# 数传算逻指令与顺序结构

信息工程学院　　陈梅

# 第4讲-2 传送算逻指令和顺序结构 重点

要点

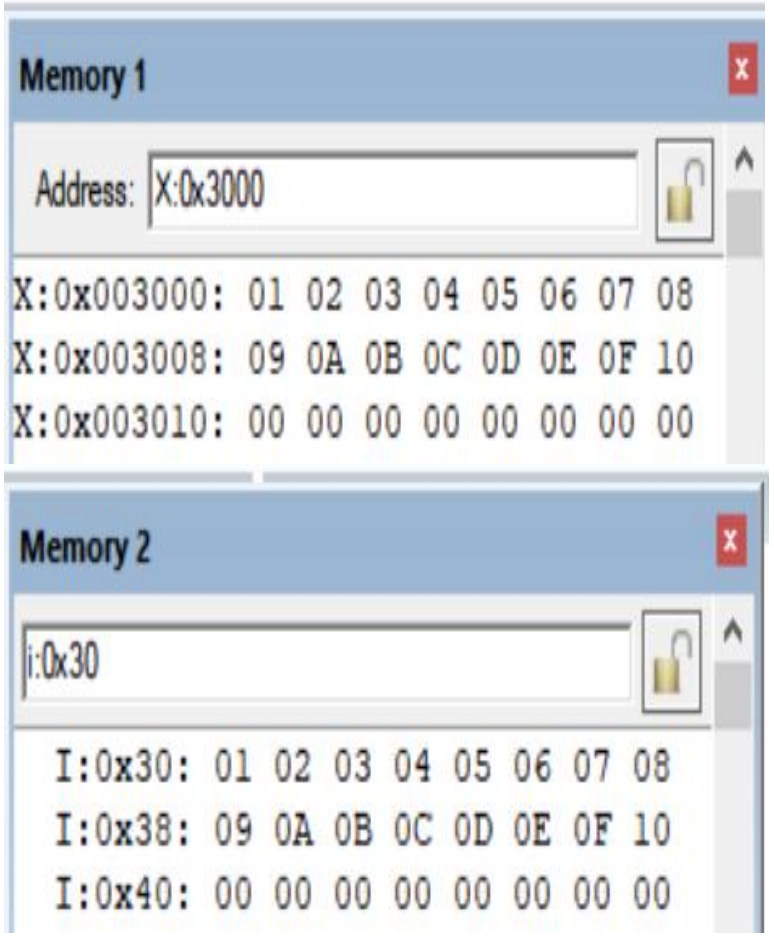
- 数据传送指令
- 算逻运算指令
- 数据传送位操作指令
- 算逻运算位操作指令
- 顺序结构举例

# 引课——编程举例

【例3- 24&3-31】设单片机片内数据存储器首地址为30H，片外数据存储器首地址为3000H，存取数据字节个数16个。先将片内数据存储区的这16个字节的内容设置为01H~10H，再将片内首地址为30H开始的16个单元的内容传送到片外首地址为3000H开始的数据存储区中保存。

MOVDATAC.C

```
1  #include<reg52.h>
2  #define LENTH 16
3  unsigned char idata dADDR[LENTH] _at_ 0x30;
4  unsigned char xdata xADDR[LENTH] _at_ 0x3000;
5
6  void main()
7  {
8      unsigned int i;
9
10     for(i=0;i<LENTH;i++)
11         dADDR[i]=i+1;
12
13     for(i=0;i<LENTH;i++)
14         xADDR[i]=dADDR[i];
15 }
```



```
4           ORG     0000H
5   DADDR   EQU     30H             ;片内数据区首地址
6   XADDR   EQU     3000H           ;片外数据区首地址
7   COUNT   EQU     10H             ;传送数据大小，共16个字节
8   MAIN:   MOV     SP,#60H         ;重置堆栈指针
9           MOV     R0,#DADDR       ;设置片内数据区首地址
10          MOV     R2,#COUNT       ;设置传送数据区大小即16个字节
11  /**********片内数据区初始化***********/
12  Init:  MOV     A,#01H
13  LOOP1: MOV     @R0,A
14         INC     A
15         INC     R0
16         DJNZ    R2,LOOP1
17  /************片内外数据传送*************/
18  DXMOV:    MOV     R0,#DADDR     ;设置片内数据区首地址
19            MOV     DPTR,#XADDR   ;设置片外数据区首地址
20            MOV     R2,#COUNT     ;设置传送数据区大小即16个字节
21  LOOP2:    MOV     A,@R0
22            MOVX    @DPTR,A
23            INC     R0
24            INC     DPTR
25            DJNZ    R2,LOOP2
26            END
```

**Memory 1**

D:30H

```
D:0x30: 01 02 03 04 05 06 07 08
D:0x38: 09 0A 0B 0C 0D 0E 0F 10
D:0x40: 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00
```

**Memory 2**

Address: X:3000H

```
X:0x003000: 01 02 03 04 05 06 07 08
X:0x003008: 09 0A 0B 0C 0D 0E 0F 10
X:0x003010: 00 00 00 00 00 00 00 00
X:0x003018: 00 00 00 00 00 00 00 00
```

# STC89C52单片机汇编指令分类介绍

STC89C52指令系统与传统MCS-51单片机的指令完全兼容

按功能分为五类：

（1）数据传送类
（2）算术运算类
（3）逻辑操作类
（4）控制转移类
（5）位操作类

（1）数据传送类&其位操作
（2）算术运算类&其位操作
（3）逻辑操作类&其位操作
（4）控制转移类&其位操作

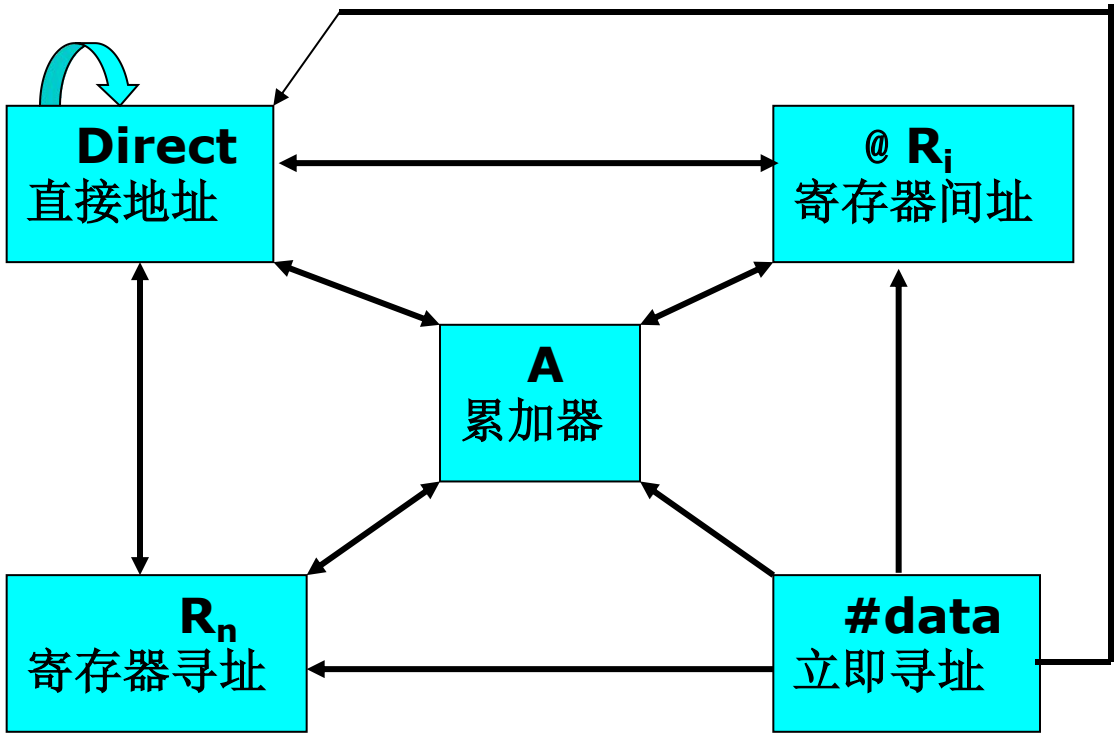
## 3.2.1 数据传送类指令

- 数据传送类指令是使用最频繁的指令，有MOV、MOVX、MOVC、XCH、XCHD、SWAP、PUSH、POP 这8种。

- 数据传送属"复制"性质，而不是"搬家"，即该类指令执行后，源操作数不变，目的操作数被源操作数取代。

- 数据传送类指令不影响进位标志位Cy、辅助进位标志位Ac和溢出标志位OV，但不包括检验累加器奇偶性的标志位P。

# MCS-51单片机片内数据传送途径




1、目的地址 A

2、目的地址 Rn

3、目的地址 Direct

4、目的地址 @Ri

对于所有MOV类指令，累加器A是一个特别重要的8位寄存器，CPU对它具有其他寄存器所没有的操作指令。

# 1. 内部RAM传送指令MOV

- 以累加器A为目的操作数

```
MOV  A，Rn          ；A← Rn
MOV  A，direct      ；A←(direct)
MOV  A，@Ri         ；A←((Ri))
MOV  A，#data       ；A← #data
```

```
MOV  A,R6       ；源操作数寄存器寻址
MOV  A,@R0      ；源操作数间接寻址
MOV  A,70H      ；源操作数直接寻址
MOV  A,#78H     ；源操作数立即寻址
```

- 以直接地址为目的操作数

```
MOV  direct，A          ; direct ←(A)
MOV  direct，Rn         ; direct ←(Rn)
MOV  direct1，direct2 ; direct1←(direct2)
MOV  direct，@Ri        ; direct ←((Ri))
MOV  direct，#data      ; direct ← data
```

```
MOV  30H,A      ;源操作数寄存器寻址
MOV  30H,R0     ;源操作数寄存器寻址
MOV  30H,70H    ;源操作数直接寻址
MOV  30H,@R0    ;源操作数间接寻址
MOV  30H,#78H   ;源操作数立即寻址
```

# 1. 内部RAM传送指令MOV

- 以Rn为目的操作数 (n=0~7)

  MOV Rn，A ; Rn ←(A)

  MOV Rn，direct ; Rn ←(direct)

  MOV Rn，#data ; Rn ←data

*MOV Rn, @Ri*

*MOV Rn，Rn*

- 以Ri间接地址为目的操作数(i=0,1)

  MOV @Ri,A ;((Ri))←A

  MOV @Ri,direct ;((Ri))←direct

  MOV @Ri,#data ;((Ri))←data

*MOV @Ri，Rn*

*MOV @Ri，@Ri*

# 1. 内部RAM传送指令MOV

- 16位数传送指令

MOV DPTR，#data16

；把高8位立即数送入DPH，低8位立即数送入DPL

；地址指针DPTR由DPH和DPL组成。

**整个指令系统唯一的一条16位数据传送指令，用来设置地址指针。**

例：**MOV DPTR, #3000H** 一般用于接着执行下列指令

MOVX A, @DPTR 或 MOVX @DPTR, A

# 2. 外部RAM传送指令MOVX

- MOV后面加“X”，表示访问的是片外RAM或I/O口;
- 必须通过累加器A与外部RAM或I/O间相互传送数据。

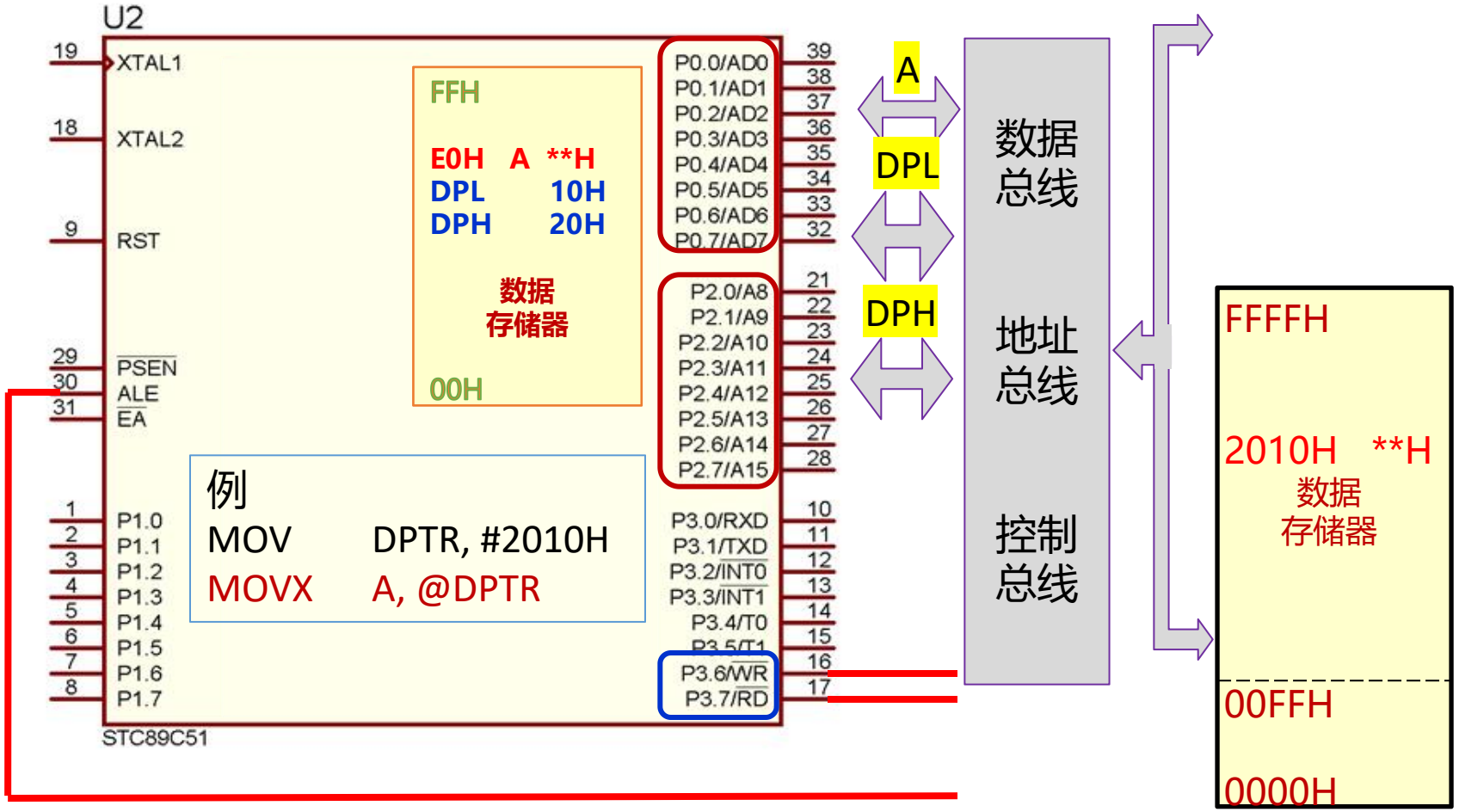
```
MOVX  A，@Ri          ; A←（（Ri））
MOVX  A，@DPTR        ; A←（（DPTR））
```

这两条指令是单片机读取外部RAM或I/O的数据，此时引脚P3.7（$\overline{RD}$）有效。

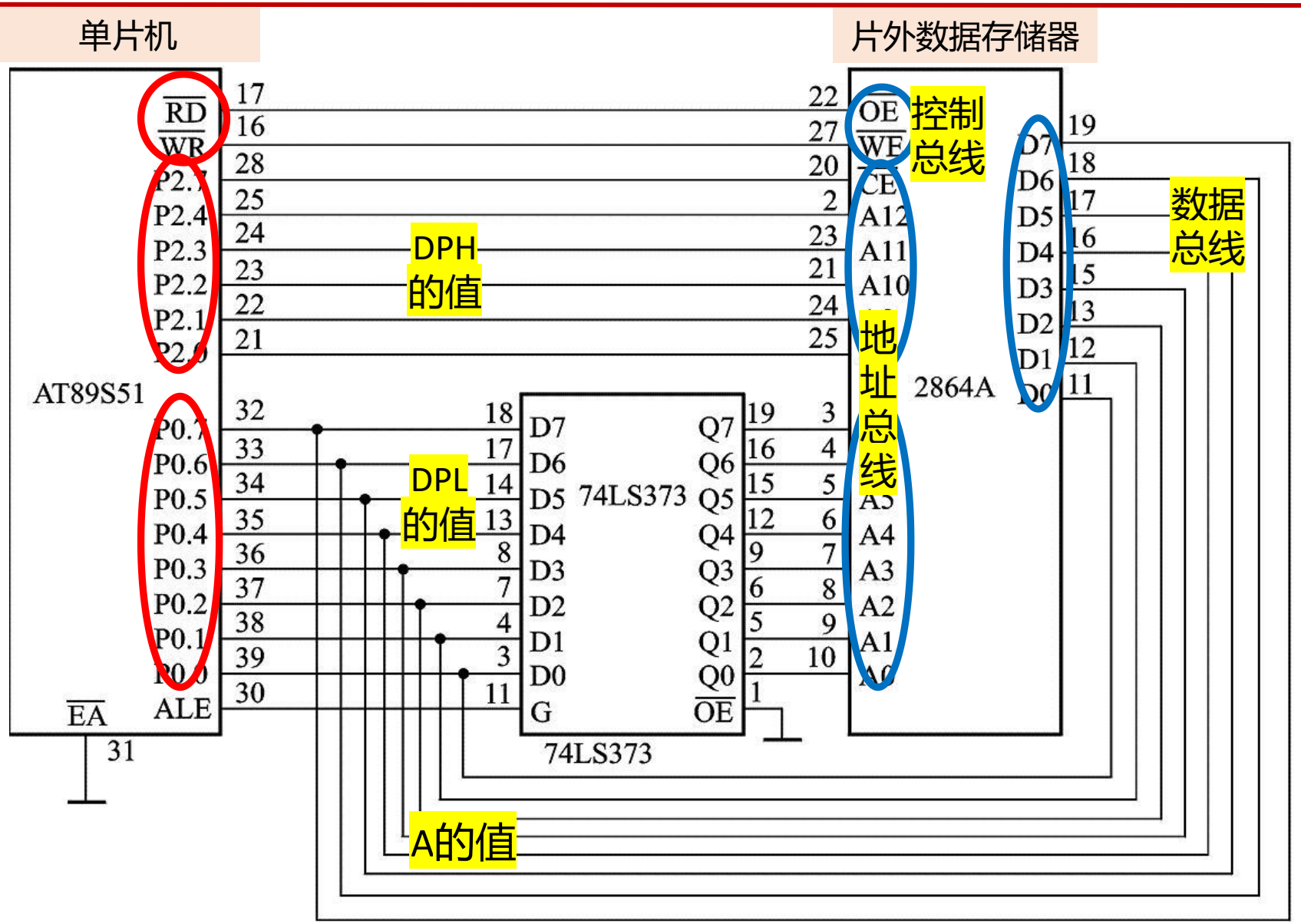
```
MOVX  @Ri，A          ;（（Ri））  ← A
MOVX  @DPTR，A        ;（（DPTR））← A
```

这两条指令是单片机写入数据到外部RAM或I/O，此时引脚P3.6（$\overline{WR}$）有效。

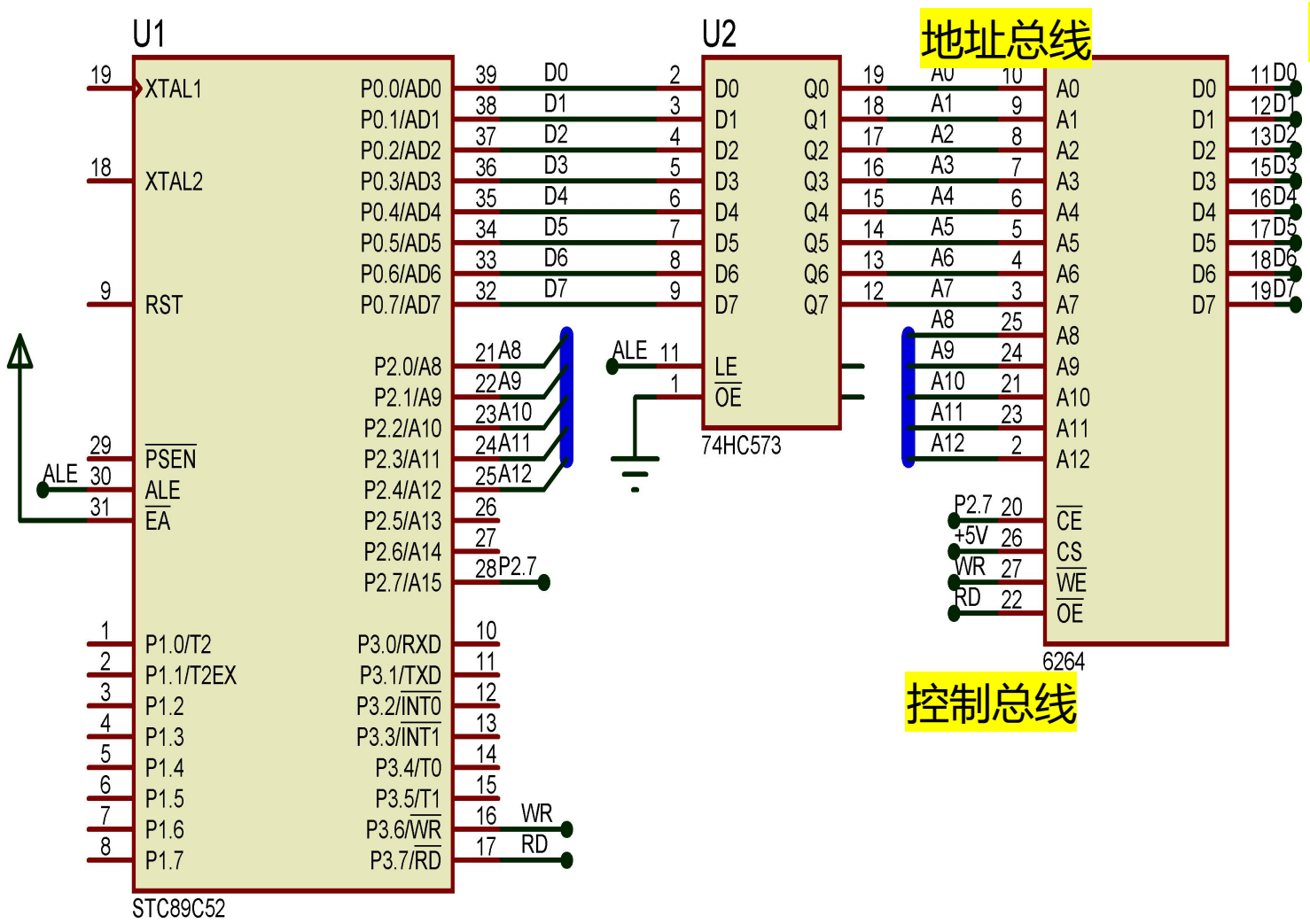
# MOVX @DPTR,A 或 MOVX A,@DPTR

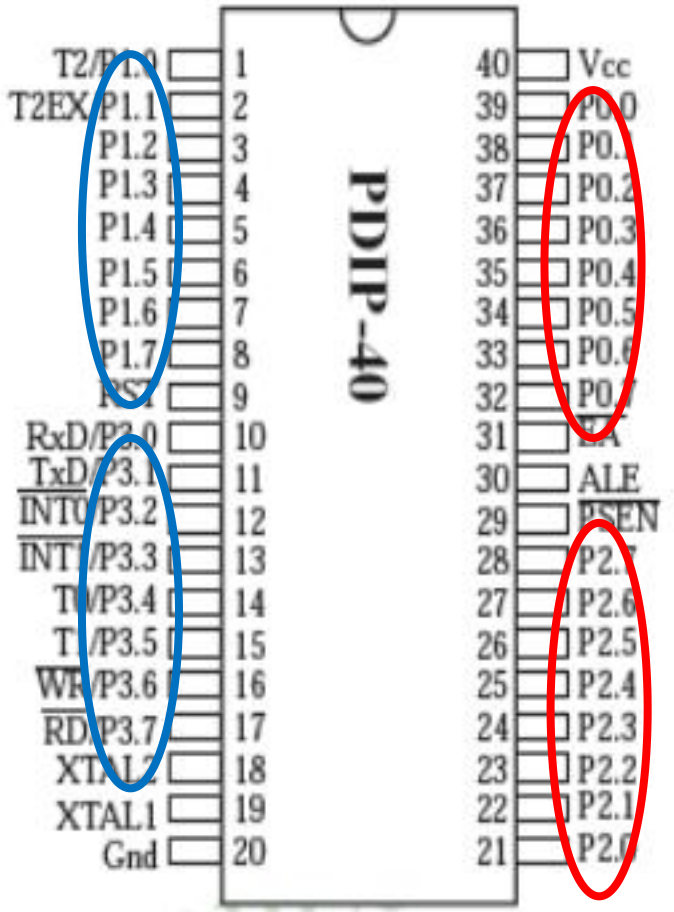



用16位DPTR作间接寻址的，可寻址片外RAM整个64K。
高8位地址(A15~A8)由P2输出，低8位地址(A7~A0)由P0输出。

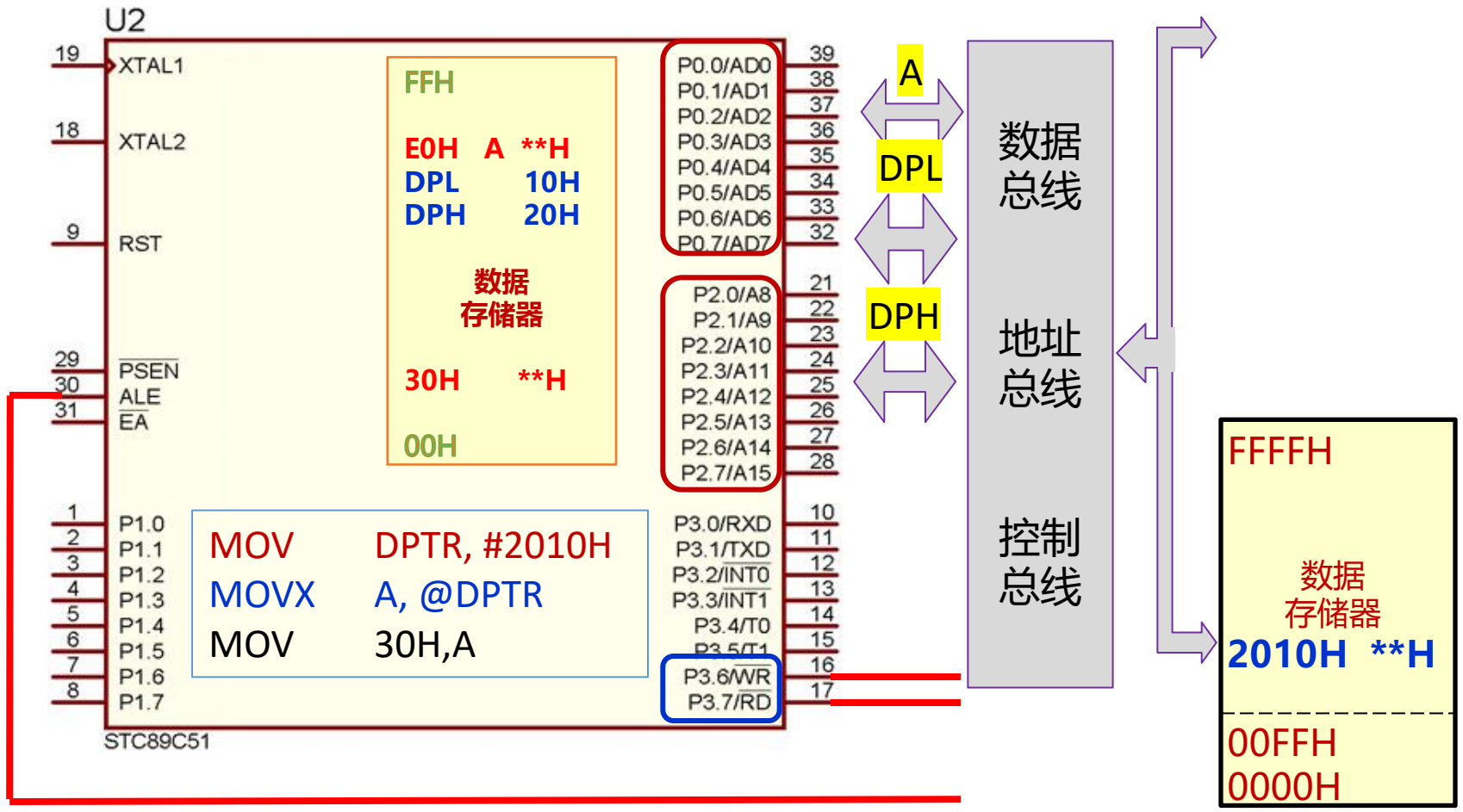

单片机
片外数据存储器
AT89S51
2864A
74LS373
74LS373
RD
WR
P2.4
P2.3
P2.2
P2.1
P0.6
P0.5
P0.4
P0.3
P0.2
EA
ALE
31
OE
WE
CE
A12
A11
A10
A4
A3
A2
A1
D7
D6
D5
D4
D3
D2
D1
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
G
DPH
的值
DPL
的值
A的值
控制
总线
地
址
总
线
数据
总线

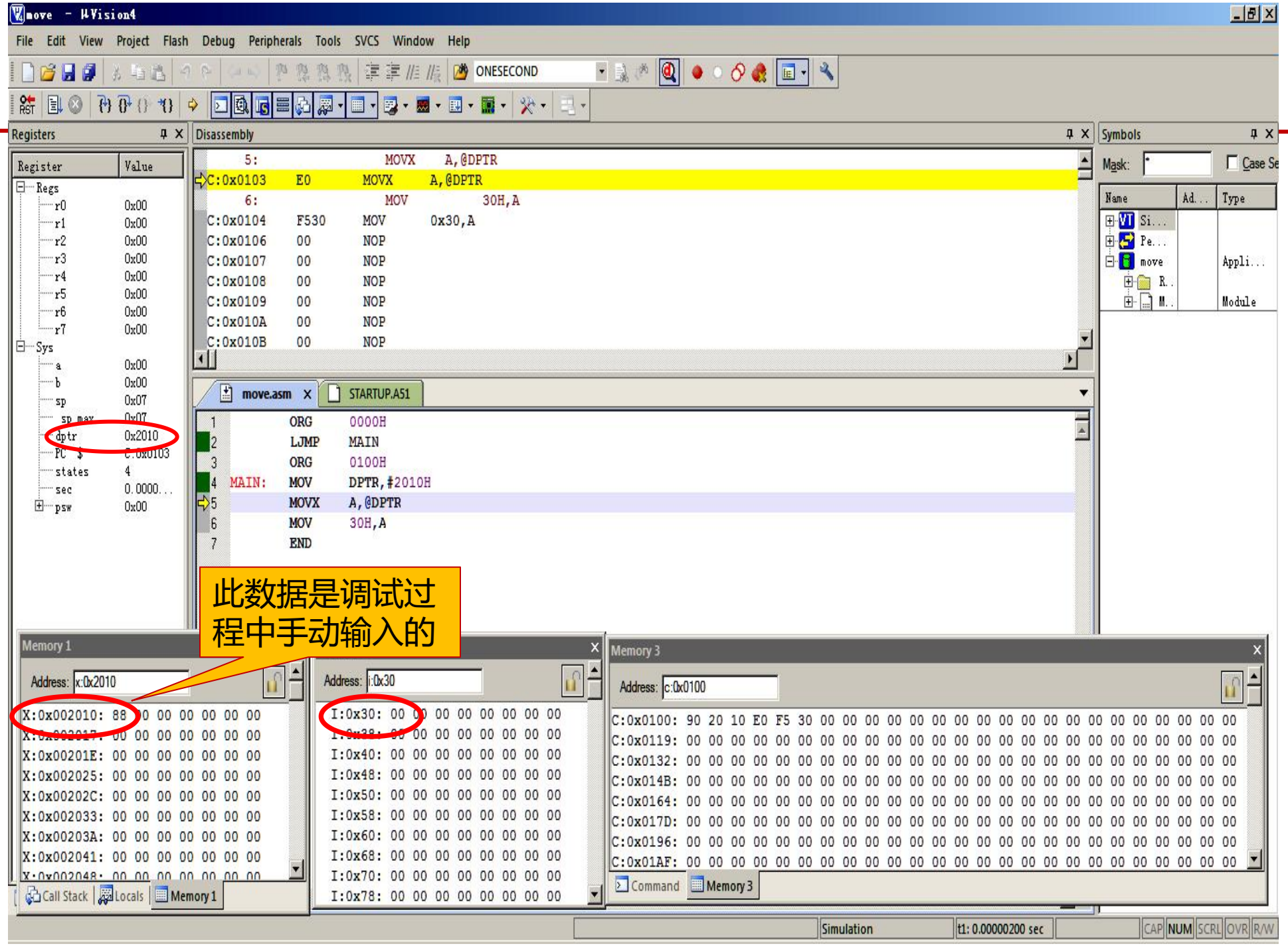
# 详细的扩展电路讲解将于第8章存储器扩展讲授

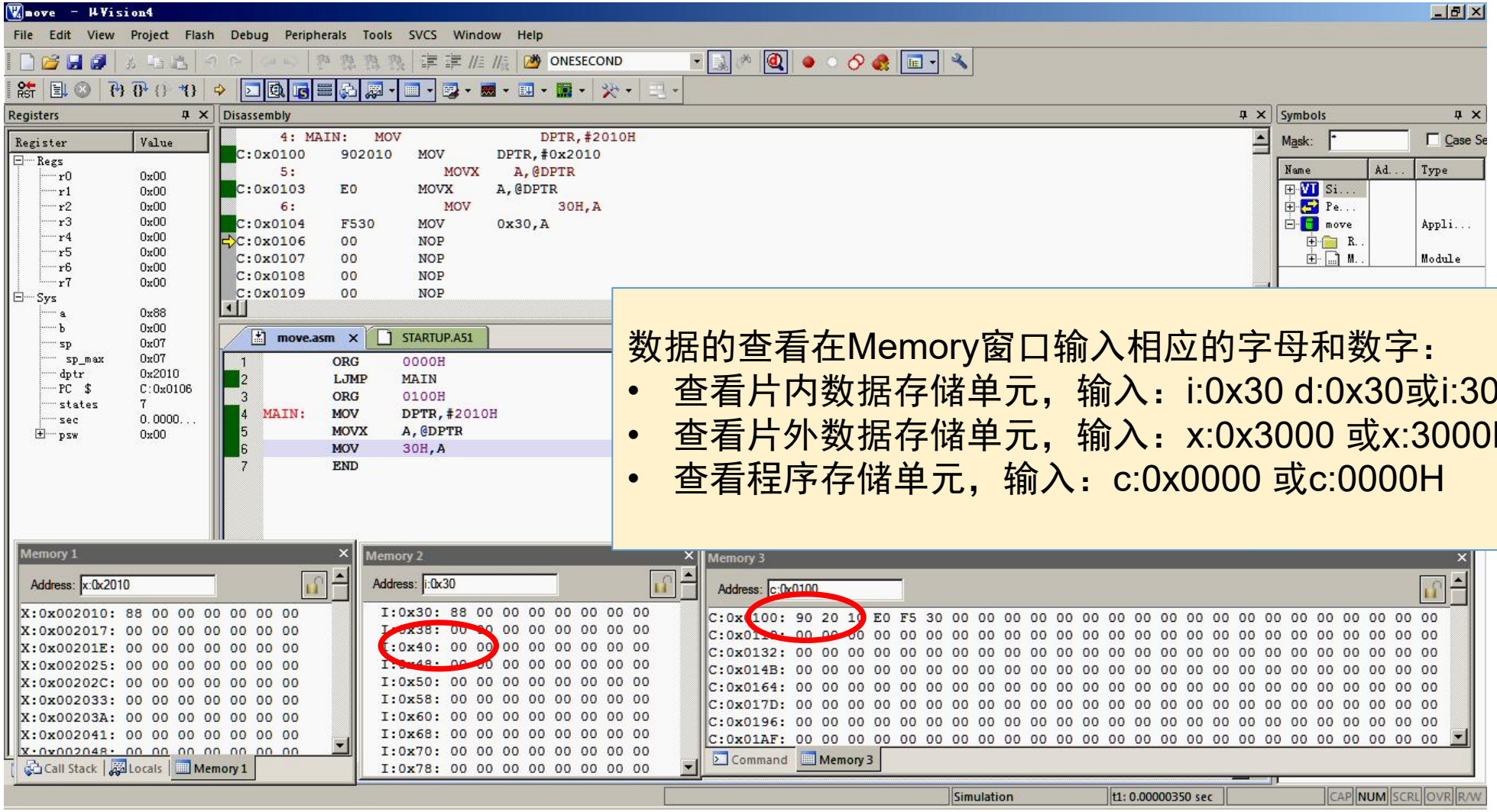


# 例：将片外2010H单元内容送到片内30H单元中。



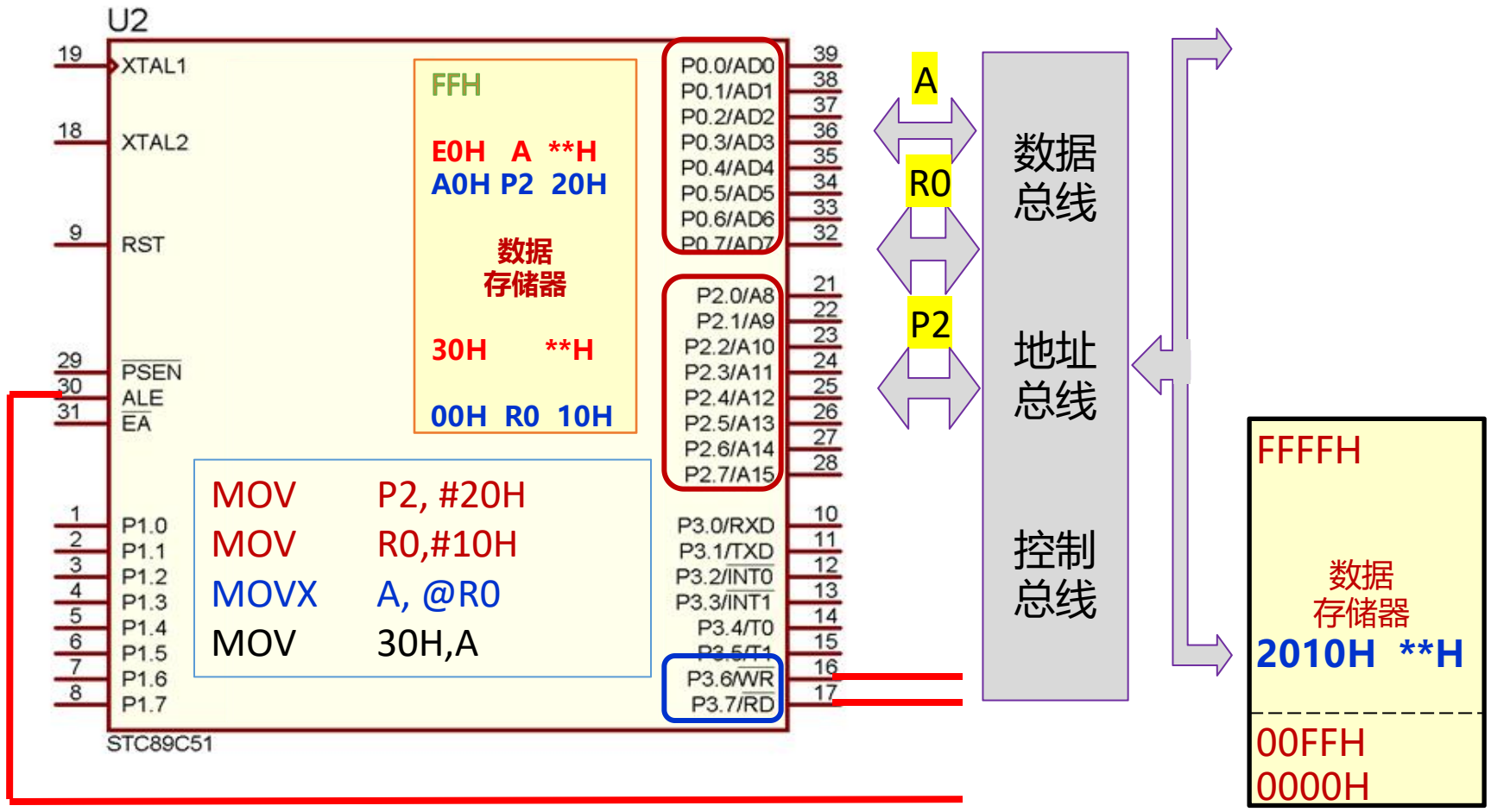

move - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
ONESECOND
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0x00
b 0x00
sp 0x07
sp_max 0x07
dptr 0x2010
PC $ C:0x0103
states 4
sec 0.0000...
psw 0x00
Disassembly
5: MOVX A,@DPTR
C:0x0103 E0 MOVX A,@DPTR
6: MOV 30H,A
C:0x0104 F530 MOV 0x30,A
C:0x0106 00 NOP
C:0x0107 00 NOP
C:0x0108 00 NOP
C:0x0109 00 NOP
C:0x010A 00 NOP
C:0x010B 00 NOP
Symbols
Mask: *
Case Se
Name Ad... Type
Si...
Pe...
move Appli...
R..
M.. Module
move.asm
STARTUP.A51
1 ORG 0000H
2 LJMP MAIN
3 ORG 0100H
4 MAIN: MOV DPTR,#2010H
5 MOVX A,@DPTR
6 MOV 30H,A
7 END
此数据是调试过程中手动输入的
Memory 1
Address: x:0x2010
X:0x002010: 88 00 00 00 00 00 00 00
X:0x002017: 00 00 00 00 00 00 00 00
X:0x00201E: 00 00 00 00 00 00 00 00
X:0x002025: 00 00 00 00 00 00 00 00
X:0x00202C: 00 00 00 00 00 00 00 00
X:0x002033: 00 00 00 00 00 00 00 00
X:0x00203A: 00 00 00 00 00 00 00 00
X:0x002041: 00 00 00 00 00 00 00 00
X:0x002048: 00 00 00 00 00 00 00 00
Call Stack Locals Memory 1
Address: i:0x30
I:0x30: 00 00 00 00 00 00 00 00
I:0x38: 00 00 00 00 00 00 00 00
I:0x40: 00 00 00 00 00 00 00 00
I:0x48: 00 00 00 00 00 00 00 00
I:0x50: 00 00 00 00 00 00 00 00
I:0x58: 00 00 00 00 00 00 00 00
I:0x60: 00 00 00 00 00 00 00 00
I:0x68: 00 00 00 00 00 00 00 00
I:0x70: 00 00 00 00 00 00 00 00
I:0x78: 00 00 00 00 00 00 00 00
Memory 3
Address: c:0x0100
C:0x0100: 90 20 10 E0 F5 30 00 00 00 00 00 00 00 00 00 00
C:0x0119: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0132: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x014B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0164: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x017D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0196: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x01AF: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Command Memory 3
Simulation t1: 0.00000200 sec
CAP NUM SCRL OVR R/W



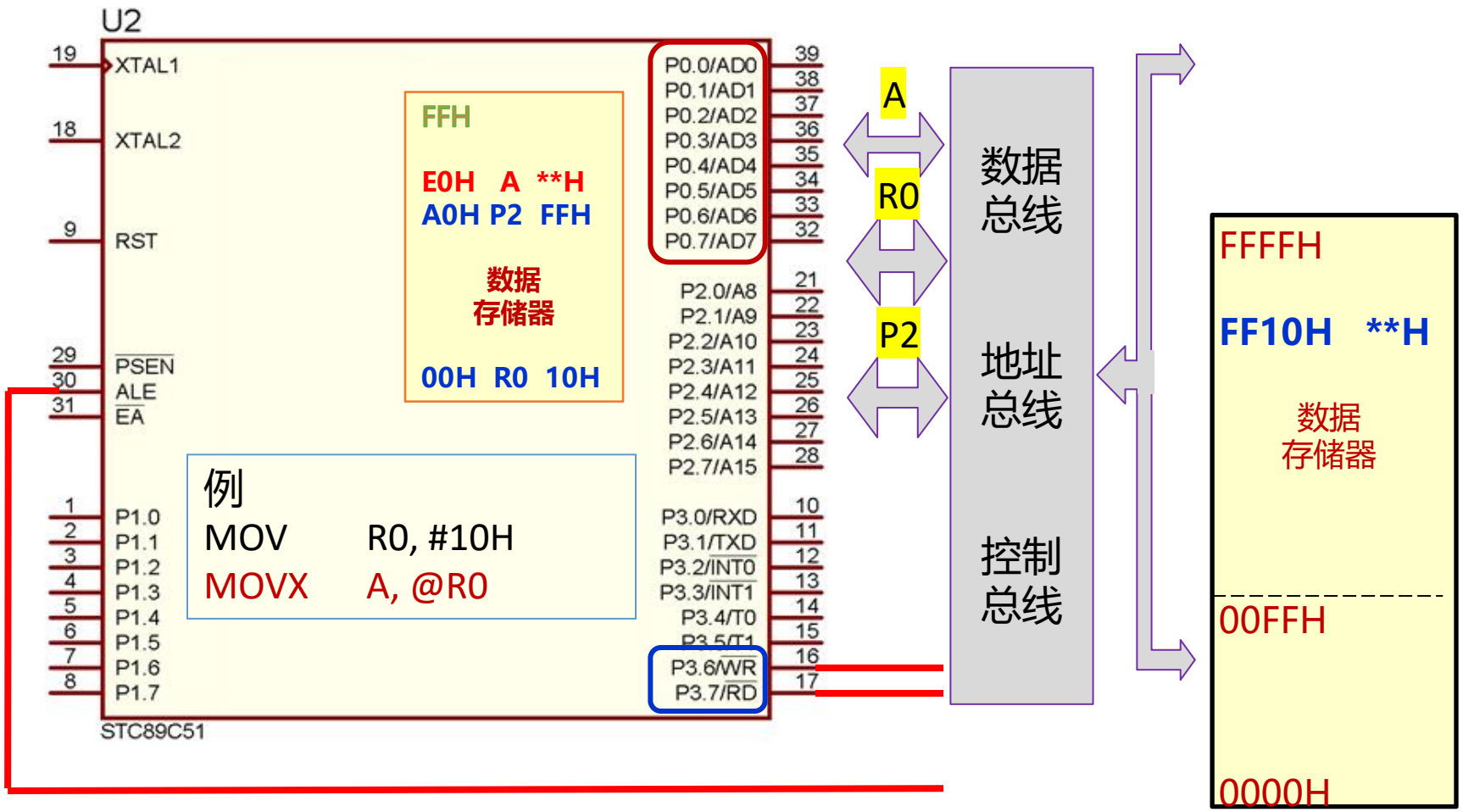

数据的查看在Memory窗口输入相应的字母和数字：

- 查看片内数据存储单元，输入：i:0x30 d:0x30或i:30H d:30H
- 查看片外数据存储单元，输入：x:0x3000 或x:3000H
- 查看程序存储单元，输入：c:0x0000 或c:0000H

# 例：将片外2010H单元内容送到片内30H单元中。



# MOVX @Ri,A 或 MOVX A,@Ri




用8位Ri作间接寻址的，可寻址片外RAM的256字节。
这8位地址(A7~A0)由P0口输出。P2口为原状态。

# 3. 程序存储器访问指令MOVC——又称查表指令

获取程序存储器中的常数的指令。MOV的后面加“C”，是CODE的第一个字母，即表示程序存储器中的代码。

单片机的 $\overline{\text{PSEN}}$ 引脚使用程序存储器读有效，程序存储器只读不写，传送为单向，仅有以下两条：

①**MOVC A,@A+DPTR**；PC=DPTR基址+A变址，

；把由该PC地址的内容——>累加器A

②**MOVC A,@A+PC** ；PC=PC基址+变址A，

；把由该地址的内容——>累加器A。

无 MOVC @A+DPTR,A 和 MOVC @A+PC,A

# 由以下两例说明这两条指令的特点

例3-9：设(DPTR)=1100H，(A)=40H，执行1000H处的指令。

1000H：MOVC A，@A+DPTR

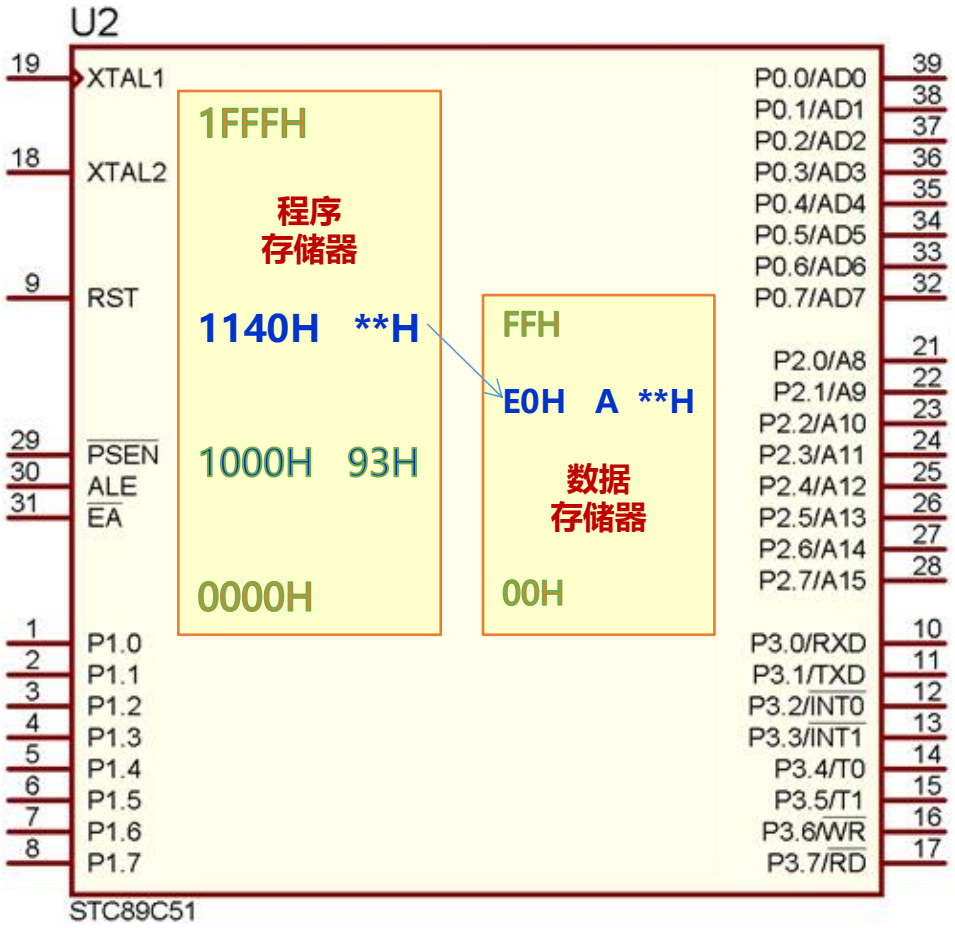
执行时，

(PC)=(DPTR+A)

=(1100H+40H)

=(1140H)→A

与指令的当前地址1000H无关。



# 由以下两例说明这两条指令的特点

例3- 10：设(A)=30H，执行地址1000H处的指令。

1000H：MOVC A，@A+PC

执行时，

(PC)=(PC+A)

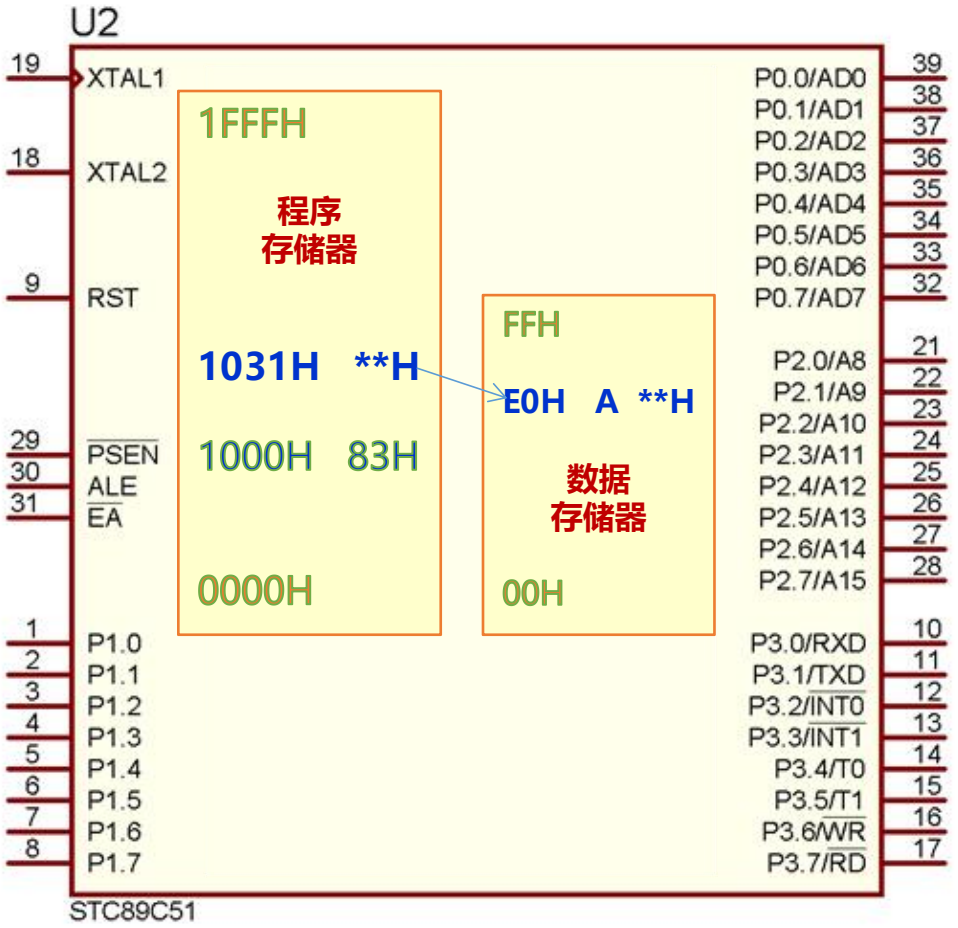
=(1001H+30H)

=(1031H)→A

PC即下一条指令的地址

=当前指令地址+该指令字节数 = 1000H+1=1001H

与当前指令地址有关



# 顺序结构——查表程序

- 查表程序

  当y =f(x)函数很复杂的情况下使用，根据自变量x，在表格寻找y。

  y的数据表格存放于程序存储器内。

  在执行查表指令时，将程序存储器中存放的数据读取后进行处理。

## 常用的查表指令：

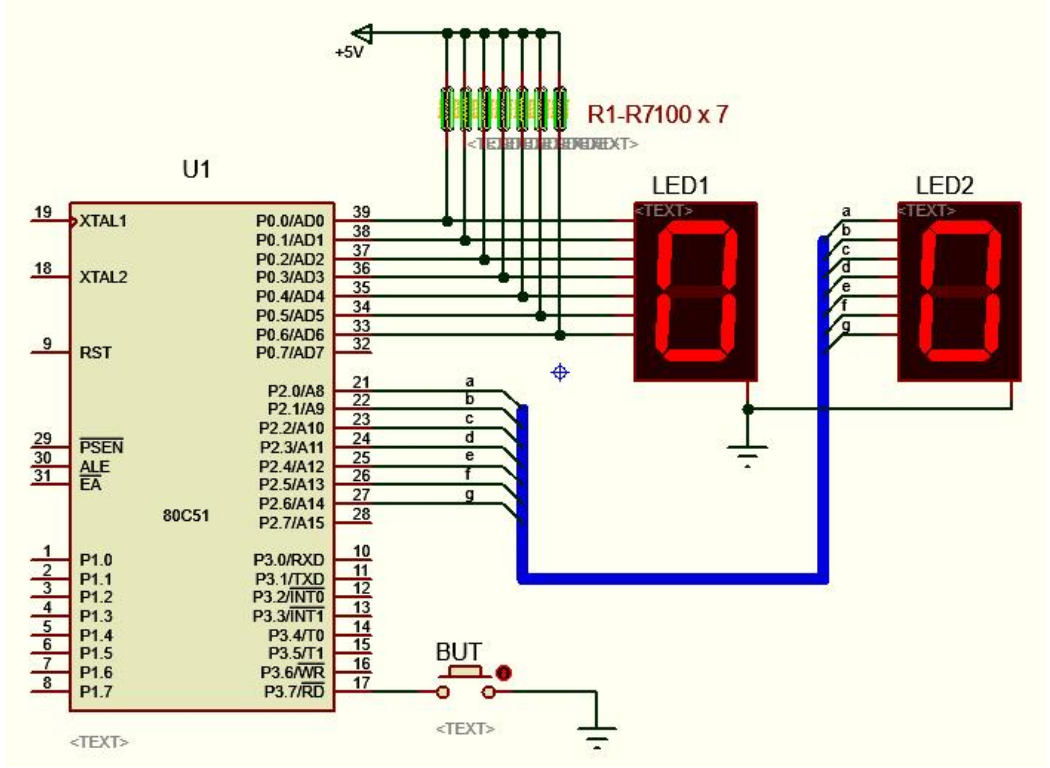
（1）MOVC A，@A+DPTR ;PC=DPTR+A，常数存放的地址只和DPTR及A的内容有关，与指令的位置无关；因此，常数的个数和位置可以在64K程序存储器中任意安排。

（2）MOVC A，@A+PC；PC=PC+A，常数存放的地址是由下一条指令的首地址和A的内容有关，即与当前指令的位置有关，且A最大为256。因此，常数只能存放在该条查表指令后面的256个字节单元之内，常数的个数受到限制。尽量少使用。

**例** 统计按键次数并直接显示在P0口所接的数码管上。

设按次数值存放在累加器A中。

```
        ORG    0000H
        LJMP   MAIN
        ORG    0100H
MAIN:   MOV    A,#03H
        MOV    DPTR,#TAB
        MOVC   A,@A+DPTR
        MOV    P0,A
        SJMP   $
TAB:    DB   3FH,06H,5BH,4FH,66H,6DH,7DH
        DB   07H,7FH,6FH,7FH,7CH,39H,5EH
        DB   79H,71H,00H  ;共阴极段码表
        END
```

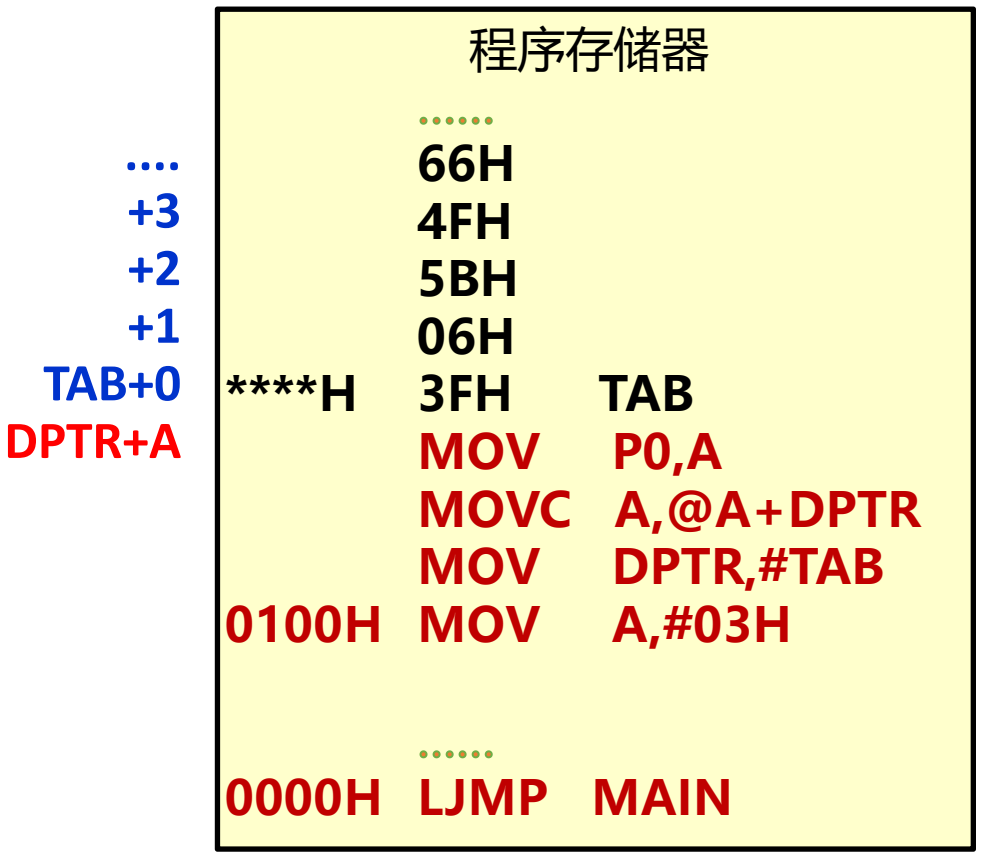


**例** 统计按键次数并直接显示在P0口所接的数码管上。

设按次数值存放在累加器A中。

```
        ORG    0000H
        LJMP   MAIN
        ORG    0100H
MAIN:   MOV    A,#03H
        MOV    DPTR,#TAB
        MOVC   A,@A+DPTR
        MOV    P0,A
        SJMP   $
TAB:    DB   3FH,06H,5BH,4FH,66H,6DH,7DH
        DB   07H,7FH,6FH,7FH,7CH,39H,5EH
        DB   79H,71H,00H  ;共阴极段码表
        END
```

使用指令“MOVC A，@A+DPTR” 查表时，
表格可以设在64KB程序存储器空间内的任何地方。



**例** 统计按键次数并直接显示在P0口所接的数码管上。

设按次数值存放在累加器A中，设统计到3次。

```
        ORG    0000H
        LJMP   MAIN
        ORG    0100H
MAIN:   MOV    A,#03H
        ADD    A，#02H
        MOVC   A，@A+PC
        MOV    P0,A
        SJMP   $
TAB:    DB   3FH,06H,5BH,4FH,66H,6DH,7DH,
        DB   07H,7FH,6FH,7FH,7CH,39H,5EH,
        DB   79H,71H,00H  ;共阴极段码表
        END
```

**ADD中A的值是多少？作用是什么？**

```
....    +3
        +2
        +1
PC+2+0  PC+2
        PC
```

程序存储器

```
        ……
        66H
        4FH
        5BH
        06H
****H   3FH     TAB
        MOV     P0,A
        MOVC    A,@A+PC
        ADD     A,#02H
0100H   MOV     A,#03H


        ……
0000H   LJMP    MAIN
```

使用“MOVC A，@A+PC”指令查表时,必计算偏移量，表格只能设在PC下面的256个单元中。

ADD指令A中为MOVC查表指令与数据表之间指令所占的字节数。其作用是为使PC指向表首。

# 4. 堆栈操作指令

堆栈特点：先进后出。

MCS-51系列堆栈结构—向上生长型。

栈底：第一个进栈数据所在单元

栈顶：最后进栈数据所在单元

**PUSH direct**；压栈

；先SP自动加1，SP←SP+1，

；再数据压栈（SP）←（direct）

**POP direct**；出栈

；先数据出栈（direct）←（SP）

；再SP自动减1，SP←SP-1

| 字节地址 | 内容 | |
|---|---|---|
| FFH<br>…<br>81H<br>80H | 0000 0111 | SP |
| 7FH<br>…<br>30H | 00H<br>…<br>00H | |
| 2FH<br>…<br>20H | 00H<br>…<br>00H | 7FH.....78H<br>……<br>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 |

特别说明：在堆栈操作中，累加器A一定要用ACC表示。

PUSH direct ；SP+1->SP, (direct)->(SP)
POP direct ；(SP)->(direct), SP-1->SP

例：执行下列指令后A、B在执行SUB子程序前后的变化.

```
MAIN: MOV SP,#30H
      MOV A,#40H
      MOV B,#41H
      ......
SUB:  PUSH ACC
      PUSH B
      ......
      MOV  A, #00H
      MOV  B, #00H
      ......
      POP  B
      POP  ACC
      RET
```

SP →
SP →
SP →

| 字节地址 | 内容 | |
|---|---|---|
| FFH | | |
| ... | | |
| F0H | 00H | B |
| ... | ... | |
| E0H | 00H | A |
| ... | ... | |
| 81H | 30H | SP |
| 80H | | |
| 7FH | 00H | |
| ... | | |
| 32H | 41H | |
| 31H | 40H | |
| 30H | 00H | |
| 2FH...20H | 00H.....00H | 7FH.....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 |

# 特殊功能寄存器——SP

**堆栈是为子程序和中断操作而设，主要用来保护断点和现场。**

**（1）保护断点。无论是子程序调用操作还是中断服务子程序调用，最终都要返回主程序。应预先把主程序的断点在堆栈中保护起来，为程序正确返回做准备。**

**（2）现场保护。执行子程序或中断服务子程序时，要用到一些寄存器单元，会破坏原有内容。要把有关寄存器单元的内容保存起来，送入堆栈，这就是所谓的“现场保护”。**

# 注意

- 在汇编主程序中需要执行：

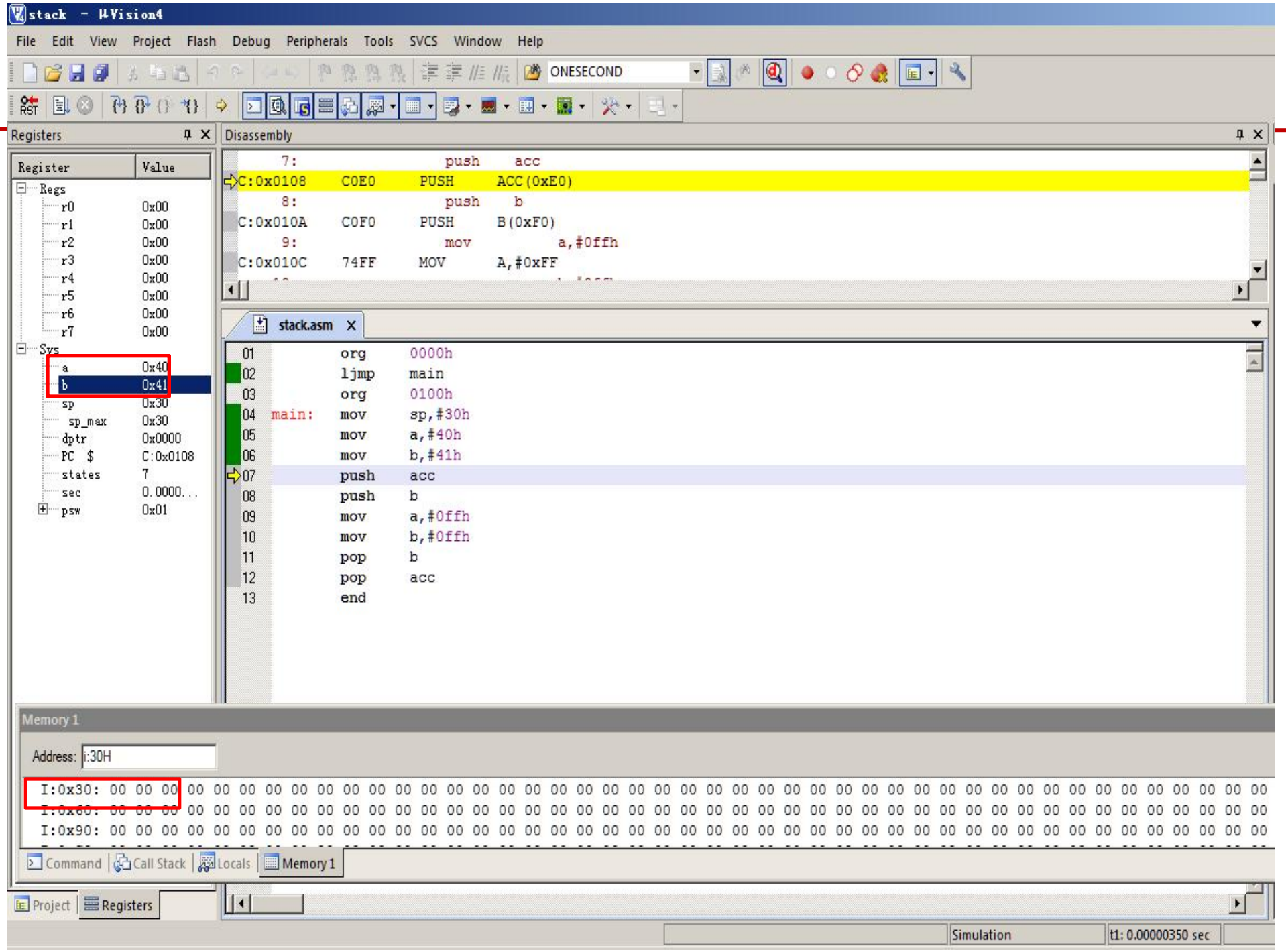
  MOV SP, #30H

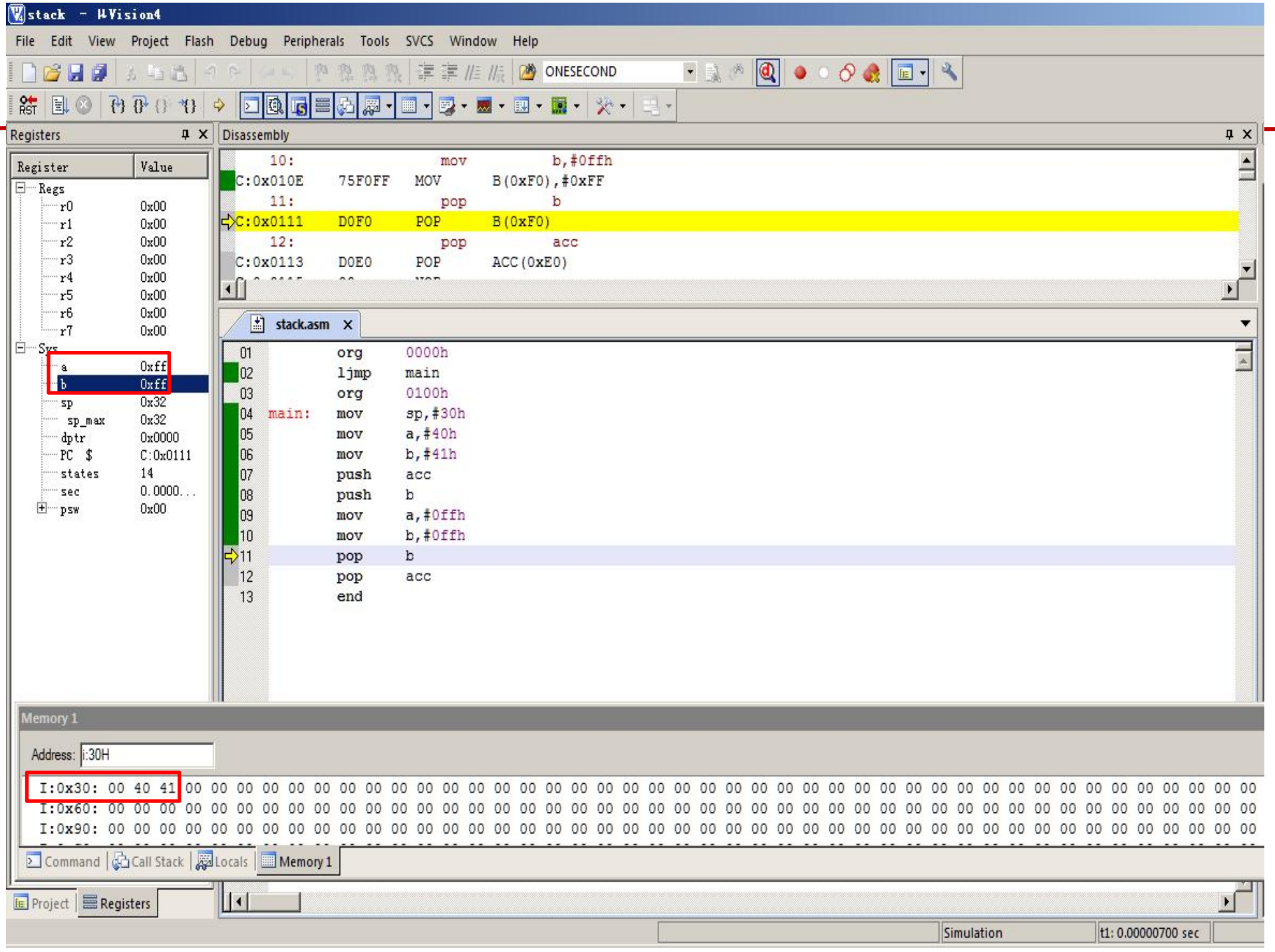
- 在Keil C51中的启动文件中执行，

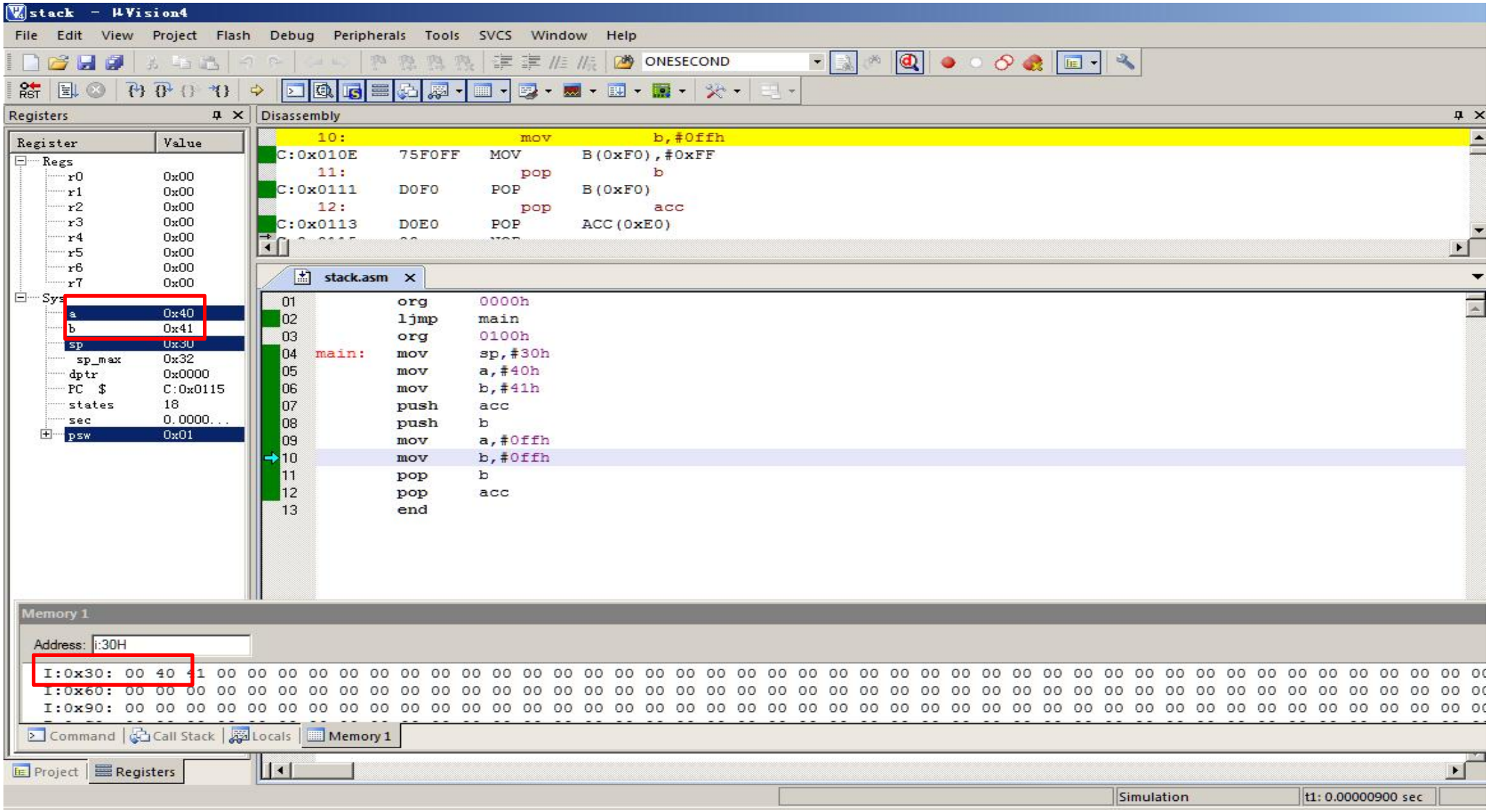
  MOV SP,#?STACK-1

?STACK在链接时由编译器自动分配的，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。




stack - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
ONESECOND
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0x40
b 0x41
sp 0x30
sp_max 0x30
dptr 0x0000
PC $ C:0x0108
states 7
sec 0.0000...
psw 0x01
Disassembly
7: push acc
C:0x0108 C0E0 PUSH ACC(0xE0)
8: push b
C:0x010A C0F0 PUSH B(0xF0)
9: mov a,#0ffh
C:0x010C 74FF MOV A,#0xFF
stack.asm
01 org 0000h
02 ljmp main
03 org 0100h
04 main: mov sp,#30h
05 mov a,#40h
06 mov b,#41h
07 push acc
08 push b
09 mov a,#0ffh
10 mov b,#0ffh
11 pop b
12 pop acc
13 end
Memory 1
Address: i:30H
I:0x30: 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
I:0x60: 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
I:0x90: 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
Command Call Stack Locals Memory 1
Project Registers
Simulation
t1: 0.00000350 sec


stack - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
ONESECOND
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
Sys
a 0xff
b 0xff
sp 0x32
sp_max 0x32
dptr 0x0000
PC $ C:0x0111
states 14
sec 0.0000...
psw 0x00
Disassembly
10: mov b,#0ffh
C:0x010E 75F0FF MOV B(0xF0),#0xFF
11: pop b
C:0x0111 D0F0 POP B(0xF0)
12: pop acc
C:0x0113 D0E0 POP ACC(0xE0)
stack.asm
01 org 0000h
02 ljmp main
03 org 0100h
04 main: mov sp,#30h
05 mov a,#40h
06 mov b,#41h
07 push acc
08 push b
09 mov a,#0ffh
10 mov b,#0ffh
11 pop b
12 pop acc
13 end
Memory 1
Address: i:30H
I:0x30: 00 40 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x60: 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 00 00 00 00 00 00 00 00 00 00 00 00
I:0x90: 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 00 00 00 00 00 00 00 00 00 00 00 00
Command Call Stack Locals Memory 1
Project Registers
Simulation
t1: 0.00000700 sec


stack - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
ONESECOND
Registers
Register Value
Regs
r0 0x00
r1 0x00
r2 0x00
r3 0x00
r4 0x00
r5 0x00
r6 0x00
r7 0x00
a 0x40
b 0x41
sp 0x30
sp_max 0x32
dptr 0x0000
PC $ C:0x0115
states 18
sec 0.0000...
psw 0x01
Disassembly
10: mov b,#0ffh
C:0x010E 75F0FF MOV B(0xF0),#0xFF
11: pop b
C:0x0111 D0F0 POP B(0xF0)
12: pop acc
C:0x0113 D0E0 POP ACC(0xE0)
stack.asm
01 org 0000h
02 ljmp main
03 org 0100h
04 main: mov sp,#30h
05 mov a,#40h
06 mov b,#41h
07 push acc
08 push b
09 mov a,#0ffh
10 mov b,#0ffh
11 pop b
12 pop acc
13 end
Memory 1
Address: i:30H
I:0x30: 00 40 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x60: 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 00 00 00 00 00 00 00 00 00 00 00 00
I:0x90: 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 00 00 00 00 00 00 00 00 00 00 00 00
Command Call Stack Locals Memory 1
Project Registers
Simulation
t1: 0.00000900 sec




```
PUSH   A          PUSH   A
PUSH   B          PUSH   B
.....             .....
POP    A          POP    B
POP    B          POP    A
```

## 5.交换指令

（1）字节交换：将累加器A和源操作数的内容交换。

XCH A，Rn

XCH A，direct

XCH A，@Ri

（2）半字节交换：将累加器A的低4位与内部RAM低4位交换。

XCHD A，@Ri

（3）累加器内交换：将累加器A内的高4位与低4位交换。

SWAP A

例: A=80H,R7=08H,(40H)=0F0H,R0=30H,(30H)=0FH.

连续执行下列指令：

XCH A,R7 ;(A)←→(R7)

XCH A,40H ;(A)←→(40H)

XCH A,@R0 ;(A)←→((R0))




结果：A=0FH,R7=80H,(40H)=08H,(30H)=F0H

例：R0=60H,(60H)=3EH,A=59H

执行 XCHD A，@R0 指令.

则A=5EH，(60H)=39H。

## 3.2.2 算术运算类指令

- 算术运算类指令主要是对8位无符号数进行算术操作。

- 这类指令会影响PSW的有关位，对这一类指令要特别注意正确地判断结果对标志位的影响。

1.加法指令

ADD A, Rn ; A←（A）+(Rn)

ADD A, direct ; A←（A）+(direct)

ADD A, @Ri ; A←（A）+((Ri))

ADD A, #data ; A←（A）+data

2. 带进位加法指令

ADDC A, Rn ; A←(A)+(Rn)+Cy

ADDC A, direct ; A←(A)+(direct)+Cy

ADDC A, @Ri ; A←(A)+((Ri))+Cy

ADDC A, #data ; A←(A)+data+Cy

这两组指令的特点：

1)一个加数总是来自累加器A;

2)运算结果总是放在累加器A中；

3)对各个标志位的影响：

i ）位7有进位，则Cy=1，否则Cy=0;

ii ）位3有进位，则Ac=1，否则Ac=0;

iii）最高两位同时有进位或同时无进位，不溢出OV=0；否则 溢出OV=1。

iv）累加器A的结果影响P标志。

3. 带进位减法指令

SUBB A,Rn ; A ←(A)-(Rn)- Cy

SUBB A,direct; A←(A)-(direct) - Cy

SUBB A,@Ri ; A←(A) -(@Ri) - Cy

SUBB A,#data; A←(A)-data - Cy

(注意:无不带进位的减法指令)

这组指令特点:

1）从累加器A中减去指定变量和进位标志；

2）结果存在累加器A中。

3）SUBB指令对标志位的影响：

i ）位7需借位则Cy=1，否则Cy=0；

ii ）位3需借位则Ac=1，否则Ac=0；

iii）最高两位同时有借位或同时无借位，不溢出OV=0；否则 溢出OV=1。

iv）累加器A的结果影响P标志。

4.增量指令

INC A ；A←(A)+1

INC Rn ；Rn←(Rn)+1

INC @Ri ；direct←(direct)+1

INC direct ；Ri←(Ri)+1

INC DPTR；DPTR←(DPTR)+1

5. 减1指令

DEC A ; A← (A)-1

DEC Rn ; Rn← (Rn) -1

DEC direct ; direct← (direct)-1

DEC @Ri ; (Ri) ← ((Ri))-1

~~DEC DPTR~~ 注意：无DEC DPTR指令

这组指令特点：不影响任何标志，除了A加1指令和A减1指令影响奇偶标志位P外。

**例：若原来为FFH，则加1后溢出为00H，但不影响标志位。**

**例：若原来为00H，则减1后溢出为FFH，但不影响标志位。**

例：

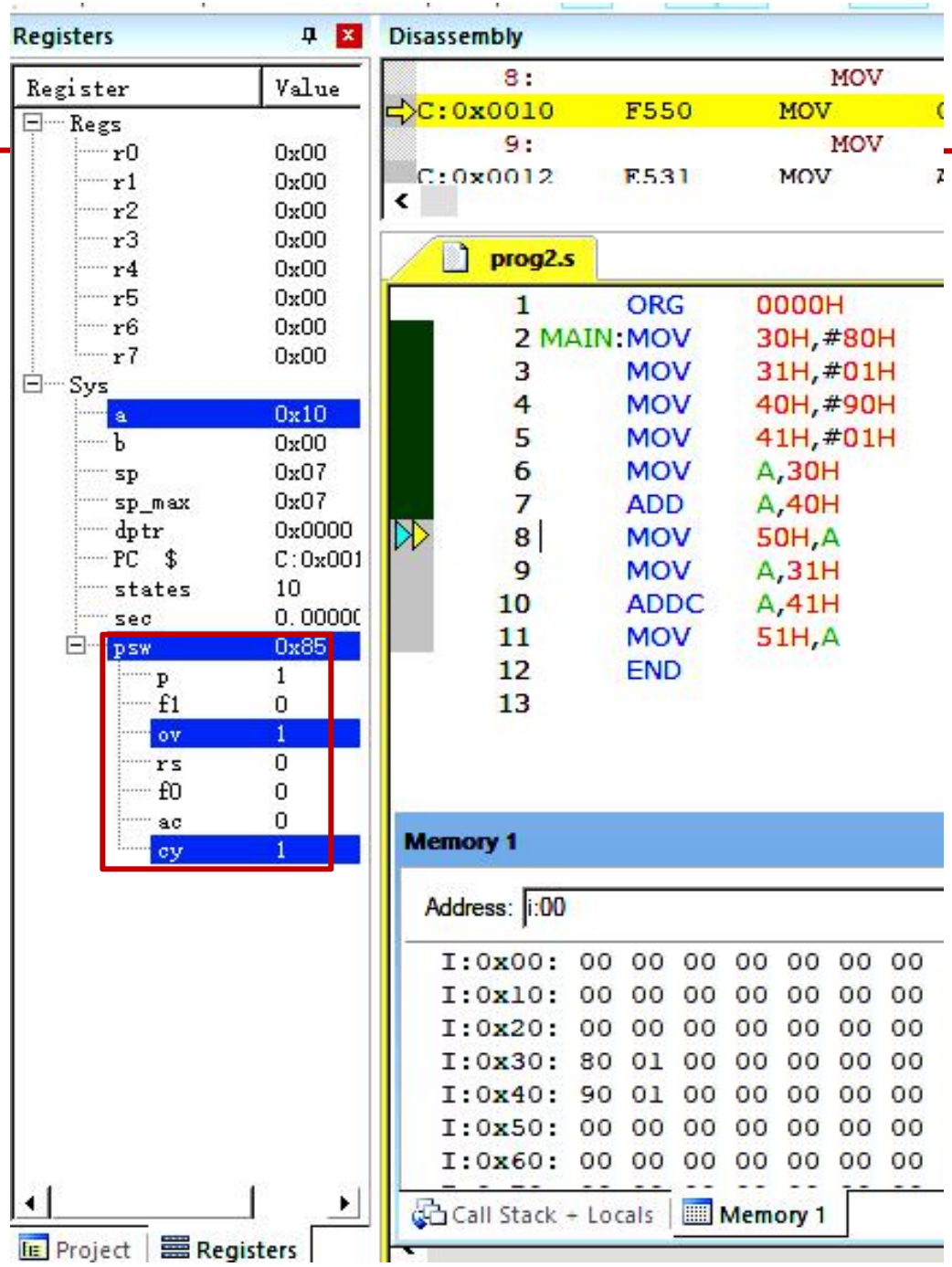
设(30H)=80H，(31H)=01H

与(40H)=90H，(41H)=01H相加，

结果存入(50H)(51H),观察PSW的值

```
MOV   A,30H
ADD   A,40H
MOV   50H,A
MOV   A,31H
ADDC   A,41H
MOV   51H,A
```

| | | |
|---|---|---|
| | (31H) | (30H) |
| + ) | (41H) | (40H) |
| | 1 | |
| | (51H) | (50H) |
| | ADDC | ADD |



例：

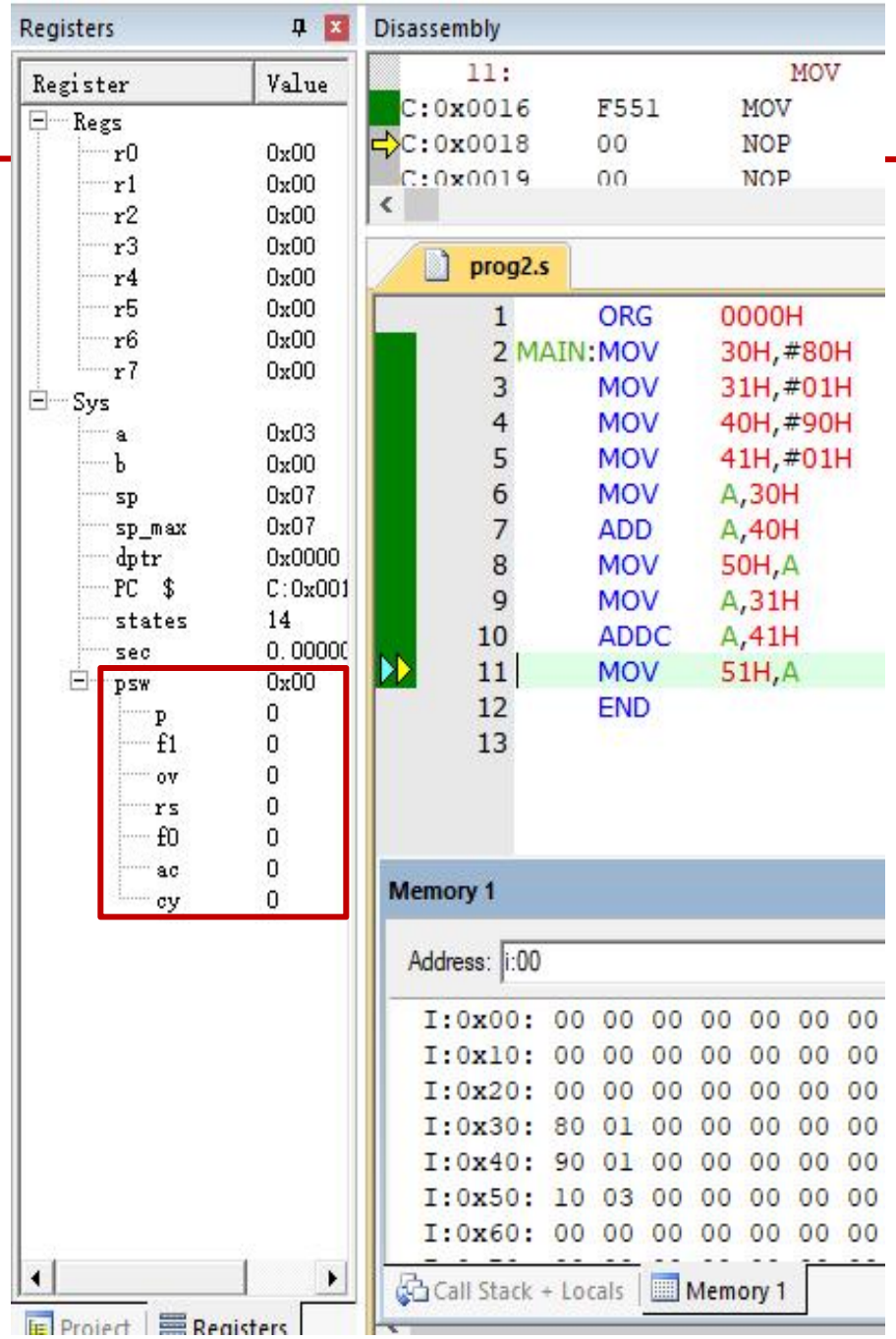
设(30H)=80H，(31H)=01H

与(40H)=90H，(41H)=01H相加，

结果存入(50H)(51H),观察PSW的值

```
MOV   A,30H
ADD   A,40H
MOV   50H,A
MOV   A,31H
ADDC  A,41H
MOV   51H,A
```

| | (31H) | (30H) |
|---|---|---|
| +) | (41H) | (40H) |
| | 1 | |
| | (51H) | (50H) |
| | ADDC | ADD |



6.乘法指令　　MUL AB　；BA←A*B

7.除法指令　　DIV　AB　；A←A/B的商，B←A/B的余数

说明:1）无符号数相乘/除；

2）运算结果会使A,B两个寄存器都获得确定的数；

对标志位的影响:

i）进位标志Cy总是清“0”

ii）累加器A的结果影响P标志。

iii）对于乘法：如果积大于0FFH即255，则OV=1，否则OV=0。

iv）对于除法：如果除数B为0，结果A,B中的内容不定，则溢出标志位OV=1。

例：（A）=01H，（B）=0FFH

执行：MULAB

结果：(A)= 0FFH，(B)=00H

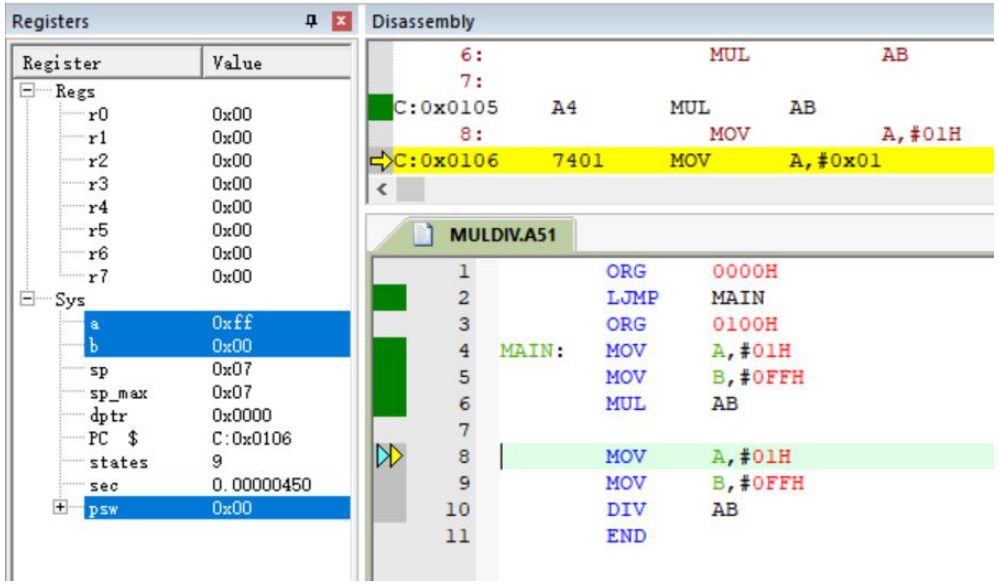
Cy=0，OV=0

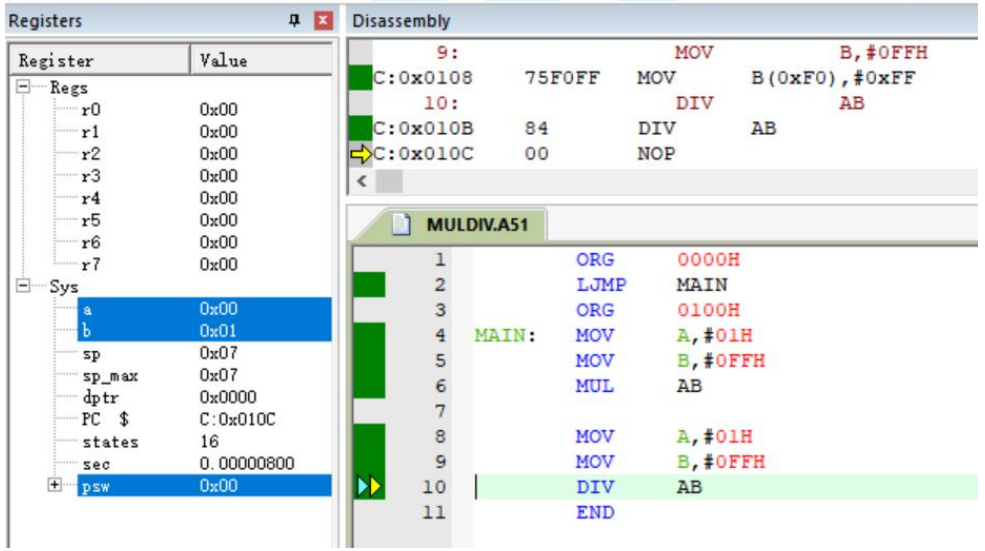
执行：DIV AB

结果：(A)=00H，(B)=01H

Cy=0，OV=0

注意：要记得运算后有为0的寄存器结果。



# 算术运算类指令小结：

- 算术运算指令都是针对8位二进制无符号数的，如要进行带符号或多字节二进制数运算，需编写具体的运算程序，通过执行程序实现。

- 算术运算的结果将影响PSW的进位(Cy)、辅助进位(Ac)、溢出(OV)这3种标志位。

- 但增1和减1指令不影响这些标志。

# 逻辑操作类指令

- 简单逻辑操作指令

（1）CLR A

功能：累加器A清“0”。不影响Cy、Ac、OV等标志。

（2）CPL A

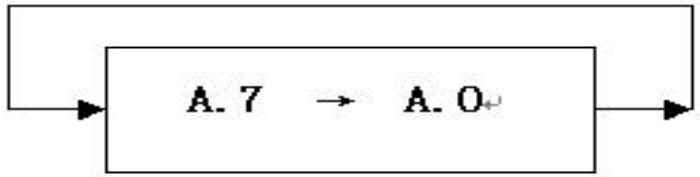
功能：累加器A按位取反，不影响Cy、Ac、OV等标志。

- 移位指令

（1）左环移：RL A ；累加器A的8位向左环移一位，不影响标志。

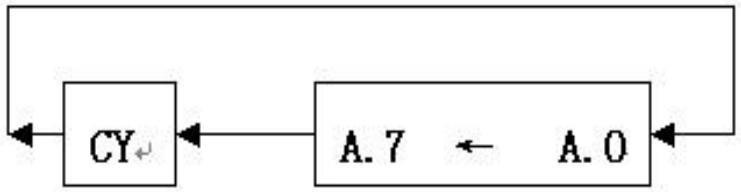
（2）右环移：RR A ；累加器A的8位向右环移一位，不影响标志。

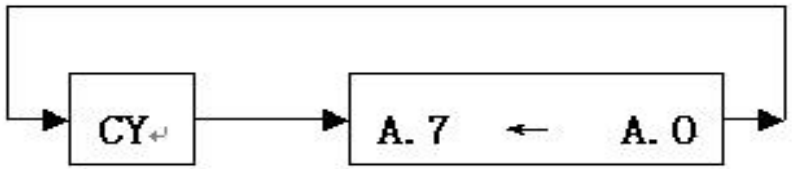



（3）带进位左环移：RLC A ；A的内容和Cy标志位一起左环移1位，不影响其它标志。

（4）带进位右环移：RRC A ；A的内容和Cy标志位一起右环移1位，不影响其它标志。



- 逻辑与指令

ANL A，Rn
ANL A，direct
ANL A，#data
ANL A，@Ri
ANL direct,A
ANL direct,#data

- 逻辑或指令

ORL A，Rn
ORL A，direct
ORL A，#data
ORL A，@Ri
ORL direct, A
ORL direct, #data

- 逻辑异或指令

XRL A，Rn
XRL A，direct
XRL A，#data
XRL A，@Ri
XRL direct,A
XRL direct,#data

功能：按位进行逻辑“与”、逻辑“或”、逻辑“异或”操作，结果存放到目的变量所在的寄存器或存储器中去。

- (A)=0A7H，(R0)=0F0H

  指令：ANL A,R0

  1010 0111

  ∧） 1111 0000

  1010 0000

  结果为：（A）＝0A0H

- (A)=0A7H，(R0)=0F0H

  指令: ORL A,R0

  1010 0111

  ∨） 1111 0000

  1111 0111

  结果：（A）=0F7H

- (A)=0A7H，(R0)=0F0H

  指令: XRL A,R0

  1010 0111

  ‘+’）1111 0000

  0101 0111

  结果：（A）=57H

**说明：**

- **“与”运算常用于对某些位清零；**
- **“或”运算常用于对某些位置1；**
- **“异或”运算常用于对某些位取反.**

# 传送类位操作指令

- 针对位地址空间的操作指令
- 只影响本身的CY（写作C），不影响其它标志位。

位累加器传送指令：一个操作数必须为进位位C。

**MOV C , bit**
**MOV bit , C**

累加器传送指令

**MOV A , direct**
**MOV direct , A**

**MOV C，40H**

**MOV A，40H**

**这两条指令怎么区别？**

# 位变量修改指令

```
CLR   C          ;清0
CLR   bit
CPL   C          ;求反
CPL   bit
SETB  C          ;置1
SETB  bit
```

简单逻辑修改字节指令

```
CLR   A
CPL   A
```

```
CLR  C
MOV  40H,C
```

```
CLR  A
MOV  40H,A
```

这两组指令怎么区别？

# 位变量逻辑与/或指令

**ANL C,bit**

**ANL C,/bit**

**ORL C,bit**

**ORL C,/bit**

逻辑与/或字节指令

| | |
|---|---|
| ANL A，direct | ORL A，direct |
| ANL direct,A | ORL direct,A |

注意：指令中的/bit，不影响直接寻址位求反前原来的状态.

# 某些指令的说明

## 关于累加器A与Acc的书写问题

- “INC A”与“INC Acc”功能相同，但机器码不同。

  INC A 的机器码是 04H

  INC Acc是由INC direct的机器码得到 05 E0H

- “PUSH Acc和POP Acc”不能写成“PUSH A和POP A”

  因为只有 PUSH direct 和 POP direct 格式

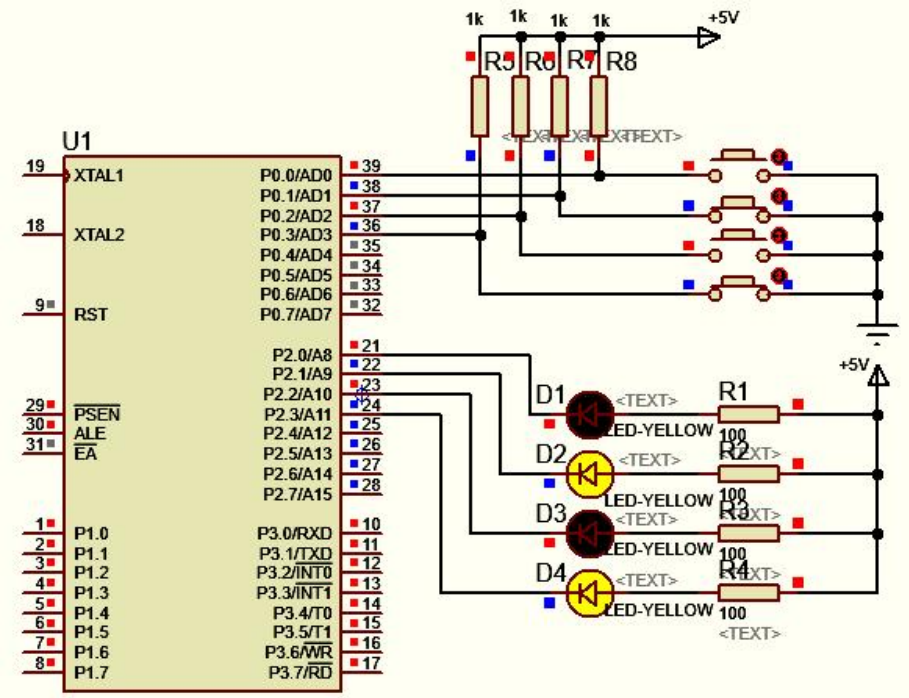
- “SETB Acc.0”不能写成“SETB A.0”

  因为在对A直接寻址和对A的某一位寻址要用Acc。

- **例1** 独立按键识别，要求开机时LED全熄，然后根据按键动作使相应灯亮，并将亮灯状态保持到按压其它键时为止。

```
        ORG   0000H
        MOV   P2,#0FFH
LOOP:   MOV   P0,#0FFH
        MOV   A,P0
        MOV   P2,A
        LJMP  LOOP
        END
```

```
#include <REG51.H>

void main() {
    char key = 0;          //定义按键变量
    P2 = 0xff;             //初始状态为灯全灭

    while(1){
        P0  = 0xff;        //读P0口之前置1
        key = P0;          //读按键状态
        P2  = key;         //显示按键状态
    }
}
```

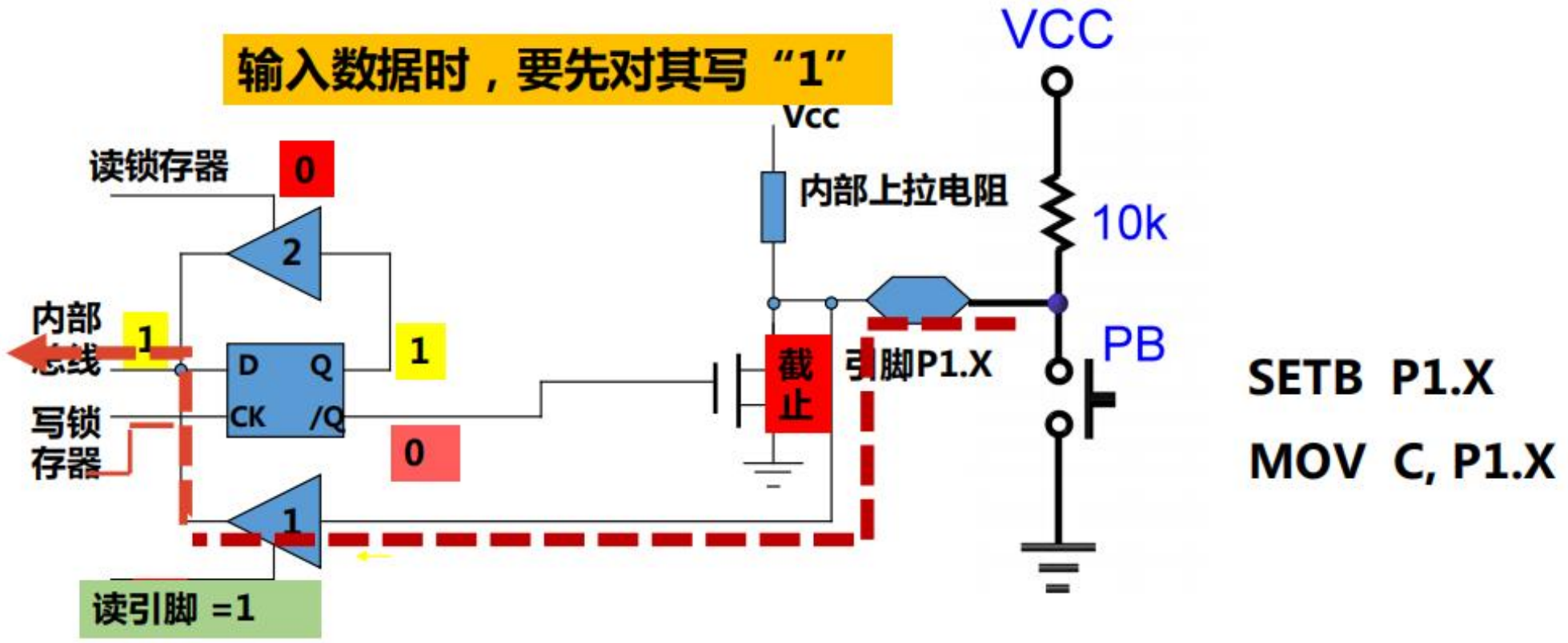


# 通用I/O口输入使用时的注意点——以P1口为例

读取Px口状态之前要先置1，才能正确读取Px口状态 。

例：先执行：**SETB P1.0** 或 **MOV P1, #0FFH**

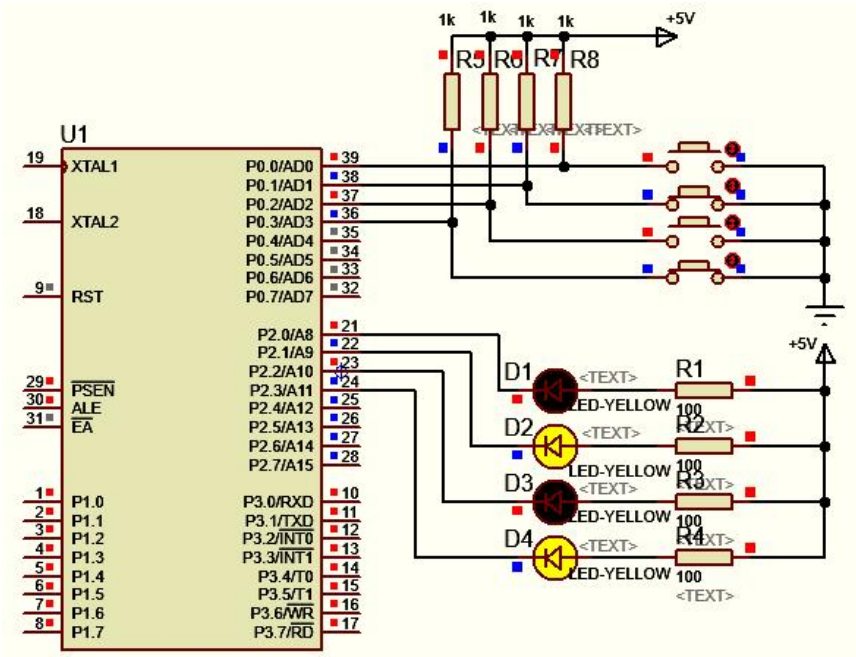
再执行 MOV C, P1.0 或 MOV A, P1



- **例1** 独立按键识别，要求开机时LED全熄，然后根据按键动作使相应灯亮，并将亮灯状态保持到按压其它键时为止。

```
        ORG     0000H
        MOV     P2,#0FFH
LOOP:   ORL     A,#0FH
        MOV     P0,A
        MOV     A,P0
        ANL     A,#0FH
        MOV     P2,A
        LJMP    LOOP
        END
```

```
#include <REG51.H>

void main() {
    char key = 0;          //定义按键变量
    P2 = 0xff;             //初始状态为灯全灭

    while(1){
        P0  = P0 | 0x0f;   //读P0口低4位之前置1
        key = P0 & 0x0f;   //读P0口按键状态高4位清零
        // if ( ~key != 0 )
        P2 = key;          //有键按下，显示按键状态
    }
}
```



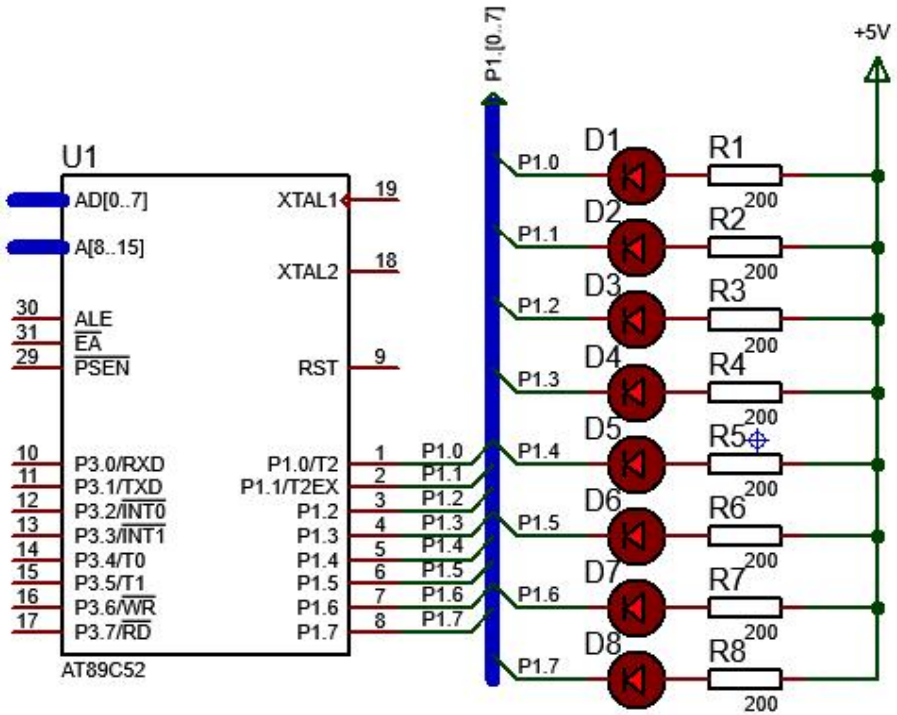
# 改为汇编代码：提示用移位指令

**例2 流水灯。要求编写程序实现：**

**（1）全灭全亮1次，再隔一交替灭亮1次；**

**（2）从下往上依次点亮一个LED，当点亮所有LED时，全灭；重复这一步。**

```
#include <REG52.H>

#define DELAY_TIME 400 //延时
#define LED_PORT1 P1  //用P1口驱动灯，低亮，高灭

void time(unsigned int ucMs);/* 延时单位：毫秒 */
void main(void)
{
    unsigned char ucTimes;


     LED_PORT1=0xff;time(DELAY_TIME);//全灭
     LED_PORT1=0;time(DELAY_TIME);    //全亮

     LED_PORT1=0x55;time(DELAY_TIME);//隔一个点亮
     LED_PORT1=0xaa;time(DELAY_TIME);//交换

     while(1)
     {
     LED_PORT1=0xff;//灭P1口灯
     time(DELAY_TIME);

      //依次点亮LED
     for(ucTimes=0;ucTimes<8;ucTimes++){//循环点亮P1口灯

         LED_PORT1 = 0xfe<<ucTimes;time(DELAY_TIME);//从上往下依次亮灯
         //LED_PORT1 = 0x7f>>ucTimes;time(DELAY_TIME); //从下往上依次亮灯
         //LED_PORT1 = LED_PORT1 - (0x80>>ucTimes);time(DELAY_TIME);//从下往上依次亮灯
         }
     }
}
```



# 第4讲-2 单片机汇编传送算逻指令和顺序结构 重点

## 重点

- 数据传送指令和位操作中的数据传送指令：
  - MOV、MOVX、MOVC的区别及使用注意点
    顺序结构举例：查表程序
  - PUSH、POP指令的使用
  - XCH、XCHD、SWAP交换指令的区别

- 算逻运算指令和位操作指令中的算逻运算指令：
  - 加减乘除指令对状态标志的影响、与或非运算的特点
    顺序结构举例：简单输入输出控制