
微机原理与汇编语言实验

指导书(一)汇编部分

xx 编

系 : 计算机科学与技术
班级 :
学号 :
姓名 : Banban

xx 大学计算机学院

二〇二二 年 十一 月

注意不要雷同

banban

<https://github.com/dream4789/Computer-learning-resources.git>

目 录

实验一	汇编运行环境及方法.....	1
实验二	寻址方式及指令.....	15
实验三	用查表的方法将一位十六进制数转换成与它相应的 ASCII 码	17
实验四	将键盘输入的小写字母用大写字母显示出来	19
实验五	分支程序设计.....	21
实验六	循环程序设计.....	23
实验七	按学号顺序把学生名次从终端上显示出来	2
实验八	统计不同成绩段学生的人数.....	1
附录	实验报告.....	1

实验一 汇编运行环境及方法

一、实验目的

1. 熟悉汇编语言运行环境和方法
2. 了解如何使用汇编语言编制程序
3. 熟悉 DEBUG 有关命令的使用方法
4. 利用 DEBUG 掌握有关指令的功能
5. 利用 DEBUG 运行简单的程序段

二、实验内容

1. 学会输入、编辑汇编语言程序
2. 学会对汇编语言程序进行汇编、连接和运行
3. 进入和退出 DEBUG 程序
4. 学会 DEBUG 中的 D 命令、E 命令、R 命令、T 命令、A 命令、G 命令等的使用。
对于 U 命令、N 命令、W 命令等，也应试一下。

三、实验准备

1. 仔细阅读有关汇编语言环境的内容，事先准备好使用的例子。
2. 准备好源程序清单、设计好调试步骤、测试方法、对运行结果的分析。
3. 编写一个程序：比较 2 个字符串所含的字符是否相同。若相同则显示 ' Match. '，否则显示 ' No match! '；仔细阅读有关 DEBUG 命令的内容，对有关命令，都要事先准备好使用的例子。

四、实验步骤

1. 在 DOS 提示符下，进入 MASM 目录。
2. 在 MASM 目录下启动 EDIT 编辑程序，输入源程序，并对其进行汇编、连接和运行。
 - 1) 调用 edit 输入、编辑源程序并保存在指定的目录中:例:
 > edit abc.asm
 - 2) 用汇编程序 masm 对源程序汇编产生目标文件.obj:例:
 > masm abc;
 不断修改错误，直至汇编通过为止。
 - 3) 用连接程序 link 产生执行文件.exe:例:
 > link abc;
 - 4) 执行程序
 可直接从 DOS 执行程序，即在 DOS 环境中，输入文件名即可。
3. 详细记录每一步所用的命令，以及查看结果的方法和具体结果。

五、实验方法

有关汇编语言程序的上机过程请读者参阅清华大学出版社 1991 年出版的《IBMPC 汇编语言程序设计》的 4.4 节。在这里，我们举例简要说明该过程以及程序的调试方法。

例1.1 比较字符串 sample

试编写一程序：比较两个字符串 string1 和 string2 所含的字符是否相同。若相同则显示'Match'，否则，显示'Nomatch'。

我们可以用串比较指令来完成程序所要求的功能。上机过程如下：

1. 调用字处理程序 wordstar 建立.asm 文件，当然也可以用其它编辑程序如 pced 或行编辑程序 edlin 来建立源文件。

```
*****
;C>WS
;使用非文本文件方式(n 命令)建立以 sample.asm 为文件名的源文件如图 1.1 所示
;然后用 CTRL K X 命令将文件存入磁盘，并使系统返回 DOS。
;PROGRAM TITLE GOES HERE--Compare string
*****
dataarea segment           ;define data segment
    string1 db 'Move the cursor backward.'
    string2 db 'Move the cursor backward.'
    ;
    mess1    db 'Match.',13,10,'$'
    mess2    db 'No match!',13,10,'$'
dataarea ends
*****
prognam segment           ;define code segment
main proc far
    assume cs:prognam, ds:dataarea, es:dataarea
start:                       ;starting execution address
                                ;set up stack for return
    push     ds               ;save old data segment
    sub      ax,ax           ;put zero in AX
    push     ax              ;save it on stack
                                ;set DS register to current data segment
    mov      ax,dataarea     ;dataarea segment addr
    mov      ds,ax           ;into DS register
    mov      es,ax           ;into ES register
                                ;MAIN PART OF PROGRAM GOES HERE
    lea      si,string1
    lea      di,string2
    cld
    mov      cx,25
    repz     cmpsb
```

```

        jz      match
        lea     dx, mess2
        jmp     short disp
match:
        lea     dx, mess1
disp:
        mov     ah, 09
        int     21h
        ret     ;return to DOS
main endp      ;end of main part of program
program ends   ;end of code segment
;*****
end start      ;end assembly

```

图 1.1 例 1 的原文件 sample.asm

2. 用汇编程序 masm(或 asm)对源文件汇编产生目标文件 obj

```

C:\>masm sample;
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

```

51520 + 423584 Bytes symbol space free

```

0 Warning Errors
0 Severe Errors

```

3. 用连接程序 link 产生执行文件 exe

```

C:\>link sample;
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

```

LINK : warning L4021: no stack segment

4. 执行程序

```

C:\>sample
Match.

```

终端上已显示出程序的运行结果。为了调试程序的另一部分，可重新进编辑程序修改两个字符串的内容，使它们互不相同。如修改后的数据区为：

```

;*****
dataarea segment ;define data segment
        string1 db 'Move the cursor backward.'
        string2 db 'Move the cursor forward.'
;
        mess1 db 'Match.', 13, 10, '$'
        mess2 db 'No match!', 13, 10, '$'

```

```
dataarea ends
;*****
```

然后，重新汇编、连接、执行，结果为：

```
C:\>sample
No match!
```

至此，程序已调试完毕，运行结果正确。

另一种调试程序的方法是使用 debug 程序。可调用如下：

```
C>debug sample.exe
```

```
-
```

此时，debug 已将执行程序装入内存，可直接用 -g 命令运行程序：

```
-g
Match.
Program terminated normally
```

为调试程序的另一部分，可在 debug 中修改字符串内容。可先用 -u 命令显示程序，以便了

解指令地址。显示结果如图 1.2 所示。

```
-u
0D36:0000 1E          PUSH    DS
0D36:0001 2BC0          SUB     AX, AX
0D36:0003 50          PUSH    AX
0D36:0004 B8310D       MOV     AX, 0D31
0D36:0007 8ED8       MOV     DS, AX
0D36:0009 8EC0       MOV     ES, AX
0D36:000B 8D360000    LEA     SI, [0000]
0D36:000F 8D3E1900    LEA     DI, [0019]
0D36:0013 FC          CLD
0D36:0014 B91900       MOV     CX, 0019
0D36:0017 F3          REPZ
0D36:0018 A6          CMPSB
0D36:0019 7406       JZ      0021
0D36:001B 8D163B00    LEA     DX, [003B]
0D36:001F EB04       JMP     0025
0D36:0021 8D163200    LEA     DX, [0032]
0D36:0025 B409       MOV     AH, 09
0D36:0027 CD21       INT     21
0D36:0029 CB          RETF
0D36:002A 8A4608       MOV     AL, [BP+08]
0D36:002D 98          CBW
```

```

0D36:002E 50          PUSH    AX
0D36:002F 8B4604      MOV     AX, [BP+04]
0D36:0032 03C6          ADD     AX, SI
0D36:0034 50          PUSH    AX
0D36:0035 E858FF      CALL   FF90
0D36:0038 83C406      ADD     SP, +06
0D36:003B 8BF8      MOV     DI, AX
0D36:003D 83FFFF      CMP     DI, -01
0D36:0040 750C      JNZ     004E

```

图 1.2 例 1.1 用 debug 调试时，-u 命令的显示情况将端点设置在程序的主要部分以前。

```

-g0b
AX=0D31 BX=0000 CX=007A DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=000B NV UP EI PL ZR NA PE NC
0D36:000B 8D360000      LEA     SI, [0000]          DS:0000=6F4D

```

根据其中指示的 ds 寄存器内容查看数据段的情况如下：

```

-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20  Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68  backward.Move th
0D31:0020  65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72  e cursor backwar
0D31:0030  64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61  d.Match...$No ma
0D31:0040  74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00  tch!...$.
0D31:0050  1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D  .+.P.1.....6...
0D31:0060  3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB  >.....t...;...
0D31:0070  04 8D 16 32 00 B4 09 CD-21 CB 8A 46 08 98 50 8B  ...2....!..F..P.

```

可用 -e 命令修改数据区的字符串：（这个方法是一个一个替换，下面有一次性替换的方法）

```

-e29
0D31:0029  62.66  61.6f  63.72  6B.77  77.61  61.72  72.64
0D31:0030  64.2e  2E.20

```

再次用 -d 命令查看修改结果：

```

-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20  Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68  backward.Move th
0D31:0020  65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64  e cursor forward
0D31:0030  2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61  . Match...$No ma
0D31:0040  74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00  tch!...$.
0D31:0050  1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D  .+.P.1.....6...
0D31:0060  3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB  >.....t...;...
0D31:0070  04 8D 16 32 00 B4 09 CD-21 CB 8A 46 08 98 50 8B  ...2....!..F..P.

```

用 -g 命令运行程序，结果为：

```
-g
```

No match!
Program terminated normally

用 -q 命令退出 debug:
-q

至此，程序已调试完毕。

为了进一步数名 debug 命令的使用方法，我们再次重复上述程序的调试过程，只是使用 -e、-a 和 -f 命令来修改数据区的内容而已。必须注意，由于在用 debug 调试程序时，只能修改当时有关的内存单元内容，因此重新用 debug 装入程序时，仍是原来在磁盘中的内容。

操作如下：

C:\>debug sample.exe

-g0b

AX=0D31 BX=0000 CX=007A DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=000B NV UP EI PL ZR NA PE NC
0D36:000B 8D360000 LEA SI,[0000] DS:0000=6F4D

-d0

0D31:0000	4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20	Move the cursor
0D31:0010	62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68	backward.Move th
0D31:0020	65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72	e cursor backwar
0D31:0030	64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61	d.Match...\$No ma
0D31:0040	74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00	tch!...\$......
0D31:0050	1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D	.+.P.1.....6...
0D31:0060	3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB	>.....t...;..
0D31:0070	04 8D 16 32 00 B4 09 CD-21 CB 46 FA 8B 5E FC C1	...2....!.F..^..

-e29 'forward.'20

-d0

0D31:0000	4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20	Move the cursor
0D31:0010	62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68	backward.Move th
0D31:0020	65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64	e cursor forward
0D31:0030	2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61	. Match...\$No ma
0D31:0040	74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00	tch!...\$......
0D31:0050	1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D	.+.P.1.....6...
0D31:0060	3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB	>.....t...;..
0D31:0070	04 8D 16 32 00 B4 09 CD-21 CB 46 FA 8B 5E FC C1	...2....!.F..^..

-g

No match!
Program terminated normally

可见 -e 命令的方式避免使用 ASCII 码进入，对用户是比较方便的。其中最后一个 20 是空格键的 ASCII 码，以补足原来的内容恢复原状，具体如下：

```
-a0d31:29
0D31:0029 db 'backward.'
0D31:0032

-d0
0D31:0000 4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20 Move the cursor
0D31:0010 62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68 backward.Move th
0D31:0020 65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72 e cursor backwar
0D31:0030 64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61 d.Match...$No ma
0D31:0040 74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00 tch!...$.
0D31:0050 1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D .+.P.1.....6...
0D31:0060 3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB >.....t...;..
0D31:0070 04 8D 16 32 00 B4 09 CD-21 CB 56 FA 89 7E EE 8B ...2....!.V..~..

-g
Match.
```

由于 -a 命令是汇编命令，因此信息是用汇编格式进入的。如果修改的是程序中的语句，方法也是相同的，下面我们还会看到这类的操作。现在再看一下用 -f 命令修改数据区的方法：

```
C:\>debug sample.exe
-g0b
AX=0D31 BX=0000 CX=007A DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=000B NV UP EI PL ZR NA PE NC
0D36:000B 8D360000 LEA SI,[0000] DS:0000=6F4D

-d0
0D31:0000 4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20 Move the cursor
0D31:0010 62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68 backward.Move th
0D31:0020 65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72 e cursor backwar
0D31:0030 64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61 d.Match...$No ma
0D31:0040 74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00 tch!...$.
0D31:0050 1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D .+.P.1.....6...
0D31:0060 3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB >.....t...;..
0D31:0070 04 8D 16 32 00 B4 09 CD-21 CB 56 FA 89 7E EE 8B ...2....!.V..~..
```

-f 命令中的 29 为修改区的首地址，19 表示需要修改的长度为 9 个字节。

```
-f29 L9 'forward.'20
```

```
-g
No match!
Program terminated normally
```

-q

为进一步说明程序的调试过程，现假设程序编制错误：在源文件中把 jz 改为 jnz。

该程序汇编、连接后，调试如下：

C>debug sample.exe

-g

No match!

Program terminated normally

结果是错误的（因源文件中的两个字符是相同的）。为检查程序的错误，将断点设在比较串之后。

执行到 19 单元的地方：

-g19

AX=0D31 BX=0000 CX=0000 DX=0000 SP=FFFC BP=0000 SI=0019 DI=0032

DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0019 NV UP EI PL ZR NA PE NC

0D36:0019 7506 JNZ 0021

此时零标志为 ZR，即 ZF=1，即表示比较结果相等，说明比较结果正确的。

现在可用 -p 命令再执行一条指令以观察指令的转向：

-p

AX=0D31 BX=0000 CX=0000 DX=0000 SP=FFFC BP=0000 SI=0019 DI=0032

DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=001B NV UP EI PL ZR NA PE NC

0D36:001B 8D163B00 LEA DX, [003B] DS:003B=6F4E

为查到 003B 单元的内容，可查数据区如下：

-d0

```
0D31:0000 4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20 Move the cursor
0D31:0010 62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68 backward.Move th
0D31:0020 65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72 e cursor backwar
0D31:0030 64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61 d.Match...$No ma
0D31:0040 74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00 tch!...$.
0D31:0050 1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D .+.P.1.....6...
0D31:0060 3E 19 00 FC B9 19 00 F3-A6 75 06 8D 16 3B 00 EB >.....u...;..
0D31:0070 04 8D 16 32 00 B4 09 CD-21 CB 46 FA 8B 5E FC C1 ...2....!.F..^..
```

可见 003B 单元的内容为 4E，即 N 的 ASCII 码，后面跟的是 No match!，这说明指令使用错误，应该改为 jz。可用 -a 命令修改，再应用 -u 命令检查修改结果。运行结果说明程序修改正确。

-a19

0D36:0019 jz 0021

0D36:001B

-u0

0D36:0000 1E PUSH DS

0D36:0001 2BC0 SUB AX, AX

```

0D36:0003 50          PUSH    AX
0D36:0004 B8310D      MOV     AX, 0D31
0D36:0007 8ED8      MOV     DS, AX
0D36:0009 8EC0      MOV     ES, AX
0D36:000B 8D360000    LEA     SI, [0000]
0D36:000F 8D3E1900    LEA     DI, [0019]
0D36:0013 FC          CLD
0D36:0014 B91900      MOV     CX, 0019
0D36:0017 F3          REPZ
0D36:0018 A6          CMPSB
0D36:0019 7406      JZ      0021
0D36:001B 8D163B00    LEA     DX, [003B]
0D36:001F EB04      JMP     0025

```

```

-rip
IP 001B
:

```

```
-q
```

在这里应该注意，在使用 `-a` 命令修改数据区时，必须给出数据段的地址，而在修改程序区时由于 `-a` 命令的缺省段位代码段，所以直接给出偏移地址就可以了。

在调试过程中，也可以用 `-t` 命令逐条跟踪程序的执行。下面列出断点停在 0b 后，用 `-f` 命令修改数据区中字符串的内容，然后逐条执行指令的情况。

```
-f29 19 'forward.'20
```

```
-d0
```

```

0D31:0000 4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20  Move the cursor
0D31:0010 62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68  backward.Move th
0D31:0020 65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64  e cursor forward
0D31:0030 2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61  . Match...$No ma
0D31:0040 74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00  tch!...$.
0D31:0050 1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D  .+.P.1.....6...
0D31:0060 3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB  >.....t...;..
0D31:0070 04 8D 16 32 00 B4 09 CD-21 CB 8A 46 08 98 50 8B  ...2....!..F..P.

```

```
-t
```

```

AX=0D31 BX=0000 CX=007A DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=000F NV UP EI PL ZR NA PE NC
0D36:000F 8D3E1900      LEA     DI, [0019]          DS:0019=6F4D

```

```
-t
```

```

AX=0D31 BX=0000 CX=007A DX=0000 SP=FFFC BP=0000 SI=0000 DI=0019
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0013 NV UP EI PL ZR NA PE NC
0D36:0013 FC          CLD

```

-t

AX=0D31 BX=0000 CX=007A DX=0000 SP=FFFC BP=0000 SI=0000 DI=0019
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0014 NV UP EI PL ZR NA PE NC
0D36:0014 B91900 MOV CX, 0019

进入比较 REPZ cmpsb, IP 停止计数:

-t

AX=0D31 BX=0000 CX=0019 DX=0000 SP=FFFC BP=0000 SI=0000 DI=0019
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
0D36:0017 F3 REPZ
0D36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0018 DX=0000 SP=FFFC BP=0000 SI=0001 DI=001A
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
0D36:0017 F3 REPZ
0D36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0017 DX=0000 SP=FFFC BP=0000 SI=0002 DI=001B
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
0D36:0017 F3 REPZ
0D36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0016 DX=0000 SP=FFFC BP=0000 SI=0003 DI=001C
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
0D36:0017 F3 REPZ
0D36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0015 DX=0000 SP=FFFC BP=0000 SI=0004 DI=001D
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
0D36:0017 F3 REPZ
0D36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0014 DX=0000 SP=FFFC BP=0000 SI=0005 DI=001E
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
0D36:0017 F3 REPZ
0D36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0013 DX=0000 SP=FFFC BP=0000 SI=0006 DI=001F

DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0012 DX=0000 SP=FFFC BP=0000 SI=0007 DI=0020
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0011 DX=0000 SP=FFFC BP=0000 SI=0008 DI=0021
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0010 DX=0000 SP=FFFC BP=0000 SI=0009 DI=0022
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=000F DX=0000 SP=FFFC BP=0000 SI=000A DI=0023
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=000E DX=0000 SP=FFFC BP=0000 SI=000B DI=0024
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=000D DX=0000 SP=FFFC BP=0000 SI=000C DI=0025
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=000C DX=0000 SP=FFFC BP=0000 SI=000D DI=0026
 DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
 OD36:0017 F3 REPZ
 OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=000B DX=0000 SP=FFFC BP=0000 SI=000E DI=0027
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
OD36:0017 F3 REPZ
OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=000A DX=0000 SP=FFFC BP=0000 SI=000F DI=0028
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
OD36:0017 F3 REPZ
OD36:0018 A6 CMPSB

-t

AX=0D31 BX=0000 CX=0009 DX=0000 SP=FFFC BP=0000 SI=0010 DI=0029
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0017 NV UP EI PL ZR NA PE NC
OD36:0017 F3 REPZ
OD36:0018 A6 CMPSB

退出比较 REPZ cmpsb, IP 继续计数:

-t

AX=0D31 BX=0000 CX=0008 DX=0000 SP=FFFC BP=0000 SI=0011 DI=002A
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0019 NV UP EI NG NZ AC PE CY
OD36:0019 7406 JZ 0021

-t

AX=0D31 BX=0000 CX=0008 DX=0000 SP=FFFC BP=0000 SI=0011 DI=002A
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=001B NV UP EI NG NZ AC PE CY
OD36:001B 8D163B00 LEA DX, [003B] DS:003B=6F4E

-t

AX=0D31 BX=0000 CX=0008 DX=003B SP=FFFC BP=0000 SI=0011 DI=002A
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=001F NV UP EI NG NZ AC PE CY
OD36:001F EB04 JMP 0025

-t

AX=0D31 BX=0000 CX=0008 DX=003B SP=FFFC BP=0000 SI=0011 DI=002A
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0025 NV UP EI NG NZ AC PE CY
OD36:0025 B409 MOV AH, 09

-t

AX=0931 BX=0000 CX=0008 DX=003B SP=FFFC BP=0000 SI=0011 DI=002A
DS=0D31 ES=0D31 SS=0D31 CS=0D36 IP=0027 NV UP EI NG NZ AC PE CY
OD36:0027 CD21 INT 21

-g

No match!
Program terminated normally

从这一过程可清楚地看出每次比较的结果，一旦比较不等，则立即从串指令退出，执行下面的指令。应该注意，如果遇到系统功能调用，则不能再使用 `-t` 或 `-p` 命令跟踪，而应该用断点停在功能调用完成之后，然后再接着跟踪。本例中，由于不需要再跟踪，所以直接用 `-g` 命令运行到程序结束。

前面已经提到，debug 调试期间所修改的数据段或代码段的内容只是修改了内存中的内容，而磁盘文件中的内容并未修改。如果你的执行文件是 .com，则可在 debug 中用 `-n`、`-w` 命令直接把经修改后的内存单元中的内容存入磁盘，但是 .exe 文件则不允许这样做，因此，应该重新进入编辑程序，根据调试结果把源文件修改正确，经汇编、连接、执行检查正确后再保存下来。

六、实验报告要求

1. 程序流程图和源程序清单。
2. 如何启动和退出 EDIT 程序。
在命令行中直接输入 `edit` 启动 `edit` 文本编辑器
3. 如何对源程序进行汇编及编辑。
`-a` 编辑汇编指令
`-f` 编辑数据
4. 如何启动和退出 DEBUG 程序。
启动：输入 `debug`
退出：输入 `-q`
5. 整理每个 DEBUG 命令使用的方法，实际示例及执行结果。

分类	命令格式	功能简介
读写寄存器	<code>-R</code>	显示所有寄存器的当前内容
	<code>-R 寄存器名</code>	显示和修改指定寄存器内容
	<code>-RF</code>	显示和修改标志寄存器内容
汇编和反汇编	<code>-A[内存地址]</code>	从指定地址开始汇编指令（默认代码段）
	<code>-U[内存块]</code>	对指定内存块进行反汇编（默认代码段）
执行指令	<code>-T[: 内存地址][条数]</code>	单步或多步执行指令（会进入子程序，IP+1）
	<code>-P[=内存地址][条数]</code>	单步或多步执行指令（不会进入子程序，IP+1）
	<code>-G[=内存地址]</code>	连续执行指令（执行到某个 IP 单元）
	<code>-G[=内存地址]断点地址</code>	设断点执行程序

分类	命令格式	功能简介
读写内存	-D[内存块]	显示指定内存块内容（默认数据段）
	-E 内存地址 字符或数值串	修改指定内存内容
	-F 内存块 字符或数值串	填充指定内存块（默认数据段）
	-S 内存块 字符串或数值	在指定内存块中查找串
	-M 内存块 1 内存块 2 的首地址	复制内存块内容
	-C 内存块 1 内存块 2 的首地址	比较两个指定内存块
读写磁盘	-N[d:][path]文件名.扩展名	指定欲读写的磁盘文件
	-W 内存地址	将指定内存块写入文件
	-L [内存地址]	将文件调入内存
读写 I/O 端口	-I 端口地址	读入指定端口的内容
	-O 端口地址 数值	将数据写入指定端口
十六进制加减	-H 数值 1 数值 2	计算并显示两数之和，两数之差
退出 DEBUG	-Q	退出 DEBUG，返回 DOS

6. 启动 DEBUG 后，要装入某一个 .EXE 文件，应通过什么方法实现？

> debug 文件名

-l

或：

> debug

-n 文件名

-l

实验二 寻址方式及指令

一、实验目的：

1. 熟练掌握 DEBUG 的常用命令，学会用 DEBUG 调试程序。
2. 掌握数据在内存中的存放方式和内存操作数的几种寻址方式。
3. 掌握简单指令的执行过程。

二、实验内容：

1. 设堆栈指针 SP=2000H，AX=3000H，BX=5000H；请编一程序段将 AX 和 BX 的内容进行交换。请用堆栈作为两寄存器交换内容的中间存储单元，用 DEBUG 调试程序进行汇编与调试。
2. 设 DS=当前段地址，BX=0300H，SI=0002H；请用 DEBUG 的命令将存储器偏移地址 300H~304H 连续单元顺序装入 0AH，0BH，0CH，0DH，0EH。在 DEBUG 状态下送入下面程序，并用单步执行的方法，分析每条指令源地址的形成过程，当数据传送完毕时，AX 中的内容是什么。

程序清单如下：

```
MOV AX, BX
MOV AX, 0304H
MOV AX, [0304H]
MOV AX, [BX]
MOV AX, 0001[BX]
MOV AX, [BX][SI]
MOV AX, 0001[BX][SI]
HLT
```

三、实验要求：

1. 实验前要做好充分准备，包括汇编程序清单、调试步骤、调试方法，以及对程序结果的分析等。
2. 本实验只要求在 DEBUG 调试程序状态下进行，包括汇编程序、调试程序和执行程序。

四、实验报告：

1. 程序说明。说明程序的功能、结构。
2. 调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是怎样解决的，并对调试过程中的问题进行分析，对执行结果进行分析。

没有问题。。。。

3. 写出源程序清单和执行结果。

一问源程序清单：

```
stack segment stack
```

```

        db    200 dup(0)
stack ends
code segment
        assume cs:code,ss:stack
main:
        MOV SP,2000H
        MOV AX,3000H
        MOV BX,5000H
        MOV SS,AX
        PUSH AX
        PUSH BX
        POP  AX
        POP  BX
        MOV AX,4C00H
        INT 21H
code ends
        end main

```

二问源程序清单:

```

stack segment stack
        db 200 dup(0)
stack ends
code segment
        assume cs:code,ss:stack
main:
        MOV BX,0300H
        MOV SI,0002H
        MOV AX,BX           ;ax==0300H
        MOV AX,0304H        ;ax==0304H
        MOV AX,[0304H]
        MOV AX,[BX]         ;ax==0B0AH
        MOV AX,0001[BX]     ;ax==0C0BH
        MOV AX,[BX][SI]     ;ax==0D0CH
        MOV AX,0001[BX][SI] ;ax==0E0DH
        HLT                 ;ax==0E0DH
        MOV AX,4C00H
        INT 21H
code ends
        end main

```

运行结果:



实验三 用查表的方法将一位十六进制数转换成与它相应的 ASCII 码

一、实验目的：

1. 熟练掌握编写汇编语言原程序的基本方法和基本框架。
2. 掌握查表法和查表指令 XLAT。
3. 熟练使用 DEBUG 调试程序。

二、实验内容：

用查表的方法将一位十六进制数转换成与它相应的 ASCII 码，并将结果存放到 ASCII 单元中。

三、编程提示：

既然指定用查表的方法，那么首先要建立一个表 TABLE。我们在表中按照十六进制数从小到大的顺序放入他们对应的 ASCII 码值。

```
DATA    SEGMENT

TABLE   DB  30H, 31H, 32H, 33H, 34H, 35H, 36H, 37H
        DB  38H, 39H, 41H, 42H, 43H, 44H, 45H, 46H

HEX     DB  X    ;X 为待转换的十六进制数

ASCII   DB  ?    ;存放转换后的 ASCII 码

DATA    ENDS
```

四、实验要求：

实验前要做好充分准备，包括汇编程序清单、调试步骤、调试方法，以及对程序结果的分析等

五、实验报告：

1. 程序说明。说明程序的功能、结构。
2. 调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是怎样解决的，并对调试过程中的问题进行分析，对执行结果进行分析。
没问题。。。
3. 写出源程序清单和执行结果。

源程序清单：

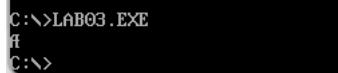
```
stack segment stack
    db 200 dup(0)
stack ends
```

```

DATA SEGMENT
    TABLE DB 30H, 31H, 32H, 33H
            DB 34H, 35H, 36H, 37H
            DB 38H, 39H, 41H, 42H
            DB 43H, 44H, 45H, 46H
    HEX DB 10
    ASCI DB ?
DATA ENDS
code segment
    assume cs:code,ds:data,ss:stack
main:
    MOV AX, data
    MOV DS, AX
    LEA BX, table
    MOV AL, HEX
    XLAT
    MOV asci, AL
    MOV AH, 2
    INT 21H
    MOV AH, 4CH
    INT 21h
code ends
    end main

```

执行结果:



```

C:\>LAB03.EXE
C:\>

```

实验四 将键盘输入的小写字母用大写字母显示出来

一、实验目的：

1. 掌握接受键盘数据的方法，并了解将键盘数据显示时，须转为 ASCII 码的原理。
2. 掌握 DOS 功能调用的编程方法。

二、实验内容：

试编写一个汇编语言程序，要求对键盘输入的小写字母用大写字母显示出来。

三、编程提示：

利用 DOS 功能调用 INT 21H 的 1 号功能从键盘输入字符和 2 号功能在显示器上显示一个字符。

四、实验要求：

实验前要做好充分准备，包括汇编程序清单、调试步骤、调试方法，以及对程序结果的分析等。

五、实验报告：

1. 程序说明。说明程序的功能、结构。
2. 调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是怎样解决的，并对调试过程中的问题进行分析，对执行结果进行分析。
没问题。。。
3. 写出源程序清单和执行结果。

源程序清单：

```
CODE SEGMENT
    ASSUME CS:CODE
START:
    MOV AH, 01H
    INT 21H
    SUB AL, 20H
    MOV DL, AL
    MOV AH, 02H
    INT 21H
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END START
```

执行结果：

```
C:\>LAB04-F.EXE
aA
```

```
C:\>LAB04-F.EXE
aA
C:\>debug LAB04-F.EXE
-u 0
076A:0000 B401      MOV     AH,01
076A:0002 CD21      INT     21
076A:0004 2C20      SUB     AL,20
076A:0006 8AD0      MOV     DL,AL
076A:0008 B402      MOV     AH,02
076A:000A CD21      INT     21
076A:000C B44C      MOV     AH,4C
076A:000E CD21      INT     21
076A:0010 0D3C3A     OR      AX,3A3C
076A:0013 7409      JZ      001E
076A:0015 FF46FE     INC     WORD PTR [BP-02]
076A:0018 8B5EFE     MOV     BX,[BP-02]
076A:001B C6005C     MOV     BYTE PTR [BX+SI],5C
076A:001E B80900     MOV     AX,0009
```


实验五 分支程序设计

一、实验目的：

1. 掌握分支程序的结构。
2. 掌握分支程序的设计、调试方法。

二、实验内容：

假设有一组数据：5，-4，0，3，100，-51，请编一程序，判断：每个数大于0，等于0，还是小于0；并输出其判断结果。

$$y = \begin{cases} 1 & \text{当 } x > 0 \\ 0 & \text{当 } x = 0 \\ -1 & \text{当 } x < 0 \end{cases}$$

即：

三、实验要求：

实验前要做好充分准备，包括汇编程序清单、调试步骤、调试方法，以及对程序结果的分析等。

四、编程提示：

1. 首先将原始数据装入起始地址为 XX 的字节存储单元中。
2. 将判断结果以字符串的形式存放在数据区中，以便在显示输出时调用。
3. 其中判断部分可采用 CMP 指令，得到一个分支结构，分别输出“y=0”，“y=1”，“y=-1”。
4. 程序中存在一个循环结构，循环 6 次，调用 6 次分支结构后结束。

五、思考题：

程序中的原始数据是以怎样的形式存放在数据区中的？请用 DEBUG 调试程序观察并分析。

六、实验报告：

1. 程序说明。说明程序的功能、结构。
2. 调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是怎样解决的，并对调试过程中的问题进行分析，对执行结果进行分析。
3. 画出程序框图。
4. 写出源程序清单和执行结果。

源程序清单：

```
DATAS SEGMENT
```

```
    X DB 5, -4, 0, 3, 100, -51
```

```
DATA1 DB 'Y= 0', 0DH, 0AH, '$'
```

```
DATA2 DB 'Y= 1', 0DH, 0AH, '$'
```

```
DATA3 DB 'Y=-1', 0DH, 0AH, '$'
```

```

DATAS ENDS

stack segment stack
    ;db 200 dup(0)
stack ends

CODES SEGMENT
    ASSUME CS:CODES, DS:DATAS, SS:stack
START:
    MOV AX,DATAS
    MOV DS,AX
    MOV SI,OFFSET X
    MOV CX,6

CHECK:
    MOV AL,[SI]
    CMP AL,00H
    JE NEXT1
    JG NEXT2
    MOV DI,OFFSET DATA3
    JMP DONE

NEXT1:
    MOV DI,OFFSET DATA1
    JMP DONE

NEXT2:
    MOV DI,OFFSET DATA2

DONE:
    MOV DX,DI
    MOV AH,09H
    INT 21H

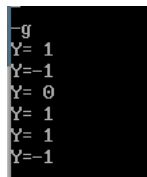
    INC SI
    LOOP CHECK

    MOV AH, 4CH
    INT 21H

CODES ENDS
    END START

```

运行结果:



```

-9
Y= 1
Y=-1
Y= 0
Y= 1
Y= 1
Y=-1

```


实验六 循环程序设计

一、实验目的

1. 加深对循环结构的理解。
2. 掌握循环程序的设计方法。

二、实验内容与要求

1. 编制程序计算 $S=1+2 \cdot 3+3 \cdot 4+4 \cdot 5+\cdots+N(N+1)+\cdots$ 直到 $N(N+1)$ 大于 200 为止，并将结果由屏幕上显示出来。其程序的流程图如图 6.1 所示。
2. 将从 3000H 内存单元开始的 100 个字节存储单元全部清 0。

本实验要求在 DEBUG 调试状态下进行，包括汇编程序、运行程序、检查结果。

三、程序框图

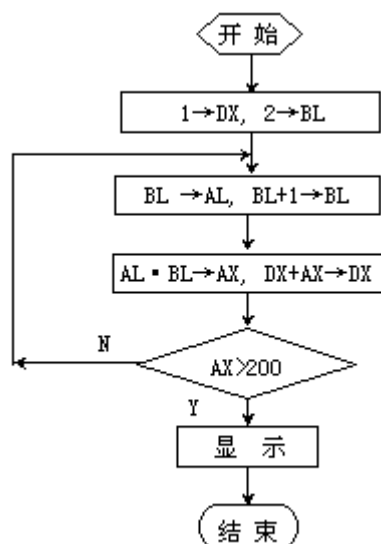


图 6.1 程序框图

四、实验步骤

实验内容一：

- 1) 按程序流程图编制实验程序。
- 2) 输入源程序。
- 3) 汇编、连接程序，执行程序，检查结果。

实验内容二：

- 1) 输入源程序并检查无误。
- 2) 对内存单元 3000H 开始的 100 个存储单元用 -E 命令输入任意数。

```

075A:3000 74 16 83 7E FA 00 74 10-8B 5E FA D1 E3 8B 36 C0 t...t...^....6.
075A:3010 30 A1 14 42 39 00 76 12-C7 06 0A 3C 00 00 C7 06 0..B9.v....<....
075A:3020 26 3C 00 00 5E 8B E5 5D-C3 90 8B 76 FA D1 E6 8B &<...^...J...v....
075A:3030 1E 88 3F 8B 00 03 46 FC-A3 A2 43 8B 1E C0 30 8B ..?...F...C...0.
075A:3040 00 A3 0A 3C A1 D4 33 A3-26 3C 24 FE 3D A2 00 75 ...<...3.&<$...=.u
075A:3050 33 81 3E DC 40 00 02 76-0E FF 06 22 42 8B DD 07 3.>.e...v..."B...
075A:3060 50 E8 14 1D 83 C4 02 A1-DC 40 05 E6 37 A3 24 3C P.....e...7.$<
075A:3070 FF 36 DC 40 2B C0 50 A1-DC 40 05 E6 37 50 E8 1F .6.e+.P...e...7P..
-e3000
075A:3000 74.00 16.00 83.00 7E.00 FA.00 00.00 74.00 10.00
075A:3008 8B.00 5E.00 FA.00 D1.00 E3.00 8B.00 36.00 C0.00
075A:3010 30.00 A1.00 14.00 42.00 39.00 00.00 76.00 12.00
075A:3018 C7.00 06.00 0A.00 3C.00 00.00 00.00 C7.00 06.00
075A:3020 26.00 3C.00 00.00 00.00 5E.00 8B.00 E5.00 5D.00
075A:3028 C3.00 90.00 8B.00 76.00 FA.00 D1.00 E6.00 8B.00
075A:3030 1E.00 88.00 3F.00 8B.00 00.00 03.00 46.00 FC.00
075A:3038 A3.00 A2.00 43.00 8B.00 1E.00 C0.00 30.00 8B.00
075A:3040 00.00 A3.00 0A.00 3C.00 A1.00 D4.00 33.00 A3.00
075A:3048 26.00 3C.00 24.00 FE.00 3D.00 A2.00 00.00 75.00
075A:3050 33.00 81.00 3E.00 DC.00 40.00 00.00 02.00 76.00
075A:3058 0E.00 FF.00 06.00 22.00 42.00 8B.00 DD.00 07.00
075A:3060 50.00 E8.00 14.00 1D.00

```

3) 程序的执行可用 DEBUG 的-G 命令, 也可用-T 命令单步跟踪执行。

```

AX=0772 BX=0000 CX=0107 DX=0000 SP=0080 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076A CS=0775 IP=0003 NU UP EI PL NZ NA PO NC
0775:0003 8ED8      MOV     DS,AX
-t
AX=0772 BX=0000 CX=0107 DX=0000 SP=0080 BP=0000 SI=0000 DI=0000
DS=0772 ES=075A SS=076A CS=0775 IP=0005 NU UP EI PL NZ NA PO NC
0775:0005 BA0100    MOV     DX,0001
-t
AX=0772 BX=0000 CX=0107 DX=0001 SP=0080 BP=0000 SI=0000 DI=0000
DS=0772 ES=075A SS=076A CS=0775 IP=0008 NU UP EI PL NZ NA PO NC
0775:0008 B302      MOV     BL,02
-t
AX=0772 BX=0002 CX=0107 DX=0001 SP=0080 BP=0000 SI=0000 DI=0000
DS=0772 ES=075A SS=076A CS=0775 IP=000A NU UP EI PL NZ NA PO NC
0775:000A 8AC3      MOV     AL,BL
-t
AX=0702 BX=0002 CX=0107 DX=0001 SP=0080 BP=0000 SI=0000 DI=0000
DS=0772 ES=075A SS=076A CS=0775 IP=000C NU UP EI PL NZ NA PO NC
0775:000C FEC3      INC     BL
-g
S=1+2*3+3*4+4*5+...+N(N+1)+...=1119
Program terminated normally

```

4) 用-D 命令检查执行结果。

```

-d3000
0772:3000 5D C3 A1 B8 3D 48 50 2B-C0 50 E8 93 16 83 C4 04 1...=HP+.P.....
0772:3010 8B 46 EC 3D 05 00 76 1B-BE AA 43 8B 1C FF 04 D1 .F.=...v...C.....
0772:3020 E3 8B 36 F2 31 C7 00 00-00 5E 5F 8B E5 5D C3 90 ..6.1....^...J...
0772:3030 FF 06 2A 3C 8B 46 E8 D1-E8 D1 E8 25 07 00 89 46 ...*<.F.....F
0772:3040 FE 0B C0 74 4F 2B C0 50-B8 01 00 50 8B 5E FA D1 ...t0+.P...P...^..
0772:3050 E3 D1 E3 8B 36 5C 3C FF-70 02 FF 30 E8 31 E8 83 ...6\<.p...0.1..
0772:3060 C4 08 89 46 F8 89 56 FA-C4 5E F8 26 80 7F 04 00 ...F..U...^&....
0772:3070 74 5F 26 80 7F 04 01 75-12 8B 46 F0 8B 56 F2 26 t_&....u..F..U.&

```

五、实验报告

1. 列出源程序。

内容一源程序:

STACK SEGMENT STACK

DW 64 DUP(?)

STACK ENDS

DATA SEGMENT

BUF

DB

'S=1+2*3+3*4+4*5+...+N(N+1)+...=',
\$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:STACK

START:

MOV AX, DATA

MOV DS, AX

MOV DX, 0001H

MOV BL, 02H

A1:

MOV AL, BL

INC BL

MUL BL

ADD DX, AX

CMP AX, 00C8H

JNA A1

MOV AX, DX

CALL pri_str

CALL dec_out

MOV AX, 4C00H

INT 21H

dec_out PROC near

push ax

push bx

push cx

push dx

mov bx, 10

mov cx, 0

yazhan:

mov dx, 0

div bx

push dx

inc cx

cmp ax, 0

jz chuzhan

jmp yazhan

chuzhan:

pop dx

add dl, 30h

mov ah, 2

int 21h

loop chuzhan

pop dx

pop cx

pop bx

pop ax

ret

dec_out ENDP

pri_str PROC

PUSH DX

PUSH AX

MOV DX, OFFSET BUF

MOV AH, 09H

INT 21H

POP AX

POP DX

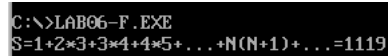
RET

pri_str ENDP

CODE ENDS

END START

运行结果:



```
C:\>LAB06-F.EXE
S=1+2*3+3*4+4*5+...+N(N+1)+...=1119
```

注意不要雷同

banban

<https://github.com/dream4789/Computer-learning-resources.git>

2. 对程序中用到的寄存器说明其功能。

- 数据寄存器

数据寄存器主要用来保存操作数和运算结果等信息,从而节省读取操作数所需占用总线和访问存储器的时间。

32 位 CPU 有 4 个 32 位的通用寄存器 EAX、EBX、ECX 和 EDX。对低 16 位数据的存取,不会影响高 16 位的数据。这些低 16 位寄存器分别命名为:AX、BX、CX 和 DX,它和先前的 CPU 中的寄存器相一致。

4 个 16 位寄存器又可分割成 8 个独立的 8 位寄存器(AX: AH-AL、BX: BH-BL、CX: CH-CL、DX: DH-DL),每个寄存器都有自己的名称,可独立存取。程序员可利用数据寄存器的这种“可分可合”的特性,灵活地处理字/字节的信息。

寄存器 AX 和 AL 通常称为累加器(Accumulator),用累加器进行的操作可能需要更少时间。累加器可用于乘、除、输入/输出等操作,它们的使用频率很高;寄存器 BX 称为基址寄存器(Base Register)。它可作为存储器指针来使用;寄存器 CX 称为计数寄存器(Count Register)。在循环和字符串操作时,要用它来控制循环次数;在位操作中,当移多位时,要用 CL 来指明移位的位数;寄存器 DX 称为数据寄存器(Data Register)。在进行乘、除运算时,它可作为默认的操作数参与运算,也可用于存放 I/O 的端口地址。

在 16 位 CPU 中,AX、BX、CX 和 DX 不能作为基址和变址寄存器来存放存储单元的地址,但在 32 位 CPU 中,其 32 位寄存器 EAX、EBX、ECX 和 EDX 不仅可传送数据、暂存数据保存算术逻辑运算结果,而且也可作为指针寄存器,所以,这些 32 位寄存器更具有通用性。

- 变址寄存

32 位 CPU 有 2 个 32 位通用寄存器 ESI 和 EDI。其低 16 位对应先前 CPU 中的 SI 和 DI,对低 16 位数据的存取,不影响高 16 位的数据。

寄存器 ESI、EDI、SI 和 DI 称为变址寄存器(Index Register),它们主要用于存放存储单元在段内的偏移量,用它们可实现多种存储器操作数的寻址方式,为以不同的地址形式访问存储单元提供方便。

变址寄存器不可分割成 8 位寄存器。作为通用寄存器,也可存储算术逻辑运算的操作数和运算结果。它们可作一般的存储器指针使用。在字符串操作指令的执行过程中,对它们有特定的要求,而且还具有特殊的功能。

3. 总结计数控制循环程序的设计方法。

- 计数控制型循环程序设计

这种程序设计方法直观,方便,但必须在循环次数已知的条件下才能采用。

- 条件控制型循环程序设计

在实际工作中,有时循环次数无法事先确定,但循环次数与问题中的某些条件有关,这时就应根据给定的条件满足与否来判断是否结束循环。

- 多重循环程序设计

在实际工作中,一个循环结构常常难以解决实际应用问题,所以人们引入了多重循环。这些循环是一层套一层的,因此又称为循环的嵌套。

注意:

1) 内层循环必须完全包含于外层循环内,不允许循环结构交叉。

2) 转移指令只能从循环结构内转出或可在同层循环内转移,而不能从一个循环结构外转入该循环结构内。

4. 说明怎样使用DEBUG进行程序调试的。调试过程中所遇到的问题是怎样解决的。
没问题。。。

实验七 按学号顺序把学生名次从终端上显示出来

一、实验目的

1. 掌握程序设计方法，合理划分层次
2. 掌握子程序的调用与返回的方法
3. 了解子程序的嵌套与递归

二、实验内容与要求

编制一程序，要求键入一个班的学生成绩，并存放于 50 字的 ERADE 数组中，然后根据 ERADE 中的成绩，把学生名次填入 50 字的 RANK 数组中，再按学号顺序把名次从终端上显示出来。

提示：

① 程序 MAIN

功能：根据输入的学生成绩，计算并显示出学生名次。

② 程序 INPUT

功能：接收一个班级学生的成绩，各成绩之间用空格隔开。

③ 子程序 RANKP

功能：计算一个班级学生的名次。

④ 子程序 OUTPUT

功能：输出（显示）一个班级的学生名次

⑤ 子程序 DECIBIN

功能：十进制转换二进制，存入 BX

⑥ 程序 BINDEC

功能：十进制转换二进制，并在屏幕上显示。

⑦ 子程序 DEC_DIV

功能：BX 的内容除以 CX 的内容，并在屏幕上显示一位商。

三、程序框图：

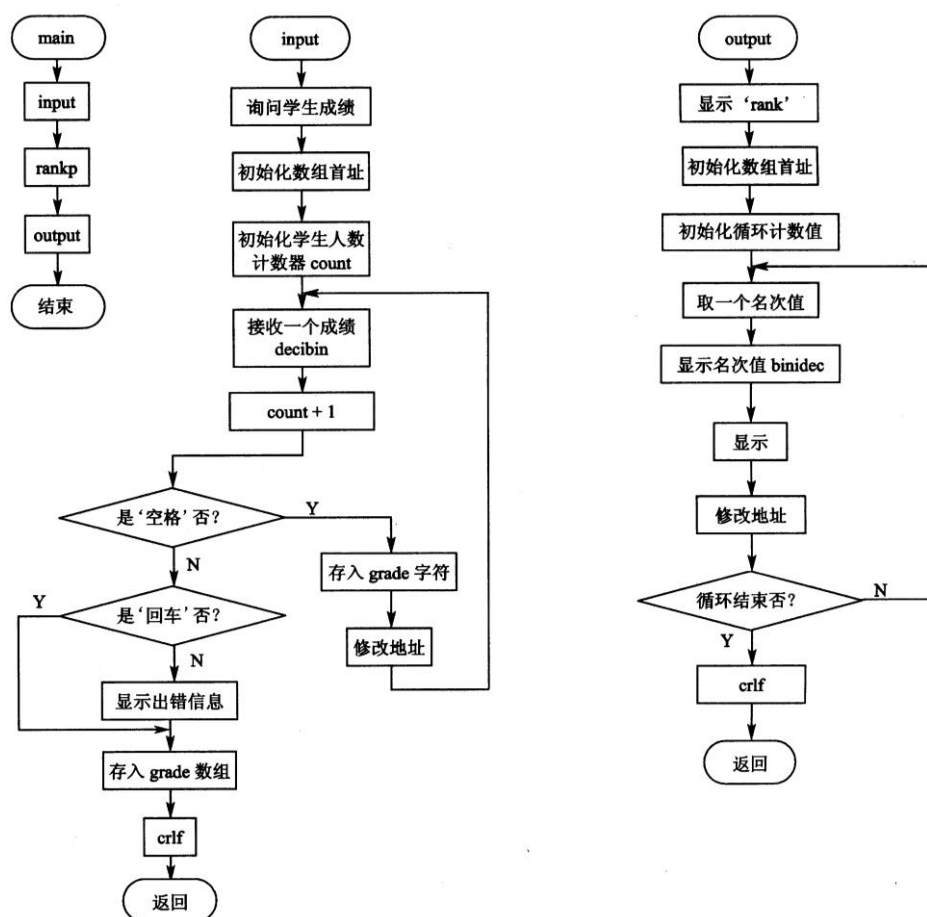


图 7.1 程序流程图

四、步骤

1. 自编主程序与子程序。
2. 输入本班级某门基础课成绩。

五、思考题

1. 写出 4 位 BCD 码转二进制数的算法。
2. 写出 AX 中二进制数转 BCD 码的算法。
3. 将上述子程序结构改为模块化程序设计。

六、实验报告：

1. 程序说明。说明程序的功能、结构。
2. 调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是怎样解决的，并对调试过程中的问题进行分析，对执行结果进行分析。
3. 写出源程序清单和执行结果。
4. 回答思考题。

源程序：

```

DATA SEGMENT
    SHOW02 DB '---INPUT ID(0000) AND
GRADE(00)---', ODH, 0AH, '$'
    SHOW04 DB '---      GRADE      SORTING
---', ODH, 0AH, '$'

    COUNT  DW 0
    COUNT1 DW 4, 2, 1
    COUNT2 DW 10
    SHOW2   DB      '      ID
GRADE', ODH, 0AH, '$'
    SHOW3  DB 'PLEASE ENTER THE NUMBER
OF STUDENT', ODH, 0AH, '$'
    SHOW4  DB '      ', '$'
    SHOW6  DB 'AVRTAGE GRADE:', '$'
    DIVISORS DW 1000, 100, 10, 1
    RESULTS DB 0, 0, 0, 0, '$'
    CUN     DW 10, 10
    ID      DW 100 DUP(?)
    GRADE   DW 100 DUP(?)
DATA ENDS

STACK SEGMENT STACK 'stack'
    DW 100H DUP(?)
    TOP LABEL WORD
STACK ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:STACK

OUT_DX MACRO Y
    LEA DX, Y
    MOV AH, 9H
    INT 21H
ENDM

CHANGE_LINE MACRO
    MOV AH, 02H
    MOV DL, ODH
    INT 21H
    MOV DL, 0AH
    INT 21H
ENDM

INT 21H
MOV DL, 0AH
INT 21H
ENDM
NUMBER MACRO
    OUT_DX SHOW3
    CALL INPUT
    LEA SI, COUNT
    MOV WORD PTR [SI], BX
ENDM
START:
    MOV AX, DATA
    MOV DS, AX
    NUMBER
    OUT_DX SHOW02
    CALL FUNCTION1
    OUT_DX SHOW04
    CALL FUNCTION2
    MOV AH, 4CH
    INT 21H
FUNCTION1 PROC NEAR
    OUT_DX SHOW2
    LEA SI, COUNT
    MOV CX, WORD PTR [SI]
    LEA SI, ID
    LEA DI, GRADE
ONE:
    CALL INPUT1
    MOV WORD PTR [SI], BX
    ADD SI, 2
    OUT_DX SHOW4
    CALL INPUT2
    MOV WORD PTR [DI], BX
    ADD DI, 2
    CHANGE_LINE
    LOOP ONE
    RET
FUNCTION1 ENDP

```



```

FUNCTION2 PROC NEAR
    LEA SI, COUNT
    MOV CX, WORD PTR [SI]
    LEA SI, ID
    LEA DI, GRADE
    DEC CX
    MOV AX, CX
    MUL COUNT1[2]
    ADD DI, AX
    ADD SI, AX
RP1:
    PUSH CX
    PUSH SI
    PUSH DI
RP2:
    MOV BX, WORD PTR [DI]
    CMP BX, WORD PTR [DI-2]
    JAE EXCHANGE
    XCHG BX, WORD PTR [DI-2]
    MOV WORD PTR [DI], BX
    MOV BX, WORD PTR [SI]
    XCHG BX, WORD PTR [SI-2]
    MOV WORD PTR [SI], BX
EXCHANGE:
    SUB SI, 2
    SUB DI, 2
    LOOP RP2
    POP DI
    POP SI
    POP CX
    LOOP RP1

    OUT_DX SHOW2
    LEA SI, COUNT
    MOV CX, WORD PTR [SI]
    MOV COUNT1[4], CX
    LEA SI, ID
    LEA DI, GRADE
    MOV CUN[0], DI
THREE:

    MOV AX, WORD PTR [SI]
    ADD SI, 2
    PUSH SI
    CALL OUTPUT
    OUT_DX SHOW4

    MOV DI, CUN[0]
    MOV AX, WORD PTR [DI]
    ADD DI, 2
    PUSH DI
    CALL OUTPUT
    CHANGE_LINE
    POP DI
    POP SI
    MOV CUN[0], DI
    DEC COUNT1[4]
    CMP COUNT1[4], 0
    JA THREE
RET
FUNCTION2 ENDP
INPUT PROC NEAR
    MOV BX, 0
NUM:
    MOV AH, 1
    INT 21H
    SUB AL, 30H
    JL EXIT
    CMP AL, 9
    JG EXIT
    CBW
    XCHG AX, BX
    MUL COUNT2[0]
    XCHG AX, BX
    ADD BX, AX
    JMP NUM
EXIT:
    RET
INPUT ENDP

INPUT1 PROC NEAR

```

```

MOV BX, 0
PUSH COUNT1[0]
NUM1:
MOV AH, 1
INT 21H
SUB AL, 30H
JL EXIT1
CMP AL, 9
JG EXIT1
CBW
XCHG AX, BX
MUL COUNT2[0]
XCHG AX, BX
ADD BX, AX
DEC COUNT1[0]
CMP COUNT1[0], 0
JA NUM1
EXIT1:
POP COUNT1[0]
RET
INPUT1 ENDP

INPUT2 PROC NEAR
MOV BX, 0
PUSH COUNT1[2]
NUM2:
MOV AH, 1
INT 21H
SUB AL, 30H
JL EXIT2
CMP AL, 9
JG EXIT2
CBW
XCHG AX, BX
MUL COUNT2[0]
XCHG AX, BX
ADD BX, AX
DEC COUNT1[2]
CMP COUNT1[2], 0
JA NUM2
EXIT2:
POP COUNT1[2]
RET
INPUT2 ENDP

OUTPUT PROC NEAR
MOV SI, OFFSET DIVISORS
MOV DI, OFFSET RESULTS
MOV CX, 4
CAL:
MOV DX, 0
DIV WORD PTR [SI]
ADD AL, 30H
MOV BYTE PTR [DI], AL
INC DI
ADD SI, 2
MOV AX, DX
LOOP CAL
MOV CX, 3
MOV DI, OFFSET RESULTS
NZ:
CMP BYTE PTR [DI], '0'
JNE PRINT
INC DI
LOOP NZ
PRINT:
MOV DX, DI
MOV AH, 9
INT 21H
RET
OUTPUT ENDP

CODE ENDS
END START

```

运行结果:

```
C:\>LAB07-F.EXE
PLEASE ENTER THE NUMBER OF STUDENT
5
-----INPUT ID(0000) AND GRADE(00)-----
ID      GRADE
0001    98
0002    78
0003    89
0004    68
0005    79
-----
ID      GRADE      GRADE SORTING      -----
4       68
2       78
5       79
3       89
1       98
```

实验八 统计不同成绩段学生的人数

一、实验目的：

1. 掌握分支、循环、子程序调用、DOS 功能调用等基本的程序结构。
2. 掌握综合程序的编制及调试方法。

二、实验内容：

设有十个学生成绩分别是 76, 69, 84, 90, 73, 88, 99, 63, 100 和 80 分。试编制一个子程序，统计低于 60 分，60~69 分，70~79 分，80~89 分，90~99 分和 100 分的人数，并输出显示统计结果。

三、编程提示：

1. 成绩分等部分采用分支结构，统计所有成绩则用循环结构完成，显示统计结果采用 DOS 功能调用。
2. 统计学生成绩和显示统计结果两部分内容用子程序结构来完成。

四、实验报告：

1. 程序说明。说明程序的功能、结构。
2. 调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是怎样解决的，并对调试过程中的问题进行分析，对执行结果进行分析。
3. 画出程序框图。
4. 写出源程序清单和执行结果。

源程序清单：

```
STACK SEGMENT PARA STACK 'stack'      data ends
    DB 1024 DUP(0)

STACK ENDS                             code segment
data segment                            assume cs:code, ds:data
    grade                                dw    start:
46,33,84,90,73,88,99,63,100,55          mov    ax,data
    grade50    db    0DH, 0AH, '<60'    mov    ds,ax
points: the result $'                   mov    cx,10
    grade60    db    0DH, 0AH, '60~69'   call    count
points: the result $'                   call    displ
    grade70    db    0DH, 0AH, '70~79'   mov     AH,4ch
points: the result $'                   int     21h
    grade80    db    0DH, 0AH, '80~89'   count proc near
points: the result $'                   mov     si,0
    grade90    db    0DH, 0AH, '90~99'   next:
points: the result $'                   mov     ax,grade[si]
    grade100   db    0DH, 0AH, '=100'    mov     bl,10
points: the result $'                   div     bl
    s          db    0                   mov     bl,al
```

```

        mov bh,0
        cmp bx,6
        jnb next1

        inc s[0]
        add si,2
        loop next
        ret

next1:
        inc s[bx]
        add si,2
        loop next
        ret
count endp

displ proc near
        lea si,s

        mov DX,OFFSET grade50
        call print_str

        add si,5

        mov DX,OFFSET grade60
        call print_str

        mov DX,OFFSET grade70

        call print_str

        mov DX,OFFSET grade80
        call print_str

        mov DX,OFFSET grade90
        call print_str

        mov DX,OFFSET grade100
        call print_str

        ret
displ endp

print_str proc near
        mov ah,9
        int 21h
        mov dl,[si]
        add dl,30h
        mov AH,2
        int 21h

        add si,1
        ret
print_str endp

code ends
end start

```

运行结果:

```

C:\>LAB08-s.EXE
<60 points: the result 3
60~69 points: the result 1
70~79 points: the result 1
80~89 points: the result 2
90~99 points: the result 2
=100 points: the result 1

```

附录 实验报告

_____实验报告
学院 _____ 专业 班 级 _____ 指导教师 _____
学号 _____ 姓 名 _____ 同 组 者 _____

_____实验报告
学院 _____ 专业 班 级 _____ 指导教师 _____
学号 _____ 姓 名 _____ 同 组 者 _____

实验报告

学院 _____ 专业 班 级 _____ 指导教师 _____
学号 _____ 姓 名 _____ 同 组 者 _____

