Microprocessors and Interfaces: Lecture 14
8086 Logical Instructions : Part-1

By Dr. Sanjay Vidhyadharan

Reference : UNDERGRADUATE TEXTS IN COMPUTER SCIENCE
Springer Science+Business Media, LLC
Editors
David Gries
Fred B. Schneider

Logical Instructions

The logic instructions include

- AND
- OR
- Exclusive-OR
- NOT
- NEG
- Shifts
- Rotates
- TEST (logical compare).

All are Bit wise Operations

Logical Instructions

AND Destination, Source

E.g. AND AL BL

Before execution of and		After execution of and			
AL	BL	AL	ZF	SF	PF
0101 1110	1110 0001	0100 0000	0	0	0
0110 1011	1001 0100	0000 0000	1	0	1
1111 1111	1111 1111	1111 1111	0	1	1
1001 0110	0110 1001	0000 0000	1	0	1

- CF and OF both become zero
- PF, SF and ZF affected
- AF undefined

Logical Instructions

AND Destination, Source

E.g

AND DX,[SI]

Bitwise AND of contents of DX register (word) with word from memory location with offset SI and result stored in DX

- CF and OF both become zero
- PF, SF and ZF affected
- AF undefined

Logical Instructions

Usage of AND Instruction

1. Clearing BITS/Masking

```
          AL = 11010110   ← operand to be manipulated
          BL = 11111100   ← mask byte
and   AL,BL = 11010100
```

```
x x x x  x x x x   Unknown number
0 0 0 0  1 1 1 1   Mask
---------------
0 0 0 0  x x x x   Result
```

2. Clearing Sign Bit/(Parity Bit)

```
and     AL,7FH
```

7FH (01111111B)

Logical Instructions

Usage of AND Instruction

3. ASCII-to-numeric conversion

Decimal digit	ASCII code (in binary)	8-bit binary code (in binary)
0	0011 0000	0000 0000
1	0011 0001	0000 0001
2	0011 0010	0000 0010
3	0011 0011	0000 0011
4	0011 0100	0000 0100
5	0011 0101	0000 0101
6	0011 0110	0000 0110
7	0011 0111	0000 0111
8	0011 1000	0000 1000
9	0011 1001	0000 1001

and AL,0FH

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(	0010 1000
N	0100 1110	l	0110 1100	)	0010 1001
				space	0010 0000

4. Finding an odd or even number

```
and AL,1 ;          mask = 00000001
jz even_number
odd_number:
```

Logical Instructions

OR Destination, Source

Before execution of or		After execution of or			
AL	BL	AL	ZF	SF	PF
0100 0010	0000 1010	0100 1010	0	0	0
0000 0000	0000 0000	0000 0000	1	0	1
1010 0000	0000 1111	1010 1111	0	1	1
1000 0000	1000 0000	1000 0000	0	1	0

- CF and OF both become zero
- PF, SF and ZF affected
- AF undefined

Logical Instructions

Usage of OR Instruction

1. Setting BITS/Masking

```
                AL = 11010110B  ← operand to be manipulated
                BL = 00000011B  ← mask byte
or         AL,BL = 11010111B
```

```
  x x x x  x x x x   Unknown number
+ 0 0 0 0  1 1 1 1   Mask
-------------------
  x x x x  1 1 1 1   Result
```

2. Setting Sign Bit/(Parity Bit)

```
or      AL,80H
```

Logical Instructions

Usage of OR Instruction

3. Conversion of digits to ASCII characters

Decimal digit	ASCII code (in binary)	8-bit binary code (in binary)
0	0011 0000	0000 0000
1	0011 0001	0000 0001
2	0011 0010	0000 0010
3	0011 0011	0000 0011
4	0011 0100	0000 0100
5	0011 0101	0000 0101
6	0011 0110	0000 0110
7	0011 0111	0000 0111
8	0011 1000	0000 1000
9	0011 1001	0000 1001

or AL,30H

Logical Instructions

XOR Destination, Source

Before execution of or		After execution of or			
AL	BL	AL	ZF	SF	PF
0100 0010	0001 1010	0101 1000	0	0	0
0110 1101	0110 1101	0000 0000	1	0	1
1010 0110	0101 1001	1111 1111	0	1	1
1011 0110	0000 1111	1011 1001	0	1	0

- CF and OF both become zero
- PF, SF and ZF affected
- AF undefined

Logical Instructions

Usage of XOR Instruction

1. Toggling/Inverting Bits

```
                AL = 11010110B  ← operand to be manipulated
                BL = 01010101B  ← mask byte
xor         AL,BL = 10000011B
```

2. Parity conversion

```
    01000001B  ← even-parity encoded ASCII character A
xor 10000000B  ← mask byte
    11000001B  ← odd-parity encoded ASCII character A
```

Logical Instructions

Usage of XOR Instruction

3. Data encryption

```
01000010B  ← ASCII character B
00100110B  ← encryption key (mask)
01100100B  ← ASCII character d
```

4. Initialization of Registers

```
mov     AX,0
```

but the same result can be achieved by

```
xor     AX,AX
```

The xor instruction affects flags, whereas the mov instruction does not.

Logical Instructions

NOT Destination

Truth table for the not operation

Input bit	Output bit
destination b_i	destination b_i
0	1
1	0

AL contains 00110101 B before executing
not AL
the AL register will have 11001010 B

- CF and OF both become zero
- PF, SF and ZF affected
- AF undefined

Logical Instructions

Usage of NOT Instruction

1’s Complementing Data

not AL : 1’s completement
inc AL : 2’s completement
Neg instruction is also available for : 2’s completement

Logical Instructions

TEST Source1, Source2

The test instruction Performs a bitwise logical AND operation between two operands, but, unlike the and instruction, test does not alter the destination operand. That is, test is a nondestructive and instruction. PF, SF and ZF affected

Usage

and AL,1 destroys the contents of the AL register.

If our purpose is to test whether the unsigned number in the AL register is an odd number, we can do this using test without destroying the original number.

```
        test    AL,1        ; mask = 00000001B
        jz      even_number
odd_number:
                .
        <code for processing odd number>
                .
even_number:
                .
        <code for processing even number>
```

Thankyou