Title: Introduction to Microprocessor 8086, 8086 instructions and programming with 8086.

Abstract:

In this experiment, the main objective is to familiarize emulator EMU8086 by using a simple program to test its different uses and introduction to segmented memory technology used by Microprocessor 8086.

Equipment:

Desktop PC, EMU8086 software.

Theory and Methodology:

The 8086 Microprocessor

The 8086 is a 16-bit microprocessor chip designed by Intel between early 1976 and mid-1978, when it was released. It is Intel third generation microprocessor which perform like minicomputer. The 8086 become the basic x86- architecture of Intel's future processors.

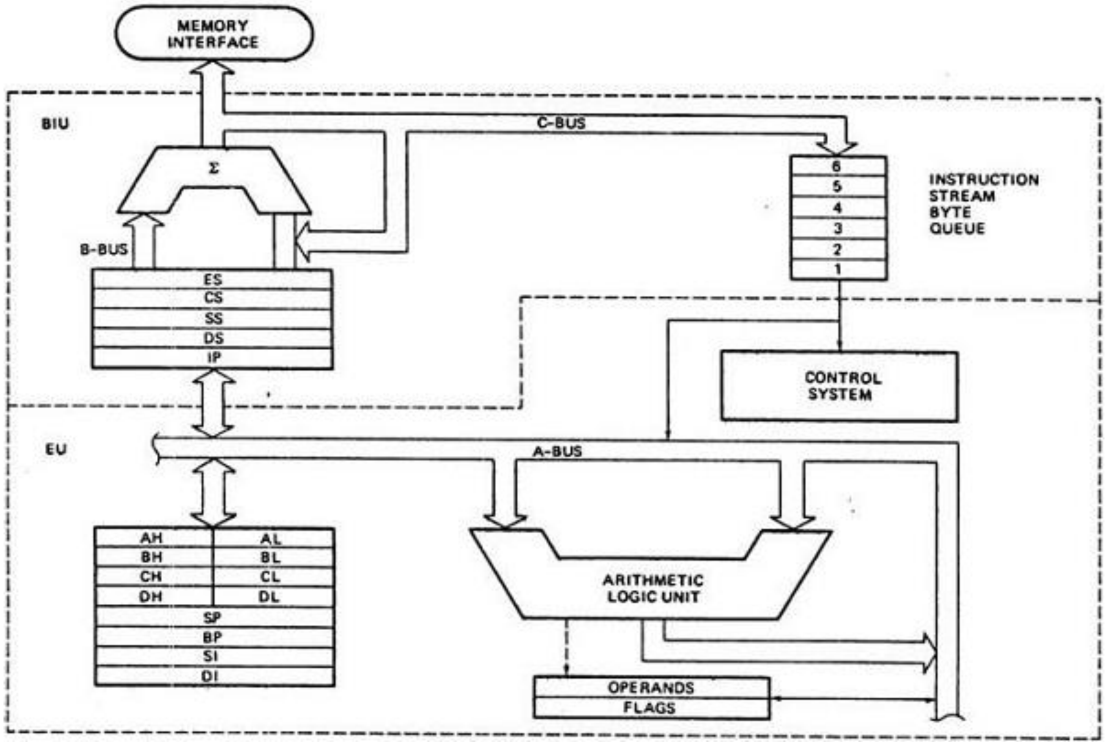


Figure 01: Intel 8086 internal architecture.

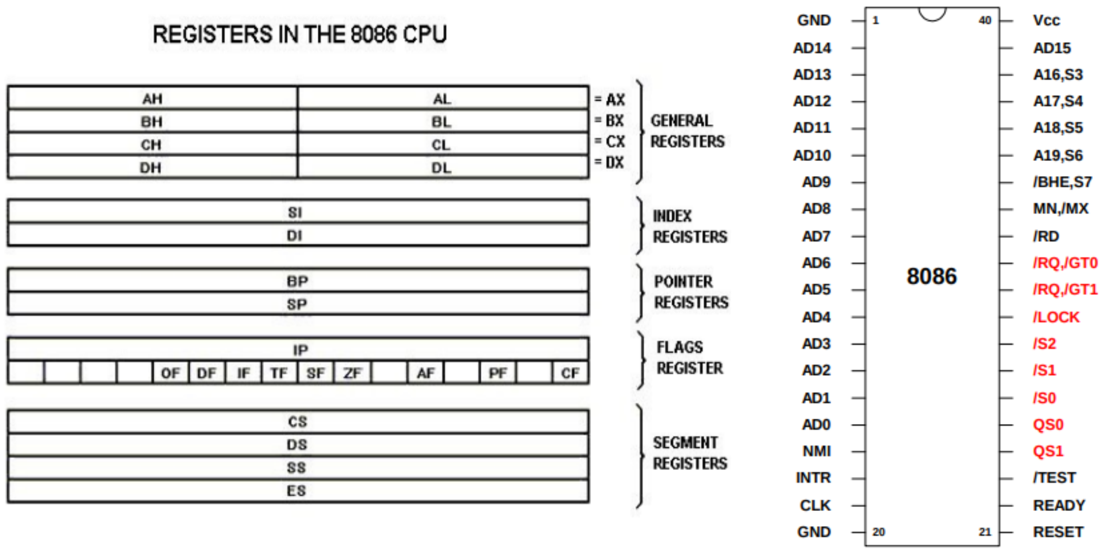


Figure 02: Internal Diagram, Registers and PIN diagram of the 8086 microprocessor.

8086 has total 14 registers (each 16 bit long) where 4 general purpose registers, 9 address registers and 1 status registers.

Data Registers/General purpose

- Instructions execute faster if the data is in a register
- AX, BX, CX, DX are the data registers
- Low and High bytes of the data registers can be accessed separately.
- AH, BH, CH, DH are the high bytes
- AL, BL, CL, and DL are the low bytes

AX (Accumulator Register)

1. Generates shortest machine code
2. Arithmetic, logic and data transfer
3. One number must be in AL or AX
4. Multiplication & Division
5. Input & Output

BX (Base Register)

1. Also serves as an address register
2. Used in array operations
3. Used in Table Lookup operations (XLAT)

CX (Count Register)

1. Iterative code segments using the LOOP instruction
2. Repetitive operations on strings with the REP command
3. Count (in CL) of bits to shift and rotate

DX (Data Register)

1. Used in 32 bit multiplication (higher 16 bit of a 32 bit result is stored in DX).
2. In division (32bit/16 bit) operation the 16 bit remainder is stored in DX.

Pointer and Index Registers

Contain the offset addresses of memory locations
Can also be used in arithmetic and other operations

SP: Stack pointer

1. Always points to top item on the stack
2. Offset address relative to SS
3. Always points to word (byte at even address)
4. An empty stack will had SP = FFFFh

BP: Base Pointer

1. Primarily used to access parameters passed via the stack
2. Offset address relative to SS

SI: Source Index register

1. Can be used for pointer addressing of data
2. Used as source in some string processing instructions
3. Offset address relative to DS

DI: Destination Index register

1. Can be used for pointer addressing of data
2. Used as destination in some string processing instructions
3. Offset address relative to ES

The SI and the DI registers may also be used to access data stored in arrays

Special Purpose Registers

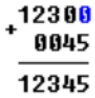
IP: the instruction pointer

1. Always points to next instruction to be executed
2. Offset address relative to CS IP register always works together with CS segment register and it points to currently executing instruction.
3.

Segment Registers

CS - points at the segment containing the current program.
DS - generally points at segment where variables are defined.
ES - extra segment register, it's up to a coder to define its usage.
SS - points at the segment containing the stack.
Although it is possible to store any data in the segment registers, this is never a good idea. The segment registers have a very special purpose - pointing at accessible blocks of memory. Segment registers work

together with general purpose register to access any memory value. For example if we would like to access memory at the physical address 12345h(hexadecimal), we could set the DS = 1230h and SI = 0045h. This way we can access much more memory than with a single register, which is limited to 16 bit values. The CPU makes a calculation of the physical address by multiplying the segment register by 10h and adding the general purpose register to it (1230h * 10h + 45h = 12345h):



The address formed with 2 registers is called an effective address. By default, BX, SI and DI registers work with DS segment register; BP and SP work with SS segment register. Other general-purpose registers cannot form an effective address. Also, although BX can form an effective address, BH and BL cannot.

Flags Register

Flags Register - determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. Generally, you cannot access these registers directly.

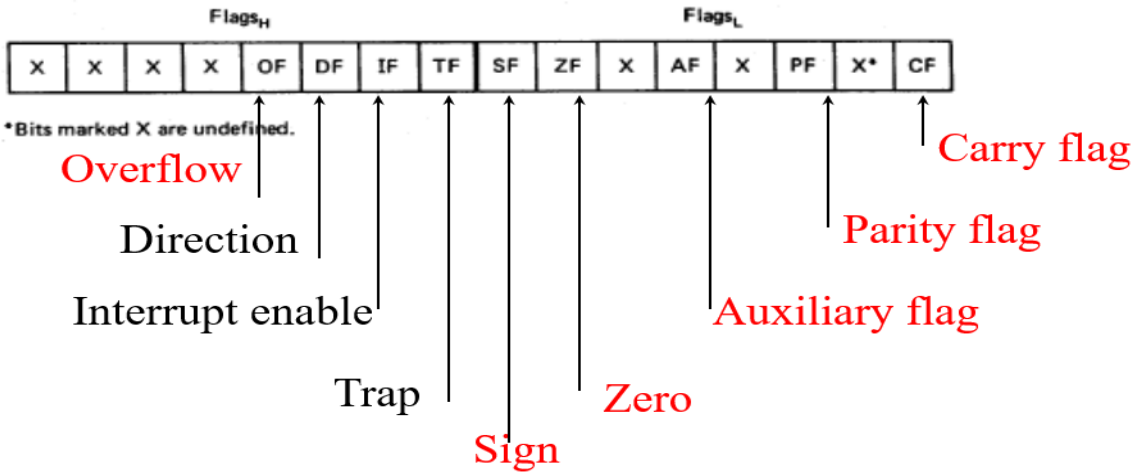


Figure 03: Flags.

Here, 6 are status flags: OF, SF, ZF, AF, PF, CF and 3 are control flag: DF, IF, TF.

1. Carry Flag (CF) - this flag is set to 1 when there is an unsigned overflow. For example, when you add bytes 255 + 1 (result is not in range 0...255). When there is no overflow this flag is set to 0.
2. Parity Flag (PF) - this flag is set to 1 when there is even number of ‘1’ bits in result, and to 0 when there is odd number of ‘1’ bits.
3. Auxiliary Flag (AF) - set to 1 when there is an unsigned overflow for low nibble (4 bits).
4. Zero Flag (ZF) - set to 1 when result is zero. For non-zero result this flag is set to 0.
5. Sign Flag (SF) - set to 1 when result is negative. When result is positive it is set to0. (This flag takes the value of the most significant bit.)

6. Trap Flag (TF) - Used for on-chip debugging.
7. Interrupt enable Flag (IF) - when this flag is set to 1 CPU reacts to interrupts from external devices.
8. Direction Flag (DF) - this flag is used by some instructions to process data chains, when this flag is set to 0 - the processing is done forward, when this flag is set to 1the processing is done backward.
9. Overflow Flag (OF) - set to 1 when there is a signed overflow. For example, when you add bytes 100 + 50 (result is not in range -128...127).

8086 Segmented Memory/Memory Organization

Each byte in memory has a 20 bit address starting with 0 to 2^{20}-1 = 1 megabyte of addressable memory. Addresses are expressed as 5 hex digits starting from 00000h, 00001h, 00002h ………FFFFE h, FFFFF h. The problem is 20 bit addresses are too big to fit in 16 bit registers. So for the solution we can dividing the memory into Segments.

A memory segment is a block of 2^16 (64K) consecutive memory bytes. Each segment is defined by a segment number. A segment number is 16 bits long, so the Segment numbers range from 0000 h to FFFF h. Within a segment, a particular memory location is specified with an offset. An offset also ranges from 0000h to FFFFh.

So Segmented memory addressing: absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset.

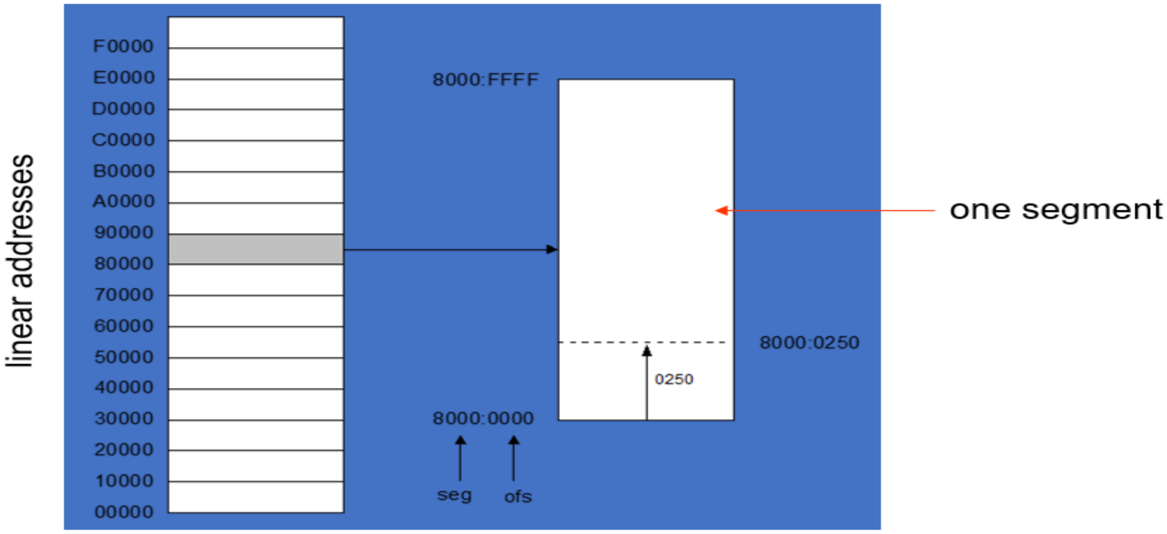


Figure 04: Segment memory.

Logical Address is specified as **16 bit segment: 16 bit offset** and Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address.

Thus the physical address of the logical address A4FB:4872 is:

$$\begin{array}{r} \text{A4FB0} \\ +\ \underline{4872} \\ \text{A9822} \end{array}$$

And 1256Ah=1256:000A that is physical address 1256Ah can be represented as offset 000A of segment 1256. Again, the same physical address 1256Ah can be represented as offset 016A of segment 1240 as:

1256Ah=1240:016A

8086 Instruction and Assembly language

8086 instruction set consists of the following instructions:

Data Transfer

1. MOV AX, BX; register: move contents of BX to AX
2. COUNT to AX
   MOV AX, COUNT; direct: move contents of the address labeled
3. MOV CX, 0F0H; immediate: load CX with the value 240
4. MOV CX, [0F0H]; memory: load CX with the value at address 240
5. MOV [BX], AL; register indirect: move contents of AL to memory location in BX

I/O Operations

The 8086 has separate I/O and memory address spaces. Values in the I/O space are accessed with IN and OUT instructions. The port address is loaded into DX and the data is read/written to/from AL or AX: MOV DX, 372H; load DX with port address

Arithmetic/Logic

Arithmetic and logic instructions can be performed on byte and 16-bit values. The first operand has to be a register and the result is stored in that register.
ADD BX, 4; increment BX by 4 ¤ ADD AX, CX; AX= AX + CX
SUB AL, 1; subtract 1 from AL ¤ SUB DX, CX; DX= DX – CX.
Also MUL and DIV for multiplication and division.

Codes:

Code for Addition:

```
org 100h
MOV BX, 1234h
MOV CX, 5678h
ADD BX, CX ; BX= BX+CX= 1234h+5678h
ret
```

Code for Subtraction:

```
org 100h
MOV AX, 5678h
MOV BX, 1234h
SUB AX, BX ; AX= AX-BX= 5678h-1234h
ret
```

Code for Multiplication:

(For, 8 bit * 8 bit = 16 bit)

```
org 100h
MOV AX, 5678h
MOV BX, 1234h
MUL BL ; AX= AL (fixed operand)*BL =78 * 34
ret
```

(For, 16 bit * 16 bit = 32 bit)

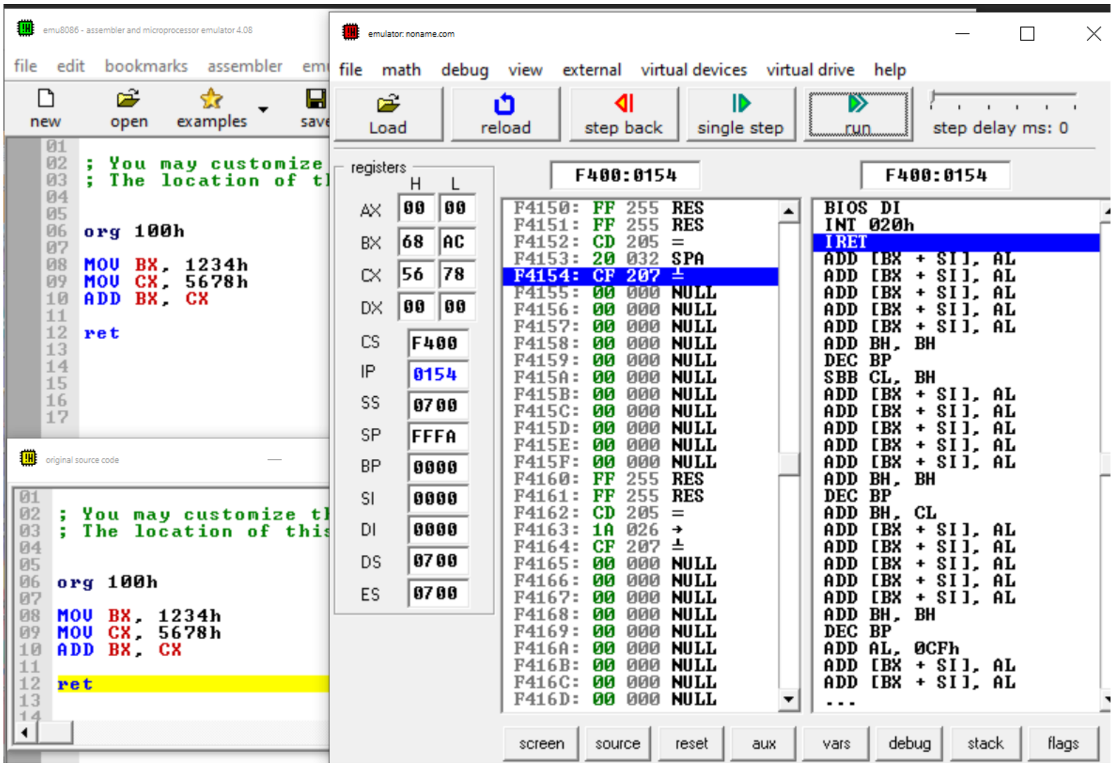
```
org 100h
MOV AX, 5678h
MOV BX, 1234h
MUL BX ; DX:AX= AX (fixed operand)*BX= 5678h * 1234h
ret
```

Code for Division:

```
org 100h
MOV AX, 15
MOV DL, 2
DIV DL ; AX= AX(fixed operand)/DL where quotient is in AL and remainder in AH
ret
```

Results:

Result for Addition:



```
; You may customize
; The location of tl

org 100h

MOV BX, 1234h
MOV CX, 5678h
ADD BX, CX

ret
```

```
registers
     H   L
AX   00  00
BX   68  AC
CX   56  78
DX   00  00
CS   F400
IP   0154
SS   0700
SP   FFFA
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

```
F400:0154

F4150:  FF 255 RES
F4151:  FF 255 RES
F4152:  CD 205 =
F4153:  20 032 SPA
F4154:  CF 207 ±
F4155:  00 000 NULL
F4156:  00 000 NULL
F4157:  00 000 NULL
F4158:  00 000 NULL
F4159:  00 000 NULL
F415A:  00 000 NULL
F415B:  00 000 NULL
F415C:  00 000 NULL
F415D:  00 000 NULL
F415E:  00 000 NULL
F415F:  00 000 NULL
F4160:  FF 255 RES
F4161:  FF 255 RES
F4162:  CD 205 =
F4163:  1A 026 →
F4164:  CF 207 ±
F4165:  00 000 NULL
F4166:  00 000 NULL
F4167:  00 000 NULL
F4168:  00 000 NULL
F4169:  00 000 NULL
F416A:  00 000 NULL
F416B:  00 000 NULL
F416C:  00 000 NULL
F416D:  00 000 NULL
```

```
F400:0154

BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD AL, 0CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

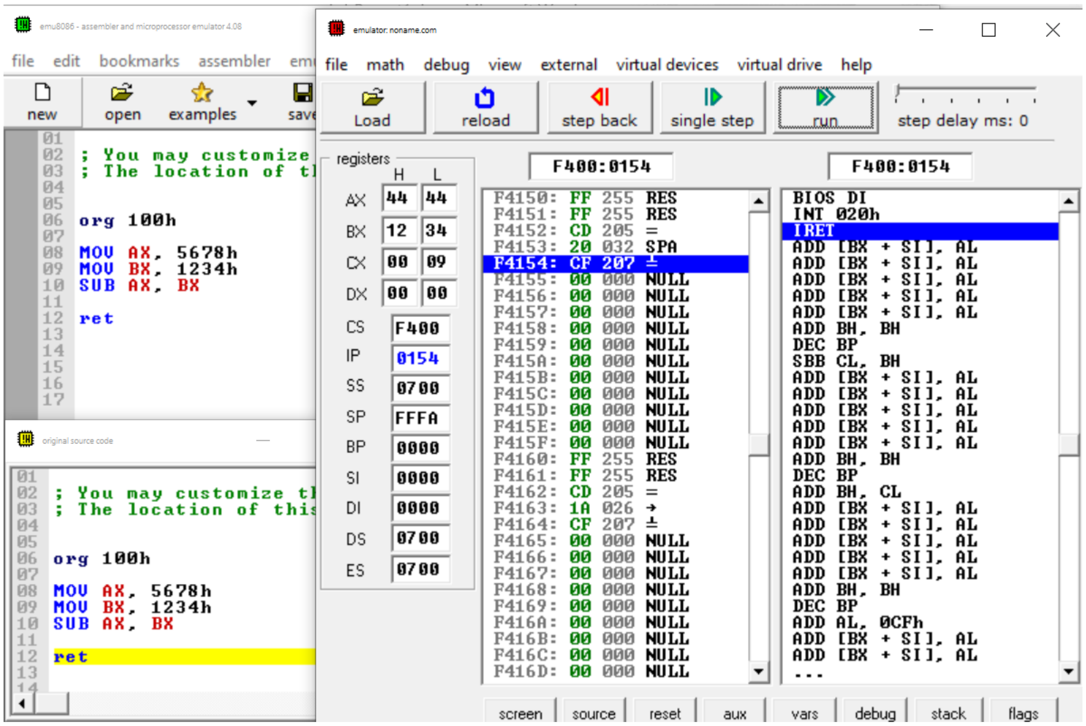
```
; You may customize tl
; The location of this

org 100h

MOV BX, 1234h
MOV CX, 5678h
ADD BX, CX

ret
```

Result for Subtraction:


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler em
new open examples save
original source code
emulator: noname.com
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags

```
; You may customize
; The location of t

org 100h

MOV AX, 5678h
MOV BX, 1234h
SUB AX, BX

ret
```

```
; You may customize t
; The location of this

org 100h

MOV AX, 5678h
MOV BX, 1234h
SUB AX, BX

ret
```

registers

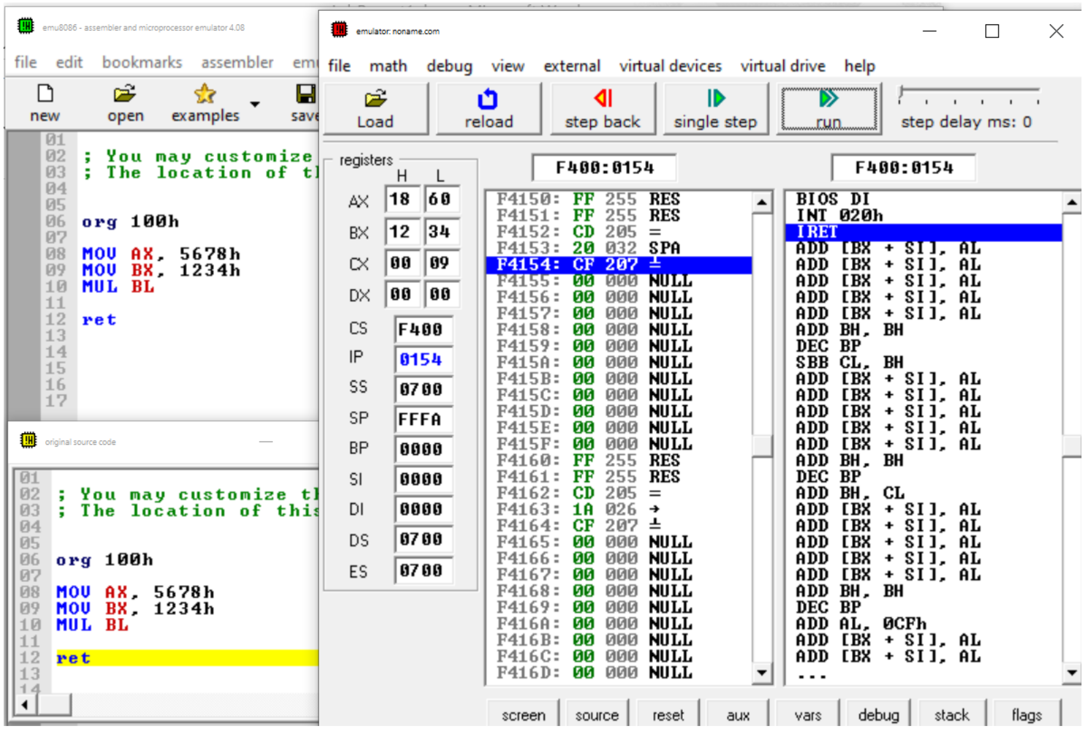
	H	L
AX	44	44
BX	12	34
CX	00	09
DX	00	00
CS	F400	
IP	0154	
SS	0700	
SP	FFFA	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
F400:0154
F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ±
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ±
F4165: 00 000 NULL
F4166: 00 000 NULL
F4167: 00 000 NULL
F4168: 00 000 NULL
F4169: 00 000 NULL
F416A: 00 000 NULL
F416B: 00 000 NULL
F416C: 00 000 NULL
F416D: 00 000 NULL
```

```
F400:0154
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD AL, 0CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

Result for Multiplication:

(For, 8 bit * 8 bit = 16 bit)


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler em
new open examples save
original source code
emulator: noname.com
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags

```
; You may customize
; The location of t

org 100h

MOV AX, 5678h
MOV BX, 1234h
MUL BL

ret
```

```
; You may customize t
; The location of this

org 100h

MOV AX, 5678h
MOV BX, 1234h
MUL BL

ret
```

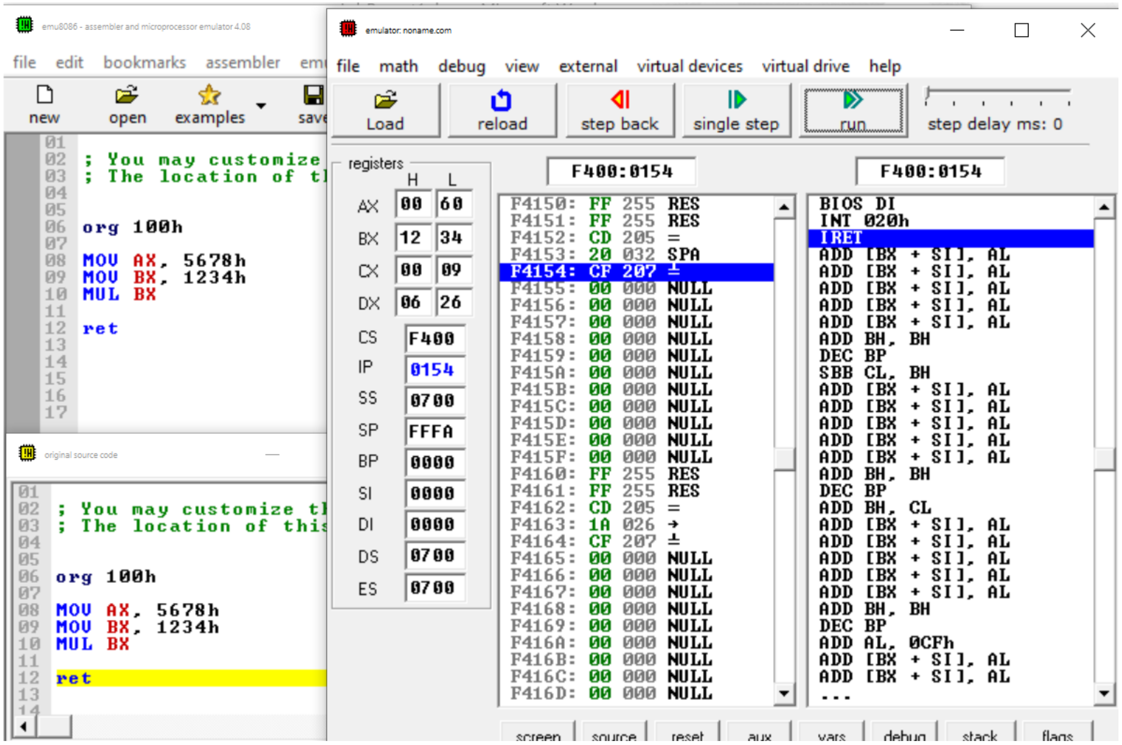
registers

	H	L
AX	18	60
BX	12	34
CX	00	09
DX	00	00
CS	F400	
IP	0154	
SS	0700	
SP	FFFA	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
F400:0154
F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ±
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ±
F4165: 00 000 NULL
F4166: 00 000 NULL
F4167: 00 000 NULL
F4168: 00 000 NULL
F4169: 00 000 NULL
F416A: 00 000 NULL
F416B: 00 000 NULL
F416C: 00 000 NULL
F416D: 00 000 NULL
```

```
F400:0154
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD AL, 0CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

(For, 16 bit * 16 bit = 32 bit)


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emu
new open examples save
original source code
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
screen source reset aux vars debug stack flags

```
; You may customize
; The location of t

org 100h

MOV AX, 5678h
MOV BX, 1234h
MUL BX

ret
```

```
; You may customize t
; The location of this

org 100h

MOV AX, 5678h
MOV BX, 1234h
MUL BX

ret
```

```
registers
     H   L
AX   00  60
BX   12  34
CX   00  09
DX   06  26
CS   F400
IP   0154
SS   0700
SP   FFFA
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

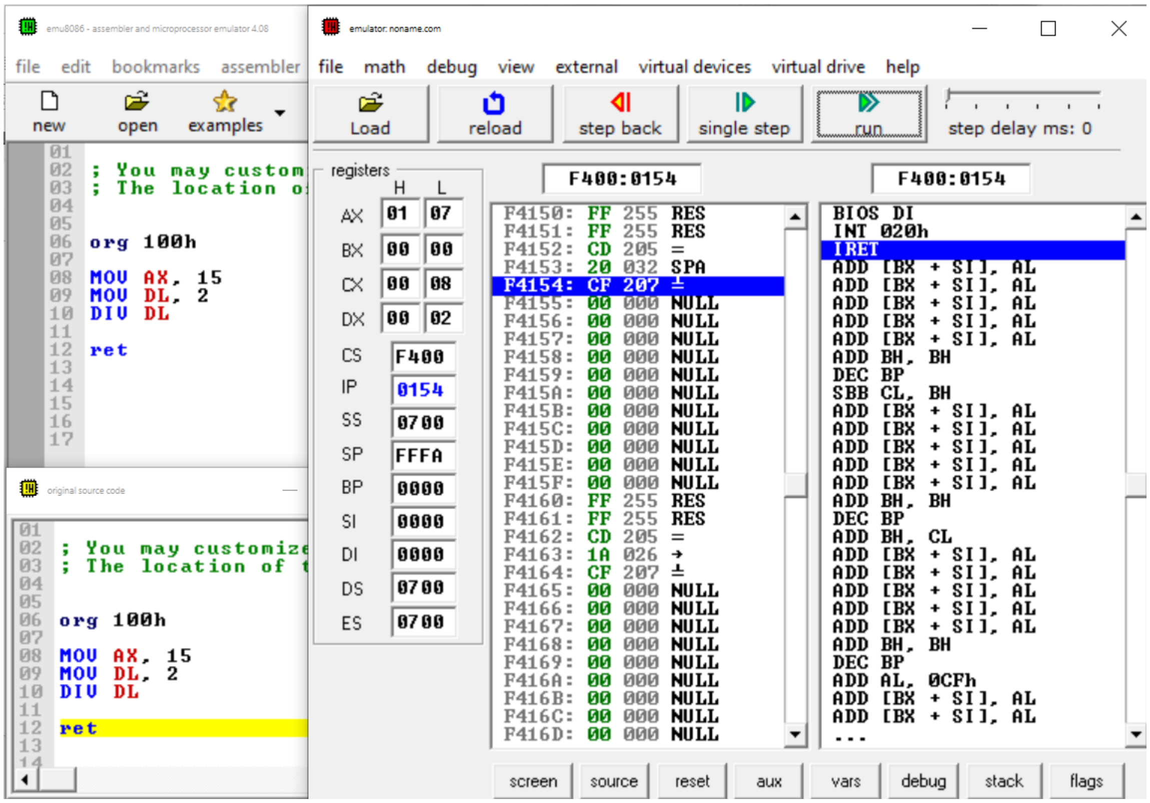
```
F400:0154

F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ╧
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ╧
F4165: 00 000 NULL
F4166: 00 000 NULL
F4167: 00 000 NULL
F4168: 00 000 NULL
F4169: 00 000 NULL
F416A: 00 000 NULL
F416B: 00 000 NULL
F416C: 00 000 NULL
F416D: 00 000 NULL
```

```
F400:0154

BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD AL, 0CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

Result for Division:


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler
new open examples
original source code
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
screen source reset aux vars debug stack flags

```
; You may custom
; The location o

org 100h

MOV AX, 15
MOV DL, 2
DIV DL

ret
```

```
; You may customize
; The location of t

org 100h

MOV AX, 15
MOV DL, 2
DIV DL

ret
```

```
registers
     H   L
AX   01  07
BX   00  00
CX   00  08
DX   00  02
CS   F400
IP   0154
SS   0700
SP   FFFA
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

```
F400:0154

F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ╧
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ╧
F4165: 00 000 NULL
F4166: 00 000 NULL
F4167: 00 000 NULL
F4168: 00 000 NULL
F4169: 00 000 NULL
F416A: 00 000 NULL
F416B: 00 000 NULL
F416C: 00 000 NULL
F416D: 00 000 NULL
```

```
F400:0154

BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD AL, 0CFh
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

Discussion:

In the first code for Addition, we transmitted the first hexadecimal value 1234h to BX register and second value 5678h to CX register. After addition the result value 68ACh updated to BX register. Next for Subtraction, we transmitted the first hexadecimal value 5678h to AX register and second value 1234h to BX register. After subtraction the result value 4444h updated to AX register. After that for Multiplication of two 8 bit numbers, we transmitted the first value to AX or AL register which is mandatory and second value to BX register. From the result we get 16 bit value which is updated to AX register. And for Multiplication of two 16 bit numbers, we transmitted the first value to AX register which is mandatory and second value to BX register. From the result we get 32 bit value which is updated to DX (Higher 16 bit) : AX (Lower 16 bit) register. At the last for Division, we transmitted the first value to AX register which is mandatory and second value to DL register. After division the quotient value updated to AL register and the remainder value to AH register.

References:

1. “Microprocessors and Micro-Computer based System Design”, Second edition – by Dr. M. Rafiquzzaman.

2. EMU8086 Manual.