The 8086 instruction set

BEKKOUCHE Mohammed

m.bekkouche@esi-sba.dz

Introduction

The instruction set is the set of machine instructions that a computer processor can execute. These machine instructions make it possible to carry out elementary or more complex operations. The instruction set defines which instructions are supported by the processor.

Eight groups of 8086 instructions

- Data transfer and conversion instructions (MOV, XCHG, CBW, ...)
- Instructions relating to flags (flag register) (CLC, STC, CMC, ...)
- Arithmetic instructions (ADD, SUB, MUL, DIV, ...)
- Logical instructions (NOT, AND, OR, XOR, ...)
- Shift instructions (SHR, SHL, ROL, ROR, ...)
- Branch instructions (JMP, Jxx, LOOP, ...)
- Instructions for handling tables or character strings (LODSB, LODSW, STOSB, STOSW, etc.)
- Interrupt Instructions

Data transfer and conversion instructions — MOV

Syntax

`MOV destination, source`

Copies the source operand to the destination operand.

- The MOV instruction cannot modify the value of the CS and IP registers.
- A segment register cannot be copied directly to another segment register.
- An immediate value cannot be copied directly to a segment register.

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

SREG, memory
memory, SREG
REG, SREG
SREG, REG

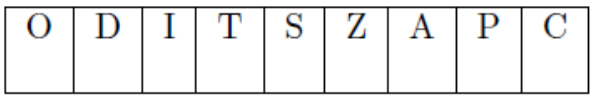
Algorithm

operand1 = operand2

Data transfer and conversion instructions — MOV

Examples

```
MOV AX, BX        ; Transfer from a 16-bit register to a 16-bit register
MOV AH, CL        ; Transfer from 8-bit register to 8-bit register
MOV AX, [Val1]    ; Transfer contents of Val1 to AX
MOV [Val2], AL    ; Transfer contents of AL to memory at Val2
```



Data transfer and conversion instructions — MOV

Remarks

It is strictly forbidden to transfer the contents of one memory slot directly to another:

```
MOV [Val1], [Val2]
```

Use two steps:

```
MOV AL, [Val2]
MOV [Val1], AL
```

Similarly, a segment register cannot be transferred directly to another segment register:

```
MOV AX, ES
MOV DS, AX
```

The CS is never used as destination register.

Data transfer and conversion instructions — LEA

LEA — Load Effective Address

It transfers the displacement address (shift) of a memory operand into a 16-bit register. This has the same role as the MOV instruction with offset.

`LEA BX, TAB_VAL` is equivalent to `MOV BX, offset TAB_VAL`

REG, memory

O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — LEA

Example 1

```
MOV BX, 35h
MOV DI, 12h
LEA SI, [BX+DI]     ; SI = 35h + 12h = 47h
```

Example 2

```
org 100h
LEA AX, m            ; AX = offset of m
RET
m dw 1234h
END
```

Data transfer and conversion instructions — XCHG

XCHG

It allows to switch the source with the destination.

REG, memory
memory, REG
REG, REG

Algorithm

operand1 $\leftrightarrow$ operand2

O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — XCHG

```
MOV AL, 5
MOV AH, 2
XCHG AL, AH     ; AL = 2, AH = 5
XCHG AL, AH     ; AL = 5, AH = 2
RET
```

Data transfer and conversion instructions — PUSH

Syntax: `PUSH SOURCE`

It allows to push a 16-bit value at the top of the stack.

Algorithm

$$SP = SP - 2$$
$$SS : [SP] = \text{operand}$$

REG
SREG
memory
immediate

O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — PUSH

```
MOV AX, 1456h
PUSH AX
POP DX          ; DX = 1456h
RET
```

Data transfer and conversion instructions — POP

Syntax: `POP destination`

It allows you to pop a 16-bit value from the top of the stack.

Algorithm

$$\text{operand} = SS : [SP]$$
$$SP = SP + 2$$

REG
SREG
memory

O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — POP

```
MOV AX, 1456h
PUSH AX
POP DX          ; DX = 1456h
RET
```

Data transfer and conversion instructions — PUSHA

Stacks all general purpose registers AX, CX, DX, BX, SP, BP, SI, DI into the stack.

Algorithm

```
PUSH AX
PUSH CX
PUSH DX
PUSH BX
PUSH SP
PUSH BP
PUSH SI
PUSH DI
```

The original value of the SP register (before PUSHA) is used.

O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — POPA

Pops all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack.

Algorithm

```
POP DI
POP SI
POP BP
POP xx   ; SP value is ignored
POP BX
POP DX
POP CX
POP AX
```

SP value is ignored: it is popped but does not modify the SP register.

O	D	I	T	S	Z	A	P	C

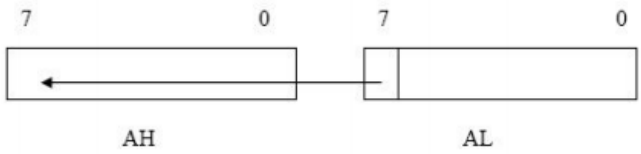
Data transfer and conversion instructions — CBW

CBW — Convert Byte to Word

Extends AL into AH respecting the sign.

Algorithm

If the significant bit of AL = 1, then AH = 255 (0FFh). Otherwise, AH = 0.



O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — CBW

```
MOV AX, 0          ; AH=0, AL=0
MOV AL, -5         ; AX = 000FBh (251)
CBW                ; AX = 0FFFBh (-5)
RET
```

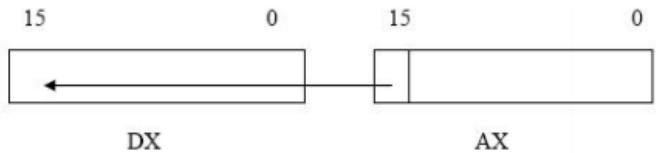
Data transfer and conversion instructions — CWD

CWD — Convert Word to Double Word

Extends AX in DX respecting the sign.

Algorithm

If the significant bit of AX = 1, then DX = 65535 (0FFFFh). Otherwise, DX = 0.



O	D	I	T	S	Z	A	P	C

Data transfer and conversion instructions — CWD

```
MOV DX, 0          ; DX = 0
MOV AX, 0          ; AX = 0
MOV AX, -5         ; DX AX = 00000h:0FFFBh
CWD                ; DX AX = 0FFFFh:0FFFBh
RET
```

Instructions relating to flags

CLC	CLear Carry: sets C to 0.
STC	Set Carry: sets C to 1.
CMC	Complements the C flag.
CLD	Sets D to 0.
STD	Sets D to 1.
CLI	Sets I to 0.
STI	Sets I to 1.

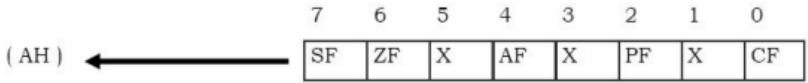
Direction and Interrupt flags

DF = 0: left to right in memory.
DF = 1: right to left in memory.
When IF = 1, the processor allows maskable interrupts. When IF = 0, the processor ignores those interrupts. An NMI is a hardware interrupt that cannot be disabled or ignored by the CPU.

Instructions relating to flags — LAHF / SAHF

LAHF — Load AH from Flags

Copies the low byte of the status register into AH.



SAHF — Store AH into Flags

Inverse operation of LAHF: transfers AH into the low byte of the status register.

O	D	I	T	S	Z	A	P	C
				*	*	*	*	*

Instructions relating to flags — PUSHF / POPF

PUSHF

Stacks the FLAGS register. SP is diminished by 2.

$$SP = SP - 2$$

$$SS : [SP] = flags$$

O	D	I	T	S	Z	A	P	C

POPF

Pops the FLAGS register. SP is increased by 2.

$$flags = SS : [SP]$$

$$SP = SP + 2$$

O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

Arithmetic instructions — ADD

ADD — addition without carry

`ADD AX, 5` means: $AX \leftarrow AX + 5$.

Syntax

`ADD destination, source`

It adds the content of the source (byte or word) with that of the destination; the result is put in the destination.

Algorithm

operand1 = operand1 + operand2

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

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Arithmetic instructions — ADD

The C flag is set according to the result of the operation.

Example 1

```
MOV AL, 0A9h
ADD AL, 72h
```

Then AL = 1Bh, C = 1, A = 0.

Example 2

```
MOV AL, 09h
ADD AL, 3Ah
```

Then AL = 43h, C = 0, A = 1.

Arithmetic instructions — ADC

ADC — addition with carry

`ADC AX, 5` means: $AX \leftarrow AX + 5 + C$.

Syntax

`ADC Destination, source`

It adds the content of the source with the destination and the carry CF; the result is put in the destination.

Algorithm

operand1 = operand1 + operand2 + CF

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

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Arithmetic instructions — ADC

Example of addition on 4 bytes

We want to add data located in AX and DX to data found in BX and CX and store the result in AX and DX. AX and BX contain the low-order bytes; DX and CX the high-order bytes.

```
ADD AX, BX
ADC DX, CX
```

Arithmetic instructions — ADC

```
ADC AX, BX        ; AX = AX + BX + CF (16-bit addition)
ADC AL, BH        ; AL = AL + BH + CF (8-bit addition)
ADC AL, [SI]      ; AL = AL + contents pointed to by SI + CF
ADC [DI], AL      ; memory at DI is added with AL + CF
```

Arithmetic instructions — SUB

SUB — subtraction without carry

`SUB AX, 3` means: $AX \leftarrow AX - 3$.

Syntax

`SUB Destination, source`

It subtracts the source from the destination (byte or word); the result is put in the destination.

Algorithm

operand1 = operand1 - operand2

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

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Arithmetic instructions — SUB

The C flag is positioned according to the result of the operation.

Example 1

```
MOV AL, 39h
SUB AL, 18h
```

AL = 21h. Z, S and C are 0 because the result is not zero, its sign is positive and no carry is generated.

Example 2

```
MOV AL, 26h
SUB AL, 59h
```

AL = CDh. Z = 0; C, A and S = 1.

Arithmetic instructions — SUB

```
SUB AX, BX        ; AX = AX - BX (16-bit subtraction)
SUB AL, BH        ; AL = AL - BH (8-bit subtraction)
SUB AL, [SI]      ; AL = AL - contents pointed to by SI
SUB [DI], AL      ; memory at DI is subtracted from AL; result at DI
```

Arithmetic instructions — SBB

SBB — subtraction with carry

`SBB AX, 3` means: $AX \leftarrow AX - 3 - C$.

Syntax

`SBB Destination, source`

It subtracts the source and the carry from the destination.

Algorithm

operand1 = operand1 - operand2 - CF

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

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Arithmetic instructions — SBB

Example

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

Example of 4 bytes subtraction

To subtract data in BX and CX from data in AX and DX and store the result in AX and DX:

```
SUB AX, BX
SBB DX, CX
```

Arithmetic instructions — INC

INC — Increment

`INC AX` means: $AX \leftarrow AX + 1$. It does not touch the C flag.

Syntax

```
INC Destination
```

Adds 1 to the contents of the operand, without modifying the carry flag.

Algorithm

operand = operand + 1

REG
memory

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

Arithmetic instructions — INC

```
MOV AL, 3Fh
INC AL
```

AL then contains 40h. The Z bit is 0, A is 1, and O and S are 0.

Arithmetic instructions — DEC

DEC — Decrement

`DEC AX` means: $AX \leftarrow AX - 1$. It does not touch the C flag.

Syntax

`DEC Destination`

Subtracts 1 from the contents of the operand, without changing the carry flag.

Algorithm

operand = operand - 1

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	

Arithmetic instructions — DEC

Example 1

```
MOV AL, 01h
DEC AL
```

AL = 00. Z = 1; O, P, A and S are 0.

Example 2

```
DEC AX        ; AX = AX - 1 (16-bit decrement)
DEC AL        ; AL = AL - 1 (8-bit decrement)
DEC [SI]      ; contents pointed to by SI are decremented
```

Arithmetic instructions — MUL / IMUL

Multiplication (unsigned / signed)

`MUL/IMUL Source`

MUL performs unsigned multiplication; IMUL performs signed multiplication. Carry C and overflow O are set to 1 if the result cannot be stored in the destination operand.

Algorithm

For a byte operand: $AX = AL \times operand$.
For a word operand: $(DX\ AX) = AX \times operand$.

O	D	I	T	S	Z	A	P	C
*				?	?	?	?	*

Arithmetic instructions — MUL / IMUL

Example 1 — MUL

```
MOV AL, 200       ; AL = 0C8h
MOV BL, 5
MUL BL            ; AX = 03E8h (1000)
RET
```

Example 2 — IMUL

```
MOV AL, -2
MOV BL, -4
IMUL BL           ; AX = 8
RET
```

Arithmetic instructions — DIV / IDIV

Division (unsigned / signed)

`DIV/IDIV Source`

DIV performs unsigned division; IDIV performs signed division and remainder calculation.

Algorithm

For a byte operand:

$$AL = AX/operand$$
$$AH = remainder$$

For a word operand:

$$AX = (DX\ AX)/operand$$
$$DX = remainder$$

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

REG
memory

Arithmetic instructions — DIV / IDIV

Example 1

```
MOV AX, 457       ; AX=01C9h
MOV BL, 4
DIV BL            ; AL = 114 (72h), AH = 1
RET
```

Example 2

```
MOV AX, -457      ; AX = 0FE37h
MOV BL, 4
IDIV BL           ; AL = -114 (08Eh), AH = -1 (0FFh)
RET
```

Arithmetic instructions — NEG

NEG — Negation by 2's complement

`NEG AX` changes a value to its two's-complement negation.

Algorithm

Invert all bits of the operand; add 1 to the inverted operand.

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

REG

memory

Arithmetic instructions — NEG

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

Arithmetic instructions — CMP

CMP

Compares the operands and sets the flags according to the result. The operand is not modified.

Syntax

```
CMP Destination, Source
```

CMP is used to set flags before a conditional jump. It subtracts source from destination, but the result is not stored. The flags likely to be affected are OF, SF, ZF, AF, PF, CF.

Algorithm

operand1 - operand2; result is not stored

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

O	D	I	T	S	Z	A	P	C
*				*	*	*	*	*

Arithmetic instructions — CMP

```
MOV AL, 44
CMP AL, 54
```

AL is not modified and still contains 44. C is set to 1, indicating that the second operand is greater than the first.
Z is 0, indicating that the two data values are different.
S is 1 because $44 - 54$ is negative.