**信息科学与工程学院**

**2020－2021学年第二学期**

实 验 报 告

课程名称： 微处理器原理与应用

实验名称： 子程序汇编 实验 学习和提高

专 业 班 级 通信工程 二班

学 生 学 号 201922121209

学 生 姓 名 陈泽宇

实 验 时 间 2021年3月27日

实验报告

【实验目的】

1. 子程序汇编实验学习和提高

【实验要求】

1. 理解汇编语言中的ASSUME 伪指令和标准的汇编程序
2. 复习Debug的使用，提高调试的能力
3. 编写、阅读若干汇编程序，提高编程能力

【实验具体内容】

1. 复习一下Debug -P和-G
2. 将键盘上输入的十六进制数转换成十进制数，并在屏幕上显示。

（编写程序，详细注释并画程序流程图）

1. 较为复杂的汇编实例学习：判断该年是否为闰年

(通过注释重点学习并理解程序，画出程序的流程图，在理解的基础上，如果觉得程序的写法需要修改也可以自行修改)

1. 汇编实例学习与改进：两位数加法

【第一个实验：十六进制转十进制】

1. 实验流程图

主流程：



子流程1：输入判定



子流程2：存储与更新



子流程3：数据处理与输出



（2）实验源代码（粘贴源代码）

; 程序输入无效字符不读入，例如1jc=1c，只输入非法字符则输出0

; 支持4位转化

; 借用了2.2.1&2.2.2的写法进行字母判断

; upd1：修复了程序没有考虑中间0的问题

; upd2：勉强支持部分4位Hex的进制转换，但还是有大量的bug,逻辑不清晰（不上传）

; upd3：重构程序的逻辑，去掉了无用的的位数判断使得程序更为清晰

; upd4（Final）：细节优化，比如FFFF不显示的问题；完善程序注释

DATA *SEGMENT*

    STRING DB 13,10,"Please input a Hex number(Up to 4 bits):",'$'

DATA *ENDS*

STACKS *SEGMENT*

    DW 200 DUP(0)

STACKS *ENDS*

CODES *SEGMENT*

    ASSUME CS:CODES, DS:DATA, SS:STACKS

START:

WELCOME: ;输入提示信息

    MOV AX, DATA

    MOV BX, 0

    MOV DS, AX

    LEA DX, STRING

    MOV AH, 9H

    INT 21H

MAIN\_INPUT:

    MOV DX,0

    MOV AH,1

    INT 21H                            ;系统等待输入一个字符，键入一个字符之后会自动转为ASCII值存入AL中

    MOV DL,AL                          ;向DL中写入AL,DL作为每次存储时新的一位Hex

    CMP AL,0DH                         ;如果输入字符为回车则跳到标识符Init处执行，进行寄存器初始化

    JE Init

    CMP AL,39H

    JBE NUMBER                         ;如果=<9则跳到标识符NUMBER处执行

    CMP AL,41H                         ;（>9成立）如果>=A则跳到WORD\_处执行

    JAE WORD\_                          ;如上解析

    JMP MAIN\_INPUT                     ;（<A成立）继续输入字符

STORING\_:

    MOV CL, 4                          ;向CL中写入4，作为二进制下的逻辑右移位数，实现了BX十六进制角度上整体左移一位，起到了取高位的作用

    SHL BX, CL                         ;逻辑左移指令，实现了BX十六进制角度上整体左移一位

    ADD BX, DX                         ;BX更新存储的十六进制数

    JMP MAIN\_INPUT

NUMBER:                                ;字符0-9

    CMP AL, 30H                        ;判断是否>=0，匹配成功则进一步执行，否则必然是除回车外的其他字符，进行返回字符重新输入

    JAE NUM\_PROCING

    JMP MAIN\_INPUT

NUM\_PROCING:

    SUB DL, 30H

    JMP STORING\_

WORD\_:

    CMP AL, 46H                         ;大于A的情况下与F进行比较

    JBE WORD\_PROCING\_1                  ;<=F的情况

    CMP AL, 61H                         ;(通过比较，大于Z的情况下)与a进行比较

    JB MAIN\_INPUT                       ;小于a的情况：其他字符，跳转重新输出

    ;大于等于a的情况

    CMP AL, 66H                         ;与f进行比较

    JBE WORD\_PROCING\_2                  ;<=f成立则跳转

    JMP MAIN\_INPUT                      ;不成立(>f)则说明是其他字符，重新输入

WORD\_PROCING\_1:

    SUB DL, 37H;大写字母转十六进制字

    JMP STORING\_

WORD\_PROCING\_2:

    SUB DL, 57H;小写字母转十六进制字

    JMP STORING\_

Init:

; 寄存器初始化，便于后续处理，AX作为原始数据寄存器，BX作为压栈次数寄存器，CX和DX全部置零，为除法做准备

    MOV CX, 0

    MOV AX, BX

    MOV BX, DX

    MOV DX, 0

    MOV BX, 0

; 以上实现了输入1-4bit Hex数后将数据存入AX寄存器中，并初始化寄存器，BX用于记录十进制位数便于后续输出

; 下面进行连续/10入栈运算处理进行分类讨论，难点在于对DIV的理解

    CMP AX, 0FFH

    JBE SIMPLE\_PROCESS

    CMP AX, 0FFFFH

    JA ENDING

GENERAL:;一般的处理流程，针对除数为16位的情况，也是转化到simple\_process中

    MOV CX, 0AH

    DIV CX

    PUSH DX;余数入栈

    ADD BX, 1;压栈次数记录，便于后续输出存在栈中的所有数字组成一个完整的十进制数

    MOV DX, 0;这是考虑到后续循环而采用的措施，DX必须置零，否则会出现错误的结果

    CMP AX, 0FFH;这里也要进行二次/多次判断是否除数还是16位，因为两种情况的处理逻辑是不同的

    JAE GENERAL

SIMPLE\_PROCESS:;简单8位处理情况

; 每次DL取AH中存放的余数后需要将AH置零

    MOV CX, 0

    MOV CL, 0AH

    DIV CL

    MOV DL, AH

    MOV AH, 0

    PUSH DX

    ADD BX, 1;压栈次数记录，便于后续输出存在栈中的所有数字组成一个完整的十进制数

    CMP AL, 0

    JNE SIMPLE\_PROCESS

; 以上完成了转化为十进制数并入栈的工作

; 出栈输出操作，与前面的BX位数相联系进行输出即可，比较简单

Decimal\_Disp:

    POP DX

    ADD DL, 30H;这里是为了正常输出数字，转换成ASCII的Hex形式

    MOV AH, 02H

    INT 21H

    SUB BX,1;压栈次数减一，类似起到循环控制的作用

    CMP BX,0;BX=0则表明输出完成，否则继续输出

    JNE Decimal\_Disp

ENDING:

    MOV AH, 4CH

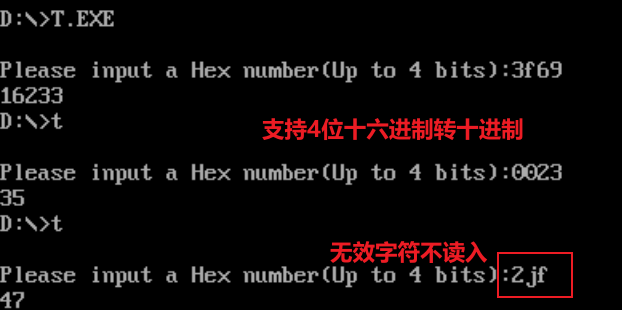
    INT 21H

CODES *ENDS*

END START

（3）实验代码、过程、相应结果（截图）并对实验进行说明和分析：

**代码运行结果如下所示**



**程序分析过程如下：（以3f69为例）**

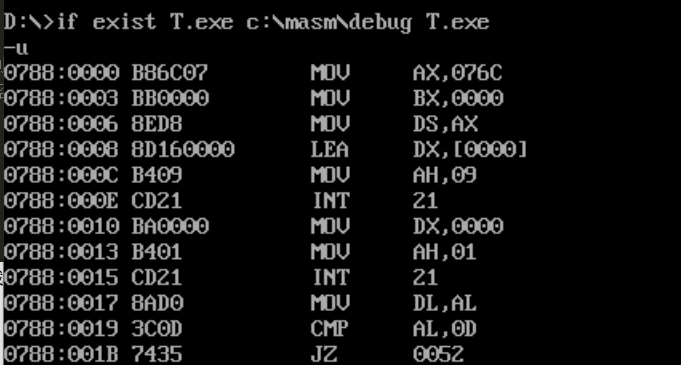
如流程图所示，整个程序可以大致分为以下几个阶段

1. 循环输入直到输入中止字符，输入结束后同时完成如下过程
   1. 无效字符不读入
   2. 数字、大小写字母统一转化为对应十六进制数存入寄存器内，
2. 寄存器初始化
3. 循环除10压栈，余数存入栈中，并记录十进制的位数便于输出十进制内容
4. 出栈输出十进制数

如下所示，在文本编辑器编辑完成后Debug装载程序



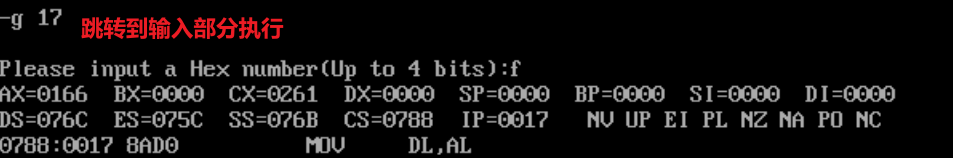
装载后如下图所示，利用-u命令查看对应内容，可见命令已被写入对应的内存中



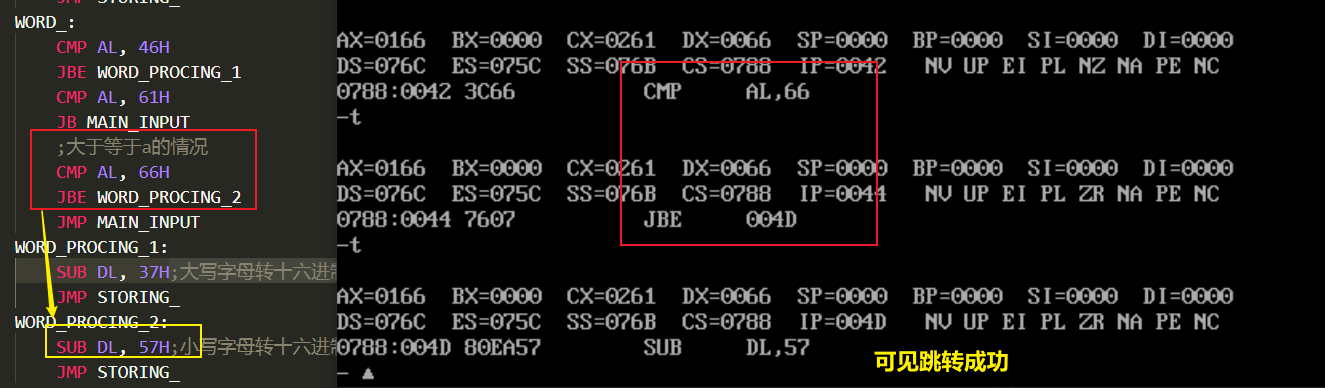
由于子流程中的某些过程分析已经在实验2.2中完成，因此后续会利用-g命令进行跳转，仅对程序关键部分的调试分析以便于说明问题。

【输入判断、变换、存储过程分析】

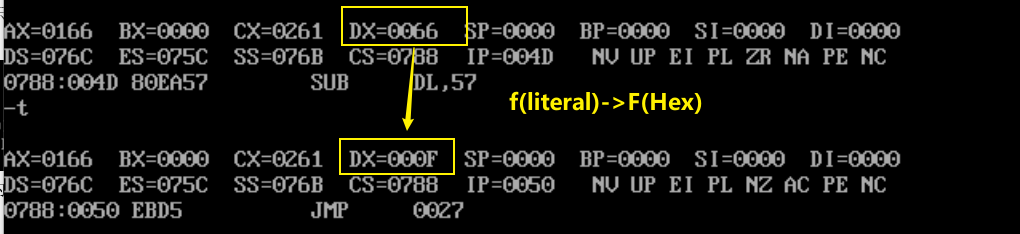
与实验2.2中的思路相仿，这里简单叙述，以输入f为例，如下图所示，输入f后AL=66H，为f的ASCII Hex形式



下面进行判断流程，判断为合法后跳转到存储部分，这里不再赘述，内容与2.2判断逻辑完全相同，如下图所示，判断合法，并且转到了字母转化的部分



根据代码内容，小写字母十六进制-57H会转换成为等价十六进制存储，如下所示



【存储】

转化完成后，可见跳转到了存储流程



【第二个实验：闰年程序分析画流程图】

主流程图



子流程1：数据处理



子流程2：闰年判断



【第三个实验：汇编实例学习与改进：两位数加法】

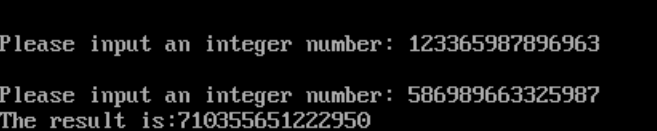
【编程逻辑】

1. 建立缓冲区，将输入的数字存入缓冲区内部，随后再存入预设好的数据段内
2. 利用内存单元遍历操作将其倒序存放，最后一个内存单元放最低位，前面的首零内存单元留空，作为预进位单元
3. 二次存入数据后，两个数据相加，覆盖ADD2
4. 进行遍历做十进制基础下的位数修正
5. 遍历处理成ASCII可输出形式
6. 遍历进行输出

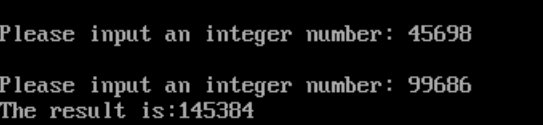
这样就可以实现小于15位的任意数加法运算

【程序运行示例】

15位+15位



小于15位但高位有进位



【程序流程图】



【实验代码】

; 程序版本：3.0

; 编程逻辑：缓冲区、内存

; 实现的功能：n位+m位相同位数的10进制加法运算（n<=m且m,n<15）

; 修复了最高位进位的bug

; 另一个bug，n>m时计算失效，没有建立好的输入异常机制（Exception）

DATA *SEGMENT*

    INFON DB 0AH, 0AH, 'Please input an integer number: $',0AH

          DB 12 DUP(?)

    ADD1 DB 16 DUP(?)

    ADD2 DB 16 DUP(?)

    ; 存放输入端十五位数字，同时ADD2存放后续求和的数字

    ADD\_BUFFER DB 16

        DB ?

        DB 16 DUP(?)

    OUTPUTINFO DB 0AH,'The result is:$',0AH

    ; 缓冲区作为加数输入端

DATA *ENDS*

STACKS *SEGMENT*

    DB 200 DUP(0)

STACKS *ENDS*

CODES *SEGMENT*

    ASSUME CS:CODES, DS:DATA, SS:STACKS

START:

    MOV AX, DATA

    MOV DS, AX

    LEA SI, ADD1

    ADD SI, 15

    ; 指向最后一位，便于倒序处理

    ; 这里倒序处理的目的是将没有存入数据的内存单元放在前面

    ; 例如默认存入为 09 02 03 04 00 00 00 00，则倒序后是 00 00 00 00 09 02 03 04，这样是为了处理高位进位时的情况

    CALL NUMBER\_PROCESS

    LEA SI, ADD2

    ADD SI, 15

    CALL NUMBER\_PROCESS

    CALL ADDING

    ; 进行数据处理，遍历每一位，对大于十的进行进位

    CALL BIT\_PROCESS

    CALL NUM\_DISP

    MOV AH, 4CH

    INT 21H

NUMBER\_PROCESS *PROC* NEAR

        LEA DX, INFON

        MOV AH, 9H

        INT 21H

        LEA DX, ADD\_BUFFER

        MOV AH, 0AH

        INT 21H

        MOV CL, [ADD\_BUFFER + 1]

        MOV CH, 0

        ; CX记录缓冲区实际数字的数量，为后续循环做准备

        LEA DI, ADD\_BUFFER + 2

        ADD DI, CX

        DEC DI

    NUMBER\_CONVERT:

        SUB *BYTE* PTR [DI], 30H

        MOV BL, [DI]

        ; BL寄存器起到中间变量的作用

        DEC DI

        ; DI自减1，整个循环过程类似于for循环（i--）

        MOV *BYTE* PTR [SI], BL

        DEC SI

    LOOP NUMBER\_CONVERT

        RET

NUMBER\_PROCESS *ENDP*

ADDING *PROC* NEAR

    LEA SI, ADD1

    ADD SI, 15

    LEA DI, ADD2

    ADD DI, 15

     ; 指向最后一位，便于倒序处理

    MOV CL, [ADD\_BUFFER+1]

    MOV CH, 0

    MOV BX, 0

    ADDING\_LOOP:

        MOV *BYTE* PTR BL,[SI]

        DEC SI

        ADD [DI], BL

        DEC DI

    LOOP ADDING\_LOOP

    RET

ADDING *ENDP*

BIT\_PROCESS *PROC* NEAR

    LEA DI, ADD2

    MOV CL, [ADD\_BUFFER+1]

    MOV CH, 0

    ADD DI, 15

    BIT\_LOOP:

        MOV *BYTE* PTR AL, [DI]

        CMP AL, 0AH

        JB LOOP\_ENDING

        ;大于十的情况

        SUB AL, 0AH

        MOV *BYTE* PTR [DI], AL;位调整

        DEC DI

        ADD [DI],1;前一位加一

        JMP FORCE\_END;因为上边已经减了一次做进位处理，这里是为了防止DI被多减1次

    LOOP\_ENDING:

        DEC DI

    FORCE\_END:

    LOOP BIT\_LOOP

    RET

BIT\_PROCESS *ENDP*

NUM\_DISP *PROC* NEAR

    LEA DX, OUTPUTINFO

    MOV AH, 9H

    INT 21H

    LEA DI, ADD2

    MOV SI, 0;作为一个“标志性”寄存器存在，当碰到第一个非零数时更改状态

    MOV CX, 16

    NUM\_DISP\_LOOP:

        MOV *BYTE* PTR DL, [DI]

        CMP SI, 1

        JE GENERAL\_PROCESS

        CMP DL, 0

        JE  LOOP\_END

        MOV SI, 1

    GENERAL\_PROCESS:

        ADD DL, 30H

        MOV AH, 2H

        INT 21H

    LOOP\_END:

        INC DI

    LOOP NUM\_DISP\_LOOP

    RET

NUM\_DISP *ENDP*

CODES *ENDS*

END START

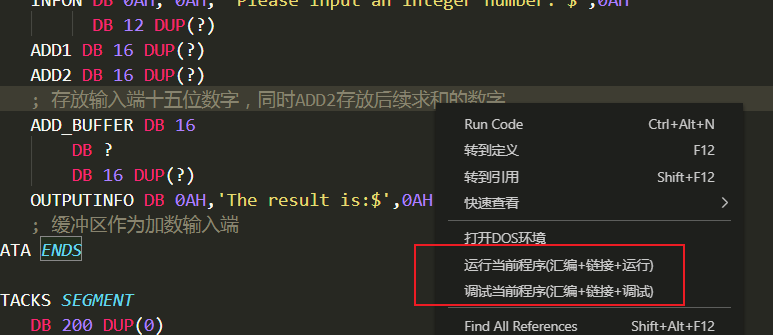
【实验过程分析】

要完成多位可输入型的十进制加法计算，首先要明确无论是缓冲区内输入后存储还是寄存器内存储，都是以16位ASCII的形式完成的。

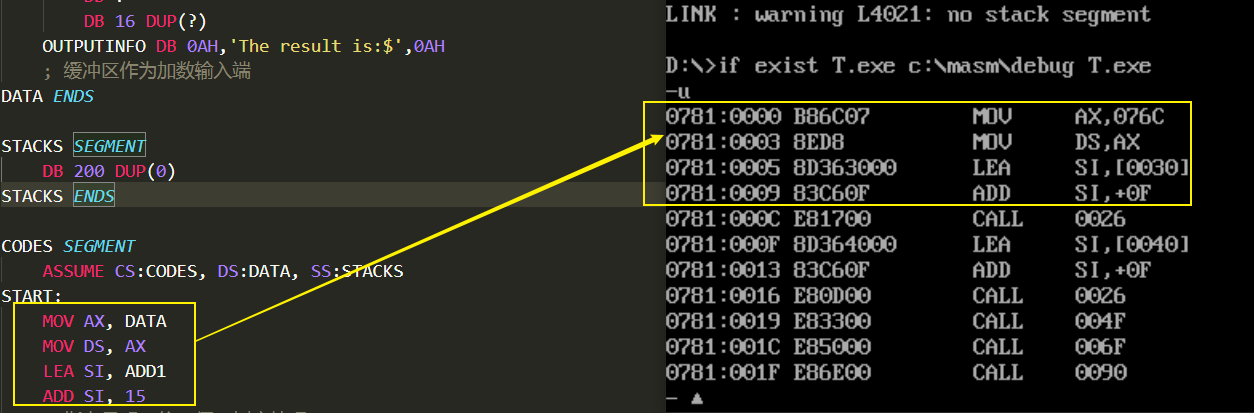
在本次实验中，由于涉及的量都是数字0-9，没有涉及字母，所以不需要进行3.1中16进制转10进制的判断，这里只需要统一处理数字，也就是遍历-30H变成十进制数意义上的0-9存储在预先定义好的数据段就可以了。

根据这个思路，先分析数据存储和相加的部分

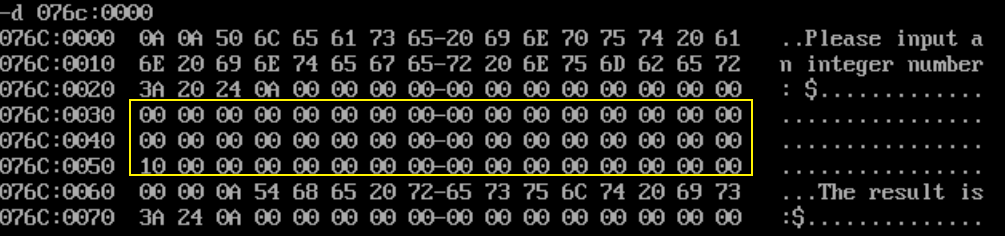
在文本编辑器VScode中完成程序编辑，并利用插件功能进行调试



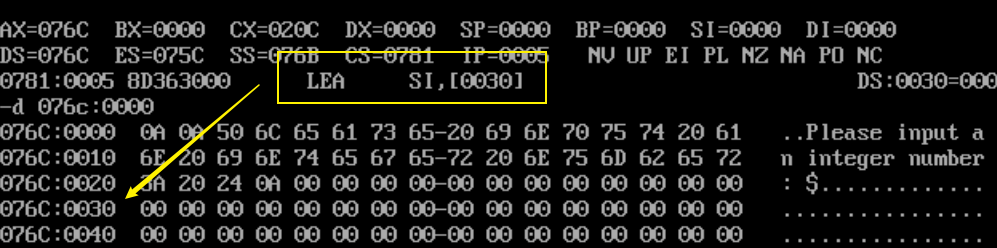
导入结果如下所示，利用-u命令进行查看，可知导入成功



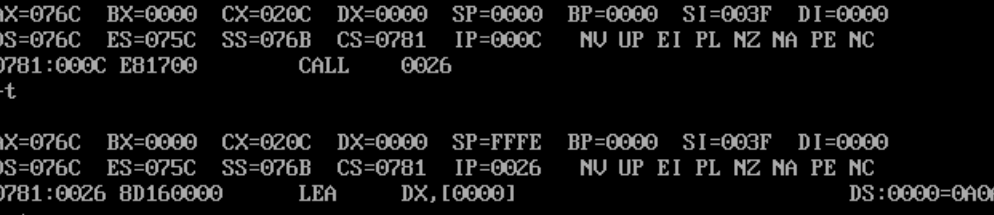
首先查看DATA的内存布局，如下图所示，这样是为了将后续要处理的数字进行对齐



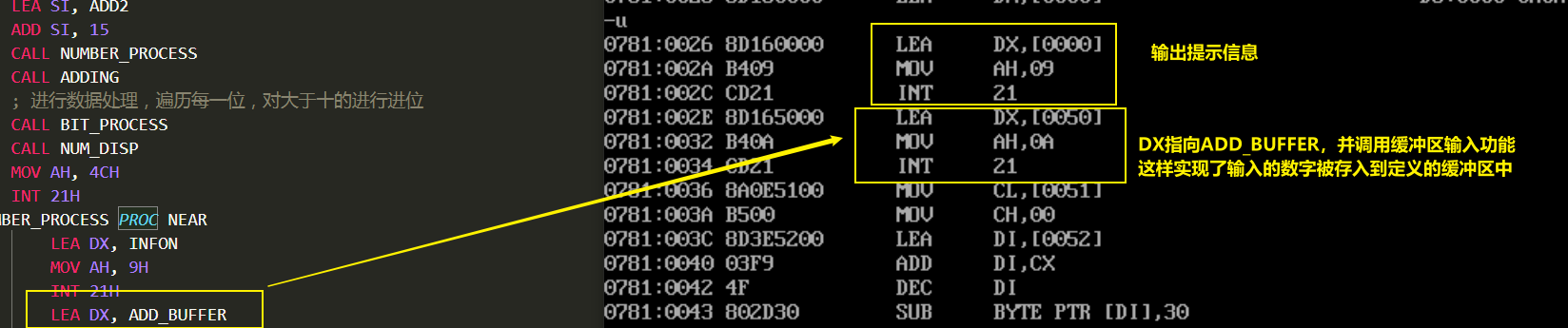
预留的存储单元区域分别为ADD1、ADD2、ADD\_BUFFER



程序首先指向ADD1，并+15，表示指向最末端，便于后续倒序处理



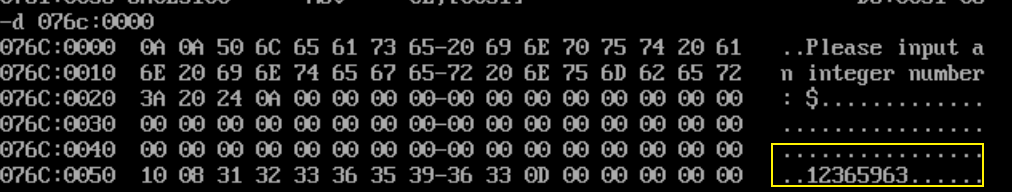
随后程序调用第一个子程序，数字的输入



程序开头的代码段注释如上图所示，利用-g 36将其执行完毕，如下图所示



利用-d命令查看内存



可见输入的数字以ASCII的形式被存入到了预先设定好的内存区域中，前面的**08表示数字个数，后面会用到**

**下面进行数字的存储部分，分析如下**

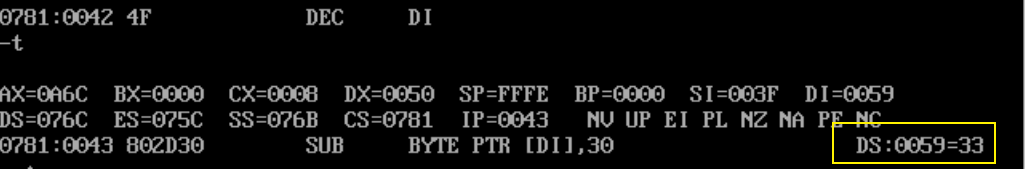


该段程序实现了对输入数字个数的相关记录，CX作为Loop过程中的计数器，同时使得DI寄存器指向了缓冲区的数据段，为后续各位的数字变换做准备。



上图程序段的作用表示将DI指向缓冲区内存的末端

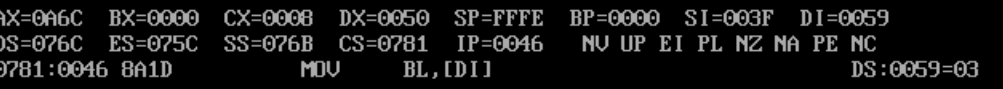
执行完毕后如下所示



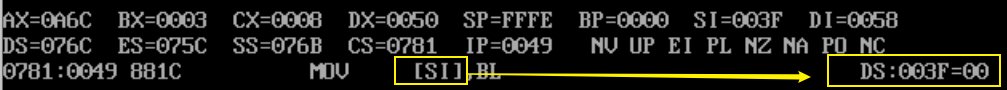
可见[DI]指向了最后一个数字ASCII量’33’

**下面进行循环的数字处理部分，前面已经将SI指向了内存空间的最后一个单元程序段如下所示，仅以单次循环进行说明**

先执行末尾单元数字（最低位）数字转换，如下图所示，可见转换成功，转换为数字03

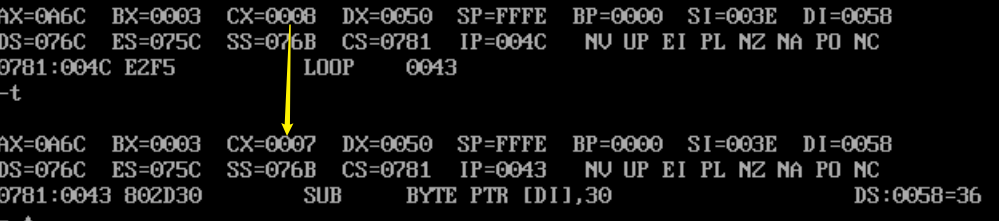


再将03存入中间量寄存器BL中，并使DI自减1，指向高位



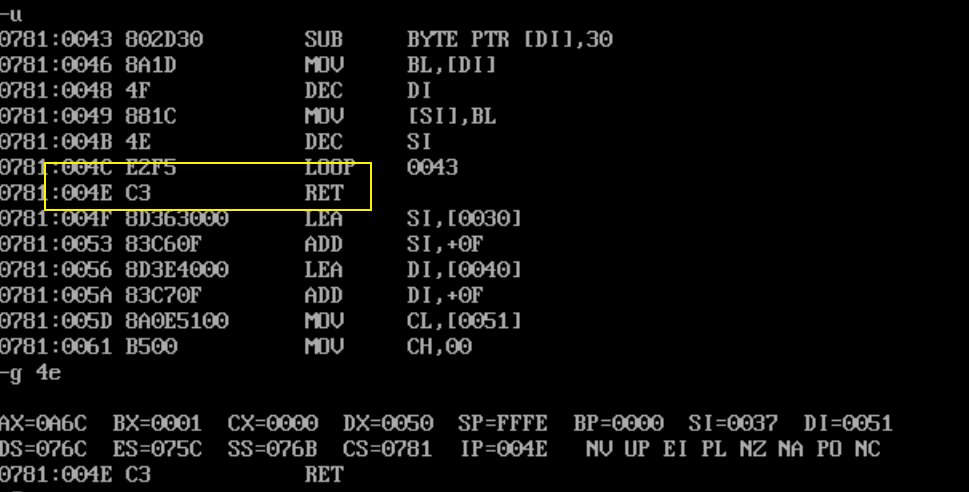
下面将BL的值存入[SI]中，第一次会存入到末尾内存单元中，如上图所示，再将SI自减一，指向高位，循环结束

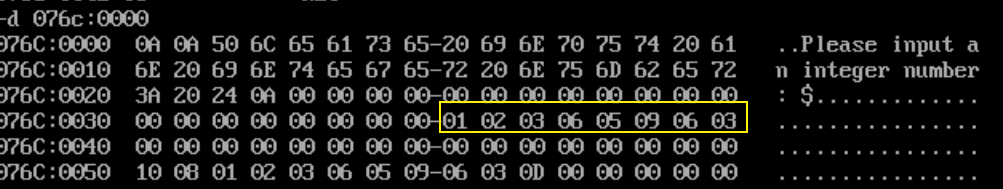
循环执行完毕后-d如下图所示



CX自减1，CX初始表示输入了8位数字，从而控制循环次数

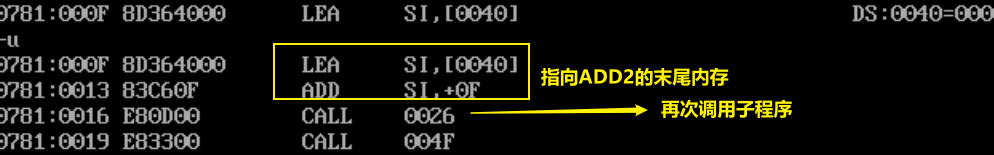
-g执行所有的循环，执行完毕后子程序结束，利用-d查看如下图所示



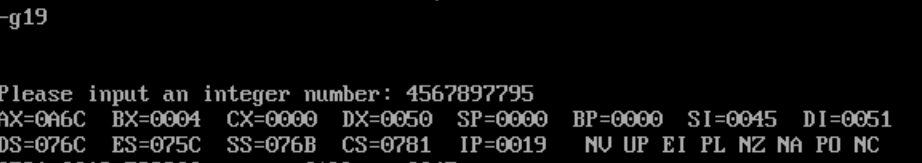


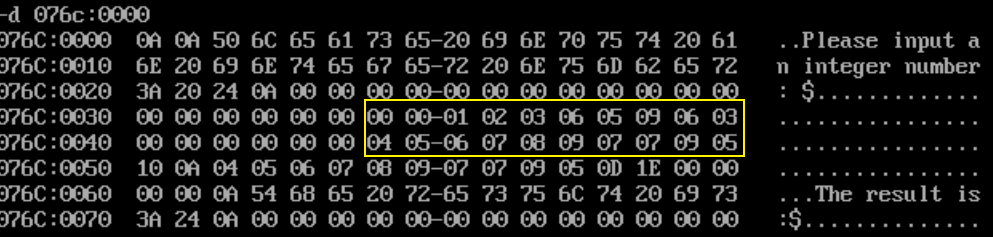
可见数字处理成功

同理下面进行第二个加数的输入，如下图所示



-g19执行，-d查看，如下所示





数字同样被成功存入

**下面进行各位的数字相加过程**

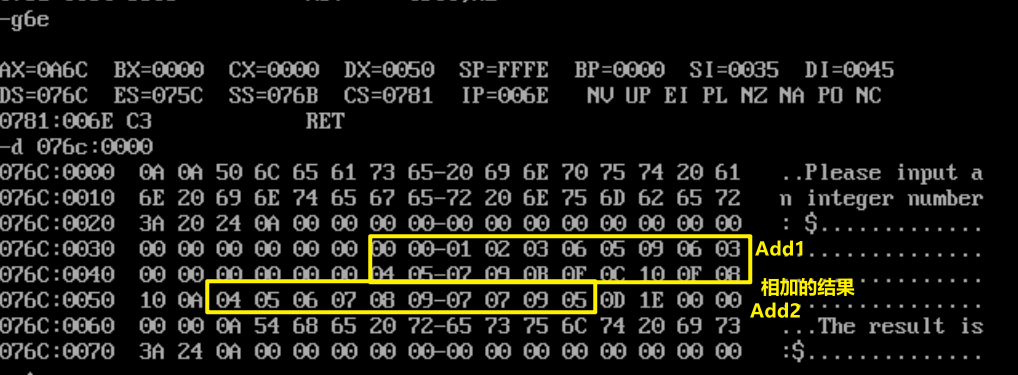


**Bug说明：这里仅利用ADD2的位数记录作诸位相加过程的循环变量，忽略了ADD1的位数大于ADD2的情况，如果不进行比较，仅记录小数字的位数的进行循环，则很明显会出现加法错误。**

**下面进行循环体的相关分析，-g进入循环体，这里CX=A表示数字位数为10**



**循环体的思路仍为倒序遍历相加，过程与数字输入的实现逻辑类似，这里不再赘述，利用-g完成循环体，查看结果如下所示**



**可见相加的结果出现了十六进制的形式，也就是部分不进位的问题，考虑到每一位最大进位数不可能**

【实验心得】

给爷整吐了