ORG 0000H

AJMP START

ORG 0100H

START: //存放地址的初始化

MOV R0 ,#40H //存放数据的起始地址

MOV R1 ,#40H //存放数据的起始地址（更新数据）

MOV 4AH, #00H //用于是否使用上次计算保留的数

MOV 48H ,#00H //用于op操作后的判断，为1说明需要跳过按键取值

MOV 4EH,#00H //存放负号的次数

MOV DPTR, #TAB //数字表的起始地址

KEY: //主程序

LCALL SHOW //将存放在RAM里的数据展示在数码管上

LCALL KC //进行按键的检测

JNB P3.1,KEY2 //判断独立键盘1，负号按键是否被按下

JNB P3.0,KEY3 //判断独立键盘2，退格按键是否被按下

LJMP KEY //无限循环

KEY2:

LCALL D10ms //延迟子程序

JNB P3.1,Independent1 //去抖后判断是否按下

LJMP KEY //没按下返回主函数继续判断

KEY3:

LCALL D10ms //延迟子程序

JNB P3.0,Independent2 //去抖后判断是否按下

LJMP KEY //没按下返回主函数继续判断

KC: //检查键盘输入

MOV P1, #0F0H //令P1的高四位为1（行），低四位为0（列）

LCALL D10ms //延迟10ms，去抖动

MOV A, P1 //将这时P1的值传到A中

LCALL D10ms //延迟10ms，去抖动

CJNE A, #0F0H, K0 //如果P1传到A的值发生改变，则代表按键被按下，进入行的判断

LJMP RETURN //如果P1传到A的值没有发生改变，则代表没有按键被按下

K0: //第一行的判断

CJNE A, #70H, K1 //如果按键不在第一行，则跳转到第二行

MOV B, #0 //第一行的首地址的值

AJMP CK //跳转到列的判断

K1: //第二行的判断

CJNE A, #0B0H, K2 //如果按键不在第二行，则跳转到第三行

MOV B, #4 //第二行的首地址的值

AJMP CK //跳转到列的判断

K2: //第三行的判断

CJNE A, #0D0H, K3 //如果按键不在第三行，则跳转到第四行

MOV B, #8 //第三行的首地址的值

AJMP CK //跳转到列的判断

K3:

MOV B, #12 //第四行的首地址的值

CK: //列的判断

MOV P1, #0FH //令P1的高四位为0（行），低四位为1（列）

MOV A, P1 //将这时P1的值传到A中

CJNE A, #0FH, C0 //判断是否第一列的按键被按下

LJMP RETURN

C0: //第一列的判断

CJNE A, #7H, C1 //如果按键不在第一列，则跳转到第二列

MOV A, #0 //第一列的首地址的值

ADD A, B //将行和列的值相加即为按键的值

SJMP WRITE //将按键的值写入到RAM中

C1: //第二列的判断

CJNE A, #0BH, C2 //如果按键不在第二列，则跳转到第三列

MOV A, #1 //第二列的首地址的值

ADD A, B //将行和列的值相加即为按键的值

SJMP WRITE //将按键的值写入到RAM中

C2: //第三列的判断

CJNE A, #0DH, C3 //如果按键不在第三列，则跳转到第四列

MOV A, #2 //第三列的首地址的值

ADD A, B //将行和列的值相加即为按键的值

SJMP WRITE //将按键的值写入到RAM中

C3: //第四段的判断

MOV A, #3 //第四列的首地址的值

ADD A, B //将行和列的值相加即为按键的值

SJMP WRITE //将按键的值写入到RAM中

Independent1: //负号独立按键函数

Wait1:LCALL SHOW //调用显示函数

JNB P3.1,Wait1 //判断按键是否松开

MOV A,4EH //取出负号次数

INC A //负号次数加1

MOV 4EH,A //存入负号次数

MOV @R1,#10H //把负号的查表码放入操作数第一位

INC R1 //指针加1

LJMP KEY //返回主函数

Independent2: //退格独立按键函数

Wait2:LCALL SHOW //调用显示函数

JNB P3.0,Wait2 //判断按键是否松开

DEC R1 //回退指针减1

MOV @R1,#00H //把退格位数码管置零

LJMP KEY //返回主函数

WRITE: //将按键的值写入到RAM中

CJNE A,#0AH,Reset\_OP\_end //判断不等于10就不做重置

LCALL Reset\_OP1 //跳转到重置函数

MOV R7,48H //取出标志位

CJNE R7,#00H,Operation\_end //判断是否要进行按键取值

Reset\_OP\_end:

CJNE A,#0BH,Equal\_OP\_end //判断不等于11就不做等于

LCALL Equal\_OP //跳转到等于函数(可能需要改)

MOV R7,48H //取出标志位

CJNE R7,#00H,Operation\_end //判断是否要进行按键取值

Equal\_OP\_end:

MOV R6,4AH //取出标志位

CJNE A,#0CH,ADD\_OP\_end //判断不等于12就不做加法

CJNE R6,#01H,A\_C //判断是否要进行加法保留数运算

LJMP ADD\_REMAIN1 //跳转到加法保留数运算函数

A\_C:

LCALL ADD\_OP //跳转到加法函数

MOV R7,48H ////取出标志位

CJNE R7,#00H,Operation\_end //判断是否要进行按键取值

ADD\_OP\_end:

MOV R6,4AH //取出标志位

CJNE A,#0DH,SUB\_OP\_end //判断不等于13就不做加法

CJNE R6,#01H,S\_C //判断是否要进行减法保留数运算

LJMP SUB\_REMAIN1 //跳转到减法保留数运算函数

S\_C:

LCALL SUB\_OP //跳转到减法函数

MOV R7,48H //取出标志位

CJNE R7,#00H,Operation\_end //判断是否要进行按键取值

SUB\_OP\_end:

MOV R6,4AH //取出标志位

CJNE A,#0EH,MUL\_OP\_end //判断不等于14就不做乘法

CJNE R6,#01H,M\_C //判断是否要进行乘法保留数运算

LJMP MUL\_REMAIN1 //跳转到乘法保留数运算函数

M\_C:

LCALL MUL\_OP //跳转到乘法函数

MOV R7,48H //取出标志位

CJNE R7,#00H,Operation\_end //判断是否要进行按键取值

MUL\_OP\_end:

MOV R6,4AH //取出标志位

CJNE A,#0FH,DIV\_OP\_end //判断不等于15就不做除法

CJNE R6,#01H,D\_C //判断是否要进行除法保留数运算

LJMP DIV\_REMAIN1 //跳转到除法保留数运算函数

D\_C:

LCALL DIV\_OP //跳转到除法函数

MOV R7,48H //取出标志位

CJNE R7,#00H,Operation\_end //判断是否要进行按键取值

DIV\_OP\_end:

Need\_Value:

MOV @R1, A //将值写入对应的地址中

INC R1 //将地址移到下一位

Operation\_end:

CH: MOV P1, #0F0H //令P1的高四位为1（行），低四位为0（列）

MOV A, P1 //将这时P1的值传到A中

LCALL SHOW //显示值

CJNE A, #0F0H, CH //判断按键是否松开

RETURN: //返回

RET

HALF\_Reset\_OP: //清除数码管后四位 用于保留数计算

MOV 44H,#0

MOV 45H,#0

MOV 46H,#0

MOV 47H,#0

RET

Reset\_OP: //手动重置数据 清除数码管

MOV 40H,#0

MOV 41H,#0

MOV 42H,#0

MOV 43H,#0

MOV 44H,#0

MOV 45H,#0

MOV 46H,#0

MOV 47H,#0

MOV R1,#40H //指针初始化

MOV 48H,#06H //按键取值标志位标记

MOV 4AH,#00H //保留数标志位清空

RET

Reset\_OP1: //手动重置数据 与上函数差别在多了负号标志位清空 若负数运算用这个函数清空会有bug所以添加此函数

MOV 40H,#0

MOV 41H,#0

MOV 42H,#0

MOV 43H,#0

MOV 44H,#0

MOV 45H,#0

MOV 46H,#0

MOV 47H,#0

MOV R1,#40H //指针初始化

MOV 48H,#06H //按键取值标志位标记

MOV 4AH,#00H //保留数标志位清空

MOV 4EH,#00H //负号标志位清空

RET

ADD\_REMAIN1: //保留数加法

Push ACC

//先把计算结果前移位到操作数

MOV A,47H //取出计算结果第一位

MOV 43H,A //存放计算结果第一位

MOV A,46H //取出计算结果第二位

MOV 42H,A //存放计算结果第二位

MOV A,45H //取出计算结果第三位

MOV 41H,A //存放计算结果第三位

MOV A,44H //取出计算结果第四位

MOV 40H,A //存放计算结果第四位

LCALL HALF\_Reset\_OP

MOV A,40H //取出第一个操作数第一位

MOV 49H,A //存放第一个操作数第一位

MOV A,41H //取出第一个操作数第二位

MOV 50H,A //存放第一个操作数第二位

MOV A,42H //取出第一个操作数第三位

MOV 51H,A //存放第一个操作数第三位

MOV A,43H //取出第一个操作数第四位

MOV 52H,A //存放第一个操作数第四位

MOV R1,#44H //指针跳转第二个操作数

MOV 48H,#01H //1是加法

MOV 4AH,#01H //保留数标志位清空

POP ACC

LJMP Operation\_end

SUB\_REMAIN1:

Push ACC

//先把计算结果前移位到操作数

MOV A,47H //取出计算结果第一位

MOV 43H,A //存放计算结果第一位

MOV A,46H //取出计算结果第二位

MOV 42H,A //存放计算结果第二位

MOV A,45H //取出计算结果第三位

MOV 41H,A //存放计算结果第三位

MOV A,44H //取出计算结果第四位

MOV 40H,A //存放计算结果第四位

LCALL HALF\_Reset\_OP

MOV A,40H //取出第一个操作数第一位

MOV 49H,A //存放第一个操作数第一位

MOV A,41H //取出第一个操作数第二位

MOV 50H,A //存放第一个操作数第二位

MOV A,42H //取出第一个操作数第三位

MOV 51H,A //存放第一个操作数第三位

MOV A,43H //取出第一个操作数第四位

MOV 52H,A //存放第一个操作数第四位

MOV R1,#44H //指针跳转第二个操作数

MOV 48H,#02H //2是减法

MOV 4AH,#01H //保留数标志位清空

POP ACC

LJMP Operation\_end

MUL\_REMAIN1:

Push ACC

//先把计算结果前移位到操作数

MOV A,47H //取出计算结果第一位

MOV 43H,A //存放计算结果第一位

MOV A,46H //取出计算结果第二位

MOV 42H,A //存放计算结果第二位

MOV A,45H //取出计算结果第三位

MOV 41H,A //存放计算结果第三位

MOV A,44H //取出计算结果第四位

MOV 40H,A //存放计算结果第四位

LCALL HALF\_Reset\_OP

MOV A,40H //取出第一个操作数第一位

MOV 49H,A //存放第一个操作数第一位

MOV A,41H //取出第一个操作数第二位

MOV 50H,A //存放第一个操作数第二位

MOV A,42H //取出第一个操作数第三位

MOV 51H,A //存放第一个操作数第三位

MOV A,43H //取出第一个操作数第四位

MOV 52H,A //存放第一个操作数第四位

MOV R1,#44H //指针跳转第二个操作数

MOV 48H,#03H //3是乘法

MOV 4AH,#01H //保留数标志位清空

POP ACC

MOV A,4EH //取出负号标志位

CJNE A,#01H,LABEL8 //判断是不是负数

MOV 40H,#10H //显示负号

LABEL8:

LJMP Operation\_end

DIV\_REMAIN1:

Push ACC

//先把计算结果前移位到操作数

MOV A,47H //取出计算结果第一位

MOV 43H,A //存放计算结果第一位

MOV A,46H //取出计算结果第二位

MOV 42H,A //存放计算结果第二位

MOV A,45H //取出计算结果第三位

MOV 41H,A //存放计算结果第三位

MOV A,44H //取出计算结果第四位

MOV 40H,A //存放计算结果第四位

LCALL HALF\_Reset\_OP

MOV A,40H //取出第一个操作数第一位

MOV 49H,A //存放第一个操作数第一位

MOV A,41H //取出第一个操作数第二位

MOV 50H,A //存放第一个操作数第二位

MOV A,42H //取出第一个操作数第三位

MOV 51H,A //存放第一个操作数第三位

MOV A,43H //取出第一个操作数第四位

MOV 52H,A //存放第一个操作数第四位

MOV R1,#44H //指针跳转第二个操作数

MOV 48H,#04H //4是除法

MOV 4AH,#01H //保留数标志位清空

POP ACC

MOV A,4EH //取出负号标志位

CJNE A,#01H,LABEL9 //判断是不是负数

MOV 40H,#10H //显示负号

LABEL9:

LJMP Operation\_end

//十进制转十六进制函数开始

LOOP3: MOV A,R2

MOV B,#0AH ;DEC = 10

MUL AB

MOV R5,B ;JW

MOV R2,A

MOV A,R1

MOV B,#0AH ;DEC = 10

MUL AB

ADD A,R5 ;+JW

MOV R1,A ;R1R2\*10

MOV A,R6

ADDC A,R2

MOV R2,A

MOV A,#00H

ADDC A,R1

MOV R1,A

RET

Change\_10to16: MOV 4BH,#00H

MOV R1,#00H

MOV R2,#00H ;output

MOV R4,#6

MOV R5,#00H ;tmp--JW

MOV R6,#00H ;HALF BYTE TAKEN FROM SRC

MOV B,#0AH ;DEC = 10

MOV A,4BH

ANL A,#0FH

MOV R6,A ;R6 = 05H

LCALL LOOP3

MOV A,4CH

SWAP A

ANL A,#0FH

MOV R6,A ;R6 = 04H

LCALL LOOP3

MOV A,4CH

ANL A,#0FH

MOV R6,A ;R6 = 03H

LCALL LOOP3

MOV A,4DH

SWAP A

ANL A,#0FH

MOV R6,A ;R6 = 01H

LCALL LOOP3

MOV A,4DH

ANL A,#0FH

MOV R6,A ;R6 = 05H

LCALL LOOP3

RET

//十进制转十六进制函数开始

Equal\_OP: //等于按钮

PUSH ACC

MOV A,44H //取出第二个操作数第一位

CJNE A,#10H,LABEL3 //判断是不是负号

MOV A,#00H //内部赋值为0

LABEL3:

MOV 53H,A //存放第二个操作数第一位

MOV A,45H //取出第二个操作数第二位

MOV 54H,A //存放第二个操作数第二位

MOV A,46H //取出第二个操作数第三位

MOV 55H,A //存放第二个操作数第三位

MOV A,47H //取出第二个操作数第四位

MOV 56H,A //存放第二个操作数第四位

MOV R7,48H //取出符号判断标志位

OP1: CJNE R7,#01H,OP2 //选择位不为1跳转

LCALL ADD1 //做加法

OP2: CJNE R7,#02H,OP3 //选择位不为2跳转

LCALL SUB1 //做减法

OP3: CJNE R7,#03H,OP4 //选择位不为3跳转

LCALL MUL1 //做乘法

OP4: CJNE R7,#04H,OP5 //选择位不为4跳转

LCALL DIV1 //做除法

OP5: MOV 48H,#06H

POP ACC

MOV A,4EH

CJNE A,#01H,LABEL4

MOV 40H,#10H

LABEL4: MOV 4EH,#00H

MOV A,40H

CJNE A,#10H,LABEL7

MOV 4EH,#01H

LABEL7:

RET //回到显示

//加法开始

ADD1:

LCALL Reset\_OP

CLR C

MOV B, #10

MOV A, 52H

MOV R1, #56H

ADDC A, @R1

DEC R1

DIV AB

MOV 47H, B

LCALL ADD\_C\_CHANGE

MOV B, #10

MOV A, 51H

ADDC A, @R1

DEC R1

DIV AB

MOV 46H, B

LCALL ADD\_C\_CHANGE

MOV B, #10

MOV A, 50H

ADDC A, @R1

DEC R1

DIV AB

MOV 45H, B

LCALL ADD\_C\_CHANGE

MOV B, #10

MOV A, 49H

ADDC A, @R1

DEC R1

DIV AB

MOV 44H, B

LCALL ADD\_C\_CHANGE

MOV A, #0

MOV R2, #0

ADDC A, R2

MOV 43H, A

MOV 4AH,#01H

RET

//加法结束

//减法开始

SUB1: CLR C

MOV A,49H

CJNE A,53H,SUB\_PANDUAN

SUB2: CLR C

MOV A,50H

CJNE A,54H,SUB\_PANDUAN

SUB3: CLR C

MOV A,51H

CJNE A,55H,SUB\_PANDUAN

SUB4: CLR C

MOV A,52H

CJNE A,56H,SUB\_PANDUAN

SUB\_PANDUAN:

JC SUBF

SUBZ: LCALL Reset\_OP //减法结果为正数

CLR C

MOV A, 52H

MOV R1,#56H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 47H,A

MOV A, 51H

MOV R1,#55H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 46H,A

MOV A, 50H

MOV R1,#54H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 45H,A

MOV A, 49H

MOV R1,#53H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 44H,A

MOV 4AH,#01H

RET

SUBF: LCALL Reset\_OP //减法结果为负数

CLR C

MOV A, 56H

MOV R1,#52H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 47H,A

MOV A, 55H

MOV R1,#51H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 46H,A

MOV A, 54H

MOV R1,#50H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 45H,A

MOV A, 53H

MOV R1,#49H

SUBB A,@R1

DEC R1

LCALL SUB\_C\_CHANGE

MOV 44H,A

MOV 40H,#10H //显示负号

MOV 4AH,#01H

RET

//减法结束

//乘法开始

MUL1:

MOV 4FH,#00H

MOV A,53H

CJNE A,#00H,MUL\_R

MOV A,54H

CJNE A,#00H,MUL\_R

MOV A,55H

CJNE A,#00H,MUL\_R

MOV A,56H

CJNE A,#01H,MUL\_R

MOV A,52H

CJNE A,#00H,MUL\_L

LJMP MUL\_R

MUL\_L:

MOV 4FH,#01H

MUL\_R:

CLR C

MOV A,49H

MOV B,#10H

MUL AB

MOV R2,50H

ADD A,R2

MOV R2,A

MOV A,51H

MOV B,#10H

MUL AB

MOV R3,52H

ADD A,R3

MOV R3,A

MOV A,53H

MOV B,#10

MUL AB

MOV R4,54H

ADD A,R4

MOV R4,A

MOV A,55H

MOV B,#10

MUL AB

MOV R5,56H

ADD A,R5

MOV R5,A

CLR C

MOV R0,#00H

MOV R1,#00H

MOV 49H, R0 //结果

MOV 50H, R1 //结果

MOV 51H, R2 //结果

MOV 52H, R3 //结果

//

CJNE R4,#00H,MULX

CJNE R5,#01H,MULX1

MOV 51H,#00H

MOV 52H,#00H

INC R4

SJMP MUL3

MULX1:CJNE R5,#00H,MULX

INC R4

MOV R2,#00H

MOV R3,#00H

MOV 51H,R2

MOV 52H,R3

SJMP MUL3

MULX:

INC R4

DEC R5

CJNE R5,#00H,MUL4

MOV R5, #100

DEC R4

SJMP MUL3

MUL4:

CJNE R5,#0FFH, MUL3

MOV R5, #99

DEC R4

SJMP MUL3

MUL3:

PUSH B

PUSH PSW //现场保护

PUSH ACC

MUL2:

MOV A, 52H

ADDC A, R3

DA A

MOV R3, A

MOV A, 51H

ADDC A, R2

DA A

MOV R2, A

MOV A, 50H

ADDC A, R1

DA A

MOV R1 ,A

MOV A, 49H

ADDC A, R0

DA A

MOV R0 , A

DJNZ R5, MUL2

MOV R5, #100

DJNZ R4, MUL2

MOV A,R0

MOV B,#10H

DIV AB

MOV 40H,A

MOV 41H,B

MOV A,R1

MOV B,#10H

DIV AB

MOV 42H,A

MOV 43H,B

MOV A,R2

MOV B,#10H

DIV AB

MOV 44H,A

MOV 45H,B

MOV A,R3

MOV B,#10H

DIV AB

MOV 46H,A

MOV A,B

CJNE A,#00H,MODIFY

SJMP MODIFY1

MODIFY:

DEC A

MODIFY1:

MOV 47H,A

MOV A,4FH

CJNE A,#00H,LABEL5

SJMP LABEL6

LABEL5:

MOV A,47H

INC A

MOV 47H,A

LABEL6:

POP B

POP ACC

POP PSW

MOV 4AH,#01H

RET

//乘法结束

//除法开始

DIV1:

//(修改 商和余数的地址，根据判断是否与乘法的地址一样)

//（修改 被除数与除数从地址里读取数）

//十进制转十六进制开始

MOV A,49H

MOV B,#10H

MUL AB

MOV R1,#50H

ADDC A,@R1

MOV 4CH,A

MOV A,51H

MOV B,#10H

MUL AB

MOV R1,#52H

ADDC A,@R1

MOV 4DH,A

LCALL Change\_10to16

MOV 49H,R1

MOV 50H,R2

MOV A,53H

MOV B,#10H

MUL AB

MOV R1,#54H

ADDC A,@R1

MOV 4CH,A

MOV A,55H

MOV B,#10H

MUL AB

MOV R1,#56H

ADDC A,@R1

MOV 4DH,A

LCALL Change\_10to16

MOV 51H,R1

MOV 52H,R2

//十进制转十六进制结束

PUSH PSW

PUSH Acc

MOV R1, 49H //被除数高位

MOV R2, 50H //被除数低位

MOV R3, 51H //除数高位

MOV R4, 52H //除数低位

MOV R5, #00H //前

MOV R6, #00H //后

MOV R7,#10H //移位次数

MOV 54H, #00H //商低位

MOV 53H, #00H //商高位

MOV 55H, #00H //余数高位

MOV 56H, #00H //余数低位

DIV\_LOOP:

CLR C

MOV A,R2 //加载低位

RLC A

MOV R2,A //保存左移后结果

MOV A,R1 //加载高位

RLC A

MOV R1,A //保存左移后结果

MOV A,R6 //加载后

RLC A

MOV R6,A //保存左移后结果

MOV A, R5 //加载前

RLC A

MOV R5, A //保存左移后结果

CLR C

MOV A, R6

SUBB A, R4

MOV A, R5

SUBB A, R3

JC DIV\_DEAL //不够减

JNC DIV\_DONE //够减

DIV\_DEAL: //不够减

//（修改商的地址）

CLR C

MOV A,54H //商低位

RLC A //左移商低位

MOV 54H,A //储存商低位

MOV A,53H //商高位

RLC A //左移商高位

MOV 53H,A //储存商高位

DJNZ R7,DIV\_LOOP//继续移位

SJMP DIV\_DIVOVER//除法结束

DIV\_DONE: //够减

//（修改商的地址）

MOV A, R6

SUBB A, R4

MOV R6, A //存放余数低位

MOV A, R5

SUBB A, R3

MOV R5, A //存放余数高位

MOV A,54H //商低位

RLC A //左移商低位

INC A //结果+1

MOV 54H,A //储存商低位

MOV A,53H //商高位

RLC A //左移商高位

MOV 53H,A //储存商高位

DJNZ R7,DIV\_LOOP//继续移位

SJMP DIV\_DIVOVER//除法结束

DIV\_DIVOVER:

//（修改余数的地址）

MOV 55H,R5 //保存余数

MOV 56H,R6 //保存余数

POP Acc //现场恢复

POP PSW

//十六进制转十进制开始

MOV TH0,53H

MOV TL0,54H

LCALL CHANGE\_16to10

MOV 47H,R0

MOV 46H,R1

MOV 45H,R2

MOV 44H,R3

MOV TH0,55H

MOV TL0,56H

LCALL CHANGE\_16to10

MOV 43H,R0

MOV 42H,R1

MOV 41H,R2

MOV 40H,R3

//十六进制转十进制结束

//除法结束

MOV 4AH,#01H

RET

//十六进制转十进制函数开始

CHANGE\_16to10:

MOV R0, TL0

MOV R1, TH0

CLR A

MOV R2, A

MOV R3, A

MOV R4, A

MOV R5, #16 ;共转换十六位数

LOOP:

CLR C

MOV A, R0

RLC A

MOV R0, A

MOV A, R1

RLC A

MOV R1, A

MOV A, R4 ;送到BCD码的低端

ADDC A, R4 ;带进位加。自身相加，相当于左移一位

DA A ;十进制调整，变成BCD码

MOV R4, A

MOV A, R3

ADDC A, R3

DA A

MOV R3, A

MOV A, R2

ADDC A, R2

MOV R2, A

DJNZ R5, LOOP ;共转换十六位数

;-------至此，已经把TH1 TL1中的数字，转换成BCD码，送到了R2 R3 R4。

;-------下面，分别存入R4 R3 R2 R1 R0。

MOV A, R4

MOV B, #16

DIV AB

MOV R1, A

MOV R0, B

MOV A, R2

MOV R4, A

MOV A, R3

MOV B, #16

DIV AB

MOV R3, A

MOV R2, B

RET

//十六进制转十进制函数结束

ADD\_C\_CHANGE:

CLR C

JZ ADD\_CHANGE

SETB C

ADD\_CHANGE:

RET

SUB\_C\_CHANGE:

JNC SUB\_CHANGE

ADD A,#0AH

SUB\_CHANGE:

RET

ADD\_OP: //加法按钮

Push ACC

//MOV 48H,#0 //地址48H存放op选择，0是加法

MOV A,40H //取出第一个操作数

MOV 49H,A //存放第一个操作数

MOV A,41H

MOV 50H,A

MOV A,42H

MOV 51H,A

MOV A,43H

MOV 52H,A

MOV R1,#44H //指针跳转第二个操作数

POP ACC

MOV 48H,#01H //1是加法

RET

SUB\_OP: //减法按钮

Push ACC

//MOV 48H,#0 //地址48H存放op选择，0是加法

MOV A,40H //取出第一个操作数

MOV 49H,A //存放第一个操作数

MOV A,41H

MOV 50H,A

MOV A,42H

MOV 51H,A

MOV A,43H

MOV 52H,A

MOV R1,#44H //指针跳转第二个操作数

POP ACC

MOV 48H,#02H //2是减法

RET

MUL\_OP: //乘法按钮

Push ACC

//MOV 48H,#0 //地址48H存放op选择，0是加法

MOV A,40H //取出第一个操作数

CJNE A,#10H,LABEL1

MOV A,#00H

LABEL1:

MOV 49H,A //存放第一个操作数

MOV A,41H

MOV 50H,A

MOV A,42H

MOV 51H,A

MOV A,43H

MOV 52H,A

MOV R1,#44H //指针跳转第二个操作数

POP ACC

MOV 48H,#03H //3是乘法

RET

DIV\_OP: //乘法按钮

Push ACC

//MOV 48H,#0 //地址48H存放op选择，0是加法

MOV A,40H //取出第一个操作数

CJNE A,#10H,LABEL2

MOV A,#00H

LABEL2:

MOV 49H,A //存放第一个操作数

MOV A,41H

MOV 50H,A

MOV A,42H

MOV 51H,A

MOV A,43H

MOV 52H,A

MOV R1,#44H //指针跳转第二个操作数

POP ACC

MOV 48H,#04H //4是乘法

RET

SHOW: //显示值

PUSH ACC //现场保护

MOV R0, #40H //值的初始地址

MOV A, @R0 //将值传入到A中

MOVC A, @A+DPTR //找出对应的值

MOV P2, #28 //位选

MOV P0, A //段选

LCALL D10ms

INC R0 //移动到下一位

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #24

MOV P0, A

LCALL D10ms

INC R0

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #20

MOV P0, A

LCALL D10ms

INC R0

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #16

MOV P0, A

LCALL D10ms

INC R0

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #12

MOV P0, A

LCALL D10ms

INC R0

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #8

MOV P0, A

LCALL D10ms

INC R0

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #4

MOV P0, A

LCALL D10ms

INC R0

MOV A, @R0

MOVC A, @A+DPTR

MOV P2, #0

MOV P0, A

LCALL D10ms

POP ACC

RET

D10ms: //延时10ms

MOV R7, #25

D1:

MOV R6, #20

DJNZ R6, $

DJNZ R7, D1

RET

TAB:

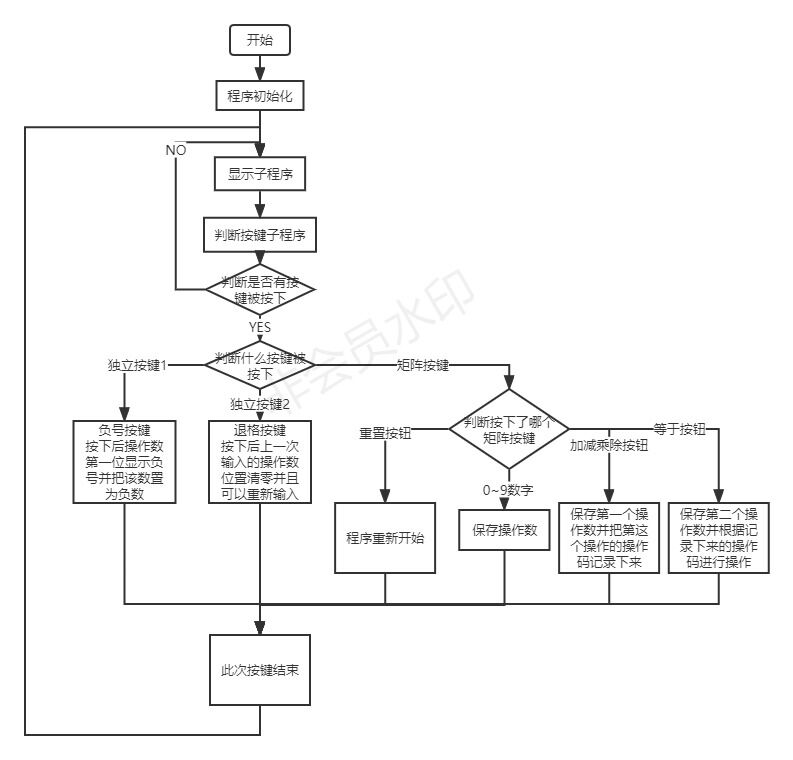
DB 3FH,06H,5BH,4FH,66H,6DH,7DH

DB 07H,7FH,6FH,77H,7CH,39H,5EH,79H,71H,40H

END

**三 设计思路**

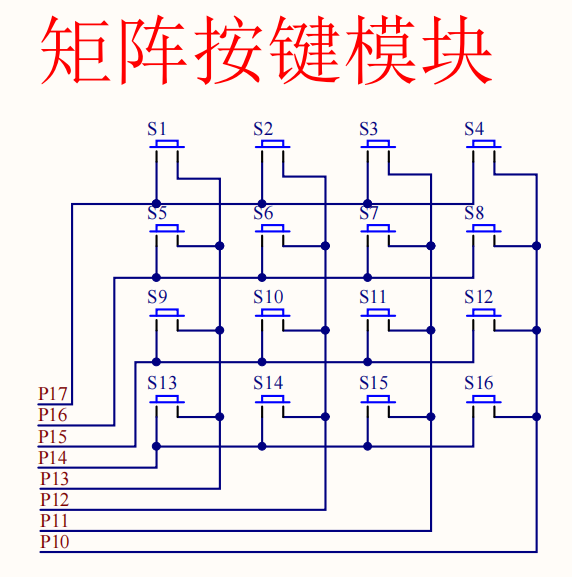
3.1 总体框架



3.2 模块划分

**①矩阵键盘模块**

构成:该模块由16个轻触按钮构成，其中同一行的按钮的一端都相连，同一列的按钮的一端也相连。

主要用途:根据用户使用的按键情况完成相应的操作，即输入操作数和运算符。

此外特别说明下矩阵键盘布局。此次矩阵键盘共有16个键，其中10个为数字键入键，另外6为功能键，下面对其各自用图进行说明。

计算器按键分布：

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | CL | = |
| + | - | \* | / |

1.0~9：数字键。按下该键后，数字输入到计算器

2.CL：全部清零键。按下该键后，计算器将清零一切信息。

3."="：等于号。按下该键后，单片机将会执行运算并显示结果。

4.”+”:加法键。按下该键后，单片机将会保存第一个操作数，并在记录下加法操作。

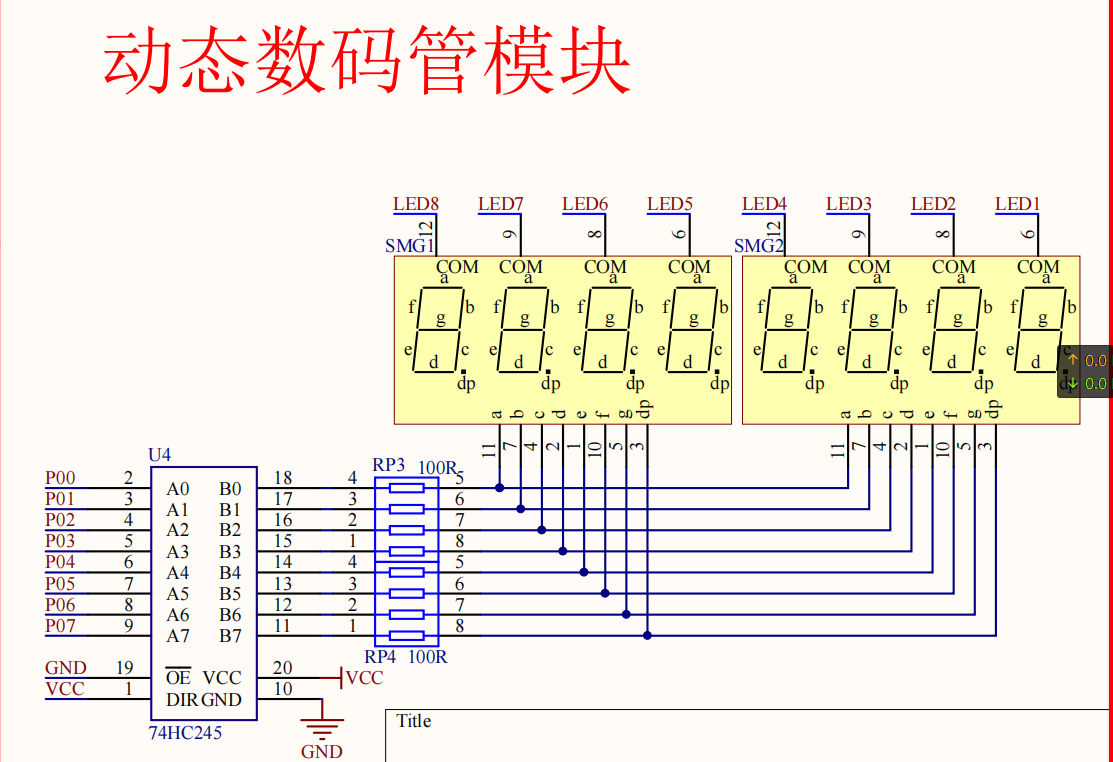
4.”-”:减法键。按下该键后，单片机将会保存第一个操作数，并在记录下减法操作。

4.”\*”:乘法键。按下该键后，单片机将会保存第一个操作数，并在记录下乘法操作。

4.”/”:除法键。按下该键后，单片机将会保存第一个操作数，并在记录下除法操作。

**②数码管显示模块**

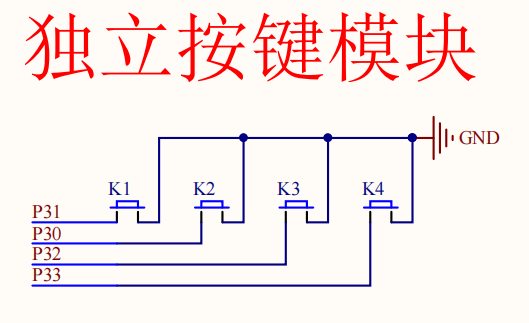
构成:该模块由八个的七段共阴极数码管构成。



主要用途:是用于显示矩阵键盘所按下的键值，以及最后计算的结果。

**③独立键盘退格模块**

构成:该模块由一个独立键盘构成。

主要用途:是用于回退用户上次按下的数字键并且可以重新输入。

**④独立键盘负号模块**

构成:该模块由一个独立键盘构成。

主要用途:是用于输入负数。

3.3 程序清单

|  |  |  |  |
| --- | --- | --- | --- |
| 1.SHOW | 数码管显示 | 2.D10ms | 软件延迟 |
| 3.MUL\_OP | 乘法按键 | 4.CHANGE\_16to10 | 十六进制转十进制 |
| 5.DIV\_OP | 除法按键 | 6.Equal\_OP | 等于按键 |
| 7.SUB\_OP | 减法按键 | 8.Change\_10to16 | 十进制转十六进制 |
| 9.ADD\_OP | 加法按键 | 10.Reset\_OP | 重置函数 |
| 11.KC | 按键判断函数 |  |  |