Assembly Language Lecture Series:

x86-64 Integer Register

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

Copyright Notice

What are Registers?

Registers are storage inside the processor. No need to fetch the data. Registers are referred to by their register name.

x86-64 Registers

Register type	
Byte registers (8-bit)	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8B-R15B
Word registers (16-bit)	AX, BX, CX, DX, DI, SI, BP, SP, R8W-R15W
Doubleword registers (32-bit)	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D-R15D
Quadword registers (64-bit)	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15

General Purpose x86-64 Registers

- RAX, RBX, RCX, RDX
- RSI, RDI, RBP, RSP, RIP
- R8, R9, R10, R11, R12, R13, R14, R15

General Purpose x86-64 Registers

- **(R)AX – Accumulator for operands and results data**
- (R)BX – pointer to data in the data segment
- (R)CX – Counter for string and loop operations
- (R)DX – I/O pointer

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td>AH</td><td>AL</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">AX</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">EAX</td></tr>
<tr><td>64-bit</td><td colspan="8">RAX</td></tr>
</table>

General Purpose x86-64 Registers

- (R)AX – Accumulator for operands and results data
- **(R)BX – pointer to data in the data segment**
- (R)CX – Counter for string and loop operations
- (R)DX – I/O pointer

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td>BH</td><td>BL</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">BX</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">EBX</td></tr>
<tr><td>64-bit</td><td colspan="8">RBX</td></tr>
</table>

General Purpose x86-64 Registers

- (R)AX – Accumulator for operands and results data
- (R)BX – pointer to data in the data segment
- **(R)CX – Counter for string and loop operations**
- (R)DX – I/O pointer

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td>CH</td><td>CL</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">CX</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">ECX</td></tr>
<tr><td>64-bit</td><td colspan="8">RCX</td></tr>
</table>

General Purpose x86-64 Registers

- (R)AX – Accumulator for operands and results data
- (R)BX – pointer to data in the data segment
- (R)CX – Counter for string and loop operations
- **(R)DX – I/O pointer**

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td>DH</td><td>DL</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">DX</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">EDX</td></tr>
<tr><td>64-bit</td><td colspan="8">RDX</td></tr>
</table>

General Purpose x86-64 Registers

64-bit General purpose registers **RAX, RBX, RCX, and RDX**

can also be viewed as:

- **32-bit** (EAX, EBX, ECX, and EDX)
- **16-bit** (AX, BX, CX, DX)
- **8-bit** (AH, AL, BH, BL, CH, CL, DH, DL)

General Purpose x86-64 Registers

As a rule, 8-bit and 16-bit variants of the register will overwrite only the corresponding 8-bit and 16-bit of its 64-bit register.

Example:

```
MOV RAX, 0x1234_5678_9ABC_DEF1
MOV AL, 0x22
After execution:
RAX: 1234_5678_9ABC_DE22
```

Example:

```
MOV RAX, 0x1234_5678_9ABC_DEF1
MOV AX, 0x1357
After execution:
RAX: 1234_5678_9ABC_1357
```

General Purpose x86-64 Registers

On the other hand, the 32-bit variant **overwrites** the entire 64-bit register.

Example:

```
MOV RAX, 0x1234_5678_9ABC_DEF1
MOV EAX, 0x2222_2222
After execution:
RAX: 0000_0000_2222_2222
```

General Purpose x86-64 Registers

More examples

```
MOV RAX, 0x1234_5678_ABCD_EF01
MOV CX, AX
MOV EDX, 0xCAFE_7123
```

RCX								
8-bit							EF	01
16-bit							EF01	
32-bit					0000EF01			
64-bit	00000000 0000EF01							

RAX								
8-bit							EF	
16-bit							EF01	
32-bit					ABCDEF01			
64-bit	12345678ABCDEF01							

RDX								
8-bit							71	23
16-bit							7123	
32-bit					CAFE7123			
64-bit	00000000 CAFE7123							

General Purpose x86-64 Registers

64-bit general purpose registers **R8,R9,R10,R11,R12,R13,R14 and R15** can also be viewed as:

- **32-bit** (R8D,R9D,R10D,R11D,R12D,R13D,R14D and R15D)
- **16-bit** (R8W,R9W,R10W,R11W,R12W,R13W,R14W and R15W)
- **8-bit** (R8B,R9B,R10B,R11B,R12B,R13B,R14B and R15B)

Other General Purpose x86-64 Registers

R8, R9, R10, R11, R12, R13, R14, R15

8-bit								R8B
16-bit							R8W	
32-bit					R8D			
64-bit	R8							

8-bit								R10B
16-bit							R10W	
32-bit					R10D			
64-bit	R10							

8-bit								R9B
16-bit							R9W	
32-bit					R9D			
64-bit	R9							

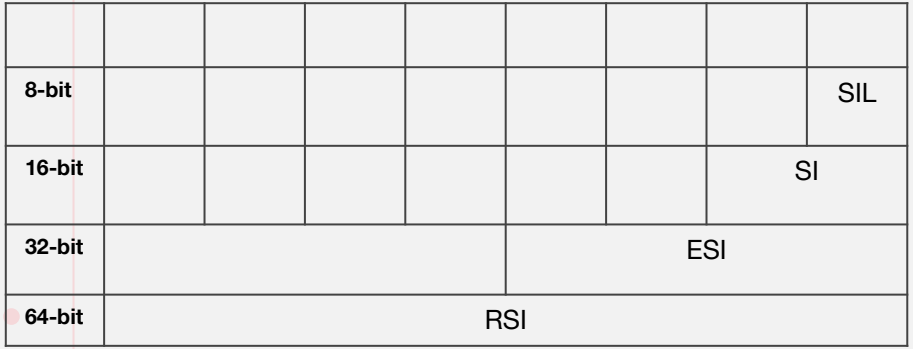
x86-64 Index Registers

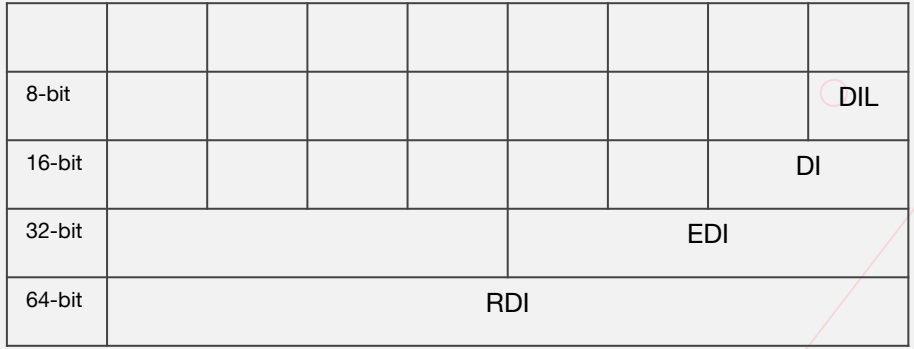
- **(R)SI** – Source Index: pointer for string operations; pointer to data in the data segment
- **(R)DI** – Destination Index: pointer for string operations; pointer to data in the extra segment

x86-64 Index Registers

64-bit index registers **RSI and RDI** can also be viewed as:

- **32-bit** (ESI, EDI)
- **16-bit** (SI, DI)
- **8-bit** (SIL, DIL)





x86-64 Pointer Registers

- **(R)SP** – Stack pointer: always points to the top of the stack. Updated only when using stack-related instructions, and it is not advisable to modify the value.
- **(R)BP** – Base Pointer: pointer to data in the stack segment
- **(R)IP** - Instruction Pointer: pointer to instruction in the code segment (i.e., contains the address of the next instruction to be executed). Updated automatically by the processor. Do not modify the value at any time.

x86-64 Pointer Registers

64-bit pointer registers RSP, RBP, RIP can also be viewed as:

- **32-bit** (ESP, EBP, EIP)
- **16-bit** (SP, BP, IP)
- **8-bit** (SPL, BPL).

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>BPL</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">BP</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">EBP</td></tr>
<tr><td>64-bit</td><td colspan="8">RBP</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>SPL</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">SP</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">ESP</td></tr>
<tr><td>64-bit</td><td colspan="8">RSP</td></tr>
</table>

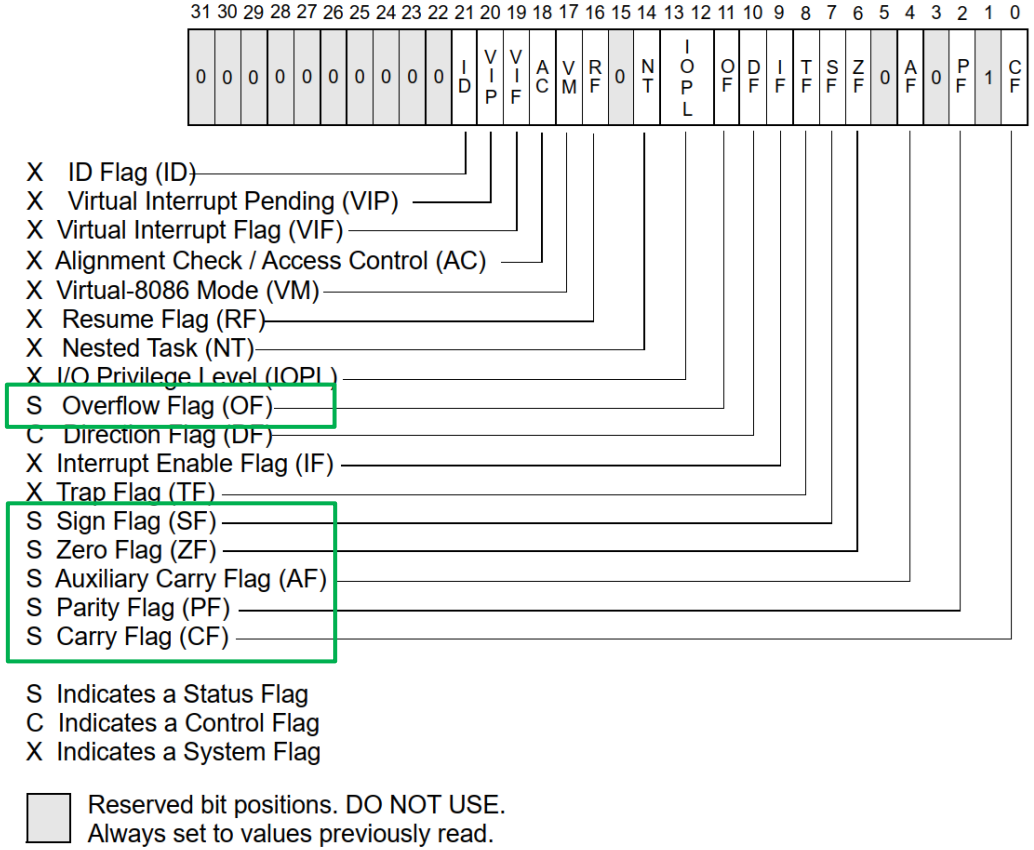
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>---</td></tr>
<tr><td>16-bit</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">IP</td></tr>
<tr><td>32-bit</td><td colspan="4"></td><td colspan="4">EIP</td></tr>
<tr><td>64-bit</td><td colspan="8">RIP</td></tr>
</table>

x86-64 Registers

RFLAGS is a register that contains the following:

- 6 status flags
- 1 control flag
- Various system flags
- The **upper 32 bits** of **RFLAGS** is reserved
- **Bits 1,3,5,15,** and **22-63** are reserved

x86-64 Registers



Note: The upper 32 bits of RFLAGS register is reserved

x86-64 Register: Status Flags

Status Flags: Status indicate the conditions that are produced as a result of executing an arithmetic or logic instruction

- **Carry flag (CF)**: bit 0
- **Parity flag (PF)**: bit 2
- **Auxiliary carry flag (AF)**: bit 4
- **Zero flag (ZF)**: bit 6
- **Sign flag (SF)**: bit 7
- **Overflow flag (OF)**: bit 11

x86-64 Register: Status Flags

- **Carry flag (CF)**: CF is set if there is a **carry-out from the MSb** of the result or a **borrow-in to the MSb** of the result during the execution of an arithmetic instruction.

- **Parity flag (PF)**: PF is set if the result produced has **even parity (i.e., even number of 1s)**. Only the **least significant byte** is tested.

x86-64 Register: Status Flags

- **Auxiliary carry flag (AF)**: Also known as "**half-carry**" flag. AF is set if there is a **carry-out** from the **low nibble (bit 3) to the high nibble (bit 4)** of the result, or a **borrow-in** from the **high-nibble (bit 4) to the low nibble (bit 3) of the result**. Only the **least significant byte** is tested. Note that counting of bits starts from 0.

- **Zero flag (ZF)**: ZF is set if the **result** of the operation is **zero**

x86-64 Register: Status Flags

- **Sign flag (SF)**: The **MSb** of the result is copied to the **SF**

- **Overflow flag (OF)**: OF is set if the **signed** result is **out-of-range** (i.e., too large a positive number or too small a negative number)

x86-64 Register: Status Flags

Example

```
MOV AL, 0x25
ADD AL, AL
```

```
25h     =     0010 0101
25h     =     0010 0101
-----------------------
4Ah     =     0100 1010
```

1. Is there a carry-out from /borrow-in to MSB?

CF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0x25
ADD AL, AL
```

```
25h     =     0010 0101
25h     =     0010 0101
-----------------------
4Ah     =     0100 1010
```

2. Are the resulting bits all 0s?

CF = 0

ZF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0x25
ADD AL, AL
```

```
25h     =     0010 0101
25h     =     0010 0101
-----------------------
4Ah     =     0100 1010
```

3. Is there an even number of 1s (in the least significant byte)?

CF = 0

ZF = 0

PF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0x25
ADD AL, AL
```

```
25h     =     0010 0101
25h     =     0010 0101
-----------------------
4Ah     =     0100 1010
```

4. What is the MSB?

CF = 0

ZF = 0

PF = 0

SF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0x25
ADD AL, AL
```

```
25h     =     0010 0101
25h     =     0010 0101
-----------------------
4Ah     =     0100 1010
```

4 3 2 1 0 – *Bit position*

5. Is there a carry-out from bit 3 or borrow-in to bit 4?

CF = 0

ZF = 0

PF = 0

SF = 0

AF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0x25
ADD AL, AL
```

```
25h     =     0010 0101
25h     =     0010 0101
-----------------------
4Ah     =     0100 1010
```

6. Is the signed result out of range?

CF = 0
ZF = 0
PF = 0
SF = 0
AF = 0
OF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0xFF
ADD AL, AL
```

```
FFh     =     1111 1111
FFh     =     1111 1111
-----------------------
FEh     =     1111 1110
```

1. Is there a carry-out from /borrow-in to MSB?

CF = 1

x86-64 Register: Status Flags

Example

```
MOV AL, 0xFF
ADD AL, AL
```

```
FFh     =     1111 1111
FFh     =     1111 1111
-----------------------
FEh     =     1111 1110
```

2. Are the resulting bits all 0s?

CF = 1

ZF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0xFF
ADD AL, AL
```

```
FFh     =     1111 1111
FFh     =     1111 1111
-----------------------
FEh     =     1111 1110
```

3. Is there an even number of 1s (in the least significant byte)?

CF = 1

ZF = 0

PF = 0

x86-64 Register: Status Flags

Example

```
MOV AL, 0xFF
ADD AL, AL
```

```
FFh     =     1111 1111
FFh     =     1111 1111
-----------------------
FEh     =     1111 1110
```

4. What is the MSB?

CF = 1

ZF = 0

PF = 0

SF = 1

x86-64 Register: Status Flags

Example

```
MOV AL, 0xFF
ADD AL, AL
```

```
FFh     =     1111 1111
FFh     =     1111 1111
-----------------------
FEh     =     1111 1110
```

5. Is there a carry-out from bit 3 or borrow-in to bit 4?

CF = 1

ZF = 0

PF = 0

SF = 1

AF = 1

x86-64 Register: Status Flags

Example

```
MOV AL, 0xFF
ADD AL, AL
```

```
FFh     =     1111 1111
FFh     =     1111 1111
-----------------------
FEh     =     1111 1110
```

6. Is the signed result out of range?

CF = 1
ZF = 0
PF = 0

SF = 1
AF = 1
OF = 0

x86-64 Register: Control Flag

- **Direction flag (DF, bit 10)**: The value of DF determines the direction in which the string operations will occur. **Setting the DF flag** causes the string instructions to **auto-decrement**. **Clearing the DF flag** causes the string instructions to **auto-increment**.