

中南大学

CENTRAL SOUTH UNIVERSITY

计算机原理与汇编语言 课程实验报告

题 目 计算机原理与汇编语言课程实验报告

学生姓名 赖涵

班级学号 智能 2003 班 8207201805

指导教师 李仪 邹逸群

设计时间 2022 年 6 月

目录

第 1 章 实验准备	1
1.1 实验目的	1
1.2 实验环境	1
1.3 实验中主要用到的汇编 debug 方法	2
第 2 章 实验解析、代码及结果	4
2.1 实验一	4
2.2 实验二	5
2.3 实验三	6
2.4 实验四	9
2.5 实验五	10
2.6 实验六	13
2.7 实验七	15
2.8 实验八	15
2.9 实验九	21
2.10 实验十	25
2.11 实验的优点和不足	28
第 3 章 心得体会	30
参考文献	31

第1章 实验准备

1.1 实验目的

汇编语言及汇编语言程序设计是从事计算机研究与应用、软件开发的专业人员必须学习和掌握的专业基础训练之一。其是一种可以直接访问计算机硬件，执行效率高，占用资源少，不可移植，只适用于特定的处理器的语言。汇编语言是很多课程，如数据结构、操作系统、微机原理、计算机组成原理等的重要基础，是人和机器沟通的桥梁。，学习汇编语言，有利于我们充分获得底层编程的体验，深刻理解机器运行程序的处理。

在该实验中，我们主要面对的是以8086CPU为中央处理器的计算机。8086CPU常用而且结构简介，方便实践，便于学习，有利于我们开展实验和实践。

1.2 实验环境

本次实验使用的平台是vscode。通过vscode中的MASM/TASM拓展和VSCode DOSBox拓展，配置环境，配置好后可在vscode中编写并运行汇编程序。



MASM是Microsoft Macro Assembler 的缩写，它是微软为x86微处理器家族，所写的一套宏组译器。它最初是用来发展在MS—DOS上面执行的软件，同时，它也该系统最流行的组译器。TASM是Borland公司推出的汇编编译器，也是一种使用很广泛的编译器，与MASM相比，TASM的升级没有这么频繁，其中 4.0版是TASM系列编译器编写DOS程序使用最广泛的版本。TASM的最后一个版本是5.0版，其支持WIN32编程，并单独为WIN32编程附带有一整套32位程序：32位的编译器TASM32.EXE、连接器TLINK32.EXE和资源编译器BRC32.EXE。与这些32位程序对应的16位工具在软件包中依然存在，文件名为TASM.EXE, TLINK.EXE和BRC.EXE等。

DOSBox 是一个 DOS 模拟程序，由于它采用的是 SDL 库，所以可以很方便的移植到其他平台。DOSBox 的最新版本支持在 Windows、Linux、Mac OS X、Android 、

webOS 等多种系统中运行。

1.3 实验中主要用到的汇编 debug 方法

1. R 命令

作用：显示出当前所有寄存器和标志位的状态。

格式：R 。

2. H 命令

作用：计算两个十六进制数的和与差。

格式：H。

3. D 命令

作用：显示内存区域的内容。显示内容中，最左边是内存的起始地址，中间以十六进制的形式显示内存值，最右边以 ASCII 码的形式显示内存值。每行最多显示 16 个字节的内容。

命令 D 可以带参数也可省略参数。设 DEBUG 启动时 DS 的值为 X，当省略参数时，命令 D 显示内容以 X: 100 为起始，每次显示 128 个字节的内容。以后再执行不带参数的命令 D 时，DEBUG 将按上次的位置接着显示下去。

带参数的三种格式为：

格式一：d [起始位置]。DEBUG 从起始位置开始显示 128 个字节的内容。

格式二：d [起始位置] [结束位置]。DEBUG 从起始位置开始一直显示到结束位置。

格式三：d [起始位置] [L 长度]，长度以 L 参数为标识。DEBUG 从起始位置开始显示指定长度的内容。

4. E 命令

作用：改变内存单位的内容。

格式：E [起始位置]。

5. F 命令：

作用：使用指定的值填充指定内存区域中的地址。

格式：F [范围] [填充列表]。

6. M 命令：

作用：将指定内存区域的数据复制到指定的地址去。

格式：M [范围] [指定地址]。

7. C 命令:

作用: 将两块内存的内容进行比较。

格式: C [范围] [指定地址]。将指定范围的内存区域与从指定地址开始的相同长度的内存区域逐个字节进行比较, 列出不同的内容。

8. G 命令:

作用: 执行汇编指令。

格式: G [=起始地址] [断点地址]。从起始地址开始执行到断点地址。如果不设置断点, 则程序一直运行到中止指令才停止。

9. U 命令:

作用: 对机器代码反汇编显示。

格式: U [范围]。

10. T 命令:

作用: 执行汇编程序, 单步跟踪。

格式: T [=地址] [指令数]。如果忽略“地址”的话, T 命令从 CS:IP 处开始运行。
“指令数”是要单步执行的指令的数量。

11. P 命令:

作用: 执行汇编程序, 单步跟踪。

与 T 命令不同的是: P 命令不会跟踪进入子程序或软中断。

P 命令的使用方式与 T 命令的使用方式完全相同。

第2章 实验解析、代码及结果

2.1 实验一

1. 实验题目：

编写一个累计加法，从1加到5，将结果保存至AX中。

2. 实验原理及解析

该程序主要需要使用mov指令和add指令。

mov指令主要有以下几种形式：

mov 寄存器, 数据 ; mov 寄存器, 寄存器; mov 寄存器, 内存单元; mov 内存单元, 寄存器; mov 段寄存器, 寄存器。

add指令主要有以下几种形式：

add 寄存器, 数据; add 寄存器, 寄存器; add 寄存器, 内存单元; add 内存单元, 寄存器。

sub指令和add指令形式相同。

在此题中，从1加到5数据量较小，可以直接写在代码中，通过手动累加或者叠加实现。也可以通过loop循环、jmp实现。因为写在代码中手动累加的方法不利于代码和移植，因此为提高代码的可重用性，在该实验中采用loop指令来组织程序。

loop指令的格式是：loop 标号。CPU执行loop指令的时候，要进行两步操作：

①(cx)=(cx)-1 ②判断cx中的值，若不为0则转至标号处执行程序，若为0则向下执行。通常，在使用loop指令来实现循环功能的时候，我们使用cx来存放循环次数。

每段程序的最后应该有结束语。在该实验中，我们使用的是指令：

```
mov ax, 4c00H
```

```
int 21H
```

3. 代码实现：

;编写一个累计加法，从1加到5，将结果保存至AX中

```
mov ax,0      ;初始化 ax 寄存器
mov cx,5      ;循环 5 次
mov bx,1      ;循环变量
```

s:

```
add ax,bx      ;ax+bx, 结果保存在 ax 中
inc bx         ;bx 自加
loop s         ;循环相加
```

;相加之后 AX 结果为 000FH

4. 结果展示:

1+2+3+4+5=15, 结果正确。

2.2 实验二

1. 实验题目:

编写一个累计减法, 被减数是1001 1000B, 减数是0100 0000B, 连续减5次, 观察FLAGS的变化。

2. 实验原理及解析:

由题中可以看出, 被减数和减数都是16位二进制数。因此可将被减数和减数分别存放在两个寄存器中, 并通过loop指令进行循环相减。

在减法指令 sub ax, bx 中, 相减后结果的值保存在ax中。

3. 代码实现:

```
mov ax,10011000B      ;被减数
mov dx,01000000B      ;减数
mov cx,5               ;循环 5 次
```

S:

```
sub ax,dx              ;相减
loop s                 ;循环相减
```

4. 结果展示:

```

AX=0058 BX=0000 CX=0004 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0009  NU UP EI PL NZ NA PO NC
076C:0009 2BC2 SUB AX,DX
-t

AX=0018 BX=0000 CX=0004 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000B  NU UP EI PL NZ NA PE NC
076C:000B E2FC LOOP 0009
-t

AX=0018 BX=0000 CX=0003 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0009  NU UP EI PL NZ NA PE NC
076C:0009 2BC2 SUB AX,DX
-t

AX=FFD8 BX=0000 CX=0003 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000B  NU UP EI NG NZ NA PE CY
076C:000B E2FC LOOP 0009
-t

AX=FFD8 BX=0000 CX=0002 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000B  NU UP EI NG NZ NA PO NC
076C:000B E2FC LOOP 0009
-t

AX=FF98 BX=0000 CX=0001 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0009  NU UP EI NG NZ NA PO NC
076C:0009 2BC2 SUB AX,DX
-t

AX=FF58 BX=0000 CX=0001 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000B  NU UP EI NG NZ NA PO NC
076C:000B E2FC LOOP 0009
-t

AX=FF58 BX=0000 CX=0000 DX=0040 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000D  NU UP EI NG NZ NA PO NC
076C:000D B8004C MOV AX,4C00

```

FLAGS标志寄存器的变化为:

第一次不变, 第二次不变。

第三次相减时SF符号位由PL变为NG, 表示结果由正数变为负数; CF标志位由NC变为CY, 表示借位。

第四次时CF标志位由CY变为NC, 表示此时不需要借位; 奇偶标志位由PE变为PO, 即运算后结果中含有奇数个1。

第五次不变。

2.3 实验三

1. 实验题目:

编写一个 16 位的乘法, 被乘数是 100H, 乘数是 100H, 观察 Flags 的变化, 编写一个 32 位的乘法, 被乘数是 0F0FH, 乘数是 FF00H, 观察 Flags 的变化。编写一个 32 位的乘法, 被乘数是 0F0FH, 乘数是 FF00H, 观察 Flags 的变化。

2. 实验原理及解析:

16位乘法中, 被乘数存放在ax中, mul bx 指令后, 实现了ax*bx, 并将乘积的高位存放在dx中, 低位存放在ax中。在该题中我们进行的是无符号数的乘法。

FLAGS寄存器指的是标志寄存器，标志寄存器有三种作用：存储相关指令的某些执行结果、位CPU执行相关指令提供行为依据、控制CPU的相关工作方式。其中存储的信息通常被称为程序状态字（PSW）。

标志寄存器的结构：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

标志寄存器所展示的信息的意义：

标志位	标志位名称	=1	=0
CF	进位标志	CY/进位	NC/无进位
PF	奇偶标志	PE/偶	PO/奇
AF	辅助进位标志	AC/进位	NA/无进位
ZF	零标志	ZR/等于零	NZ/不等于零
SF	符号标志	NG/负	PL/非负
TF	跟踪标志		
IF	中断标志	EI/允许	DI/禁止
DF	方向标志	DN/减少	UP/增加
OF	溢出标志	OV/溢出	NV/未溢出

因为8086中没有32位寄存器，因此在该实验中，我们将32位乘法分解为4个16位乘法来进行运算，并将结果保存在内存中。

3. 代码实现：

16 位乘法：

```

mov ax,data
mov ds,ax          ;初始化段寄存器
mov ax,100H
mov dx,100H
mul dx
  
```

32 位乘法：

```
data segment
```

```
x1 dw 0F0FH      ;被乘数低位
xh dw 0000H      ;被乘数高位
y1 dw 0FF00H     ;乘数低位
yh dw 0000H      ;乘数高位
xy dw 8 dup(?)   ;存放结果
data ends
code segment
main:
    ;编写一个 16 位的乘法，被乘数是 100H，乘数是 100H，观察 Flags 的变化，编写一个 32 位的乘法，被乘数是 0F0FH，乘数是 FF00H，观察 Flags 的变化

    mov ax,data
    mov ds,ax          ;初始化段寄存器

    ;32 位乘法可以分解为 4 个 16 位乘法来进行
    mov ax,x1
    mov dx,y1
    mul dx
    mov [xy],ax
    mov [xy+2],dx
    ;x 低位和 y 低位相乘结果高位存放在 xy+2 中，低位存放在 xy 中

    mov ax,xh
    mov dx,y1
    mul dx
    add [xy+2],ax
    adc [xy+4],dx        ;带进位加法
    adc [xy+6],0         ;保存进位
    ;x 高位和 y 低位相乘结果低位累加到 xy+2 中，高位累加到 xy+4 中

    mov ax,x1
    mov dx,yh
    mul dx
    add [xy+2],ax
    adc [xy+4],dx
    adc [xy+6],0         ;将进位保存到 xy+6
    ;x 低位和 y 高位相乘结果低位累加到 xy+2 中，高位带进位累加到 xy+4 中，产生的进位保存到 xy+6 中

    mov ax,xh
    mov dx,yh
    mul dx
    add [xy+4],ax
```

adc [xy+6],dx

;带进位加法

4. 结果展示:

16 位乘法:

```
AX=0100 BX=0000 CX=0012 DX=0100 SP=0000 BP=0000 SI=0000 DI=0000
DS=076C ES=075C SS=076B CS=076C IP=000B 00 UP EI PL NZ NA PO NC
076C:000B F7E2 MUL DX
-t
AX=0000 BX=0000 CX=0012 DX=0001 SP=0000 BP=0000 SI=0000 DI=0000
DS=076C ES=075C SS=076B CS=076C IP=000D 00 UP EI PL ZR NA PO CY
076C:000D B8094C MOV AX,4C09
```

乘积结果为 0001 0000，其中高位保存在 dx 中，低位保存在 ax 中。相乘后，零标志从 NZ 不等于零变为 ZR 等于零，进位标志由 NC 无进位变成 CY 进位，溢出标志由 NV 未溢出变成了 OV 溢出。

32 位乘法:

```
077 DX=0EFF SP=0000 BP=0000 SI=0000 DI=0000
76B CS=076E IP=0018 00 UP EI PL NZ NA PO CY
MOV DX,[0004] DS:0000

077 DX=FF00 SP=0000 BP=0000 SI=0000 DI=0000
76B CS=076E IP=001C 00 UP EI PL NZ NA PO CY
MUL DX DS:0000

077 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
76B CS=076E IP=001E 00 UP EI PL ZR NA PE NC
ADD [000A],AX DS:0000

077 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
76B CS=076E IP=0022 00 UP EI PL ZR NA PE NC
ADC [000C],DX DS:0000

077 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
76B CS=076E IP=0026 00 UP EI PL ZR NA PE NC
ADC WORD PTR [000E],*03 DS:0000

077 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
76B CS=076E IP=0041 00 UP EI PL ZR NA PE NC
MOV AX,[0002] DS:0000

-d ds:0
076C:0000 0F 0F 00 00 00 FF 00 00 00 F1 FF 0E 00 00 00 00
076C:0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
076C:0020 00 6C 67 BF DB 01 00 00 00 00 00 00 00 00 00
```

在该实验中，我们将 32 位乘法运算拆成了 4 个 16 位乘法运算。在第一次乘法运算后，溢出标志由 NV 未溢出变成了 OV 溢出，进位标志由 NC 无进位变成 CY 进位，零标志依然为 NZ 不等于零。在第二次乘法运算后，溢出标志由 OV 溢出变成了 NV 未溢出，进位标志由 CY 进位变成 NC 无进位，零标志从 NZ 不等于零变为 ZR 等于零。随后的运算中随数据发生类似上面的转变。

最终结果存在内存中。结果如上图，低位在前，高位在后。最终结果为：0EFFF100H，结果正确。

2.4 实验四

1. 实验题目:

编写一个 16 位的除法，被除数是 100H，除数是 100H，观察 Flags 的变化，编写一个 32 位的除法，被除数是 0F0FH，除数是 00FFH，观察 Flags 的变化。

2. 实验原理及解析:

16位除法中，被除数为16位，除数为8位。div bl，运算完后，ax中高位放余数，低位放商，即：(al)=(ax)/(bl), (ah)=(ax)%(bl)。

32位除法中，被除数为32位，除数为16位。dx中存放被除数的高16位，ax中存放被除数的低16位。运算完后，dx中存放余数，ax中存放商。

在进行乘法和除法的时候应注意观察标志寄存器的变化，关注是否有产生溢出。

3. 代码实现：

16 位除法：

```
mov ax,100H      ;被除数存放在 AX 中
mov bx,100H      ;除数存放在 BX 中
div bx
```

32 位除法：

```
mov dx,0000H     ;被除数高位存放在 DX 中
mov ax,0F0FH     ;被除数低位存放在 AX 中
mov bx,00FFH     ;除数存放在 BX 中
div bx
```

4. 结果展示：

16 位除法：

```
AX=0100 BX=0100 CX=0018 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0006 NU UP EI PL NZ NA PO NC
076C:0006 F7F3          DIV     BX
-t
AX=0001 BX=0100 CX=0018 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0008 NU UP EI PL NZ NA PO NC
076C:0008 BA0000      MOV     DX,0000
```

ax 的高位存放着余数，低位存放着商。商为 1，余数为 0。

32 位除法：

```
AX=0F0F BX=00FF CX=0018 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0011 NU UP EI PL NZ NA PO NC
076C:0011 F7F3          DIV     BX
-t
AX=000F BX=00FF CX=0018 DX=001E SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0013 NU UP EI PL NZ NA PO NC
076C:0013 B8004C      MOV     AX,4C00
```

2.5 实验五

1. 实验题目：

编写一个累计加法，被加数是0FH，加数是01H，观察Flags的变化，被加数是0FFH，加数是01H，观察Flags的变化，被加数是0FFFH，加数是01H，观察Flags的变化，被加数是FFFFH，加数是01H，观察Flags的变化，被加数是FFFFFFFFH加数是01H，观察Flags的变化。

2. 实验原理及解析:

当被加数位 0FH, 0FFH, 0FFFH, FFFFH 时, 皆只需要使用一个寄存器, 并使用 mov 寄存器, 数据, add 寄存器, 数据, 这两条指令即可完成。

当被加数位 FFFFFFFFH 时, 需将被加数的高位和低位分开存储, 低位存储在 ax 中, 高位存储在 dx 中, 先低位和加数相加, 再加上进位。通过观察标志寄存器 flags 的变化可知是否产生进位。

adc 是带进位的加法指令, 它利用了 CF 位上记录的进位值。指令格式为: adc 操作对象 1, 操作对象 2, 功能为: 操作对象 1=操作对象 1+操作对象 2+CF。adc 指令和 add 指令相配合可以完成对更大的数的加法运算。

类似的, 有 sbb 带借位的减法指令, 它利用的是 CF 位上记录的借位值。指令格式为: sbb 操作对象 1, 操作对象 2, 功能为: 操作对象 1=操作对象 1-操作对象 2-CF。

如果 CF 的值是被 sub 指令设置的, 那么他的含义就是借位值, 如果是被 add 指令设置的, 那么他的含义就是进位值。

3. 代码实现:

```
;被加数是 0FH, 加数是 01H
mov ax,0FH
add ax,01H

;被加数是 0FFH, 加数是 01H
mov ax,0FFH
add ax,01H

;被加数是 0FFFH, 加数是 01H
mov ax,0FFFH
add ax,01H

;被加数是 0FFFFH, 加数是 01H
mov ax,0FFFFH
add ax,01H

;被加数是 FFFFFFFFH, 加数是 01H
mov ax,0FFFFH ;ax 为低位
mov dx,0FFFFH ;dx 为高位
add ax,01H ;低位与加数相加
adc dx,0 ;保存进位
```

4. 结果展示:

被加数是 0FH, 加数是 01H:

```
AX=FFFF BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0000 NU UP EI PL NZ NA PO NC
076C:0000 B80F00 MOV AX,000F
-t
AX=000F BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0003 NU UP EI PL NZ NA PO NC
076C:0003 B3C001 ADD AX,+01
-t
AX=0010 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0006 NU UP EI PL NZ AC PO NC
```

结果为 0010, FLAGS 寄存器没有变化。

被加数是 0FFH, 加数是 01H:

```
AX=0010 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0006 NU UP EI PL NZ AC PO NC
076C:0006 B8FF00 MOV AX,00FF
-t
AX=00FF BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0009 NU UP EI PL NZ AC PO NC
076C:0009 B3C001 ADD AX,+01
-t
AX=0100 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000C NU UP EI PL NZ AC PE NC
```

结果为 0100, FLAGS 寄存器没有变化。

被加数是 0FFFH, 加数是 01H:

```
AX=0100 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000C NU UP EI PL NZ AC PE NC
076C:000C B8FF0F MOV AX,0FFF
-t
AX=0FFF BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000F NU UP EI PL NZ AC PE NC
076C:000F B3C001 ADD AX,+01
-t
AX=1000 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0012 NU UP EI PL NZ AC PE NC
```

结果为 1000, FLAGS 寄存器没有变化。

被加数是 0FFFFH, 加数是 01H:

```
AX=1000 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0012 NU UP EI PL NZ AC PE NC
076C:0012 B8FFFF MOV AX,FFFF
-t
AX=FFFF BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0015 NU UP EI PL NZ AC PE NC
076C:0015 B3C001 ADD AX,+01
-t
AX=0000 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0018 NU UP EI PL ZR AC PE CY
```

结果为 0000, 零标志从 NZ 不等于零变为 ZR 等于零, 进位标志由 NC 无进位变成 CY 进位。

被加数是 FFFFFFFFH, 加数是 01H:

AX=FFFF	BX=0000	CX=0029	DX=0000	SP=0000	BP=0000	SI=0000	DI=0000
DS=075C	ES=075C	SS=076B	CS=076C	IP=001B	NU	UP	EI PL ZR AC PE CY
076C:001B	BAFFFF	MOV	DX,FFFF				
AX=FFFF	BX=0000	CX=0029	DX=FFFF	SP=0000	BP=0000	SI=0000	DI=0000
DS=075C	ES=075C	SS=076B	CS=076C	IP=001E	NU	UP	EI PL ZR AC PE CY
076C:001E	83C001	ADD	AX,+01				
AX=0000	BX=0000	CX=0029	DX=FFFF	SP=0000	BP=0000	SI=0000	DI=0000
DS=075C	ES=075C	SS=076B	CS=076C	IP=0021	NU	UP	EI PL ZR AC PE CY
076C:0021	83D200	ADC	DX,+00				
AX=0000	BX=0000	CX=0029	DX=0000	SP=0000	BP=0000	SI=0000	DI=0000
DS=075C	ES=075C	SS=076B	CS=076C	IP=0024	NU	UP	EI PL ZR AC PE CY

add 指令后，ax 为 0000，dx 为 FFFF，ax 中存放的是低位，dx 中存放的是高位。add 指令后，FLAGS 寄存器没有变化，说明此时 ax 产生了进位，通过 adc 指令，高位加上这个进位，让 dx 等于 0000。此时 FLAGS 寄存器还是没有发生变化，因为高位也产生了一个进位。

此时结果为：ax 中为 0000，dx 中为 0000，CF=1。

2.6 实验六

1. 实验题目：

编写一个移位运算，将 8F1DH 存至 AX，然后用指令右移 1 位然后左移 1 位，显示结果并观察 Flags 的变化。将 8F1DH 存至 AX 中，然后带 CF 位左移 5 位，并右移 7 位，观察 Flags 的变化，并给出结果。

2. 实验原理及解析：

8086CPU 中有 8 条移位指令，主要分为两大类：非循环移位指令和循环移位指令。其中第一小问为非循环移位指令，第二小问为循环移位指令。

因为我们使用的是无符号数，因此进行的是逻辑移位，第一小问应使用 SHL 和 SHR 指令进行移位。第二小问为带进位的循环左移和带进位的循环右移，使用 RCL 和 RCR 指令进行移位。其中 SHL 将目标操作数向左移动指定的位数，低位补入相应个数的 0。CF 的内容为最后移入位的值；SHR 将目标操作数向右移动指定的位数，最高位补入相应个数的 0。CF 的内容为最后移入位的值。RCL 将目的操作数连同 CF 标志一起向左循环移动规定的位数；RCR 的移动方式和 RCL 相同，只是 RCR 指令向右移动。

3. 代码实现：

```
mov ax,8F1DH      ;1000 1111 0001 1101
shr ax,1          ;逻辑右移一位
shl ax,1          ;逻辑左移一位

mov ax,8F1DH
mov cl,5h
```

RCL ax,cl ;带进位的循环左移 5 次

mov cl,7h

RCR ax,cl ;带进位的循环右移 7 次

4. 结果展示:

将 8F1DH 右移 1 位后再左移 1 位:

```
AX=8F1D BX=0000 CX=0017 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0003  OV UP EI PL NZ NA PO NC
076C:0003 D1E8 SHR AX,1
--t
AX=478E BX=0000 CX=0017 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0005  OV UP EI PL NZ AC PE CY
076C:0005 D1E0 SHL AX,1
--t
AX=8F1C BX=0000 CX=0017 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0007  OV UP EI NG NZ AC PO NC
076C:0007 B81D8F MOV AX,8F1D
```

逻辑右移后, ax 变为 478E, 移位后最后一位移入 CF 中, 因此 CF 变为 CY, CF 与最高位相反, 溢出, 因为 0F 变为 0V。移位时向低字节移入 1, 因此 AF 变为 AC。

逻辑左移一位后, 由于最低位移丢, AX 变为 8F1CH。最高位的 0 移入 CF 中, 因此 CF 变为 NC; 移位时向低字节移入 1, 因此 AF 变为 AC; CF 为 0, 最高位为 1, 因此 0F 变为 0V。

最终结果为 8F1CH。

将 8F1DH 存至 AX 中, 然后带 CF 位左移 5 位, 并右移 7 位:

```
AX=8F1D BX=0000 CX=0005 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000C  OV UP EI NG NZ AC PO NC
076C:000C D3D0 RCL AX,CL
--t
AX=E3A8 BX=0000 CX=0005 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=000E  OV UP EI NG NZ AC PO CY
076C:000E B107 MOV CL,07
--t
AX=E3A8 BX=0000 CX=0007 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0010  OV UP EI NG NZ AC PO CY
076C:0010 D3DB RCR AX,CL
--t
AX=A3C7 BX=0000 CX=0007 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075C ES=075C SS=076B CS=076C IP=0012  OV UP EI NG NZ AC PO NC
076C:0012 B8004C MOV AX,4C00
```

带进位循环左移 5 次后有, 循环移位前 0 1000 1111 0001 1101, 循环移位后 1 1110 0011 1010 1000。带 CF 进位循环左移 5 次之后 AX 为 E3A8H, 最高有效位和 CF 均为 1, 未溢出, CF 为 1, 产生进位。

带进位循环右移 7 次后有, 循环移位前 1 1110 0011 1010 1000, 循环移位后 0 1010 0011 1100 0111。带 CF 进位循环右移 7 次之后 AX 为 A3C7H, 最高有效位为 1, CF 为 0, 溢出, CF 为 0, 未产生进位。

2.7 实验七

1. 实验题目：

将 71D2H 存至 AX 中, 5DF1H 存至 CX 中, DST 为 AX, REG 为 CX, 实现双精度右移 2 次, 交换 DST 与 REG, 然后左移 4 次, 分别查看结果。

2. 实验原理及解析：

通过在 8086 中写入 .386 可以在代码中使用 80386 的指令。在此我们使用 SHLD (双精度左移) 和 SHRD (双精度右移) 进行移位。

双精度左移: SHLD dest, source, count 指令将目的操作数向左移动指定位数。移动形成的空位由源操作数的高位填充。源操作数不变, 但是符号标志位、零标志位、辅助进位标志位、奇偶标志位和进位标志位可能会受影响。

双精度右移: SHRD dest, source, count 指令将目的操作数向右移动指定位数。移动形成的空位由源操作数的低位填充。

3. 代码实现：

```
.386
mov ax, 71D2H
mov cx, 5DF1H
SHRD ax, cx, 1
SHRD ax, cx, 1
SHLD cx, ax, 1
SHLD cx, ax, 1
SHLD cx, ax, 1
SHLD cx, ax, 1
```

4. 实验结果：

AX=FFFF BX=0000 CX=0043 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=0000 NU UP EI PL NZ NA PO NC 076E:0000 B8D271 MDU AX,71D2	AX=DC74 BX=0000 CX=5DF1 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=000E NU UP EI NG NZ NA PE CY 076E:000E 0F DB 0F
AX=71D2 BX=0000 CX=0043 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=0003 NU UP EI PL NZ NA PO NC 076E:0003 D9F15D MDU CX,5DF1	AX=DC74 BX=0000 CX=B8E3 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=0012 NU UP EI NG NZ NA PO CY 076E:0012 0F DB 0F
AX=71D2 BX=0000 CX=5DF1 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=0006 NU UP EI PL NZ NA PO NC 076E:0006 0F DB 0F	AX=DC74 BX=0000 CX=77C7 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=0016 DU UP EI PL NZ NA PO CY 076E:0016 0F DB 0F
AX=B8E3 BX=0000 CX=5DF1 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=000A DU UP EI NG NZ NA PO NC 076E:000A 0F DB 0F	AX=DC74 BX=0000 CX=EF8F DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=001A NU UP EI NG NZ NA PO NC 076E:001A 0F DB 0F
AX=DC74 BX=0000 CX=5DF1 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000 DS=075C ES=075C SS=076B CS=076E IP=000E NU UP EI NG NZ NA PE CY 076E:000E B8D271 MDU AX,4C00	

2.8 实验八

1. 实验题目：

实现压缩BCD码的加减法, 用压缩BCD码实现 (21+71), (12+49), (65 +82), (46-33), (74-58), (43-54) 的十进制加减法。然后又用非压缩BCD实现上述6个式子。

2. 实验原理及解析:

BCD码即8421码,的特点是4位用二进制数0000B~1001B,来表示一位十进制数0~9,而每4位二进制数之间的进位又是十进制的形式。因此BCD码既具有二进制数的特点又具有十进制数的特点。BCD码的使用为十进制数在计算机内的表示提供了一种简单而实用的手段。

未压缩的BCD码每个字节中存放一个十进制数字位,而压缩的BCD码在一个字节中存放两个十进制数字位。

题中的数据使用压缩的BCD码表示分别为: 21H, 71H, 12H, 49H, 65H, 82H, 46H, 33H, 74H, 58H, 43H, 54H。

使用未压缩的BCD码表示分别为: 0201H, 0701H, 0102H, 0409H, 0605H, 0802H, 0406H, 0303H, 0704H, 0508H, 0403H, 0504H。

将数据存放在data中,已方便后续代码编写,程序使用和数据存取。

aaa (ASCII adjust after addition) 指令,是BCD指令集中的一个指令,用于在两个未打包的BCD值相加后,调整al和ah寄存器的内容。

aaa指令做两件事情:

如果al的低4位是在0到9之间,保留低4位,清除高4位,如果al的低4位在10到15之间,则通过加6,来使得低4位在0到9之间,然后再对高4位清零;如果al的低4位是在0到9之间,ah值不变,CF和AF标志清零,否则,ah=ah+1,并设置CF和AF标志。

das指令也是用于调整AL的值,AL是由指令SUB或SBB运算二个压缩型BCD码所得到的结果。其调整规则如下:

如果AL的低四位大于9,或AF=1,那么,AL=AL-06H,并置AF=1;如果AL的高四位大于9,或CF=1,那么,AL=AL-60H,并置CF=1;

如果以上两点都不成立,则,清除标志位AF和CF。

经过调整后,AL的值仍是压缩型BCD码,即:二个压缩型BCD码相减,并进行调整后,得到的结果还是压缩型BCD码。

3. 代码实现:

压缩BCD码:

```
data segment
a db 21H,12H,65H,46H,74H,43H
b db 71H,49H,82H,33H,58H,54H
```

```
res db 12 dup (0)      ;存放结果
data ends
```

主要操作:

```
mov ax,data
mov ds,ax              ;初始化数据段寄存器

mov bx,0               ;循环变量,用于保存结果
mov di,0               ;变址寻址
mov cx,3               ;循环 3 次

ad:
sahf                   ;清空标志寄存器
mov ah,0H              ;清空高位
mov al,0[di]            ;送加数
mov dl,6[di]            ;送加数
add al,dl
daa                    ;daa 指令来进行十进制调整,
;如果 AL 寄存器中低 4 位大于 9 或者辅助进位 AF=1,则 AL=AL+6,且 AF=1
;如果 AL>=0A0H 或 CF=1,则 AL=AL+60H,且 CF=1
adc ah,0               ;保存进位

mov 12[bx],ah          ;存放结果高位
inc bx
mov 12[bx],al          ;存放结果低位
inc bx
inc di                 ;变址寻址
loop ad

mov cx,3
;压缩 BCD 码减法
su:
sahf                   ;清空标志寄存器
mov ah,0H              ;高位置零
mov al,0[di]            ;送被减数
mov dl,6[di]            ;送减数
sub al,dl
das                    ;BCD 码调整
;如果 AF=1 或 AL 寄存器中低 4 位大于 9,则 AL=AL-6,且 AF=1
;如果 AL>=0A0H 或 CF=1,则 AL=AL-60H,且 CF=1

;可能会出现负数的情况
jnc next               ;不借位则跳转至 next
;如果借位,则说明结果为负数,此时变换为 bl-al
```

```
mov al,6[di]
mov dl,0[di]
sub al,dl
das                                ;调整 BCD 码,大于 9 则 AL=AL-6, 高四位-60, 都不成立则清除标志位
mov ah,10h                        ;最高位置为 1, 表示结果为负数
next:
    ;保存结果
    mov 12[bx],ah                ;保存结果高位
    inc bx
    mov 12[bx],al                ;保存结果低位
    inc bx                        ;bx 自加
    inc di                        ;变址寻址变量自加
    loop su
    未压缩 BCD 码:
data segment
    a dw 0201H,0102H,0605H,0406H,0704H,0403H    ;12
    b dw 0701H,0409H,0802H,0303H,0508H,0504H    ;24
    res dw 12 dup (0)                ;存放结果
data ends

stack segment
    dw 16 dup(0)
stack ends

mov ax,data
mov ds,ax                        ;初始化数据段寄存器
mov ax,stack
mov ss,ax
mov sp,16                        ;初始化栈

mov bx,0                        ;循环变量,用于保存结果
mov di,0                        ;变址寻址
mov cx,3                        ;循环 3 次
ad:
    ;非压缩 BCD 码加法
    sahf                        ;清空标志寄存器
    mov ax,0[di]                ;送加数
    mov dx,12[di]                ;送被加数
    add ah,dh
    add al,dl
    aaa                        ;非压缩 BCD 码调整
    ;如果 AL 的低 4 位大于 9 或 AF=1,则 AL=AL+6,AH=AH+1,AF=CF=1,且 AL 高四位清零
```

```
;否则 CF=AF=0,AL 高 4 位清零
cmp ah,09H          ;比较 AH 是否超过 9
jna save1           ;不高于则跳转至下方
sub ah,0AH          ;减 0AH, 修正高位
mov 24[bx],0001H    ;保存结果的进位
```

save1:

```
inc bx
mov 24[bx],ah        ;保存结果高位
inc bx
mov 24[bx],al        ;保存结果低位
inc bx
add di,2
loop ad              ;循环
```

```
mov cx,3             ;循环 3 次
```

;非压缩 BCD 码减法

su:

```
sahf                 ;清空标志寄存器
mov ax,0[di]         ;送加数
mov dx,12[di]        ;送被加数
sub ah,dh
sub al,dl
aas                  ;非压缩 BCD 码调整
;如果 AL 的低 4 位大于 9 或 AF=1,则 AL=AL-6,AH=AH-1,AF=CF=1,且 AL 高四位清零
;否则 CF=AF=0,AL 高 4 位清零
```

```
cmp ah,09H          ;比较 AH 是否超过 9
jna save2           ;不高于则跳转至下方
;如果 ah 大于 09H 说明结果为负数, 需要反过来
mov ax,12[di]
mov dx,0[di]
sub ah,dh
sub al,dl
aas
mov 24[bx],0001H    ;保存结果的借位,表示负数
```

save2:

```
inc bx
mov 24[bx],ah        ;保存结果高位
inc bx
mov 24[bx],al        ;保存结果低位
```

```
inc bx
add di,2
loop su ;循环
```

4. 结果展示:

压缩 BCD 码:

```
AX=0021 BX=0000 CX=0003 DX=0071 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0016 076E:0016 02C2 ADD AL,DL
AX=0092 BX=0000 CX=0003 DX=0071 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0018 076E:0018 27 DAA
AX=0092 BX=0000 CX=0003 DX=0071 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0019 076E:0019 80D400 ADC AH,00
AX=0092 BX=0000 CX=0003 DX=0071 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=001C 076E:001C 88670C MOV [BX+0C],
```

```
AX=0065 BX=0004 CX=0001 DX=0002 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0016 076E:0016 02C2 ADD AL,DL
AX=00E7 BX=0004 CX=0001 DX=0002 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0018 076E:0018 27 DAA
AX=0047 BX=0004 CX=0001 DX=0002 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0019 076E:0019 80D400 ADC AH,00
AX=0147 BX=0004 CX=0001 DX=0002 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=001C 076E:001C 88670C MOV [BX+0C],AH
```

```
AX=0046 BX=0006 CX=0003 DX=0033 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0032 076E:0032 2AC2 SUB AL,DL
AX=0013 BX=0006 CX=0003 DX=0033 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0034 076E:0034 2F DAS
AX=0013 BX=0006 CX=0003 DX=0033 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0035 076E:0035 730A JNB 0041
AX=0074 BX=0008 CX=0002 DX=0058 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0032 076E:0032 2AC2 SUB AL,DL
AX=001C BX=0008 CX=0002 DX=0058 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0034 076E:0034 2F DAS
AX=0016 BX=0008 CX=0002 DX=0058 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0035 076E:0035 730A JNB 0041
```

```
AX=0043 BX=000A CX=0001 DX=0054 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0032 076E:0032 2AC2 SUB AL,DL
AX=00EF BX=000A CX=0001 DX=0054 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0034 076E:0034 2F DAS
AX=0009 BX=000A CX=0001 DX=0054 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0035 076E:0035 730A JNB 0041
AX=0054 BX=000A CX=0001 DX=0043 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0032 076E:0032 2AC2 SUB AL,DL
AX=0011 BX=000A CX=0001 DX=0043 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=0034 076E:0034 2F DAS
AX=0011 BX=000A CX=0001 DX=0043 SP=0000 BP=0000 SI=0000 DI=0002
DS=076C ES=075C SS=076B CS=076E IP=003F 076E:003F B410 MOV AH,10
```

未压缩 BCD 码:

```

AX=0201 BX=0000 CX=0003 DX=0701 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:001C 02E6 ADD AH,DH
-t

AX=0901 BX=0000 CX=0003 DX=0701 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:001E 02C2 ADD AL,DL
-t

AX=0902 BX=0000 CX=0003 DX=0701 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0020 37 AAA
-t

AX=0902 BX=0000 CX=0003 DX=0701 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0021 80FC09 CMP AH,09
-t

AX=0902 BX=0000 CX=0003 DX=0701 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0024 7608 JBE 002E
-t

AX=0102 BX=0003 CX=0002 DX=0409 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:001C 02E6 ADD AH,DH
-t

AX=0502 BX=0003 CX=0002 DX=0409 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:001E 02C2 ADD AL,DL
-t

AX=050B BX=0003 CX=0002 DX=0409 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0020 37 AAA
-t

AX=0601 BX=0003 CX=0002 DX=0409 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0021 80FC09 CMP AH,09
-t

AX=0601 BX=0003 CX=0002 DX=0409 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0024 7608 JBE 002E
-t

AX=0E07 BX=0006 CX=0001 DX=0002 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0026 80EC0A SUB AH,0A
-t

AX=0407 BX=0006 CX=0001 DX=0002 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0029 C747180100 MOV WORD PTR [DI],SI
-t

AX=0406 BX=0009 CX=0003 DX=0303 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0045 2AE6 SUB AH,DH
-t
Error
AX=0106 BX=0009 CX=0003 DX=0303 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0047 2AC2 SUB AL,DL
-t

AX=0103 BX=0009 CX=0003 DX=0303 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0049 3F AAS
-t

AX=0103 BX=0009 CX=0003 DX=0303 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:004A 80FC09 CMP AH,09
-t

AX=0704 BX=000C CX=0002 DX=0508 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0045 2AE6 SUB AH,DH
-t

AX=0204 BX=000C CX=0002 DX=0508 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0047 2AC2 SUB AL,DL
-t

AX=02FC BX=000C CX=0002 DX=0508 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:0049 3F AAS
-t

AX=0106 BX=000C CX=0002 DX=0508 SP=0000
DS=076C ES=075C SS=076F CS=0771 IP=0000
0771:004A 80FC09 CMP AH,09
-t

AX=0403 BX=000F CX=0001 DX=0504 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0045 2AE6 SUB AH,DH
-t

AX=FF03 BX=000F CX=0001 DX=0504 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0047 2AC2 SUB AL,DL
-t

AX=FFFF BX=000F CX=0001 DX=0504 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0049 3F AAS
-t

AX=FE09 BX=000F CX=0001 DX=0504 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:004A 80FC09 CMP AH,09
-t

AX=FE09 BX=000F CX=0001 DX=0504 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:004D 760F JBE 005E
-t

AX=0504 BX=000F CX=0001 DX=0403 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0054 2AE6 SUB AH,DH
-t

AX=0104 BX=000F CX=0001 DX=0403 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0056 2AC2 SUB AL,DL
-t

AX=0101 BX=000F CX=0001 DX=0403 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0058 3F AAS
-t

AX=0101 BX=000F CX=0001 DX=0403 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:0059 C747180100 MOV WORD PTR [DI],SI
-t

AX=0101 BX=000F CX=0001 DX=0403 SP=0001
DS=076C ES=075C SS=076F CS=0771 IP=0004
0771:005E 43 INC BX

```

加法若得到 A~F, 则应该转换为 0~9, 减法若遇到负数则反过来相减, 并加上负号。

2.9 实验九

1. 实验题目:

实现 KMP 算法, 输入两个字符串 (可以直接保存在内存中), 实现快速匹配。

2. 实验原理及解析:

KMP 算法是串的模式匹配算法的一种改进算法, 该算法可以在 $O(n+m)$ 的时间数量级上

完成串的模式匹配操作。其改进在于：每一趟匹配过程中出现字符比较不等时，不需要回溯i指针，而是利用已经得到的“部分匹配”的结果将模式向右“滑动”尽可能远的一段距离后，继续进行比较。

将串和模式串和next都存储在data中，方便后续代码编写，程序使用和数据存取。

next函数的C语言实现：

```
while(j<strlen(p)-1)
{
    if(j==0||T[i-1]==T[j-1])
    {
        i++;
        j++;
        next[i]=j;
    }
    else
        j=next[j];
}
```

KMP算法的C语言实现：

```
while (i < S.length && j < T.length)
{
    if (j == 0 || S[i-1] == P[j-1])
    {
        i++;
        j++;
    }
    else
    {
        // 匹配失败时，跳转下标
        j = next[j];
    }
}
// 匹配成功
if (j == P.length)
{
    return i - j;
}
```

3. 代码实现：

```
data segment
    s db 'ababcabcacbab'
    t db 'abcac'
;要求模式串不超过 10 个字符
```



```
n db 10 dup(0)
data ends

stack segment
    db 16 dup(0)
stack ends

code segment
    ;实现 KMP 算法, 输入两个字符串 (可以直接保存在内存中), 实现快速匹配
start:
    mov ax,data
    mov ds,ax           ;初始化数据段寄存器
    mov ax,stack
    mov ss,ax
    mov sp,16           ;初始化栈
    jmp main
;求 next 数组, 存放在数据段的数据标号 n 处
; si 相当于 i
; di 相当于 j
; ax 当前 next[]
next:
    mov si,1
    mov di,0
    mov cx,offset n - offset t    ;求 next 数组长度,即匹配串 t 的长度
    dec cx                        ;cx--
    mov ah,00h
l1:
    cmp di,0                    ;j==0
    je i1                       ;就跳转到 i1
    dec si                      ;i-1
    dec di                      ;j-1
    mov al,t[si]                ;得到 T[i-1]
    mov bl,t[di]                ;得到 T[j-1]
    inc si                      ;恢复 si,di,以便后面继续访问
    inc di
    cmp al,bl                   ;如果 T[i-1]!=T[j-1]
    jne e1                      ;就跳转到 e1
i1:
    inc si                      ;i++
    inc di                      ;j++
    mov ax,di                  ;next[i]=j
    mov n[si],al
    jmp l2
```

```
e1:
    mov al,n[di]
    mov di,ax                ;else j=next[j]
    ;i 不动, j 变为当前测试字符串的 next 值
    inc cx
12:
    loop l1
    ret

;kmp 算法主函数
main:
    call next                ;先计算 next 数组
    mov si,1
    mov di,1
    mov ah,00h
    ;循环判断是否超出长度
13:
    mov cx,offset t - offset s    ;cx 存放 S 长度
    cmp si,cx                    ;i<S.length
    ja o1                        ;不满足就跳转到 o1
    mov cx,offset n - offset t    ;cx 存放 T 长度
    cmp di,cx                    ;j<T.length
    ja o1                        ;不满足就跳转到 o1

    cmp di,0                    ;比较 j==0
    je i2                        ;j=0,跳转
    dec si                      ;i-1
    dec di                      ;j-1
    mov al,s[si]                ;S[i-1]
    mov bl,t[di]                ;T[i-1]
    inc si                      ;恢复 i,j
    inc di
    cmp al,bl                    ;S[i-1]==T[j-1]
    jne e2                      ;不等于就跳转到 e2
i2:
    inc si                      ;i++
    inc di                      ;j++
    jmp l4                      ;下一次循环
e2:
    mov al,n[di]                ;j 变为当前测试字符串的 next 值
    mov di,ax                    ;j=next[j]
14:
    loop l3                      ;循环
```

;退出循环，输出结果

o1:

```
mov cx,offset n - offset t    ;模式串长度
cmp di,cx                    ;j==T.length
ja o2                        ;di>cx,匹配成功,跳转 o2
mov dl,23H                   ;未找到匹配字符串，显示#
jmp o3
```

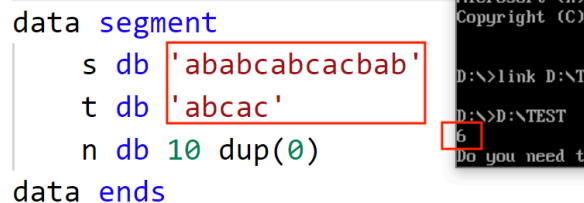
o2:

```
sub si,cx                    ;i-T.length，找到匹配字符串，显示匹配位置
add si,30H                   ;加上 30H 使得 si 变为对应数值的 ASCII 码
mov dx,si                    ;送至 dx 中输出
```

o3:

```
mov ah,2
int 21H                      ;显示输出,DL 为输出字符
```

4. 结果展示:



```
data segment
    s db 'ababcabcacbab'
    t db 'abcac'
    n db 10 dup(0)
data ends
```

输出模式串在字符串中的起始位置。

2.10 实验十

1. 实验题目:

斐波纳契数列: 1, 1, 2, 3, 5, 8, 13。通常可以使用递归函数实现，现用汇编实现该过程。

2. 实验原理及解析:

斐波那契数列满足: $f(n)=f(n+1)+f(n+2)$ ，在此我们使用loop循环来实现该操作，并将结果保存在内存中。

斐波那契数列的C语言实现:

```
int Fibo(int n)
{
    if (n == 1 || n == 2)
    {
        return 1;
    }
    else
```

```
{  
    return Fibo(n - 1) + Fibo(n - 2);  
}  
}
```

3. 代码实现:

```
data segment  
    a dw 0001h,0001h,0000h      ;f(n-1),f(n-2),结果  
    n dw 7  
data ends
```

main:

```
mov ax,data  
mov ds,ax  
mov si,0      ;数组下标  
mov bx,a[si]  ;bx 存放 f(n-1)  
mov dx,a[si+2] ;dx 存放 f(n-2)  
  
mov ax,n      ;ax 存放 n  
; sub ax,1  
cmp ax,1      ;比较 n 与 1  
; jz get_out1  
jna get_out1   ;小于等于就跳转至 get_out1  
cmp ax,2      ;比较 n 与 2  
je get_out2    ;小于等于就跳转至 get_out2  
  
;大于 2  
mov cl,al  
sub cl,2
```

fun:

```
mov ax,0      ;f(n)值从零开始  
add ax,bx     ;f(n)=f(n-1)+f(n-2)  
add ax,dx     ;f(n)=f(n-1)+f(n-2)  
mov dx,bx     ;f(n-2)=f(n-1)  
mov bx,ax     ;f(n-1)=f(n)  
mov a[si],bx  ;更新 data 中的值  
mov a[si+2],dx ;更新 data 中的值  
loop fun      ;循环调用 fun  
  
mov a[si+4],ax ;保存结果  
jmp quit
```

```
get_out1:
    mov a[si+4],ax      ;直接将 ax 放入结果单元中
    jmp quit
get_out2:
    mov ax,1            ;n==2,结果为 1
    mov a[si+4],ax
    jmp quit

quit:
    ;此时结果存放在 a[si+4]中
    mov dx,a[si+4]
    call Num2Str
    mov dx,bx
    mov AH,09H
    int 21H

    mov ax,4C00H
    int 21H

;-----数字转字符串的子程序-----
;数字在 dx,返回结果中 buffer 开始于 bx,且自动加$,长度保存在 cx
Num2Str proc near
    push ax
    push dx
    push bx
    mov ax,dx
    mov cx,0h
    _Div10:
        mov dl,10d      ;除以 10
        div dl           ;除 10 获取个位数和十位数
        add ah,30h       ;加 30H 转换位 ASCII 码
        mov [bx],ah      ;将转换的字符串存入 buffer
        mov ah,0h        ;高位清零
        inc bx           ;下一个字符
        inc cx           ;length++
        cmp ax,0
        jnz _Div10       ;非零就继续除以 10,直到 ax 变为零
        mov al,'$'       ;添加 '$'
        mov [bx],al
        pop bx           ;恢复寄存器值
        pop dx
        pop ax
```

```
call RevStr          ;调用倒置字符串的子程序,先输出高位
ret
Num2Str endp

;-----倒置字符串的子程序-----
;个数为 cx,开头在 bx;返回结果 buffer 开始于 bx
RevStr proc near
    push bx
    push cx
    push ax
    push dx
    push si           ;保存寄存器
    mov ax,bx         ;ax 为 buffer 的起始地址
    add ax,cx          ;加上长度,变为最后一个字符的下一个地址
    dec ax             ;length-1,
    mov si,ax          ;si 保存最后一个字符
;开始倒置
_OnRev:
    push [bx]         ;保存 bx 指向的字符
    mov al,[si]        ;交换
    mov [bx],al
    pop ax             ;恢复 bx 指向的字符到 ax 中
    mov [si],al
    inc bx             ;i++
    dec si             ;j--
    cmp bx,si          ;比较 i 和 j 是否相遇
    jb _OnRev
    pop si             ;恢复寄存器状态
    pop dx
    pop ax
    pop cx
    pop bx
    ret
RevStr endp
```

4. 结果展示:

;斐波纳契数列: 1,1,2,3,5,8,13,21,34,55。



结果输出题目上最后一个数据。

2.11 实验的优点和不足

本次实验的优点在于,都使用了汇编语言完成,注释清晰,并尽可能的保证了代码

的可重用性、减少冗余和重复的操作，提高了程序的效率。实验结果正确准确。

本次实验的不足在于，程序没有很大的创新，同时很多结果都需要在 debug 中查看，而没有写专门的输出语句。同时数据都是在程序中固定写死的没有专门的输入语句，动态变换程序中的变量。这样不利于人机交互和使用者动态查看程序的结果，使得使用程序不太方便。未来，我们会继续改进，让程序更加适用、实用。

第3章 心得体会

在实验前，我先花了3天的时候，通过网课学习、阅读课本的方式较为系统的过了一遍汇编知识点。刚开始做实验的时候，前面几个题的难度较易，但是后面的题目难度逐渐上升，我一边看书和翻阅笔记一边写程序和改代码，仔细体会汇编语言的奥秘。

一开始，我的思维还没有完全从高级语言中转换过来，看到题目，脑海中一下就蹦出了C和C++的程序和算法，但是却不知如何用汇编语言和寄存器来实现。汇编和高级语言略有相似性，但是却有很大的不同。让我体会最深的是汇编中寄存器和栈的运用。在高级语言中，你可以随意设置很多的变量来存放你的数据，但是在汇编中，你只有几个有限的寄存器，你需要栈和内存的配合，才能合理的组织你的程序。同时，在高级语言中，我们不需要过多的干预数据的存储以及考虑不同位数的数据的存储和运算，同时在数据运算中，不用人工处理数据的进位和溢出等，但是在汇编中，这些都是我们需要仔细考虑的。在完成了实验后，我对汇编程序的书写和组织有了进一步的了解，也更熟练汇编程序运行的过程和步骤，对数据在计算机中的存储也有了进一步的了解和认识。

通过大二下学期这半年的学习，我进一步认识到在大学中应通过科学研究的经历来学习和塑造品格，形成对生活的态度，学以成人；掌握专业知识，形成科学与理性的观点，探索真理和批判性思考的习惯，在思想上独立。未来我也会继续努力学习计算机相关的知识，并运用汇编语言进行更多的开发、设计和应用。

参考文献

- [1] 王爽. 汇编语言-第 2 版[M]. 清华大学出版社, 2008.
- [2] 沈美明, 温冬婵. IBM-PC 汇编语言程序设计[M]. 清华大学出版社, 1991.
- [3] 王元珍, 曹忠升, 韩宗芬. 80X86汇编语言程序设计[M]. 华中科技大学出版社, 2005.
- [4] 严蔚敏, 吴伟民. 数据结构 (C语言版) [M]. 清华大学出版社, 2018.