A,#0x01
C:0x0011    2402      ADD     A,#0x02
C:0x0013    F508      MOV     0x08,A
      5: }
```



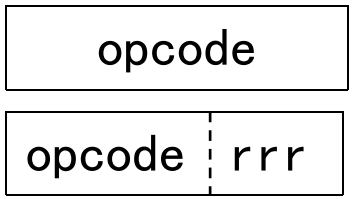
+5v

书写格式：

[标号：] 操作码 [操作数1] [，操作数2] [，操作数3] [；注释]

编码格式：

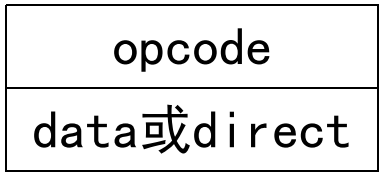
1.单字节指令



仅为操作码，操作数隐含在其中

高5位操作码，低3位寄存器的编号

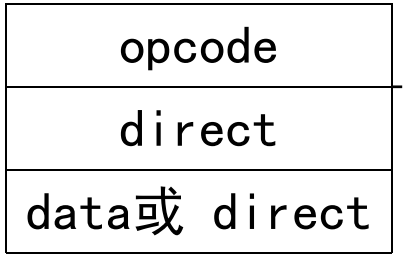
2.双字节指令



第一字节为操作码

第二字节为操作数或存放操作数的地址

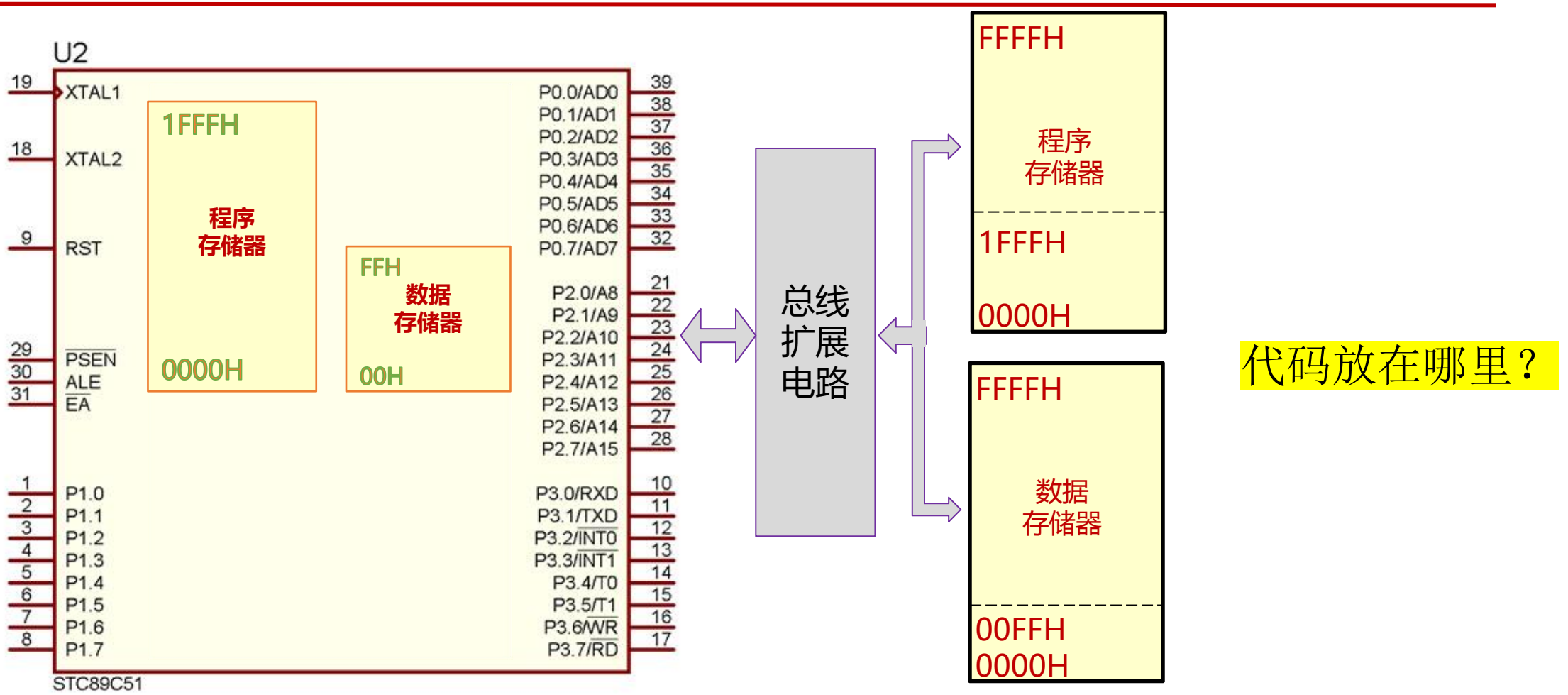
3.三字节指令



操作码

操作数或存放操作数的地址

STC89C52RC单片机存储器结构特点



代码放在哪里？

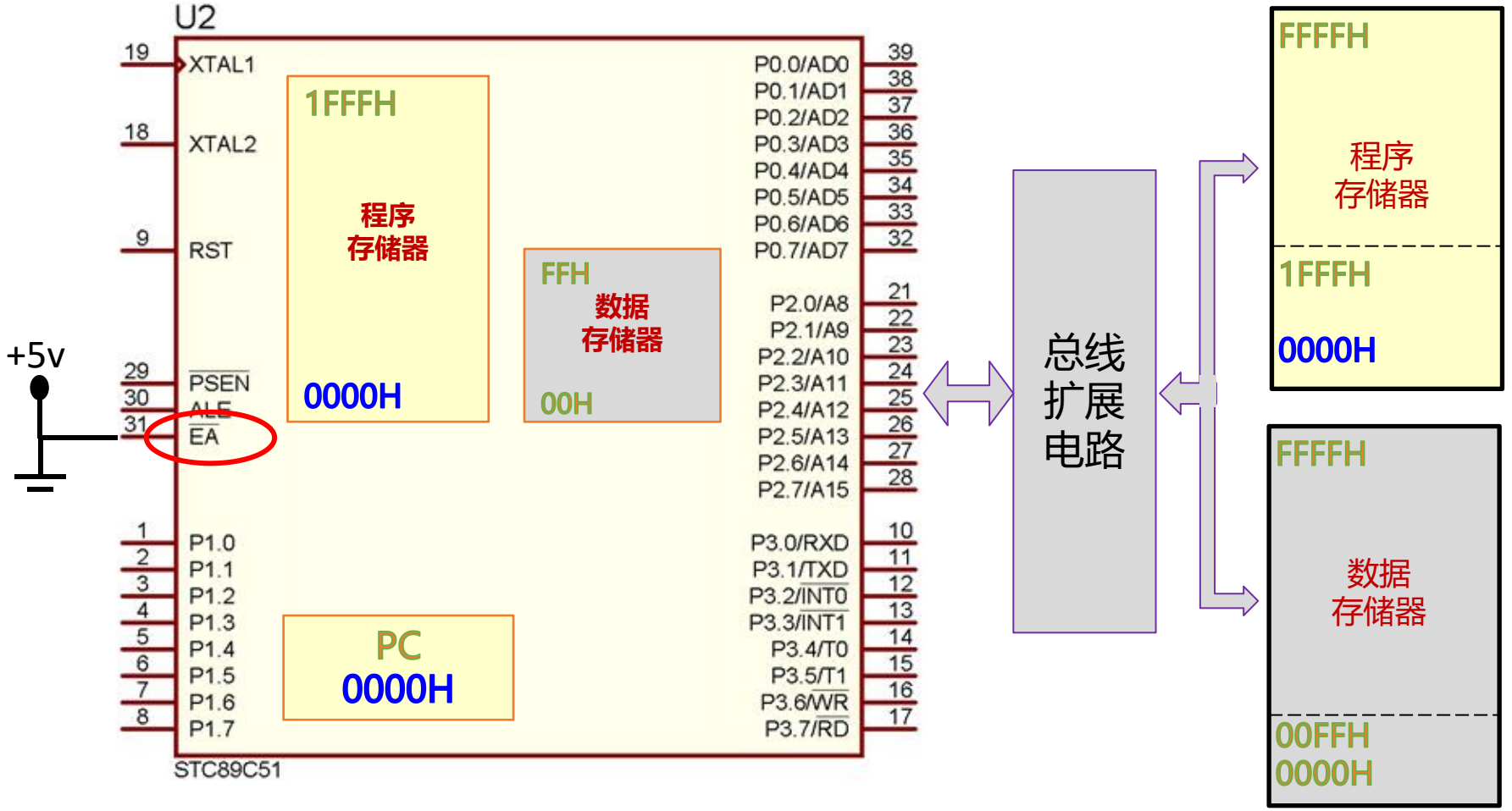
物理上有4个存储空间：片内和片外程序存储器、 片内和片外数据存储器

程序存储器和数据存储器分开（哈佛结构），并有各自的访问指令。

STC89C52单片机程序存储器

$\overline{EA}$ (Enable Address)： **外部程序存储器访问允许控制端。**

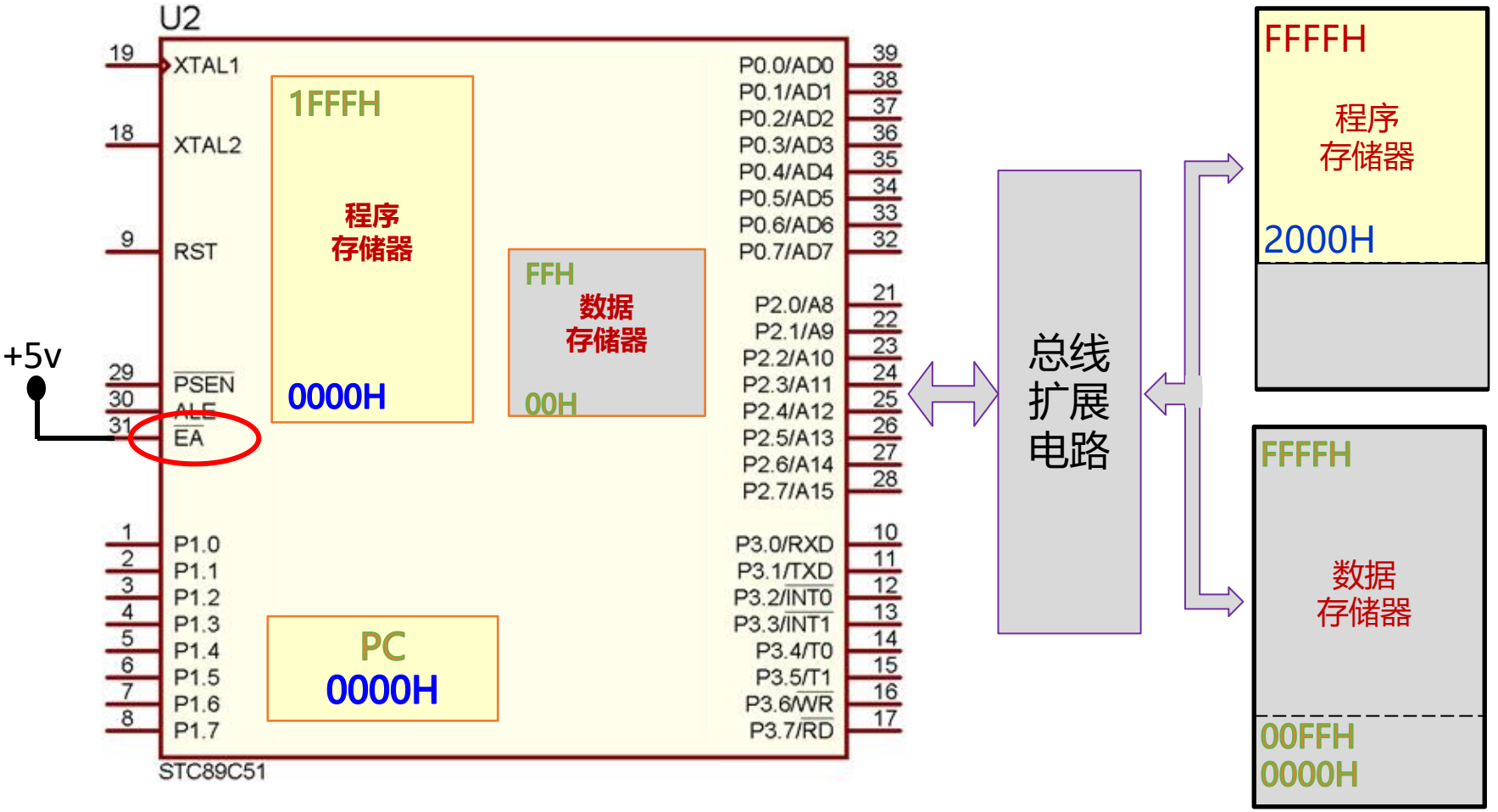
访问片内还是片外程序存储器，由 EA# 引脚电平确定。



STC89C52单片机程序存储器

EA#=1时，CPU从片内0000H开始取指令；

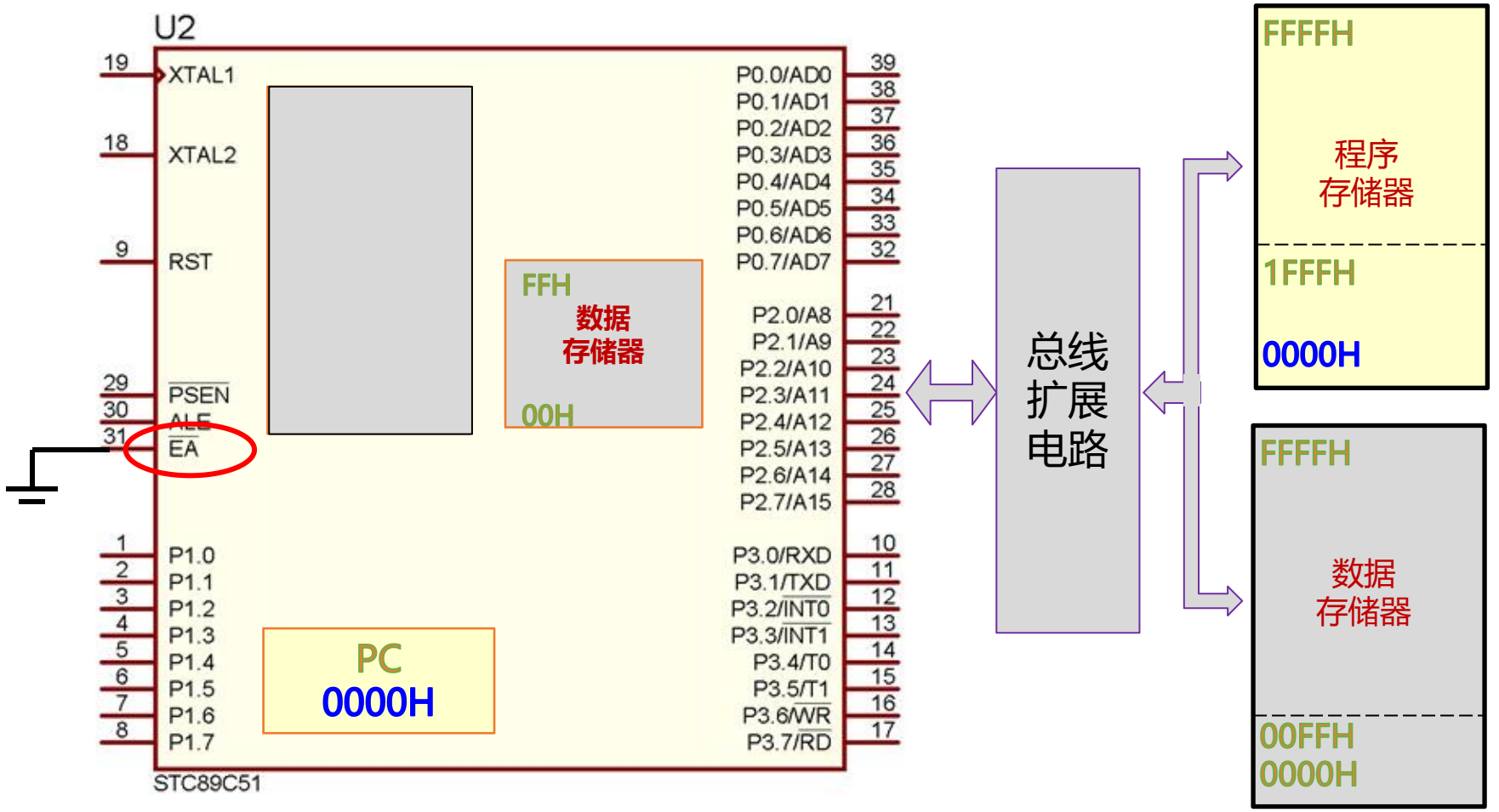
- 当PC<=1FFFH时，只访问片内Flash存储器；
- 当PC>1FFFH自动转向片外程序存储器2000H～FFFFH内的程序。



STC89C52单片机程序存储器

EA#=0时，CPU从片外0000H开始取指令；

- 只能执行片外程序存储器(0000H～FFFFH)中的程序，不理会片内程序存储器。

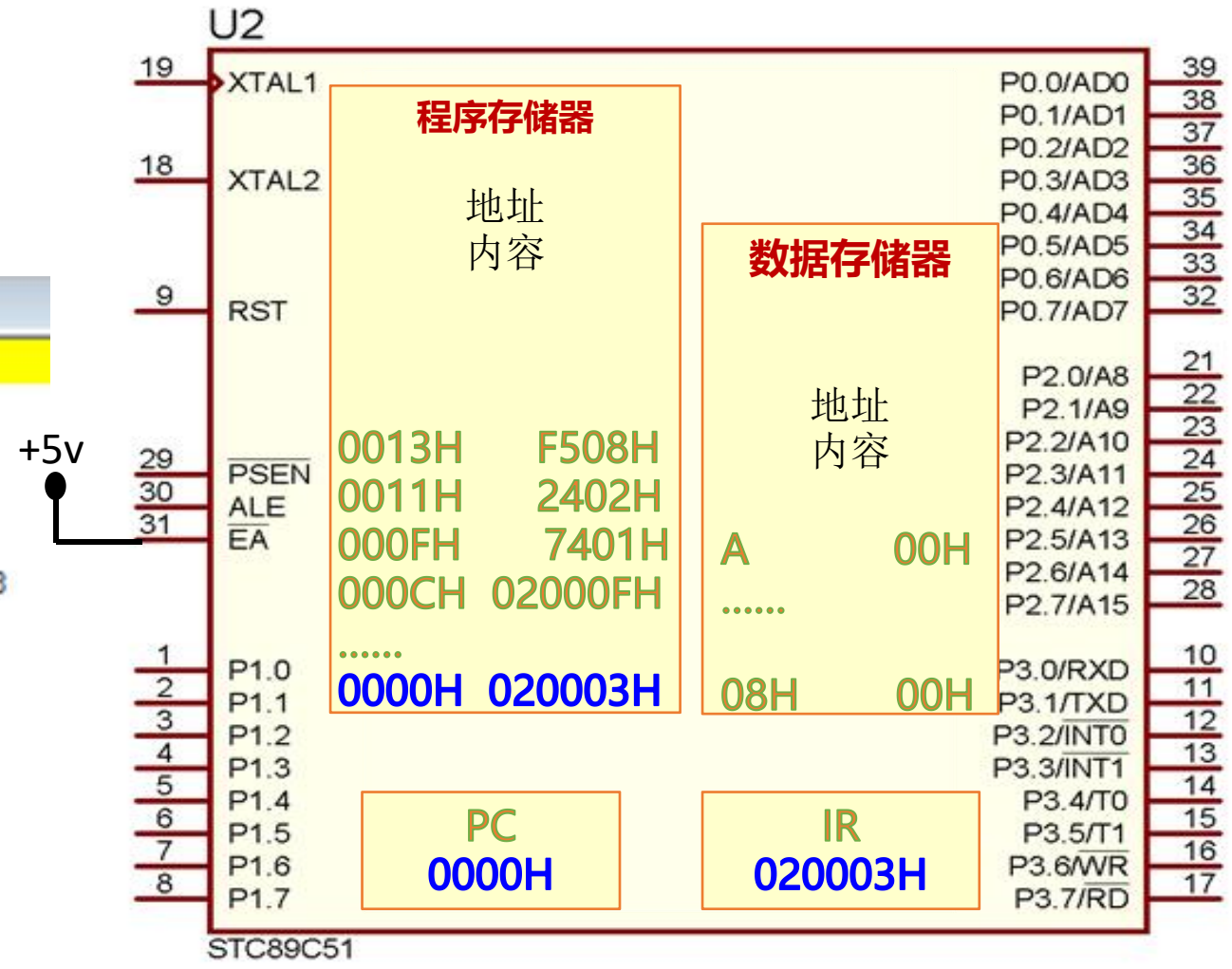


prog1.c

```
1 void main()
2 {
3   char i=1,j=2,k;
4   k=i+j;
5 }
```

Disassembly

```
C:0x0000    020003    LJMP    C:0003
C:0x0003    787F      MOV     R0,#0x7F
C:0x0005    E4        CLR     A
C:0x0006    F6        MOV     @R0,A
C:0x0007    D8FD      DJNZ    R0,C:0006
C:0x0009    758108    MOV     SP(0x81),#0x08
C:0x000C    02000F    LJMP    main(C:000F)
      2: {
      3:         char i=1,j=2,k;
      4:         k=i+j;
C:0x000F    7401      MOV     A,#0x01
C:0x0011    2402      ADD     A,#0x02
C:0x0013    F508      MOV     0x08,A
      5: }
```



+5v

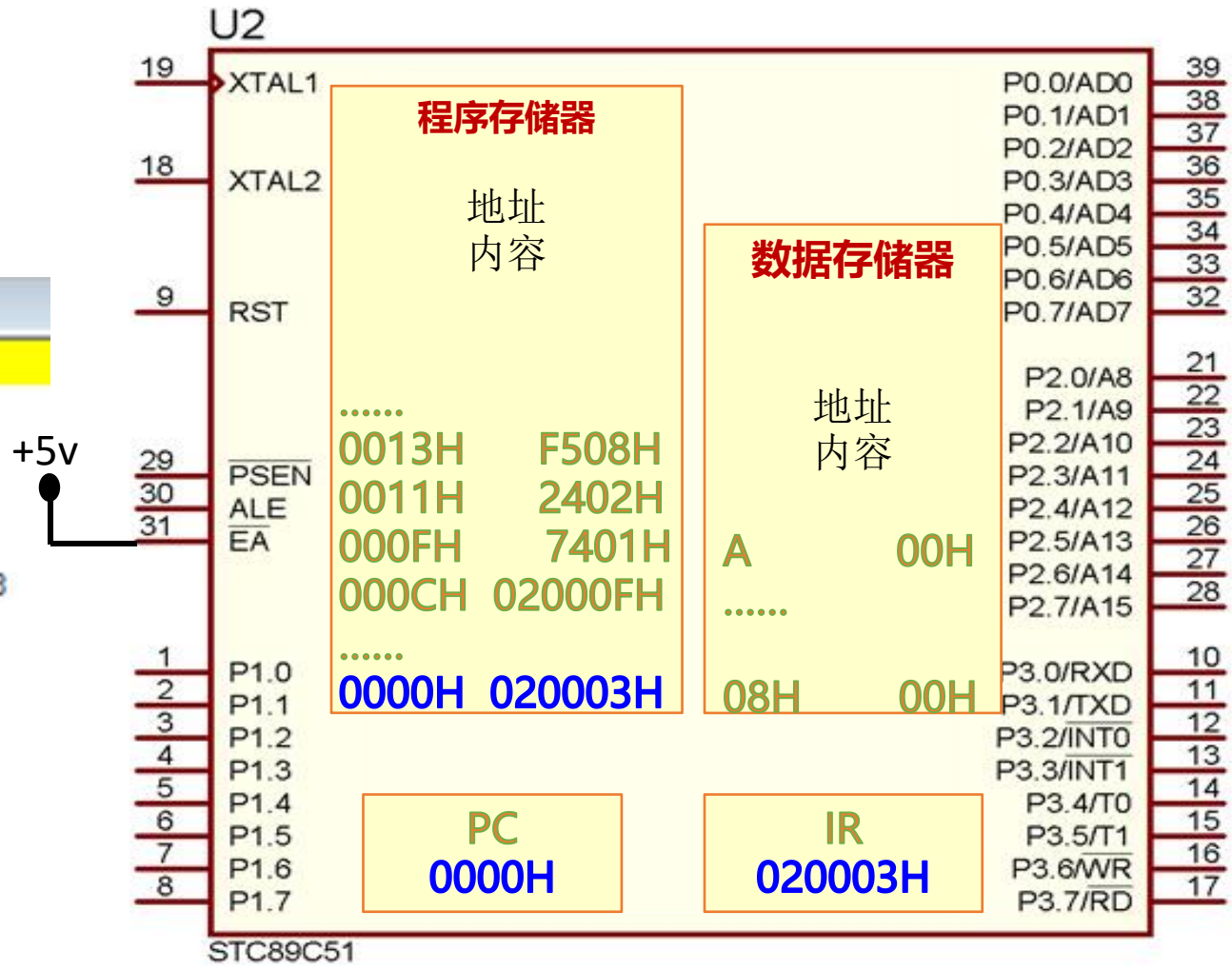
注意PC的值

prog1.c

```
1 void main()
2 {
3   char i=1,j=2,k;
4   k=i+j;
5 }
```

Disassembly

```
C:0x0000    020003    LJMP    C:0003
C:0x0003    787F      MOV     R0,#0x7F
C:0x0005    E4        CLR     A
C:0x0006    F6        MOV     @R0,A
C:0x0007    D8FD      DJNZ    R0,C:0006
C:0x0009    758108    MOV     SP(0x81),#0x08
C:0x000C    02000F    LJMP    main(C:000F)
      2: {
      3:         char i=1,j=2,k;
      4:         k=i+j;
C:0x000F    7401      MOV     A,#0x01
C:0x0011    2402      ADD     A,#0x02
C:0x0013    F508      MOV     0x08,A
      5: }
```



+5v

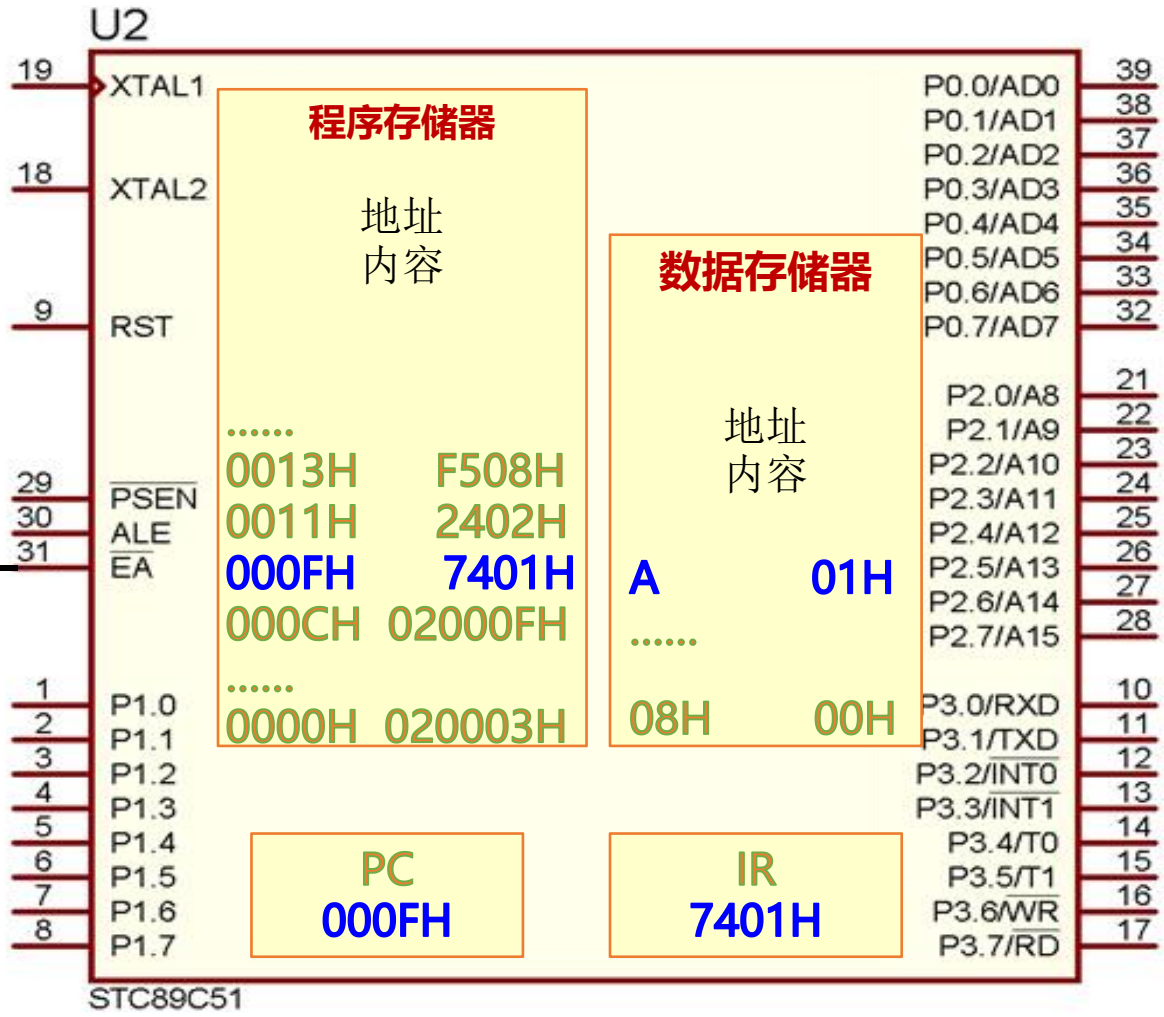
注意PC的值

prog1.c

```
1 void main()
2 {
3   char i=1,j=2,k;
4   k=i+j;
5 }
```

Disassembly

```
C:0x0000    020003    LJMP    C:0003
C:0x0003    787F      MOV     R0,#0x7F
C:0x0005    E4        CLR     A
C:0x0006    F6        MOV     @R0,A
C:0x0007    D8FD      DJNZ    R0,C:0006
C:0x0009    758108    MOV     SP(0x81),#0x08
C:0x000C    02000F    LJMP    main(C:000F)
      2: {
      3:         char i=1,j=2,k;
      4:         k=i+j;
C:0x000F    7401      MOV     A,#0x01
C:0x0011    2402      ADD     A,#0x02
C:0x0013    F508      MOV     0x08,A
      5: }
C:0x0015    22        RET
C:0x0016    00        NOP
```



+5v

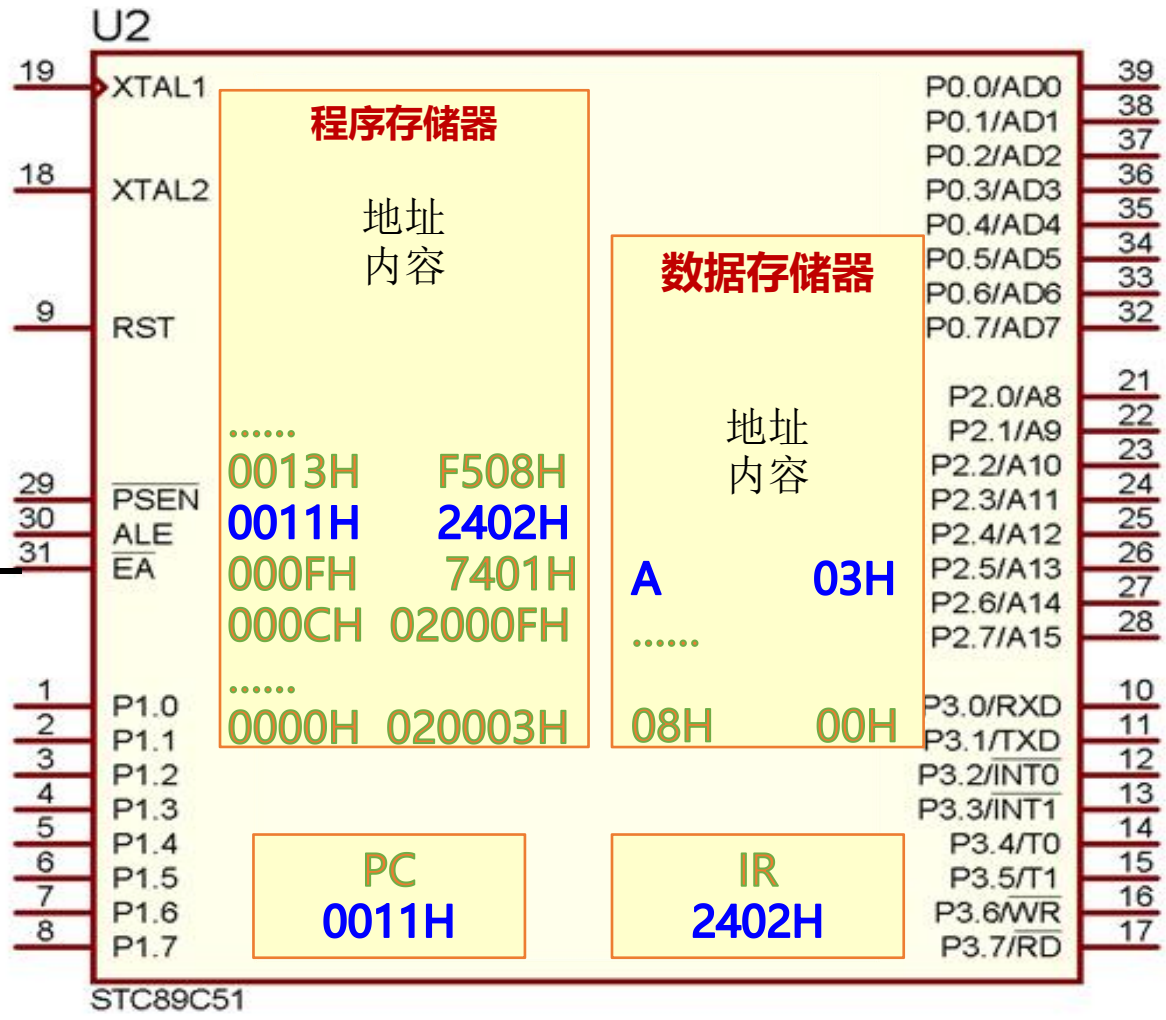
注意PC的值

prog1.c

```
1 void main()
2 {
3   char i=1,j=2,k;
4   k=i+j;
5 }
```

Disassembly

```
C:0x0000    020003    LJMP    C:0003
C:0x0003    787F      MOV     R0,#0x7F
C:0x0005    E4        CLR     A
C:0x0006    F6        MOV     @R0,A
C:0x0007    D8FD      DJNZ    R0,C:0006
C:0x0009    758108    MOV     SP(0x81),#0x0
C:0x000C    02000F    LJMP    main(C:000F)
      2: {
      3:         char i=1,j=2,k;
      4:         k=i+j;
C:0x000F    7401      MOV     A,#0x01
C:0x0011    2402      ADD     A,#0x02
C:0x0013    F508      MOV     0x08,A
      5: }
C:0x0015    22        RET
```



+5v

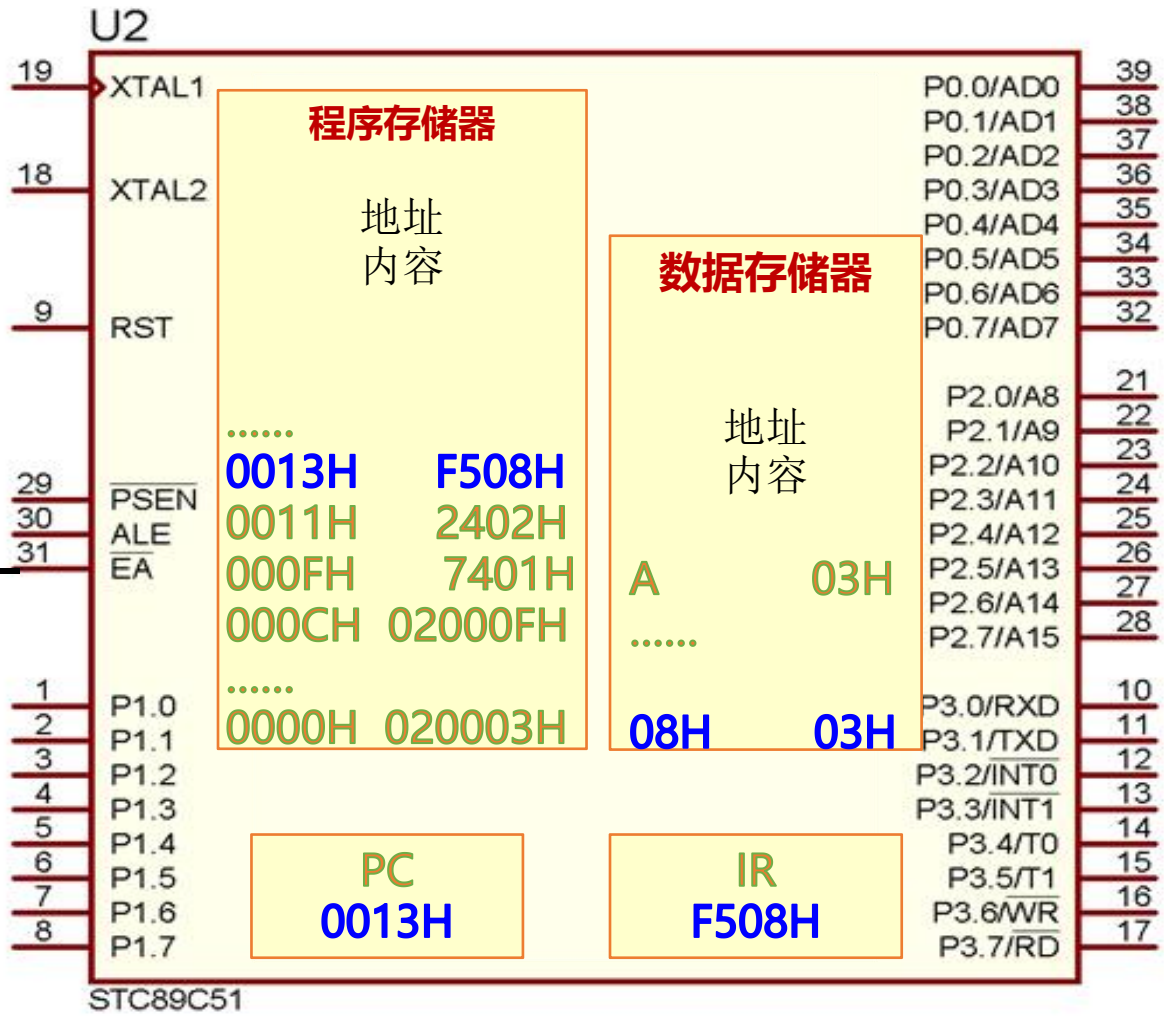
注意PC的值

prog1.c

```
1 void main()
2 {
3   char i=1,j=2,k;
4   k=i+j;
5 }
```

Disassembly

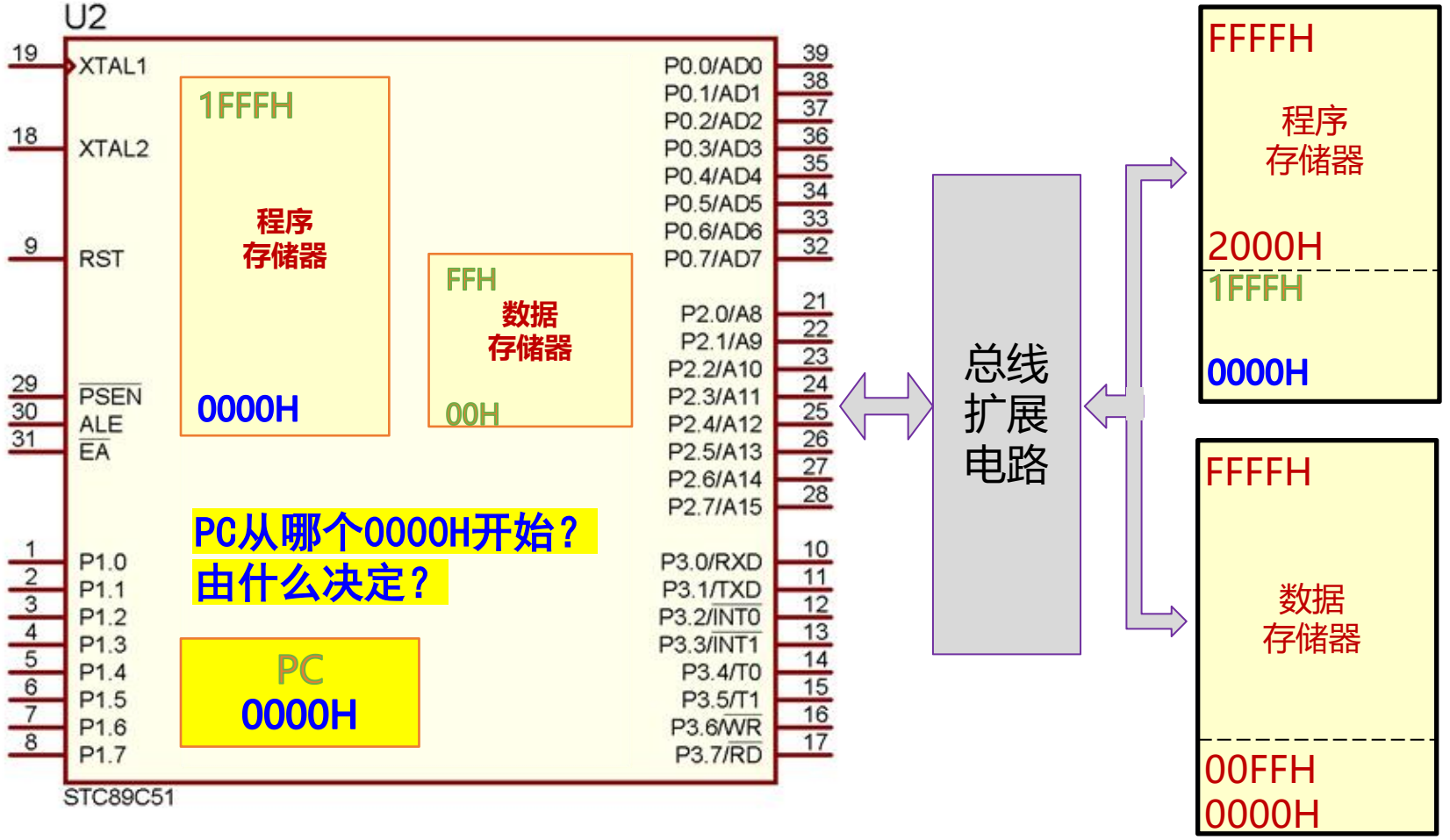
```
C:0x0000    020003    LJMP    C:0003
C:0x0003    787F      MOV     R0,#0x7F
C:0x0005    E4        CLR     A
C:0x0006    F6        MOV     @R0,A
C:0x0007    D8FD      DJNZ    R0,C:0006
C:0x0009    758108    MOV     SP(0x81),#0x08
C:0x000C    02000F    LJMP    main(C:000F)
      2: {
      3:         char i=1,j=2,k;
      4:         k=i+j;
C:0x000F    7401      MOV     A,#0x01
C:0x0011    2402      ADD     A,#0x02
C:0x0013    F508      MOV     0x08,A
      5: }
C:0x0015    22        RET
```



+5v

STC89C52单片机程序存储器

- PC——16位程序计数器，初值0000H，是存储当前正在执行的指令的地址；

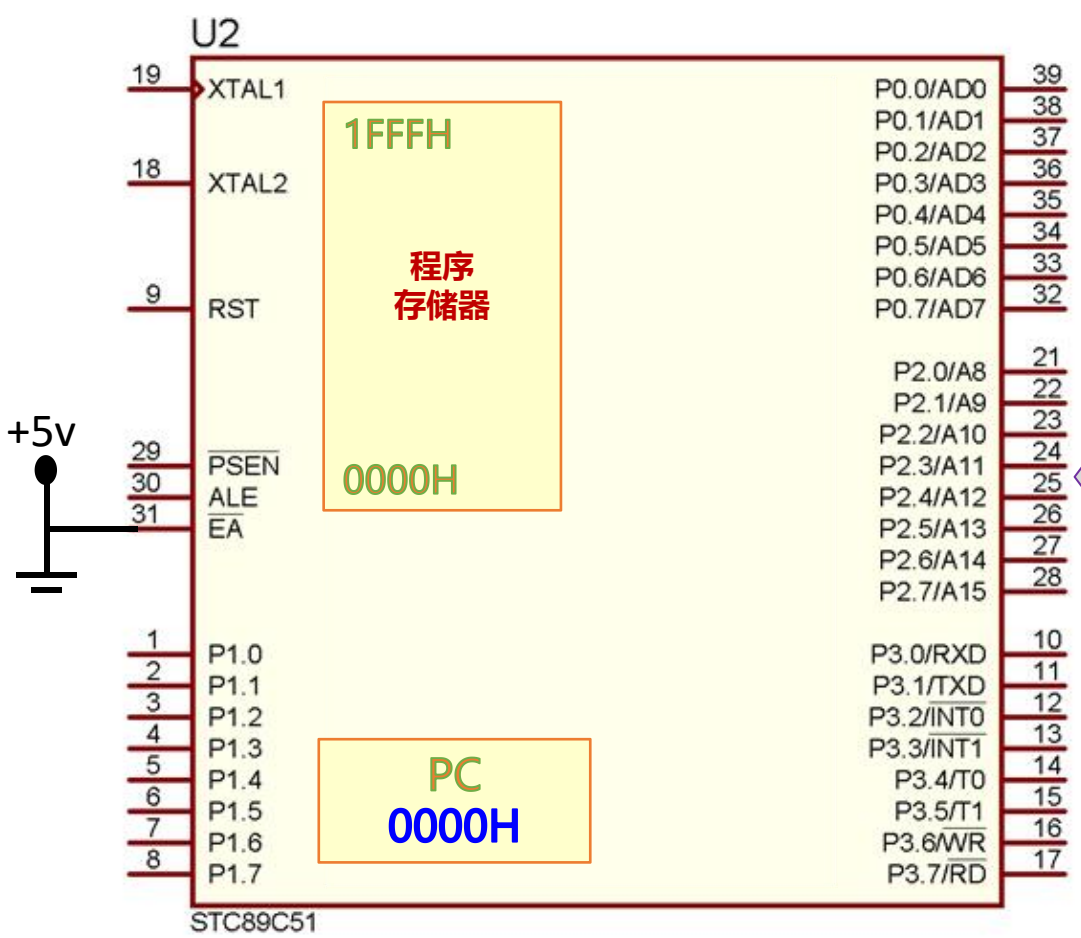


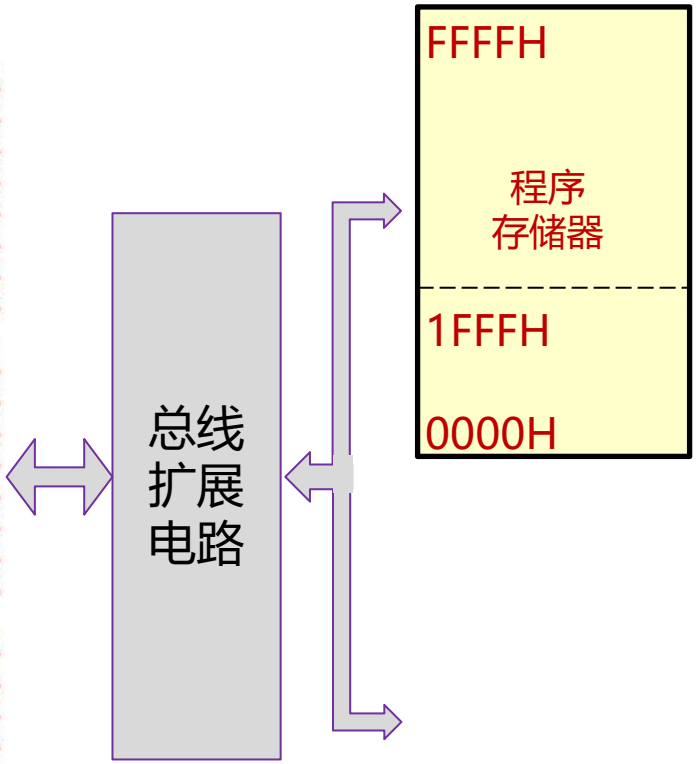
- PC具有自加“1”功能，可以确保计算机在执行程序时按照正确的顺序执行指令。

STC89C52单片机程序存储器

（1）程序存储器存放程序和表格之类的固定常数。

- 片内为8KB，地址为0000H～1FFFH。
- 外扩最大64KB，地址为0000H～FFFFH，16根地址线。

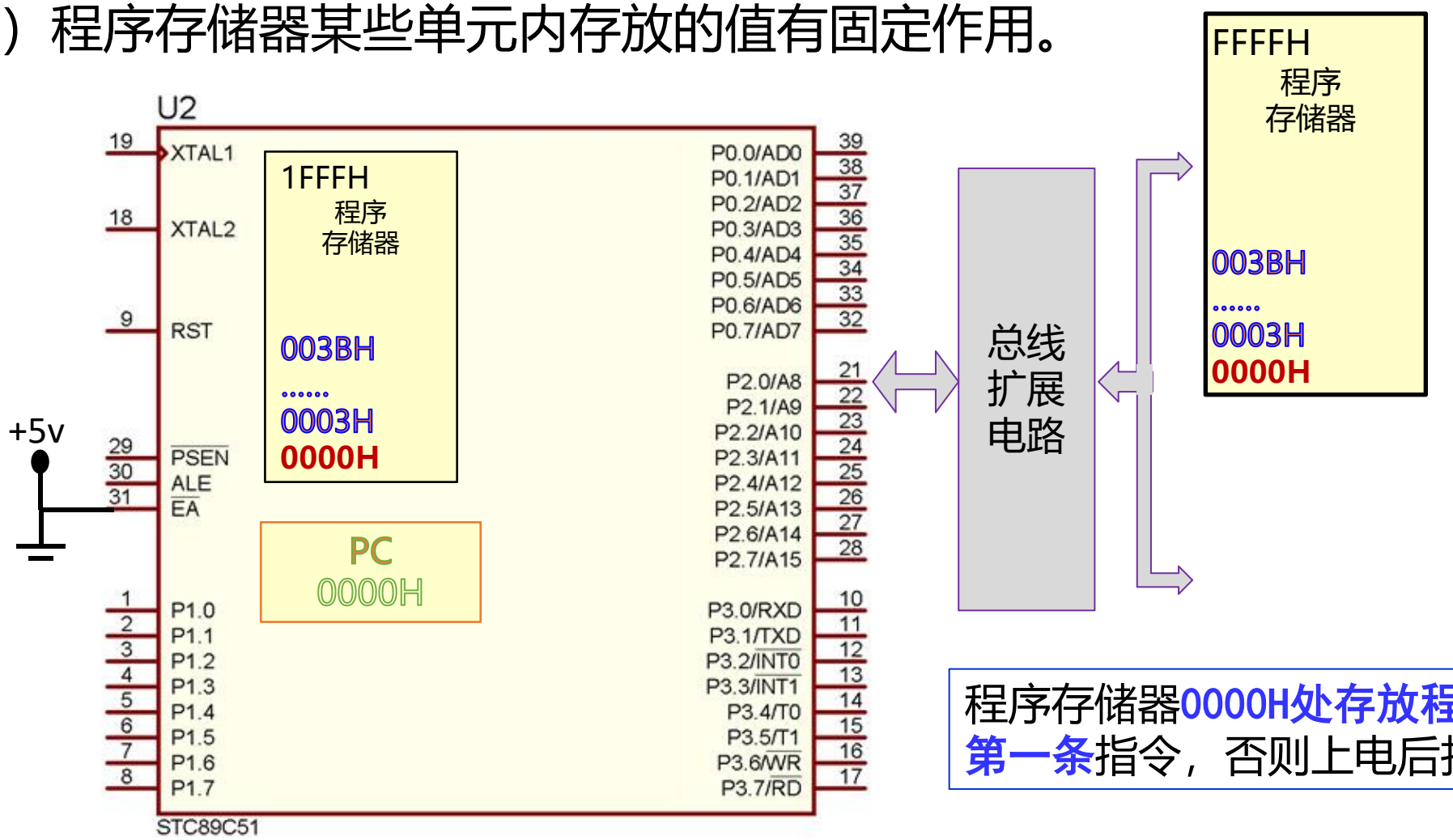




访问程序存储器的数据，将在讲解MOVC 指令时详细讲解。

STC89C52单片机程序存储器

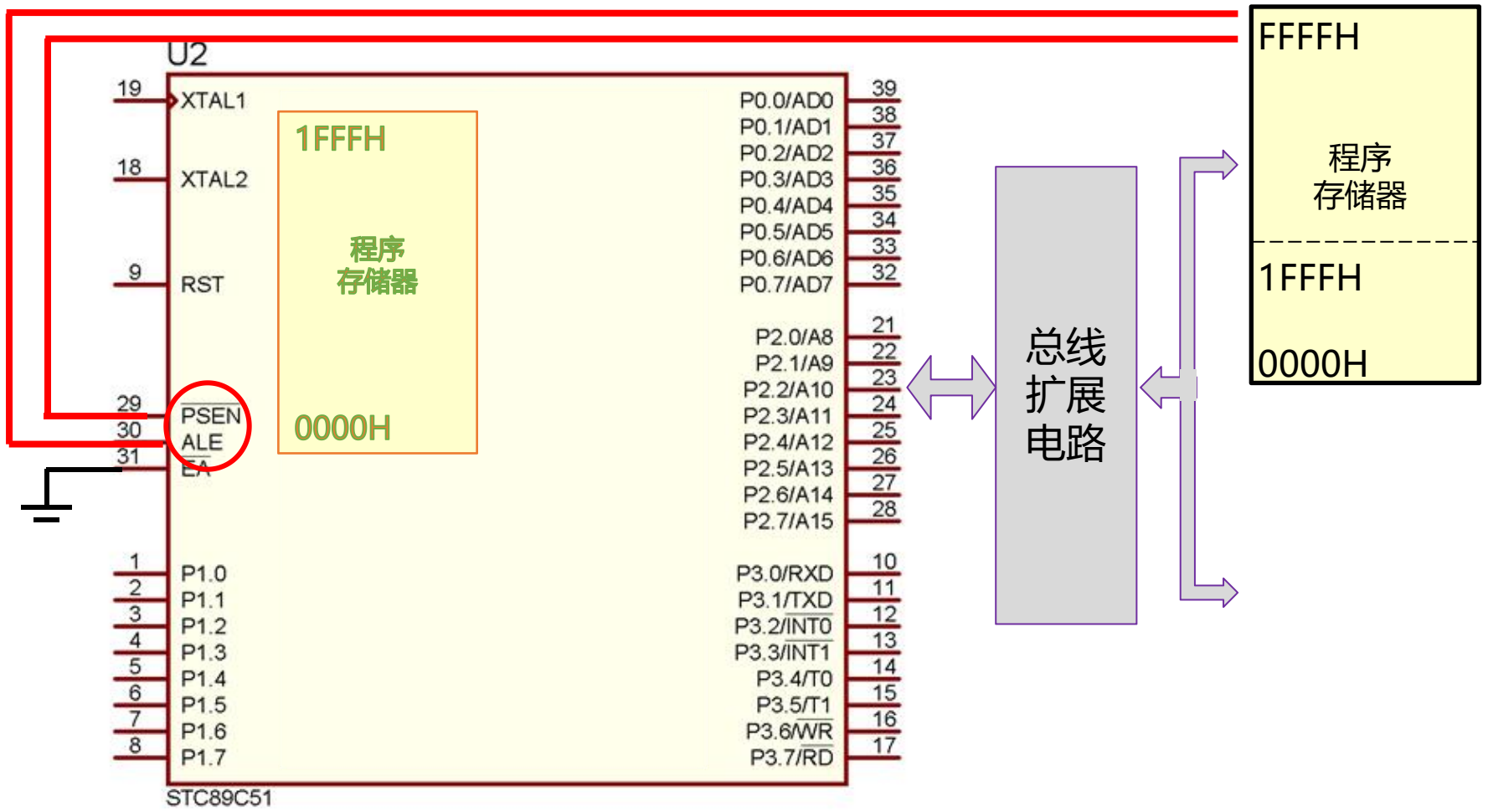
（2）程序存储器某些单元内存放的值有固定作用。



程序存储器0000H处存放程序代码的第一条指令，否则上电后找不到程序。

程序存储器0003H、000BH、0013H、001BH、0023H、002BH、0033H、003BH这8个特殊单元分别对应于8个中断源的中断服务入口地址，见表4- 3，将在中断系统章节详细讲述。

片外程序存储器将在存储器扩展章节详述



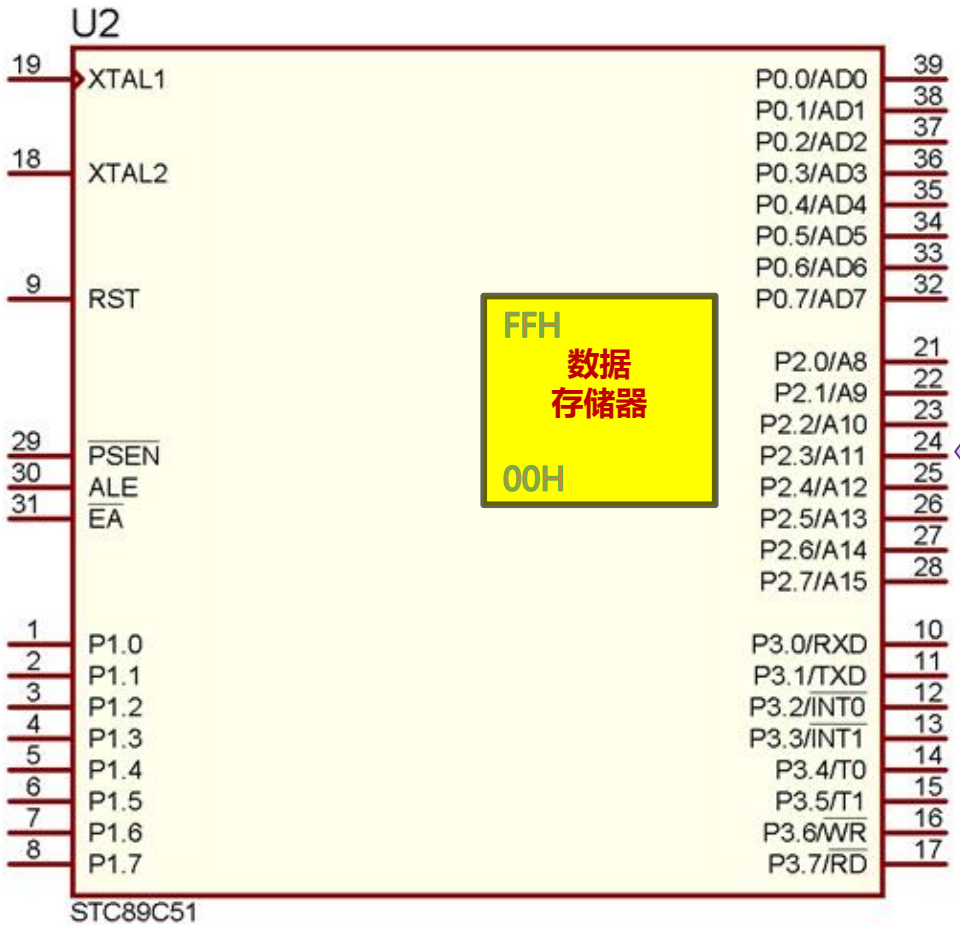
PSEN　（片外程序存储器读选通）

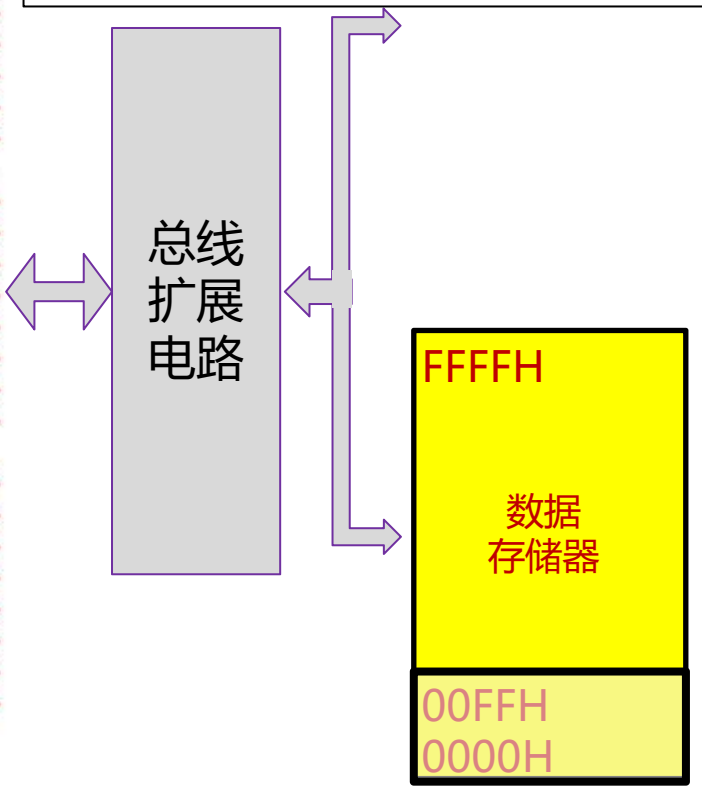
ALE（地址锁存允许信号）CPU访问外部存储器提供低8位地址锁存信号。

STC89C52单片机数据存储器

数据存储器用于存放程序执行的中间结果和过程数据。

- 内部RAM 512字节，在物理和逻辑上分为两个空间：
  - –内部RAM(00H-FFH)256字节
  - –内部扩展RAM(00H-FFH)256字节(STC新增)
- 外部RAM (0000H-FFFFH)64KB





C51与标准C的主要区别之一—存储类型

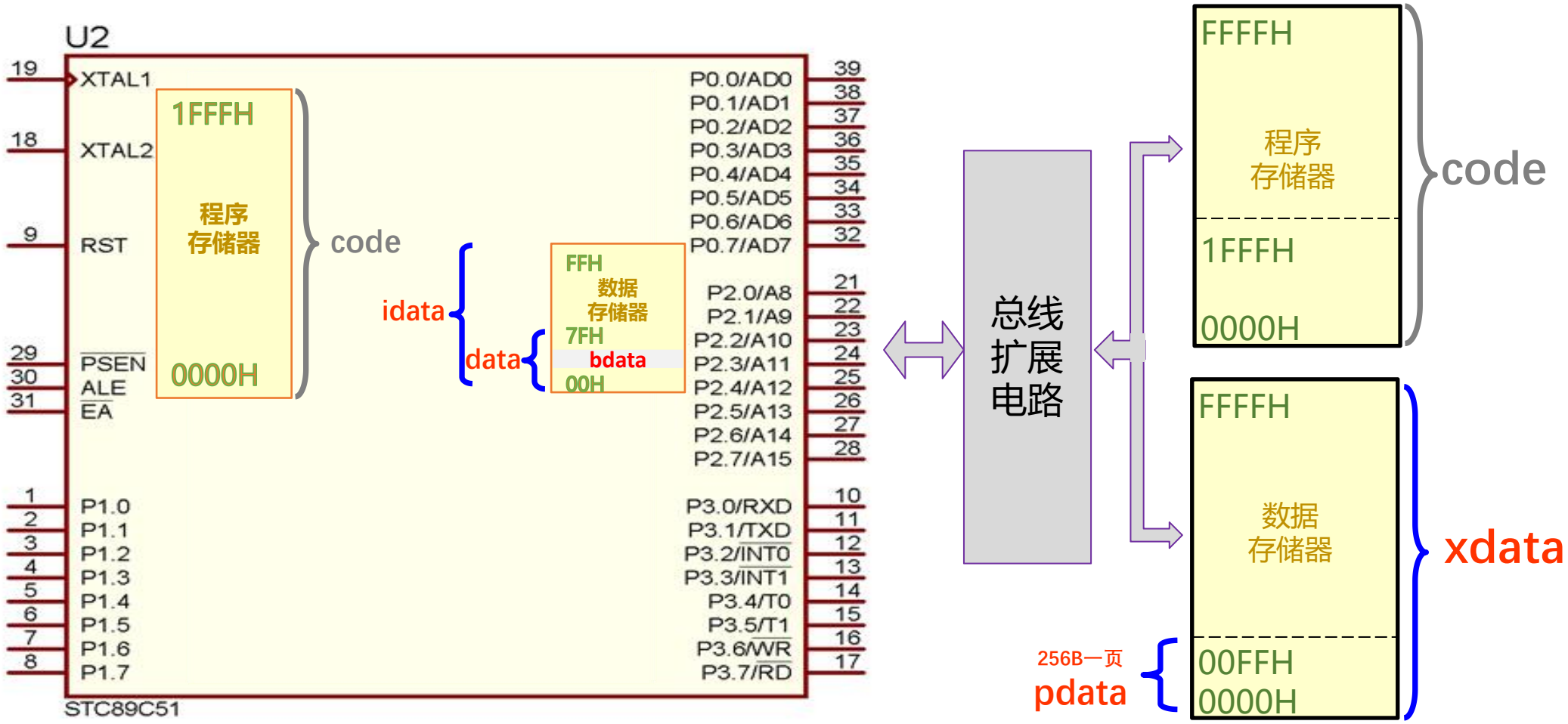
【存储种类】　数据类型　【存储类型】　变量名

（标准C）　（标准C+C51）　（C51特有）　（标准C）

例：　static　char　data　var;

```
1  #include <reg52.h>
2  #define  LENTH 16
3
4  unsigned char idata dADDR[LENTH] _at_ 0x30;
5  unsigned char xdata xADDR[LENTH] _at_ 0x3000;
6
7  void main(){
8    unsigned int i;
9    for(i=0;i<LENTH;i++)
10     dADDR[i]=i+1;
11     for(i=0;i<LENTH;i++)
12       xADDR[i]=dADDR[i];
13 }
```

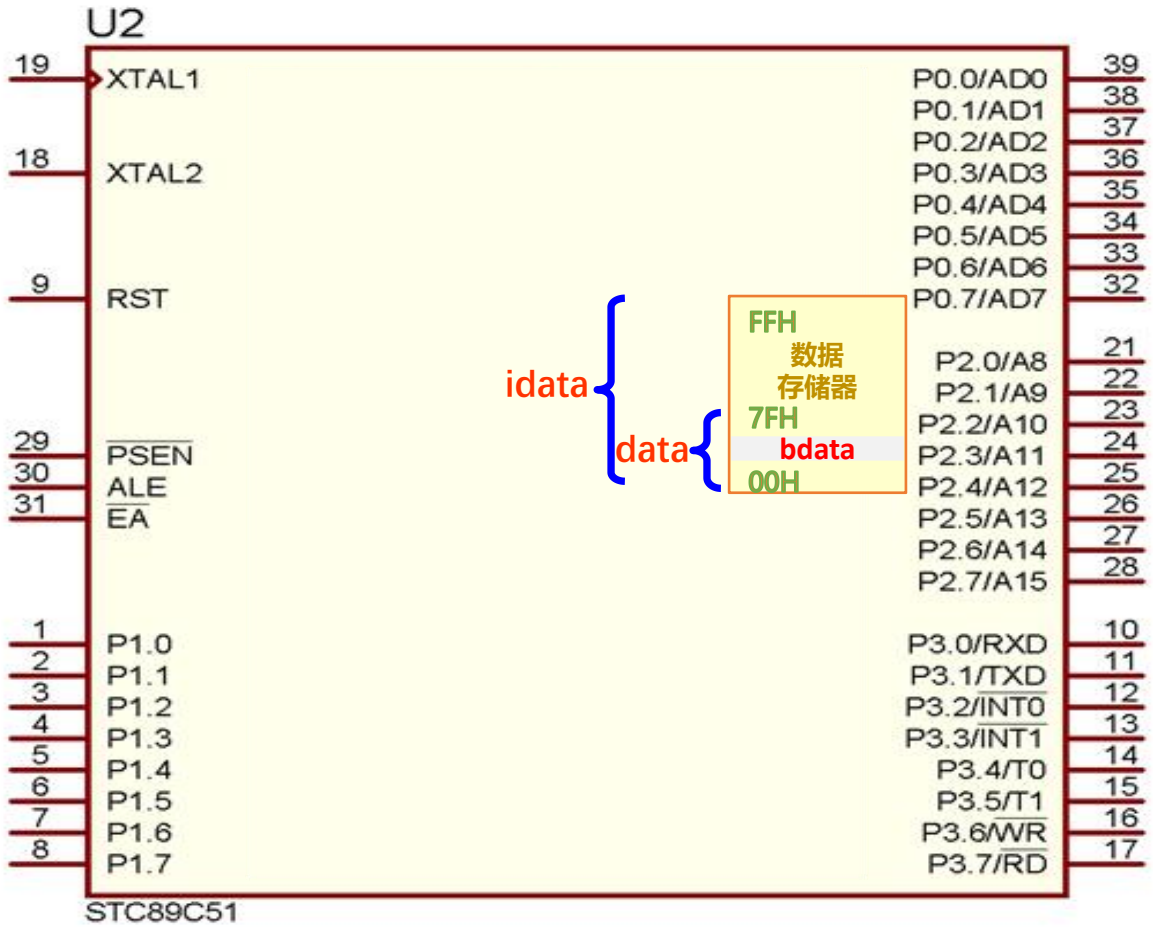
存储类型	长度bit	值域范围
data(默认)	8	0 ~ 255
bdata	8	0 ~ 255
idata	8	0 ~ 255
pdata	8	0 ~ 255
xdata	16	0 ~ 65535
code	16	0 ~ 65535

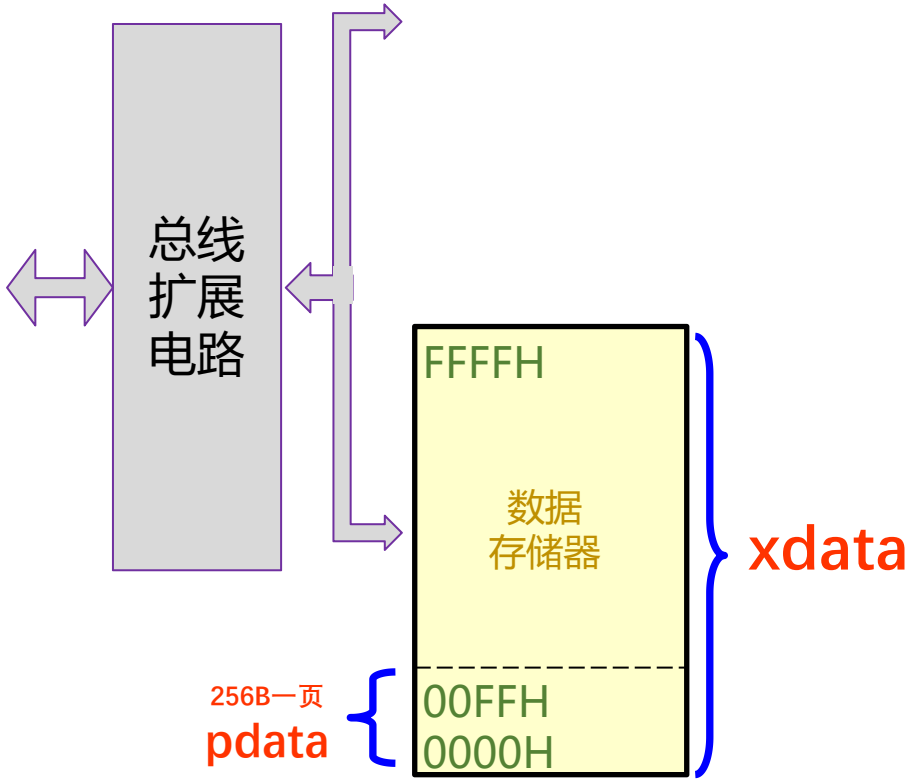

U2
19 XTAL1
18 XTAL2
9 RST
29 PSEN
30 ALE
31 EA
1 P1.0
2 P1.1
3 P1.2
4 P1.3
5 P1.4
6 P1.5
7 P1.6
8 P1.7
STC89C51
1FFFH
程序
存储器
0000H
code
idata
data
FFH
数据
存储器
7FH
bdata
00H
39 P0.0/AD0
38 P0.1/AD1
37 P0.2/AD2
36 P0.3/AD3
35 P0.4/AD4
34 P0.5/AD5
33 P0.6/AD6
32 P0.7/AD7
21 P2.0/A8
22 P2.1/A9
23 P2.2/A10
24 P2.3/A11
25 P2.4/A12
26 P2.5/A13
27 P2.6/A14
28 P2.7/A15
10 P3.0/RXD
11 P3.1/TXD
12 P3.2/INT0
13 P3.3/INT1
14 P3.4/T0
15 P3.5/T1
16 P3.6/WR
17 P3.7/RD
总线
扩展
电路
FFFFH
程序
存储器
1FFFH
0000H
code
FFFFH
数据
存储器
00FFH
0000H
xdata
256B一页
pdata

数据的存储类型

- C51编译器通过存储类型设置将变量、常量定义在不同的存储区。

存储类型	与存储空间的对应关系	长度bit	值域范围
data (默认)	片内RAM 直接寻址区 位于片内RAM的低128字节	8	0～255
bdata	片内RAM 位寻址区 位于20H~2FH空间，允许位访问与字节访问	8	0～255
idata	片内RAM 间接寻址区 全部256字节存储区00H~FFH，由MOV @Ri访问	8	0～255
pdata	片外RAM的一个分页间接寻址区 占256字节，使用指令MOVX @Ri访问	8	0～255
xdata	片外RAM 间接寻址区 全部64K字节空间，使用MOVX @DPTR访问	16	0～65535
code	程序存储区 全部64KB空间，使用MOVC @A+DPTR访问	16	0～65535





idata 片内全部256B存储空间-间接寻址区00H~FFH
data 片内低128B存储空间-直接寻址区00H~7FH（默认）
bdata 片内位寻址区20H~2FH
使用频率高的放data或idata区；bdata区不允许定义浮点数

xdata 片外全部64KB数据存储空间0000H-FFFFH。
pdata 片外一页256B数据存储空间00H-FFH。
pdata寻址只需8位地址比xdata快，尽量放pdata区

1. 片内数据存储器

（1）低128字节（00H-7FH）-data区

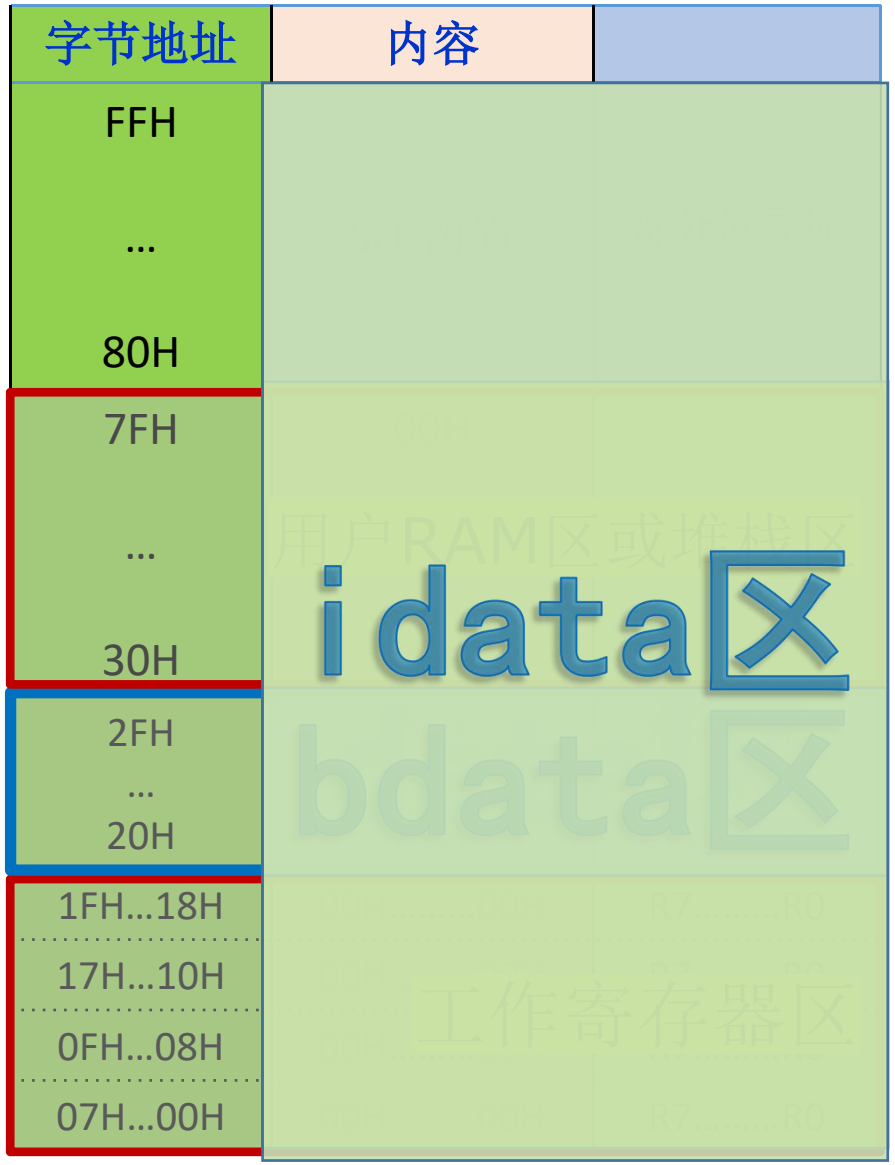
即可直接寻址也可间接寻址，分为：

– 工作寄存器区（00H--1FH）

- 第0组（00H--07H）8字节
- 第1组（08H--0FH）8字节
- 第2组（10H--17H）8字节
- 第3组（18H--1FH）8字节

– 可位寻址区（20H--2FH）16字节-bdata区

– 用户RAM和堆栈区（30H--7FH）80字节



1. 片内数据存储器

1)工作寄存器区（00H—1FH)——data区

字节地址	内容	
FFH … 80H	SFR初值	部分位寻址
7FH … 30H	00H … 00H	
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH...18H	00H.....00H	R7.....R0
17H...10H	00H.....00H	R7.....R0
0FH...08H	00H.....00H	R7.....R0
07H...00H	00H.....00H	R7.....R0

data区

寄存器同名如何区别是？

如MOV　R0,#00H

指令中的R0是哪个位置的？

记住各组的地址范围

共32个单元分为4组

每组同名为R0～R7

初始值都为00H

1. 片内数据存储器

1)工作寄存器区（00H—1FH)

- **任一时刻，CPU只能使用一组工作寄存器。（由PSW寄存器中的RS0和RS1两位决定）**

例：
```
SETB   RS1
CLR    RS0
MOV    R2, #11H
```

此时数值11H存放在哪里？

RS1	RS0	寄存器组	地址	
0	0	第0组	00H～07H	R0~R7
0	1	第1组	08H～0FH	R0~R7
1	0	第2组	10H～17H	R0~R7
1	1	第3组	18H～1FH	R0~R7

MOV direct, #data

MOV 12H, #11H

	地址		内容
第	1FH	R7	00H

	地址		内容
第3组	1FH … 18H	R7 … R0	00H … 00H
第2组	17H … 12H 11H 10H	R7 … R2 R1 R0	00H … 11H 00H 00H
第1组	0FH … 08H	R7 … R0	00H … 00H
第0组	07H … 00H	R7 … R0	00H … 00H

1. 片内数据存储器

2)可位寻址区（20H—2FH)——bdata区

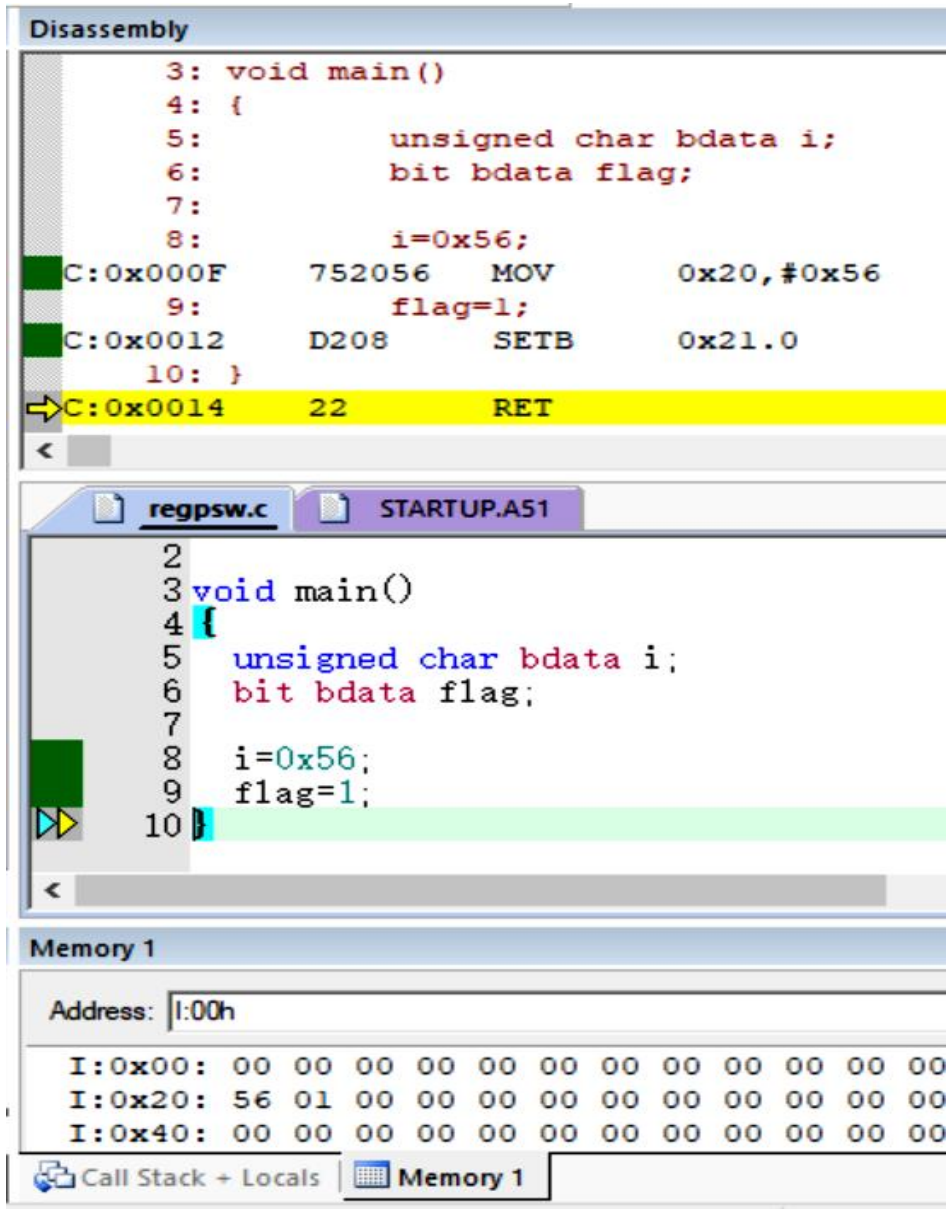
16个**字节单元**，128个**位单元**
既可**字节操作**，又可**位操作**

字节地址	内容	位地址							
		D7	D6	D5	D4	D3	D2	D1	D0
2FH	00H	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	00H	77H	76H	75H	74H	73H	72H	71H	70H
2DH	00H	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	00H	67H	66H	65H	64H	63H	62H	61H	60H
2BH	00H	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	00H	57H	56H	55H	54H	53H	52H	51H	50H
29H	00H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	00H	47H	46H	45H	44H	43H	42H	41H	40H
27H	00H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	00H	37H	36H	35H	34H	33H	32H	31H	30H
25H	00H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	00H	27H	26H	25H	24H	23H	22H	21H	20H
23H	00H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	00H	17H	16H	15H	14H	13H	12H	11H	10H
21H	00H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	00H	07H	06H	05H	04H	03H	02H	01H	00H

字节地址	内容	
FFH … 80H	SFR初值	部分位寻址
7FH … 30H	00H … 00H	
2FH … 20H	bdata区	
1FH…18H	00H……00H	R7……R0
17H…10H	00H……00H	R7……R0
0FH…08H	00H……00H	R7……R0
07H…00H	00H……00H	R7……R0

1. 片内数据存储器

2)可位寻址区（20H—2FH)——bdata区



字节地址	内容	位地址							
		D7	D6	D5	D4	D3	D2	D1	D0
2FH	00H	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	00H	77H	76H	75H	74H	73H	72H	71H	70H
2DH	00H	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	00H	67H	66H	65H	64H	63H	62H	61H	60H
2BH	00H	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	00H	57H	56H	55H	54H	53H	52H	51H	50H
29H	00H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	00H	47H	46H	45H	44H	43H	42H	41H	40H
27H	00H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	00H	37H	36H	35H	34H	33H	32H	31H	30H
25H	00H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	00H	27H	26H	25H	24H	23H	22H	21H	20H
23H	00H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	00H	17H	16H	15H	14H	13H	12H	11H	10H
21H	**01H**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
20H	**56H**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**

1. 片内数据存储器

2)可位寻址区（20H—2FH)——bdata区

字节地址	内容	位地址							
		D7	D6	D5	D4	D3	D2	D1	D0
2FH	00H	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	00H	77H	76H	75H	74H	73H	72H	71H	70H
2DH	00H	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	00H	67H	66H	65H	64H	63H	62H	61H	60H
2BH	00H	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	00H	57H	56H	55H	54H	53H	52H	51H	50H
29H	00H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	00H	47H	46H	45H	44H	43H	42H	41H	40H
27H	00H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	00H	37H	36H	35H	34H	33H	32H	31H	30H
25H	00H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	00H	27H	26H	25H	24H	23H	22H	21H	20H
23H	00H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	00H	17H	16H	15H	14H	13H	12H	11H	10H
21H	00H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	56H	0	1	0	1	0	1	1	0

字节地址	内容	位地址							
		D7	D6	D5	D4	D3	D2	D1	D0
2FH	00H	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	00H	77H	76H	75H	74H	73H	72H	71H	70H
2DH	00H	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	00H	67H	66H	65H	64H	63H	62H	61H	60H
2BH	00H	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	00H	57H	56H	55H	54H	53H	52H	51H	50H
29H	00H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	00H	47H	46H	45H	44H	43H	42H	41H	40H
27H	00H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	00H	37H	36H	35H	34H	33H	32H	31H	30H
25H	00H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	00H	27H	26H	25H	24H	23H	22H	21H	20H
23H	00H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	00H	17H	16H	15H	14H	13H	12H	11H	10H
21H	00H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	5AH	0	1	0	1	1	0	1	0

字节操作 **MOV 20H,#56H**

位操作

```
SETB 03H
CLR  02H
```

```
SETB 20.3H
CLR  20.2H
```

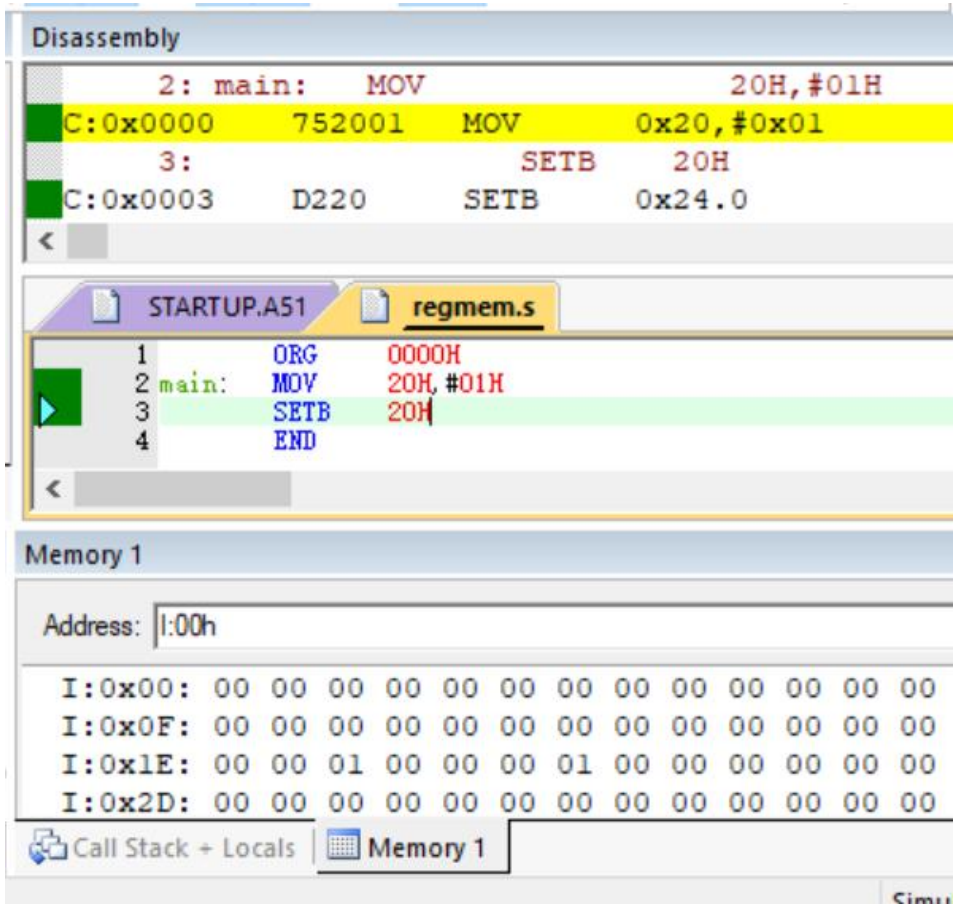
1. 片内数据存储器

2)可位寻址区（20H—2FH)——bdata区

难点：字节地址与位地址！

例：MOV 20H, #01H

SETB 20H



字节地址	内容	位地址							
		D7	D6	D5	D4	D3	D2	D1	D0
2FH	00H	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	00H	77H	76H	75H	74H	73H	72H	71H	70H
2DH	00H	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	00H	67H	66H	65H	64H	63H	62H	61H	60H
2BH	00H	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	00H	57H	56H	55H	54H	53H	52H	51H	50H
29H	00H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	00H	47H	46H	45H	44H	43H	42H	41H	40H
27H	00H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	00H	37H	36H	35H	34H	33H	32H	31H	30H
25H	00H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	**H	*	*	*	*	*	*	*	1
23H	00H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	00H	17H	16H	15H	14H	13H	12H	11H	10H
21H	00H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	**01H**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

1. 片内数据存储器　3)用户RAM区（30H～7FH）——data/idata区

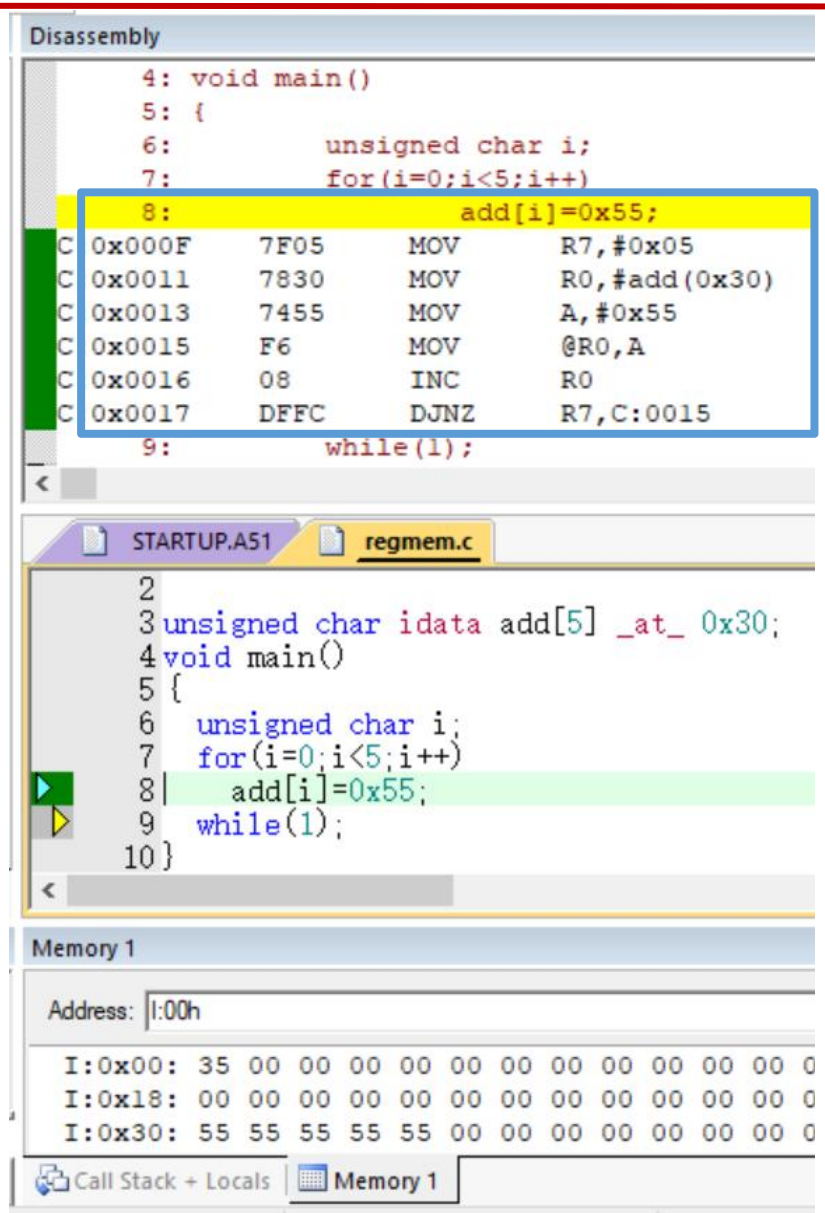
- 用于存放用户数据及作堆栈区使用，共80个单元。
- 该RAM区的使用没有任何规定或限制。

STARTUP.A51 | regmem.c

```
2
3 unsigned char idata add[5] _at_ 0x30;
4 void main()
5 {
6   unsigned char i;
7   for(i=0;i<5;i++)
8     add[i]=0x55;
9   while(1);
10}
```

```
8:                    add[i]=0x55;
C:0x000F    7F05      MOV       R7,#0x05
C:0x0011    7830      MOV       R0,#add(0x30)
C:0x0013    7455      MOV       A,#0x55
C:0x0015    F6        MOV       @R0,A
C:0x0016    08        INC       R0
```

字节地址	内容	
FFH … 80H	SFR初值	部分位寻址
7FH … 34H … 30H	00H … 55H … 05H	data区/ idata区
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH...18H	00H.....00H	R7.....R0
17H...10H	00H.....00H	R7.....R0
0FH...08H	00H.....00H	R7.....R0
07H...00H	00H.....00H	R7.....R0



字节地址	内容	
FFH … 80H	SFR初值	部分位寻址
7FH … 34H … 30H	00H … 55H … 55H	
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH...18H	00H.....00H	R7.....R0
17H...10H	00H.....00H	R7.....R0
0FH...08H	00H.....00H	R7.....R0
07H...00H	00H.....00H	R7.....R0

1. 片内数据存储器

（1）低128字节（00H-7FH）-data区

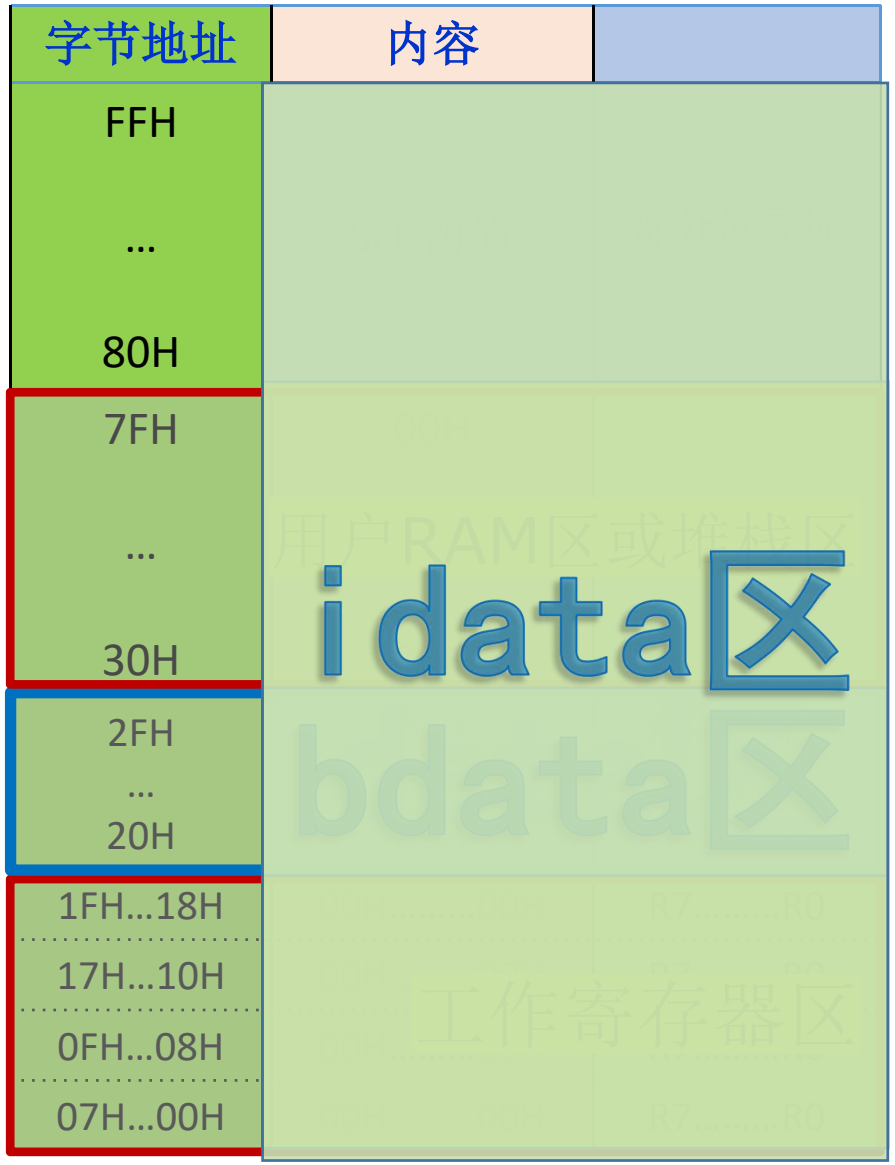
即可直接寻址也可间接寻址，分为：

– 工作寄存器区（00H--1FH）

- 第0组（00H--07H）8字节
- 第1组（08H--0FH）8字节
- 第2组（10H--17H）8字节
- 第3组（18H--1FH）8字节

– 可位寻址区（20H--2FH）16字节-bdata区

– 用户RAM和堆栈区（30H--7FH）80字节



1. 片内数据存储器

（1）高128字节（80H-FFH）-idata区

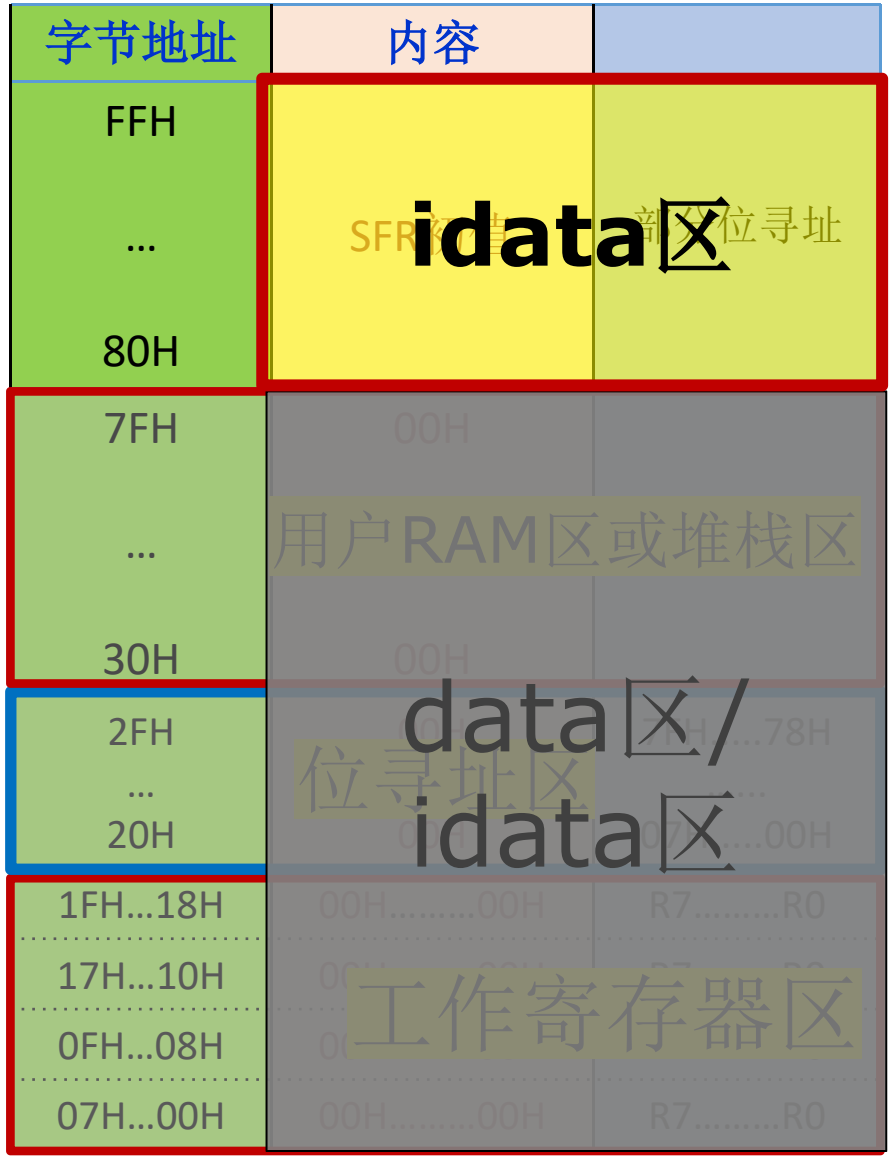
- 普通RAM空间和特殊功能寄存器区SFR的空间在逻辑上共用地址80H-FFH，但物理上是独立的。

- 通过不同的寻址方式加以区分：
  - 高128字节只能间接寻址

    如：MOV R0, #80H

    MOV @R0, A
  - 特殊功能寄存器区SFR只能直接寻址。

    如：MOV SP,#30H



STC89C52单片机特殊功能寄存器SFR

- SFR是实现单片机对片内各功能部件的控制。
- SFR是离散地分布在片内RAM80H-FFH区域中。
- SFR的字节地址末位为0H或8H的是可以位寻址的，即该SFR的每一位都有地址，且该SFR最低位的位地址与字节地址数值上是相同。

字节地址	名称	复位值	位地址							
			X.7	X.6	X.5	X.4	X.3	X.2	X.1	X.0
FFH										
……	……	……								
F0H	B	0000 0000	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H
……	……	……								
E0H	A	0000 0000	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H
……	……	……								
D0H	PSW	0000 0000	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
……	……	……								
8EH	AUXR	XXXX XX00								
……	……	……								
88H	TCON	0000 0000	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
……	……	……								
81H	SP	0000 0111								
80H			87H	86H	85H	84H	83H	82H	81H	80H

SFR要掌握名称、功能、复位值

表4- 4　　单片机内核特殊功能寄存器

序号	符号	功能介绍	字节地址	位地址	复位值
1	ACC	累加器	E0H	E7-E0H	0000 0000
2	B	B寄存器	F0H	F7-F0H	0000 0000
3	PSW	程序状态字寄存器	D0H	D7-D0H	0000 0000
4	SP	堆栈指针	81H	—	0000 0111
5	DP0L	数据地址指针DPTR0低8位	82H	—	0000 0000
6	DP0H	数据地址指针DPTR0高8位	83H	—	0000 0000
7	DP1L	数据地址指针DPTR1低8位	84H	—	
8	DP1H	数据地址指针DPTR1高8位	85H	—	

表4- 7　　单片机I/O口特殊功能寄存器

序号	符号	功能介绍	字节地址	位地址	复位值
1	P0	P0口锁存器	80H	87H-80H	1111 1111
2	P1	P1口锁存器	90H	97H-90H	1111 1111
3	P2	P2口锁存器	A0H	A7H-A0H	1111 1111
4	P3	P3口锁存器	B0H	B7H-B0H	1111 1111
5	P4	P4口锁存器	E8H	E7H-E0H	xxxx 1111

C51与标准C的主要区别之二扩展数据类型（头文件）

数据类型	位数	字节数	取值范围
signed char	8	1	-128~+127
unsigned char	8	1	0~255
signed int	16	2	-32768~+32767
unsigned int	16	2	0~65535
signed long	32	4	-2147483648~+2147483647
unsigned long	32	4	0~+4294967295
float	32	4	±
double	64	8	±
*	24	1~3	对象指针
sfr	**8**	**1**	**0~255**
sfr16	**16**	**2**	**0~65535**
sbit	**1**	**位**	可进行位寻址的SFR的某位的绝对地址
bit	**1**	**位**	**0**

C51扩展了4种数据类型

【存储种类】 数据类型 【存储类型】 变量名

（标准C） （标准C+**C51**） （C51特有） （标准C）

sfr和sfr16 声明字节寻址和字寻址的**SFR**

sbit 声明可位寻址的**SFR**的相关位

均放在头文件<reg5x.h>中

头文件<reg5x.h>

- sfr和sfr16 声明字节型和字型SFR

  声明格式：sfr SFR的名称=SFR的绝对字节地址

- sbit声明可位寻址的SFR的相关位

  声明格式：sbit SFR位名=SFR名称^位的位置数（常用）

  sbit SFR位名=SFR绝对字节地址^位的位置数

  sbit SFR位名=绝对位地址常数

```
/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
```

REG51.H

```
36  /*  BIT Register  */
37  /*  PSW   */
38  sbit CY   = 0xD7;
39  sbit AC   = 0xD6;
40  sbit F0   = 0xD5;
41  sbit RS1  = 0xD4;
42  sbit RS0  = 0xD3;
43  sbit OV   = 0xD2;
44  sbit P    = 0xD0;
```

reg52.h

```
43  /*  BIT Registers  */
44  /*  PSW  */
45  sbit CY   = PSW^7;
46  sbit AC   = PSW^6;
47  sbit F0   = PSW^5;
48  sbit RS1  = PSW^4;
49  sbit RS0  = PSW^3;
50  sbit OV   = PSW^2;
51  sbit P    = PSW^0; //
```

	D0^7	D0^6	D0^5	D0^4	D0^3	D0^2	D0^1	D0^0
PSW	CY	AC	F0	RS1	RS0	OV	F1	P
	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H

相对位地址

D0H 绝对字节地址

绝对位地址

- 被预先定义的SFR和其相关位放在头文件<reg5x.h>中，只需在程序开头加上：
  #include <reg51.h>或#include <reg52.h>

声明之后使用SFR名称赋值:

- sfr变量使用格式：SFR名称=值
- sbit变量使用格式：SFR位名=值

强调：头文件中定义的变量都是大写的，若程序采取小写变量则需要重新定义。

hello.c

```
1  #include <reg52.h>
2  #include <stdio.h>
3
4  void UartInit(void)
5  {
6    SCON=0x50;
7    TMOD=0x20;
8    TH1=0xFD;
9    TR1=1;
10   TI=1;
11 }
12
13 void main(void)
14 {
15   UartInit();
16   printf("hello world!");
17   while(1);
18 }
```

- bit——定义普通的位变量，其值只能是1或0。

  定义格式：bit　普通位变量名;

注意位变量的限制：

- 位变量定义存储类型时限制为bdata，data 或idata。
- 位变量不能用来定义指针和数组。

如：

```
bit  *ptr;          /* 错误，不能用位变量定义指针*/
bit  a-array[ ];    /* 错误，不能用位变量定义数组*/
```



介绍部分常用特殊功能寄存器

字节地址	名称	功能	复位值	位地址							
				D7	D6	D5	D4	D3	D2	D1	D0
FFH											
……			……								
F0H	**B**	**B寄存器**	**0000 0000**	F7H	F6H	F5H	F4H	F3H	F2H	F1H	**F0H**
……			……								
E0H	**A**	**累加器**	**0000 0000**	E7H	E6H	E5H	E4H	E3H	E2H	E1H	**E0H**
……			……								
D0H	**PSW**	**程序状态字寄存器**	**0000 0000**	D7H	D6H	D5H	D4H	D3H	D2H	D1H	**D0H**
……			……								
8EH	AUXR	辅助寄存器	**XXXX XX00**								
……			……								
83H	**DPH**	**数据寄存器低位**	**0000 0000**								
82H	**DPL**	**数据寄存器高位**	**0000 0000**								
81H	**SP**	**堆栈指针**	**0000 0111**								
80H				87H	86H	85H	84H	83H	82H	81H	**80H**

特殊功能寄存器——PSW

- **程序状态字寄存器PSW**(见表4- 14)
  （Program Status Word）

字节地址	内容	
FFH		
…		
D0H	CyAcF0RS1RS0OV-P	PSW
…		
80H		

➢ Cy（PSW.7）进位标志位

在算术和逻辑运算后：

- Cy = 1表示最高位有进/借位；Cy = 0表示最高位无进/借位。
- 在位处理器中是位累加器，Cy可写为C。

➢ Ac（PSW.6）辅助进位标志位

在BCD码运算时用作十进位调整。

- Ac = 1，表示当D3位向D4位产生进/借位；否则，Ac = 0。

reg52.h

```
43  /*  BIT Registers  */
44  /*  PSW  */
45  sbit CY    = PSW^7;
46  sbit AC    = PSW^6;
47  sbit F0    = PSW^5;
48  sbit RS1   = PSW^4;
49  sbit RS0   = PSW^3;
50  sbit OV    = PSW^2;
51  sbit P     = PSW^0; //
```

特殊功能寄存器——PSW

- **程序状态字寄存器PSW**(见表4- 14)
  （Program Status Word）

  ➢ OV（PSW.2）溢出标志位

  指示运算结果是否产生溢出。

  - OV=1，表示结果产生溢出；否则，OV=0。

  ➢ P（PSW.0）奇偶标志位

  累加器A中“1”的个数是奇还是偶。

  - P=1，表示A中“1”的个数为奇。
  - P=0，表示A中“1”的个数为偶。

  此标志位对串行通信有重要的意义，常用奇偶检验的方法来检验数据串行传输的可靠性。

字节地址	内容	
FFH …		
D0H	CyAcF0RS1RS0OV-P	PSW
… 80H		

reg52.h

```
43  /*  BIT Registers  */
44  /*  PSW  */
45  sbit CY    = PSW^7;
46  sbit AC    = PSW^6;
47  sbit F0    = PSW^5;
48  sbit RS1   = PSW^4;
49  sbit RS0   = PSW^3;
50  sbit OV    = PSW^2;
51  sbit P     = PSW^0; //
```

特殊功能寄存器——PSW

- **程序状态字寄存器PSW**(见表4- 14)
  （Program Status Word）

字节地址	内容	
FFH		
…		
D0H	CyAcF0**RS1RS0**OV-P	PSW
…		
80H		

- PSW.1位　保留位
- F0（PSW.5）用户设定标志位
  由用户使用的一个状态标志位。
- RS1、RS0（PSW.4、PSW.3）4组工作寄存器区选择
  选择片内RAM区中的4组工作寄存器区中的某一组为当前工作寄存区。

reg52.h

```
43  /*  BIT Registers  */
44  /*  PSW  */
45  sbit CY    = PSW^7;
46  sbit AC    = PSW^6;
47  sbit F0    = PSW^5;
48  sbit RS1   = PSW^4;
49  sbit RS0   = PSW^3;
50  sbit OV    = PSW^2;
51  sbit P     = PSW^0; //
```

特殊功能寄存器——A, B

- **累加器A或ACC**
  - 使用最频繁的寄存器，有些指令必须写为ACC。
  - 数据传送大多都通过累加器A，相当于数据的中转站。

- **寄存器B**
  - 为执行乘法和除法而设。
  - 在不执行乘、除法操作的情况下，可作一个普通寄存器来使用。

例：
```
MOV   A, #01H        MUL   AB
PUSH  ACC            DIV   AB
```

字节地址	内容	
FFH …		
F0H **E0H**	0000 0000 0000 0000	**B** **A**
… 80H		
7FH … 30H	00H … 00H	
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH…18H	00H.....00H	R7.....R0
17H…10H	00H.....00H	R7.....R0
10H…08H	00H.....00H	R7.....R0
07H…00H	00H.....00H	R7.....R0

STARTUP.A51 hello.c

```
11  }
12
13  void main(void)
14  {
15    unsigned char i=2,j=3,k;
16    UartInit();
17    printf("hello world!");
18
19    k=i*j;
20    printf("%d*%d=%d",i,j,k);
21    while(1);
22  }
```

Disassembly

```
    13: void main(void)
    14: {
    15:         unsigned char i=2,j=3,k;
C:0x03ED    750802   MOV      0x08,#0x02
C:0x03F0    750903   MOV      0x09,#0x03
    16:         UartInit();
C:0x03F3    120455   LCALL    UartInit(C:0455)
    17:         printf("hello world!");
    18:
C:0x03F6    7BFF     MOV      R3,#0xFF
C:0x03F8    7A04     MOV      R2,#0x04
C:0x03FA    793F     MOV      R1,#0x3F
C:0x03FC    120065   LCALL    PRINTF(C:0065)
    19:         k=i*j;
C:0x03FF    E508     MOV      A,0x08
C:0x0401    8509F0   MOV      B(0xF0),0x09
C:0x0404    A4       MUL      AB
    20:         printf("%d*%d=%d",i,j,k);
C:0x0405    7BFF     MOV      R3,#0xFF
C:0x0407    7A04     MOV      R2.#0x04
```

特殊功能寄存器——SP

堆栈的实质是一个特殊的RAM区。

堆栈指针SP：指示栈顶的位置。

单片机复位后，SP为初始值为07H。

说明：
初始的堆栈空间与工作寄存器区冲突。

字节地址	内容	
FFH … **81H** 80H	**0000 0111**	**SP**
7FH … **30H**	00H … 00H	
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH...18H	00H.....00H	R7.....R0
17H...10H	00H.....00H	R7.....R0
0FH...08H	00H.....00H	R7.....R0
07H...00H	00H.....00H	R7.....R0

SP—>

注意

- 在汇编主程序中需要执行：

  MOV SP, #30H

- 在Keil C51中的启动文件中执行，

  MOV SP,#?STACK-1

?STACK在链接时由编译器自动分配，Keil在链接阶段把?STACK段链接到片内RAM的最后一个数据段，以免出现栈溢出导致程序崩溃。

SP—>

字节地址	内容	
FFH … **81H** 80H	**07H**	**SP**
7FH … 30H	00H … 00H	
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH...18H	00H.....00H	R7.....R0
17H...10H	00H.....00H	R7.....R0
0FH...08H	00H.....00H	R7.....R0
07H...00H	00H.....00H	R7.....R0

Disassembly

```
    185:                    MOV     SP,#?STACK-1
    186:
    187: ; This code is required if you use L51_BANK.A51 with Banking Mode 4
    188: ;<h> Code Banking
    189: ; <q> Select Bank 0 for L51_BANK.A51 Mode 4
    190: #if 0
    191: ;     <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with Banking Mode 4.
    192: EXTRN CODE (?B_SWITCH0)
    193:                    CALL    ?B_SWITCH0      ; init bank mechanism to code bank 0
    194: #endif
    195: ;</h>
C:0x0443    758121    MOV       SP(0x81),#0x21
    196:                    LJMP    ?C_START
C:0x0446    020422    LJMP      main(C:0422)
C:0x0449    00        NOP
C:0x044A    00        NOP
```

hello.c | STARTUP.A51

```
180  IF PBPSTACK <> 0
181  EXTRN DATA (?C_PBP)
182                  MOV     ?C_PBP,#LOW PBPSTACKTOP
183  ENDIF
184
185                  MOV     SP,#?STACK-1
186
187  ; This code is required if you use L51_BANK.A51 with Banking Mode 4
188  ;<h> Code Banking
189  ; <q> Select Bank 0 for L51_BANK.A51 Mode 4
190  #if 0
191  ;     <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with Banking Mode 4.
192  EXTRN CODE (?B_SWITCH0)
193                  CALL    ?B_SWITCH0      ; init bank mechanism to code bank 0
194  #endif
195  ;</h>
196                  LJMP    ?C_START
```

Text Editor | Configuration Wizard

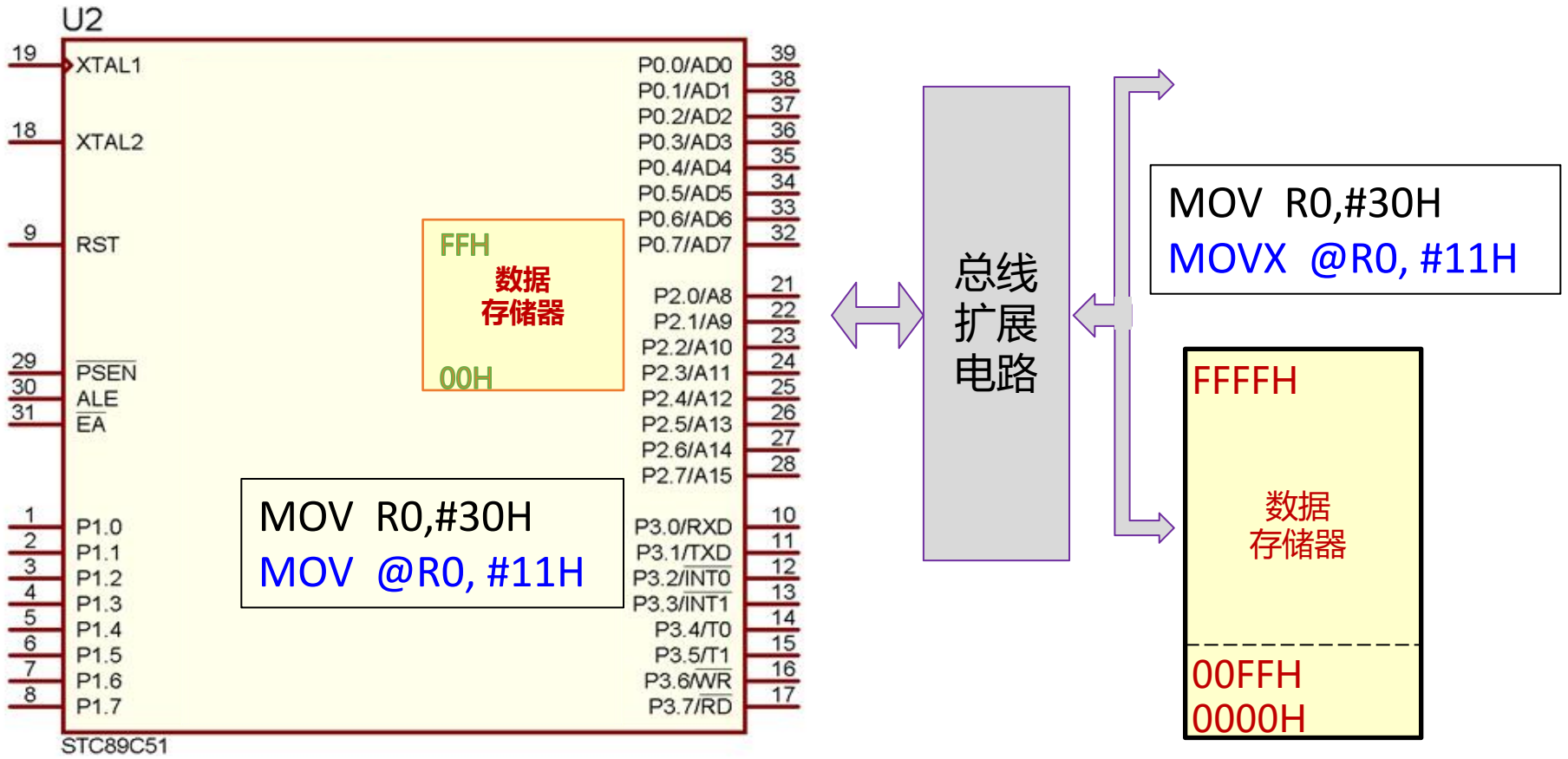
特殊功能寄存器——DPTR

- **数据指针寄存器DPTR**
- 复位时默认选择DPTR。
- DPTR即可作为16位寄存器使用，也可以DPL低8位和DPH高8位分开使用。

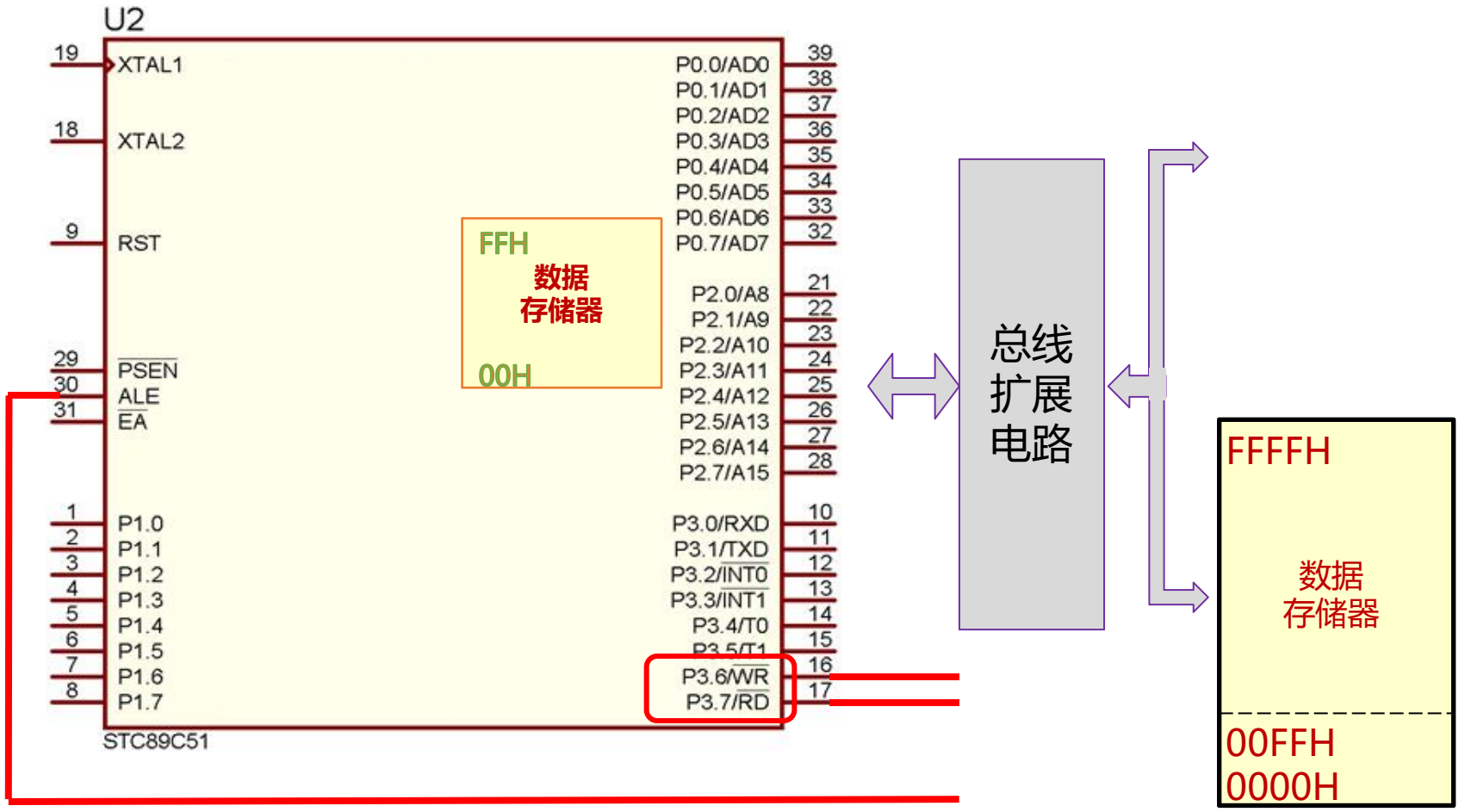
字节地址	内容	
FFH …		
83H **82H**	0000 0000 0000 0000	**DPH** **DPL**
… 80H		
7FH … 30H	00H … 00H	
2FH … 20H	00H … 00H	7FH.....78H …… 07H.....00H
1FH…18H	00H……..00H	R7……..R0
17H…10H	00H……..00H	R7……..R0
0FH…08H	00H……..00H	R7……..R0
07H…00H	00H……..00H	R7……..R0

2. 片外数据存储器：64KB（0000H~FFFFH）

- 当片内RAM不够用时最多可外扩64KBRAM，片内与片外是相互独立的；
- 片内与片外的低256字节的地址值相同，由于使用的是不同的访问指令，即片内用MOV指令，片外用MOVX指令，所以不会发生冲突。



片外数据存储器将在存储器扩展章节详述



ALE（地址锁存允许信号）

P3.6 $\overline{WR}$（片外数据存储器写选通）

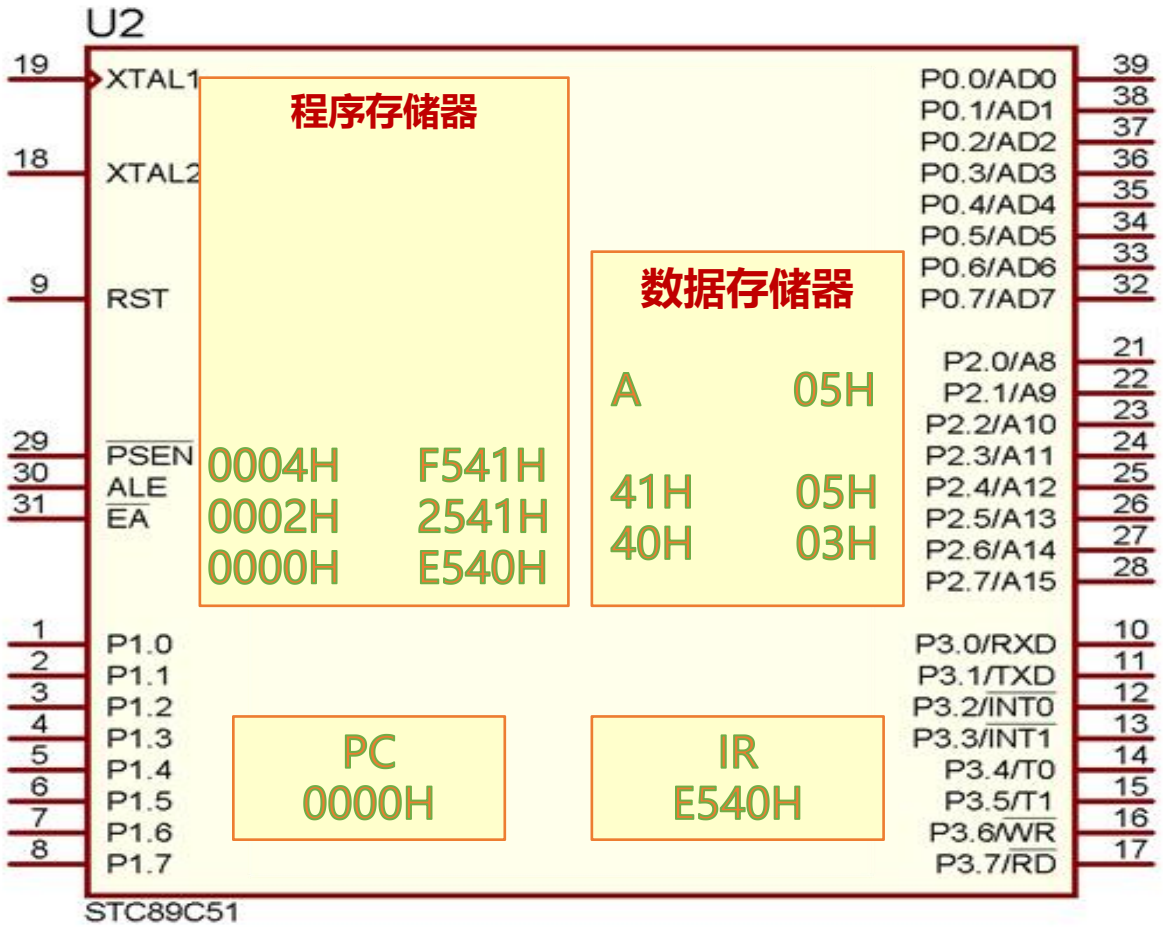
P3.7 $\overline{RD}$（片外数据存储器读选通）

汇编指令寻址方式

寻址方式是指在执行一条指令的过程中，寻找操作数或指令地址的方式。

```
MAIN( )
{   int i, j;
    j=i+j;
 }
```

```
        ORG     0000H
MAIN:   MOV     A, 40H
        ADD     A, 41H
        MOV     41H, A
        END
```



指令系统的寻址方式

- 一般来说，寻址方式更多是指操作数的寻址；
- 如果有两个操作数时，默认所指是源操作数的寻址方式。

- 寻址方式越多，功能越强，灵活性越大，指令系统越复杂。

```
        ORG    0000H
MAIN:
        MOV    A, 40H

        ADD    A, 41H
        MOV    41H, A
        END
```

```
        ORG    0000H
MAIN:   MOV    R0,#40H
        MOV    A, @R0
        MOV    R1,#41H
        ADD    A, @R1
        MOV    @R1, A
        END
```

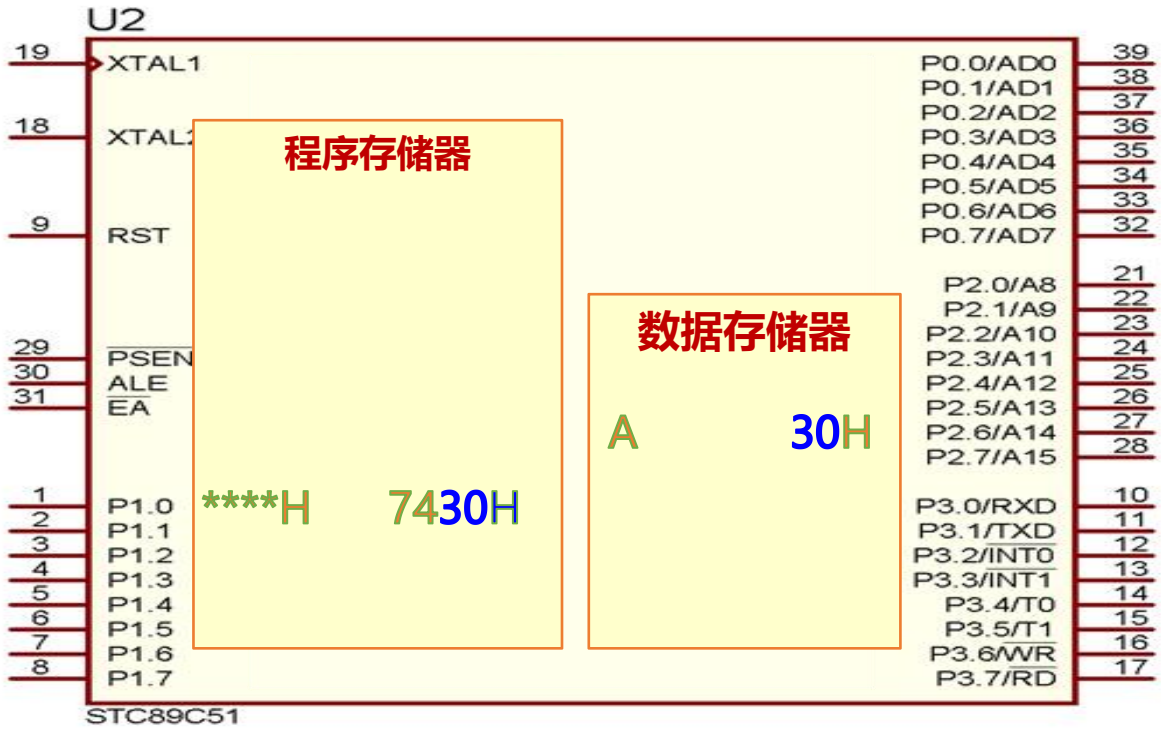
指令系统的寻址方式

序号	寻址方式	寻址空间
1	立即数寻址	程序存储器中的立即数
2	直接寻址	内部128字节RAM、特殊功能寄存器
3	寄存器寻址	R0～R7、A、B、C(位)、DPTR等
4	寄存器间接寻址	片内数据存储器、片外数据存储器
5	基址加变址寻址	读程序存储器固定数据和程序散转
6	相对寻址	程序存储器相对转移
7	位寻址	内部RAM中的可寻址位、SFR中的可寻址位

1.立即数寻址（操作数前加“#”）

直接在指令中给出操作数，即放在程序存储器内的常数。

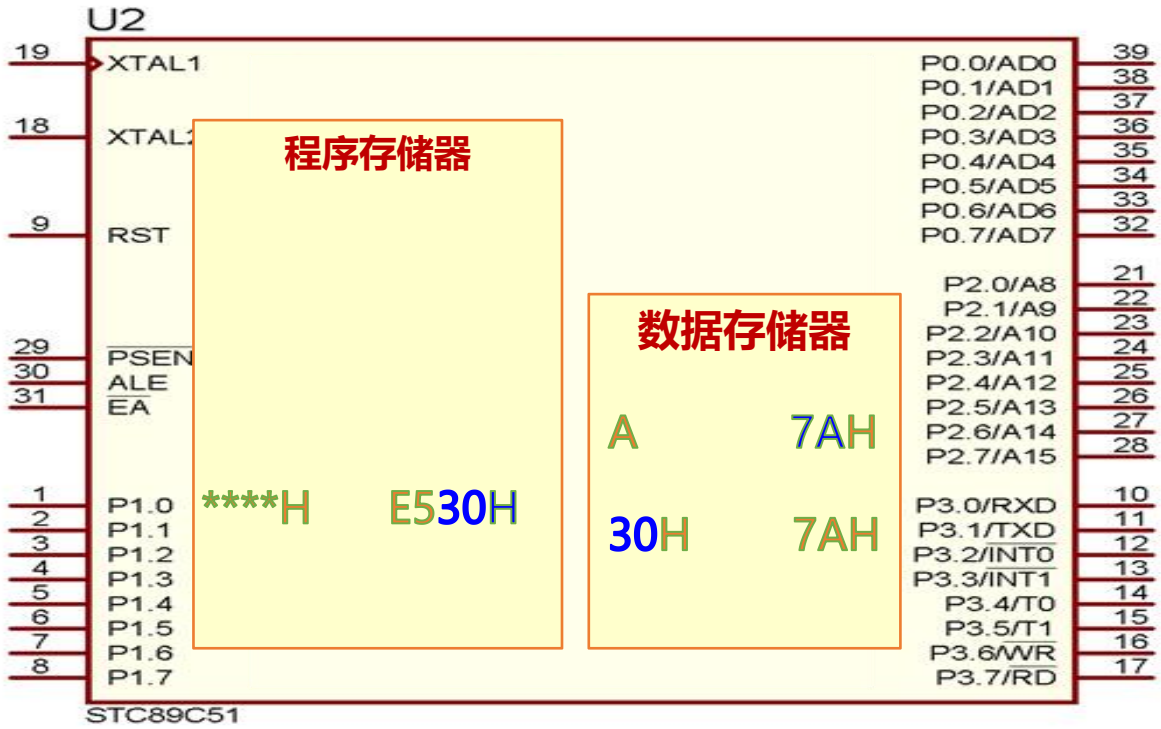
例：MOV A，**#30**H ；机器码7430H，把立即数30H送累加器A



2.直接寻址（操作数前无“#”，区别于立即数寻址）

指令中直接给出操作数的单元地址，而不是操作数，该单元地址中的内容才是真正的操作数。

例：MOV A，30H ；机器码E530H，把30H单元的内容送A



直接寻址方式访问的空间有：

1）内部RAM低128字节（00H－7FH）

指令中直接以单元地址的形式给出

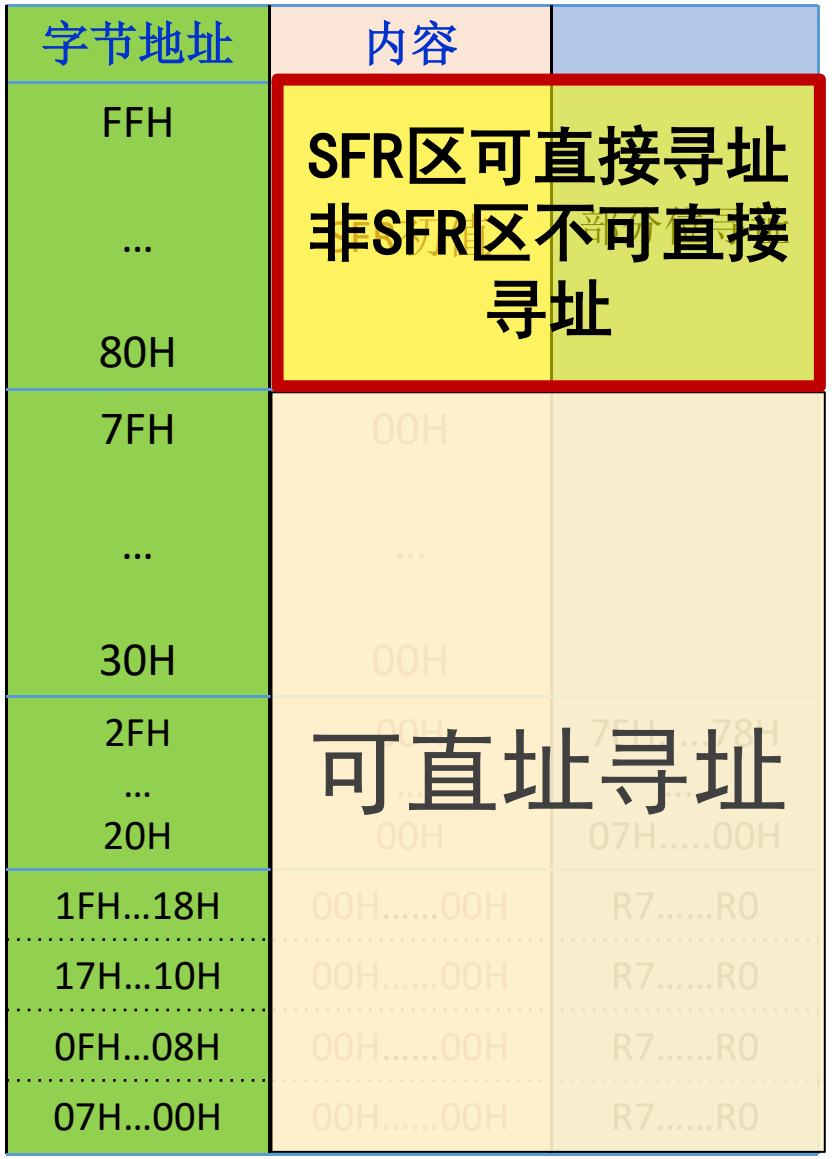
例如：MOV A,00H或MOV 30H,20H

2）特殊功能寄存器区（80H－0FFH）指令中可以以符号形式给出，也可以以单元地址形式给出。

例如：MOV A,80H等价于MOV A,P0

- 特别说明：直接寻址是访问片内所有特殊功能寄存器的唯一寻址方式

高128字节的非特殊功能寄存器区不能直接寻址



MOV A，30H



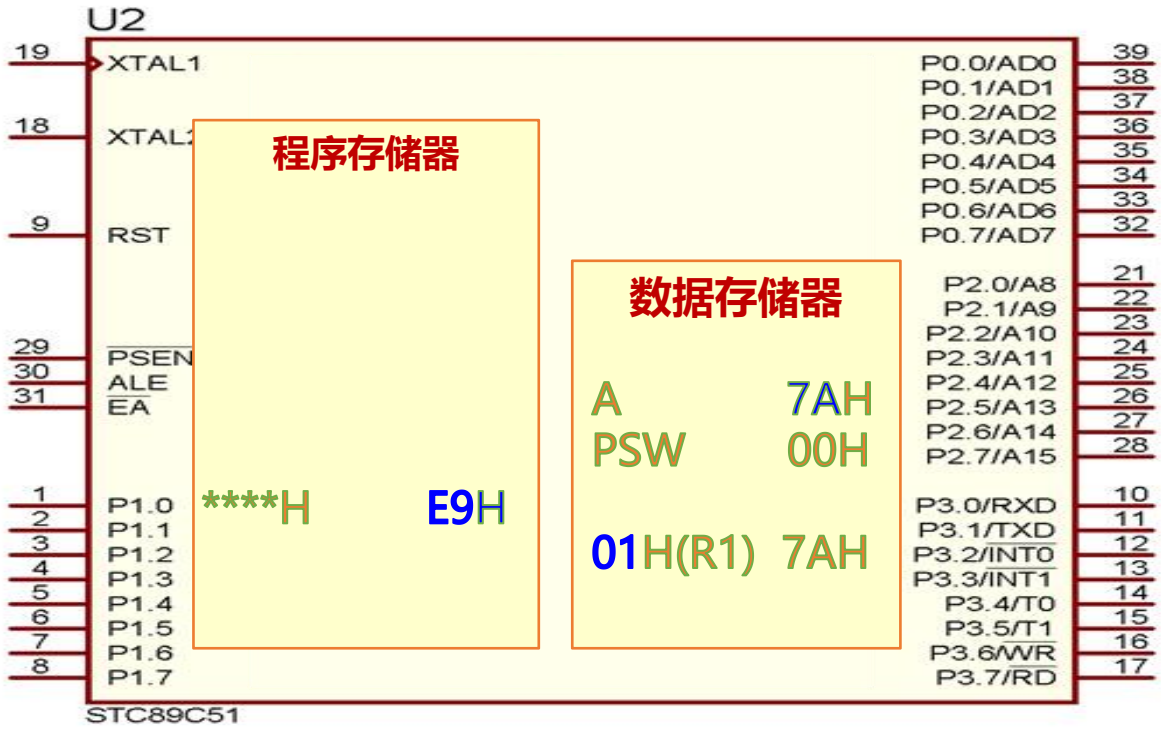


MOV A，#30H

3.寄存器寻址

指令中寄存器的内容是操作数。

例：MOV A，R1 ；机器码0E9H，把当前R1中的内容送A



寄存器包括

- 累加器A、寄存器B、数据指针DPTR、进位位CY等部分特殊功能寄存器；

- 32个4组通用工作寄存器组中的当前某组的R0~R7。

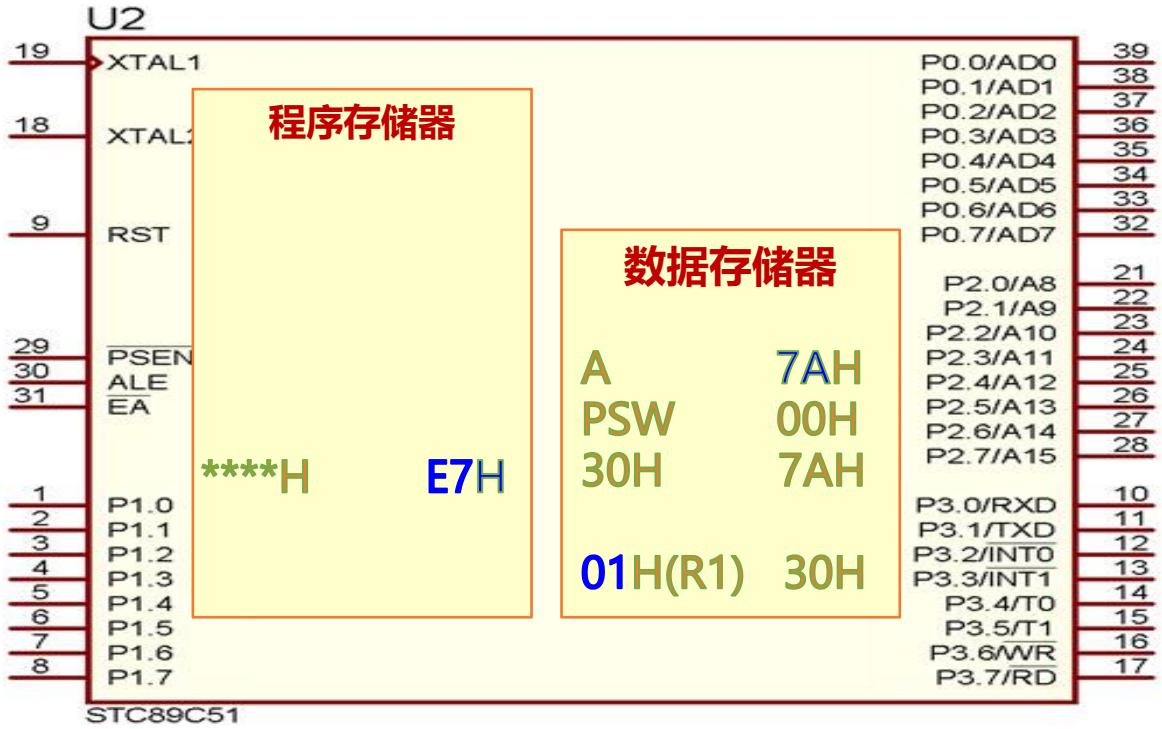
字节地址	内容	
F0H E0H PSW 83H 82H … 80H	SFR初值	**B** **A** **CY……** **DPH** **DPL**
7FH … 30H	00H … 00H	
2FH … 20H	00H … 00H	7FH…..78H …… 07H…..00H
1FH…18H	00H……00H	**R7……R0**
17H…10H	00H……00H	**R7……R0**
0FH…08H	00H……00H	**R7……R0**
07H…00H	00H……00H	**R7……R0**

4.寄存器间接寻址(寄存器名前加”@”,区别于寄存器寻址)

指令中寄存器的内容是操作数的地址，而不是操作数，从该地址中取出的数才是真正的操作数。

例：MOV A，@R1 ；机器码0E7H,

；把当前R1中的数作为地址所对应的存储单元内容送A



只有R0，R1，DPTR，SP能用于寄存器间接寻址

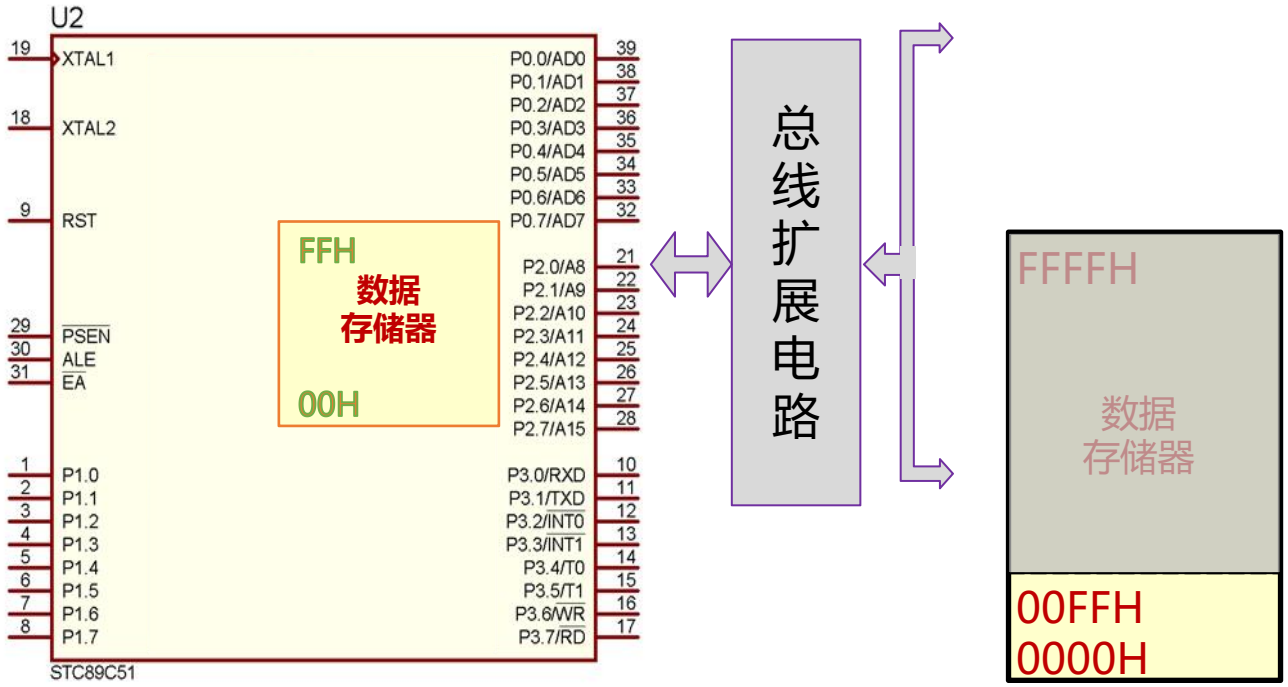
寄存器间接寻址的寻址范围

1）访问内部RAM或外部数据存储器的低256个字节时，可用R0或R1作为间址寄存器，形式为@Ri (i=0、1)。

例: MOV　　Ri, #**H

　　MOV　　A ,@Ri　;访问片内00H-FFH单元

　　MOVX　A ,@Ri　;访问片外00H-FFH单元



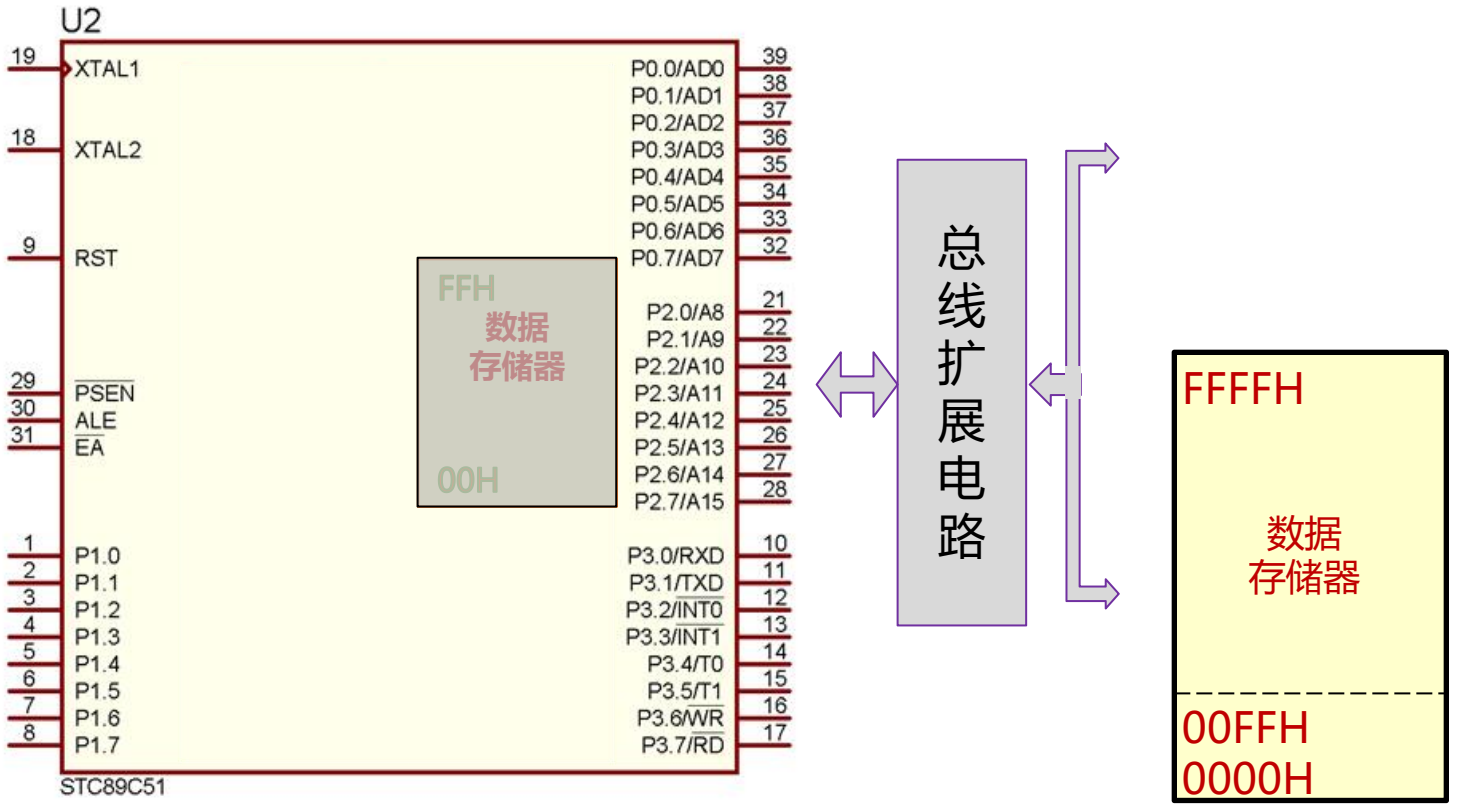
字节地址	内容
FFH … 80H	可间址寻址
7FH … 30H	
2FH … 20H	
1FH…18H	
17H…10H	
0FH…08H	
07H…00H	

寄存器间接寻址的寻址范围

2）访问片外数据存储器还可用数据指针DPTR作为间址寄存器，可对整个64K外部数据存储器空间寻址。

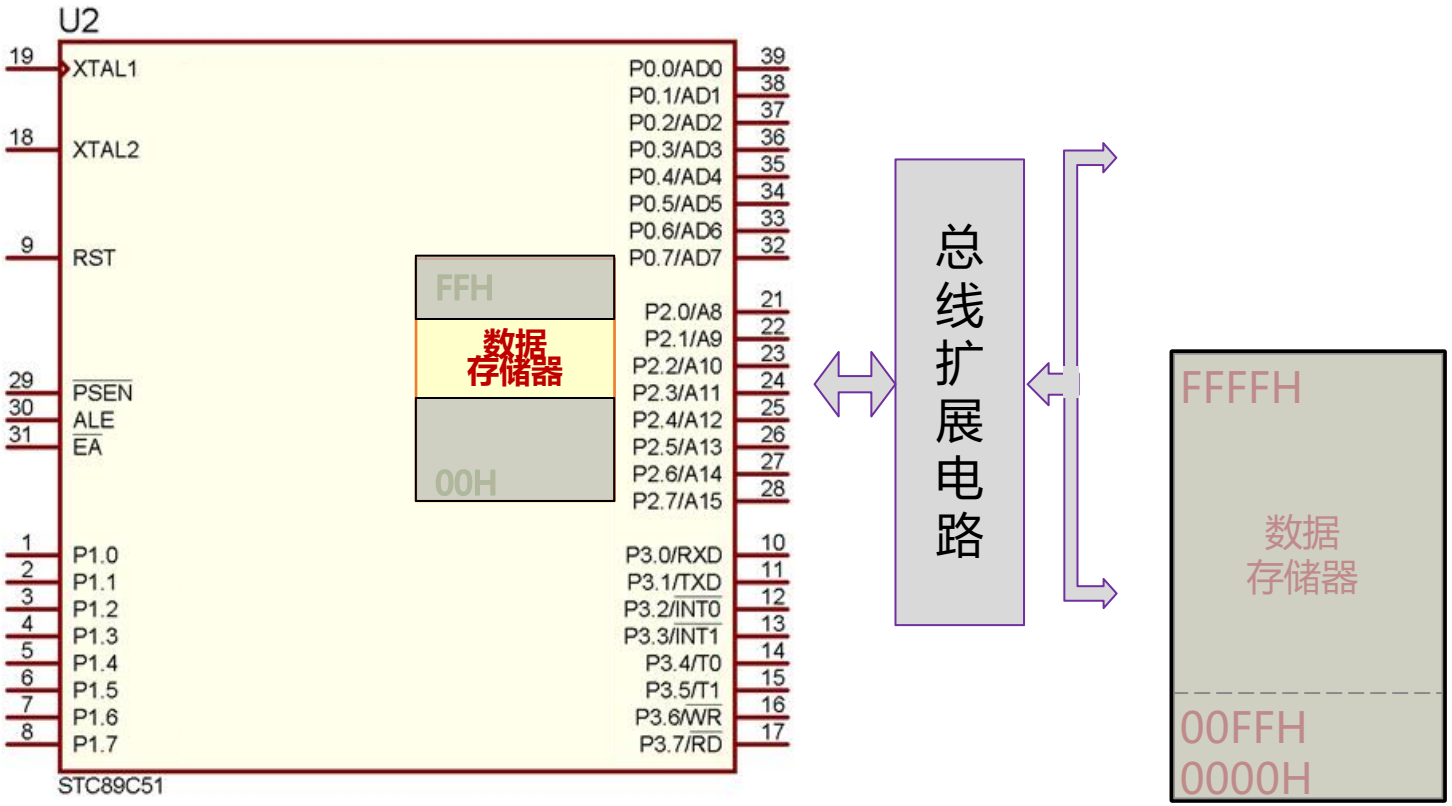
例: MOV　　DPTR,#****H

　　MOVX　A，@DPTR　　;访问片外0000H-FFFFH单元



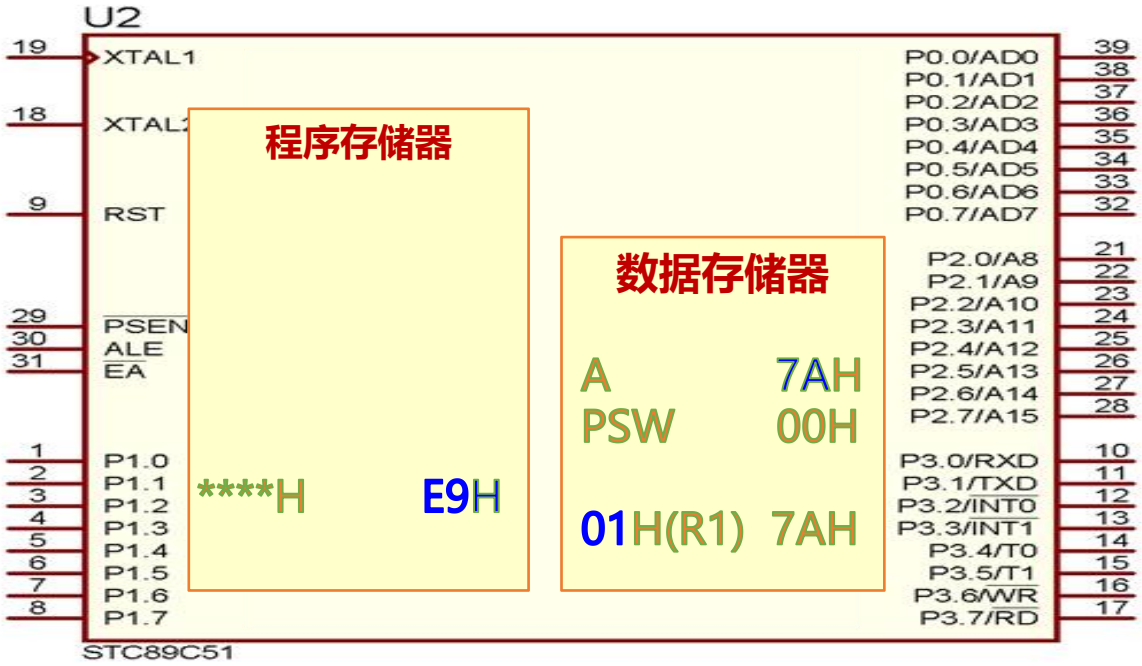
寄存器间接寻址的寻址范围

3）执行PUSH和POP指令时，堆栈指针SP是隐形操作数，使用SP作间址寄存器来进行对栈区的间接寻址。



MOV A，@R1





MOV A，R1

注意区别源操作数的四种寻址方式：

```
MOV     A,    30H
MOV     A,   #30H

MOV     A,    R1
MOV     A,   @R1
```

data与idata的区别

Disassembly

```
      3: void main()
      4: {
      5:          unsigned char data i;
      6:          i=0x11;
C:0x000F    750811    MOV       0x08,#0x11
      7: }
C:0x0012    22        RET
C:0x0013    00        NOP
C:0x0014    00        NOP
```

直接寻址方式赋值

regpsw.c | STARTUP.A51

```
1 #include <reg52.h>
2
3 void main()
4 {
5   unsigned char data i;
6   i=0x11;
7 }
```

等价于 unsigned char i;

Memory 1

Address: I:00h

```
I:0x00: 00 00 00 00 00 00 00 00 11 00 00 00 0
I:0x24: 00 00 00 00 00 00 00 00 00 00 00 00 0
I:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 0
I:0x6C: 00 00 00 00 00 00 00 00 00 00 00 00 0
```

Call Stack + Locals | Memory 1

Disassembly

```
      3: void main()
      4: {
      5:          unsigned char idata i;
      6:          i=0x11;
C:0x000F    7808      MOV       R0,#0x08
C:0x0011    7611      MOV       @R0,#0x11
      7: }
C:0x0013    22        RET
C:0x0014    00        NOP
```

间接寻址方式赋值

regpsw.c | STARTUP.A51

```
1 #include <reg52.h>
2
3 void main()
4 {
5   unsigned char idata i;
6   i=0x11;
7 }
```

Memory 1

Address: I:00h

```
I:0x00: 08 00 00 00 00 00 00 00 11 00 00 00 0
I:0x24: 00 00 00 00 00 00 00 00 00 00 00 00 0
I:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 0
I:0x6C: 00 00 00 00 00 00 00 00 00 00 00 00 0
```

Call Stack + Locals | Memory 1

	地址		内容
第3组	1FH … 18H	R7 … R0	00H … 00H
第2组	17H … 10H	R7 … R0	00H … 00H
第1组	0FH … **08H**	R7 … **R0**	00H … **11H**
第0组	07H … 00H	R7 … R0	00H … 00H