Lab 02: Arithmetic and Data Movement in Assembly

Objective:

- Understand how to use registers for arithmetic operations.
- Learn the MOV, ADD, SUB, INC, DEC, and XCHG instructions.
- Observe results in registers.

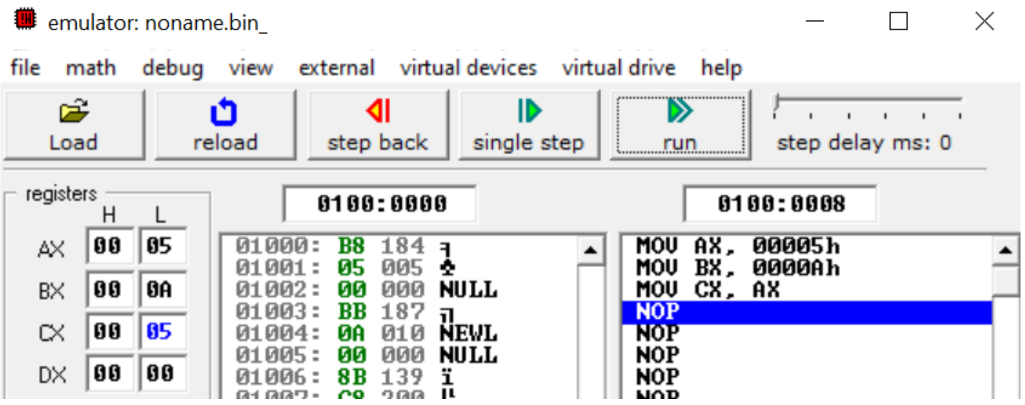
Introduction:

In this lab, students will explore fundamental arithmetic and data movement operations in 8086 assembly language. The lab focuses on using CPU registers to perform basic computations such as addition, subtraction, increment, and decrement, as well as exchanging and transferring data between registers and memory. By practicing with instructions like MOV, ADD, SUB, INC, DEC, DIV, MUL and XCHG, students will gain a practical understanding of how the processor manipulates data, how results are stored in registers, and how memory interactions are handled in low-level programming. This foundational knowledge is essential for effective assembly programming and for understanding the inner workings of a CPU.

Moving data between registers

```
MOV AX, 5
MOV BX, 10
MOV CX, AX  ; Copy AX to CX
```

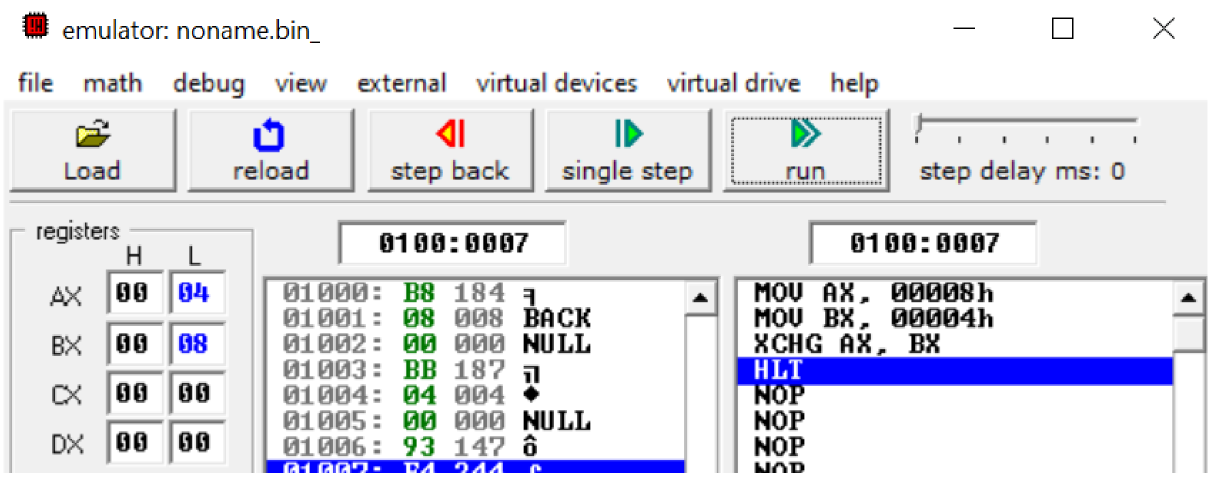
Output:



```
     H   L
AX   00  05
BX   00  0A
CX   00  05
DX   00  00

0100:0000                     0100:0008

01000:  B8 184 ╕              MOV  AX, 00005h
01001:  05 005 ♣              MOV  BX, 0000Ah
01002:  00 000 NULL           MOV  CX, AX
01003:  BB 187 ╗              NOP
01004:  0A 010 NEWL           NOP
01005:  00 000 NULL           NOP
01006:  8B 139 ï              NOP
```

Exchanging Data between registers

```
MOV AX, 8
MOV BX, 4
XCHG AX,BX    ; Exchange the data
HLT           ; Stop CPU
```



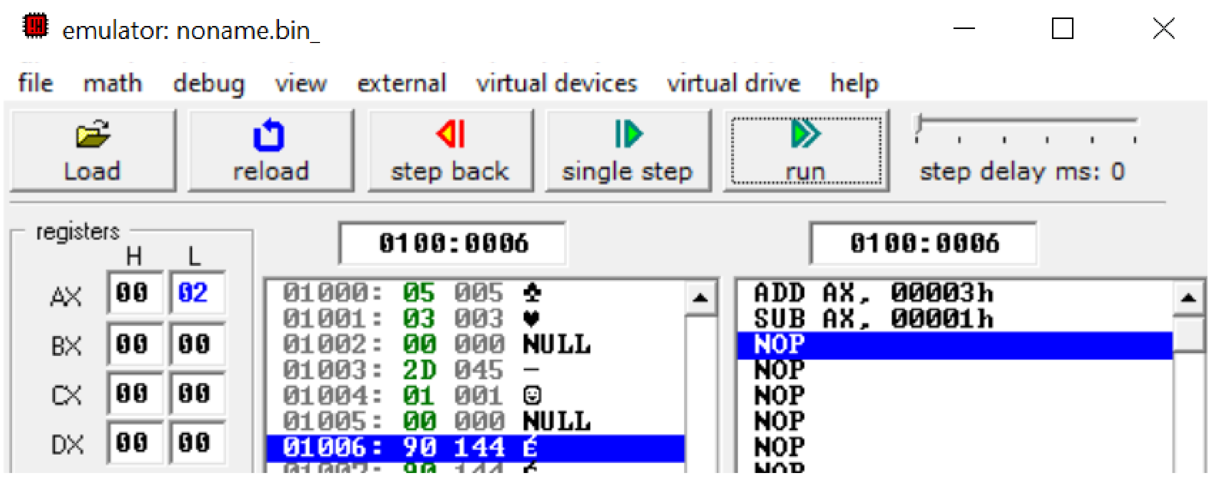
```
    H  L
AX  00 04
BX  00 08
CX  00 00
DX  00 00

0100:0007
01000: B8 184 ╕
01001: 08 008 BACK
01002: 00 000 NULL
01003: BB 187 ╗
01004: 04 004 ♦
01005: 00 000 NULL
01006: 93 147 ô

0100:0007
MOV AX, 00008h
MOV BX, 00004h
XCHG AX, BX
HLT
NOP
NOP
NOP
```

Basic Arithmetic Operations

Addition and Subtraction

```
ADD AX,3
SUB AX,1
```

Output:

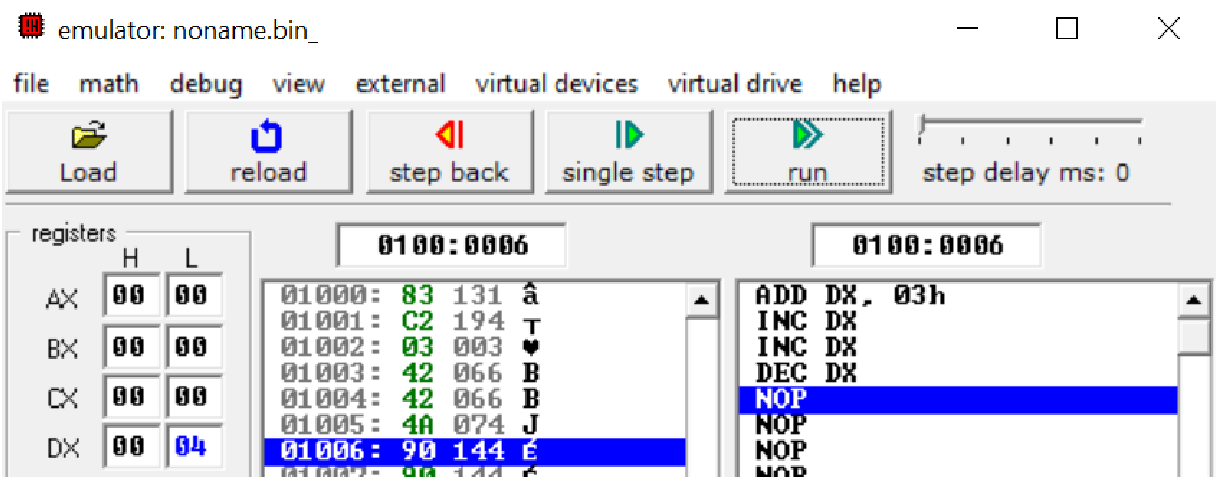


```
    H  L
AX  00 02
BX  00 00
CX  00 00
DX  00 00

0100:0006
01000: 05 005 ♣
01001: 03 003 ♥
01002: 00 000 NULL
01003: 2D 045 -
01004: 01 001 ☺
01005: 00 000 NULL
01006: 90 144 É

0100:0006
ADD AX, 00003h
SUB AX, 00001h
NOP
NOP
NOP
NOP
NOP
```

Increment and Decrement

```
ADD DX,3
INC DX
INC DX
DEC DX
```

Output



```
AX 00 00
BX 00 00
CX 00 00
DX 00 04
```

```
0100:0006

01000: 83 131 â
01001: C2 194 ┬
01002: 03 003 ♥
01003: 42 066 B
01004: 42 066 B
01005: 4A 074 J
01006: 90 144 É
```

```
0100:0006

ADD DX, 03h
INC DX
INC DX
DEC DX
NOP
NOP
NOP
```

Multiplication in 8086 Assembly

In the 8086 microprocessor, multiplication is performed using the MUL (unsigned multiply) and IMUL (signed multiply) instructions. These instructions use the AX register implicitly as one of the operands, meaning the programmer does not need to specify it — the CPU assumes it automatically.

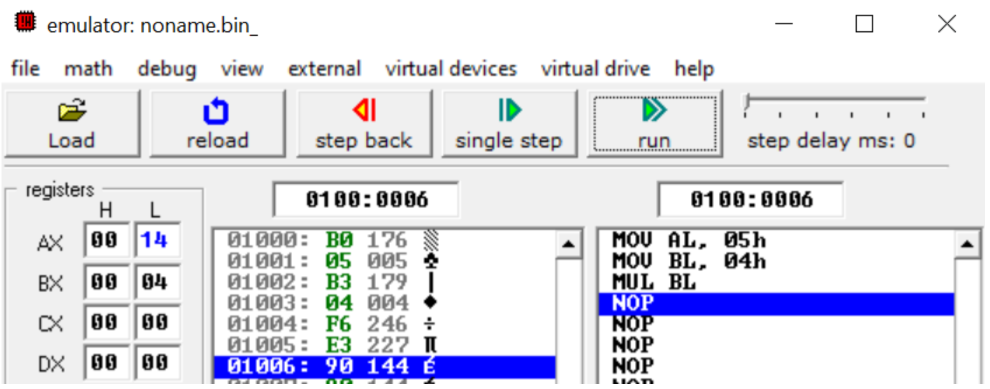
For 8-bit operands:

- The multiplicand is in AL (lower half of AX).
- The other operand is specified in the instruction.
- The result is stored in AX.

Example

```
MOV AL, 05h   ; AL = 5
MOV BL, 04h   ; BL = 4
MUL BL        ; AX = AL × BL = 20h
HLT
```

Output



```
      H   L
AX    00  14
BX    00  04
CX    00  00
DX    00  00

0100:0006                     0100:0006

01000:  B0 176 ░              MOV  AL, 05h
01001:  05 005 ♣              MOV  BL, 04h
01002:  B3 179 │              MUL  BL
01003:  04 004 ♦              NOP
01004:  F6 246 ÷              NOP
01005:  E3 227 π              NOP
01006:  90 144 É              NOP
```

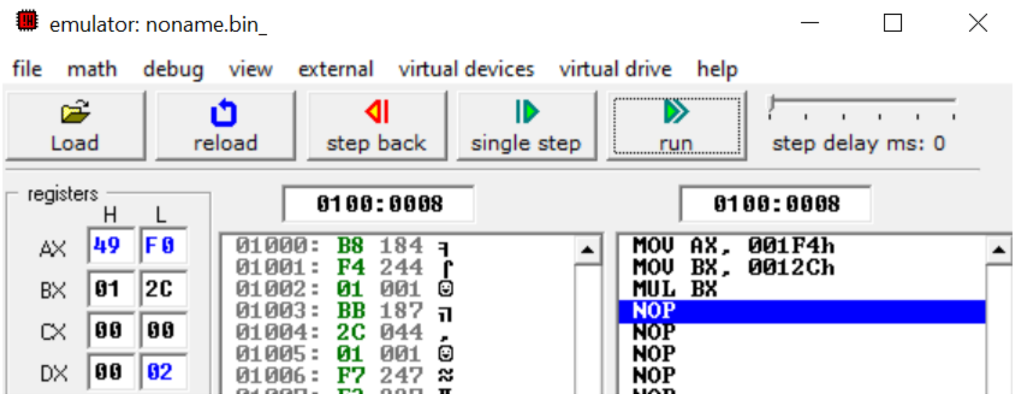
For 16-bit operands:

- The multiplicand is in AX.
- The other operand is specified in the instruction.
- The result is stored in DX:AX (DX = high word, AX = low word).

Example:

```
MOV AX, 500 ; AX = 500
MOV BX, 300 ; BX = 300
MUL BX       ; DX:AX = AX × BX = 150000h
HLT
```

Output



```
      H   L
AX    49  F0
BX    01  2C
CX    00  00
DX    00  02

0100:0008                     0100:0008

01000:  B8 184 ╕              MOV  AX, 001F4h
01001:  F4 244 ⌠              MOV  BX, 0012Ch
01002:  01 001 ☺              MUL  BX
01003:  BB 187 ╗              NOP
01004:  2C 044 ,              NOP
01005:  01 001 ☺              NOP
01006:  F7 247 ≈              NOP
```

32-bit register in 8086: On the 8086, each register is 16-bit, but sometimes you need to hold results larger than 16 bits (like after multiplication or before division). For that:

- AX + DX are paired together → called DX:AX.
- This makes a 32-bit register pair (DX is the high 16 bits, AX is the low 16 bits).

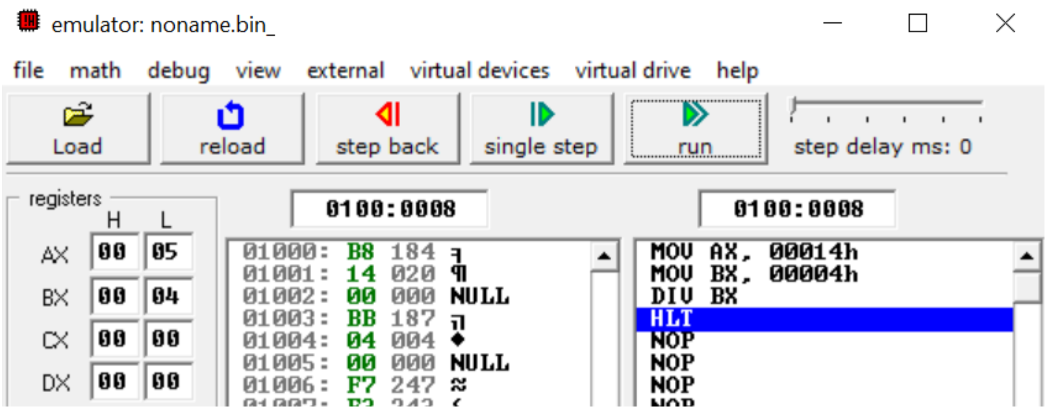
Interpretation for above result: Result = 150000 (decimal) = 0x249F0 (hex) → needs more than 16 bits.

- AX = 46F0h (low word)
- DX = 0002h (high word)
- Together: DX:AX = 0002 46F0h = 150000 decimal.

Division (Same as multiplication)

```
MOV AX, 20    ; Dividend
MOV BX, 4     ; Divisor
DIV BX        ; AX / BX → quotient in AX, remainder in DX
HLT           ; Stop CPU
```

Output:



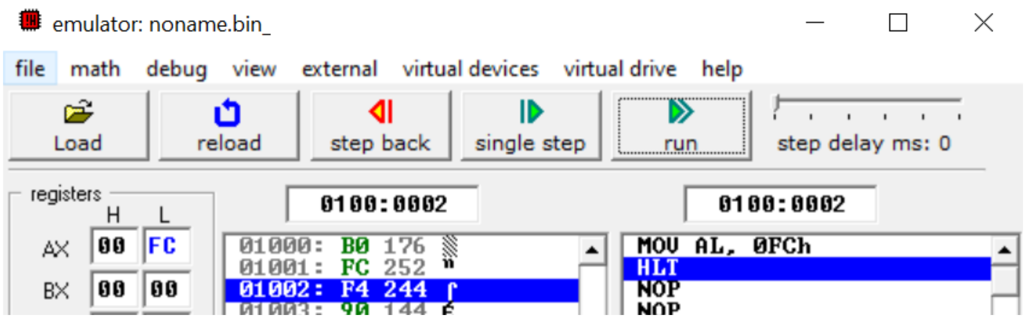
```
      H   L
AX   00  05
BX   00  04
CX   00  00
DX   00  00

0100:0008                    0100:0008
01000: B8 184 ╕              MOV AX, 00014h
01001: 14 020 ¶              MOV BX, 00004h
01002: 00 000 NULL           DIV BX
01003: BB 187 ╗              HLT
01004: 04 004 ♦              NOP
01005: 00 000 NULL           NOP
01006: F7 247 ≈              NOP
```

Dealing with negative numbers

Using 8-bit Register

```
MOV AL, -4
HLT
```

Output:



```
      H   L
AX   00  FC
BX   00  00

0100:0002                    0100:0002
01000: B0 176 ░              MOV AL, 0FCh
01001: FC 252 ⁿ              HLT
01002: F4 244 ⌠              NOP
```

For 8-bit representation of -4 using two's complement:

1. Start with +4 in 8-bit binary: 0000 0100
2. Take 1's complement (invert all bits): 1111 1011
3. Add 1 to get 2's complement: 1111 1100
4. Convert to hexadecimal: 1111 1100 → FC

So, -4 in 8-bit hexadecimal is FC.

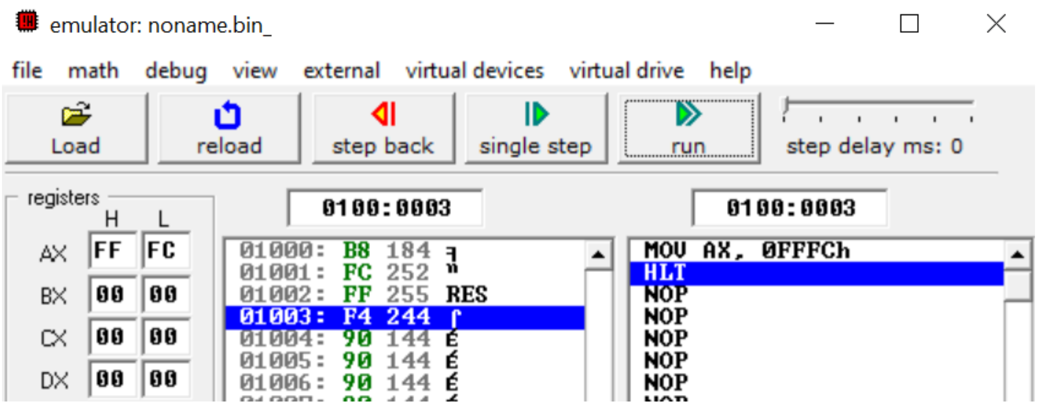
Similarly:

- 16-bit: FFFC
- 32-bit: FFFFFFFC

Using 16 Bit Register

```
MOV AX,-4
HLT
```

Output:



```
registers
     H    L
AX   FF   FC
BX   00   00
CX   00   00
DX   00   00

0100:0003
01000: B8 184 ╕
01001: FC 252 ⁿ
01002: FF 255 RES
01003: F4 244 ⌠
01004: 90 144 É
01005: 90 144 É
01006: 90 144 É

0100:0003
MOV AX, 0FFFCh
HLT
NOP
NOP
NOP
NOP
NOP
```

Exercise:

Task 1: Explore IMUL and IDIV operations and use them in example to distinguish them with MUL and DIV operations respectively.

Task 2: Perform any DIV operation using 32 bits (DX:AX).

Task 3: Use MUL and IMUL with positive numbers, and compare results.

Task 4: Use DIV and IDIV with positive numbers, and compare results.