Logical Operators

ASCII Codes

The Charcter set of the ASCII Code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Examples:
  - ASCII code for space character = 20 (hex) = 32 (decimal)
  - ASCII code for ‘A' = 41 (hex) = 65 (decimal)
  - ASCII code for 'a' = 61 (hex) = 97 (decimal)

Control Characters

- The first 32 characters of ASCII table are used for control
- Control character codes = 00 to 1F (hex)
- Examples of Control Characters
  - Character 0 is the NULL character ⇒ used to terminate a string
  - Character 9 is the Horizontal Tab (HT) character
  - Character 0A (hex) = 10 (decimal) is the Line Feed (LF)
  - Character 0D (hex) = 13 (decimal) is the Carriage Return (CR)
  - The LF and CR characters are used together
    - They advance the cursor to the beginning of next line
- One control character appears at end of ASCII table
  - Character 7F (hex) is the Delete (DEL) character

ASCII Groups

- 4 groups of 32 characters
- G1 → Codes 0-1F (31) [Non printing Control characters]
- G2 → Codes 30-39H [0---9] + punctuation symbols etc
  - Numeric digits differs in H.O Nibble from their numeric values
  - Subtract 30 to get numeric value
- G3 → Codes 41-5A (65-90) [A---Z + 6 codes for special symbols]
- G4 → Codes 61-7A [a—z] +special symbols
  - Upper and Lower differs in bit number 5

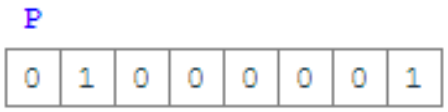
ASCII Groups

- Bits 5 and 6 determines the groups as shown in table below

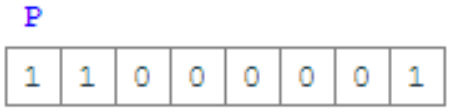
Bit 6	Bit 5	Group
0	0	Control Characters
0	1	Digits & Punctuation
1	0	Upper Case & Special
1	1	Lower Case & Special

Parity Bit

- Data errors can occur during data transmission or storage/retrieval.
- The 8th bit in the ASCII code is used for error checking.
- This bit is usually referred to as the parity bit.
- There are two ways for error checking:
  - Even Parity: Where the 8th bit is set such that the total number of 1s in the 8-bit code word is even.



  - Odd Parity: The 8th bit is set such that the total number of 1s in the 8-bit code word is odd.



Boolean Operations

- NOT
- AND
- OR
- Operator Precedence
- Truth Tables

Boolean Algebra

- Based on symbolic logic, designed by George Boole
- Boolean expressions created from:
  - NOT, AND, OR

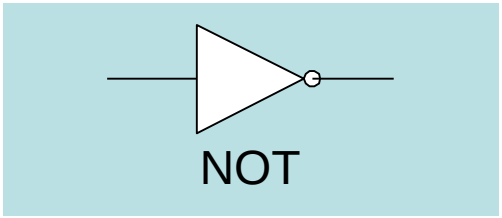
Expression	Description
$\neg X$	NOT X
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y
$\neg X \vee Y$	(NOT X) OR Y
$\neg (X \wedge Y)$	NOT (X AND Y)
$X \wedge \neg Y$	X AND (NOT Y)

NOT

- Inverts (reverses) a boolean value
- Truth table for Boolean NOT operator:

X	¬X
F	T
T	F

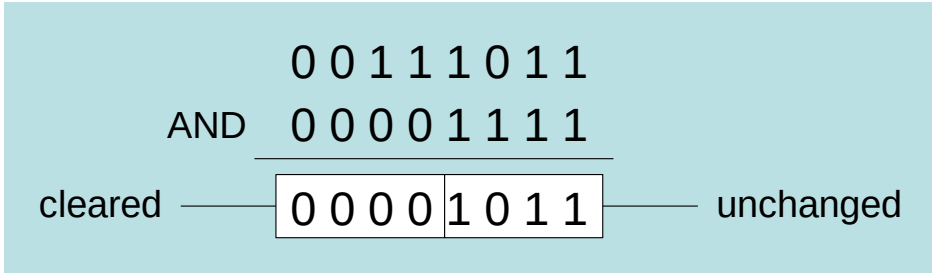
Digital gate diagram for NOT:



AND

- Performs a Boolean AND operation between each pair of matching bits in two operands
- Syntax:
  - AND *destination, source*

(same operand types as MOV)

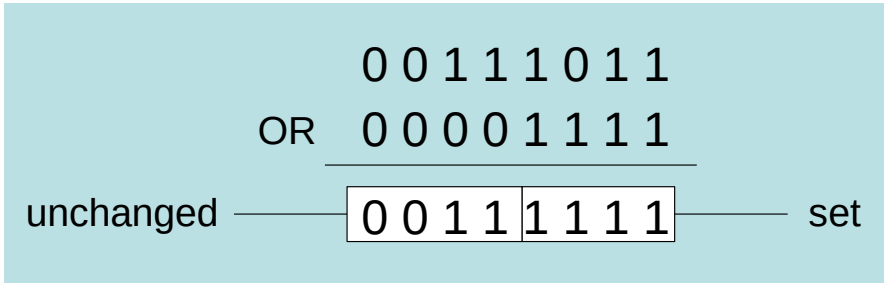


x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1

- Forcing ZERO/FALSE result (AND with ZERO)
- If NO CHANGE required, AND with ONE

OR Instruction

- Performs a Boolean OR operation between each pair of matching bits in two operands
- Syntax:
  - OR *destination, source*



OR

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

- Forcing ONE (OR with ONE)
- NO CHANGE (OR with ZERO)

XOR Instruction

- Performs a Boolean exclusive-OR operation between each pair of matching bits in two operands
- Syntax:
  - XOR *destination, source*

```
    0 0 1 1 1 0 1 1
XOR 0 0 0 0 1 1 1 1
    -----------------
    0 0 1 1 | 0 1 0 0
unchanged       inverted
```

XOR

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

XOR is a useful way to toggle (invert) the bits in an operand.

INVERSING OPERAND (XOR with ONE)
NO CHANGE (XOR with ZERO)

NOT Instruction

- Performs a Boolean NOT operation on a single destination operand
- Syntax:
  - NOT *destination*

```
NOT    0 0 1 1 1 0 1 1
       ---------------
       1 1 0 0 0 1 0 0  ——— inverted
```

NOT

X	¬X
F	T
T	F

Applications (1 of 5)

- Task: Convert the character in AL to upper case.
- Solution: Use the AND instruction to clear bit 5.

```
mov al,'a'                  ; AL = 01100001b
and al,11011111b            ; AL = 01000001b
```

Applications (2 of 5)

- Task: Convert a binary decimal byte into its equivalent ASCII decimal digit.
- Solution: Use the OR instruction to set bits 4 and 5.

```
mov al,6                    ; AL = 00000110b
or  al,00110000b            ; AL = 00110110b
```

The ASCII digit '6' = 00110110b

Applications (4 of 5)

- Task: Jump to a label if an integer is even.
- Solution: AND the lowest bit with a 1. If the result is Zero, the number was even.

```
mov ax,wordVal
and ax,1                    ; low bit set?
jz  EvenValue               ; jump if Zero flag set
```

Your turn: Write code that jumps to a label if an integer is negative.

Applications (5 of 5)

- Task: Jump to a label if the value in AL is not zero.
- Solution: OR the byte with itself, then use the JNZ (jump if not zero) instruction.

```
or  al,al
jnz IsNotZero              ; jump if not zero
```

ORing any number with itself does not change its value.

TEST Instruction

- Performs a nondestructive AND operation between each pair of matching bits in two operands
- No operands are modified, but the Zero flag is affected.
- Example: jump to a label if either bit 0 or bit 1 in AL is set.

```
test al,00000011b
jnz  ValueFound
```

- Example: jump to a label if neither bit 0 nor bit 1 in AL is set.

```
test al,00000011b
jz   ValueNotFound
```

Thanks!