

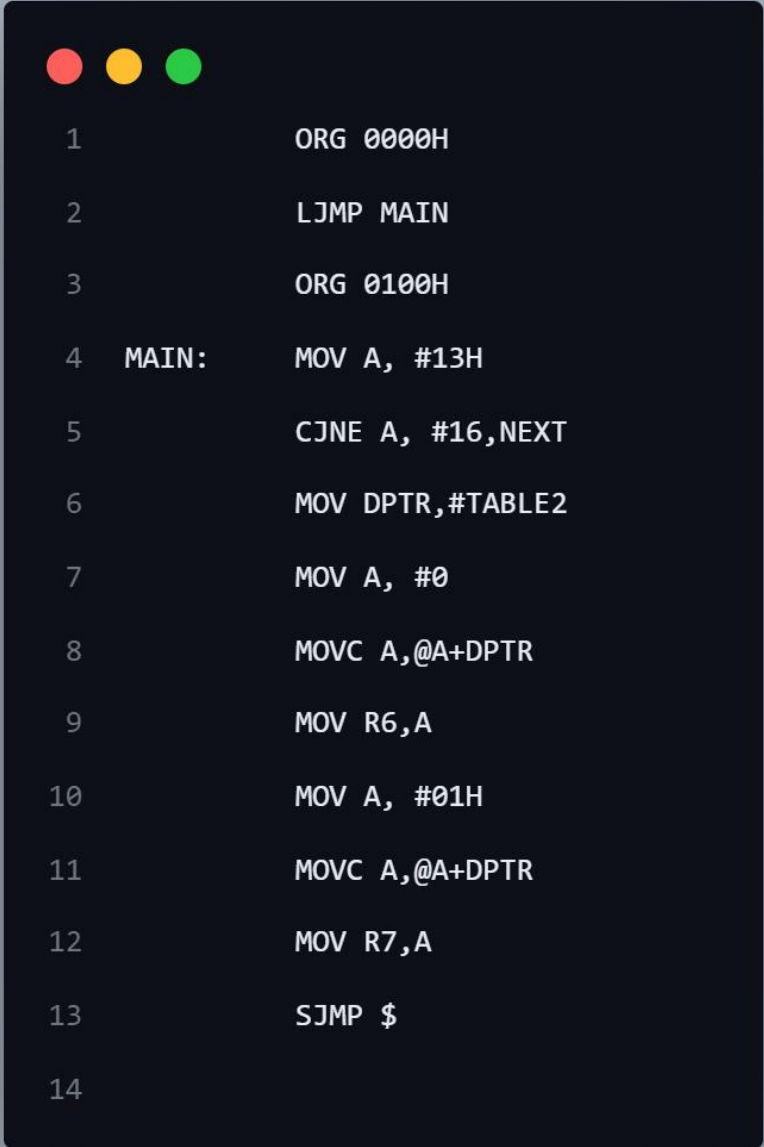
第一次作业

题目一：

题目描述：试编写采用查表法求 1-20 的平方值的程序，已知 x 的值在 1-20，存放在累加器 A 中，平方值高位存入 R6，R7

汇编语言：

代码：



```
1          ORG 0000H
2          LJMP MAIN
3          ORG 0100H
4  MAIN:    MOV A, #13H
5          CJNE A, #16,NEXT
6          MOV DPTR,#TABLE2
7          MOV A, #0
8          MOVC A,@A+DPTR
9          MOV R6,A
10         MOV A, #01H
11         MOVC A,@A+DPTR
12         MOV R7,A
13         SJMP $
14
```



```
1  NEXT:    JC  SMALL
2           MOV DPTR,#TABLE2
3           SUBB A, #10H
4           MOV B,#02H
5           MUL AB
6           MOV R1,A
7           MOVC A,@A+DPTR
8           MOV R6,A
9           MOV A,R1
10          INC A
11          MOVC A,@A+DPTR
12          MOV R7,A
13          SJMP $
14  SMALL:   MOV DPTR,#TABLE1
15          MOV R6,0
16          DEC A
17          MOVC A, @A+DPTR
18          MOV R7,A
19          SJMP $
20          ORG 0150H
21  TABLE1: DB 01,04,09,16,25,36,49,64,81,100,121,144,169,196,225
22          ORG 0160H
23  TABLE2: DW 256,289,324,361,400
24          END
```

在 150H 处，用 Table1 储存 1-15 的平方值，以一个字节储存，接着在 160H 处用 Table2 储存 16-20 的平方值，每个占用两个字节，然后程序先判断 A 中的值是否大于 16，如果大于等于就在 Table2 中进行查找（采用 MOVC 指令），指针一次移动一个字节，找到后将高字节赋值给 R6，然后指针加 1，将低字节赋值给 R7；如果小于 16 就在 Table1 中进行查找，此时高字节 R6 为 0，低字节 R7 即为查找到的值。

结果：

1. A = 13H，即计算 19 的平方，R6（高位）中值为 01，R7（低位）中值为 69，

将其转化为 10 进制数为 361，即为 19 的平方

| Register | Value |
|----------|-------|
| r0 | 0x00 |
| r1 | 0x06 |
| r2 | 0x00 |
| r3 | 0x00 |
| r4 | 0x00 |
| r5 | 0x00 |
| r6 | 0x01 |
| r7 | 0x69 |

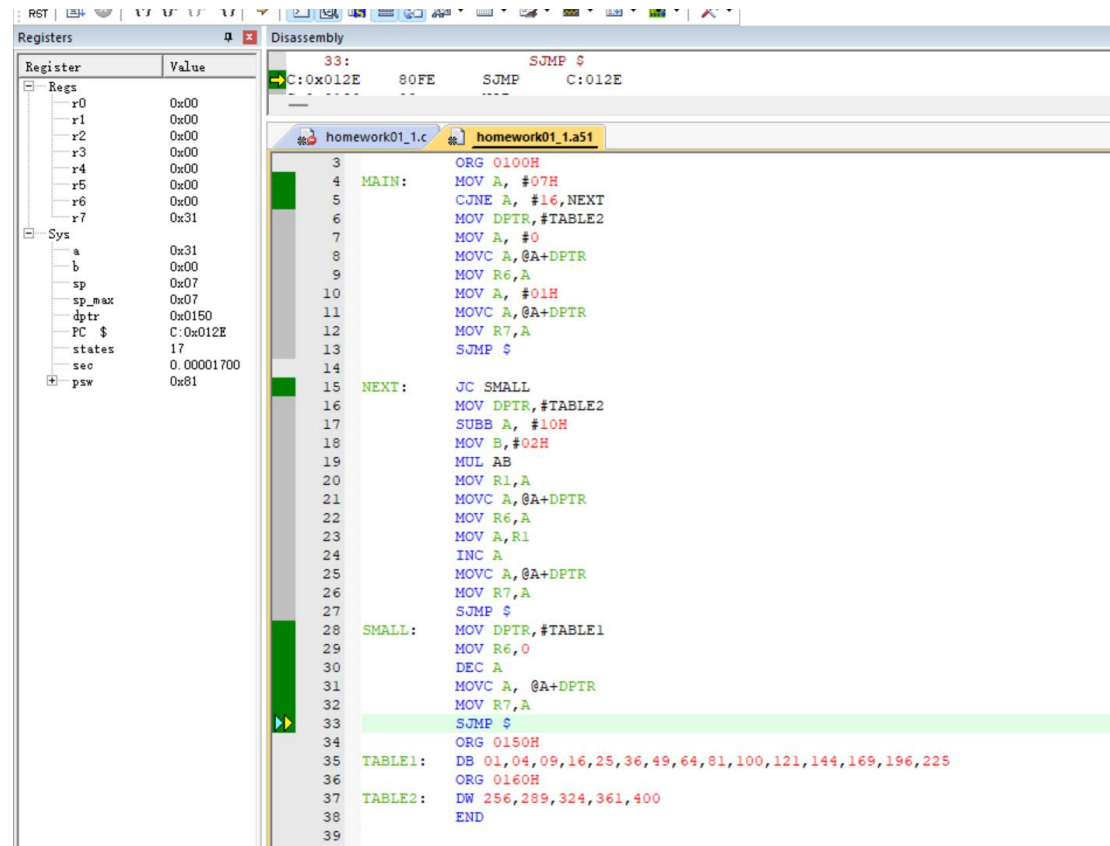
| Register | Value |
|----------|------------|
| a | 0x69 |
| b | 0x00 |
| sp | 0x07 |
| sp_max | 0x07 |
| dptr | 0x0160 |
| PC \$ | C:0x0124 |
| states | 37 |
| sec | 0.00003700 |
| psw | 0x00 |


```

27:      SJMP $
C:0x0124 80FE SJMP C:0124

homework01_1.c homework01_1.a51
3      ORG 0100H
4  MAIN:  MOV A, #13H
5          CJNE A, #16, NEXT
6          MOV DPTR, #TABLE2
7          MOV A, #0
8          MOVC A, @A+DPTR
9          MOV R6, A
10         MOV A, #01H
11         MOVC A, @A+DPTR
12         MOV R7, A
13         SJMP $
14
15  NEXT:  JC SMALL
16         MOV DPTR, #TABLE2
17         SUBB A, #10H
18         MOV B, #02H
19         MUL AB
20         MOV R1, A
21         MOVC A, @A+DPTR
22         MOV R6, A
23         MOV A, R1
24         INC A
25         MOVC A, @A+DPTR
26         MOV R7, A
27         SJMP $
28  SMALL: MOV DPTR, #TABLE1
29         MOV R6, 0
30         DEC A
31         MOVC A, @A+DPTR
32         MOV R7, A
33         SJMP $
34
35  TABLE1: DB 01,04,09,16,25,36,49,64,81,100,121,144,169,196,225
36
37  TABLE2: DW 256,289,324,361,400
38
39  END
  
```

2. $A = 07H$ ，即计算 19 的平方，R6（高位）中值为 00，R7（低位）中值为 31，将其转化为 10 进制数为 49，即为 7 的平方



The screenshot displays the Keil IDE interface. On the left, the 'Registers' window shows the state of various registers. On the right, the 'Disassembly' window shows the assembly code for 'homework01_1.a51'.

| Register | Value |
|----------|------------|
| r0 | 0x00 |
| r1 | 0x00 |
| r2 | 0x00 |
| r3 | 0x00 |
| r4 | 0x00 |
| r5 | 0x00 |
| r6 | 0x00 |
| r7 | 0x31 |
| a | 0x31 |
| b | 0x00 |
| sp | 0x07 |
| sp_max | 0x07 |
| dptr | 0x0150 |
| PC | 0x012E |
| states | 17 |
| sec | 0.00001700 |
| psw | 0x81 |

```
33: SJMP $
C:0x012E 80FE SJMP C:012E

homework01_1.c homework01_1.a51
3 ORG 0100H
4 MAIN: MOV A, #07H
5 CJNE A, #16, NEXT
6 MOV DPTR, #TABLE2
7 MOV A, #0
8 MOVC A, @A+DPTR
9 MOV R6, A
10 MOV A, #01H
11 MOVC A, @A+DPTR
12 MOV R7, A
13 SJMP $
14
15 NEXT: JC SMALL
16 MOV DPTR, #TABLE2
17 SUBB A, #10H
18 MOV B, #02H
19 MUL AB
20 MOV R1, A
21 MOVC A, @A+DPTR
22 MOV R6, A
23 MOV A, R1
24 INC A
25 MOVC A, @A+DPTR
26 MOV R7, A
27 SJMP $
28 SMALL: MOV DPTR, #TABLE1
29 MOV R6, 0
30 DEC A
31 MOVC A, @A+DPTR
32 MOV R7, A
33 SJMP $
34
35 TABLE1: DB 01,04,09,16,25,36,49,64,81,100,121,144,169,196,225
36 ORG 0160H
37 TABLE2: DW 256,289,324,361,400
38
39 END
```

C 语言：

代码：

```

1  #include <reg51.h>
2  main()
3  {
4      unsigned char data *R6 = 0x1E;
5      unsigned char data *R7 = 0x1F;
6      unsigned char Table1[15] = {1,4,9,16,25,36,49,64,81,100,121,144,169,196,225};
7      unsigned int Table2[5] = {256,289,324,361,400};
8      char x = 17;
9      *R7 = 0;
10     if(x<16)
11     {
12         *R6 = 0;
13         *R7 = Table1[x-1];
14     }
15     else
16     {
17         unsigned char *p;
18         p = Table2;
19         x -= 16;
20         p += 2*x;
21         *R6 = *p;
22         p++;
23         *R7 = *p;
24     }
25 }

```

R6, R7 储存在内部 RAM 1E 和 1F 的位置, R6, R7 分别为平方数的高 8 位和低 8 位, c51 中利用数组来存放平方表。

结果:

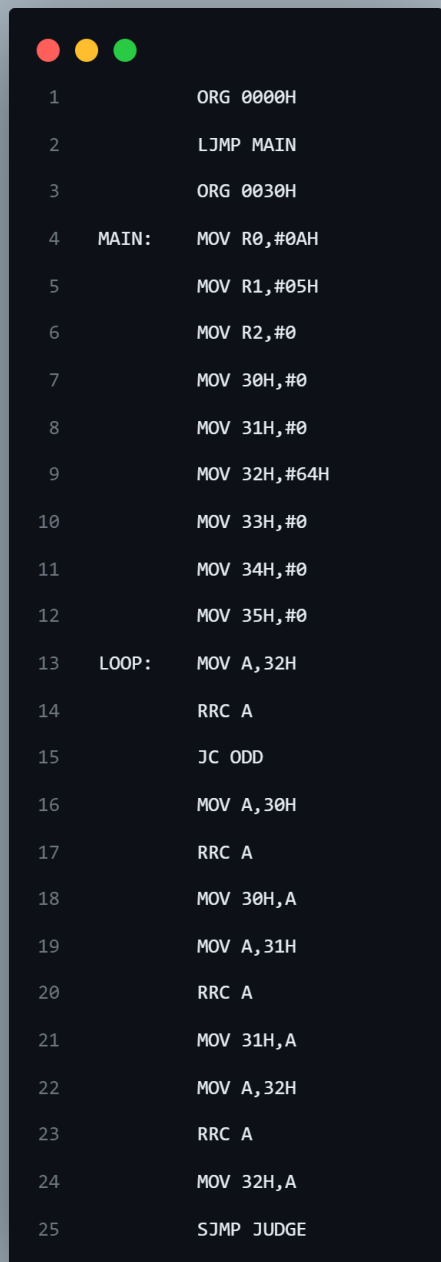
1. $x = 17$, 即计算 17 的平方, R6 即 1E 处 (高位) 中的值为 01, R7 即 1F 处 (低位) 中的值为 21, 将其转化为 10 进制数为 289, 即为 17 的平方

题目二：

题目描述：一个球从 100m 处落下又弹起，每次弹起高度为落下高度的一半，请编程计算 10 次后弹起的高度。

汇编语言：

代码：



```
1      ORG 0000H
2      LJMP MAIN
3      ORG 0030H
4  MAIN:  MOV R0, #0AH
5          MOV R1, #05H
6          MOV R2, #0
7          MOV 30H, #0
8          MOV 31H, #0
9          MOV 32H, #64H
10         MOV 33H, #0
11         MOV 34H, #0
12         MOV 35H, #0
13  LOOP:  MOV A, 32H
14         RRC A
15         JC ODD
16         MOV A, 30H
17         RRC A
18         MOV 30H, A
19         MOV A, 31H
20         RRC A
21         MOV 31H, A
22         MOV A, 32H
23         RRC A
24         MOV 32H, A
25         SJMP JUDGE
```




```
1  ODD:    CLR C
2          MOV A,32H
3          MOV B,R1
4          MUL AB
5          MOV 32H,A
6          MOV 34H,B
7          MOV A,31H
8          MOV B,R1
9          MUL AB
10         MOV 35H,B
11         ADD A,34H
12         MOV 31H,A
13         MOV A,30H
14         MOV B,R1
15         MUL AB
16         ADDC A,35H
17         MOV 30H,A
18         MOV A,33H
19         INC A
20         MOV 33H,A
21  JUDGE:  DJNZ R0,LOOP
22         SJMP $
23         end
```

由于 10 次后得到的数为浮点数，而每一个字节只能保存整型数，所以采用科学计数法来保存最终的结果 ($a \times 10^{(-b)}$)，即采用一部分内存储存 a，再用一个字节储存 b。在程序中，我用 30H, 31H, 32H 这三个字节储存 a（因为 a 较大，已超出 2 字节无符号数的最大范围，且 30H 为用户存储区的首位，00H-30H 的内存为一些寄存器的位置，不宜随便使用），33H 这一个字节储存 b。首先判断每次计算完的整数部分的奇偶性，如果是偶数，直接对三个字节进行右移（注：高字节的最低位右移完需为低字节的最高位，不能舍去，因此这里采用带 cy 的循环右移方式 RRC，高字节的最低位右移完进入 c，对低字节操作时 c 右移即进入低字节的最高位），即相当于对其除 2；如果是奇数，对其除 2 相当于对其乘 5 再除 10，除 10 相当于科学计数法中 10 的幂 b 减 1，所以对三个字节乘 5 即得到新的整数部分（代码中 odd 部分的操作）（对三个字节进行乘法时首先用最低字节与 5 相乘，低 8 位直接赋到新数的最低字节，高 8 位作为临时变量储存在 34H 中，然后中间字节与 5 相乘，低 8 位与 34H 中的临时变量相加，得到新数的中间字节，高 8 位作为临时变量储存在 35H 中，最后原数的最高字节与 5 相乘，低 8 位与 35H 中的临时变量相加，得到新数的最高字节），然后整体循环 10 次即可得到最终的结果。

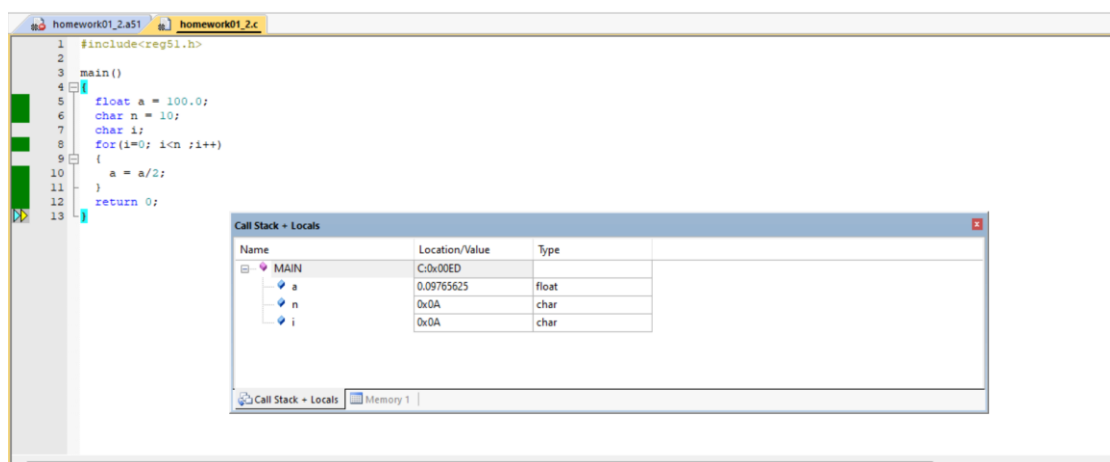
结果：


```

1  #include<reg51.h>
2
3  main()
4  {
5      float a = 100.0;
6      char n = 10;
7      char i;
8      for(i=0; i<n ;i++)
9      {
10         a = a/2;
11     }
12     return 0;
13 }

```

结果：



The screenshot shows a debugger window with the following components:

- Code Editor:** Displays the C code from the previous block, with line numbers 1 through 13. The execution is currently at line 10, `a = a/2;`.
- Call Stack + Locals:** A window showing the current state of the program.

| Name | Location/Value | Type |
|------|----------------|-------|
| MAIN | C:\00ED | |
| a | 0.09765625 | float |
| n | 0x0A | char |
| i | 0x0A | char |

a 中储存最终的结果为 0.09765625