**Chapter 5: ADDITION TO x86-PROCESSORS**

**Topic – 12: 64-Bit x86-64 Processors**

**x86 Instruction Set Features**

* Addresses are **64-bit** long with **264 bytes** of ***virtual address space***.
* **General-purpose registers** in x86 are of **32-bit** as we saw.
* So, **instruction set** uses **32-bit extension** of those registers making them of **64-bit**.
* Thus, they can have **64-bit** **integer** **operands**.
* Also, **x86 instruction set** can use **48-bit** of ***physical address space***.
* We **can’t** run **16-bit real mode** or **virtual-8086 mode** on native **64-bit mode**.
* The **16-bit real mode** can be run through a ***legacy mode*** though but it is **not** available in **64-bit Windows**.

***\*x86-64 is an instruction set, but we will treat it as a processor type onwards\****

**Topic – 13: 64-Bit Operation Mode**

**Introduction**

* **Intel** **64** architecture has a new mode called ***IA-32e***.
* **IA-32e** again has two **sub-modes**:
  + ***Compatibility mode***
  + ***64-bit mode***
* We will refer these **sub-modes** as **modes** from now onwards.

**Compatibility Mode**

* Existing **16-bit** & **32-bit** applications can run **without** recompilation.
* However, **Win16** and **DOS** applications **won’t** run in **64-bit Windows**.
* **Win16:** 16-bit Windows applications.
* Moreover, **64-bit Windows** also **don’t** have ***virtual DOS machine subsystem***, which can run Windows **DOS** applications in **virtual-8086 mode**.

**64-Bit Mode**

* Runs applications using **64-bit linear address space**.
* Native mode for **64-bit Windows**.
* This mode even enables programmers to use **64-bit operands** in instructions.

**Topic – 14: Basic 64-bit Execution Environment**

**Introduction**

* In **64-bit mode**, processors **theoretically** support **64-bit long addresses**.
* But **practically**, they can use **48-bit long addresses** right now.

**64-Bit Registers v/s 32-Bit Registers**

|  |  |
| --- | --- |
| **64-bit Processor** | **32-bit Processor** |
| **Sixteen 64-bit general-purpose registers.** | **Eight 32-bit general-purpose registers.** |
| **Eight 80-bit floating point registers.** | *-SAME-* |
| **64-bit status flag is called RFLAGS.** | **32-bit status flag is called EFLAGS.** |
| **Only lower 32-bit of status flag are used.** | **Only lower 17-bit of status flag are used.** |
| **64-bit instruction pointer is called RIP.** | **32-bit instruction pointer is called EIP.** |
| **Eight 64-bit MMX registers.** | *-SAME-* |
| **Sixteen 128-bit XMM registers.** | **Eight 128-bit XMM registers.** |

**General-Purpose Registers**

* **General-purpose registers** can access **16-bit**, **32-bit** & **64-bit** operands with a special prefix.
* By default, the size of operand in **64-bit mode** is of **32-bit**.
* But when we add **REX** prefix, we can have **64-bit long** operands & total of **sixteen** **general-purpose registers** available to us.
* **Eight** of them are **same** as they were, while the **extra eight** are named **R8** to **R15**.

**Available Registers With Enabled REX**

* We are referring to only 64-bit modes here.

|  |  |
| --- | --- |
| **Operand Size** | **Available Registers** |
| **8-bit** | **AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L, R9L, R10L, R11L, R12L, R13L, R14L, R15L** |
| **16-bit** | **AX, BX, CX, DX, DI, SI, BP, SP, R8W, R9W, R10W, R11W, R12W, R13W, R14W, R15W** |
| **32-bit** | **EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D, R9D, R10D, R11D, R12D, R13D, R14D, R15D** |
| **64-bit** | **RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8, R9, R10, R11, R12, R13, R14, R15** |

* In **64-bit mode**, a single **can’t** access a **high-byte** register (like **AH**) & a **low-byte** from any extra registers (like **DIL**).
* And in same **64-bit mode**, **EFLAGS** is replaced by **RFLAGS** sharing **same** **lower 32-bit**.
* Rest of the **32-bits** **aren’t** used & thus **reserved**.