Assembly Language Lecture Series:

x86-64 Logic Instructions

Sensei RL Uy, College of Computer Studies,
De La Salle University, Manila, Philippines

Copyright Notice

x86-64 Logic Instructions

1. **AND**
   Bitwise AND Instruction
2. **OR**
   Bitwise OR Instruction
3. **XOR**
   Bitwise XOR Instruction
4. **NOT**
   Bitwise NOT
5. **BT**
   Bit Test

x86-64 Logic Instructions: AND

AND (bitwise AND instruction)

Syntax: AND dst, src

dst ← dst • src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of AND

$$X \bullet 0 = 0$$
$$X \bullet 1 = X$$
$$X \bullet X = X$$
$$\bar{X} \bullet X = 0$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Note:

1. Immediate value up to **32-bit** only
2. When an **immediate value** is used as an **operand**, it is **sign-extended** to the **length** of the **destination** operand format
3. **Negative** number in **hex** has to be sign-extended to **64-bit**

x86-64 Logic Instructions: AND

AND (bitwise AND instruction)

Syntax: AND dst, src

dst ← dst • src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of AND

$$X \bullet 0 = 0$$
$$X \bullet 1 = X$$
$$X \bullet X = X$$
$$\bar{X} \bullet X = 0$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x25
MOV BL, 0x00
AND AL, BL
```

1. **What will AL contain after execution?**
2. **What will CF, OF, SF, ZF, PF contain after execution?**

x86-64 Logic Instructions: AND

AND (bitwise AND instruction)

Syntax: AND dst, src

dst ← dst • src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of AND

$$X \bullet 0 = 0$$
$$X \bullet 1 = X$$
$$X \bullet X = X$$
$$\bar{X} \bullet X = 0$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x25
MOV BL, 0x00
AND AL, BL
```

1. What will AL contain after execution?
2. What will CF, OF, SF, ZF, PF contain after execution?

AL = 00	OF = 0
CF = 0	PF = 1
SF = 0	ZF = 1

x86-64 Logic Instructions: AND

AND (bitwise AND instruction)

Syntax: AND dst, src

dst ← dst • src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of AND

$$X \bullet 0 = 0$$
$$X \bullet 1 = X$$
$$X \bullet X = X$$
$$\bar{X} \bullet X = 0$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x42
MOV BL, 0xFF
AND AL, BL
```

1. **What will AL contain after execution?**
2. **What will CF, OF, SF, ZF, PF contain after execution?**

x86-64 Logic Instructions: AND

AND (bitwise AND instruction)

Syntax: AND dst, src

dst ← dst • src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of AND

$$X \bullet 0 = 0$$
$$X \bullet 1 = X$$
$$X \bullet X = X$$
$$\bar{X} \bullet X = 0$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x42
MOV BL, 0xFF
AND AL, BL
```

1. What will AL contain after execution?
2. What will CF, OF, SF, ZF, PF contain after execution?

AL = 42	OF = 0
CF = 0	PF = 1
SF = 0	ZF = 0

x86-64 Logic Instructions: AND

AND (bitwise AND instruction)

Syntax: AND dst, src

dst ← dst • src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of AND

$$X \bullet 0 = 0$$
$$X \bullet 1 = X$$
$$X \bullet X = X$$
$$\bar{X} \bullet X = 0$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example: AND each bit value of RAX with max value

1. **AND RAX, 0x0000_0000_7FFF_FFFF ; pos**
2. **AND RAX, 0xffff_ffff_8000_0000 ; neg**

x86-64 Logic Instructions: OR

OR (bitwise OR instruction)

Syntax: OR dst, src

dst ← dst+src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of OR:

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$\bar{X} + X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Note:

1. Immediate value up to **32-bit** only
2. When an **immediate value** is used as an **operand**, it is **sign-extended** to the **length** of the **destination** operand format
3. **Negative** number in **hex** has to be sign-extended to **64-bit**

x86-64 Logic Instructions: OR

OR (bitwise OR instruction)

Syntax: OR dst, src

dst ← dst+src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of OR:

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$\bar{X} + X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x25
MOV BL, 0x00
OR AL, BL
```

1. **What will AL contain after execution?**
2. **What will CF, OF, SF, ZF, PF contain after execution?**

x86-64 Logic Instructions: OR

OR (bitwise OR instruction)

Syntax: OR dst, src

dst ← dst+src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of OR:

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$\bar{X} + X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x25
MOV BL, 0x00
OR AL, BL
```

1. What will AL contain after execution?
2. What will CF, OF, SF, ZF, PF contain after execution?

AL = 25	OF = 0
CF = 0	PF = 0
SF = 0	ZF = 0

x86-64 Logic Instructions: OR

OR (bitwise OR instruction)

Syntax: OR dst, src

dst ← dst+src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of OR:

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$\bar{X} + X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x42
MOV BL, 0xFF
OR AL, BL
```

1. **What will AL contain after execution?**
2. **What will CF, OF, SF, ZF, PF contain after execution?**

x86-64 Logic Instructions: OR

OR (bitwise OR instruction)

Syntax: OR dst, src

dst ← dst+src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of OR:

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$\bar{X} + X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV AL, 0x42
MOV BL, 0xFF
OR AL, BL
```

1. What will AL contain after execution?
2. What will CF, OF, SF, ZF, PF contain after execution?

AL = FF	OF = 0
CF = 0	PF = 1
SF = 1	ZF = 0

x86-64 Logic Instructions: OR

OR (bitwise OR instruction)

Syntax: OR dst, src

dst ← dst+src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of OR:

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$\bar{X} + X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example: OR each bit value of RAX with max value

1. **OR RAX, 0x0000_0000_7FFF_FFFF ; pos**
2. **OR RAX, 0xffff_ffff_8000_0000 ; neg**

x86-64 Logic Instructions: XOR

XOR (bitwise XOR instruction)

Syntax: XOR dst, src

dst ← dst⊕ src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of XOR:

$$X \oplus 0 = X$$
$$X \oplus 1 = X'$$
$$X \oplus X = 0$$
$$\bar{X} \oplus X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Note:

1. Immediate value up to **32-bit** only
2. When an **immediate value** is used as an **operand**, it is **sign-extended** to the **length** of the **destination** operand format
3. **Negative** number in **hex** has to be sign-extended to **64-bit**

x86-64 Logic Instructions: XOR

XOR (bitwise XOR instruction)

Syntax: XOR dst, src

dst ← dst⊕ src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of XOR:

$$X \oplus 0 = X$$

$$X \oplus 1 = X'$$

$$X \oplus X = 0$$

$$\bar{X} \oplus X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV RAX, 0x123456789ABCDEF1
XOR RAX, RAX
```

1. **What will AL contain after execution?**
2. **What will CF, OF, SF, ZF, PF contain after execution?**

x86-64 Logic Instructions: XOR

XOR (bitwise XOR instruction)

Syntax: XOR dst, src

dst ← dst⊕ src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of XOR:

$$X \oplus 0 = X$$
$$X \oplus 1 = X'$$
$$X \oplus X = 0$$
$$\bar{X} \oplus X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV RAX, 0x123456789ABCDEF1
XOR RAX, RAX
```

1. What will AL contain after execution?
2. What will CF, OF, SF, ZF, PF contain after execution?

RAX = 0000000000000000	OF = 0
CF = 0	PF = 1
SF = 0	ZF = 1

x86-64 Logic Instructions: XOR

XOR (bitwise XOR instruction)

Syntax: XOR dst, src

dst ← dst⊕ src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of XOR:

$$X \oplus 0 = X$$
$$X \oplus 1 = X'$$
$$X \oplus X = 0$$
$$\bar{X} \oplus X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV RAX, 0x123456789ABCDEF1
XOR RAX, 0x0000000000000000
```

1. **What will AL contain after execution?**
2. **What will CF, OF, SF, ZF, PF contain after execution?**

x86-64 Logic Instructions: XOR

XOR (bitwise XOR instruction)

Syntax: XOR dst, src

dst ← dst⊕ src

dst: reg/mem

src: reg/mem/imm8_16_32

Properties of XOR:

$$X \oplus 0 = X$$
$$X \oplus 1 = X'$$
$$X \oplus X = 0$$
$$\bar{X} \oplus X = 1$$

Flags affected:

*SF, ZF, PF

*CF=OF=0

*AF: undefined

Example:

```
section .text
MOV RAX, 0x123456789ABCDEF1
XOR RAX, 0x0000000000000000
```

1. What will AL contain after execution?
2. What will CF, OF, SF, ZF, PF contain after execution?

RAX = 1234_5678_9ABC_DEF1 OF = 0
CF = 0 PF = 0
SF = 0 ZF = 0

x86-64 Logic Instructions: NOT

NOT (bitwise NOT instruction)

Syntax: NOT dst

dst ← ~dst (i.e., 1's complement)

dst: reg/mem

Flags affected: None

x86-64 Logic Instructions: NOT

NOT (bitwise NOT instruction)

Syntax: NOT dst
dst ← ~ dst (i.e., 1's complement)
dst: reg/mem

Flags affected: None

Example:

```
section .text
MOV AL, 0x25
NOT AL
```

1. **What will AL contain after execution?**

x86-64 Logic Instructions: NOT

NOT (bitwise NOT instruction)

Syntax: NOT dst
dst ← ~ dst (i.e., 1's complement)
dst: reg/mem

Flags affected: None

Example:

```
section .text
MOV AL, 0x25
NOT AL
```

1. What will AL contain after execution?

AL = DA

x86-64 Logic Instructions: NOT

NOT (bitwise NOT instruction)

Syntax: NOT dst
dst ← ~ dst (i.e., 1's complement)
dst: reg/mem

Flags affected: None

Example:

```
section .text
MOV BL, 0xFF
NOT BL
```

1. **What will BL contain after execution?**

x86-64 Logic Instructions: NOT

NOT (bitwise NOT instruction)

Syntax: NOT dst
dst ← ~ dst (i.e., 1's complement)
dst: reg/mem

Flags affected: None

Example:

```
section .text
MOV BL, 0xFF
NOT BL
```

1. What will BL contain after execution?

BL = 00

x86-64 Logic Instructions: BT

BT (Bit Test)

Syntax: BT BitBase, BitOffset

CF ← BitBase, BitOffset

BitBase – r/m16_32_64

BitOffset – r16_32_64/imm8

Flags affected:

- **CF** - contains the value of the bit in the bit string whose position is designated by the bit offset.
- **ZF** – no change
- **OF, SF, ZF, PF** - undefined

x86-64 Logic Instructions: BT

BT (Bit Test)

Syntax: BT BitBase, BitOffset
CF ← BitBase, BitOffset
BitBase – r/m16_32_64
BitOffset – r16_32_64/imm8

Flags affected:

- **CF** - contains the value of the bit in the bit string whose position is designated by the bit offset.
- **ZF** – no change
- **OF, SF, ZF, PF** - undefined

Example:

```
section .text
MOV AX,
0x8234_5678_9ABC_DEF0
BT RAX, 0
```

1. **What will CF contain after execution?**

x86-64 Logic Instructions: BT

BT (Bit Test)

Syntax: BT BitBase, BitOffset
CF ← BitBase, BitOffset
BitBase – r/m16_32_64
BitOffset – r16_32_64/imm8

Flags affected:

- **CF** - contains the value of the bit in the bit string whose position is designated by the bit offset.
- **ZF** – no change
- **OF, SF, ZF, PF** - undefined

Example:

```
section .text
MOV AX,
0x8234_5678_9ABC_DEF0
BT RAX, 0
```

1. What will CF contain after execution?

   CF = 0

x86-64 Logic Instructions: BT

BT (Bit Test)

Syntax: BT BitBase, BitOffset
CF ← BitBase, BitOffset
BitBase – r/m16_32_64
BitOffset – r16_32_64/imm8

Flags affected:

- **CF** - contains the value of the bit in the bit string whose position is designated by the bit offset.
- **ZF** – no change
- **OF, SF, ZF, PF** - undefined

Example:

```
section .text
MOV RAX,
0x8234_5678_9ABC_DEF0
BT RAX,63
```

1. **What will CF contain after execution?**

x86-64 Logic Instructions: BT

BT (Bit Test)

Syntax: BT BitBase, BitOffset
CF ← BitBase, BitOffset
BitBase – r/m16_32_64
BitOffset – r16_32_64/imm8

Flags affected:

- **CF** - contains the value of the bit in the bit string whose position is designated by the bit offset.
- **ZF** – no change
- **OF, SF, ZF, PF** - undefined

Example:

```
section .text
MOV RAX,
0x8234_5678_9ABC_DEF0
BT RAX,63
```

1. What will CF contain after execution?

   CF = 1