



Microprocessor and Assembly Language Lab

Lab Material 6 for CSE 312 (M&AL Lab)

Dr. Shah Murtaza Rashid Al Masud

Associate Prof.

Dept. of CSE, UAP

MUL and DIV Operations

MUL Instruction

The MUL (unsigned multiply) instruction multiplies an 8, 16, or 32 bit operand by either AL, AX, or EAX. The instruction formats are:

MUL *r/m8*

MUL *r/m16*

MUL *r/m32*

The single operand is the multiplier. The following table shows the default multiplicand and product, depending on the size of the multiplier:

MUL Instruction

Multiplicand	Multiplier	Product
AL	<i>r/m8</i>	AX
AX	<i>r/m16</i>	DX:AX
EAX	<i>r/m32</i>	EDX:EAX

The register(s) holding the product are **twice the size of the multiplicand and multiplier**, guaranteeing that overflow will never occur.

MUL Instruction

Example 1: The following statements perform 8-bit unsigned multiplication ($5 * 10h$), producing 50h in AX:

```
mov al,5h    ; AL=05H
```

```
mov bl,10h   ; BL=10H
```

```
mul bl                ; AL*BL=5H*10H=50H→AX  CF = 0
```

The Carry flag is clear because AH (the upper half of the product) equals zero.

MUL Instruction

```
org 100h  
  
mov al, 5h ; AL=05H  
mov bl, 10h ; BL=10H  
mul bl  
  
ret
```

emulator: noname.com_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	50
BX	00	10
CX	00	07
DX	00	00
CS	0700	
IP	0106	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	

0700:0106		0700:0106	
07100:	B0 176	MOV	AL, 05h
07101:	05 005	MOV	BL, 010h
07102:	B3 179	MUL	BL
07103:	10 016	RET	
07104:	F6 246	NOP	
07105:	E3 227	NOP	
07106:	C3 195	NOP	
07107:	90 144	NOP	
07108:	90 144	NOP	
07109:	90 144	NOP	
0710A:	90 144	NOP	
0710B:	90 144	NOP	
0710C:	90 144	NOP	
0710D:	90 144	NOP	
0710E:	90 144	NOP	
0710F:	90 144	NOP	
07110:	90 144	NOP	
07111:	90 144	NOP	

original source co...

```
01  
02 ; You may customize this  
03 ; The location of this t  
04  
05 org 100h  
06  
07 mov al, 5h ; AL=05H  
08 mov bl, 10h ; BL=10H  
09 mul bl  
10  
11  
12 ret  
13  
14  
15  
16  
17  
18  
19
```

flags

CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

analyse

MUL Instruction

```
.MODEL SMALL
.STACK 100H
.DATA

.CODE SEGMENT
MAIN PROC

    MOV AX,10H
    MOV BX,05H
    MUL BX

    MOV AX,4C00H
    INT 21H

MAIN ENDP
END MAIN
```

emulator: mul_v1.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms:

registers

	H	L
AX	00	05
BX	00	05
CX	01	00
DX	00	00
CS	0720	
IP	0008	
SS	0710	

0720:0008

07200: B8 184 3
07201: 10 016 2
07202: 00 000 NULL
07203: BB 187 1
07204: 05 005 4
07205: 00 000 NULL
07206: F7 247 8
07207: E3 227 11
07208: B8 184 3
07209: 00 000 NULL
0720A: 4C 076 L
0720B: CD 205 =
0720C: 21 033 !

MOV AX, 00010h
MOV BX, 00005h
MUL BX
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP

First operand is multiplicand and the second one is multiplier. Result is stored in AX

MUL Instruction

```
.MODEL SMALL
.STACK 100H
.DATA

.CODE SEGMENT
MAIN PROC

    MOV AX,1010H
    MOV BX,2020H

    MUL BX

    MOV AX,4C00H
    INT 21H

MAIN ENDP
END MAIN
```

emulator: mul_v3.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	02	00
BX	20	20
CX	01	00
DX	02	04
CS	0720	
IP	0008	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0720:0008

0720:0008

MOV AX, 01010h
MOV BX, 02020h
MUL BX
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...

07200: B8 184
07201: 10 016
07202: 10 016
07203: BB 187
07204: 20 032 SPA
07205: 20 032 SPA
07206: F7 247
07207: E3 227
07208: B8 184
07209: 00 000 NULL
0720A: 4C 076 L
0720B: CD 205 =
0720C: 21 033
0720D: 90 144
0720E: 90 144
0720F: 90 144
07210: 90 144
07211: 90 144
07212: 90 144
07213: 90 144
07214: 90 144
07215: 90 144

screen source reset aux vars debug

flags

CF	1
ZF	0
SF	0
OF	1
PF	0
AF	0
IF	1
DF	0

First operand is multiplicand and the second one is multiplier. Result is stored in AX

MUL Instruction

Example 2: The following statements perform 16-bit unsigned multiplication (0010h * 2000h) producing 00020000h in DX:AX:

```
. data
```

```
val1 WORD 2000h
```

```
val2 WORD 0010h
```

```
.code
```

```
mov ax, val1; AX=2000H
```

```
mul val2      ;
```

AX*val2=2000H*0010H=00020000h→DX:AX CF = 1

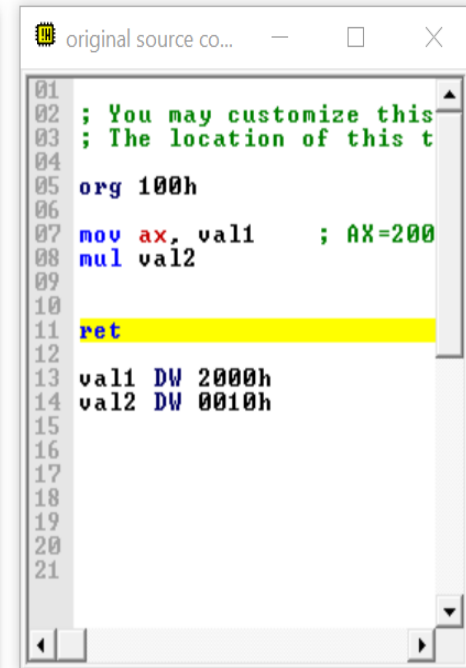
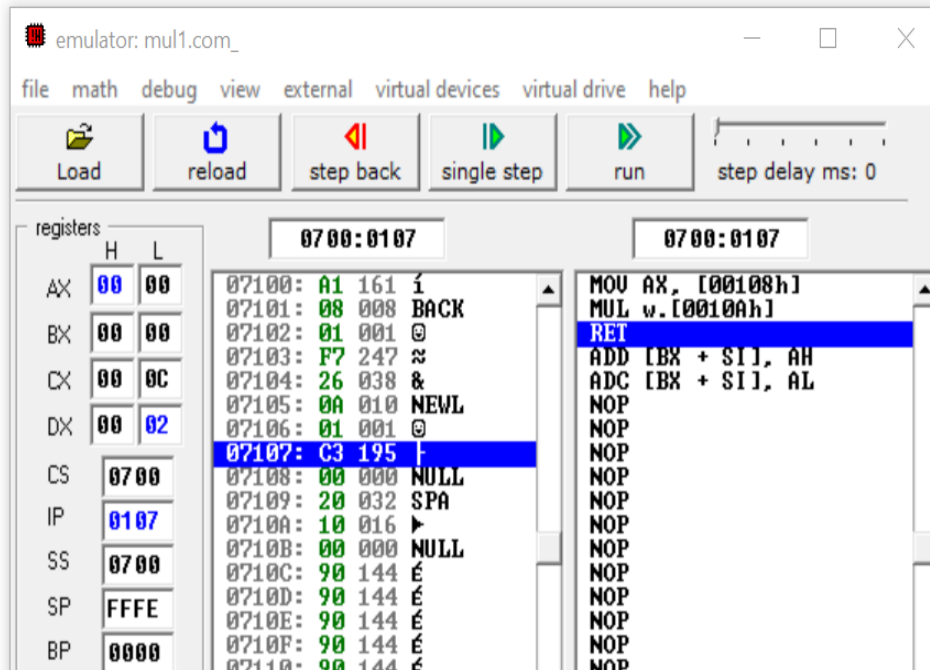
The Carry flag is set because DX is not equal to zero.

MUL Instruction

```
org 100h
mov ax, val1      ; AX=2000H
mul val2

ret

val1 DW 2000h
val2 DW 0010h
```



MUL Instruction (MOV CX, 0000H)

```
.MODEL SMALL
.STACK 100H
.DATA

val1 DW 1010h
val2 DW 2020h

.CODE SEGMENT
MAIN PROC

    MOV AX, @DATA
    MOV DS, AX

    MOV AX, val1
    MOV BX, val2

    MUL BX

    MOV AX, 4C00H
    INT 21H

MAIN ENDP
END MAIN
```

emulator: mul_v4_data_segment.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	02	00
BX	20	20
CX	01	23
DX	02	04
CS	0721	
IP	000E	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0720	
ES	0700	

0721:000E

07210:	B8	184	?
07211:	20	032	SPA
07212:	07	007	BEEP
07213:	8E	142	ä
07214:	D8	216	1
07215:	A1	161	1
07216:	00	000	NULL
07217:	00	000	NULL
07218:	8B	139	i
07219:	1E	030	▲
0721A:	02	002	0
0721B:	00	000	NULL
0721C:	F7	247	≈
0721D:	E3	227	Π
0721E:	B8	184	?
0721F:	00	000	NULL
07220:	4C	076	L
07221:	CD	205	=
07222:	21	033	!
07223:	90	144	é
07224:	90	144	é
07225:	90	144	é

0721:000E

```
MOV AX, 00720h
MOV DS, AX
MOV AX, [00000h]
MOV BX, [00002h]
MUL BX
MOV AX, 04C00h
INT 021h
```

flags

CF	1
ZF	0
SF	0
OF	1
PF	0
AF	0
IF	1
DF	0

screen source reset aux vars debug

MUL Instruction

Example 2: The following statements perform 16-bit unsigned multiplication (0011h * 2000h) producing 00022000h in DX:AX:

. data

val1 WORD 2000h

val2 WORD 0011h

.code

mov ax, val1 ; AX=2000H

mul val2 ; AX*val2=2000H*0011H=00021000h→DX:AX

CF = 1

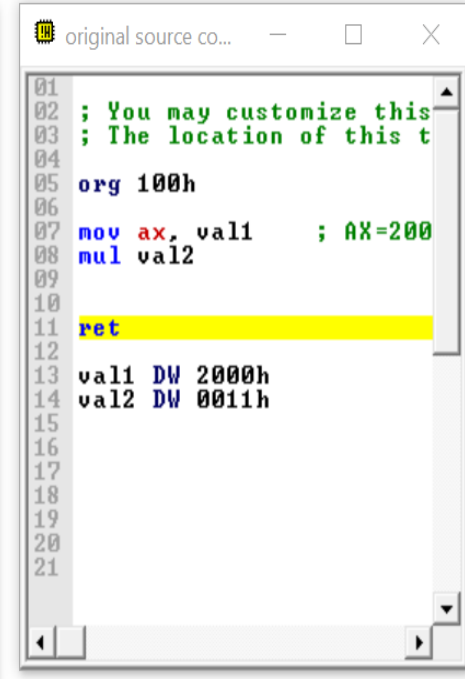
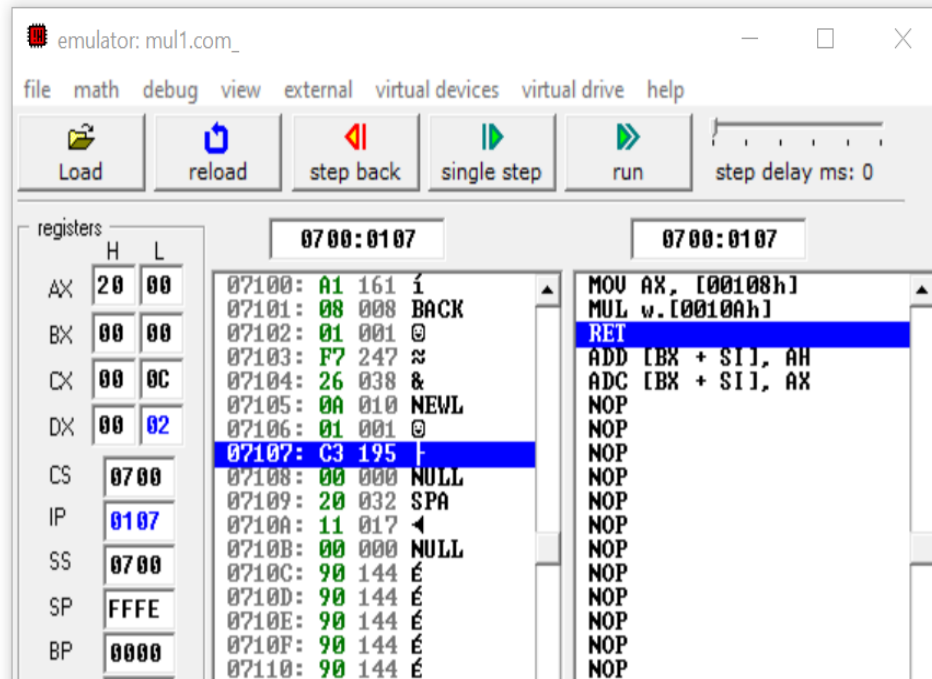
The Carry flag is set because DX is not equal to zero.

MUL Instruction

```
org 100h
mov ax, val1      ; AX=2000H
mul val2

ret

val1 DW 2000h
val2 DW 0011h
```



MUL Instruction

Another example with different operand

```
org 100h

mov ax, val1    ; AX=2000H
mul val2

ret

val1 DW 2222h
val2 DW 1111h
```

MUL Instruction

```
.MODEL SMALL
```

```
.STACK 100H
```

```
.DATA
```

```
val1 DW 1010h
```

```
val2 DW 2020h
```

```
.CODE SEGMENT
```

```
MAIN PROC
```

```
MOV AX, @DATA  
MOV DS, AX
```

```
MOV AX, val1  
MUL val2
```

```
MOV AX, 4C00H  
INT 21H
```

```
MAIN ENDP
```

```
END MAIN
```

emulator: mul_v4_data_segment2.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	02	00
BX	00	00
CX	01	21
DX	02	04
CS	0721	
IP	000C	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0720	
ES	0700	

0721:000C

07210:	B8	184	?
07211:	20	032	SPA
07212:	07	007	BEEP
07213:	8E	142	ä
07214:	D8	216	½
07215:	A1	161	¡
07216:	00	000	NULL
07217:	00	000	NULL
07218:	F7	247	≈
07219:	26	038	&
0721A:	02	002	0
0721B:	00	000	NULL
0721C:	B8	184	?
0721D:	00	000	NULL
0721E:	4C	076	L
0721F:	CD	205	=
07220:	21	033	!
07221:	90	144	é
07222:	90	144	é
07223:	90	144	é
07224:	90	144	é
07225:	90	144	é

0721:000C

```
MOV AX, 00720h  
MOV DS, AX  
MOV AX, [00000h]  
MUL w.[00002h]  
MOV AX, 04C00h  
INT 021h  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
...
```

flags

CF	1
ZF	0
SF	0
OF	1
PF	0
AF	0
IF	1

original source co...

```
val1 DW 1010h
```

```
val2 DW 2020h
```

```
.CODE SEGMENT
```

```
MAIN PROC
```

```
MOV AX, @DATA  
MOV DS, AX
```

```
MOV AX, val1  
MUL val2
```

```
MOV AX, 4C00H  
INT 21H
```

DIV Instruction

The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit division on unsigned integers. A single operand is supplied (register or memory operand), which is assumed to be the divisor. The instruction formats for DIV are:

DIV r/m8

DIV r/m16

DIV r/m32

The following table shows the relationship between the dividend, divisor, quotient, and remainder.

Everything is determined by the size of the **divisor**:

DIV Instruction

The following table shows the relationship between the dividend, divisor, quotient, and remainder. Everything is determined by the size of the **divisor**:

Dividend	Divisor	Quotient	Remainder
AX	<i>r/m8</i>	AL	AH
DX:AX	<i>r/m16</i>	AX	DX
EDX:EAX	<i>r/m32</i>	EAX	EDX

DIV Instruction

Example 1: The following instructions perform 8-bit unsigned division (83h / 2), producing a quotient of 41h and a remainder of 1:

```
mov ax,0083h          ; AX=0083H
mov bl,2              ; BL=02H
div bl                ; AX/BL=0083H/02H→AL=41H, AH=01H
```

First operand is dividend and the second one is divisor.

Quotient or Result is stored in AL and the Remainder in AH

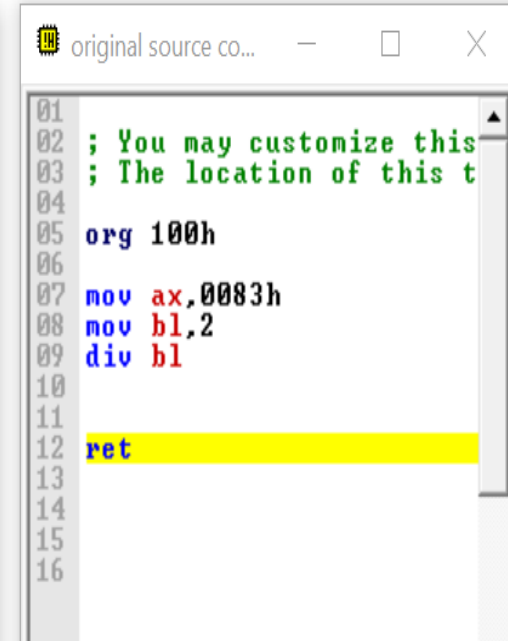
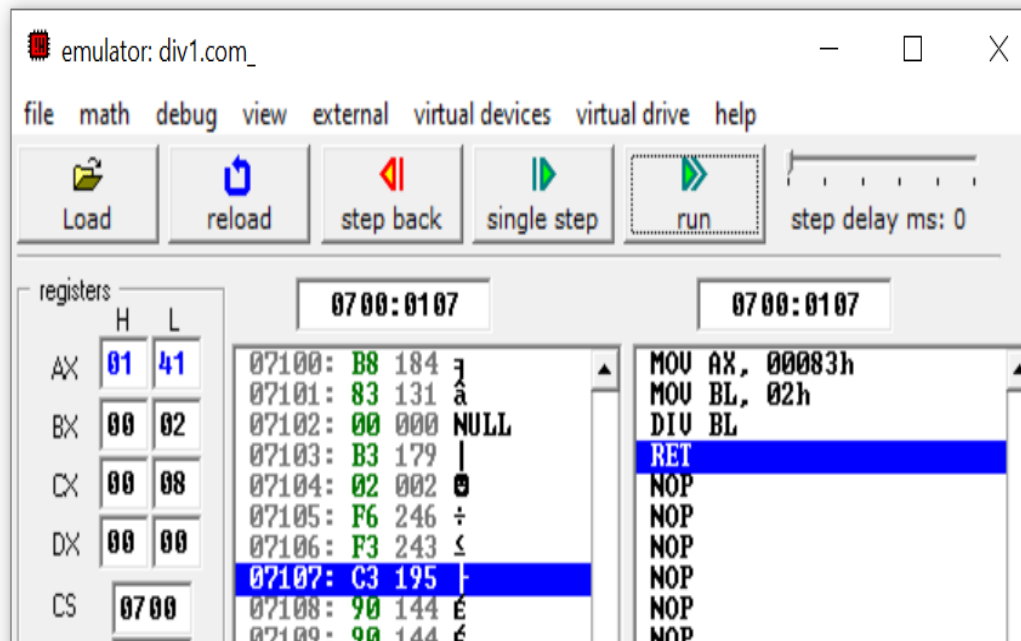
DIV Instruction

```
; You may customize this and other start-up templates;  
; The location of this template is c:\emu8086\inc\0_com_template.txt
```

```
org 100h
```

```
mov ax,0083h  
mov bl,2  
div bl
```

```
ret
```



DIV Instruction

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE SEGMENT
MAIN PROC
    mov ax,0083h
    mov bl,2
    div bl

    MOV AX,4C00H
    INT 21H

    MAIN ENDP
END MAIN
```

emulator: div1.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	01	41
BX	00	02
CX	01	0C
DX	00	00
CS	0720	
IP	0007	

0720:0007

07200:	B8	184	ɿ
07201:	83	131	ã
07202:	00	000	NULL
07203:	B3	179	
07204:	02	002	0
07205:	F6	246	÷
07206:	F3	243	≤
07207:	B8	184	ɿ
07208:	00	000	NULL
07209:	4C	076	L
0720A:	CD	205	=

0720:0007

```
MOV AX, 00083h
MOV BL, 02h
DIV BL
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
```

DIV Instruction

Example 2: The following instructions perform 16-bit unsigned division (8003h / 100h), producing a quotient of 80h and a remainder of 3.

DX contains the high part of the dividend, so it must be cleared before the DIV instruction executes:

```
mov dx,0           ; DX=0
mov ax,8003h       ; AX=8003H
mov cx,100h        ; CX=100H
div cx             ; AX=0800H DX=0003H
```

DIV Instruction

```
org 100h

mov dx,0      ; DX=0
mov ax,8003h  ; AX=8003H
mov cx,100h   ; CX=100H
div cx

ret
```

emulator: div2.com_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	80
BX	00	00
CX	01	00
DX	00	03
CS	0700	
IP	010B	
SS	0700	
SP	FFFE	

0700:010B

07100:	BA	186	
07101:	00	000	NULL
07102:	00	000	NULL
07103:	B8	184	?
07104:	03	003	▼
07105:	80	128	C
07106:	B9	185	
07107:	00	000	NULL
07108:	01	001	@
07109:	F7	247	≈
0710A:	F1	241	±
0710B:	C3	195	
0710C:	90	144	É
0710D:	90	144	É
0710E:	90	144	É
0710F:	90	144	É

MOV DX, 00000h
MOV AX, 08003h
MOV CX, 00100h
DIV CX
RET
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP

original source co...

```
01  
02 ; You may customize this  
03 ; The location of this t  
04  
05 org 100h  
06  
07 mov dx,0      ; DX=0  
08 mov ax,8003h  ; AX  
09 mov cx,100h   ; CX=  
10 div cx  
11  
12  
13 ret  
14  
15  
16  
17
```

flags

CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

analyse

DIV Instruction

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE SEGMENT
MAIN PROC
    mov dx,0      ; DX=0
    mov ax,8003h
    mov cx,100h
    div cx

    MOV AX,4C00H
    INT 21H

    MAIN ENDP
END MAIN
```

emulator: DIV 3.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	80
BX	00	00
CX	01	00
DX	00	03
CS	0720	
IP	000B	
SS	0710	
SP	0100	

0720:000B

07200:	BA	186	
07201:	00	000	NULL
07202:	00	000	NULL
07203:	B8	184	7
07204:	03	003	♥
07205:	80	128	C
07206:	B9	185	11
07207:	00	000	NULL
07208:	01	001	0
07209:	F7	247	≈
0720A:	F1	241	±
0720B:	B8	184	7
0720C:	00	000	NULL
0720D:	4C	076	L
0720E:	CD	205	=

0720:000B

MOV DX, 00000h
MOV AX, 08003h
MOV CX, 00100h
DIV CX
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP

First operand is dividend and the second one is divisor.

Quotient or Result is stored in AX and the Remainder in DX