一、 实验目的

1、能够熟练掌握变量的定义，熟练掌握各种指令的用法和对标志位的影响。
2、（新学）能够基于上课的知识点掌握部分系统功能的调用。
3、能够熟练掌握强制类型转换，对变量的类型和属性操作符有较好的理解。
4、熟悉使用各种指令实现四则运算，以及除法操作时扩展指令。
5、能够根据类型的大小设计可扩展的程序，多使用 EQU，少使用硬编码的常数。

二、 实验设备

硬件环境：个人通用 PC
软件环境：Emu8086

三、 实验内容和要求

1、已知变量 DATAX 和 DATAY 的定义如下：

DATAX DW 9148H，8316H
DATAY DW 1237H，8252H
SUMWORD DW ？
SUMDWORD DW ？，？

要求编写程序段实现以下功能(其中数据均为无符号数)

（1）DATAX 和 DATAY 两个字数据相加，和存放在 SUMWORD 中。要求考虑溢出情况。如果溢出，给出提示信息并结束程序。

（2）DATAX 和 DATAY 两个双字数据相加，和存放在 SUMDWORD 开始的字单元中。要求考虑溢出情况。如果溢出，给出提示信息并结束程序。

2、实现要求同题目 1，只是将其中的数据看做有符号数。

3、已知 X、Y、Z、V 均为 16 位有符号数，且已分别装入 X,Y,Z,V 单元中,要求计算 (V-(X*Y+Z-540))/X 的结果，将商存入 AX,余数存入 DX 中。不考虑溢出。

四、 实验步骤

1、实验 1（方法 1）

设计思路：实现加法并存储的指令是比较简单的，多出的指令主要实现溢出位的判断以及调用系统中断程序。双字的程序和单字的运算只在运算时有所区别。

伪代码：

```
ax := datax
add ax, datay
mov sumword, ax
jnc after ; 如果没有溢出，跳过溢出显示的程序
```

```
lea dx, errormsg
mov ah, 9
int 21h ; 调用系统中断程序

after:
    ...
```

实验预期：

9148h+1237h=a37fh，对于无符号没有溢出，有符号没有溢出。

8316 9148h+8252 1237h=0568 a37fh，对于无符号没有溢出，有符号溢出。

实验代码：

（1）

```
data segment
    ; add your data here!
    pkey db "press any key...$"
    datax dw 9148h, 8316h
    datay dw 1237h, 8252h
    sumword dw ?
    sumdword dw ?, ?
    errormsg db "overflow exception...$"
ends

code segment
    ...
    mov ax, datax
    add ax, datay
    mov sumword, ax

    ; jump is not overflow
    jnc after

    lea dx, errormsg
    mov ah, 9
    int 21h ; call the system method (pring string)
after:
    ...
ends
```

（2）

```
data segment
    ; add your data here!
    pkey db "press any key...$"
    datax dw 9148h, 8316h
    datay dw 1237h, 8252h
    sumword dw ?
    sumdword dw ?, ?
    errormsg db "overflow exception...$"
    dwl equ 2
ends

code segment
    ...
    mov ax, datax
    add ax, datay
    mov dx, datax + dwl
    adc dx, datay + dwl
    mov sumdword + dwl, ax
    mov sumdword + dwl, dx

    ; jump is not overflow
    jnc after

    lea dx, errormsg
    mov ah, 9
    int 21h ; call the system method (pring string)
after:
    ...
ends
```

实验结果：

（1）

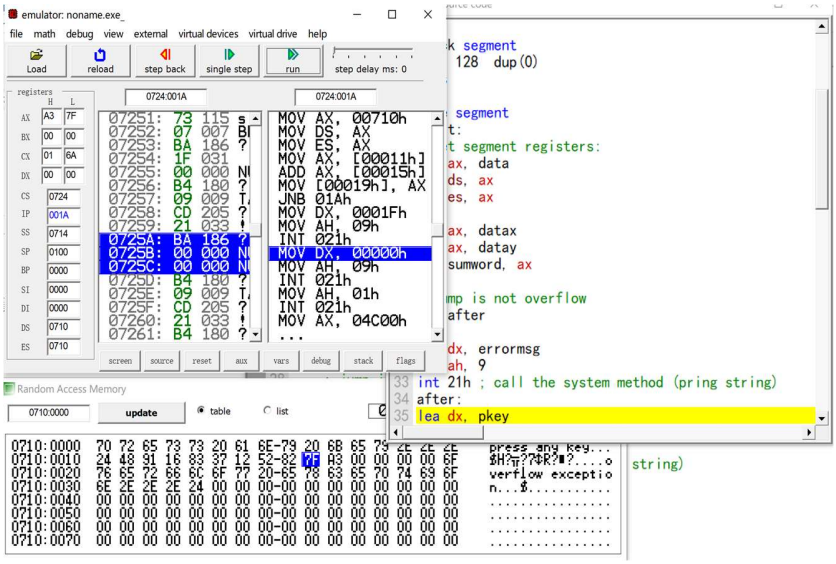

图 3-1 程序执行结果 3.1.1

（对应 sumword 的存储为 7f a3，符合预期，没有执行溢出处理代码段。）

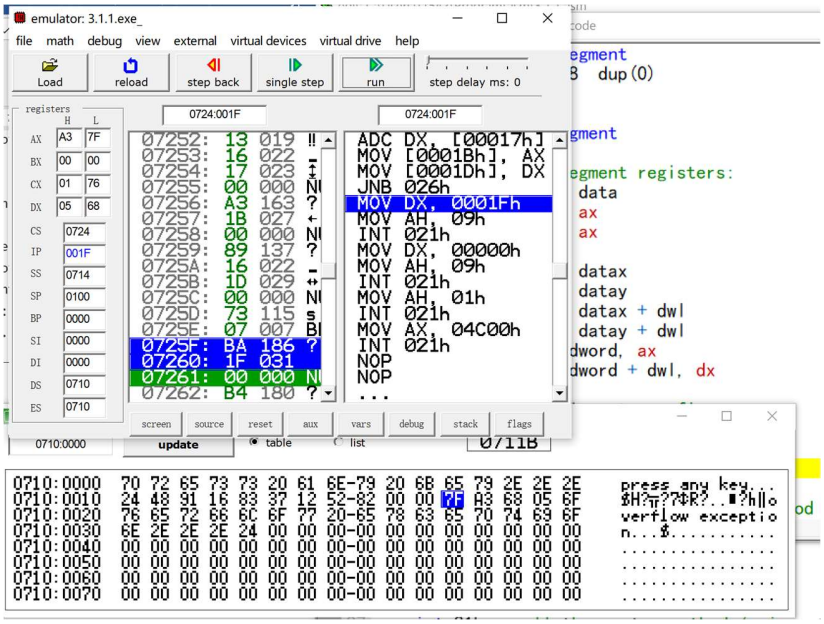

图 3-2 程序执行结果 3.1.2

（对应 sumword 的存储为 7f a3 68 05，符合预期，没有执行溢出处理代码段。）

2、实验 2

设计思路：和实验 1 的绝大部分代码相同，在使用双字节加法时，仍然使用 adc，只不过在判断有符号数是否溢出时，判断的时 OF 而不是 CF 位，所以只需要把 jnc 变成 jno 即可。

实验代码：和实验 1 过于相似，省略

实验结果：

（2）

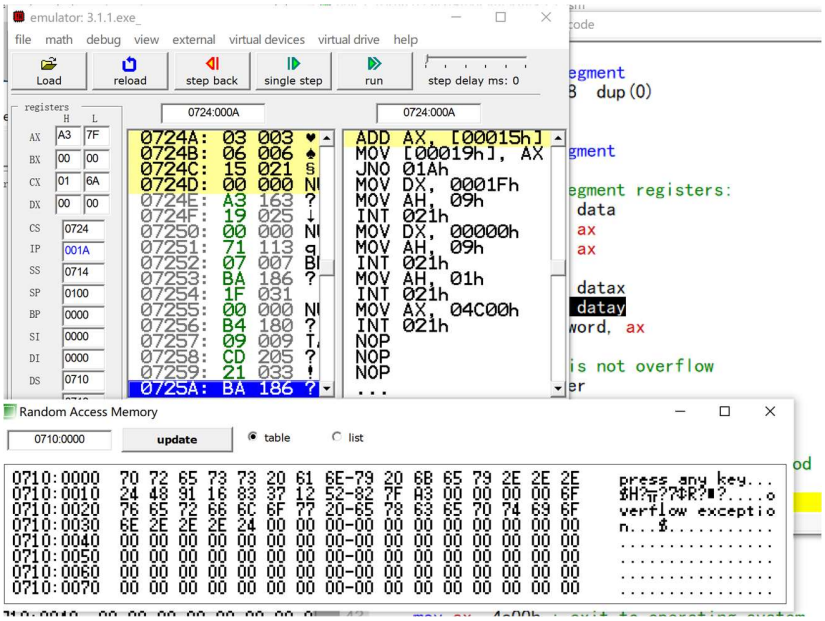

图 3-3 程序执行结果 3.2.1

（sumword 对应存储单元的内容位 7f a3，符合预期，没有执行溢出处理代码段。）

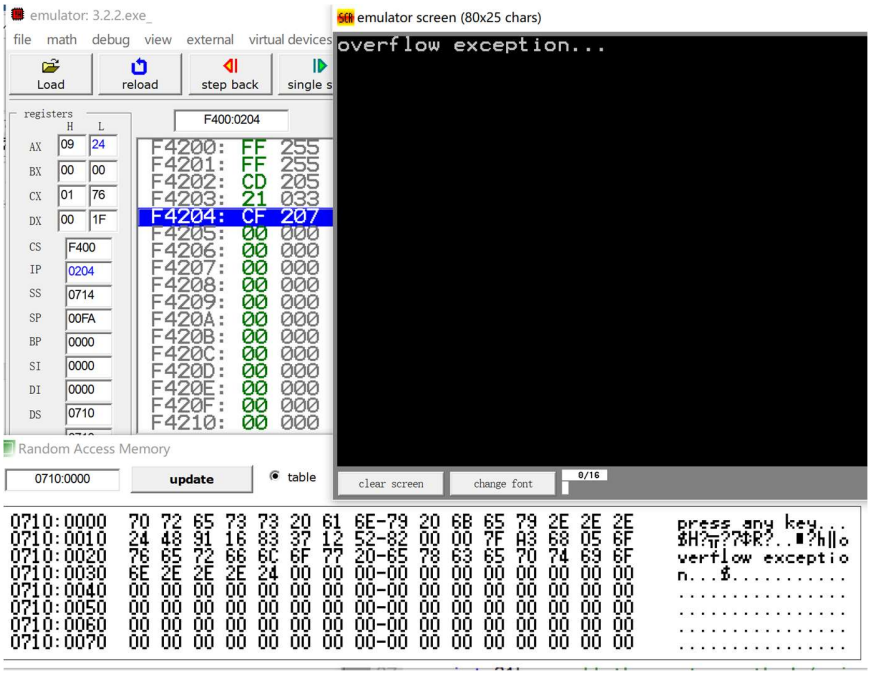

图 3-4 程序执行结果 3.2.2

（sumdword 存储单元的内容位 7f a3 68 05，符合预期，没有执行溢出处理代码段。）

3、实验 3

设计思路：本实验主要考察到数据类型和数据转换。两个字相乘会在 DX,AX 存储双字的结果。在进行加法操作时，自动向双字对齐。在计算运算时，自动使用 temp（字）保存中间的运行结果。

测试数据：x = 9274;y = 7381; z = -5621; v = 7321

预期结果：d = -7379(E3 26 H); r = -586(E9 CC H)

伪代码：

```
mov ax, x
imul y
; dx,ax store the x*y
add ax, z
adc dx, 0
sub ax, 540
sbb dx, 0
; add the x and sub the 540
mov temp, ax
mov temp + 2, dx
; temp store the temp result
mov ax, v
cwd
; mov v to ax and translate to the double word.
sub ax, temp
sbb dx, temp+2
; dx,ax store the last second result.
idiv x
```

实验代码：

```
mov ax, x
imul y
; dx,ax := x * y
add ax, z
adc dx, 0
; dx,ax := x * y + z
sub ax, 540
sbb dx, 0
; dx,ax := x * y + z - 540
mov temp, ax
```

```
mov temp + 2, dx
; temp := x * y + z - 540
mov ax, v
cwd
; dx,ax := v
sub ax, temp
sbb dx, temp + 2
; dx,ax := v - (x * y + z - 540)
idiv x
```

实验结果：

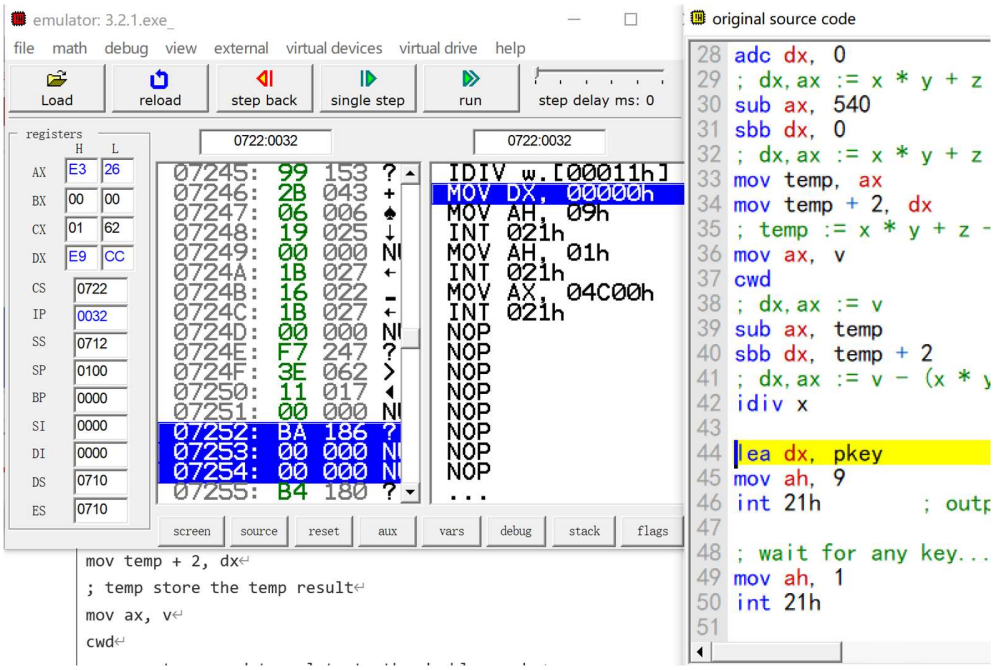

图 3-5 程序执行结果 3.2

（实验结果，计算结果符合预期，且没有溢出）

五、 实验结果分析

1、实验 1 至实验 3 均使用到了双字数字的运算，双字的运算和单字的运算并不相同，不能通过寄存器一步完成。在 8086 中，DX,AX 组合常用来表示双字的数据，而且和乘法关联的操作必须使用到这两个寄存器。这样设计是为了在汇编程序间建立一个规范，便于代码的编写和设计。

2、在双字加法和减法，中间为了传递进位必须要用到 CF，与两个相关联的命令为 ADC 和 SBB，在有符号数和无符号数中，都使用了这两个指令，因为负数使用了补码表示法，从而简化了减法运算。而乘法和除法用到了两套指令，无符号(MUL,DIV)和有符号(IMUL,IDIV)是两个不一样的运算符。

3、在进行除法的操作时，被除数必须为除数的两倍字宽。例如字/字节，双字/字，在代码中具体使用到哪种除法由 DIV/IDIV 的源操作数的类型决定。在使用字/字节时，AL 存储除法结果，AH 存储余数。双字/字时，AX 存储结果，DX 存储余数。在除法的结果为负数时，余数的结果一定为负数或者 0。及除数的结果和余数一定是同号的。

4、在执行乘法和除法其间，常常用到扩展的操作。CBW 和 CWD 仅适用于有符号数的操作，在扩展无符号数时，只要把高位变成 0 即可。

六、 结果讨论

1、汇编中的地址偏移和高级语言（这里指 C++）的指针地址偏移和数组偏移不同，汇编的偏移单位是实实在在的一个字节的偏移，而指针的偏移单位为类型所占的大小。所以在设计汇编语言时，偏移是一定要考虑到类型的大小，如果是一个字（2 个字节）则偏移单位为 2，一般通过 Type ?来编写可扩展的程序，但是由于 emu 8086 不支持这个伪指令，只能作罢。

2、如果要设计可扩展的程序（例如求数组的和），需要考虑到数组的大小和每个元素的类型，并多使用 EQU 来提高程序的扩展性，少使用硬编码常数。