Software Systems-1

8086 INSTRUCTIONS ENCODING-PART02

Dr. Azam E. Al-Rawachy
COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

1 What are Instruction formats?

When the assembler processes an Instruction it converts the instruction from its mnemonics form to standard machine language format called the "**Instruction format**". In the process of conversion, the assembler must determine the type of instruction, convert symbolic labels and explicit notation to a base/displacement format, determine the lengths of certain operands and parse any literal and constants.

An instruction format defines layout of bits of an instruction, in terms of its constituent parts. An instruction format must include an opcode and implicitly or explicitly, zero or more operands. Each explicit operand is referenced using one of addressing modes. Format must, implicitly or explicitly, indicate addressing mode for each operand. For most instruction sets, more than on instruction format is used.

1.1 INC R/M

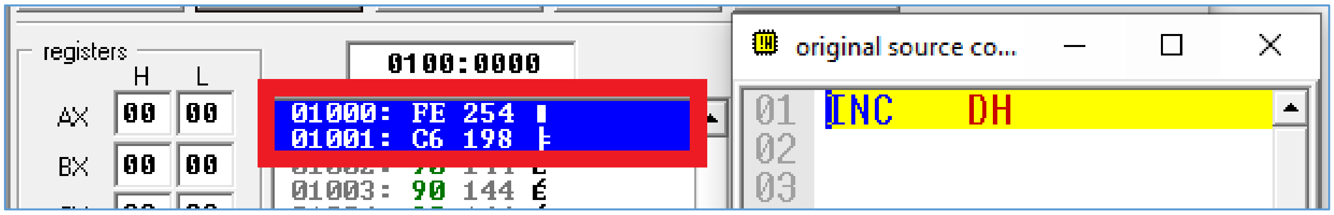
INC	INC reg INC mem	1111111w mod 000 r/m disp data
	INC reg	01000 reg

EX: INC DH = FEC6H

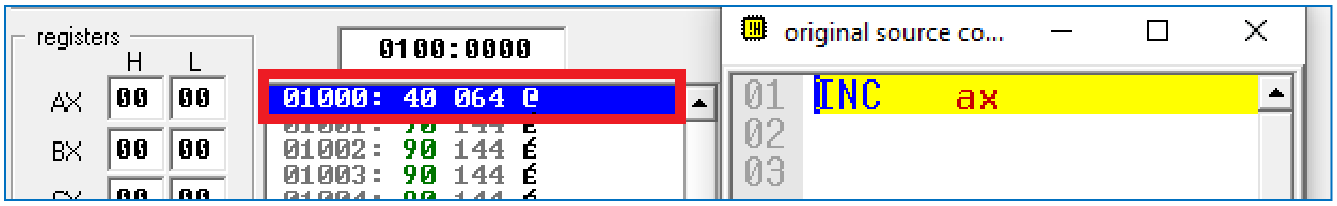
opcode: 1st byte: 1111111, w = 0 (8-bit operand) ➔ 1st byte: 1111111 0 = FE H

operand = DH register: mod = 11 r/m = 110

2nd byte: Mod **opcode** r/m= 11 000 110 = C6 H



```
registers
     H   L
AX  00  00
BX  00  00

0100:0000
01000: FE 254 ▮
01001: C6 198 ╞
01003: 90 144 É

original source co...
INC     DH
```



```
registers
     H   L
AX  00  00
BX  00  00

0100:0000
01000: 40 064 @
01002: 90 144 É
01003: 90 144 É

original source co...
INC     ax
```

EX: INC   BYTE PTR [SI – 4]

INC	INC reg INC mem	1111111w mod 000 r/m disp data

- Indexed addressing to an 8-bit memory operand
- Will need extra byte(s) to encode the immediate value (–4 = FFFC)

opcode – same as last example: 111111

w = 0   8-bit destination (memory) operand

r/m = 100 (from table) 01000100

mod could be 01 or 10 depends on constant, whichever mod can be used for shortened encodings!

“mod = 10”

16-bit constant (FFFC) encoded into instruction

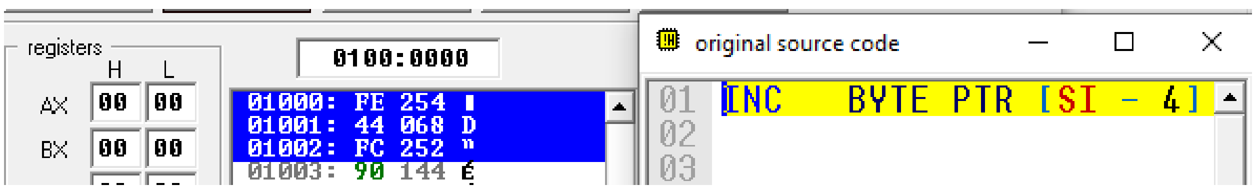
resulting instruction encoding:

byte 1	byte 2	byte 3	byte 4
1111111 0	10 000 100	11111100	11111111
FE	84	FC	FF

“mod = 01”

Constant encoded as signed 8-bit value therefore, instruction encoding includes only one byte for the encoding of – 4 resulting instruction encoding:

byte 1	byte 2	byte 3
1111111 0	01 000 100	11111100
FE	44	FC



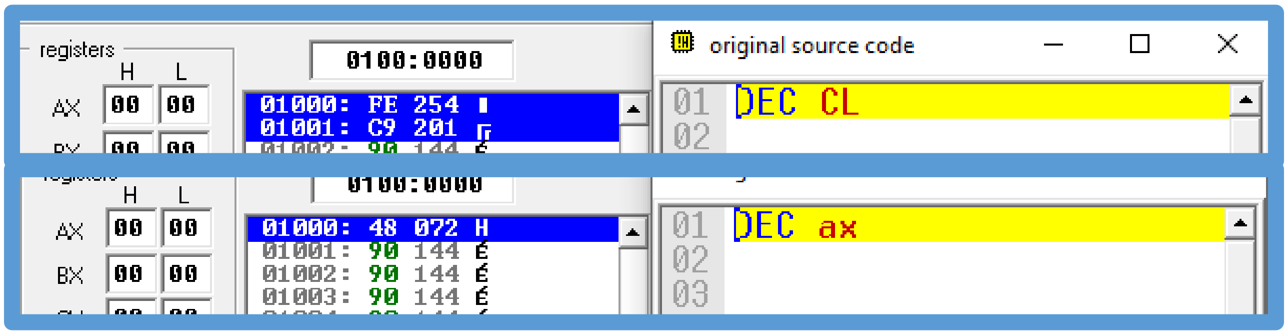
```
01000: FE 254 ▮
01001: 44 068 D
01002: FC 252 ⁿ
01003: 90 144 É
```

```
INC     BYTE PTR [SI - 4]
```

EX: DEC CL= FE C9

DEC = Decrement:		
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m
Register	0 1 0 0 1 reg	

Opcode w mod Cl ➔➔ = 11111110 11001 001= FEC9



```
registers        0100:0000
   H  L
AX 00 00        01000: FE 254 ▮
BX 00 00        01001: C9 201 ╔
                01002: 90 144 É

original source code
01 DEC CL
02
```

```
registers        0100:0000
   H  L
AX 00 00        01000: 48 072 H
BX 00 00        01001: 90 144 É
                01002: 90 144 É
                01003: 90 144 É

01 DEC ax
02
03
```

2 Immediate Operand Instructions

Immediate mode instructions have only one register or memory operand; the other is encoded in the instruction itself.

- In many instructions with immediate operands the "**d**" bit is interpreted as the "**s**" bit, which means that the single byte operand should be sign-extended to 16 bits

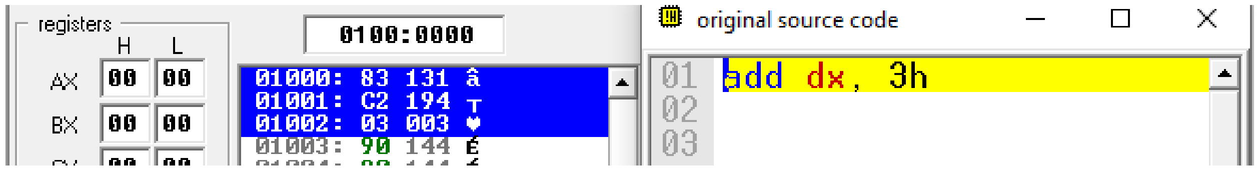
Example: add dx, 3h; (Add imm to reg16) ≤7

ADD = Add:				
Register/Memory with Register to Either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	

1000 00sw mod000r/m ➔ w=1 (DX is 16 bits) mod = 11, r/m = 010 =DX

With **s** bit **set** we have ➔ 10000011 11 000 010 = **83 C2 03**

With s bit **clear** we have: 10000001 11 000 010 Operand = 81 C2 03 **00**



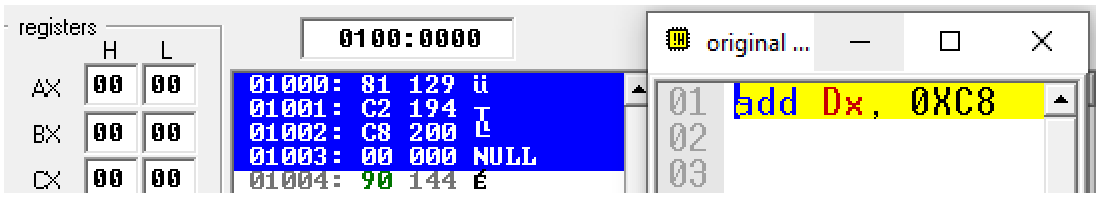
```
registers        0100:0000
   H  L
AX 00 00        01000: 83 131 â
BX 00 00        01001: C2 194 ┬
                01002: 03 003 ♥
                01003: 90 144 É

original source code
01 add dx, 3h
02
03
```

Example: add Dx, C8; ≥7

1000 00sw mod000r/m ➔ *S must equal to Zero(0)*

10000001 11 000 010➔ = 81 C2 C8 00(SW=01)

Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01



```
registers
     H    L
AX  00   00
BX  00   00
CX  00   00

0100:0000

01000: 81 129 ü
01001: C2 194 ┬
01002: C8 200 ╚
01003: 00 000 NULL
01004: 90 144 É
```

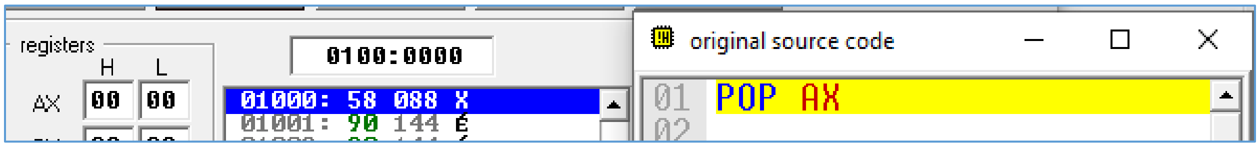
```
original ...
add Dx, 0XC8
```

3 One-Byte Opcode

EX: POP AX = 01011000 = 58H

POP = Pop:		
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m
Register	0 1 0 1 1 reg	
Segment Register	0 0 0 reg 1 1 1	

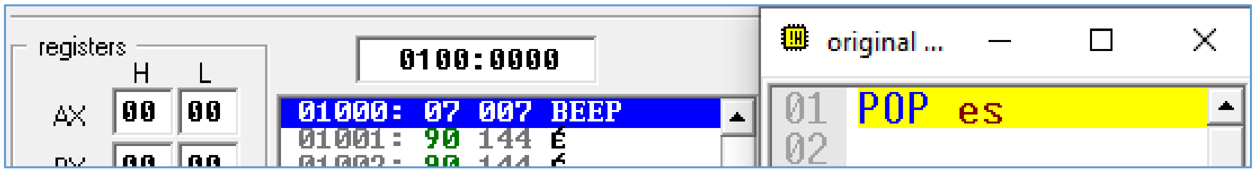
Note that there are two legal encodings of POP REG. Shorter form exists because POPs are so common. Most assemblers will use the shorter form.



```
registers
     H    L
AX  00   00

0100:0000

01000: 58 088 X
01001: 90 144 É
```

```
original source code
POP AX
```

EX: POP ES = 0000 0111 = 07H



```
registers
     H    L
AX  00   00

0100:0000

01000: 07 007 BEEP
01001: 90 144 É
```

```
original ...
POP es
```

xx	**Segment register**
00	ES
01	CS
10	SS
11	DS