The 8086 instruction set

BEKKOUCHE Mohammed

m.bekkouche@esi-sba.dz

1

The logic instructions — AND

Syntax: `AND Destination, source`

Performs a logical AND bit by bit between the source operand and the destination operand. The result is stored in the destination operand.

Rules

$$1 \wedge 1 = 1 \quad 1 \wedge 0 = 0 \quad 0 \wedge 1 = 0 \quad 0 \wedge 0 = 0$$

```
AND AL, 01011100b
; AL = 00110110b
; result = 00010100b
```

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

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

The logic instructions — AND

Example

Consider the following two lines:

```
MOV AL, 36h
AND AL, 5Ch
```

AL then contains 14h.

36h	0011	0110
5Ch	0101	1100
14h	0001	0100

S and Z bits are set to zero and the P bit to 1.

The logic instructions — OR

Syntax: `OR Destination, source`

Performs a bitwise OR between source and destination. The result is stored in the destination operand.

Rules

$$1 \vee 1 = 1 \quad 1 \vee 0 = 1 \quad 0 \vee 1 = 1 \quad 0 \vee 0 = 0$$

```
OR AL, 01011100b
; AL = 00110110b
; result = 01111110b
```

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

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

The logic instructions — OR

Example

```
MOV AL, 36h
OR AL, 5Ch
```

AL then contains 7Eh.

36h	0011 0110
5Ch	0101 1100
7Eh	0111 1110

S and Z bits are set to zero and the P bit to 1.

The logic instructions — XOR

Syntax: `XOR Destination, source`

Performs a bitwise exclusive-OR between the source and destination operands. The result is stored in the destination operand.

Rules

$$1 \oplus 1 = 0 \quad 1 \oplus 0 = 1 \quad 0 \oplus 1 = 1 \quad 0 \oplus 0 = 0$$

```
XOR AL, 01011100b
; AL = 00110110b
; result = 01101010b
```

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

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

The logic instructions — XOR

Example

```
MOV AL, 36h
XOR AL, 5Ch
```

The register AL then contains `6Ah`.

36h	0011	0110
5Ch	0101	1100
6Ah	0110	1010

S and Z bits are set to zero and the P bit to 1.

The logic instructions — NOT

Syntax: `NOT Destination`

Realizes the complementation to 1 of a number.

Algorithm

If the bit is 1, set it to 0.
If the bit is 0, set it to 1.

```
NOT AL
; 00110110b   -> 11001001b
```

O	D	I	T	S	Z	A	P	C

REG

memory

The logic instructions — NOT

Example

```
MOV AL, 36h
NOT AL
```

The register AL then contains C9h.

```
36h   0011 0110
C9h   1100 1001
```

The FLAGS register is not changed by the NOT instruction.

The logic instructions — TEST

Syntax: `TEST Destination, source`

Similar to AND, but only the flags are set. The logical AND is performed without modifying the destination operand.

$$1 \wedge 1 = 1 \quad 1 \wedge 0 = 0 \quad 0 \wedge 1 = 0 \quad 0 \wedge 0 = 0$$

```
TEST AL, 1
TEST AL, 10b
```

O	D	I	T	S	Z	A	P	C
0				*	*	?	*	0

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

The logic instructions — TEST

Example

```
MOV AL, 00000101b
TEST AL, 1      ; ZF=0, PF=0, SF=0
TEST AL, 10b    ; ZF=1, PF=1, SF=0
RET
```

The shift and rotate instructions

We describe two classic assembly operations: shifts and rotations, used for efficient data manipulation.

- Rotation treats a byte or word as a circular structure (torus).
- Shift operations move all bits to the left or right, treating the data as a linear sequence.

Syntax

`INST destination, counter`

counter = number of shifts

O	D	I	T	S	Z	A	P	C
*								*

memory, immediate
REG, immediate

memory, CL
REG, CL

The rotation instructions — RCL

RCL: Left rotation with carry

The most significant bit is put in the CF carry indicator, while the previous CF value is placed in the least significant bit.

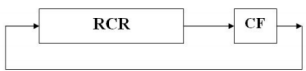
```
STC                 ; set carry (CF=1)
MOV AL, 1Ch         ; AL=00011100b
RCL AL, 1           ; AL=00111001b, CF=0
RET
```

The rotation instructions — RCR

RCR: Right rotation with carry

The least significant bit of the destination operand is put in CF, while the previous CF value is placed in the most significant bit.

```
STC                     ; set carry (CF=1)
MOV AL, 1Ch             ; AL=00011100b
RCR AL, 1               ; AL=10001110b, CF=0
RET
```



The rotation instructions — ROL

ROL: Left rotation with carry crushing

The most significant bit is placed in CF and in the low-order bit of the operand. The old value of CF is not used.

```
MOV AL, 1Ch          ; AL=00011100b
ROL AL, 1            ; AL=00111000b, CF=0
RET
```

The rotation instructions — ROR

ROR: Right rotation with carry crushing

The low-order bit is placed in CF and in the high-order bit of the operand. The old value of CF is not used.

```
MOV AL, 1Ch          ; AL=00011100b
ROR AL, 1            ; AL=00001110b, CF=0
RET
```

The shift instructions — SAL / SHL

SAL / SHL: Left shift with 0 input and CF out

SAL and SHL are synonyms. A left shift saves the most significant bit in CF and puts 0 in the least significant bit.

```
MOV AL, 0E0h          ; AL=11100000b
SAL AL, 1             ; AL=11000000b, CF=1
RET
```

The shift instructions — SHR

SHR: Right shift with 0 input and CF out

SHR shifts to the right. The least significant bit is put in CF and 0 is inserted into the most significant bit.

```
MOV AL, 00000111b
SHR AL, 1            ; AL=00000011b, CF=1
RET
```

The shift instructions — SAR

SAR: Right shift with CF output and duplication of the MSB

SAR performs a right shift, saves the least significant bit in CF, and keeps the original most significant bit.

```
MOV AL, 0D0h          ; AL=11010000b
SAR AL, 1             ; AL=11101000b, CF=0
MOV BL, 4Ch           ; BL=01001100b
SAR BL, 1             ; BL=00100110b, CF=0
RET
```

D0h=-48 E8h=-24 4Ch=76 26h=38

The shift instructions

Important observation

A shift of one position to the left corresponds to multiplication by 2, while a shift of one position to the right corresponds to integer division by 2.

Generalization

A shift of l positions to the left corresponds to multiplication by 2^l, while a shift of l positions to the right corresponds to division by 2^l.

Signed versus unsigned division

`SAR` performs the right-shift division for signed representation, while `SHR` performs the corresponding operation for unsigned representation.

Logic and shift instructions — Exercise

Exercise

Give the output of the following 8086 assembler program:

```
ORG 100h
MOV AX, 1101110011110000b
MOV BX, 0DCh
MOV CX, 16
Label:
    SHL AX, 2
    RCR BX, 2
    LOOP Label
MOV AX, BX
RET
```

Reminder: LOOP

LOOP decrements CX by 1 and transfers control to the label while CX is not zero.

Logic and shift instructions — Solution

Initial values

```
AX = 1101110011110000     BX = 0000000011011100
```

1st and 2nd iterations

```
1st: AX = 01110011110000 00  CF=1
     BX = 01 00000000110111  CF=0
2nd: AX = 11001111000000 00  CF=1
     BX = 11 01000000001101  CF=1
```

Logic and shift instructions — Solution

3rd to 5th iterations

```
3rd: AX = 00111100000000 00  CF=1
     BX = 11 11010000000011  CF=0
4th: AX = 11110000000000 00  CF=0
     BX = 10 11 110100000000  CF=1
5th: AX = 11000000000000 00  CF=1
     BX = 01 10111101000000  CF=0
```

Logic and shift instructions — Solution

6th to 8th iterations

```
6th: AX = 00000000000000 00  CF=1
     BX = 01 01101111010000  CF=0
7th: AX = 00000000000000 00  CF=0
     BX = 00 01 011011110100  CF=0
8th: AX = 00000000000000 00  CF=0
     BX = 00 00 010110111101  CF=0
```

Logic and shift instructions — Solution

9th to 11th iterations

```
9th : AX = 00000000000000 00  CF=0
      BX = 10 00 000101101111  CF=0
10th: AX = 00000000000000 00  CF=0
      BX = 10 10 000001011011  CF=1
11th: AX = 00000000000000 00  CF=0
      BX = 10 101 00000010110  CF=1
```

Logic and shift instructions — Solution

12th to 14th iterations

```
12th: AX = 00000000000000 00  CF=0
      BX = 00 10101000000101  CF=1
13th: AX = 00000000000000 00  CF=0
      BX = 10 00 101010000001  CF=0
14th: AX = 00000000000000 00  CF=0
      BX = 10 10001010100000  CF=0
```

Correction noted in the source

The 12th iteration line is marked as an error in the original material: the indicated CF value should be corrected according to the intended trace.

Logic and shift instructions — Solution

15th and 16th iterations

```
15th: AX = 00000000000000 00  CF=0
      BX = 00 10 100010101000  CF=0
16th: AX = 00000000000000 00  CF=0
      BX = 00 00101000101010  CF=0
```

Result

AX = 0A2Ah

```
MOV AX, BX
RET
```