

<p>ADC</p>	<p>REG, memory memory, REG REG, REG memory, immediate REG, immediate</p>	<p>Add with Carry.</p> <p>Algorithm:</p> <p>$operand1 = operand1 + operand2 + CF$</p> <p>Example: STC ; set CF = 1 MOV AL, 5 ; AL = 5 ADC AL, 1 ; AL = 7 RET</p> <table border="1" data-bbox="746 611 948 712"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td> </tr> <tr> <td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td> </tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
<p>ADD</p>	<p>REG, memory memory, REG REG, REG memory, immediate REG, immediate</p>	<p>Add.</p> <p>Algorithm:</p> <p>$operand1 = operand1 + operand2$</p> <p>Example: MOV AL, 5 ; AL = 5 ADD AL, -3 ; AL = 2 RET</p> <table border="1" data-bbox="746 1189 948 1290"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td> </tr> <tr> <td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td> </tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									

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	02
BX	00	00
CX	00	05
DX	00	00
CS	0700	
IP	0000	
SS	0700	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0700:0000 0700:0000

07000: CD 205 =	INT 020h
07001: 20 032 SPA	ADD LBX + SI, AL
07002: 00 000 NULL	ADD LBX + SI, AL
07003: 00 000 NULL	ADD LBX + SI, AL
07004: 00 000 NULL	ADD LBX + SI, AL
07005: 00 000 NULL	ADD LBX + SI, AL
07006: 00 000 NULL	ADD LBX + SI, AL
07007: 00 000 NULL	ADD LBX + SI, AL
07008: 00 000 NULL	ADD LBX + SI, AL
07009: 00 000 NULL	ADD LBX + SI, AL
0700A: 00 000 NULL	ADD LBX + SI, AL
0700B: 00 000 NULL	ADD LBX + SI, AL
0700C: 00 000 NULL	ADD LBX + SI, AL
0700D: 00 000 NULL	ADD LBX + SI, AL
0700E: 00 000 NULL	ADD LBX + SI, AL
0700F: 00 000 NULL	ADD LBX + SI, AL
07010: 00 000 NULL	ADD LBX + SI, AL
07011: 00 000 NULL	ADD LBX + SI, AL
07012: 00 000 NULL	ADD LBX + SI, AL
07013: 00 000 NULL	ADD LBX + SI, AL
07014: 00 000 NULL	ADD LBX + SI, AL
07015: 00 000 NULL	...

screen source reset aux vars debug stack flags

original source code

```

01 ORG 100h
02 MOV AL, 5 ; AL = 5
03 ADD AL, -3 ; AL = 2
04 RET
05
06
07
08
09
10
11
    
```

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	07
BX	00	00
CX	00	06
DX	00	00
CS	F400	
IP	0154	
SS	0700	
SP	FFFA	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

F400:0154 F400:0154

F4150: FF 255 RES	BIOS DI
F4151: FF 255 RES	INT 020h
F4152: CD 205 =	I RET
F4153: 20 032 SPA	ADD LBX + SI, AL
F4154: CF 207 ±	ADD LBX + SI, AL
F4155: 00 000 NULL	ADD LBX + SI, AL
F4156: 00 000 NULL	ADD LBX + SI, AL
F4157: 00 000 NULL	ADD LBX + SI, AL
F4158: 00 000 NULL	ADD LBX + SI, AL
F4159: 00 000 NULL	ADD BH, BH
F415A: 00 000 NULL	DEC BP
F415B: 00 000 NULL	SBB CL, BH
F415C: 00 000 NULL	ADD LBX + SI, AL
F415D: 00 000 NULL	ADD LBX + SI, AL
F415E: 00 000 NULL	ADD LBX + SI, AL
F415F: 00 000 NULL	ADD LBX + SI, AL
F4160: FF 255 RES	ADD BH, BH
F4161: FF 255 RES	DEC BP
F4162: CD 205 =	ADD BH, CL
F4163: 1A 026 →	ADD LBX + SI, AL
F4164: CF 207 ±	ADD LBX + SI, AL
F4165: 00 000 NULL	...

screen source reset aux vars debug stack flags

original source code

```

01 ORG 100h
02 STC ; set CF = 1
03 MOV AL, 5 ; AL = 5
04 ADC AL, 1 ; AL = 7
05 RET
06
07
08
09
10
11
    
```

LEA

REG,
memory

Load Effective Address.

Algorithm:

- REG = address of memory (offset)

Generally this instruction is replaced by MOV when assembling when possible.

Example:

ORG 100h

LEA AX, m

RET

m DW 1234h

END

AX is set to: 0104h.

LEA instruction takes 3 bytes, RET takes 1 byte, we start at 100h, so the address of 'm' is 104h.

C	Z	S	O	P	A
unchanged					

The screenshot shows an emulator window with the following details:

- Assembly Window:**

```

01 ORG 100h
02
03 LEA AX, m
04
05 RET
06
07 m DW 1234h
08
09 END
10
11
12
13
14

```
- Registers Window:**

registers		0700:0100		0700:0100	
AX	00 00	07100: B8 184	↓	MOU AX, 00104h	↑
BX	00 00	07101: 04 004	↓	RET	↑
CX	00 06	07102: 01 001	⊙	XOR AL, 012h	↑
DX	00 00	07103: C3 195	↑	NOP	↑
CS	0700	07104: 34 052	4	NOP	↑
IP	0100	07105: 12 018	↓	NOP	↑
SS	0700	07106: 90 144	É	NOP	↑
SP	FFFE	07107: 90 144	É	NOP	↑
BP	0000	07108: 90 144	É	NOP	↑
SI	0000	07109: 90 144	É	NOP	↑
DI	0000	0710A: 90 144	É	NOP	↑
DS	0700	0710B: 90 144	É	NOP	↑
ES	0700	0710C: 90 144	É	NOP	↑
		0710D: 90 144	É	NOP	↑
		0710E: 90 144	É	NOP	↑
		0710F: 90 144	É	NOP	↑
		07110: 90 144	É	NOP	↑
		07111: 90 144	É	NOP	↑
		07112: 90 144	É	NOP	↑
		07113: 90 144	É	NOP	↑
		07114: 90 144	É	NOP	↑
		07115: 90 144	É	...	↑

DEC

REG
memory

Decrement.

Algorithm:

operand = operand - 1

Example:

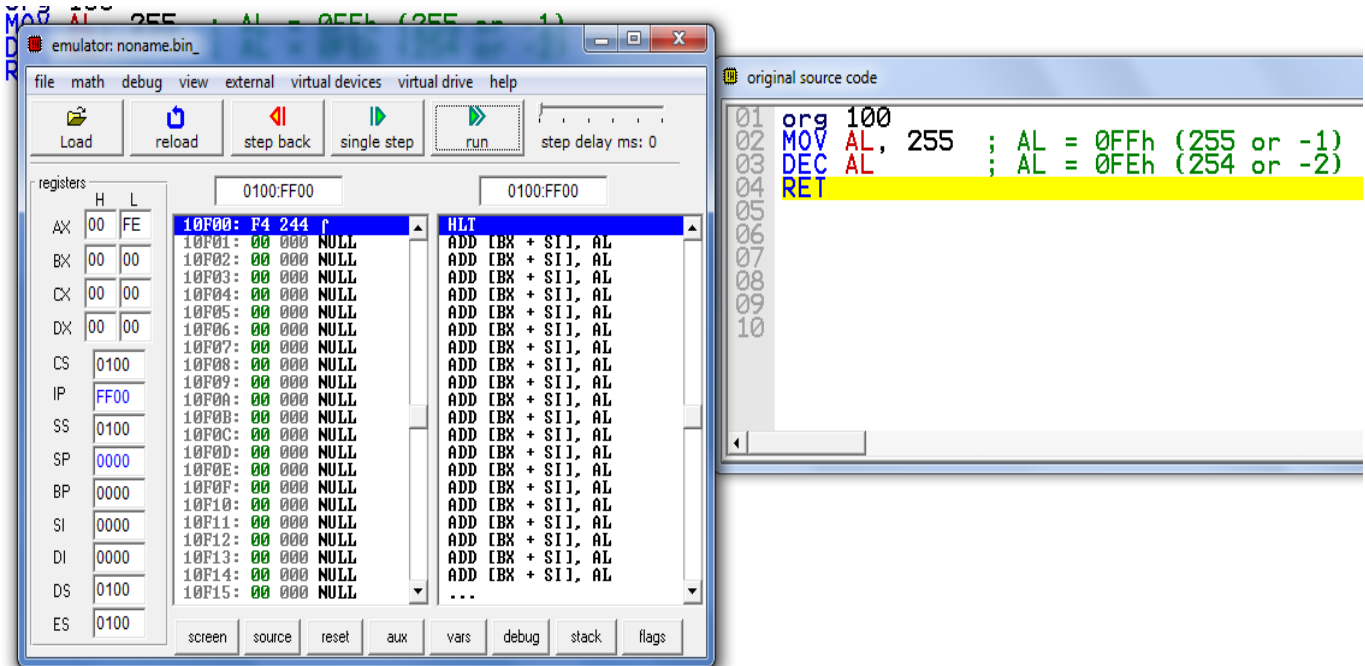
MOV AL, 255 ; AL = 0FFh (255 or -1)

DEC AL ; AL = 0FEh (254 or -2)

RET

Z	S	O	P	A
r	r	r	r	r

CF - unchanged!



INC

REG
memory

Increment.

Algorithm:

operand = operand + 1

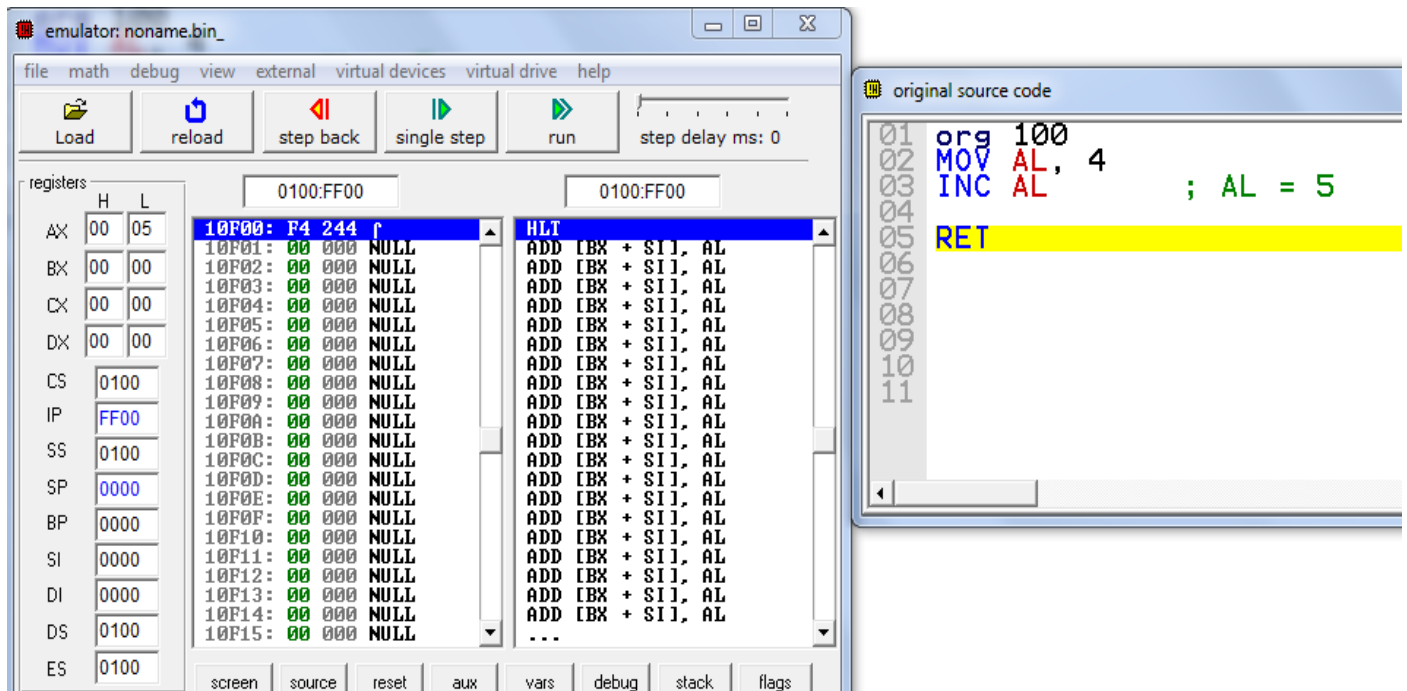
Example:

```

MOV AL, 4
INC AL ; AL = 5
RET
            
```

Z	S	O	P	A
r	r	r	r	r

CF - unchanged!



SUB

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

Subtract.

Algorithm:

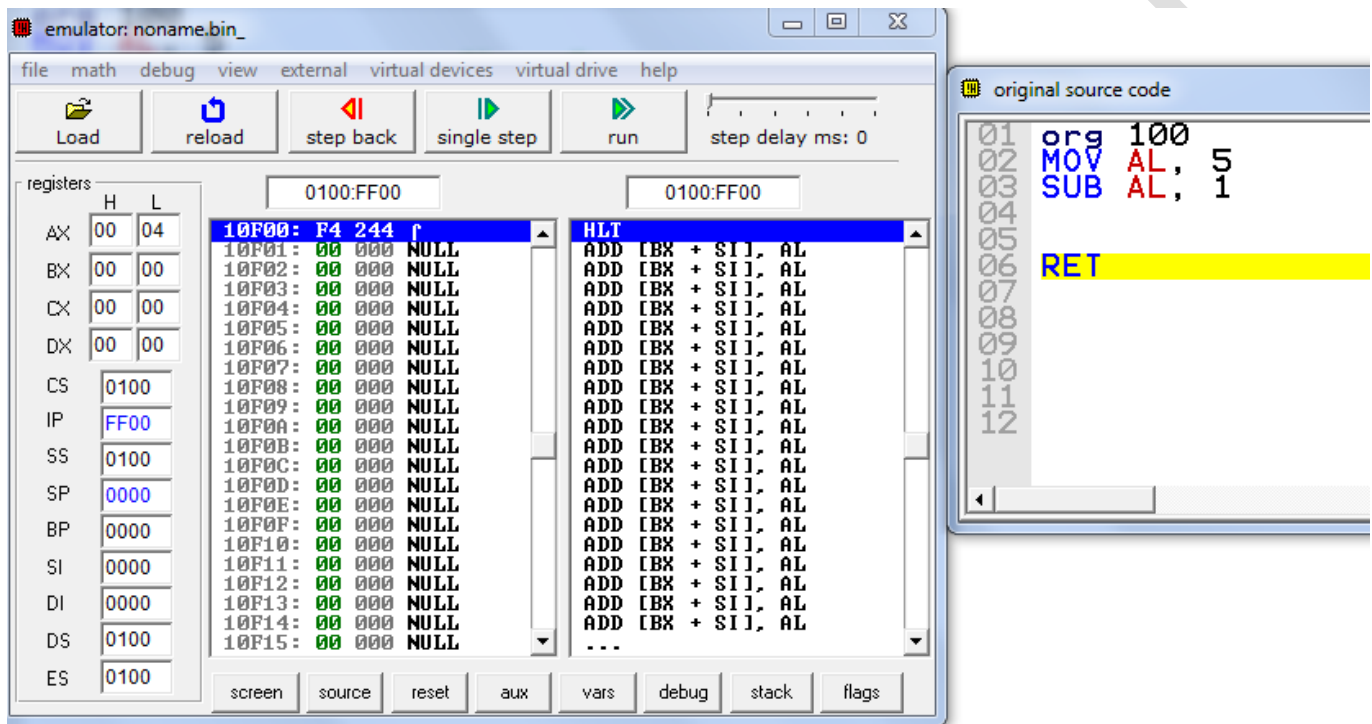
operand1 = operand1 - operand2

Example:

MOV AL, 5
SUB AL, 1 ; AL = 4

RET

C	Z	S	O	P	A
r	r	r	r	r	r



SBB

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

Subtract with Borrow.

Algorithm:

operand1 = operand1 - operand2 - CF

Example:

STC
MOV AL, 5
SBB AL, 3 ; AL = 5 - 3 - 1 = 1

RET

C	Z	S	O	P	A
r	r	r	r	r	r

emulator: noname.bin

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	01
BX	00	00
CX	00	00
DX	00	00
CS	0100	
IP	FF00	
SS	0100	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

0100:FF00 0100:FF00

Address	Hex	Comment	Instruction
10F00:	F4 244	↑	HLT
10F01:	00 000	NULL	ADD [BX + SI], AL
10F02:	00 000	NULL	ADD [BX + SI], AL
10F03:	00 000	NULL	ADD [BX + SI], AL
10F04:	00 000	NULL	ADD [BX + SI], AL
10F05:	00 000	NULL	ADD [BX + SI], AL
10F06:	00 000	NULL	ADD [BX + SI], AL
10F07:	00 000	NULL	ADD [BX + SI], AL
10F08:	00 000	NULL	ADD [BX + SI], AL
10F09:	00 000	NULL	ADD [BX + SI], AL
10F0A:	00 000	NULL	ADD [BX + SI], AL
10F0B:	00 000	NULL	ADD [BX + SI], AL
10F0C:	00 000	NULL	ADD [BX + SI], AL
10F0D:	00 000	NULL	ADD [BX + SI], AL
10F0E:	00 000	NULL	ADD [BX + SI], AL
10F0F:	00 000	NULL	ADD [BX + SI], AL
10F10:	00 000	NULL	ADD [BX + SI], AL
10F11:	00 000	NULL	ADD [BX + SI], AL
10F12:	00 000	NULL	ADD [BX + SI], AL
10F13:	00 000	NULL	ADD [BX + SI], AL
10F14:	00 000	NULL	ADD [BX + SI], AL
10F15:	00 000	NULL	...

screen source reset aux vars debug stack flags

original source code

```
01 org 100
02 STC
03 MOV AL, 5
04 SBB AL, 3 ; AL = 5 - 3 - 1 = 1
05
06
07
08 RET
09
10
11
12
13
14
```

Eng. Younis

NEG

REG
memory

Negate. Makes operand negative (two's complement).

Algorithm:

- Invert all bits of the operand
- Add 1 to inverted operand

Example:

```
MOV AL, 5 ; AL = 05h
NEG AL ; AL = 0FBh (-5)
NEG AL ; AL = 05h (5)
RET
```

C	Z	S	O	P	A
r	r	r	r	r	r

The screenshot shows an emulator window titled 'emulator: noname.bin_'. The interface includes a menu bar (file, math, debug, view, external, virtual devices, virtual drive, help) and a toolbar with buttons for Load, reload, step back, single step, and run. A progress bar for 'step delay ms: 0' is also present.

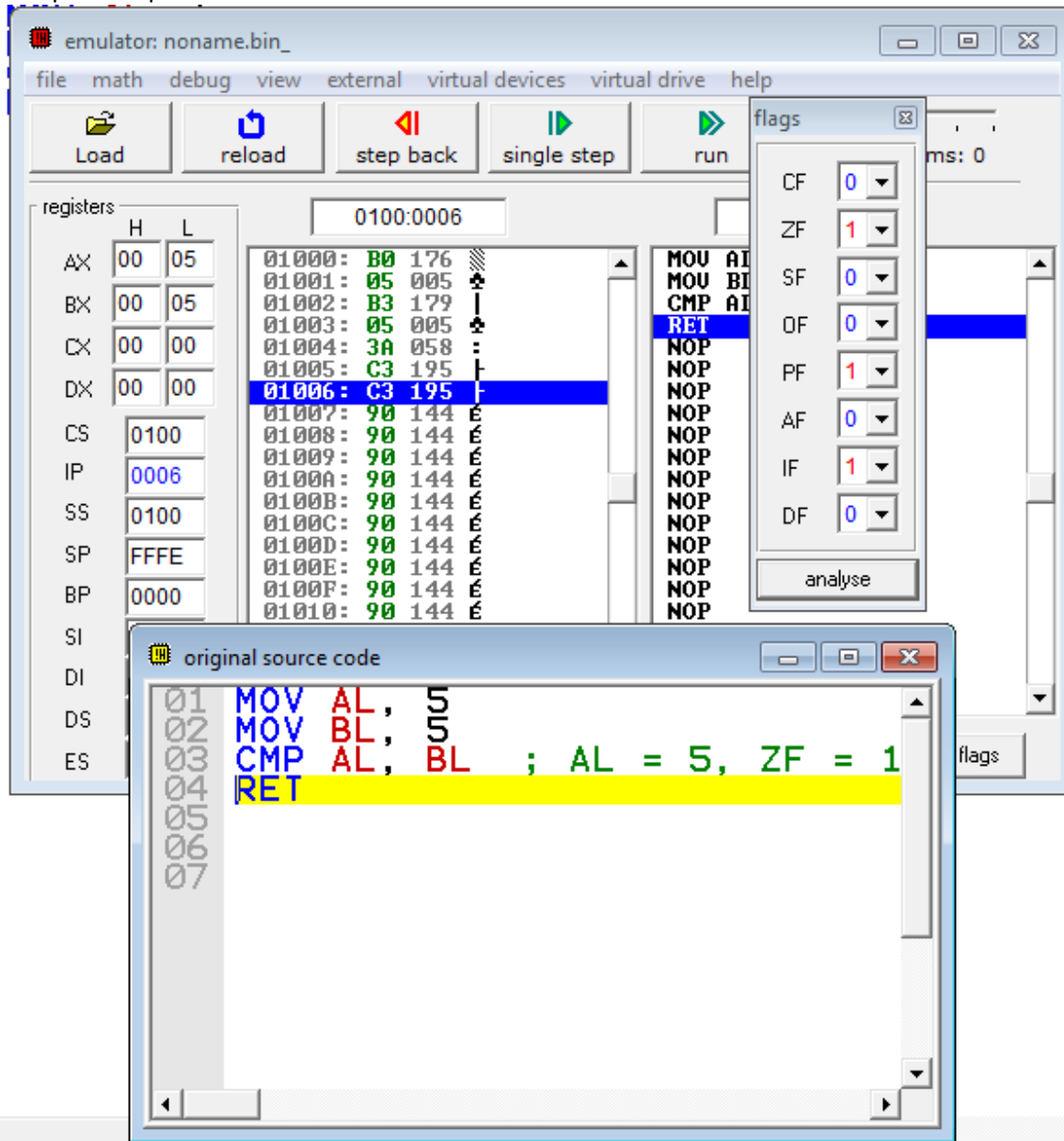
The 'registers' section shows the state of various registers (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES). The '0100:0004' memory address is highlighted, showing the instruction 'NEG AL'.

The 'original source code' window displays the following assembly code with comments:

```

01 MOV AL, 5 ; AL = 05h
02 NEG AL ; AL = 0FBh (-5)
03 NEG AL ; AL = 05h (5)
04 RET
05
06
07
```

CMP	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Compare.</p> <p>Algorithm:</p> <p>operand1 - operand2</p> <p>result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.</p> <p>Example: MOV AL, 5 MOV BL, 5 CMP AL, BL ; AL = 5, ZF = 1 (so equal!) RET</p> <table border="1" data-bbox="502 824 703 922"><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									



MUL

REG
memory

Unsigned multiply.

Algorithm:

when operand is a **byte**:

AX = AL * operand.

when operand is a **word**:

(DX AX) = AX * operand.

Example:

MOV AL, 200 ; AL = 0C8h

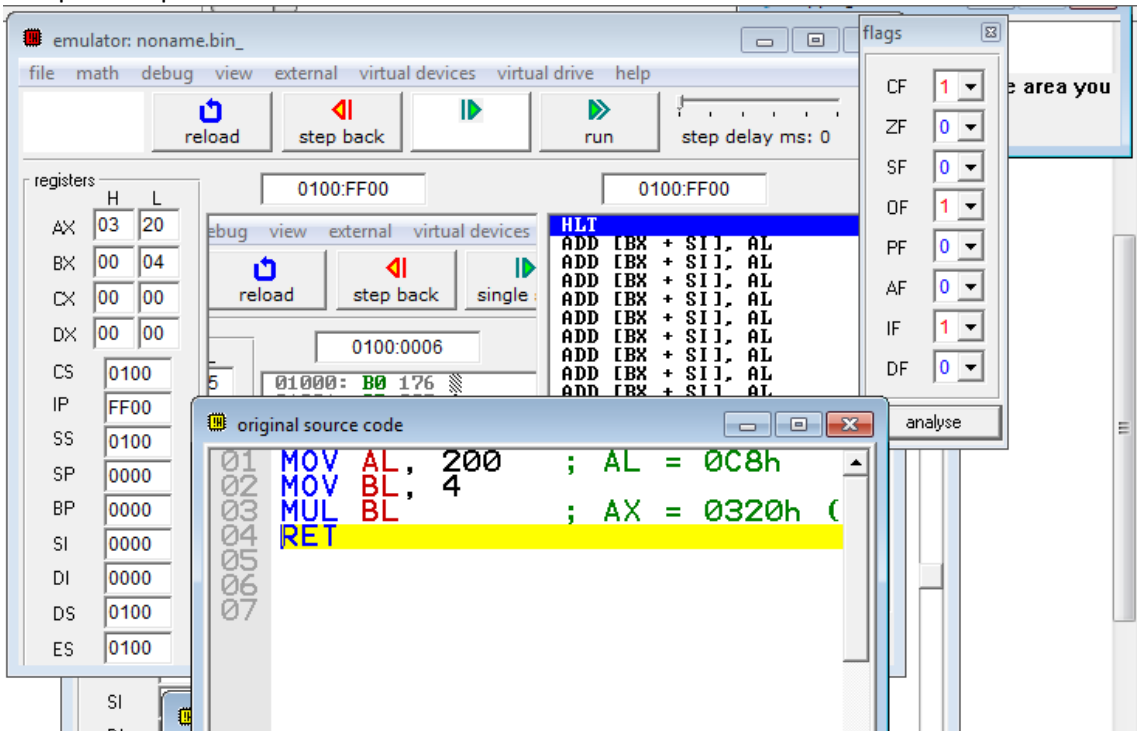
MOV BL, 4

MUL BL ; AX = 0320h (800)

RET

C	Z	S	O	P	A
r	?	?	r	?	?

CF=OF=0 when high section of the result is zero.



<p>IMUL</p>	<p>REG memory</p>	<p>Signed multiply.</p> <p>Algorithm:</p> <p>when operand is a byte: AX = AL * operand.</p> <p>when operand is a word: (DX AX) = AX * operand.</p> <p>Example: MOV AL, -2 MOV BL, -4 IMUL BL ; AX = 8 RET</p> <table border="1" data-bbox="502 1892 702 1993"> <tr> <td>C</td> <td>Z</td> <td>S</td> <td>O</td> <td>P</td> <td>A</td> </tr> <tr> <td>r</td> <td>?</td> <td>?</td> <td>r</td> <td>?</td> <td>?</td> </tr> </table> <p>CF=OF=0 when result fits into operand of IMUL.</p>	C	Z	S	O	P	A	r	?	?	r	?	?
C	Z	S	O	P	A									
r	?	?	r	?	?									

The image shows a screenshot of an x86 emulator window titled "emulator: noname.bin_". The window has a menu bar with "file", "math", "debug", "view", "external", "virtual devices", "virtual drive", and "help". Below the menu bar are several control buttons: "Load", "reload", "step back", "single step", "run", and a "step delay ms: 0" slider.

On the left side, there is a "registers" panel with a table showing the state of various registers:

	H	L
AX	00	08
BX	00	FC
CX	00	00
DX	00	00
CS	0100	
IP	FF00	
SS	0100	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

The main window displays assembly code at memory address 0100:FF00. The code is as follows:

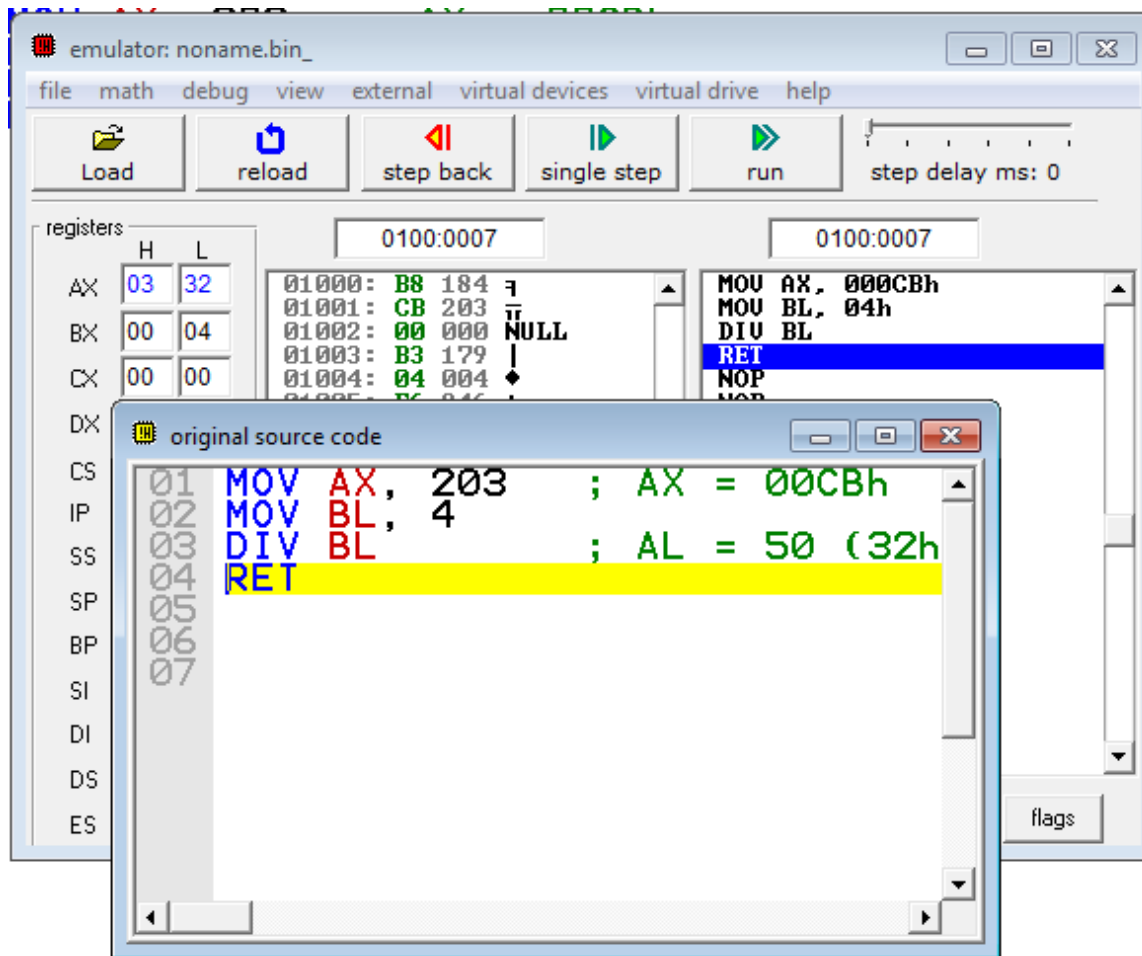
```
10F00: F4 244 ↑ HLT
10F01: 00 000 NULL ADD [BX + SI], AL
10F02: 00 000 NULL ADD [BX + SI], AL
10F03: 00 000 NULL ADD [BX + SI], AL
```

An "original source code" window is overlaid on top, showing the following assembly code:

```
01 MOV AL, -2
02 MOV BL, -4
03 IMUL BL ; AX = 8
04 RET
05
06
07
```

The "RET" instruction on line 04 is highlighted in yellow.

<p>DIV</p>	<p>REG memory</p>	<p>Unsigned divide.</p> <p>Algorithm:</p> <p>when operand is a byte: $AL = AX / \text{operand}$ $AH = \text{remainder (modulus)}$</p> <p>when operand is a word: $AX = (DX AX) / \text{operand}$ $DX = \text{remainder (modulus)}$</p> <p>Example:</p> <p>MOV AX, 203 ; AX = 00CBh</p> <p>MOV BL, 4</p> <p>DIV BL ; AL = 50 (32h), AH = 3</p> <p>RET</p> <table border="1" data-bbox="558 1030 758 1131"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td> </tr> <tr> <td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td> </tr> </table>	C	Z	S	O	P	A	?	?	?	?	?	?
C	Z	S	O	P	A									
?	?	?	?	?	?									
<p>IDIV</p>	<p>REG memory</p>	<p>Signed divide.</p> <p>Algorithm:</p> <p>when operand is a byte: $AL = AX / \text{operand}$ $AH = \text{remainder (modulus)}$</p> <p>when operand is a word: $AX = (DX AX) / \text{operand}$ $DX = \text{remainder (modulus)}$</p> <p>Example:</p> <p>MOV AX, -203 ; AX = 0FF35h</p> <p>MOV BL, 4</p> <p>IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)</p> <p>RET</p> <table border="1" data-bbox="558 1713 758 1814"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td> </tr> <tr> <td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td> </tr> </table>	C	Z	S	O	P	A	?	?	?	?	?	?
C	Z	S	O	P	A									
?	?	?	?	?	?									



The screenshot shows an emulator window titled "emulator: noname.bin_". The menu bar includes "file", "math", "debug", "view", "external", "virtual devices", "virtual drive", and "help". The toolbar contains buttons for "Load", "reload", "step back", "single step", "run", and a "step delay ms: 0" slider.

On the left, a "registers" panel displays the following values:

	H	L
AX	FD	CE
BX	00	04
CX	00	00
DX	00	00
CS	0100	
IP	0007	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

The "original source code" window shows the following assembly instructions:

```
01 MOV AX, -203 ; AX = 0FF35h
02 MOV BL, 4
03 IDIV BL ; AL = -50 (0CE)
04 RET
05
06
07
```

The instruction "RET" on line 04 is highlighted in yellow. At the bottom of the emulator window, there are buttons for "screen", "source", "reset", "aux", "vars", "debug", "stack", and "flags".