Lecture 19: Boolean Algebra

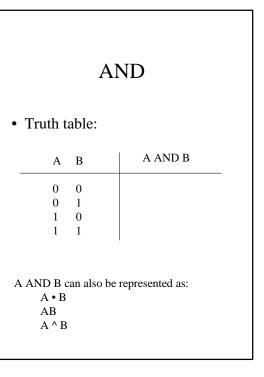
- Basic Boolean Algebra
- Boolean Functions in Assembly

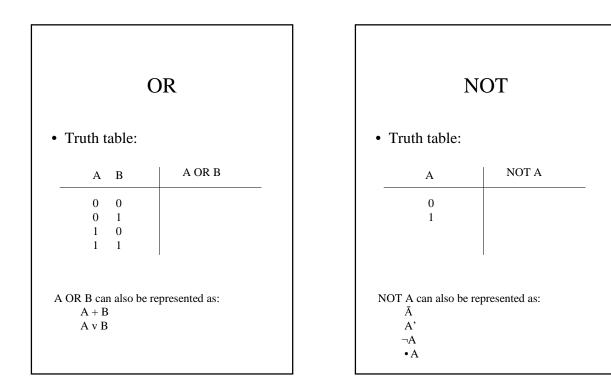
Boolean Algebra

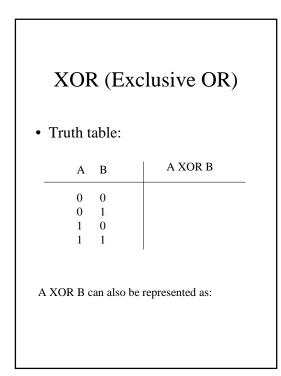
- Two valued algebra
- Used to analyze the basic elements that digital computers are built from.
- A way of manipulating true/false values.



- Basic operations:
 - AND true iff both operands are true
 - OR true if either or both operands are true
 - NOT true when its operand is false (inverts the operand)
- Other common operations:
 - XOR true if inputs are different
 - NAND inversion of AND
 - NOR inversion of OR
- 1 =true, 0 =false



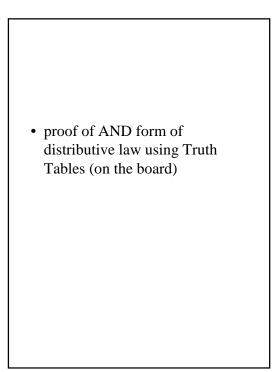




16 Possible Boolean Functions of Two Variables

• table from AoA, Chapter 2

Identity law	$1\mathbf{A} = \mathbf{A}$	0 + A = A
Null law	$0\mathbf{A} = 0$	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse Law	$A\bar{A} = 0$	$A + \overline{A} = 1$
Communative Law	AB = BA	$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$
Associative Law	(AB)C = A(BC)	(A + B) + C = A $+ (B + C)$
Distributive Law	A + BC = (A + B)(A + C)	$\begin{array}{c} A(B+C) = AB + \\ AC \end{array}$
Absorption Law	A(A + B) = A	A + AB = A
De Morgan's Law		



 proof of OR-form of DeMorgan's Law using truth tables (on board)

Generating a Logic Function from a Truth Table

АВС	M
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

- Find all the combinations that result in a one.
- Put them into an expression:

Boolean Functions in Assembly

- Boolean functions fall into the category of bit-operations.
- What other bit operations have we seen?
- For boolean functions, the operations take place between the individual bits of the two operands.

AND

- AND performs a bitwise AND operation between each bit of the two operands and places the result in the first operand.
- Formats: AND reg, reg AND reg, mem AND reg, immed AND mem, reg AND mem, immed

AND, cont.

 AND can clear selected bits in an operand while preserving (masking) the remaining bits. mov al, 00111011b and al, 00001111b; al = 00001011b

The 00001111b is called a bit mask, it clears the upper four bits while preserving the lower four bits.

AND Example

- Converting from lower case to upper case.
- Upper case letters have bit 5 set .data char db ?;put uppercase letter here mask db 0DFh ; 11011111b .code mov ah, 1 int 21h :get the char into A
 - and al, mask mov char, al
- ;get the char into AL ;mask out bit 5 ;store uppercase char

OR

- Performs a bitwise OR operation between each bit of the two operands and places the result in the first operand.
- Same formats as AND.
- OR is useful for setting certain bits to one while leaving the other bits unchanged.

mov al, 00111011b ;3Bh

- or al, 00001111b ;AL = 3Fh
- the lower four bits of the result are set, the others remain unchanged.

OR Example

- Converting from upper case to lower.
- When we converted the other way, we cleared bit 5. Now we need to set it:

.data char db ?;put lowercase letter here setb db 20h ; 00100000b .code mov ah, 1 int 21h ;get the char into AL or al, setb;set bit 5 mov char, al ;store lowercase char

Another OR Example

;converting a decimal digit to ASCII

DIGIT DW 7 ASCBias DW 30h

••••

MOV AX, DIGIT OR AX, ASCBias

Checking for Set Bits

• AND can be used to see if a bit is set in a word: ; test if bit 2 of BX = 0. If yes, jump mov ax, bx ; save original bx and ax. 0004h : zero out all but bit 2 ;if zero (bit 2 zero), jump jz zbit • You can also use the TEST instruction: it does an AND but doesn't load results (implied AND) ; test using TEST TEST bx, 0004h ;BX not changed JΖ zbit ;but flags are set

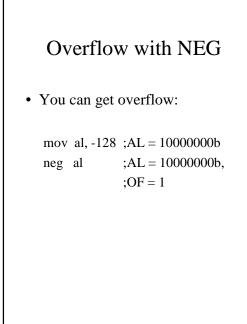


- NOT reverses all the bits in an operand (takes the 1's complement).
- Formats:
 - NOT reg
 - NOT mem
 - mov al, 11110000b not al ;al = 00001111b

NEG

- NEG reverses the sign of a number by converting it to its two's complement.
- Formats:
 - NEG reg
 - NEG mem

mov al, +127; AL = 01111111b neg al; AL = 10000001b



XOR

• Performs a bit-by-bit exclusive OR, puts the result in the first operand.

mov al, 10110100b

- xor al, 10000110b ; al = 00110010b
- Commonly used to set a register to zero:

XOR AX, AX ; same effect as mov ax, $\boldsymbol{0}$