

Computer Organization and Assembly Language

Assembly Instructions

Instructions

- ▶ An *instruction* is a statement that becomes executable when a program is assembled.
- ▶ Instructions are translated by the assembler into machine language.
- ▶ The machine code is then executed by the CPU
- ▶ An instruction contains four basic parts:
 - Label (optional)
 - Instruction mnemonic (required)
 - Operand(s) (required)
 - Comment (optional)
- ▶ This is the basic syntax:
 - *[label:] mnemonic [operands] [;comment]*

1. *Instruction Label*

- ▶ *A label is an identifier that acts as a place marker for instructions*
- ▶ A label in the code area of a program must end with a colon (:) character.
- ▶ Code labels are used as targets of jumping and looping instructions.

- ▶ Example

target:

mov ax,bx

...

N

jmp target

2. *Instruction Mnemonic*

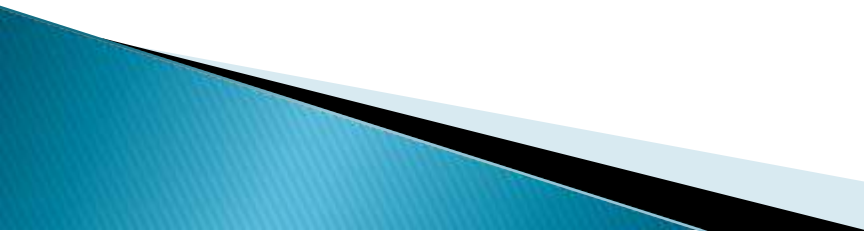
- ▶ An *instruction mnemonic* is a short word that identifies an instruction.
- ▶ It identifies the type of operation e.g. such as *mov, add, and sub etc.*
- ▶ Example
 - `mov ax,bx`

3. Operands

- ▶ Assembly language operands can be a register, memory operand, and constant expression
- ▶ The following table contains sample operands:

Example	Operand Type
96	Constant (<i>immediate value</i>)
2 + 4	Constant expression
eax	Register
count	Memory

4. Comments

- ▶ Comments are an important way for the writer of a program to communicate information about the program's design
 - ▶ Comments are optional
 - ▶ Comments can be specified in two ways:
 - Single-line comments, beginning with a semicolon character (;).
 - Block comments, beginning with the COMMENT directive and a user-specified symbol.
- 

Comments Example

- ▶ Single line

- `inc eax ; add 1 to EAX`

- ▶ Block comments

COMMENT !

This line is a comment.

This line is also a comment.

!

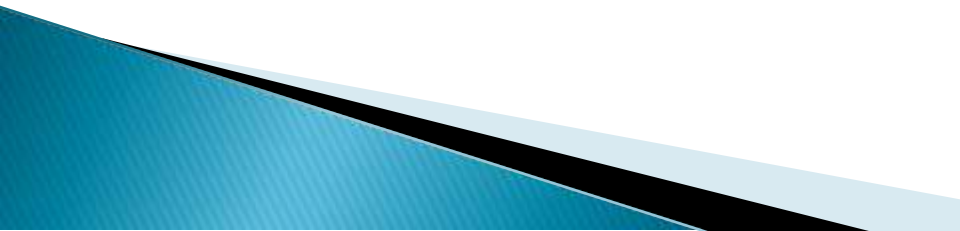


Assembly Instruction set

- ▶ Before start learning Instructions keep in mind that:
 - Operand types can be:
 1. **REG:** AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
 2. **SREG:** DS, ES, SS, and only as second operand: CS.
 3. **immediate:** 5, -24, 3Fh, 10001101b, etc...
 4. **Memory :** [0103]

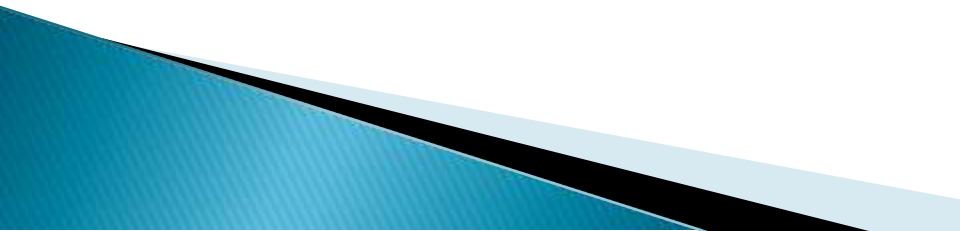


Assembly Instruction set writing Rules

1. When two operands are required for an instruction they are separated by comma.
For example:
 - REG, memory
 - REG, immediate
 - memory, REG
 2. When there are two operands, both operands must have the same size. For example:
 - Mov AL, DL
 - Mov DX, AX
- 



ADD Instruction

- ▶ ADD is used for addition of operands
 - ▶ Operands used:
 - REG, memory
 - memory, REG
 - REG, REG
 - memory, immediate
 - REG, immediate
 - ▶ Algorithm:
 - $\text{operand1} = \text{operand1} + \text{operand2}$
- 



ADD example

- Org 100h
- MOV AL, 5 ; AL = 5
- ADD AL, -3 ; AL = 2
- RET

RET: means Return






MOV Instruction

- It Copies operand2 to operand1.
- E.g
 - `mov operand1, operand2`

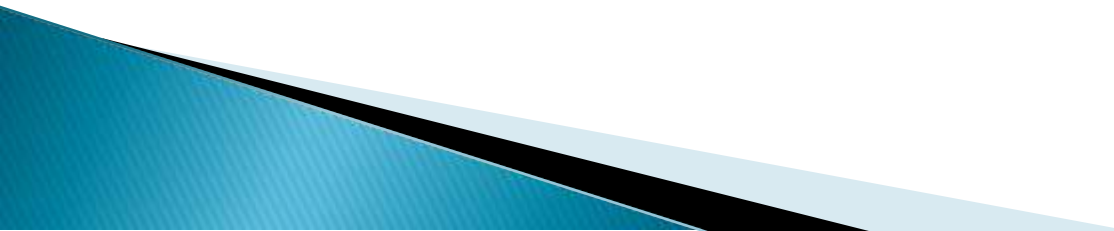
- Operands can be;

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

- REG, immediate
 - SREG, memory
 - memory, SREG
 - REG, SREG
 - SREG, REG
- 



Limitations of MOV

- ❑ The MOV instruction cannot:
 - Set the values of the CS and IP registers.
 - Copy value of one segment register to another segment register (should copy to general register first).
 - copy immediate value to segment register
- 

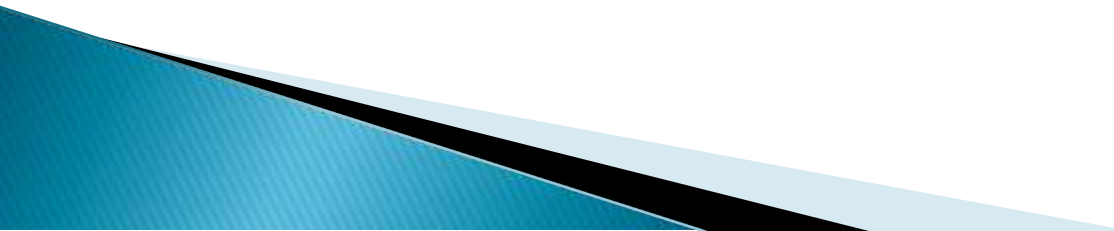


MOV example

- ▶ **ORG 100h**
 - MOV AX, 09
 - MOV DS, AX ; copy value of AX to DS.
 - MOV CL, 'A' ; CL = 41h (ASCII code).
 - MOV CH, 01011111b
- ▶ **RET**

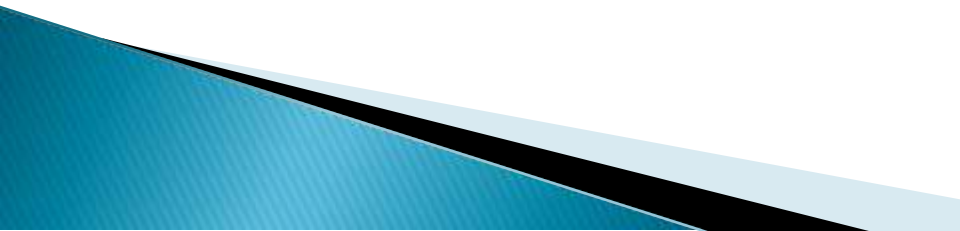


SUB Instruction

- SUB is for Subtraction
 - Algorithm:
 - $\text{operand1} = \text{operand1} - \text{operand2}$
 - Operands can be:
 - REG, memory
 - memory, REG
 - REG, REG
 - memory, immediate
 - REG, immediate
- 

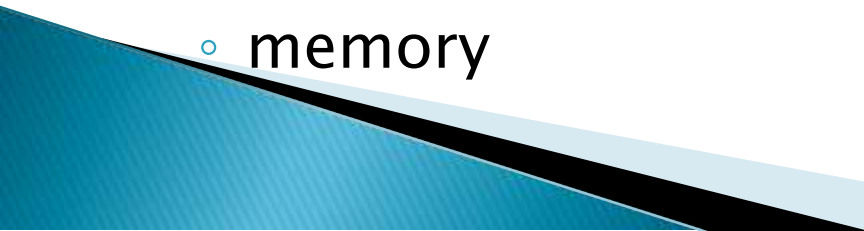


SUB example

- ▶ `ORG 100h`
 - ▶ `MOV AL, 5`
 - ▶ `SUB AL, 1 ; AL = 4`
 - ▶ `RET`
- 



JL Instruction

- ▶ MUL is for Multiplication
 - ▶ Algorithm:
 - when operand is a **byte**:
 - $AX = AL * \text{operand}$.
 - when operand is a **word**:
 - $(DX AX) = AX * \text{operand}$
 - ▶ Operands can be:
 - REG
 - memory
- 



JL example

- ▶ Example:
- ▶ Org 100h
 - MOV AL, 200 ; AL = 0C8h
 - MOV BL, 4
 - MUL BL ; AX = 0320h (800)
- ▶ RET



✓ Instruction

- ▶ DIV is for division
- ▶ Algorithm:
 - when operand is a **byte**:
 - $AL = AX / \text{operand}$
 - $AH = \text{remainder (modulus)}$
 - when operand is a **word**:
 - $AX = (DX AX) / \text{operand}$
 - $DX = \text{remainder (modulus)}$
- ▶ Operands can be:
 - REG
 - memory

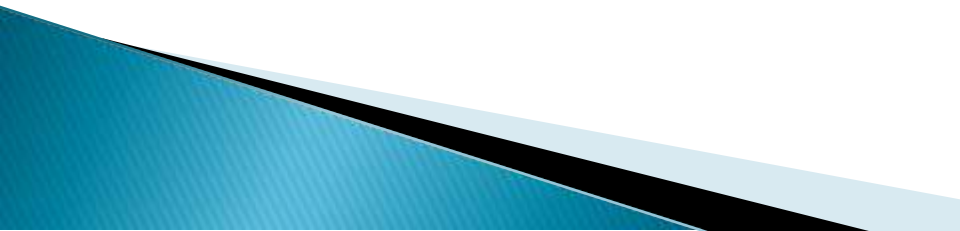


DIV example

- ▶ Example:
 - ORG 100h
 - *MOV AX, 203 ; AX = 00CBh*
 - *MOV BL, 4*
 - *DIV BL ; AL = 50 (32h), AH = 3*
 - RET

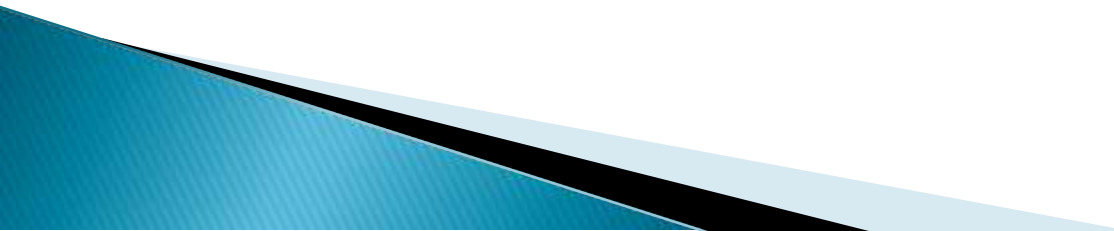


Boolean Operations

- ▶ The set of operators includes the following:
 1. NOT: notated as \neg or \sim
 2. AND: notated as \wedge
 3. OR: notated as \vee
 4. XOR: (Exclusive OR) If both operands are same the result is 0.
- 



NOT Operator

- ▶ NOT Invert each bit of the operand.
 - ▶ Algorithm:
 - if bit is 1 turn it to 0.
 - if bit is 0 turn it to 1.
 - ▶ Operands can be:
 - REG
 - memory
- 

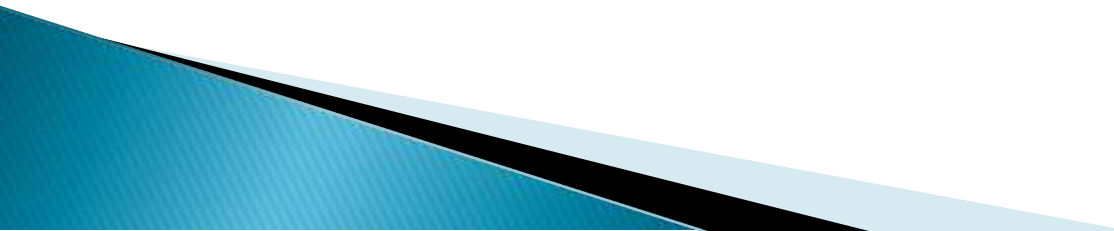


NOT example

- ▶ ORG 100h
 - MOV AL, 00011011b
 - NOT AL ; AL = 11100100b
 - *Mov dl,al*
 - *Mov ah,2*
 - *Int 21h*
- ▶ RET



AND Operator

- ▶ Logical AND between all bits of two operands. Result is stored in operand1.
 - ▶ These rules apply:
 - $1 \text{ AND } 1 = 1$
 - $1 \text{ AND } 0 = 0$
 - $0 \text{ AND } 1 = 0$
 - $0 \text{ AND } 0 = 0$
 - ▶ Operands can be:
 - memory, REG, immediate
- 



ample

- ▶ Org 100h
 - *MOV AL, 'a' ; AL = 01100001b*
 - *AND AL, 11011111b ; AL = 01000001b ('A')*
 - *Mov dl,al*
 - *Mov ah,2*
 - *Int 21h*

- ▶ RET



Operator

- ▶ Logical OR between all bits of two operands.
Result is
- ▶ stored in first operand.
- ▶ These rules apply:
 - $1 \text{ OR } 1 = 1$
 - $1 \text{ OR } 0 = 1$
 - $0 \text{ OR } 1 = 1$
 - $0 \text{ OR } 0 = 0$
- ▶ Operands can be:
 - memory, REG, immediate



ample:

- ▶ Org 100h
 - *MOV AL, 'A' ; AL = 01000001b*
 - *OR AL, 00100000b ; AL = 01100001b ('a')*
 - *Mov dl,al*
 - *Mov ah,2*
 - *Int 21h*
- ▶ RET



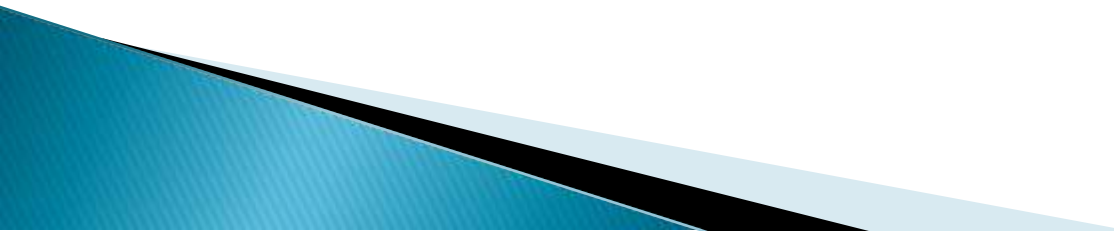
XOR Operator

- ▶ Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.
- ▶ These rules apply:
 - $1 \text{ XOR } 1 = 0$
 - $1 \text{ XOR } 0 = 1$
 - $0 \text{ XOR } 1 = 1$
 - $0 \text{ XOR } 0 = 0$
- ▶ Operands can be:
 - memory, REG, immediate



Example:

- ▶ Org 100h
 - *MOV AL, 00000111b*
 - *XOR AL, 00000010b ; AL = 00000101b*
 - *Mov dl,al*
 - *Mov ah,2*
 - *Int 21h*

 - ▶ RET
- 

▶ Thanks