1．请将下列十进制数转换为二进制数，并统一使用 8 位二进制数表达。

（1）9　　00001001

（2）25　　00011001

（3）126　　01111110

（4）207　　11001111

（5）100　　01100100

（6）10　　00001010

（7）16　　00010000

（8）92　　01011100

（9）63　　00111111

（10）255　　11111111

2．请将下列二进制数转换为十进制数。

（1）$(00000001)_2$　　1

（2）$(01000010)_2$　　66

（3）$(10010011)_2$　　147

（4）$(01010101)_2$　　85

（5）$(01100100)_2$　　100

（6）$(10000000)_2$　　128

（7）$(11100111)_2$　　231

（8）$(00000011)_2$　　3

（9）$(01111111)_2$　　127

3．请将下列八进制数或十六进制数转换为 8 位二进制数。

（1）$(137)_8$　　01011111

（2）$(062)_8$　　00110010

（3）$(005)_8$　　00000101

（4）$(5A)_{16}$　　01011010

（5）$(39)_{16}$　　00111001

（6）$(1F)_{16}$　　00011111

4．请将下列二进制数分别转换为八进制和十六进制数

（1）$(01011010)_2$　　$(132)_8$　　$(5A)_{16}$

（2）$(01100001)_2$　　$(141)_8$　　$(61)_{16}$

（3）$(01110111)_2$　　$(167)_8$　　$(77)_{16}$

5．请将下列二进制数转换为 8 位补码。

（1）$(+00110001)_2$　　00110001

（2）$(-00010011)_2$　　11101101

（3）$(-01010100)_2$　　10101100

（4）$(+00000000)_2$　　00000000

（5）$(-01110111)_2$　　10001001

（6）$(-01100101)_2$　　10011011

6．请将下列 8 位补码转换为二进制数。（**注：教材原题中方括号记法不当，在此纠正**）

（1）10010001　　-01101111

（2）00110010　　+00110010

（3）11100110　　-00011010

（4）11111111　　-00000001

（5）11111001　　-00000111

（6）11000000　　-01000000

7．请将下列十进制数转换为 8 位二进制数，作为 8 位无符号数编码完成相应运算，给出运算结果，并且判断运算的有无溢出。

（1）56+32

　　00111000

+　　00100000

　0　01011000　　最高位无进位，运算无溢出

（2）100+75

```
    01100100
+   01001011
  0 10101111      最高位无进位，运算无溢出
```

（3）123+8

```
    01111011
+   00001000
  0 10000011      最高位无进位，运算无溢出
```

（4）125+134

```
    01111101
+   10000110
  1 00000011      最高位有进位，运算有溢出
```

（5）200+105

```
    11001000
+   01101001
  1 00110001      最高位有进位，运算有溢出
```

（6）26+34

```
    00011010
+   00100010
  0 00111100      最高位无进位，运算无溢出
```

注：自行练习无符号数减法运算时的溢出判断，注意不能使用补码加法代替无符号数减法来判断

8．请将下列十进制数转换为 8 位二进制补码，完成相应运算，给出运算结果，并且判断运算有无溢出。

（1）-123+66

$[-123]_{补}$=10000101

$[66]_{补}$=01000010

10000101

\+ 01000010

11000111

溢出分析：正+负，无溢出

（2）62-54

$[62]_{补}$=00111110

$[-54]_{补}$=11001010

00111110

\+ 11001010

1 00001000

溢出分析：正+负，无溢出

（3）130+140

$[130]_{补}$=010000010

$[140]_{补}$=010001100

分析：8 位补码表示范围为-128~127，需 9 位补码才能表示这两个带符号数，因此不能作补码运算

（4）-175-90

$[-175]_{补}$=101010001

分析：8 位补码表示范围为-128~127，需 9 位补码才能表示-175，因此不能作补码运算

（5）-78-9

$[-78]_{补}$=10110010

$[-9]_{补}$=11110111

10110010

\+ 11110111

1 10101001

溢出分析：负+负=负，运算正确，无溢出

（6）66+75

[66]$_{补}$=01000010

[75]$_{补}$=01001011

01000010

\+ 01001011

0 10001101

溢出分析：正+正=负，运算结果不是正确的 8 位补码，有溢出

注：补码加减运算溢出判断方法

1）正+负，无溢出

2）正+正=正，无溢出；正+正=负，溢出；

3）负+负=负，无溢出；负+负=正，溢出；

各种加减运算均可等价为以上三种情况

9．请简述 ASCII 码的概念，并说明该编码中的优越性。

答：ASCII 码为一种字符代码，它的实质是各种字符形状（点阵）的编号。该编码中，大小写英文字母的编号是以字母顺序连续编排，便于字符处理程序设计；数字字符‘0’~‘9’的 ASCII 码为 31H~39H（十六进制记法），编码连贯且容易转换为对应的数值，与 BCD 码间也具有方便的转换关系，使得数字字符串直接参与运算成为可能。

10．请简述 BCD 码的概念，并说明该编码的主要用途。

答：BCD 码是一种用于表述十进制数的二进制编码，它使用 4 个连续的二进制位表示一个十进制位，并限制其合法范围为 0~9。BCD 码主要用于使用二进制运算指令实现十进制数的运算，此功能需要结合二进制算术运算指令（加、减、乘、除）与 BCD 码运算调整指令才能完成，需要在运算中的十六进制进、借位调整为十进制进、借位。

1．8086/8088 CPU 被划分哪两个功能模块？这样划分的目的是什么？

答：8086、8088CPU 内部被划分为 EU（Execute Unit）和 BIU（Bus Interface Unit）两个功能模块。EU 为执行单元，其主要功能为对机器指令译码，按照时序生成各种控制信号来执行指令或其它时序过程，按照指令的要求完成各种运算。该单元与系统总线没有直接的交互操作。BIU 为总线接口单元，其主要功能为内存单元的物理地址计算，将控制信号、地址信号、数据信号传递到系统总线，或从系统总线接收状态信号、数据信号，从内存中读取下一条指令并保存在指令队列中。该单元与系统总线直接进行交互操作。将 CPU 划分这两个模块的主要目的在于将取指令与执行指令两种具有独立性的操作分离，若 EU 当前执行的指令不使用总线，则 BIU 可以在 EU 执行当前指令的同时通过总线从内存中读取下一条指令，使当前指令的执行周期与下一条指令的取指周期在时间上形成重叠，从而提高程序执行的速度。此划分方式为 CPU 最初级的流水线机制体现。

2．请简述逻辑地址、物理地址的概念，并说明二者间的转换关系。

答：逻辑地址与物理地址均为内存单元的地址表示形式，其中逻辑地址为 CPU 内部的地址形式，物理地址为系统总线上的地址形式。系统总线所使用的物理地址提供对内存单元的直接定位，8086、8088CPU 的地址总线为 20 条，即物理地址为 20 位，寻址范围为 2^{20}=1M 字节。由于 8086、8088CPU 内部的寄存器均为 16 位，无法直接表示 20 位的物理地址，因此使用由 16 位段基值和 16 位偏移量构成的逻辑地址形式，并且由 BIU 单元内的地址加法器完成逻辑地址到物理地址的转换，转换方法如下：

段基值*16+偏移量=物理地址

CPU 通过总线访问内存单元时，所使用的是物理地址，但指令系统、程序设计中使用的一定是逻辑地址。

3．8086/8088 CPU 中的寄存器被分为哪几个类别，它们分别的大致功能是什么？

答：8086、8088CPU 中的寄存器被划分为数据寄存器、地址指针寄存器、段寄存器、控制寄存器几种类别，其中数据寄存器用于保存完成运算所需要的源数据、运算结果；地址指针寄存器用于保存内存单元逻辑地址中的偏移量或偏移量分量；段寄存器用于保存内存单元逻辑地址中的段基值；控制寄存器包括 IP（指令指针）与 FR（标志寄存器），无条件、条件转移指令都通过修改（IP）实现程序流程转移，而 FR 中的标志位会影响条件转移指令的判断，从而间接影响程序的执行流程。

4．IP 寄存器的功能是什么？（IP）在哪些情况下会被修改？

答：IP 是指令指针寄存器，其功能为提供 BIU 单元将要从内存读取的下一条指令的偏移量，该寄存器固定与 CS 段寄存器搭配使用，形成下一条指令的完整物理地址。CPU 顺序执行内存中的指令序列时，会不断修改（IP），使之指向下一条指令；除此之外，转移指令、循环控制指令、子程序调用、返回指令、中断调用、返回指令都是通过修改（IP）来达到程序流程转移的目的。

5．8086/8088 CPU 中的指令队列是如何工作的？它与 CPU 中的寄存器有何不同？

答：指令队列是 8086、8088CPU 中的一个重要部件，位于 BIU 单元内部，它是由总线控制逻辑自动进行操作，用于存放由 BIU 通过总线读取的机器指令，EU 单元对指令译码时仅从指令队列读取机器指令，不再访问总线。指令队列是一组存储单元，从物理结构上观察，它与指令系统中能够使用的寄存器没有区别。但是，指令系统中的机器指令并不能直接对指令队列实施控制，因为机器指令的执行在时间上处于指令的执行周期，而不是取指周期，取值周期是由 CPU 自动执行的，与机器指令规定的功能无关。换言之，对于通常的寄存器，我们的程序能够对它们实施直接控制，但却不能控制指令队列，它由 CPU 自动管理。

6．CF、OF 标志的含义是什么？它们分别在什么时候提供的才是有效的标志位信息？

答：CF（Carry Flag）标志的含义是无符号数进位标志，多数情况下用于标识最近的算术运算（通常是加减运算）中，如果将操作数解释为无符号数，其运算结果有没有超出字节或字（与操作数类型有关）的表示范围。CF 提供有效标志位信息的前提如下：

1）最近影响标志位的指令要影响 CF 标志

2）该指令对 CF 标志的影响是有意义的

3）程序员将该指令的操作数解释为完整的无符号数（若为超过 16 位的长操作数运算，操作数可能不是完整的，其解释请参见第 5 章中的带进位加法指令，CF 对长补码运算的进位衔接仍然有效）

OF（Overflow Flag）标志的含义是带符号数溢出标志，多数情况下用于标识最近的算术运算（通常是加减运算）中，如果将操作数解释为补码，其运算结果有没有超出字节或字（与操作数类型有关）的表示范围。OF 提供有效标志位信息的前提如下：

1）最近影响标志位的指令要影响 OF 标志

2）该指令对 OF 标志的影响是有意义的

3）程序员将该指令的操作数解释为完整的补码（若为超过 16 位的长操作数运算，操作数可能不是完整的，其解释请参见第 5 章中的带进位加法指令，OF 仅对长补码的最高字节或字的运算结果有效）

7．TF 标志的含义是什么？它的重要性体现在哪里？

答：TF 标志为单步跟踪标志位，若 TF=0，则 CPU 处于连续执行机器指令的状态，CPU 不间断的从内存读取指令并执行指令；若 TF=1，则 CPU 处于单步执行机器指令的状态，CPU 每执行一条指令就触发一次单步中断，引起单步中断服务程序的执行，该中断服务程序在屏幕上显示 CPU 内各寄存器的当前状态。在 DEBUG 调试工具中，单步调试命令 T 就是基于该标志位的。TF 标志位的重要性在于它为单步调试提供了必要的硬件基础。

8．请按照你自己的理解，简述状态标志位、控制标志位的区别。

答：通常情况下，状态标志位是由运算指令间接影响的，这些运算指令并没有把它们作为目的操作数。状态标志位是指各种运算指令完成运算后，根据不同操作数解释下运算结果的特点提供的状态信息，这些状态信息可能会在后续程序执行中影响条件转移指令的判断，从而影响程序执行流程，实现程序的分支、循环结构。与状态标志位不同，控制标志位由特定的标志位设置指令来设置，修改这些控制标志位会影响 CPU 的工作模式与特定事件的处理。例如 TF 标志用于控制 CPU 工作于连续执行或单步方式，IF 标志用于控制 CPU 是否响应可屏蔽中断请求等。

9．请简述状态标志位提供有效信息的必要条件。

答：（请参见教材 4.2.4 小节最后一部分的详细说明，总共三个必要条件。）

10．请说明寄存器隐含使用、特定使用的含义。

答：（请参见教材 4.2.5 小节中的详细说明。）

习题 5

1．试说明以下指令中各操作数的寻址方式，如果是存储器寻址，请给出其 EA 计算公式，并说明所使用的段寄存器。

（1）MOV　AL，08H

源操作数：立即数寻址

目的操作数：寄存器寻址

（2）MOV　[0120H]，BL

源操作数：寄存器寻址

目的操作数：直接寻址，EA=0120H，使用 DS 段寄存器

（3）ADD　[BX]，AL

源操作数：寄存器寻址

目的操作数：寄存器间接寻址，EA=(BX)，使用 DS 段寄存器

（4）PUSH　[SI]0200H

源操作数：变址寻址，EA=(SI)+0200H，使用 DS 段寄存器

目的操作数：隐含寻址（指令中未直接体现）

（5）SUB　AX，[BP]

源操作数：寄存器间接寻址，EA=(BP)，使用 SS 段寄存器

目的操作数：寄存器寻址

（6）AND　VAR1+4，DL

源操作数：寄存器寻址

目的操作数：直接寻址，EA=VAR1+4，使用 DS 段寄存器

（7）PUSHF

源操作数、目的操作数均为隐含寻址

（8）MOV　ES：[BX]0100H，AL

源操作数：寄存器寻址

目的操作数：基址寻址，EA=(BX)+0100H，使用 ES 段寄存器

（9）ADC　BYTE PTR [BP][SI]0210H，45H

源操作数：立即数寻址

目的操作数：基址变址寻址，EA=(BP)+(SI)+0210H，使用 SS 段寄存器

（10）OR　ARRY[BX][DI]，CL

源操作数：寄存器寻址

目的操作数：基址变址寻址，EA=(BX)+(DI)+ARRY，使用 DS 段寄存器

2．试分析下列汇编指令是否存在语法错误，如果有语法错误存在，请说明是怎样的错误。

（1）PUSH　8243H

错误，单操作数指令不能使用立即数

（2）POP　AL

错误，进栈、出栈指令的操作数应为 16 位

（3）MOV　AL，6543H

错误，源、目的操作数类型不匹配

（4）ADD　[0100H]，64H

错误，目的操作数应使用 PTR 运算符指出类型，否则具有二义性

正确的写法：ADD BYTE PTR [0100H],64H，（或使用 WORD PTR）

（5）ADC　VAR1，VAR2

错误，8086 指令系统的双操作数指令中，必须有一个是寄存器，不能两个操作数同为内存单元

（6）MOV　DS，ES

错误，段寄存器间不能使用 MOV 指令直接传递数据，必须通过通用寄存器作为中转

（7）MOV　DS，0620H

错误，使用 MOV 指令向段寄存器传递数据时，不能使用立即数

（8）LEA　BX，AX

错误，LEA 指令的源操作数必须为内存单元

（9）DEC　AL，AH

错误，DEC 指令为单操作数指令

（10）SHR　BL，3

错误，当移位次数大于 1 时，在移位指令中特定使用 CL 寄存器给出移位次数

正确的写法：MOV　CL, 3

　　SHR　BL, CL

3．试说明分别执行下列各组指令后，CF、OF、AF、ZF、SF、PF 这六个状态标志分别是怎样的取值。

（1）MOV　AL，08H

　　ADD　AL，0F9H

CF=1; OF=0; AF=1; ZF=0; SF=0; PF=0

（2）MOV　AL，0E1H

　　ADD　AL，0F4H

CF=1; OF=0; AF=0; ZF=0; SF=1; PF=0

（3）MOV　AL，01H

　　SUB　　AL，02H

CF=1; OF=0; AF=1; ZF=0; SF=1; PF=1

（4）MOV　AL，02H

　　INC　　AL

CF 维持 MOV 指令前的取值 (INC 指令不影响 CF)；OF=0; AF=0; ZF=0; SF=0; PF=1

（5）MOV　AL，01H

　　AND　AL，02H

CF=OF=0; AF 不确定; ZF=1; SF=0; PF=1

4．按要求分析下面程序片段的执行结果。

MOV　AL，0C2H

MOV　AH，0E4H

ADD　AL，AH

执行该程序片段后，（AL）=？，（AH）=？，如果将 ADD 指令的两个操作数解释为无符号数，运算有没有溢出？为什么？如果将 ADD 指令的两个操作数解释为补码，运算有没有溢出？为什么？

答：执行该程序片段后，（AL）=0A6H，（AH）=0E4H，如果操作数解释为无符号数，运算溢出，因为加法运算后最高位产生了进位，CF=1，需使用 9 个二进制位才能表达完整运算结果；如果将操作数解释为补码，则运算没有溢出，因为从操作数与运算结果的符号位观察，两个操作数均为负数补码，相加后所得结果仍然为负数补码，符号位正确，表明加法结果未超出补码表示范围（这里是 8 位补码的表示范围），加法运算后 OF=0。

5．按要求分析下面程序片段的执行结果。

MOV　AL，98H

MOV　BL，42H

XCHG　AL，BL

SUB　AL，BL

执行该程序片段后，（AL）=？，（BL）=？，如果将 SUB 指令的两个操作数解释为无符号数，运算有没有溢出？为什么？如果将 SUB 指令的两个操作数解释为补码，运算有没有溢出？为什么？如果将 SUB 指令的两个操作数解释为补码，其减法运算对应的十进制真值表达式应如何书写？

答：执行该程序片段后，（AL）=0AAH，（BL）=98H，如果将操作数解释为无符号数，则运算溢出，从操作数判断，此运算属于被减数小于减数的情况，这在无符号数运算中是不允许的（如果当前操作数仅为长数据的一部分，则另当别论），减法运算后最高位必然产生借位，CF=1；如果操作数解释为补码，运算也溢出，从操作数判断，此运算属于“正-负”类型，等价于“正+正”类型，正确的运算结果应为正数或零的补码，而运算结果的符号位却为“负”，表明运算结果超出补码表示范围（这里为 8 位补码表示范围），减法运算后 OF=1。

SUB 指令所使用的被减数补码为 42H=01000010B，减数补码为 98H=10011000B，由于被减数为正数补码，它等于真值本身，而减数补码为负数补码，将其取反加 1 后，添上负号，得到其二进制真值为 -01101000B。将被减数、减数的二进制真值转换为十进制后，得到真值运算表达式：66 -（-104）=170，很明显运算结果超出 8 位补码的最大值+127。

6．按要求分析下面程序片段的执行结果。

STC

MOV　AL，03H

AND　AL，02H

ADC　AL，00H

执行该程序片段后，（AL）=？

答：（AL）=02H，此题应注意 AND 指令会强置 CF 为 0。

7．假设（DS）=1000H，（SS）=2000H，字内存单元（10200H）=0870H，（10202H）=2000H，（20870H）=0203H，（20872H）=0405H，括号内所给为内存单元物理地址，括号表示该地址所指示单元中保存的数据，分别执行下列程序片段后，按要求分析各程序片段的执行结果。

（1）MOV　AL，[0200H]

执行该程序片段后，（AL）=?

答：源操作数地址为（DS）*16+0200H=10000H+0200H=10200H, 因此执行该程序片段后，（AL）=70H （逆序存放，低地址对应低数据位）

（2）MOV　BP，0871H

　　MOV　BL，[BP]

执行该程序片段后，（BL）=?

答：第二条指令的源操作数地址为（SS）*16+（BP）=20871H，执行该程序片段后，（BL）=02H （逆序存放，高地址对应高数据位）

（3）LEA　SI，[0200H]

执行该程序片段后，（SI）=?

答：LEA 指令将源操作数的 EA 传送到目的操作数保存，（SI）=0200H

（4）MOV　SI，[0200H]

　　LEA　 SI，[SI]

执行该程序片段后，（SI）=?

答：MOV 指令中源操作数地址为（DS）*16+0200H=10200H，执行后（SI）=0870H，第二条指令源操作数的 EA 直接为（SI）=0870H，执行后仍有（SI）=0870H。

（5）LDS　BX，[0200H]

　　MOV　AL，[BX]0002H

执行该程序片段后，（AL）=?

答：LDS 指令中源操作数地址为（DS）*16+0200H=10200H，将（10200H）字单元内容传

递到 BX 保存，将（10202H）字单元内容传递到 DS 保存，执行后（BX）=0870H，（DS）=2000H；MOV 指令中源操作数地址为（DS）*16+（BX）+0002H=20872H，执行后（AL）=05H。

8．按要求分析下面程序片段的执行结果。

MOV　AX，651CH

SHL　AL，1

RCL　AH，1

执行该程序片段后，（AX）=？，该程序片段的功能是什么？如果将（AX）解释为无符号数，那么运算是否溢出？为什么？如果将（AX）解释为补码，运算是否溢出？为什么？SHL 与 SAL 指令间有什么关联和区别？

答：执行该程序片段后，（AX）=0CA38H，该程序片段的功能为将 AX 中的 16 位编码左移 1 位，等价于乘以 2（也可理解为自加一次）。如果将（AX）解释为无符号数，那么运算没有溢出，因为最后一次移位操作后，最高移出位为 0，即 CF=0（自加完成后最高位无进位）；如果将（AX）解释为补码，运算溢出，因为移位前后（AX）的最高位发生了变化（由 0 变为 1），符号位在运算中丢失，可以理解为自加运算结果超出了 16 位补码表示范围。

SHL 与 SAL 指令本质上对应同一条机器指令，在功能上并无区别，因为无符号数与补码的左移操作是完全相同的，但为了指令系统设计的规整性，在汇编指令中将它们区分开，SHL 针对无符号数左移，SAL 针对补码左移。

9．按要求分析下面程序片段的执行结果。

MOV　AL，35H

AND　AL，0FH

执行该程序片段后，（AL）=？CF、OF、AF、ZF、SF、PF 标志取值是什么？该程序片段的功能是什么？

答：执行该程序片段后，（AL）=05H，CF=0；OF=0；AF 不确定；ZF=0；SF=0；PF=1；该程序片段的功能是将（AL）中的低 4 位数据分离出来，屏蔽高 4 位。

10．假设一个 48 位的补码按照由低位到高位的顺序保存在字类型的内存单元 VA1、VA1+2、VA1+4 中，试按下列要求完成程序片段设计。(红字部分请在教材中纠正)

（1）设计程序片段，实现将该 48 位补码除以 4 的功能，运算结果仍然保存在原内存单元中。

解：

```
SAR   VA1+4, 1
RCR   VA1+2, 1
RCR   VA1, 1
SAR   VA1+4, 1
RCR   VA1+2, 1
RCR   VA1, 1
```

（2）设计程序片段，求该 48 位补码的相反数补码，运算结果仍然保存在原内存单元中。

解：

```
NOT   VA1
NOT   VA1+2
NOT   VA1+4
ADD   VA1, 1
ADC   VA1+2, 0
ADC   VA1+4, 0
```

11．试说明如何使用 CMP 指令提供的标志位判断两个补码操作数大小关系的原理。

答：请参考本章关于比较指令（CMP）中的详细介绍加以说明。

12．假设（SP）=0060H，执行两次 PUSH 指令后，（SP）=？假设（SP）=0038H，执行三次 POP 指令后，（SP）=？

答：执行两次 PUSH 指令后，（SP）= 005CH；执行三次 POP 指令后（SP）

=0038H+0002H*3=0038H+0006H=003EH

13．按要求分析下面程序片段的执行结果。

MOV　AL，01H

NEG　AL

INC　 AL

执行该程序片段后（AL）=？，CF、OF 标志的状态是什么？

答：执行该程序片段后（AL）=0，CF=1，注意，这是受 NEG 指令影响的结果，INC 指令不影响 CF 标志；OF=0，加法运算并无溢出，因为 0FFH 为-1 的补码，加 1 后等于 0 是正确的。

14．按要求分析下面程序片段的执行结果。

MOV　BL，51H

AND　BL，0FEH

XOR　BL，50H

DEC　BL

执行该程序片段后（BL）=？，CF、OF 标志的状态是什么？

答：执行该程序片段后（BL）=0FFH，CF=0，注意，这是受 XOR 指令的影响，XOR 指令将 CF 强置为 0，而 DEC 指令不影响 CF；OF=0，此标志是受 DEC 指令影响的结果。此题中应注意，逻辑运算指令会将 CF、OF 强置为 0，而 DEC 指令不影响 CF 标志。

15．按照各小题的要求分别设计程序片段。

（因存在多种设计方式，程序设计题目的答案仅作为参考）

（1）将 AL 寄存器的高 4 位与低 4 位交换

MOV　CL, 4

ROL　 AL, CL

（2）将 TF 标志位置 1

```
PUSHF
POP     AX
OR      AX, 0100H
PUSH    AX
POPF
```

（3）将 AL 寄存器的第 7 位清 0，但不影响其它数据位

```
AND    AL, 7FH
```

（4）分离 AL 寄存器的最低两位，其它数据位清 0。

```
AND    AL, 03H
```

（5）分离 AL 寄存器的高 4 位与低 4 位，并分别保存在 BL、BH 的低 4 位

```
PUSH    AX
AND     AL, 0FH
MOV     BL, AL
POP     AX
AND     AL, 0F0H
MOV     CL, 4
ROL     AL, CL
MOV     BH, AL
```

1.试说明下述概念的异同：

① 指令和伪指令。

答：汇编指令与机器指令原则上一一对应，在汇编过程中，汇编指令会被替换为机器指令，最终生成可执行的目标代码；指令的功能是由特定 CPU 的指令译码器确定的，因此，指令是一个硬件相关的概念；汇编过程仅将汇编指令替换为机器指令，而机器指令的执行是在可执行程序执行阶段完成的。

汇编语言伪指令在汇编过程中并不产生机器指令，它们的功能在于指示汇编程序按照指定的方式对汇编语言源程序做出解释，伪指令的功能是由汇编程序解释的，与硬件结构无关；伪指令的执行是在汇编过程中完成的，在程序执行阶段则不存在伪指令的概念。

② 符号常量、变量和标号。

答：符号常量的功能是定义一个简单符号表示一个较复杂的常量，便于源程序编写，符号常量不会产生内存空间分配，且仅在源程序汇编阶段有效，程序执行阶段无符号常量的概念；

变量是指特定类型（字、字节等）的内存空间，若在指令或伪指令中引用变量，则实质上是引用其段内偏移量，在机器指令中变量对应位移量字段，与符号常量不同，变量定义会导致内存空间的分配，空间分配是在源程序汇编阶段完成的，而对该内存空间的使用则是在程序的执行阶段；

标号的概念与变量有相似之处，引用变量或标号时，实质上都是在引用它们对应的段内偏移量，在机器指令中引用的变量或标号都对应位移量字段；但是，变量通常是在指令操作数的寻址中使用，其对应的位移量通常解释为无符号数编码，标号却通常在转移类指令中的用作目标地址，其对应的位移量通常解释为补码（涉及向前、向后转移两个方向，详见分支、循环程序设计相关章节）。

③ 表达式的运算符和运算指令。

答：表达式的运算符由汇编程序解释，属于纯语法成分，运算符对应的运算是在源程序汇编阶段完成的，在汇编结束后，表达式均被替换为相应的数值，在程序执行阶段，不再有表达式这一概念；运算指令是由 CPU 解释执行的，与机器硬件相关，其运算是在程序执行阶段完成的，汇编过程中仅将汇编指令替换为机器指令，并不完成运算。若要区分运算符与运算指

令，根据其在指令中的位置就可以判断，若关键字位于操作助记符位置，则它表示指令，若关键字位于操作数位置，则它表示运算符。

④ ENDS 和 END。

答：ENDS 伪指令用于结束段定义，它一定与 SEGMENT 伪指令成对的使用；END 伪指令用于结束整个源程序的定义，汇编程序检测到 END 伪指令后就认为源程序已经结束，位于 END 伪指令之后的所有语句都不会做出解释，通常 END 伪指令后紧跟一个标号，用于指示源程序中第一条指令的位置。

2.用示意图说明下述为指令语句分配的字节数及各字节内容（假设数据段的段基址是 0ABCDH,且各变量依序从偏移地址 0 开始定义）。

```
VARA    EQU     10
VARB    DB      10
VARC    DW      1234H
VARD    DD      12345678H
VARE    DB      2 DUP(1,2),0
VARF    DW      VARB
VARG    DD      VARB
```

答：（注：VARA 为符号，因此未分配空间）

变量名称及对应的物理地址	内存单元中的内容（字节为单位）
VARB，0ABCDH	0AH
VARC，0ABCEH	34H
0ABCFH	12H
VARD，0ABD0H	78H
0ABD1H	56H
0ABD2H	34H
0ABD3H	12H

VARE, 0ABD4H	01H
0ABD5H	02H
0ABD6H	01H
0ABD7H	02H
0ABD8H	00H
VARF, 0ABD9H	00H (VARB 的偏移量)
0ABDAH	00H
VARG, 0ABDBH	00H (VARB 的偏移量)
0ABDCH	00H
0ABDDH	0CDH (VARB 的段基值)
0ABDEH	0ABH

3.说明下面代码中 N1、N2 和 N3 的异同：

```
N1      EQU     10
N2      DB      10
N3      DW      10
```

答：N1 为符号，引用 N1 等价于引用数值 10，汇编结束后，所有引用 N1 的位置都被替换为数值 10，汇编程序不会为 N1 分配空间；N2 为字节类型的变量，其长度为 1 个字节；N3 为字类型变量其长度为 2 个字节，N2 与 N3 的初值相同，引用 N2 与 N3 等价于引用这两个变量的段内偏移量。

4.按下述要求在数据段定义变量：

① 定义一字符串变量且初值为“Hello,World!”。

```
STR1   DB   'Hello,World!'
```

② 定义一字节型变量且初值为 0。

```
VAR1    DB    0
```

③ 定义一未指定初值的 100 个单元的字型变量。

```
ARY1    DW    100 DUP(?)
```

④ 定义一值为 1000 的符号常量。

```
SYM1    EQU    1000
```

5.对数据段定义的下述变量:

```
NUM1    DB      60
NUM2    EQU     50
NUM3    DB      20H
```

判断下述语句是否有错并指出是什么错误。

① MOV NUM2,AL

错误，NUM2 为符号常量，在指令中仅能作为源操作数，不能作为目的操作数，因为符号常量不会被分配空间，自然也就没有相应的地址。

② MOV AX,NUM3

错误，NUM3 为字节类型的变量，该指令试图将其内容传递到 16 位寄存器 AX 中，汇编程序无法明确应该传递到 AL 还是 AH，因此指令具有二义性，无法唯一解释。

③ MOV NUM3,NUM1

错误，8086/8088 指令系统的双操作数指令最多允许一个操作数使用存储器寻址方式。

④ MOV NUM1,NUM2

正确，这里 NUM2 是符号，在指令的源操作数处引用符号，对应其立即数字段。

⑤ MOV BX,AX+NUM2

错误，表达式中不能使用存储单元名称（包括寄存器、内存单元、端口），因为表达式是在汇编阶段完成运算的，而存储单元的内容必须要到程序执行阶段才能确定，因此包含存储单元的表达式无法在汇编阶段完成运算，自然也无法得到语法的允许。

⑥ MOV NUM1,AL and 0F0H

错误，原因同上题。

⑦ MOV AX,NUM3-NUM1

正确，在指令操作数位置引用两个变量之差，实质上是应用两个变量段内偏移量之差，即两个变量间的字节距离，而不是指两个变量内容之差。（注：在表达式中，两个变量间的运算仅允许减法，不允许加法，因为加法无意义）

6.对于下面的数据定义,写出各条指令执行的结果:

```
FLDB DW 0A24FH
TABLE DB 32H,52,0C2H,213
TEA EQU WORD PTR TABLE
ARRAY DB 'ABCD'
COUNT EQU $-ARRAY
```

① MOV AX,FLDB

(AX)=0A24FH

② MOV BX,TEA

(BX)=0002H （假定 FLDB 的段内偏移量为 0）

③ MOV CH,TABLE+2

(CH)=0C2H

④ MOV DL,ARRAY

(DL)= ‘A’（或 41H）

⑤ MOV DH,COUNT

(DH)=04H

本题表达式中出现的$，是“编译（汇编）位置指针”，在编译（汇编）时将被替换成数据段的当前地址偏移量。

7.假设数据段有下属变量定义：

```
INAME   DB  31 DUP(?)
```

```
ADDRESS DB  31 DUP(?)
CITY    DB  16 DUP(?)
ID      DB  1,2,1,6,8
```

① 用一条 MOV 语句将 INAME 的偏移量送入 SI。

```
MOV  SI, OFFSET INAME
```

② 用一条指令将 ID 的头两个字节的内容送入 BX。

```
MOV  BX, WORD PTR ID
```

③ 写出后三个字段的长度表达式。

```
CITY-ADDRESS
ID-CITY
$-ID
```

8.在下述程序框架中分号“;”后面，描述同一行代码的功能。

```
stack0  segment stack 'stack'    ;
        dw 40h dup(0)            ;
stack0  ends

data    segment                  ;
        ;

data    ends

code    segment                  ;
        ;在此说明下一句
        assume  cs:code,ds:data,ss:stack0
main:   mov ax,data              ;
        mov ds,ax
        ;

        mov ah,4ch               ;
        int 21h

code    ends                     ;
        end main                 ;
```

参考注释如下：

；定义一个段，名称为 stack0，segment 关键字后的 stack 关键字指明该段属性为堆栈段，运行该程序时，操作系统自动初始化 SS、SP，使用单引号括住的标识符为段的类别名称，运行程序时，操作系统会尽量将具有同一类别名的段分配在连续的内存空间内。

；分配 40H 个字的内存空间作为堆栈段空间

；定义数据段

；定义代码段

；使用 ASSUME 伪指令说明每个段寄存器默认指向的段（注：与段寄存器内容的初始化无关，DS、ES 初始化需使用指令完成；若指明堆栈段的堆栈属性，SS 的初始化由操作系统完成，若未指明堆栈属性，则也需要使用指令来完成；CS 的初始化由操作系统完成。

；此两条指令初始化 DS 段寄存器

；此两条指令使用系统调用结束程序运行，并将 CPU 控制权返还给操作系统。

；ENDS 结束代码段定义

；END 结束源程序，并给出程序中第一条指令所在的标号。

9.编写一完整程序，该程序将数据段定义的字符串“Hello,World.”末尾的句号替换成感叹好“!”。

参考答案（不唯一）

```
data segment
str1 db  ‘Hello, World.’
len  equ  $-str1
data ends

code segment
assume ds:data
begin: mov  ax, data
       mov  ds, ax
```

```
        lea  si, str1
        add  si, len-1
        mov  byte ptr [si],  '!'
        mov  ah, 4ch
        int  21h
code ends
end  begin
```

10.编写一完整程序，分别在两个数据段 DSEG1 和 DSEG2 中各定义一个字型变量 X 和 Y 并计算两数差的绝对值。

参考答案（不唯一）

```
; 此处假定运算结果保存在 DSEG1 段的 Z 变量中，由于要求求解绝对值，因此假定变量中
; 的编码为补码
; 说明，由于求绝对值涉及到条件转移指令，在本章还未介绍，因此可只要求同学们计算两
; 数之差即可，或将 X、Y 大小关系指定为已知条件来求解绝对值。
DSEG1   SEGMENT
X       DW      4723H
Z       DW      ?
DSEG1   ENDS

DSEG2   SEGMENT
Y       DW      528AH
DSEG2   ENDS

CODE    SEGMENT
ASSUME  DS:DSEG1, ES:DSEG2
BEGIN:  MOV   AX, DSEG1
        MOV   DS, AX
        MOV   AX, DSEG2
        MOV   ES, AX
        MOV   AX, X
        SUB   AX, Y
        MOV   Z, AX
        MOV   AH, 4CH
        INT   21H
CODE    ENDS
END     BEGIN
```

1．简述无条件转移指令在指令系统中存在的必要性。

答：无条件转移指令对于实现分支结构是必要的，因为前一个分支的代码执行完毕后，必须使用无条件转移指令才能使程序流程跨过后面分支的代码，到达分支出口。

2．简述条件转移指令的分类，以及各类条件转移指令的功能。

此题请参考教材中关于条件转移指令的介绍。

3．按要求分析下面程序片段的执行结果。

```
        MOV   AL，-5
        MOV   BL，-3
        CMP   AL，BL
        JGE    L1
        JMP    L2
L1：    MOV   AH，00H
        JMP    L3
L2：    MOV   AH，01H
L3：    ……
```

执行该程序片段后，（AH）=?

（AH）=01H

4．按要求分析下面程序片段的执行结果。

```
        MOV   AL，6CH
        MOV   CX，8
        XOR   AH，AH
L1：    ROL   AL，1
        ADC   AH，0
        LOOP  L1
```

执行该程序片段后，（AH）=?

（AH）=04H

5．试分析并说明下面程序片段的功能。

……

```
ARY1   DB   35，23，-6，-21，90，65，73，-54，-7
LEN    EQU   $ - ARY1
……
       LEA    BX，ARY1
       XOR    AL，AL
       MOV    CX，LEN
       JCXZ   EX1
L1：   TEST   [BX]，01H
       JNZ    L2
       JMP    L3
L2：   INC    AL
L3：   INC    BX
       LOOP   L1
EX1：  ……
```

程序功能：统计补码数组 ARY1 中奇数的数量（也可解释为统计 ARY1 中最低位为 1 的数据个数），统计结果保存在 AL 寄存器中。

6．假设有一字符串 STR，试编写一个程序查找 STR 字符串中第一个非空格字符的位置，如果找到则将字符位置保存在字节变量 POS1 中，如果没有找到则将 0FFH 保存至 POS1 中。

参考答案（不唯一）

```
DATA   SEGMENT
STR    DB     ‘       HELLO!     ’
LEN    EQU    $-STR
POS1   DW     ?
DATA   ENDS

CODE   SEGMENT
ASSUME   DS: DATA
```

```
BEGIN: MOV    AX, DATA
       MOV    DS, AX
       MOV    POS1, 0FFH
       LEA    BX, STR1
       MOV    CX, LEN
LOP1:  CMP    [BX], 20H
       JNZ    L1
       JMP    L2
L1:    MOV    POS1, BX
       JMP    L3
L2:    INC    BX
       LOOP   LOP1
L3:    MOV    AH, 4CH
       INT    21H
CODE   ENDS
END    BEGIN
```

7．假设有一字类型的补码数组 ARY1，试编写一个程序查找 ARY1 数组中的最大值，并将查找结果保存在字类型变量 RES1 中。

参考答案（不唯一）

```
DATA   SEGMENT
ARY1   DB   -53, 44, 75, 36, -2, 64, -27, -1, 9
LEN    EQU  $-ARY1
RES1   DB    ?
DATA   ENDS

CODE   SEGMENT
BEGIN: MOV   AX, DATA
       MOV   DS, AX
       LEA   BX, ARY1
```

```
        MOV    CX, LEN-1
        MOV    AL, ARY1
LOP1:   CMP    [BX], AL
        JGE    L1
        JMP    L2
L1:     MOV    AL, [BX]
L2:     INC    BX
        LOOP   LOP1
        MOV    RES1, AL
        MOV    AH, 4CH
        INT    21H
CODE    ENDS
END     BEGIN
```

8．假设有一字节类型的补码数组 DAT1，试编写一个程序统计数组 DAT1 中正数（包括 0）和负数分别的数量，正数统计结果保存在字节变量 RES1 中，负数统计结果保存在字节变量 RES2 中。

参考答案（不唯一）

```
DATA    SEGMENT
DAT1    DB     -34, -2, -1, 55, -3, 0, 94, 100, 7, -7
LEN     EQU    $-DAT1
RES1    DB     ?
RES2    DB     ?
DATA    ENDS

CODE    SEGMENT
ASSUME    DS: DATA
BEGIN: MOV    AX, DATA
        MOV    DS, AX
        XOR    AX, AX
```

```
        MOV   CX, LEN
        LEA   BX, DAT1
LOP1:   CMP   [BX], 0
        JS    L1
        JMP   L2
L1:     INC   AH
        JMP   L3
L2:     INC   AL
L3:     INC   BX
        LOOP  LOP1
        MOV   RES1, AL
        MOV   RES2, AH
        MOV   AH, 4CH
        INT   21H
CODE    ENDS
END     BEGIN
```

9．假设有两个长度为 8 个字的补码，分别保存在字类型数组 DA1 和 DA2 中，低地址保存低位，试设计一个程序实现两个长补码的减法，减法结果保存在字类型数组 RES1 中，是否溢出用 OF 标志反映。

参考答案（不唯一）

```
DATA    SEGMENT
DA1     DW    0143H, 435AH, FA1CH, 7754H, 6A11H, B2FFH, 1B2BH, 0FF1H
DA2     DW    E227H, A82BH, 33BAH, 72B6H, 1F21H, 2614H, 4FC1H, 2CBAH
LEN     EQU   8
RES1    DW    LEN    DUP(?)
DATA    ENDS

CODE    SEGMENT
BEGIN: MOV    AX, DATA
```

```
        MOV    DS, AX
        MOV    CX, LEN
        XOR    BX, BX
        CLC
LOP1:   MOV    AX, DA1[BX]
        SBB    AX, DA2[BX]
        MOV    RES1[BX], AX
        ADD    BX, 2
        LOOP   LOP1
        MOV    AH, 4CH
        INT    21H
CODE    ENDS
END     BEGIN
```

10．假设有一个长度为 4 个字的无符号数，保存在字类型数组 DA1 中，低地址保存低位，试设计一个程序实现这个长无符号数除以 8 的功能，运算结果仍然保存在数组 DA1 中。

参考答案（不唯一）

```
DATA    SEGMENT
DA1     DW    7B24H, F2BBH, 114FH, 6A82H
LEN     EQU   $-DA1
DATA    ENDS

CODE    SEGMENT
BEGIN: MOV    AX, DATA
        MOV    DS, AX
        MOV    CH, 3
LOP1:   MOV    CL, 3
        MOV    BX, LEN-2
        SHR    DA1[BX], 1
LOP2:   SUB    BX, 2
        RCL    DA1[BX], 1
        DEC    CL
        JNZ    LOP2
        DEC    CH
        JNZ    LOP1
        MOV    AH, 4CH
```

```
        INT     21H
CODE    ENDS
END     BEGIN
```

汇编语言程序设计

第 8 章作业答案

1. 试说明子程序调用指令与返回指令分别实现返回地址保存与恢复的原理。

答：子程序调用指令执行时，首先将返回地址入栈保存，若为段内调用，则仅将当前（IP）入栈保存；若为段间调用，则将当前（IP）、（CS）按照先后顺序入栈保存。保存返回地址后，调用指令会将程序执行流程转移到子程序入口地址，若为段内调用，则仅修改（IP），使（IP）指向子程序的第一条指令；若为段间调用，则修改（IP）、（CS），使它们指向子程序（位于与主程序不同的代码段）的第一条指令。总体上，子程序调用指令按顺序完成返回地址保存、程序流程转移两个步骤。若存在子程序嵌套调用，则按照调用顺序将各返回地址入栈保存，栈顶字单元（段内调用）或双字单元（段间调用）内是最近一次调用所保存的返回点。

子程序返回指令执行时，将使程序流程转向栈顶字或双字单元指示的返回地址，若为段内返回，则将栈顶字单元的内容出栈，传递至 IP 保存；若为段间返回，则将栈顶双字单元出栈后，按出栈顺序分别传递至 CS、IP 保存。总体上，子程序返回指令使程序流程返回到主程序中的返回点（CALL 指令后的那一条指令）。若存在子程序嵌套调用，则按照与调用顺序相反的顺序返回到各返回点。

以上原理也可以结合教材中例 8.1.2 加以说明。

2. 子程序参数传递方式可以分为哪几类？简述各类参数传递方式的原理。

答：子程序参数传递方式可以分为寄存器传递方式、堆栈传递方式、数据区传递方式三类。

寄存器传递方式是指主程序与子程序间约定使用特定寄存器传递特定参数，主程序在调用子程序前，先将入口参数传送到约定寄存器保存，然后再调用子程序；子程序执行时从约定寄存器取得入口参数，执行既定功能后，将出口参数传送到约定寄存器保存，然后才返回主程序，主程序则从约定寄存器取得出口参数。主、子程序约定用于传递参数的寄存器不属于 CPU 现场，子程序可不对它们实施保护，因为主、子程序都能明确它们各自的含义。

堆栈参数传递方式则是指主程序与子程序间约定使用堆栈传递入、出口参数，主程序调用子程序前，先将入口参数按照约定顺序入栈保存，然后再调用子程序；子程序执行时按照约定顺序从堆栈中取得入口参数，完成既定功能后，将出口参数按照约定顺序传递到堆栈中保存（一般是覆盖最先入栈的入口参数），然后子程序返回主程序，返回时常使用 RET N 格式的返回指令，用于清除栈顶的入口参数。

数据区传递方式是指主、子程序使用一块公共内存区域实施参数传递，主程序在调用子程序前先将入口参数按照约定顺序、类型保存至数据区，然后调用子程序，子程序执行时则按照约定顺序、类型从数据区取得入口参数，执行既定功能后，将出口参数按照约定顺序、类型保存至数据区，然后返回主程序，主程序则按照约定的顺序、类型从数据区取得出口参数。

各种参数传递传输方式可以在入口、出口参数传递中混合使用，但应注意主、子程序应当按照共同的参数传递约定来实施参数传递。

3. 试说明在子程序中保护 CPU 现场的必要性。

答：由于子程序与主程序的设计人员不一定是同一位程序员，主程序的设计者仅需了解子程序的参数传递约定，就可正确调用子程序。但若子程序不加保护的任意使用 CPU 内的寄存器，特别是与参数传递无关的寄存器，则会导致子程序修改了某一寄存器内容，而主程序认为该寄存器并无变化的情况，而这种情况通常会导致程序执行的逻辑错误。为了使子程序使用寄存器对主程序透明（主程序设计者不需要担心子程序会破坏某些寄存器内容），子程序设计者应养成良好习惯，在实施实质性操作前，将不用于参数传递、而又需要在子程序中使用的寄存器作为 CPU 现场加以保护，并在返回前加以恢复。

4. 题目见教材

参考答案（不唯一）：

```
；参数传递约定
；使用寄存器 B X 传递字符串起始偏移量，使用寄存器 C X 传递字符串长度

STRPROC1    PROC
            PUSH    AX              ;现场保护
LOP1:       MOV     AL, [BX]
            CMP     AL, ‘a’         ;判断当前字符是否为小写字符
            JAE     L1
            JMP     L3
L1:         CMP     AL, ‘z’
            JBE     L2
            JMP     L3
L2:         SUB     [BX], 20H       ;将小写字符转换为大写字符
L3:         INC     BX              ;地址指针指向下一字符
            LOOP    LOP1            ;计数循环，循环次数为字符串长度
            POP     AX              ;现场恢复
            RET
STRPROC1    ENDP
```

5. 题目见教材

参考答案（不唯一）：

```
；参数传递约定
；待处理数据地址（偏移量）、待处理数据类型（使用 8 与 16 分别表示字节与字类型）、
；位编号三个入口参数都通过堆栈进行传递，入栈顺序如前面给出的排列顺序

BITPROC1    PROC
            PUSH    BP              ；将 BP 作为现场进行保护
```

```
        MOV   BP, SP          ; 将 BP 指向当前栈顶
        PUSH  AX              ; 现场保护
        PUSH  BX
        PUSH  CX
        PUSH  DX
        MOV   BX, [BP+8]      ; 参数传递，取待处理数据偏移量，假定段基值为（DS）
        MOV   DX, [BP+6]      ; 参数传递，取待处理数据类型（8 或 16）
        MOV   CX, [BP+4]      ; 参数传递，取数据位编号
        CMP   DX, 8           ; 按照数据类型获取待处理数据
        JZ    L1
        JMP   L2
L1:     MOV   AL, BYTE PTR [BX]
        XOR   AH, AH
        JMP   L3
L2:     MOV   AX, WORD PTR [BX]
L3:     ROR   AX, CL                ; 将（AX）中的待处理数据位循环移位至第 0 位
        AND   AX, 0FFFEH            ; 将 AX 的第 0 位清零，其余位不影响
        ROL   AX, CL                ; 将各数据位恢复到原来的位置
        CMP   DX, 8                 ; 根据数据类型保存处理结果
        JZ    L4
        JMP   L5
L4:     MOV   BYTE PTR [BX], AL
        JMP   L6
L5:     MOV   WORD PTR [BX], AX
L6:     POP   DX              ; 现场恢复
        POP   CX
        POP   BX
        POP   AX
        POP   BP
        RET   6               ; 返回时清除入口参数
BITPROC1  ENDP
```

6. 题目见教材

参考答案（不唯一）：

```
; 参数传递说明
; 两个长补码起始地址（偏移量，指向补码最低位）分别使用 SI、DI 寄存器传递
; 补码长度（以字为单位）使用 CX 寄存器传递
; 运算类型采用‘+’与‘-’字符的 ASCII 码加以描述，使用 DL 寄存器传递
; 运算结果保存的起始地址（偏移量，指向补码最低位）使用 BX 寄存器传递

DATAPROC1  PROC
        PUSH  AX              ; 现场保护
        PUSHF
```

```
        CMP    DL, '+'
        JZ     L1
        JMP    L2
L1:     CLC                    ; 使循环体内第 1 条 ADC 或 SBB 指令等同于 ADD 或 SUB
LOP1:   MOV    AX, [SI]
        ADC    AX, [DI]
        MOV    [BX], AX
        PUSHF                  ; 保护 CF
        ADD    SI, 2
        ADD    DI, 2
        ADD    BX, 2
        POPF                   ; 恢复 CF
        LOOP   LOP1
        JMP    L3
L2:     CLC                    ; 使循环体内第 1 条 ADC 或 SBB 指令等同于 ADD 或 SUB
LOP2:   MOV    AX, [SI]
        SBB    AX, [DI]
        MOV    [BX], AX
        PUSHF                  ; 保护 CF
        ADD    SI, 2
        ADD    DI, 2
        ADD    BX, 2
        POPF                   ; 恢复 CF
        LOOP   LOP1
L3:     POPF                   ; 现场恢复
        POP    AX
        RET
DATAPROC1   ENDP
```

其余程序设计题目请同学们自行完成，并上机调试通过。