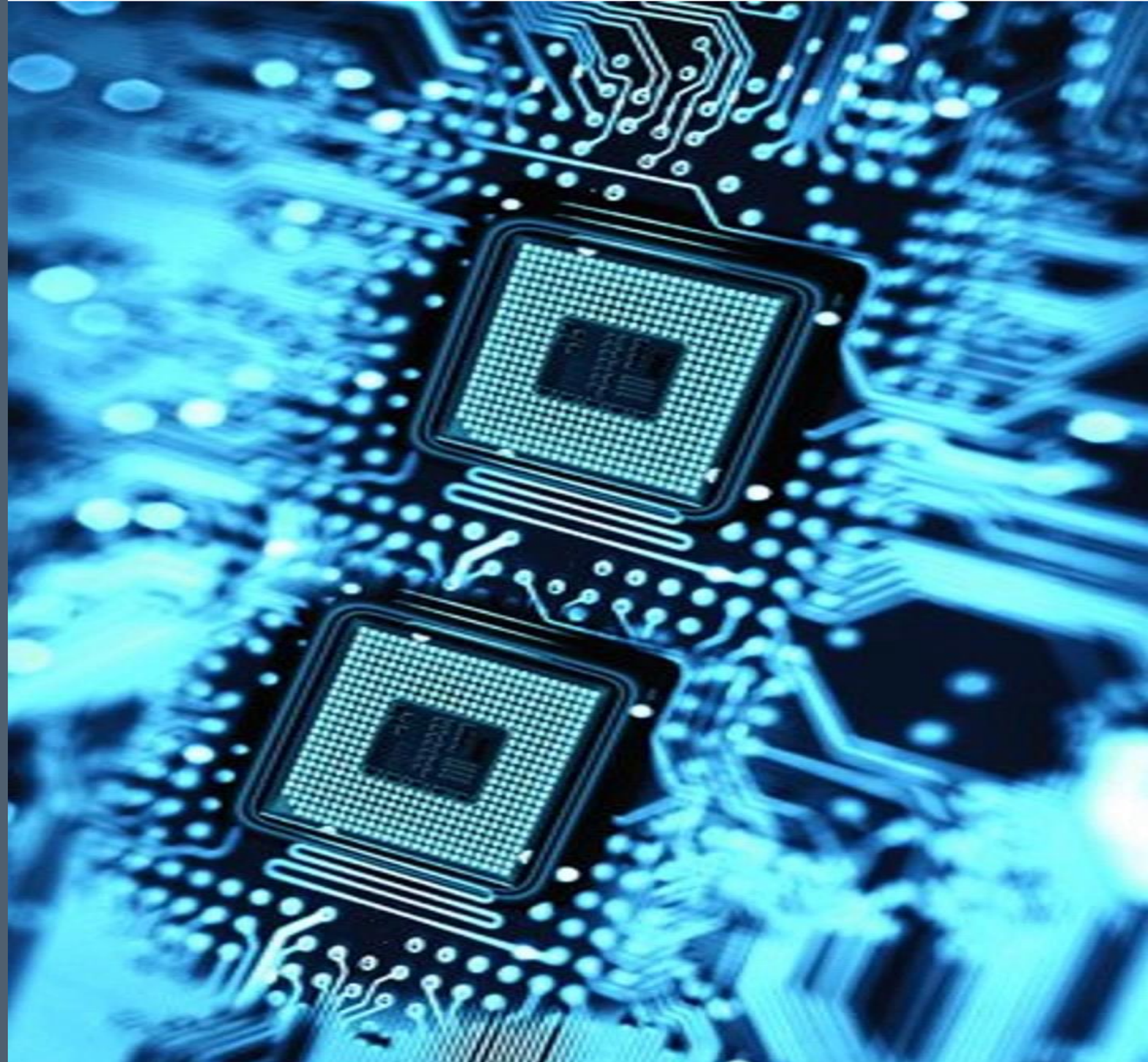
Computer organization & architecture

Course by: Dr. Ahmed Sadek

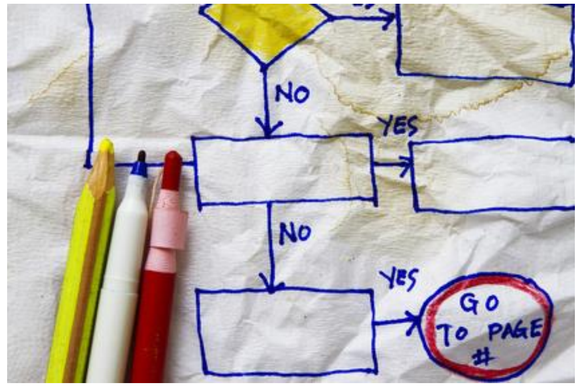
Lab By: Mahmoud Badry

Conditional Processing

Chapter 6



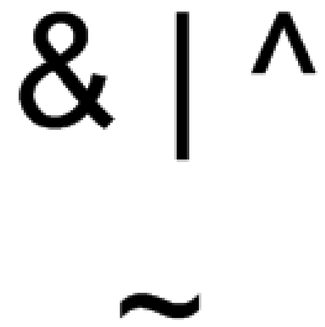
Introduction



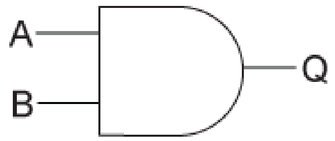
- We've **managed** to **create** counting **loops**, **procedures**, **data definitions**, and **array processing**, while carefully **avoiding decision** making.
- **IF statements** and **conditional** processing are a little more **complicated** in **assembly** language **than** in **high-level** languages that's why we've made it **last thing** to discuss.
- After this chapter you will be able to **answer** this **questions**:
  - How can I **use** the **Boolean** operations (**AND** – **OR** - **NOT**)?
  - How do I **write** an **IF** statement in assembly language?
  - How do I tell the computer I'm **comparing signed** numbers **versus unsigned** numbers?
  - Isn't there any way to **create** the kinds of **IF** - **ELSE** - **ENDIF** structures in assembly language that I'm used to using in C++ and Java?
  - Is **GOTO** really **considered harmful**?

Boolean and Comparison Instructions

Section 2



AND Instruction



- The **AND** instruction performs a **boolean** (**bitwise**) AND operation between each **pair** of **matching bits** in **two operands** and **places** the **result** in the **destination** operand.

  `AND destination, source`

- `AND reg,reg`

  `AND reg,mem`

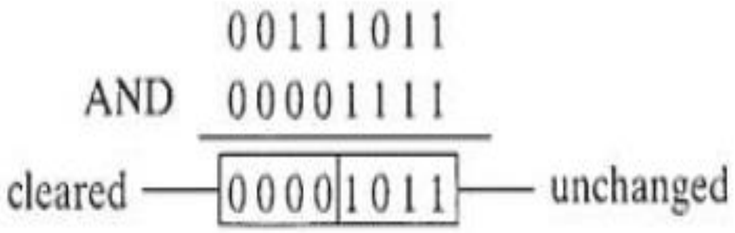
  `AND reg,imm`

  `AND mem, reg`

  `AND mem,imm`

- Operands **must** be the **same size**.

AND Instruction



- Example (**Extracting** first **four** bits):

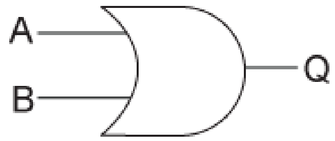
```
MOV al, 00111011b
AND al, 00001111b
```

- Example (Converting **Characters** to **Upper Case**)

```
MOV al, 'a'
AND al, 11011111b  ;Clearing bit number 6
PUTC al            ;A
```

- **Flags***:* The AND instruction always **clears** the **Overflow** and **Carry** flags. It **modifies** the **Sign**, **Zero** , and **Parity** flags according to the value of the **destination** operand.

OR Instruction



- The **OR** instruction performs a **boolean** (**bitwise**) OR operation between each **pair** of **matching bits** in **two operands** and **places** the **result** in the **destination** operand.

  `OR destination, source`

- `OR reg,reg`

  `OR reg,mem`

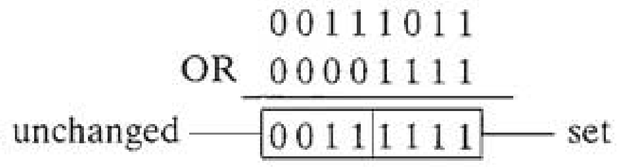
  `OR reg,imm`

  `OR mem, reg`

  `OR mem,imm`

- Operands **must** be the **same size**.

OR Instruction



- The **OR** instruction performs a **boolean** (**bitwise**) OR operation between each **pair** of **matching bits** in **two operands** and **places** the **result** in the **destination** operand.
- Example (**Setting** first **four** bits):

```
MOV al, 00111011b
OR al,  00001111b
```

- Example (Converting **Characters** to **lower Case**)

```
MOV al, 'A'
OR al, 00100000b      ;setting bit number 6
PUTC al                ;a
```

- **Flags***:* The OR instruction always **clears** the **Overflow** and **Carry** flags. It **modifies** the **Sign**, **Zero** , and **Parity** flags according to the value of the **destination** operand.

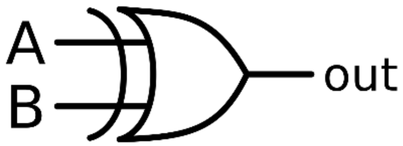
OR Instruction

- To **check** a **value** is **zero**, **less** than **zero** or **greater** than **zero**:

```
OR al , al
```

Zero Flag	Sign Flag	Value in AL is . . .
clear	clear	greater than zero
set	clear	equal to zero
clear	set	less than zero

XOR Instruction



- The **XOR** instruction performs a **boolean** execlusive-OR operation between each **pair** of **matching bits** in **two operands** and **places** the **result** in the **destination** operand.

  `XOR destination, source`

- `XOR reg,reg`

  `XOR reg,mem`

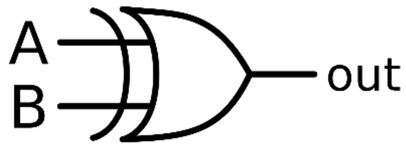
  `XOR reg,imm`

  `XOR mem, reg`

  `XOR mem,imm`

- Operands **must** be the **same size**.
- If **bits** are **different**, the **output** is **one** else its **zero**.

XOR Instruction

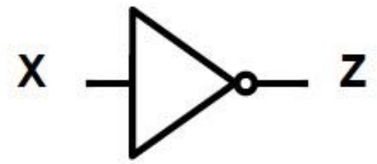


- When **XOR** is applied **twice** to the **same value**, the same value is **returned** in destination.

```
MOV al, 00001111b
XOR al, 11111111b     ;al = 11110000
XOR al, 11111111b     ;al = 00001111
```

- **Flags***:* The XOR instruction always **clears** the **Overflow** and **Carry** flags. It **modifies** the **Sign**, **Zero** , and **Parity** flags according to the value of the **destination** operand.

NOT Instruction



- The **NOT** instruction **toggles all bits** in an operand. The **result** is **called** the **one's complement**.

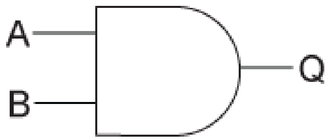
```
NOT reg
NOT mem
```

- Example:

```
MOV al,11110000b
NOT al                  ; AL = 00001111b
```

- **Flags**: **No flags** are **affected** by the NOT instruction.

TEST Instruction



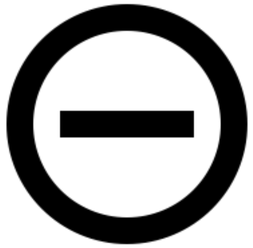

```
0 0 1 0 0 1 0 1   <- input value
0 0 0 0 1 0 0 1   <- test value
0 0 0 0 0 0 0 1   <- result: ZF = 0

0 0 1 0 0 1 0 0   <- input value
0 0 0 0 1 0 0 1   <- test value
0 0 0 0 0 0 0 0   <- result: ZF = 1
```

- The **TEST** instruction performs an **implied AND** operation **between each pair** of matching bits in two operands and **sets** the **flags** accordingly. The only **difference between TEST** and **AND** is that TEST does **not modify** the **destination** operand.

```
Test Val1,Val2
```

- **Flags**: The TEST instruction always **clears** the **Overflow** and **Carry** flags. It **modifies** the **Sign**, **Zero** , and **Parity** flags according to the value of the **destination** operand.

CMP Instruction



CMP Results	ZF	CF
destination < source	0	1
destination > source	0	0
destination = source	1	0

- The **CMP** (**compare**) instruction performs an **implied subtraction** of a **source** operand **from** a **destination** operand. **Neither operand** is **modified**.

  `CMP destination, source`

- `CMP reg,reg`

  `CMP reg,mem`

  `CMP reg,imm`

  `CMP mem, reg`

  `CMP mem,imm`

- **Flags**: The **CMP** instruction **changes** the **Overflow**. **Sign**. **Zero** . **Carry**, **Auxiliary** Carry, and **Parity** flags **according** to the value the **destination** operand would have had if the **SUB** instruction were used.

Setting and Clearing Individual CPU Flags

- To set or clear zero flag:

```
AND al, 0     ; set Zero flag, al = 0 always
OR  al, 1     ; clear Zero flag, al > 1 always
```

- To set or clear sign flag:

```
OR al, 80h    ; set Sign flag, al MSB always 1
AND al, 7Fh   ; clear Sign flag, al MSB always 0
```

- To set or clear carry flag:

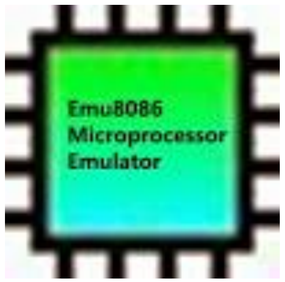
```
stc           ;set Carry flag
clc           ;clear Carry flag
```

- To set or clear overflow flag:

```
mov al ,7Fh   ; al = +127
inc al        ; set overflow  flag, al = -128
or ax,0       ; clear overflow flag, al = 0
```

Conditional Jumps

Section 3



Conditional Structures


```
    cmp   al,0
    jz    L1                ; jump if ZF = 1
    .
    .
L1:
```

- There are **no high-level logic structures** in the **IA-32** instruction set, but you **can implement** any **logic structure**, no matter how complex, using a **combination** of **comparisons** and **jumps**.
- **First**, an **operation** such as **CMP**, **AND**, or **SUB** modifies the CPU flags. **Second**, a **conditional jump** instruction **tests** the **flags** and causes a **branch** to a **new address** . Let 's look at a couple of examples.
- **Example** are on the left.

Jcond Instruction

- A **conditional jump** instruction **branches** to a **destination label** when a **flag condition** is **true**.

```
Jcond destinationLabel
```

- Don’t be **surprised** of conditional jumps are a **lot**, they are a lot to **serve** all **programing purposes**.
- **Types** of **Conditional Jump** Instructions:
  - Based on specific **flag** values.
  - Based on **equality** between **operands**, or the value of **CX**.
  - Based on **comparisons** of **unsigned operands**.
  - Based on **comparisons** of **signed operands**.

Based on specific flag values

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0

Equality Comparisons

- **Call** this instruction **before** using the **conditional jump**:

  CMP *leftOp,rightOp*

- Example:

```
MOV ax, 5
CMP ax, 5
JE L1                ;jump if equal
```

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

Unsigned Comparisons

- **Call** this instruction **before** using the **conditional jump**:

  CMP *leftOp,rightOp*

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

Signed Comparisons

- **Call** this instruction **before** using the **conditional jump**:

  CMP *leftOp,rightOp*

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

Conditional Jump Applications

- Testing Status Bits (Test fifth bit):

```
MOV al , status
TEST a1 ,00100000b  ; test bit 5
JNZ EquipOffline
```

- Testing Status Bits (Test zero, first and fourth bits):

```
MOV al , status
TEST a1 ,00110011b  ; test bits 0,1,4
JNZ InputDataByte
```

- Larger of Two Integers to DX:

```
MOV DX, AX     ; assume AX is larger
CMP AX, BX     ; if AX is >= BX then
JAE L1           ; jump to L1
MOV DX, BX
L1:
```

Conditional Jump Applications

- Smallest of Three Integers to AX:

```
.data
V1 DW ?
V2 DW ?
V3 DW ?
. Code
mov AX, V1          ; assume V1 is smallest
cmp AX, V2          ; if AX <= V2 then
jbe Ll              ; jump to L1
mov aX, V2          ; else move V2 to AX
Ll: cmp AX,V3       ; if AX <= V3 then
jbe L2              ; jump to L2
mov aX,V3           ; else move V3 t o AX
L2 :
```

Study Section 5

For more examples on conditional jumps

THANKS