Intel IA-32 Architecture

80x86 Evolution

- 4004
  - 4-bit microprocessor.
  - 4KB main memory.
  - 45 instructions.
  - 50 KIPS
- 8008 (1972)
  - **8-bit** version of 4004.
  - **16KB** main memory.
  - 48 instructions.
- 8080 (1974)
  - 8-bit microprocessor.
  - **64KB** main memory.
  - 2 MHz clock rate; 500,000 instructions/sec.
  - 10X faster than 8008.

80x86 Evolution

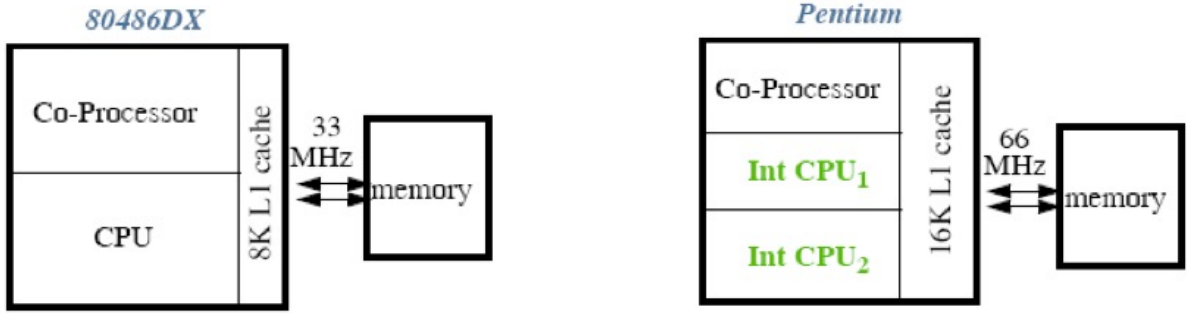
- 8085 (1976)
  - 8-bit microprocessor - upgraded version of the 8080.
  - 64KB main memory.
  - **1.3 microseconds** clock cycle time; 769,230 instructions/sec.
  - Intel sold 100 million copies of this 8-bit microprocessor.
- 8086 (1978) / 8088 (1979)
  - **16-bit** microprocessor.
  - **1MB** main memory.
  - 2.5 MIPS (400 ns).
  - 4- or 6-byte instruction cache.
  - Other improvements included more registers and additional instructions.
- 80286 (1982)
  - 16-bit microprocessor very similar in instruction set to the 8086.
  - **16MB** main memory.
  - 4.0 MIPS (250 ns/8MHz).

80x86 Evolution

- 80386 (1985)
  - **32-bit** microprocessor.
  - **4GB** main memory.
  - 12-33MHz.
  - Memory management unit added.
  - Variations: DX, EX, SL, SLC (cache) and SX.
    - 80386SX: 16MB through a 16-bit data bus and 24 bit address bus.
- 80486 (1989)
  - 32-bit microprocessor, 32-bit data bus and 32-bit address bus.
  - 4GB main memory.
  - 20-50MHz. Later at 66 and 100MHz
  - Incorporated an 80386-like microprocessor, 80387-like floating point coprocessor and an 8K byte cache on one package.
  - About half of the instructions executed in 1 clock instead of 2 on the 386.
  - Variations: SX, DX2, DX4.
    - DX2: Double clocked version:
    - 66MHz clock cycle time with memory transfers at 33MHz.

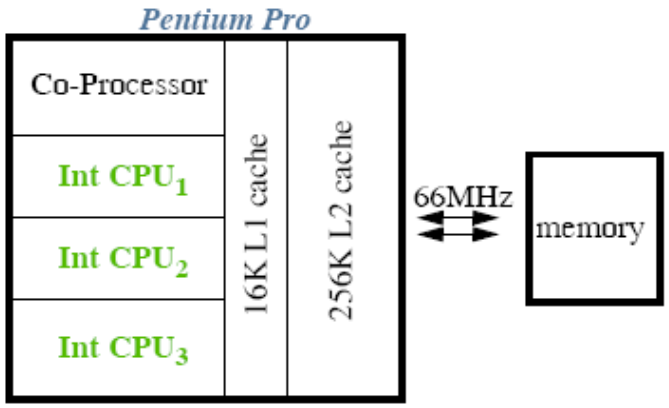
80x86 Evolution

- Pentium (1993)
  - 32-bit microprocessor, 64-bit data bus and 32-bit address bus.
  - 4GB main memory.
  - 60, 66, 90MHz.
    - 1-and-1/2 100MHz version.
    - Double clocked 120 and 133MHz versions.
    - Fastest version is the 233MHz (3-and-1/2 clocked version).
  - **16KB L1 cache** (split instruction/data: 8KB each).
  - Memory transfers at **66MHz** (instead of 33MHz).
  - Dual integer processors.



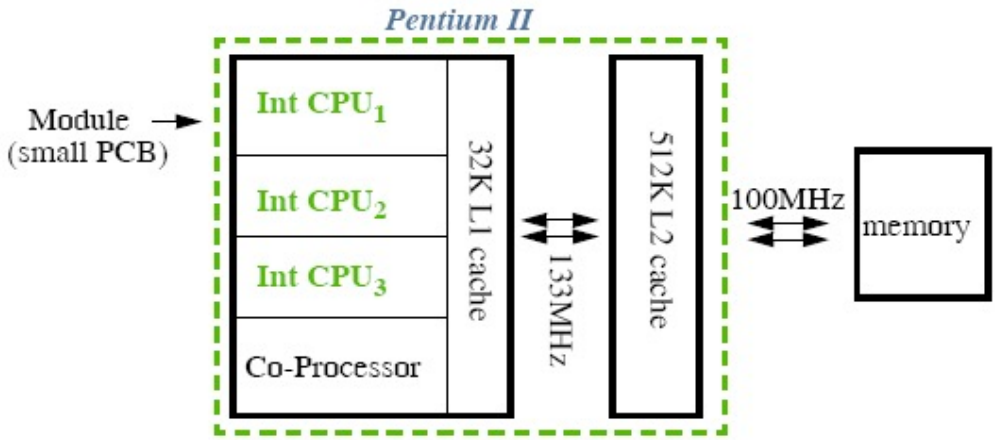
80x86 Evolution

- Pentium Pro (1995)
  - 32-bit microprocessor, 64-bit data bus and 36-bit address bus.
  - **64GB** main memory.
  - Starts at 150MHz.
  - 16KB L1 cache (split instruction/data: 8KB each).
  - **256KB L2 cache**
  - Memory transfers at 66MHz.
  - **3 integer processors**



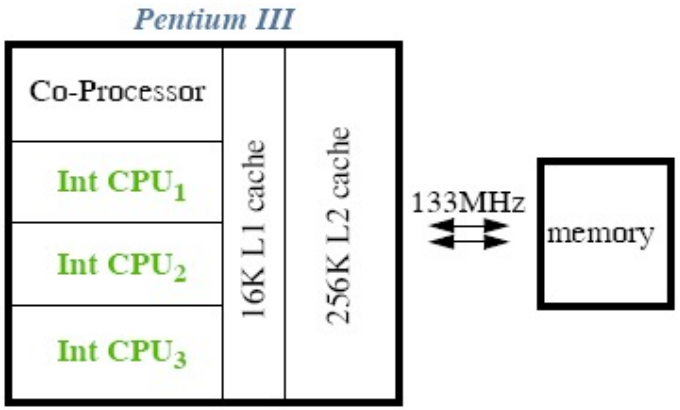
80x86 Evolution

- Pentium II (1997)
  - 32-bit microprocessor, 64-bit data bus and 36-bit address bus.
  - 64GB main memory.
  - Starts at 233MHz.
  - **32KB** split instruction/data L1 caches (16KB each).
  - **Module integrated 512KB L2 cache** (133MHz).
  - Memory transfers at 66MHz to 100MHz (1998).



80x86 Evolution

- Pentium III (1999)
  - 32-bit microprocessor, 64-bit data bus and 36-bit address bus.
  - 64GB main memory.
  - 800MHz and above.
  - 32KB split instruction/data L1 caches (16KB each).
  - **On-chip 256KB L2 cache**
  - Memory transfers 100MHz to 133MHz.



80x86 Evolution

- The Intel Pentium M Processor (2003-2006)
  - On-die, primary 32-KByte instruction cache and 32-KByte write-back data cache
  - On-die, second-level cache (up to 2 MByte)
  - Advanced Branch Prediction and Data Prefetch Logic
  - Support for MMX technology, Streaming SIMD instructions, and the SSE2 instruction set
  - A 400 or 533 MHz, Source-Synchronous Processor System Bus
  - Advanced power management using Enhanced Intel SpeedStep® technology

- The Intel Core Duo and Intel Core Solo Processors (2006-2007)
  - 2MB L2 cache: Intel® Smart Cache which allows for efficient data sharing between two processor cores
  - Improved decoding and SIMD execution
  - Intel® Dynamic Power Coordination and Enhanced Intel® Deeper Sleep to reduce power consumption
  - Intel® Advanced Thermal Manager which features digital thermal sensor interfaces
  - Support for power-optimized 667 MHz bus

80x86 Evolution

- Pentium 4 (2000~2006)
  - Netburst microarchitecture
  - 1.4 to 1.9GHz and the latest at 3.20 GHz and 3.46GHz (Hyper-Threading)!
  - 1MB/512KB/256KB L2 cache.
  - 800 MHz (about 6.4GB/s)/533 MHz (4.3 GB/s)/ 400MHz (3.2 GB/s) system bus.
  - Specialized for streaming video, game and DVD applications (144 new SIMD 128-bit instructions).
- Mobile Pentium 4-M / Pentium 4E / Pentium 4 EE

80x86 Evolution

- Itanium / Itanium2 (2001/2002~2010)
  - 733, 800MHz(Itanium), 900MHz – 1.6GHz (Itanium2)
  - IA-64 architecture
- Pentium 4F / Pentium D (2005)
  - NetBurst microarchitecture
  - Intel Extended Memory 64 Technology
  - Compatible with AMD's AMD64 architecture
  - PentiumD: desktop dual-core 64-bit x86-64 microarchitecture with Net Burst microarchitecture
- Intel Core 2 (2006)
  - core microarchitecture
  - Two cores on one die
  - 64KB L1 cache per core
  - Intel VT-x support
  - ~3.0GHz clock rate
- Xeon (2004~)

80x86 Evolution

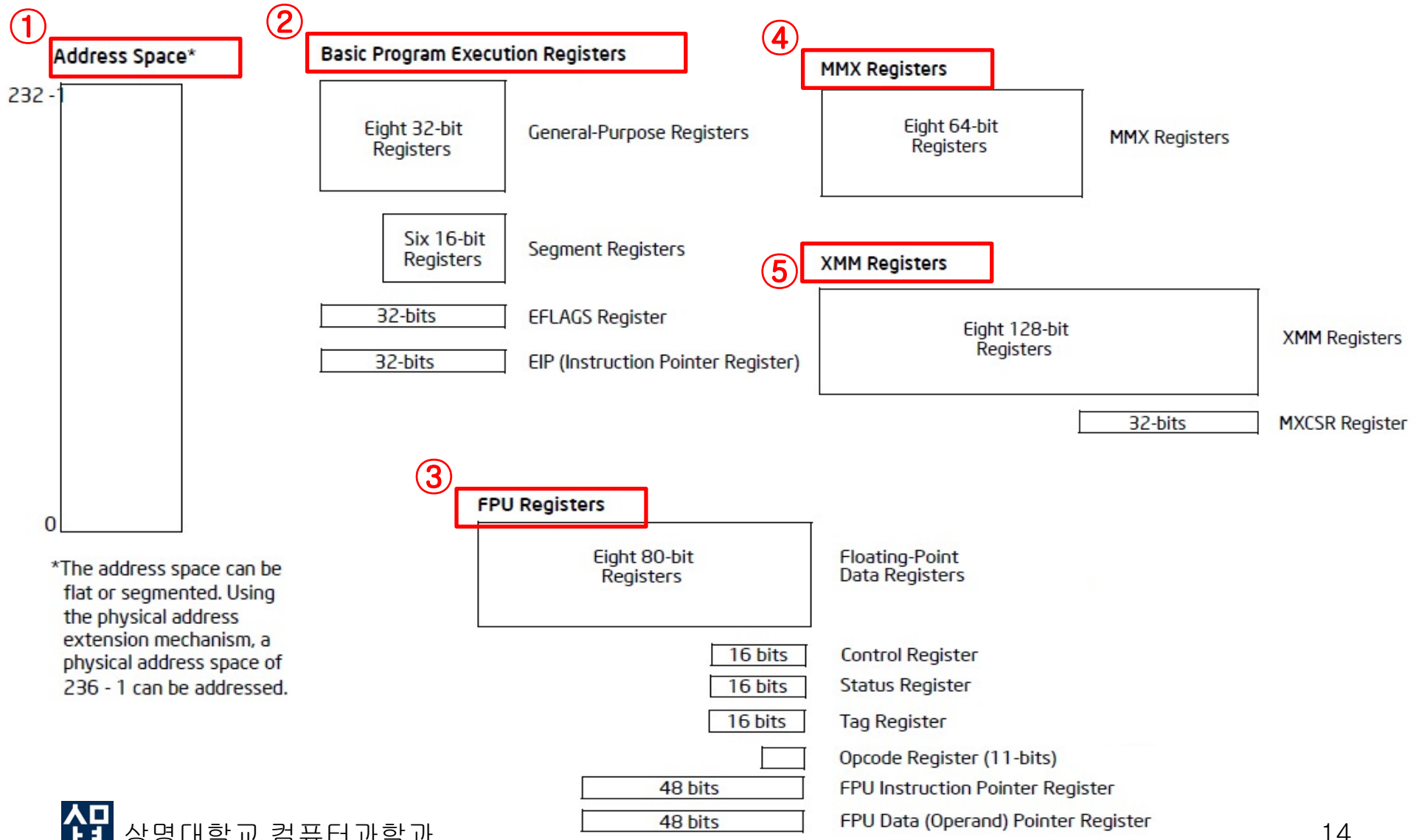
- Core i3/Core i5/ Core i7 (2010 -)
  - 1st generation: Nehalem microarchitecture(2010)
  - 2nd/3rd generation: Sandy Bridge/Ivy Bridge microarchitecture (2011-2012)
  - 4th generation: Haswell (2013)
  - 5th generation: Broadwell (2014)
  - 6th generation: Skylake (2015-2019)
  - Kaby lake, coffee lake, cannon lake, ice lake, comet lake, …

IA-32 Architecture

- **Computer architecture** is concerned with how the CPU acts and how it uses computer memory
- It includes *instruction set architecture, microarchitecture, system design* including all the other h/w components

- **IA-32 architecture** and **Intel 64 architecture** are the names of computer architecture starting from the Intel 8086 processor to the latest Intel Core 2 Duo and Intel Xeon processor 5100 series
  - Intel 64 / x86-64 / AMD64

IA-32 Basic Execution Environment



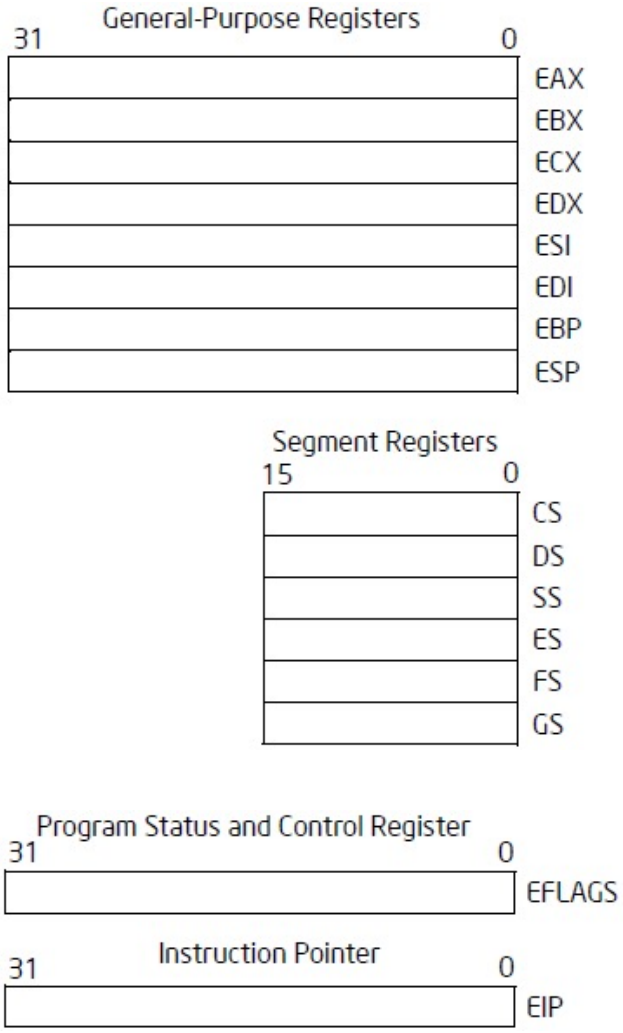
IA-32 Basic Execution Environment

- **Address space** — Any task or program running on an IA-32 processor can address a linear address space of up to 4 GB (2^{32} bytes) and a physical address space of up to 64 GB (2^{36} bytes).
- **Basic program execution registers** — The 8 general-purpose registers, the 6 segment registers, the *EFLAGS* register, and the *EIP* register comprise a basic execution environment to execute a set of general-purpose instructions. (integer arithmetic, program flow control, operate on bit and byte strings, and address memory)
- **Stack** — To support procedure or subroutine calls and the passing of parameters between procedures or subroutines, a stack and stack management resources are included in the execution environment

IA-32 Basic Execution Environment

- **x87 FPU registers** — The eight x87 FPU data registers, the x87 FPU control register, the status register, the x87 FPU instruction pointer register, the x87 FPU operand (data) pointer register, the x87 FPU tag register, and the x87 FPU opcode register for floating-point values
- **MMX registers** — The eight MMX registers support execution of single-instruction, multiple-data (SIMD) operations on 64-bit packed integers
- **XMM registers** — The eight XMM data registers and the MXCSR register support execution of SIMD operations on 128-bit packed integers.

Basic Program Execution Registers



- **General-purpose registers**. These eight registers are available for storing operands and pointers.
- **Segment registers**. These registers hold up to six segment selectors.
- ***EFLAGS* (program status and control) register**. The *EFLAGS* register report on the status of the program being executed and allows limited (application program level) control of the processor.
- ***EIP* (instruction pointer) register**. The *EIP* register contains a 32-bit pointer to the next instruction to be executed.

General Purpose Registers

- The 32-bit general-purpose registers *EAX*, *EBX*, *ECX*, *EDX*, *ESI*, *EDI*, *EBP*, and *ESP* are provided for holding the following items:
  - Operands for logical and arithmetic operations
  - Memory pointers
  - Operands for address calculations

- The *ESP* register holds the stack pointer and as a general rule should not be used for another purpose.

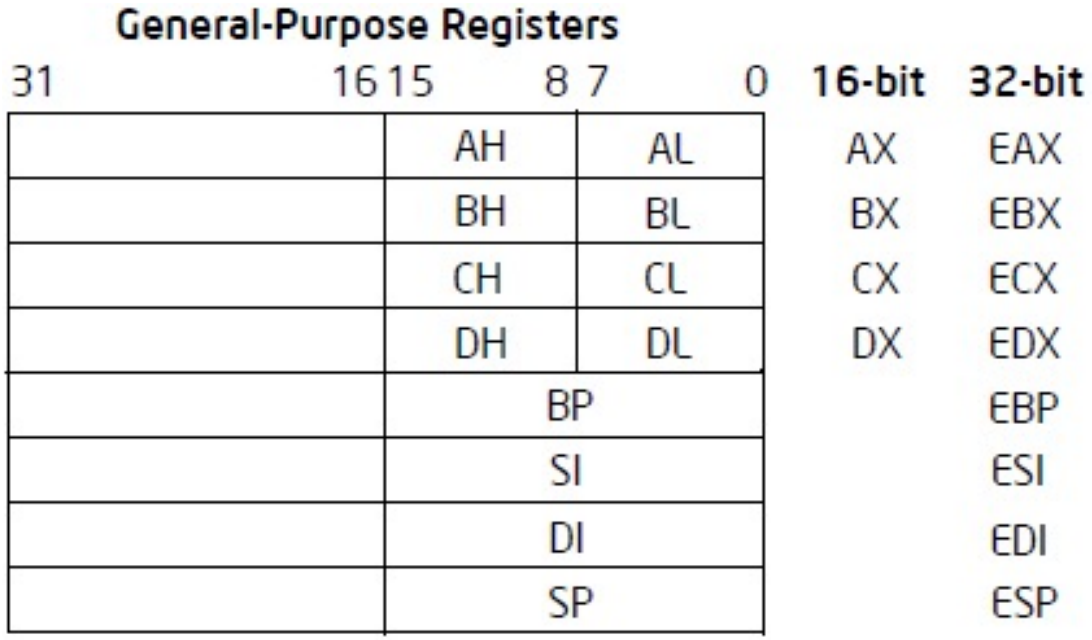
- Many instructions assign specific registers to hold operands (eg. string instructions - *ECX*, *ESI*, and *EDI*)

- *EAX*: Accumulator: Referenced as *EAX*, *AX*, *AL* or *AH*
  - Used for *mult*, *div*, etc.
  - Used to hold an offset.

General Purpose Registers

- *EBX*: Base Index
  - Used to hold the offset of a data pointer.
  - Pointer to *DS* segment
- *ECX*: Count
  - Used to hold the count for some instructions, *REP* and *LOOP*.
  - Used to hold the offset of a data pointer.
- *EDX*: Data
  - Used to hold a portion of the result for *mult*, of the operand for *div*.
  - I/O pointer
- *ESI*: Source Index
- *EDI*: Destination Index
  - Holds the base source/destination pointer for string instructions.
- *ESP*: Stack pointer
- *EBP*: Base Pointer
  - Holds the base pointer for memory data transfers.
  - Pointer to data on the stack

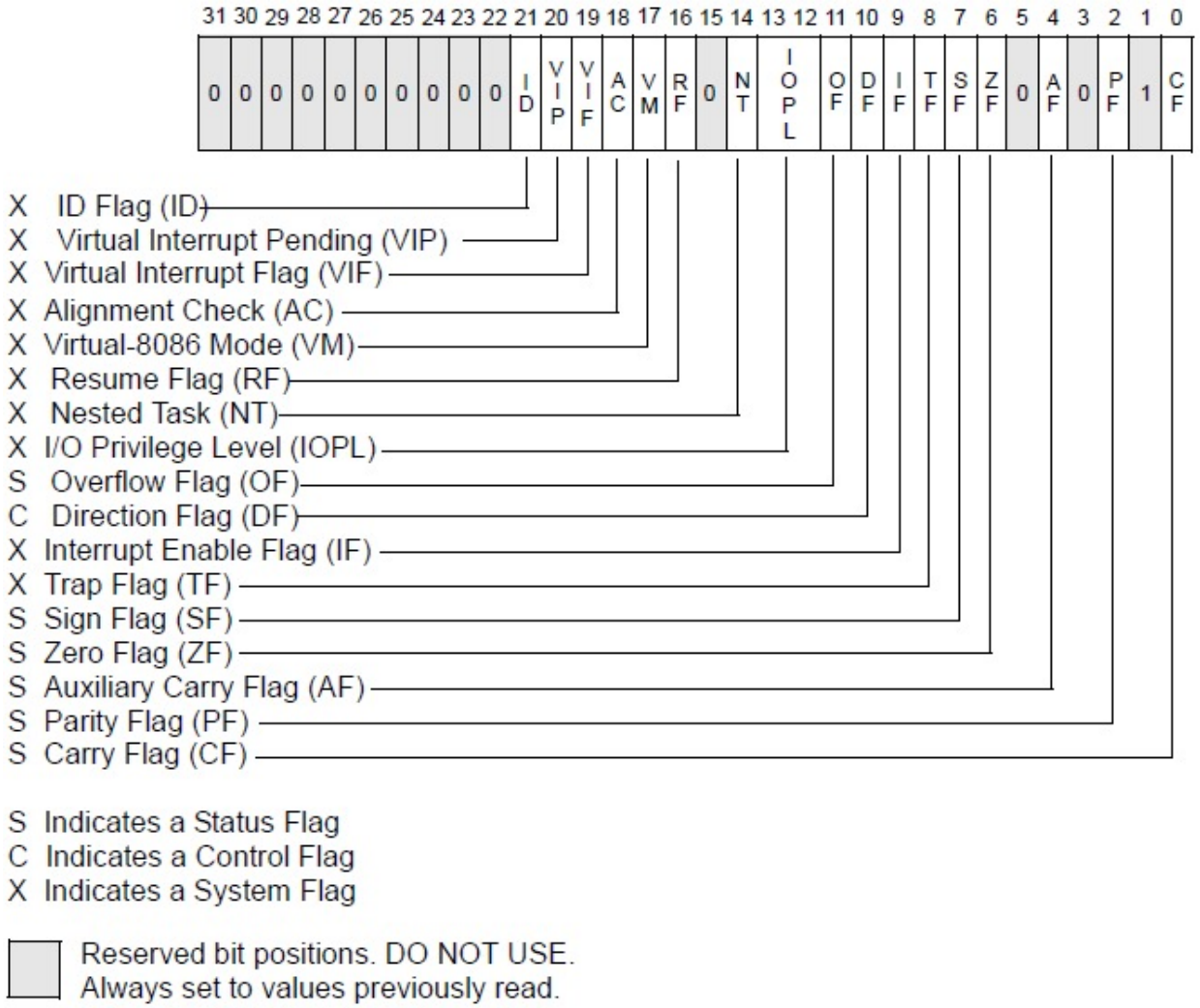
Alternate General-Purpose Register Names



EFLAGS Register

- The 32-bit *EFLAGS* register contains a group of status flags, a control flag, and a group of system flags.
- The initial state of the *EFLAGS* register is 00000002H. Bits 1, 3, 5, 15, and 22 through 31 of this register are reserved.
- The following instructions can be used to move groups of flags to/from the procedure stack or the EAX register: *LAHF*, *SAHF*, *PUSHF*, *PUSHFD*, *POPF*, and *POPFD*
- When a call is made to an interrupt / exception handler procedure, the processor automatically saves the state of the *EFLAGS* registers on the procedure stack

EFLAGS Register



Status flags

- The status flags (bits 0, 2, 4, 6, 7, and 11) of the *EFLAGS* register indicate the results of arithmetic instructions, such as the *ADD*, *SUB*, *MUL*, and *DIV* instructions.

- ***CF* (bit 0) Carry flag —** Set if an arithmetic operation generates a carry or a borrow out of the most-significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic.
- ***PF* (bit 2) Parity flag —** Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise.
- ***AF* (bit 4) Adjust flag —** Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; cleared otherwise. This flag is used in binary-coded decimal (BCD) arithmetic.
- ***ZF* (bit 6) Zero flag —** Set if the result is zero; cleared otherwise.
- ***SF* (bit 7) Sign flag —** Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)
- ***OF* (bit 11) Overflow flag —** Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.

Status flags

- Of these status flags, only the *CF* flag can be modified directly, using the *STC*, *CLC*, and *CMC* instructions. Also the bit instructions (*BT*, *BTS*, *BTR*, and *BTC*) copy a specified bit into the *CF* flag.
- The status flags allow a single arithmetic operation to produce results for three different data types: unsigned integers, signed integers, and BCD integers.
- The condition instructions *Jcc* (jump on condition code *cc*), *SETcc* (byte set on condition code *cc*), *LOOPcc*, and *CMOVcc* (conditional move) use one or more of the status flags as condition codes and test them for branch, set-byte, or end-loop conditions.

Direction flag

- The direction flag (*DF*, located in bit 10 of the *EFLAGS*) controls string instructions (*MOVS*, *CMPS*, *SCAS*, *LODS*, and *STOS*).
- Setting the **DF** flag causes the string instructions to auto-decrement (to process strings from high addresses to low addresses).
- Clearing the *DF* flag causes the string instructions to auto-increment (process strings from low addresses to high addresses).
- The *STD* and *CLD* instructions set and clear the *DF* flag, respectively.

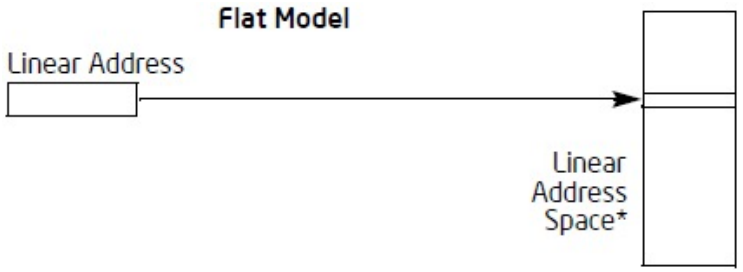
Instruction Pointer Register

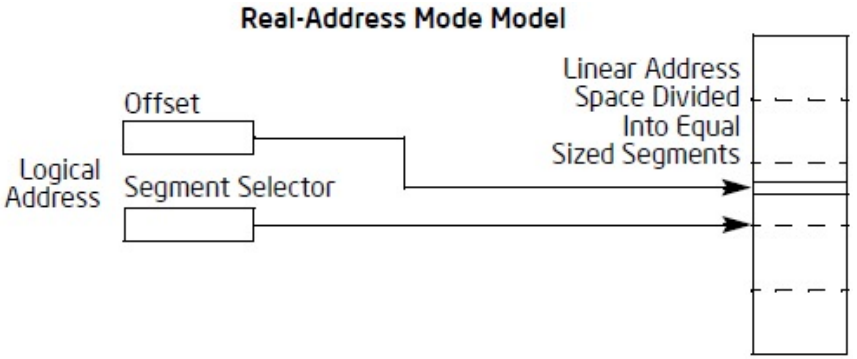
- The instruction pointer (*EIP*) register contains the offset in the current code segment for the next instruction to be executed.
- It is advanced from one instruction boundary to the next in straight-line code or it is moved ahead or backwards by a number of instructions when executing *JMP*, J*cc*, *CALL*, *RET*, and *IRET* instructions.
- The *EIP* register cannot be accessed directly by software; it is controlled implicitly by control-transfer instructions (such as *JMP*, J*cc*, *CALL*, and *RET*), interrupts, and exceptions. The only way to read the *EIP* register is to execute a *CALL* instruction and then read the value of the return instruction pointer from the procedure stack.
- The *EIP* register can be loaded indirectly by modifying the value of a return instruction pointer on the procedure stack and executing a return instruction

Modes of Operation

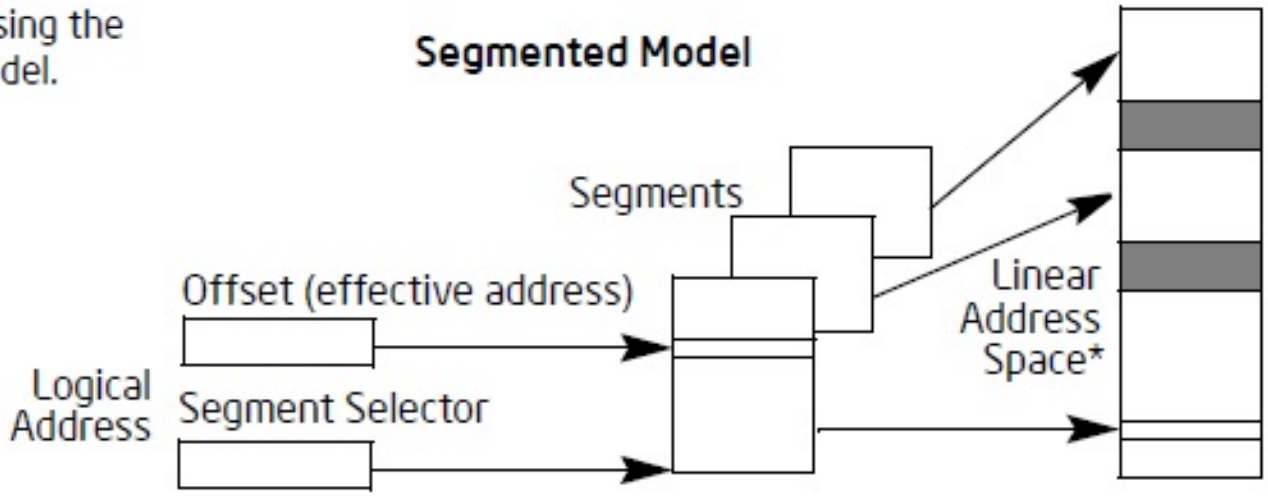
- The IA-32 architecture supports three basic operating modes. The operating mode determines which instructions and architectural features are accessible
  - Protected mode
  - Real-address mode
  - System management mode
- IA-32 memory models
  - Flat memory model
  - Segmented memory model
  - Real-address memory model

IA-32 Memory Models

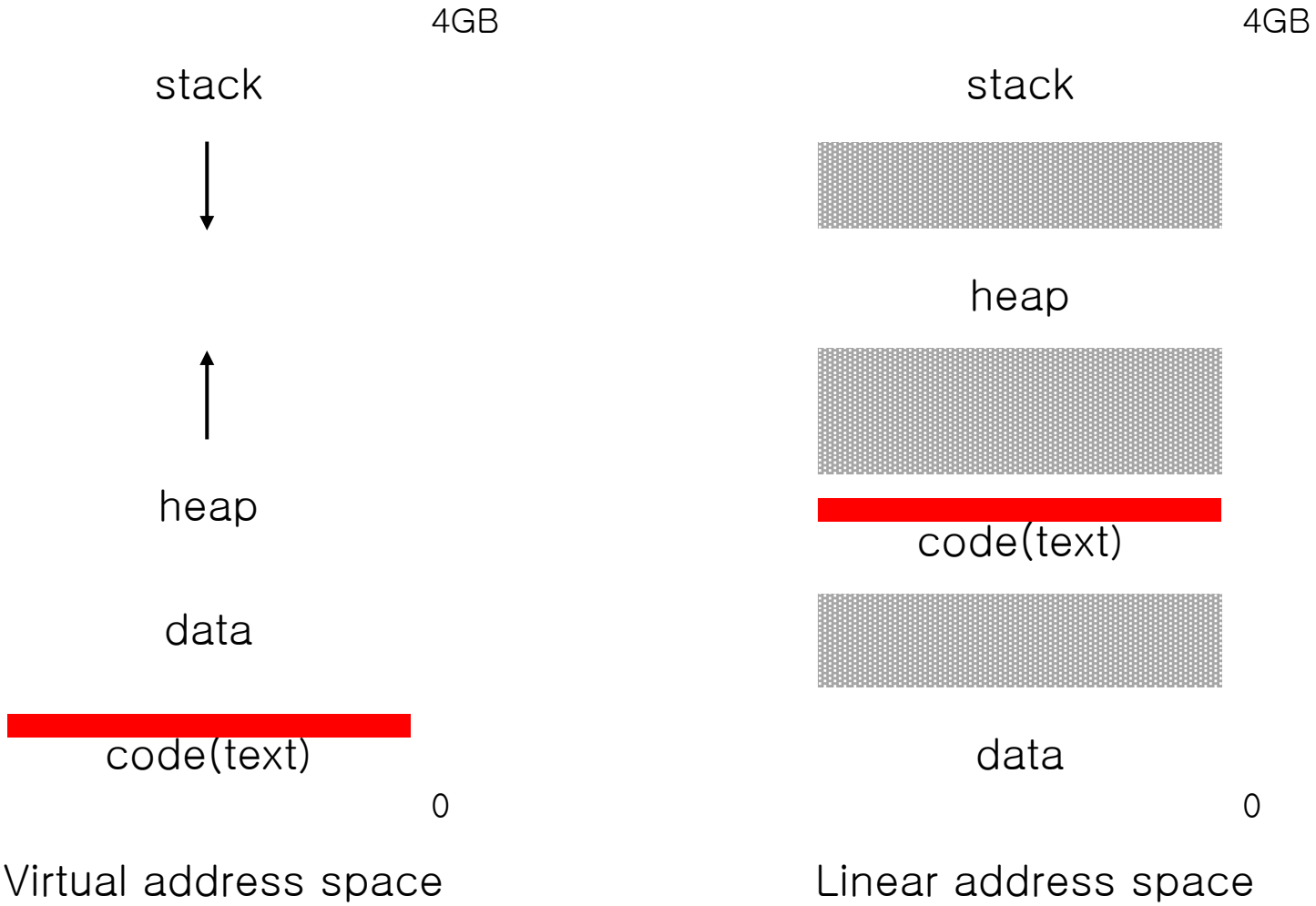




* The linear address space can be paged when using the flat or segmented model.



Program in Memory



Segment Registers

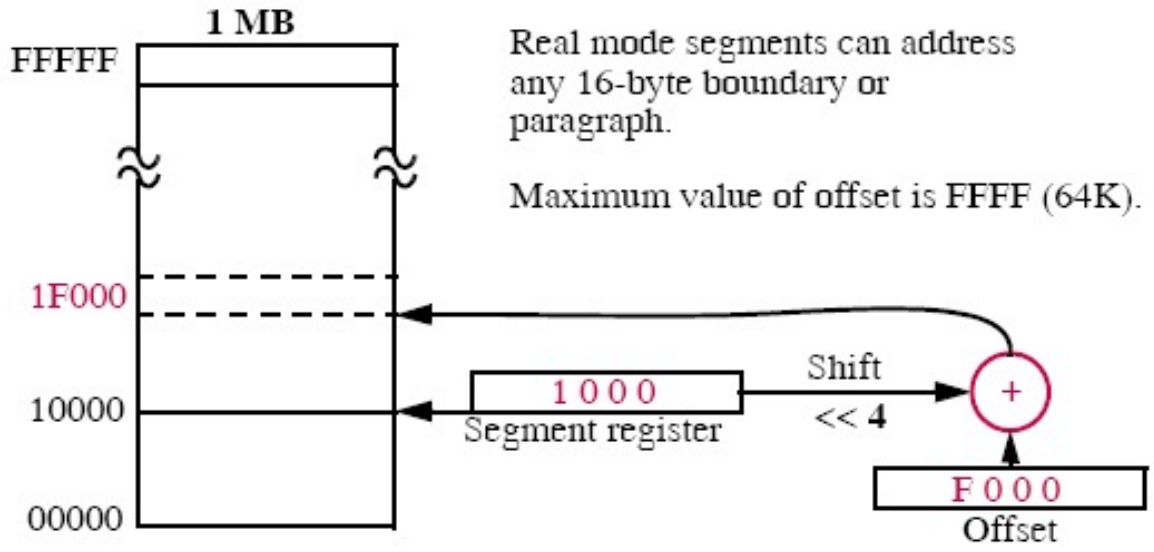
- The segment registers, *CS*, *DS*, *SS*, *ES*, *FS*, and *GS* hold 16-bit segment selectors (A segment selector is a special pointer that identifies a segment in memory).

- *CS* (Code Segment):
  - In real mode, this specifies the start of a 64KB memory segment.
  - In protected mode, it selects a descriptor.
- *DS* (Data Segment):
  - Similar to the *CS* except this segment holds data.
- *ES* (Extra Segment):
  - Data segment used by some string instructions to hold destination data.
- *SS* (Stack Segment):
  - Similar to the *CS* except this segment holds the stack.
  - *ESP* and *EBP* hold offsets into this segment.
- *FS* and *GS*
  - Allows two additional memory segments to be defined.

Real- Address Memory Model in Real-Address Mode

- Only mode available to the 8086 / 8088 and real-address mode for IA-32 architecture.
  - Allow the processor to address only the first 1MB of memory.
- ***Segments and Offsets***
  - *segment:offset*
  - Effective address = Segment address + an offset.

eg)1000:F000



Real Mode Memory Addressing

- ***Segments and Offsets***:
  - Syntax is usually given as *seg_addr:offset*, e.g. *1000:F000* in the previous example to specify *1F000H*.
  - Implicit combinations of segment registers and offsets are defined for memory references.
  - For example, the code segment (*CS*) is always used with the instruction pointer (IP for real mode or EIP for protected mode).
- Examples
  - ***CS:IP***
  - ***SS:SP, SS:BP***
  - ***DS:AX, DS:BX, DS:CX, DS:DX, DS:DI, DS:SI, DS:8-bit_literal, DS:16-bit_literal***
  - ***ES:DI***
  - ***FS*** and ***GS*** have no default.

Memory Addressing in Real-Address Mode